Contention-aware task mapping and scheduling with Hybrid Search for heterogeneous NoC-based MPSoCs

No Author Given

No Institute Given

Abstract.

1 Introduction

The scalable communication architecture makes network-on-chip (NoC) a prevailing solution for the next generation multiprocessor system-on-chips (MPSoCs). In the NoC paradigm, an important issue is how to map and schedule tasks of an application to processing elements (PEs), such that the system performance is maximized, while minimizing total energy consumption [**?**,**?**]. Within this architecture, communication is the main concern in the optimization process. To reduce communication consumption, 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. Reducing contention requires detailed scheduling and long latency in communication. Therefore, it is a great challenge for NoC designers to tackle the issues on performance and energy consumption.

With contention awareness in the design of NoCs, performance and energy consumption optimization need to consider how to map tasks to available PEs and how to schedule tasks on same PEs to avoid contention. In this paper, system performance is evaluated by *makespan*, i.e., the scheduling length of the system. And energy consumption encompasses both computation and communication cost. Traditional works mainly focus on communication consumption with respect to the distance and the amount of data between various PEs, and the cost of computation is simplified as the sum of task execution cost on the allocated PEs. Therefore, the design optimization can be reduced to mapping optimization. However, various data transmission activities may require same resources and cause contention, which can lead to the big gap between ideal predication and real performance.

To optimize performance and energy consumption with the consideration of potential contentions in communication, we construct formulations for mapping and scheduling constraints and objectives to be optimized. Meanwhile, we integrate a problem-specific local search into a multi-objective evolutionary-based algorithm. The main contributions of the paper are as follows. First, to reduce

spacial contention in mapping, we encode the probability of contention in terms of overlapped paths, and formulate the distribution of contention probability as an objective that can be minimized. Second, to avoid temporal contention in data transmission, we construct the corresponding constraints in scheduling behavior. Third, the energy model encodes the power consumption on links and routers, as well as different modes of PEs. Finally, we construct a hybrid search algorithm, by integrating a local search into a generic algorithm. The algorithm is capable of dealing with large scale applications in approximating better Parent fronts. While various design choices exist for NoCs, in this paper, we limit our consideration to 2D mesh networks. However, our idea is applicable to other network topologies such as torus and 3D networks.

The rest of the paper is as follows. Section **??** discusses related work in the domain static mapping and scheduling. Section **??** introduces the task model and architecture model. Section **??** provides the formulation of the problem. Section **??** presents the details of the algorithm. Experimental results are presented in Section **??**. Section **??** concludes the paper.

2 Related work

For embedded systems, energy efficiency is one of critical design issues. Consequently, communication energy minimization is the main concern for the mapping problem of NoCs [**?**]. The corresponding cost uses directly or indirectly the average number of packet hops, with bandwidth or latency constraints [**?**,**?**,**?**]. To have a global view of the entire NoC, task mapping and scheduling with the consideration of communication latency is another important problem for the design of NoCs [**?**,**?**]. For contention-aware works, considering it in mapping without time information is an over-approximation of contentions [**?**]. Though the work in [**?**] considers contention in scheduling, it introduces latency to avoid contention and mainly focuses on makespan minimization. He et al. estimate traffic congestions by using heuristics [**?**]. Han et al. consider contention and energy-aware in mapping and scheduling, with pre-defined priorities for tasks and data transmissions [**?**]. To evaluate performance globally, we introduce contention reduction in both mapping and scheduling, and both performance and energy consumption can be optimized at the same time.

Various algorithms exist for mapping optimization, such as tabu search [**?**], branch-and-bound [**?**], binary particle swarm optimization [**?**], and diagonal map [**?**]. Similar to the solutions for MPSoCs, genetic algorithms [**?**], simulated annealing algorithm [**?**], mixed integer linear programming (MILP) [**?**] have also used to solve the mapping and scheduling problems. An ILP formulation has been proposed for contention aware application mapping in tile based NoC to reduce inter-tile network contention[**?**]. Huang et al. propose a simulated annealing algorithm with timing adjustment heuristic is proposed to minimize energy consumption [**?**]. These works only consider mapping issues for NoCs. The work in [**?**] employs population based incremental learning algorithm for mapping and scheduling optimiztation. He et al. introduce a unified model in mixed integer

linear programming, for task mapping and scheduling [**?**]. An ILP formulation encoding fine-grain communication optimization is proposed in [**?**]. And a heuristic algorithm is designed for application mapping and scheduling. To optimize the objectives in our constraint formulation, we put forward a hybrid search algorithm for both performance and energy optimization. Therefore, from the aspect of our formulation and the algorithm, the techniques discussed in the paper are different from existing works.

3 Preliminaries

We consider applications represented using task graphs (TGs). A task graph is a directed acyclic graph (DAG).

Definition 1 (Task Graph) *A task graph is a directed acyclic graph (DAG) with* $G = \langle T, E \rangle$*, where* T *is a finite set of tasks, with* $\delta : T \to \mathbb{N}$ *to indicate the amount of work to be dealt with on each task, and* $E \subseteq T \times T$ *is a set of precedence relations with* $c : E \to \mathbb{N}$ *to indicate the amount of data transferred between each pair of tasks.*

Each vertex may have a relative deadline, specifying the latest tolerable finishing time w.r.t. the start time of the first task in a period. The model shown in Figure **??**(a) is a task graph with 4 tasks and 5 dependency relations.

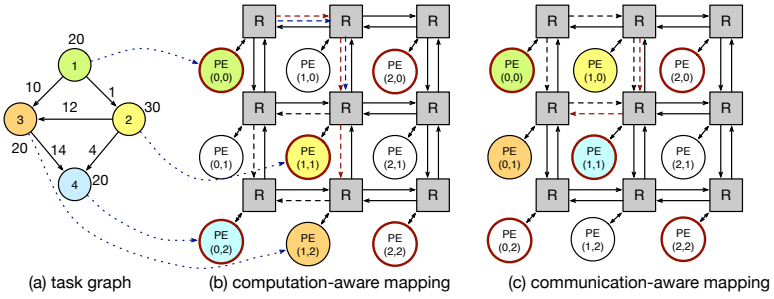

Fig. 1. A running example.

The target hardware is a 2D mesh NoC-based heterogeneous MPSoCs, as illustrated in Figure **??**(b). Each PE in the model is connected to a *router*. And routers are connected with each other through bidirectional *links*. Data transmission through the NoC is in the form 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 a pair of coordinates (x_i, y_i). And the distance between any two PEs p_i and p_j is the Manhattan distance:

$$D_{ij} = abs(x_i - x_j) + abs(y_i - y_j) \tag{1}$$

Table 1. Variables

Item	explanation
m_{ik}	task t_i is mapped to p_k
n_{ij}	task t_j is executed next to t_i on the same processor
s_i	the start time of executing task t_i
f_i	the end time of executing task t_i
se_{ij}	the start time of data transfer from t_i to t_j
fe_{ij}	the finish time of data transfer from t_i to t_j
d_{ij}	task t_j depends on the output of t_i
o_{ij}	task t_i needs to send data to its successor
D_{ij}	distance between p_i and p_j
P_{ij}	the contention probability of data transfer from two tasks

Figure **??**(b) shows an example with 3×3 nodes of two different types of processors, where the speed of processors with red cycles is two times faster than others. The communication is assumed to have cost 1 per unit payload per link.

We present two kinds of mappings for the model in Figure **??**(a), where one utilizes faster processors and the other minimizes communication paths. With the mapping in Figure **??**(b), the cost of computation is 55, and the cost of communication is 66, if we only consider the cost in links, and ignore the latency caused by contention and communication. For the case with mapping in Figure **??**(c), the cost of computation is 70, and the cost of communication is 53. The first mapping have a contention between communications from p_{00} to p_{11} and from p_{00} to p_{12}. However, the caused delay is only 1. Therefore, it is hard to choose without performance information.

4 Constraint formulation

Compared with other formulations, the features of our model are as follows. First, we evaluate the allocation and scheduling strategy for a certain number of iterations, instead of once execution of every task. This allows a more accurate system evaluation. Second, additional to the deadline for every task, we put forward the concept of latest response time between two tasks, to meet the requirements of time-critical systems. Third, we consider fine-grained communication in the case of contentions. Finally, additional to energy consumption and performance optimization, we also consider to minimize the probability of contention caused by communication from different resources in a communication network, which is evaluated by the trigger time and the overlapped area between two communication paths.

First, we introduce the variables and constants used in our formulation in Table **??** and Table **??**, respectively, to present the constraints in allocating and scheduling issues.

Table 2. Constants

Item	explanation
bw	the bandwidth of a link
τ	the time of transferring a unit with unit distance
τ'	the time of transferring a unit through a node
ϵ	the energy of transferring a unit with unit distance
ϵ'	the energy of transferring a unit through a node
a_i	the amount of computation for task t_i
c_{ij}	the amount of data to be transferred from task t_i to t_j
dl_{ij}	maximal time allowed between executing task t_i and t_j
ρ_k	processing speed of p_k
ω_k	the memory limit for p_k
$\gamma_{i_1 j_1 i_2 j_2}$	the number of shared transmission path exists from p_{i_1} to p_{j_1} and from p_{i_2} to p_{j_2}
ξ	the time bound for simultaneous communication
N	the number of iterations for all the tasks

In Table **??** can be computed according to the positions of processors. Given processors p_{i_1}, p_{j_1}, p_{i_2} and p_{j_2}, let (x_i, y_i) be the location of p_i. We have

$$oX' = (x_{j1} - x_{i_1}) + (x_{j2} - x_{i_2}) - (max\{x_{j_1}, x_{j_2}\} - min\{x_{i_1}, x_{i_2}\})$$

$$oY' = (y_{j1} - y_{i_1}) + (y_{j2} - y_{i_2}) - (max\{y_{j_1}, y_{j_2}\} - min\{y_{i_1}, y_{i_2}\}).$$

Let $oX = max\{oX', 0\}$, and $oY = max\{oY', 0\}$. We have $\gamma_{i_1 j_1 i_2 j_2} = oX + oY$.

4.1 Constraints

The constraints can be divided into two types: static mapping and dynamic behaviors. For behavior constraints, they consist of computation and communication behaviors.

Static constraints

$$\sum_{k=1}^{|P|} m_{ik} = 1 \tag{2}$$

$$\sum_{i=1}^{|T|} m_{ik} \cdot q_i \leq \omega_k \tag{3}$$

$$n_{ij} + n_{ji} \leq 1 \wedge (n_{ij} \wedge n_{jl} \rightarrow \neg n_{il}), \quad n_{ij} \leq m_{ik} \cdot m_{jk'} \tag{4}$$

$$d_{ij} \wedge m_{ik} \wedge m_{jk'} \wedge (k \neq k') \rightarrow o_{ij}, \quad o_{ij} \rightarrow d_{ij} \tag{5}$$

Constraint **??** requires that one task can only be mapped to one processor. Constraint **??** specifies that the capacity of a processor cannot be violated. Constraint **??** explains that the next-door execution relation is not transitive and reflexive. Constraint **??** reasons the case with communication cost.

4.2 Dynamic behaviors

First, we present precedence relation for computation and communication.

$$s_j^v - s_i^u \leq dl_{ij} \tag{6}$$

$$\sum_{j=1}^{|T|} d_{ji} > 0 \rightarrow fe_{ji}^u \leq s_i^u \tag{7}$$

$$f_i^u \leq se_{ij}^u \tag{8}$$

$$n_{ij} \rightarrow f_i^u \leq s_j^v, \text{ for } u \leq v \tag{9}$$

$$m_{ik} \wedge m_{jk} \rightarrow f_j^u \leq s_i^v, \text{ for } u < v \tag{10}$$

Constraint **??** specifies that the response time between certain tasks should less than the defined limitation. Constraint **??** requires that the start of executing t_i should wait for data being transferred, except for the one without any predecessor. Constraint **??** specifies that the start of data transfer has to wait for the finish of task execution. Constraint **??** requires that for two tasks scheduled next to each other in the same processor, the start of the latter should wait for the finish of the former, where $u \leq v$. Constraint **??** explains that the execution of task in later iteration v should not be earlier than any other tasks executed on the same processor in an earlier iteration u, where $u < v$.

Next, we present constraints to avoid overlapped computation and communication.

$$m_{ik} \wedge m_{jk} \rightarrow s_j^u \geq f_i^u \vee s_i^u \geq f_j^u \tag{11}$$

$$\gamma_{ijir} \rightarrow se_{ij}^u - se_{ir}^u \geq \xi \tag{12}$$

$$\gamma_{ijlr} \wedge abs(f_i - f_l) \leq \xi \rightarrow se_{ij}^u \geq fe_{lr}^u \vee se_{lr}^u \geq fe_{ij}^u \tag{13}$$

Constraint **??** specifies that two tasks executed on the same processor cannot be overlapped. Constraint **??** says that data transfer sharing same links from the same source cannot be triggered simultaneously. Constraint **??** says that two communication with shared resource cannot overlap, if their tasks finish execution almost at the same time. This constraint aims to allow the following case shown in Figure **??**. A short communication path from p_2 to p_3 is overlapped by a long communication path from p_0 to p_3. But the communication of the short path can be finished before the long transmission arrives the shared path.

Next we present quantitative relation for the variables on computation and communication.

$$f_i^u = s_i^u + a_i / (\sum_{k=1}^{|P|} m_{ik} \cdot \rho_k) \tag{14}$$

$$fe_{ij}^u \geq se_{ij}^u + o_{ij} \cdot (c_{ij} \cdot \tau \cdot D_{ij}/bw + \tau' \cdot (D_{ij} + 1)) \tag{15}$$

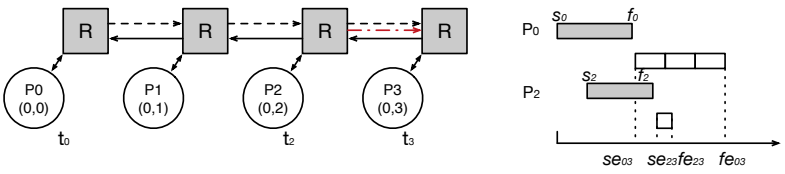

Fig. 2. An example with communication overlap.

Constraint **??** presents task execution relation. Constraint **??** shows that the finish of a data transfer considers the distance of the destinationwhere D_{ij} is the distance from the located processor of t_i to the target processor.

4.3 Optimization objectives

With the defined variables, the makespan of executing an application within N iterations is

$$\mathcal{M} = max\{f_i^N\} \tag{16}$$

However, energy consumption includes cost for computation E_p and communication E_m. For computation, we distinguish different status in a processor and accumulate the total cost. For communication, we consider the cost for routing and data transfer. In a NoC network, the number of available processors may be larger than that of tasks. We use $P' = \{p_k \mid \sum_{i=1}^{|T|} m_{ik} > 0\}$ to denote the set of occupied processors.

A processor may switch between dynamic, static and sleep modes, to save energy consumption. If the idle time is less than t_o, the processor will not switch into sleep mode. Otherwise, a time penalty t_Δ for switching into and out of sleep is considered, together with an energy penalty ϵ_Δ. The execution duration of every iteration in a processor may be different, thus the cost should differ. Let t_h be head of scheduled task in a processor p_k in every iteration, where the constraint $\sum_{j=1}^{|T|} m_{jk} - 1 = \sum_{j=1}^{|T|} q_{k_hj}$ holds. The duration $\mathcal{D}_k^u$ for processor p_k in iteration u is

$$\mathcal{D}_k^u = \begin{cases} \tau_h^{u+1} - \tau_h^u & u < N \\ \mathcal{M} - \tau_h^N & u = N \end{cases} \tag{17}$$

Let t_l be the tail of scheduled task in p_k, with $\sum_{i=1}^{|T|} q_{k_li} + 1 = m_{lk}$. Then the spare time $\mathcal{S}_{k_{ij}}^u$ for a processor p_k executing two successive tasks t_i and t_j in iteration u can be computed as follows:

$$\mathcal{S}_{k_{ij}}^u = \begin{cases} 0 & \gamma_{k_{ij}} = 0 \wedge i \neq l \\ \tau_j^u - e_i^u & \gamma_{k_{ij}} = 1 \wedge i \neq l \\ \mathcal{D}_k^u - e_i^u & i = l \end{cases} \tag{18}$$

Let $\mathcal{P}_{d_k}$, $\mathcal{P}_{i_k}$ and $\mathcal{P}_{s_k}$ be the rate of dynamic, static (idle) and sleep power consumption of processor p_k, respectively. The costs for dynamic (E_d), static

and sleep (E_{is}) are calculated as follows:

$$E_d = N \cdot \sum_{k=1}^{|P'|}(\mathcal{P}_{d_k} \cdot \sum_{i=1}^{|T|} \theta_i \cdot m_{ik}) \tag{19}$$

$$E_{is} = \sum_{u=1}^{N} \sum_{k=1}^{|P'|} \sum_{i,j=1}^{|T|} \begin{cases} \mathcal{P}_{s_k} \cdot (\mathcal{S}^u_{k_{ij}} - t_\Delta) + \epsilon_\Delta & \mathcal{S}^u_{k_{ij}} \geq t_o \\ \mathcal{P}_{i_k} \cdot \mathcal{S}^u_{k_{ij}} & 0 \leq \mathcal{S}^u_{k_{ij}} < t_o \end{cases} \tag{20}$$

The energy cost for communication is evaluated according to the amount of data and the distance of the transmission.

$$E_m = N \cdot \sum_{k,k'=1}^{|P'|} \sum_{i,j=1}^{|T|} c_{ij} \cdot o_{ij} \cdot m_{ik} \cdot m_{jk'} \cdot (\epsilon \cdot D_{kk'} + \epsilon' \cdot (D_{kk'} + 1)) \tag{21}$$

Addition to the energy and makespan optimization, we also expect that the probability of communication contention is reduced. As data transfer occurs in the end of task execution, we consider the potential contention when two tasks at various processors finish execution almost simultaneously (their difference is less than a limited time bound. The probability is evaluated according to the overlapped path between two communication paths in a communication network. Suppose two paths are from processor i_1 to j_1, and i_2 to j_2 respectively. The contention probability of two paths is

$$p_c(i_1, j_1, i_2, j_2) = (oX + oY)/(D_{i_1 j_1} \cdot D_{i_2 j_2}) \tag{22}$$

For any two tasks t_{i_1} and t_{i_2}, let $\mathcal{T}_i$ be the set of successors of task t_i. Then the measure of contention of communication is

$$P_{i_1 i_2} = \sum_{j_1 \in \mathcal{T}_{i_1}, j_2 \in \mathcal{T}_{i_2}} (abs(f^u_{i_1} - f^v_{i_2}) \leq \xi) \cdot o_{i_1 j_1} \cdot o_{i_2 j_2} \cdot p_c(k_{i_1}, k_{j_1}, k_{i_2}, k_{j_2}) \tag{23}$$

According Equation **??**, the measure for a mapping and scheduling strategy is

$$P_c = \sum_{i,j=1}^{|T|} P_{ij} \tag{24}$$

We apply the average contention probability to evaluate the quality of the allocation strategy, which is denoted by

$$\bar{P}_c = \sum_{i,j=1}^{|T|} abs(P_{ij} - P_c/(|T| \cdot N)) \tag{25}$$

Based on the discussion above, we can have three objectives to be optimized: makespan, total amount of energy cost, and contention:

$$minimize(\mathcal{M}) \tag{26}$$

$$minimize(E_{pd} + E_{is} + E_m) \tag{27}$$

$$minimize(\bar{P}_c) \tag{28}$$

Discussion. Constraint **??** and Objective **??** are complementary. Constraint **??** considers the temporal relation of contention, and separates the potential

overlapped communication mandatory to avoid contention. Objective **??** focus on spatial contention and tries to minimize the overlapped communication paths.

5 The hybrid search algorithm

As a result of the complexity of the multi-objective optimization and the enormity of searching exploration, some efficient strategies should be proposed to differentiate non-dominated solutions and enhance the selection pressure towards the pareto front. Accordingly, we have integrated a pareto local search method into the NSGAII framework together with objective-related heuristic extensions, aiming to tackle the mapping and scheduling on NoC paradigm. This approach consists of three stages of optimization based on the characteristic of the model mentioned above. The first stage is a task-grouping process, which aims to cluster the tasks with more dependence into one core. It is obvious that if more dependent tasks are gathered in the same core, less communication will occur, without other factors considered in this stage. Then, due to the heterogeneous architecture and communication constraints, a core-tile mapping stage is needed to optimize the makespan and average contention probability. An inadaptable mapping will cause more congestion or remote transmission distance. At last, we are looking for a better sequence scheduling in each tile so as to minimize the makespan and satisfy the response time requirement between certain tasks. It should be noted that these three stages are initialized in a heuristic way in order, but are evolved in the evolution process as a whole. The optimization of each stage can be regarded as a multi-objective problem, so the incorporation of these three difficult problems make the mapping and scheduling more intractable. Also, three realistic objectives are considered, which increases the difficulty of this issue to a great extent. Detailed information about the algorithm is discussed in the following subsections.

5.1 Encoding

Before introducing the evolutionary algorithm, the representation of chromosome, which indicates the form of solution to the problem, is given. As is discussed in section 3, the location of each PE can be determined by a pair of coordinates (x_i, y_i) within 2D mesh topology, which is troublesome for genetic operators. Each operative tile p_k in the NoC has an associated gene which encodes the identifier of the core mapped in the tile. Thus, for a $n * m$ Noc, the chromosome encoding for the mapping and scheduling problem is shown in Fig.**??**. Every chromosome contains a set of mapped tasks, and the scheduled execution sequence for the allocated processors, which varies from conventional encoding for mapping optimization. Therefore, we need extra pointers to separate the genes for various processors. Moreover, three tips should be considered.

(1) $p_k \in P'$ is guaranteed, which means only the occupied processors are taken into consideration in the chromosome.

(2) p_k can be translated into the location of 2-dimension coordinates (x_i, y_i) in such a way: $x_i = k/m, y_i = k \ mod \ m$.
(3) The index of p_k may not be continuous but should be in ascending order for the reason of idle tiles. As is shown in Fig.**??**, p_2 does not appear because no tasks are assigned on tile $(0, 2)$. However, if there are tasks migrating to tile $(0, 2)$ during evolution process, p_2 should be added into the position between p_1 and p_3 in order to map the tile $(0, 2)$.

Next, we will discuss some primary components of the algorithm in detail and provide the whole framework.

Fig. 3. Chromosome encoding

5.2 Initialization

The idea behind the proposed initialization is to blend some priori knowledge or heuristic information into the pure random construction adopted by general research, for the reason that the exploration space for three-objective is fairly wide. 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. By this token, some candidate solutions would be generated randomly to cover a more comprehensive solution domain, while a few constructive solutions are also needed to store problem-heuristic information for a faster convergence. So given the three stages of optimization, the initialization process could be divided into t-

wo steps, and the scheduling optimization is mixed together with grouping and mapping to reduce the complexity of the method.

We adopt different strategies against various situation of the task graph. Most techniques deflect to the reduction of communication cost, as communication is the major concern in the optimization process. The first heuristic solution for grouping is generated by following steps, which takes makespan and communication into consideration.

(1) If the task graph is disconnected, the number of groups we classify is equal to the amount of connected components, and each connected component is regarded as a group. In this way, no packets will be transmitted in the network.
(2) Conversely, if there are no disconnected graphs in the task graph, a distribution of parallel tasks is recommended to be executed simultaneously. For each task t, in-degree $In(t)$ is calculated according to the dependency relation of tasks. For the task whose $In(t)$ is zero, it is pushed into the execution sequence, and if there are other tasks whose $In(t)$ is zero, they are pushed into the execution sequence and locked. Every time an unlocked task is popped and assigned to the current group. Then, the in-degrees of its successors decrease by 1. When no tasks are allowed to allocate, unlock one task in the execution sequence and divide it into a new group. Loop continues until all the tasks are assigned. The sequence of popped tasks also represents a scheduling plan to the problem, because every task is guaranteed to be executed after its dependent tasks.
(3) It should be noted that too many parallel tasks will cause an exceeding number of groups comparing with limited PEs. Thus, some groups should be mapped into the same PE. The fulfilment is inspired by the fact that the processors with minor tasks will be idle for a long time, which is a waste of energy consumption. So, the groups are ranked in descending order by the computation time, and we try to merge the last two groups into a new one every time until the number of groups meets the quantitative requirement. The rule of combination follows an insertion-in-turn operator, meaning that the element selected to be inserted into the new group from one group is followed by the task from another, which keeps the sequence of tasks in the original group unchanged. For the tasks whose execution sequence are inadequate within the group, delay them until their dependent tasks having been performed.

The output of grouping stage is an input of mapping process, including a set of groups.

In order to accelerate the convergence of mapping, an easy implementing called Spiral algorithm is adopted as [**?**] does. Different from the original task searching, a group searching is proposed to reduce communication cost ulteriorly. The placement of a group is searched in a spiral path from centre to the boundary of the network architecture. It tries to place the communicating groups close to each other, decreasing the transmission path between two PEs. Furthermore, the characteristic of spiral architecture makes communicating groups locate on

different directions in order to lower the probability of potential contention. With the mapping rules above, the transmission cost between each group and the total cost of one group is calculated respectively. Afterwards, the most costly group is assigned to the center and next to it is the group with the highest transmission cost to communicate. Break ties randomly if there exist more than one feasible groups. The initial solution with heuristic information is hereby generated and other solutions are produced in a random mode to cover a wider searching space. In this way, we can obtain an initial population with a balance between the diversified population and the convergence. Then the initial population will be divided into several sub-populations by the non-dominated sorting strategy in NSGAII, each of which is sorted according to the crowding distance to ensure a wider distribution.

5.3 Genetic Process

In this process, we propose an 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 performs a random selection of two parents from the dynamic elitist population and the evolution population. Then each task in the offspring inherits the position in one of its parents randomly. So, the positions of every task in the parents are recorded. The uniform crossover can help avoid the case that a task is assigned to different processors.

The mutation operator focuses on the conversion of task mapping for the purpose that subsequent local search is applied based on the fixed mapping, which is easier to trap into the local optimum given a constant mapping. To be specific, each individual receives a mutation probability, neither too small nor too large. A small probability will cause a long time stagnation on evolution, while a large one will lead to an insufficient search for the existing structure. For mutation, an occupied probability P_o is calculated as $P_o = |P'|/|P|$, and a split perturbation will be adapted on a probability of P_o, together with a merge perturbation executed on a probability of $(1 - P_o)$. In this way, it is possible to adjust the number of running processors dynamically according to the occupied status. More processors are likely to be allocated if tasks have gathered into a few processors, and the combination of processors with low utilization is preferred if the distribution of tasks is scattered. In a split operator, the most time-consuming processor is supposed to decompose into two light processors, each of which shares half of the computation load. The new generated processors are mapped into the first two available PEs in terms of the spiral rule. As for the merge operator, it is performed the same as initial procedure does except that the merge stage is conducted only once and the incorporative processor is mapped into the first available PE according to the spiral rule.

5.4 Pareto local search

In the ϵ-MOHA algorithm, we use a pareto local search (Alg**??**) to enhance the intensive searching capacity. The procedure consists of a repair step for infeasible solutions, an insert-based neighbourhood comparison and generation step.

A repair operator on the infeasible solutions is needed to spare the resource (Alg**??**.line 2). The main idea of *repair* operator is to repeatedly check the feasibility of executing tasks and ensure that each task begins to execute later than all its dependent tasks.

Another important factor in the Pareto local search procedure is the neighborhood structure. Given a candidate solution S and two tasks t and t', if we only insert the position of the task t into the head of task t' and fix the positions of other tasks, we can obtain a neighbor solution of S. We then use *neighbor(S_t)* to denote the set of neighbors of S by inserting t into other positions. In the main Pareto local search loop (Alg**??**.line 3-10), the algorithm randomly selects a task t of solution S (Alg**??**.line 4) and tries to find a solution that can ϵ-dominate S from its neighborhood space (Alg**??**.line 5). Specifically, if *neighbor(S_t)* is better than S based on Pareto optimal evaluation, it will replace S for further exploitation and *iter* is set to 1 to look for a better neighorhood than the current. Meanwhile, the elitist population S_E is updated (Alg**??**.line 6-8). Otherwise, another task will be selected and *iter* will be increased by 1 (Alg**??**.line 10). This process continues until *iter* reaches the maximum limit.

Algorithm 1 Pareto-Localsearch(S)

```
1: Input: a starting solution S, the maximum allowed iterations MaxIter
2: Output: Elite solution set S_E
3: iter = 1;
4: repair(S);
5: while iter ≤ MaxIter do
6:     t= Random-Task-selection(S);
7:     Nh(S_t) = Neighborhood(S_t);
8:     if Nh(S_t) ϵ-dominates S then
9:         S=Nh(S_t); S_E=Update(S);
10:        iter = 1;
11:    else
12:        iter = iter + 1;
13: return S;
```

5.5 Value-based Dominance

The contention-aware task mapping and scheduling problem with three objectives has a huge solution domain, which is exhausive to solve and will produce a large amount of solutions. For a decision maker, too many candidate solutions are more likely to bring extra artificial workload. Consequently, a value-based dominance method is adopted to enlarge the dominating area of the non-dominated solutions so that some of them are more likely to be dominated by others. Among the value-based dominance category, ϵ-dominance proposed by Laummans et al

[**?**] has drawn greater attention and achieved a superior performance. So, it is also integrated into our algorithm called ϵ-MOHA. Given two solutions $x, y \in \Omega$ and $\epsilon > 0$, x is said to ϵ-dominate y, iff $\forall i \in \{1, ...m\}$, $(1-\epsilon)f_i(x) \leq f_i(y)$. For example, if ϵ is set to 0.1, and the makespan, total amount of energy cost, contention of solution $x = (100, 200, 0.9)$, another solution $y = (102, 100, 0.9)$. x and y are non-dominance solutions and will be determined by decision makers. As can be seen, the makespan of y is slightly worse than x, but the energy cost is much better than x, leading to the preferred selection of solution y. Under the concept of ϵ-dominance, only solution y is recommended to decision makers, which reduces the Pareto set.

6 Experimentation

This section presents computational experiments to test the efficiency of our ϵ-MOHA algorithm. The effectiveness of the model and the efficiency of the algorithm have been evaluated with realistic benchmarks, including an H264 decoder [**?**], a TMNR procedure [**?**], a Multi-Window Display (MWD) application [**?**], a Livermore Loop and a Bufferfly [**?**], and an MP3 decoder. All the experiments are performed on a PC with intel 2.2 GHz processor and 8.0 GB memory.

6.1 Experimental setup

The constant factors are set to E1=240μW/MHz for the energy cost of per processor clock cycle. E2=24μW/MHz for per hop cost for router , and E3=6.8μW/MHz for per packet transmission [**?**].

6.2 Random generated cases

Effectiveness for contention-aware optimization To evaluate the effectiveness of Constraint **??** and Objective **??**, we set three groups of experiments: only consider Constraint **??** (CC), only consider Objective **??** (OC), and consider the two (OCC). We compare performance, energy consumption, and the number of contentions in Table **??**.

Table 3. Contention-aware strategy comparison

Case	Scale	CC			OC			OCC		
	tasks	edges	makespan	energy	contention	makespan	energy	contention	makespan	energy	contention

Comparison with other algorithms To show the efficiency of MOHA, we compare the performance with NSGAII.

6.3 Real benchmarks

7 Conclusion