A hybrid algorithm for multi-objective mapping and scheduling optimization of MPSoCs

***Abstract*—Multi-processors are widely used in embedded systems. The resource limitation requires designers to find optimizations among various design considerations. In this paper, we model tradeoffs between makespan and the balance of processor workloads for the problem of tasks mapping and scheduling in a multi-processor environment. To encode the effect of precedence relations between different tasks in scheduling, the model also considers communication cost in the formulation. Furthermore, to approximate the Pareto front of the multi-objective optimization problem, we propose a multi-objective hybrid algorithm(MOHA) by integrating pareto local search into an evolutionary process. We evaluate the efficiency of the method on a series of benchmarks and compare the results with existing multi-objective optimization methods. Experimental results show that the proposed algorithm can obtain better solutions for various real-world cases, compared with state-of-the-art methods.**

I. Introdution

Mapping and scheduling applications onto a multi-core platform is a key factor, to cope with continuously increasing conflicting demands of both high performance and low power consumption. For example, makespan minimization requires allocating more computations on faster processors. Consequently, the unbalanced workload may keep fast processors busy in computation, while other processors being waiting, which leads to more power consumption. On the other hand, computation workload balance leads to a more parallel implementation. However, the increasing communication between tasks on different processors may result in the increase in the makespan. Therefore, there exists a set of mutually incomparable solutions representing different design tradeoffs.

To achieve high performance and low power consumption for MPSoCs, we consider workload balance and makespan minimization in static task allocation and scheduling problems. Meanwhile, we model the corresponding constraints together, to avoid local optimal obtained from separated optimization of the two criteria.

In this context, the traditional solution for mapping and scheduling problems with integer linear programming (ILP), is less feasible to cope with multiple objectives and large scale systems [14], [4]. There are various algorithms and heuristics for computing solutions for multi-objective optimization problems. For example, evolutionary algorithms [13], [7], based on successive optimization and simulation steps, use heuristics to refine optimal solutions. Some constraint solvers are also applied to approximate the Pareto fronts and guarantee the computable bounds [8]. However, the efficiency and scalability are still the bottlenecks for the methods to be applied for large scale applications. To tackle this problem, we propose an alternative method, which is based on a mixture of multi-objective genetic and local search techniques. By integrating a pareto local search into an evolutionary procedure, we could obtain both the expansive searching ability of the population-based technique and the intensive searching ability of the local search method. Specifically, the proposed algorithm mainly consists of three ingredients, including a problem-specific initialization to generate promising initial candidate solutions, a genetic operator to roughly search the space, and a local search operator to exhaustively seek through the better feasible solutions.

Compared with the-state-of-the-art methods, the algorithm is more efficient for computing Pareto optimals, and is capable of dealing with large scale realistic applications.

The contributions are as follows. First, we propose to optimize task allocation and scheduling of MPSoCs with respect to two criteria, i.e., makespan minimization and workload balance, involving both performance and energy consumption considerations. Second, to avoid local optimization from stepwise methods or decomposition methods, we model the constraints on mapping and scheduling together. The model characterizes computation, inter-processor communication, and can be extended for other optimization criteria. Finally, to provide solutions for realistic applications, we propose a Pareto local search based algorithm within a novel genetic framework, called Multi-objective hybrid algorithm(MOHA), to approximate Pareto optimal. Experimental results show that we could obtain the solutions even for realistic applications with more than two hundred tasks.

The paper is structured as follows. After a brief summary of the related work, we define task graphs for application models, provide the assumption for target platforms, and describe the mapping and scheduling problem in Seciton III. Section IV models constraints for mapping and scheduling problem. Section V introduces MOHA algorithm. We present the experimental results in Section VI. Section VII concludes.

II. Related work

Optimizations on computation, communication, throughput and energy consumption are always conflicting [9], [5], [16]. When facing multiple objectives, a solution is to formulate objectives with different weights [5], thus transforming multi-objective into one to be solved with integer linear solvers. However, the shortage of scalability limits its application for realistic problems. The alternative methods, such as stepwised optimization [5], the integration of evolutionary algorithms [9], may still fall into the problem of local optimization, and the results may not be real Pareto fronts.

Search based algorithms are also considered in the literature for mapping and scheduling optimization of MPSoCs. Zhu et. al. model the constraints and task dependency into timed automata and compute Pareto front with model checking techniques [16]. [8] provides a multi-dimensional binary search algorithm for approximating the Pareto fronts and guaranteeing the computable bounds. These techniques also suffer from scalability problems.

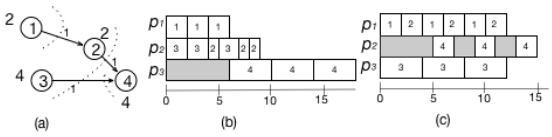


Fig. 1. A running example.

To obtain Pareto optimal for large scale systems, [10] addresses the mapping problems with multiple optimization objectives, based on evolutionary algorithms. [15] applies genetic algorithms for multi-objective optimization in workflow scheduling. Among the techniques, the Non-dominated Sorting Genetic Algorithm (NSGAII) [3] is recognized as one of the most classical framework for multi-objective optimization problems. In this setting, initial solution and heuristics are important factors for the efficiency of the algorithms. On the other hand, local search based methods have achieved great success in solving various combinatorial problems by obtaining near-optimal solutions with a small cost [11]. Therefore, the aim of this paper is to integrate a local search technique into a genetic framework to improve both the efficiency and accuracy when computing Pareto optimals.

III. Preminaries

This section first defines the concept of an application model to be mapped and scheduled. Further, the assumption of the target platform is presented. Finally, the problem to be optimized is sketched.

A. Weighted task graph

In this paper, we concern coarse-grained application models. An application consists of a set of tasks with well-defined precedence constraints. Therefore, we adopt task graphs as the formulation [2]. To record the amount of work to be dealt with on a task, and data to be transferred, we assign weights on the tasks and data flows between different tasks, respectively.

Definition 1 (Weighted Task Graph): A weighted task graph is a tuple $G = \langle T, E \rangle$, 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.

If there is no explicit precedence relation between a pair of tasks, the amount of communicated data is zero. We also assume that the graph is acyclic. For an application with loops, we need to reduce it to acyclic graph by unfolding techniques. Take the model in Fig. 1 as an example. There are four tasks, each of which is labelled with the amount of work, and three edges are labeled with the amount of transfered data.

B. Target platform

We consider architectures connected with heterogeneous processors, each of which has a private memory, as shown in Fig.2. Each processor can access its private memory directly, while accessing private memories of peer processors is not permitted. In order to communicate with peer processors, DMAs (Direct Memory Access) are implemented in the hardware platform. They can read from and write to all private memories under the control of processors. We make the following assumptions for the target architecture:

1) The cost for intra-processor communication is negligible. We only consider inter-processor communication.

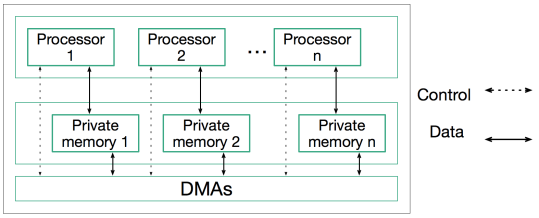


Fig. 2. The target architecture.

2) A processor is occupied for initializing communication. Once the initialization is ready, the processor is released.
3) An inter-processor communication can start after the initialization. And multiple transmission to different destinations can be executed simultaneously.
4) There is no memory access collisions.
5) Private memories are big enough for data storage.

When the weight of a precedence relation in a task graph is greater than zero, and the two tasks are allocated to different processors, inter-processor communication is required. To simplify the model description, we restrict the communication architecture to a direct connection between different processors via inter-processor communication.

C. Problem description

We aim to construct static schedules for the target implementation. More precisely, the tasks are statically allocated among the processors and all executions (instances) of a task are executed on the same processor, and no instance is permitted to migrate among processors. Hence, mapping relations between tasks and processors are fixed during execution, and the sequences of task execution remain unchanged.

The tasks of an application will be repeatedly executed for a certain scenario. Therefore, it is not enough to estimate the makespan of the whole with one time execution of the tasks. And we consider the minimisation of makespan within certain number of iterations.

Given a task graph G and a set of processors P, a mapping is a function $m : T \times P \to \mathbb{B}$ with $m(t_i, p_k) = true$ meaning that task t_i executes on processor p_k. Then the scheduling of tasks is a function $\tau : T \times \mathbb{N} \to \mathbb{N}$ with $\tau(t_i, u) = n$ meaning that the time of uth invocation of task t_i is n. Suppose the function of processor speeds is $\rho : P \to \mathbb{N}$. The duration of executing t_i is $\kappa(t_i) = \delta(t_i)/(\sum_{k=1}^{|P|} \rho(k) * m(t_i, p_k))$. By abuse of notation, $\tau(t_i, u)$ is denoted by τ_i^u and others only use the subscripts instead. Then given the number of iterations N, the makespan is

$$\mathcal{M} = max\{i : \forall i \ \tau_i^N + \kappa_i\}.$$

The ideal situation for a balanced workload is that the workload on each processor is equal, i.e., $aver = \sum_{i=1}^{|T|} \delta_i / |P|$. Hence, the difference between real workloads and the perfectly balanced workload is measure with

$$\mathcal{W} = \sum_{k=1}^{|P|} \mid \sum_{i=1}^{|T|} \delta_i * m_{i,k} - aver \mid .$$

Consider the example in Fig. 1 again. There are three processors, where the speed of p_2 is twice as fast as that of p_1 and p_3. In Fig. 1(b), tasks 1 and 4 are mapped to processors p_1 and p_3, respectively, and tasks 2 and 3 are allocated to p_2. If we ignore the cost for communication initialization, the makespans of first and third iterations in Fig. 1(b) are 10 and

TABLE I. INPUT OF THE ILP MODEL

Notation	Definition
N	The number of executions for every task
$a_{i,k}$	Whether task t_i can be mapped to processor p_k
$d_{i,j}$	Precedence relation between tasks t_i and t_j
$c_{i,j}$	The size of data transfer from task t_i to t_j
δi	The amount of work for task t_i
η	The cost for communication initialization

18 respectively. If we change the allocation to Fig. 1(c), we can obtain a more balanced workload with a smaller makespan.

IV. CONSTRAINT FORMULATIONS

In this section, for the problem of mapping and scheduling of an application on certain platforms, we formalize it as an integer linear programming model, which is the input of our algorithm. The model mainly constructs constraints on scheduling, with respect to computation effort. To measure the introduced communication cost for workload balance and its effect to the makespan, we also consider the inter-processor communication in the optimization of the two objectives. To simplify the modelling, we assume that a task can start execution once the corresponding communication is finished.

A. Parameters and variables

Given a model $G = \langle T, E\rangle$ and a set of processors P, the input parameters to the ILP model is summarized in Table I.

With the input parameters, the mapping and scheduling steps consist of 1) allocating each task to a unique processor; 2) scheduling the tasks mapped to different processors. Then the following decision variables can completely capture the allocation and scheduling information.

- $m_{i,k}$: shows whether task t_i is mapped to processor p_k;
- $q_{k,i,j}$: the execution sequence between tasks t_i and t_j on p_k;
- τ_i^u: the start time of task t_i at uth iteration

To facilitate the modelling of constraints, we introduce the following auxiliary variables: 1) $b_{i,j}$: indicates whether tasks t_i and t_j are mapped to the same processor; 2) $bp_{k,i,j}$: indicates whether tasks t_i and t_j are mapped to processor p_k; 3) $o_{i,j}$: indicates whether inter-processor communication exists between tasks t_i and t_j. These variables are used to check the existence of communication cost and the precedence executing sequence of tasks on same processors.

B. Constraints

To encapsulate both the mapping and scheduling solution restricted by the underlying target architecture, we introduce two categories of constraints.

The first category of constraints restricts the mapping of tasks on processors, the inter-relation between dependent tasks, and tasks mapped on the same processors.

$$\sum_{k=1}^{|P|} m_{i,k} == 1, \quad \text{and} \quad m_{i,k} \le a_{i,k} \tag{1}$$

$$b_{i,j} == 1 - (\sum_{k=1}^{|P|} \mid m_{i,k} - m_{j,k} \mid)/2, \ \textit{for } i \neq j \tag{2}$$

$$\sum_{k=1}^{|P|} bp_{k,i,j} == b_{i,j} \quad \text{and } bp_{k,i,j} \le (m_{i,k} + m_{j,k})/2 \tag{3}$$

$$o_{i,j} \ge d_{i,j} - b_{i,j} \quad \text{and} \quad o_{i,j} \le d_{i,j} \tag{4}$$

$$\sum_{k=1}^{|P|} q_{k,i,j} == 1 \text{ and } \sum_{i,j=1}^{|T|} q_{k,i,j} == (\sum_{i,j=1}^{|T|} bp_{k,i,j})/2$$
$$q_{k,i,j} + q_{k,j,i} \ge bp_{k,i,j} \text{ and } q_{k,i,j} * q_{k,j,l} \le q_{k,i,l} \tag{5}$$

Constraint 1 requires every task can only be mapped to a unique and valid processor. Constraints 2,3 show that the physical relations between two tasks can be inferred from the mapping relation. Constraints 4 indicates that inter-processor communication is required for two tasks with precedence relation, which are not allocated to the same processor. Constraint 5 presents that the execution relation of tasks on the same processor is static and transitive.

The second category of constraints regulates the behavior of executed tasks on same processors, different processors and within various iterations. Though the cost of data transmission is invisible for processors, the cost for initializing communication and the duration for waiting data to be transmitted cannot be ignored. As we assume that data from one source can be transmitted to different destinations simultaneously, transmitting data from the same source only needs one initializing. In the following, we introduce variable $e_i^u = \tau_i^u + \kappa_i + \eta * ((\sum_{l=1}^{|T|} o_{i,l}) \ge 1)$, to be the end time of processor occupation for task t_i at uth iteration, whose duration includes task execution and communication initialization.

$$\mathcal{M} \ge \tau_i^N + \kappa_i \tag{6}$$

$$e_i^u \le \tau_i^v \text{ for } u < v \tag{7}$$

$$d_{i,j} \Rightarrow \tau_j^u \ge e_i^u + o_{i,j} * c_{i,j} \tag{8}$$

$$q_{k,i,j} \Rightarrow \tau_j^u \ge e_i^u \tag{9}$$

$$b_{ij} \Rightarrow \tau_i^u \ge e_j^v \vee \tau_j^v \ge e_i^u \tag{10}$$

$$q_{k,i,j} \Rightarrow \tau_j^v \ge e_i^u \text{ for } u < v \tag{11}$$

Constraint 6 requires that the makespan of the execution should consider the termination of all the tasks. Constraint 7 describes that the execution of a task at vth iteration should wait for the finish of its uth iteration, if $u < v$. Constraint 8 shows that the start time of its execution for t_j at the uth iteration is later than the finish time of t_i at the same iteration, if t_j depends on t_i. Constraint 9 says that the execution of two tasks on the same processor is sequential. And Constraint 10 requires that the execution of two tasks on the same processor cannot overlap. Constraint 11 requires that the execution of task t_j in later iteration v should not be earlier than any other tasks executed on the same processor in an earlier iteration u, for $u < v$. This constraint can be relaxed if we allow one task to be executed several times before other tasks. These constraints will finally be transformed to normal forms of integer linear programming models with Big-M method [12].

C. Objective

To minimize the makespan and obtain a balanced workload, we have two goals for minimization, which are discussed in the abrove section.

$$minimize(\mathcal{M}), \quad \text{and} \quad minimize(\mathcal{W}) \tag{12}$$

V. MULTI-OBJECTIVE HYBRID ALGORITHM

To solve the bi-objective optimization problem discussed above, a natural idea is to translate it into a single objective optimization problem, by assigning different weighted values to the goals, and solve it using some commercial tools such as

CPLEX[1]. But for the lack of sufficient information about the relation of the two objectives, this kind of translation usually can not obtain the optimum solutions. Another shortcoming is that though CPLEX is efficient when solving small scale problems, yet can not scale up for large scale applications.

To obtain Pareto optimal for large scale systems, we propose a novel algorithm framework, called Multi-objective hybrid algorithm (MOHA), by hybridizing the genetic algorithm with a Pareto local search. Specifically, the algorithm consists of three main parts, i.e. initialization component, evolution process component, and Pareto local search component.

A. Initialization

The initialization phase plays an important role in the effectiveness of the algorithm. The reason is that 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 easily. Accordingly, in our algorithm, a two-mode construction procedure is proposed (Alg1.line 1). More precisely, in the first mode, i.e., random mode, candidate solutions would be generated randomly to cover a more comprehensive solution domain; in the second mode, i.e., greedy mode, a greedy individual would be generated to store problem-heuristic information using the following strategy:

1) Let $|T|$ be the number of tasks, $|P|$ be the number of processors, and $aver = \sum_{i=1}^{|T|} \delta_i/|P|$ be the supposed average workload on each processor.
2) For each task t, in-degree $In(t)$ and out-degree $Out(t)$ are calculated according to the dependences of tasks.
3) For the task whose $In(t)$ is zero, it is pushed into the execution sequence and the corresponding $Out(t)$ is decreased by 1. Loop continues until all the tasks are assigned. One thing to be noticed is that the workload of each processor should be around $aver$.

In this way, the algorithm can obtain an initial population $S0$ with a balance between the diversified population and the convergence. After that, $S0$ will be divided into several sub-populations $(F_1, F_2, ...)$ by the *non-dominated sorting strategy* (Alg1.line 2-4). To evaluate the quality of solutions, we introduce the definitions of dominating relationships between solution A and solution B.

Definition 2: A dominates B ($A \preceq B$), *iff* all the objective functions of A are no worse than B.

Definition 3: A is non-dominated, *iff* $\nexists B, B \preceq A$.

Generally speaking, the *non-dominated sorting* strategy is to divide the current population Pop into several sub-populations $(F_1, F_2, ...)$. Each sub-population F_i contains a partial task sequence that is non-dominated by other parts except those sorted in front of F_i.

For each sub-population F_i, individuals are sorted by the *crowded comparison* strategy with a descending order (Alg1.line 5-6). Specifically, the *crowding distance* of the two boundary individuals are set to be ∞ to maintain a diversity. Given a candidate solution S, let S_x, S_y be the adjacent individuals of S. Then the *crowding distance* of S is calculated as follows:

$$Dis(S) = \sum_{k=1}^{m} \mid S_x.f_k - S_y.f_k \mid \quad (13)$$

where m denotes the number of objectives and f_k denotes the value of objective k. This strategy ensures a wider distribution of individuals so that too crowded solutions will be ignored for the next evolution.

[1]https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Algorithm 1 Multi-objective Hybrid Algorithm

1: $S_0 = Construct()$
2: $(F_1, F_2, ...) =$ Non-Dominated-Sort(S_0)
3: $S_0 = (F_1, F_2, ...)$
4: $S_E = F_1$
5: **for all** $F_i \in S_0$ **do**
6: Crowding-Distance-Assignment(F_i)
7: set $t = 0$
8: **while** stopping criterion not satisfied **do**
9: $Q_t =$ Generate-Child-Population(S_t)
10: $R_t = R_t \cup Q_t$
11: $(F_1, F_2, ...) =$ Non-Dominated-Sort(R_t)
12: $F = (F_1, F_2, ...)$
13: $S_{t+1} = \phi$
14: $i = 1$
15: **while** $|S_{t+1}| + |F_i| < N$ **do**
16: Crowding-Distance-Assignment(F_i)
17: $S_{t+1} = S_{t+1} \cup F_i$
18: $i = i + 1$
19: sort F_i on crowding distances
20: $S_{t+1} = S_{t+1} \cup F_i\,[1 : (N - |S_{t+1}|)]$
21: $t = t + 1$
22: return F_1

B. Evolution Process

After the initial phase, the evolution process will be called repeatedly to generate the Pareto front (Alg1.line 7-22). In this phase, a new population of candidate solutions will be generated by the procedure *Generate-Child-Population()* (Alg1.line 9-10). Then, the candidate solutions will be sorted by the *non-dominated sorting* strategy (Alg1. line 11-12). To choose the best N solutions as the beginning of next iteration, $(F_1, F_2, ...)$ is orderly added into $S_t + 1$ (Alg1. line 15-18). Note that the solutions in F_i with larger crowding distances have a higher priority to be selected (Alg1. line 19-21).

Algorithm 2 Generate-Child-Population(S)

1: $Q = \phi$
2: **while** $|Q|! = |S|$ **do**
3: (S_x,S_y)= Stochastic-Selection()
4: $q =$crossover(S_x, S_y)
5: **if** $rand \geq 0.5$ and $rand \leq 1$ **then**
6: Light-Perturbation(q)
7: **else if** $rand \geq 0$ and $rand < 0.5$ **then**
8: Heavy-Perturbation(q)
9: $q\star =$Pareto-Localsearch(q)
10: update($S_E, q\star$)
11: $Q = Q \cup q\star$
12: return Q

Obviously, the procedure *Generate-Child-Population()* plays a key role in this process (Algortihm 2). In this procedure, firstly, two parent solutions will be selected from both the current population and the elitist population (Alg2.line 3). Then, a crossover operator (Alg2.line 4) is applied to change the information between these two chosen parents, which means that the position of each task in the offspring is picked randomly from one of its parents. Moreover, for the sake of increasing the diversity of solutions, two perturbation operators are performed in mutation process (Alg2.line 5-8). The former is to swap two different tasks in any processors and the latter is to insert one task into the position of a different processor. After that, a Pareto local search procedure (Alg2.line 9) will be called, which we will discuss in the next subsection.

C. Pareto Local Search

To enhance the intensive searching capacity and accelerate the convergence of our algorithm, we introduce the Pareto local search procedure (Algorithm 3). It should be pointed out that though infeasible solutions are allowed in genetic process for more combinations, these infeasible solutions need to be repaired to feasible solutions for sparing the resource (Alg3.line 3). Let $X = (x_1, x_2, ...x_T)$ be the current executing sequence where T is the number of total tasks. For each task x_i, we use $dp(x_i)$ to denote the number of tasks that depend on x_i, and $dpd(x_i)$ to denote the set of tasks that depend on x_i. 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 r and t, if we only swap the positions of the tasks r, 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 swapping t with other tasks. In the main Pareto local search loop (Alg3.line 4-11), the algorithm randomly selects a task t of solution S (Alg3.line 5) and tries to find a solution that can dominate S from its neighborhood space (Alg3.line 6). Specifically, if *neighbor(S_t)* is better than S based on Pareto optimal evaluation, it will replace S for further exploitation and k is set 1 (Alg3.line 7-9). Otherwise, another task will be selected and k will be increased by 1 (Alg3.line 10-11). This process will continue until k reaches the maximum limit.

D. The Effectiveness of MOHA

In this section, we prove the effectiveness of MOHA for the problem of mapping and scheduling optimization for MPSoCs.

(Lemma 1) *The sufficient condition for a feasible solution of MPSoC is that for each* $x_i \in X = (x_1, x_2, ...x_T)$, $(i = 1, 2, ...T)$, $dp(x_i) = 0$.

proof. $dp(x_i) = 0$ for $(i = 1, 2, ...T)$ means that the current task x_i has no dependent tasks or its dependent tasks have all been executed before it. In this situation, x_i can go on if the relevant processor is idle. That implies that no tasks will be trapped into infinite waiting during the whole execution. It illustrates the feasibility of the solution.

(Theorem 1) *Given an MPSoC problem, solutions generated by MOHA are feasible.*

proof The initial value of $dp(x_i), (i = 1, 2, ...T)$ is assigned according to the number of its dependent tasks. Once a task is executed, $dp(dpd(x_i))$ is decreased by 1. When the task x_i needs to be executed and $dp(x_i) > 0$, the *repair* operator of MOHA puts one of its dependent tasks just in front of it to decrease $dp(x_i)$. It loops until the values $dp(x_i)$ of all tasks are 0. According to *Lemma 1*, the solutions are feasible on the MPSoC problem.

Algorithm 3 Pareto-Localsearch(S)

```
1:  k = 1
2:  kmax = 2 * n
3:  repair(S)
4:  while k ≤ kmax do
5:      t= Random-Task-selection()
6:      Nh(S_t) = Neighborhood(S_t)
7:      if Nh(S_t) dominates S then
8:          S=Nh(S_t)
9:          k = 1
10:     else
11:         k = k + 1
12: return S
```

VI. Experimentation

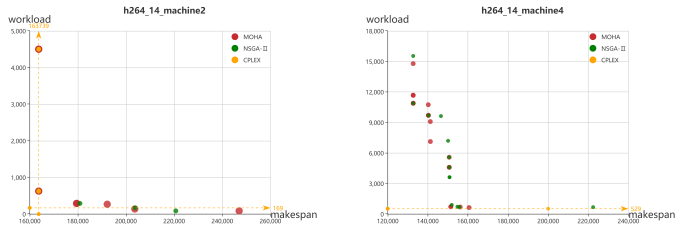
We have implemented the MOHA in C++. The efficiency of the algorithm over the constraint model has been evaluated with realistic benchmarks, including an H264 decoder [5], a TMNR [2], a Multi-Window Display (MWD) application [1], a Livermore Loop and a Bufferfly [6], and an MP3 decoder.

TABLE II. Experimental parameters

Items	Value
the number of iterations for stopping criterion	500
the size of population	100
the size of elitist population	100
non-updated times	$number\_of\_tasks/3$
execution possibility of mutation operator	0.1
the ratio of the probability of two perturbations	1:1

All the experiments are performed on a PC with intel 2.2 GHz processor and 8.0 GB memory. The detailed parameters in MOHA are listed in Table II.

A. Efficiency comparison with classical methods



(a) H264 with 2 processors (b) H264 with 4 processors

Fig. 3. Distribution of final sets obtained by MOHA and NSGAII

To evaluate the efficiency of our method, we have compared with the results from multi-objective evolutionary algorithm NSGAII [3] and CPLEX over small scale examples. To compare with CPLEX, the two objectives are transformed into one by the introduced weights, whose ratios are sampled from (1:1) to (1:9999) and (9999:1) respectively. As the timeout of CPLEX is set to 2 hours, we could only obtain its results within the scale of 20 tasks and 2 processors.

The Pareto fronts[2] of the coarse-grained H264 decoder generated by MOHA, CPLEX and NSGAII, are illustrated in Fig. 3. In the figure, the boundary of the two objectives are obtained by separated optimization for makespan and workload with CPLEX. And its Pareto fronts are calculated with various combination of weights. We can observe that the results of MOHA are closer to the Pareto-front than that of NSGAII, and they can reach some boundaries computed by CPLEX.

Additional to the comparison from the figure, we also present the comparison from Set Coverage (C-metric), which reflects the degree of dominance is measured between different algorithms. Let X and Y be two approximations of the Pareto-front of a multi-objective problems. We define $C(A, B)$ as the percentage of the solutions in B that are dominated by at least one solution in A, where

$$C(A, B) = \frac{u \in B | \exists v \in A : v \preceq u}{B}. \quad (14)$$

The higher $C(A, B)$ is, the deeper extent A dominates B.

The results of C-metric are listed in Table III. NA means the results are un-available within 2 hours. As we can see, most solutions in MOHA dominate both NSGAII and CPLEX.

TABLE III. COVERAGE VALUES BETWEEN MOHA, CPLEX, NSGAII

Benchmark	Tasks	Processors	C(MOHA, NSGAII)	C(NSGAII, MOHA)	C(MOHA, CPLEX)	C(CPLEX, MOHA)
mp3decoder	16	2 4	0.8 0	0.2 0	1 1	0 0
h264-14	14	2 4	0.4 0.58	0.17 0.15	0 0	0 0
MWD	12	2 4	0 0	0 1	1 1	0 0
TMNR	14	2 4	0.67 0	0 0	0.5 0	0.5 0.75
FFT-loop	28	2 4	0.67 0	0.33 0.33	1 NA	0 NA
FFT-butterfly	32	2 4	1 1	0 0	NA NA	NA NA

B. The scaliability of the MOHA algorithm

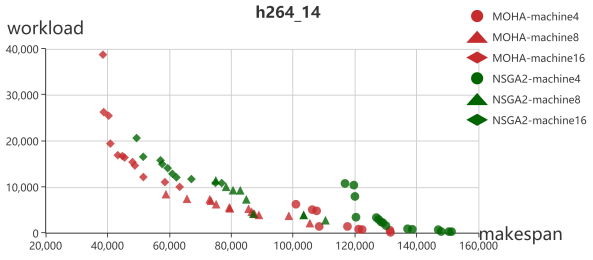


Fig. 4. Distribution of final sets for fine-grained H264

To show the scalability of our method, we have also evaluated the performance with the fine-grained H264 decoder example, with 264 tasks and 4, 8, 16 processors respectively. Its comparison with NSGAII is illustrated in Fig. 4. The results show that MOHA can gain better results even on the large scale applications.

VII. CONCLUSION AND FUTURE WORK

To meet increasing requirements on high system performance and low power consumption, we have proposed two optimization objectives, i.e., makespan minimisation and workload balance, for mapping and scheduling problem of MPSoCs. The model for optimization encapsulates task scheduling, with the consideration of both computation and communication cost. To deal with the optimization of large scale applications, we have also proposed a multi-objective hybrid algorithm. Compared from the generated solutions, our algorithm outperforms methods such as CPLEX and NSGAII. For the future work, we will refine the model to encompass more objectives to be optimized, as well as extending the current algorithm from two to more objectives optimization.

[2]As the precedent relations are simple in these benchmarks, we could only obtain a few solutions in the Pareto fronts.

REFERENCES

[1] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli. Network-on-chip design and synthesis outlook. *INTEGRATION, the VLSI journal*, 41(3):340–359, 2008.

[2] S. Cotton, O. Maler, J. Legriel, and S. Saidi. Multi-criteria optimization for mapping programs to multi-processors. In *SIES*, pages 9–17, 2011.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[4] K. Huang, M. Yu, R. Yan, X. Zhang, X. Yan, L. B. de Brisolara, A. A. Jerraya, and J. Feng. Communication optimizations for multithreaded code generation from simulink models. *ACM Trans. Embedded Comput. Syst.*, 14(3):59, 2015.

[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 Transactions*, 97-D(12):3072–3082, 2014.

[6] C. W. Kessler and J. Keller. Optimized mapping of pipelined task graphs on the cell be. In *CPC*, 2009.

[7] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[8] J. Legriel, C. Le Guernic, S. Cotton, and O. Maler. Approximating the pareto front of multi-criteria optimization problems. In *TACAS*, pages 69–83. Springer, 2010.

[9] J. Lin, A. Gerstlauer, and B. L. Evans. Communication-aware heterogeneous multiprocessor mapping for real-time streaming systems. *Journal of Signal Processing Systems*, 69(3):279–291, 2012.

[10] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping applications to tiled multiprocessor embedded systems. In *ACSD*, pages 29–40. IEEE, 2007.

[11] Y. Wang, S. Cai, and M. Yin. Two efficient local search algorithms for maximum weight clique problem. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 805–811, 2016.

[12] W. L. Winston, M. Venkataramanan, and J. B. Goldberg. *Introduction to mathematical programming*, volume 1. Thomson/Brooks/Cole Duxbury; Pacific Grove, CA, 2003.

[13] H. Yang and S. Ha. Pipelined data parallel task mapping/scheduling technique for mpsoc. In *DATE*, pages 69–74, 2009.

[14] Y. Yao, G. Zhao, and X. Wang. An ILP-based DMA data transmission optimization algorithm for mpsoc. *JCP*, 9(10):2461–2466, 2014.

[15] S. Yassa, J. Sublime, R. Chelouah, H. Kadima, G.-S. Jo, and B. Granado. A genetic algorithm for multi-objective optimisation in workflow scheduling with hard constraints. *International Journal of Metaheuristics*, 2(4):415–433, 2013.

[16] X. Zhu, R. Yan, Y. Gu, J. Zhang, W. Zhang, and G. Zhang. Static optimal scheduling for synchronous data flow graphs with model checking. In *FM*, pages 551–569, 2015.