In Proc 8th IEEE International Conference on Advanced Computational Intelligence Chiang Mai, Thailand; February 14-16, 2016

Set-based Particle Swarm Optimization for Mapping and Scheduling Tasks on Heterogeneous Embedded Systems

Xiao-Xiao Xu, Xiao-Min Hu*, and Wei-Neng Chen
Sun Yat-Sen University, Guangzhou, China
Key Lab. of Machine Intelligence and Advanced Computing
Ministry of Education, Guangzhou, China
Email: xiaominh6@mail.sysu.edu.cn

Yun Li
School of Engineering
University of Glasgow
Glasgow G12 8LT, U.K.

***Abstract*—Modern heterogeneous multiprocessor embedded platforms is important for the high volume markets that have strict performance. However, it presents many challenges that need to be addressed in order to be efficiently utilized for multitask applications. Since mapping and scheduling problems for multi processors belong to the classic of NP-Complete problems, common methods used to solve this kind of problem usually fail. In this paper, we present an algorithm based on the meta-heuristic optimization technique, set-based discrete particle swarm optimization (S-PSO), which efficiently solves scheduling and mapping problems on the target platform. This algorithm can simultaneously addressed the mapping and scheduling problems on a complex and heterogeneous MPSoC and it has better performance than other algorithms in dealing with large scale problems. This algorithm also reduces the execution time of the application by exploring various solutions for mapping and scheduling of tasks and communications. We compare our approach with other heuristics, Ant Colony Optimization (ACO), on the performance to reach the optimum value and on the potential to explore the target platform. The results show that our approach performs better than other heuristics.**

Keywords—set-based discrete particle swarm optimization (DPSO); mapping; scheduling; communications.

I. Introduction

The multiprocessor System-on-Chip (MPSoC) is a system-on-a-chip (SoC) which is made of multiple processors, usually designed for embedded applications [1]. All these components are connected into a whole through the on-chip interconnect. In order to address different kind of applications, the processors are usually designed to be heterogeneous. We use different processors to accelerate the different parts of the application to optimize the performance of the multitask applications.

In order to optimize the program performance, e.g., the program execution time, we need to find out a set of appropriate assignments for the execution tasks and the data transfers of the application to the components. However mapping and scheduling problems for multi-processors are NP-Complete [2]. General methods cannot deal with this kind of problems efficiently, especially for the large-scale problems.

Many heuristic methods to deal this kind of problems have been proposed, for example, the tabu search [3],[4], genetic algorithms [5], the Kernighan-Lin method [6], and ant colony optimization (ACO) [7],[8],[9]. Nevertheless, a majority of these approaches above usually concentrate on one aspect of the problem and when the design space is heavily constrained the optimization solutions are always not satisfactory. When they simultaneously deal with the mapping and scheduling problems, it is very suitable when the size of the problem is small. However, when they faced with large-scale problems due to the complex of the calculation and the increase of solutions these methods will lose its effectiveness. General approaches that able to efficiently generate high-quality solutions for complex application on the heterogeneous embedded architectures are definitely required.

In this paper, we present an algorithm, based on set-based discrete particle swarm optimization (S-PSO) [10], which has been proved to have good performance in TSP and 0-1 knapsack problem. Additionally, this algorithm introduces the heuristic information to accelerate the speed of convergence. For the problem to be solved, we can choose effective heuristic information to improve the performance of the algorithm. For large scale problems due to the effective heuristic information the improvement of the performance will be more noticeable than general methods. Because the scheduling and mapping problems on the target platform is a combinatorial optimization problem so the S-PSO method is also suitable for dealing with this problem. In this paper, we use the S-PSO solves scheduling and mapping problems on the target platform, with the aim of minimizing overall application execution time. The major contributions of this paper can be summarized as follows.

- The S-PSO method has been invented for solving the combinatorial optimization problem and has more simple computation procedure than Ant Colony Optimization. The S-PSO algorithm can reduce the execution time of the application by exploring various solutions for mapping and scheduling of tasks and communications.
- We compare our algorithm with the ACO [9] which can also simultaneously address the mapping and scheduling problems on a complex and heterogeneous MPSoC, and our algorithm has better performance than ACO in dealing with large scale problems.

* Corresponding Author.

This work was supported in part by the NSFC projects Nos. 61379061, 61332002, 61511130078, 61202130, in part by Natural Science Foundation of Gungdong for Distinguished Young Scholars No. 2015A030306024, in part by the "Guangdong Special Support Program" No. 2014TQ01X550, and in part by the Guangzhou Pearl River New Star of Science and Technology No. 201506010002.

The rest of this paper is organized as follows. In section II, we define and formalize the problem that we address in this paper. Section III introduces some background work, presenting the S-PSO heuristic. Section IV details the formulation proposed in this paper, then we evaluate and compare with previous heuristics dealing with the same problem in Section V. Finally, section VI includes the discussion and conclusion.

II. PRELIMINARIES

In this section, we provide a brief description of what we address in this paper. First, we abstract the model of multitask applications and the target architecture. Then we present the abstraction and definition of the mapping and scheduling problem on the target architecture.

A. *Target Architecture*

In this work, we consider a typical architectural model H for a heterogeneous multiprocessor system-on-chip (MPSoC), formal description is as follows:

$$H = PE \cup CC \tag{1}$$

where PE represents a set of processing elements and CC represents a set of communication components. A simple example of such an MPSoC is shown in Figure 1 and it is an abstract description of the target architecture. It is composed of three processing elements and a single system bus as the communication components.

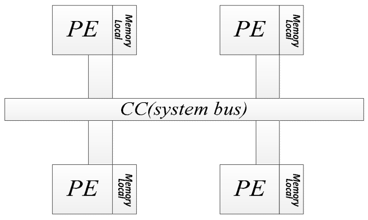

Fig. 1. The abstract model of the target architecture.

There are more details about the target architecture we should take care.

- As a formal description of the multimode resource constrained scheduling problem [11], there are a set of associated resources on the component of the target architecture (e.g., the local memory). These resources can be divided into two parts: 1) renewable resources RR, which can be re-used when the task assigned in the resource has been completed; and 2) nonrenewable resources NR, which the quantity of the resource will be consumed by the task assigned in the resource.
- No more than one task can be executed on a processor at the same time.
- When a data transfer happens, if the data dependent tasks executed on the same processor the transfer time will be zero, otherwise, the transfer time only depends on the quantity of data and the bandwidth of the system bus. We assume that the communication model is the same for all the data transfers.

B. *Application Model*

A multitask application is composed of a set of N tasks and each task to be executed on a single unit. We abstract the application as a directed acyclic graph (DAG). A DAG is formally defined as a graph $G = (V, T)$ without feedback edges, where V represents a set of operations (tasks), T represents the data transfer (communications) and the data dependences among the tasks.

There is a constraint on the application executed order. If the graph G exists an edge $tt(v', v) \in T$, this indicates that the task v can be executed only after the task v' and all the data transfers point to v have been completed. Each edge $tt(v', v) \in T$ is also annotated with a weight which means the amount of data exchanged from the parent task v' to the child task v. A simple example of an application graph is shown as follow.

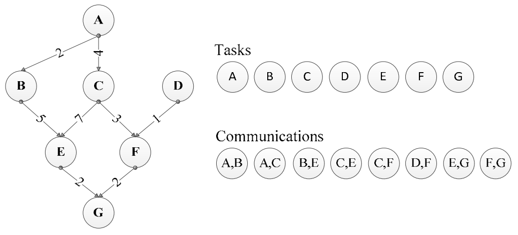

Fig. 2. An application graph. Edges are annotated with the amount of data to be transferred between source and target tasks.

C. *Problem Abstraction and Definition*

The problem is defined as follows:

$P = (H, G)$ composed of the target architecture H and multitask application G which we defined above. Under the given condition P, we try to find out the minimum over all execution time of the application G on the target platform H.

A job j: A job j is defined as an event to be executed on a component of the architecture. Thus, all the tasks and the communications in the application G can be abstracted as a set of jobs J.

The implementation points I: We abstract all the components of the target platform H as a set of implementation points I. An implementation point i is defined as a structure of resources and time, formally like (r, t), we describe the job j executed not in the real hardware but the implementation point i we have been abstracted.

There are some facts as follows. First, some of these jobs can execute on various processing unions with different performances. Second, due to the different consumption of resources, the task will take different time in the same component.

According to the facts above, we can abstract two functions as follows.

- The function δ: $J \rightarrow I$, it's the mapping from J to I, δ is one-to-many mapping function, in order to find out the minimum over all execution time of the application,

TABLE I. THE MAPPIGN RELATIONSHIP OF Δ AND Γ FOR THE EXAMPLE TASK GRAPH

job	pe₀		pe₁		pe₂		pe₃				Cc	
	i_0		i_1		i_2		i_3		i_4		i_5	
	t	*rr*	*t*	*rr*	*t*	*rr*	*t*	*nr*	*t*	*nr*	*t*	*Rr*
A	3	6	7	4	3	8	6	3	2	9	null	Null
B	5	6	2	2	7	9	2	3	1	12	null	Null
C	6	8	2	9	2	10	10	9	12	2	null	Null
D	9	12	4	8	19	5	7	13	7	9	null	Null
E	3	6	7	4	3	8	6	3	2	9	null	Null
F	8	2	7	12	1	12	2	9	11	7	null	Null
G	12	3	12	7	1	9	11	12	9	10	null	Null
(A,B)	null	null	null	null	null	null	null	null	null	null	2	8
(A,C)	null	null	null	null	null	null	null	null	null	null	4	8
(B,E)	null	null	null	null	null	null	null	null	null	null	5	8
(C,E)	null	null	null	null	null	null	null	null	null	null	7	8
(C,F)	null	null	null	null	null	null	null	null	null	null	3	8
(D,F)	null	null	null	null	null	null	null	null	null	null	1	8
(E,G)	null	null	null	null	null	null	null	null	null	null	2	8
(F,G)	null	null	null	null	null	null	null	null	null	null	2	8

the proposed algorithm should select an appropriate implementation point *i* for each job *j*.

- The function γ: *I*→*H*, it's the mapping from *I* to *H*. Where *H* represents the components in the target platform. γ is many-for-one mapping function, from this function we can get the components *h* ($h \in H$) connected with the implement point *i*.

Under the given condition *G*, in order to simplify the optimization algorithm we assume the functions of *δ* and *γ* are already known. A mapping relationship of *δ* and *γ* are show in Table I. For a given job and implementation point we can get a pair (*r, t*), if the number of resources over the number of the implementation point contains or the job and the implement point are not the same type, we think the job can't be executed in this implement point.

For each job in the application includes the following properties: start time, end time and execution time, we use *StartTime*(*j*), *EndTime*(*j*), and *ExecutionTime*(*j*) to represent them. We can easily find out the fact:

$$EndTime(\mathrm{j}) = StartTime(j) + ExecutionTime(j), \qquad (2)$$

when we chose an implementation point to execute the job *j*, we can get a correct number of execution time from the table like TABLE I.

The job's start time is decided by its direct predecessors and component of the implementation point. The job can be executed only when its direct predecessors have been completed and the component is free. The formal expression as follows:

$$max[parent(j), free(\gamma(\delta(j)))] \leq StartTime(j), \qquad (3)$$

parent(*j*) returns the maximum end time of all the direct predecessors of job *j*, a_j is a component, we can get it from the implementation point of job *j*, *free*(*γ*(*δ*(*j*)) return the end time when the component *γ*(*δ*(*j*)) is available. We assume that all the components' end time is initialized to zero.

In this paper, our aim is to optimize the overall execution time of the multitask application; the make-span can be defined as:

$$CostFuncton = max(EndTime(j)), \quad \forall j \in J \qquad (4)$$

The *CostFunction* depends on the maximum of all the jobs' end time, in order to get the minimum *CostFunction*, we need to efficiently map and schedule all the tasks and the communications of the application.

III. RELATED WORK

Many previous works have been addressed on mapping and scheduling tasks and communications on heterogeneous embedded systems. The methods can be classified as online and off-line algorithms.

Niemann and Marwedel [12] proposed an integer linear programming (ILP) to find the optimal solution for the mapping and scheduling problem, however, the ILP only consider the multiple implementations on executed union (e.g. the processor) but the different kind of communication models are not supported. In this paper, our algorithm can solve more complex and general problems. The resources (i.e., the local memory) are divided into renewable and nonrenewable and we also consider multiple implementations for each task on different components. Moreover, several methods often separate solving the mapping and scheduling problems, like simulated annealing (SA), Tabu search (TS) [13] and genetic algorithms(GAs) [14], in our method we abstract the calculating operations and data transfer operations as tasks, so we can optimal the mapping and scheduling problem as whole and it will get more optimization results. Some works also consider the communications independently from the components [15], our method make the communication as a general components, this make mapping and scheduling more efficient. However, when expand the scale of the tasks, the execution time of the algorithm is not acceptable and the optimized result is not satisfactory, like ant colony optimization (ACO) [9]. Our approach is more effective in dealing with large scale problems.

In conclusion, we define a method that is able to efficiently explore and exploit all the dimensions of the problems to obtain efficient implementations for the applications on a large class of target platforms with complex constraints.

A. S-PSO

Piratical Swarm Optimization is a heuristic search methodology proposed by Kennedy and Eberhart in 1995 [16]. Later, various improved PSO variants have been developed, making PSO one of the most popular optimization techniques in recent years. PSO is simple and efficient, but its original form is mainly used to solve continuous space problems. Since mapping and scheduling tasks is a discrete problem, traditional PSO is not suitable to solve this kind of problem. However, there are many discrete PSO methods have been proposed, such as binary Discrete PSO by Kennedy and Eberhart [18], Discrete multi-phase PSO by B.Al-Kazemi and C. K. Mohan [19], angle modulation PSO to solve binary problems by G. Pampara and N. Franken [20] and discrete PSO with genetic manipulation for the workflow scheduling problem by Pan [21], they are the advancement of using PSO to solve the problems in a discrete space. In this paper we use a set-based PSO (S-PSO) proposed by Chen *et al.* [10] to solve the optimization problem.

In the standard PSO, PSO is initialized with a population of particles with random positions and velocities in the search space of the problem. It can be described as follows. Consider a population of particles. The position of a particle is denoted by $x_k = (x_{kd} : d = 1,\dots, D)^T$ which is a D-dimensional vector in the search space of the problem. The index k ($k = 1,\dots,S$) labels the k_{th} particle in the swarm. The velocity of a particle is denoted by $v_k = (v_{kd} : d = 1,\dots,D)^T$, PSO explores the search space by modifying the velocity of each particle.

Each particle's location is a potential solution of the problem. In order to optimize the solution, the particles are exploring and exploiting the search space obeying the rules to update their positions and velocities as follows:

$$v_k^j = \omega * v_k^j + c_1 * r_1^j * \left(pbest_k^j - x_k^j\right) + c_2 * r_2^j * \left(gbest^j - x_k^j\right) \tag{5}$$

$$x_k^j = x_k^j + v_k^j \tag{6}$$

where $pbest_k^j$ is the best position of particle k in its history and $gbest^j$ is the best position state for all particles. c_1 and c_2 are positive real numbers to control the movement towards the individual and global best position. ω is the inertia weight to balance the ability of global search and local search. In addition, $r_1^j \in [0,1]$ and $r_2^j \in [0,1]$ are random numbers, used ensure the diversity of the population.

In S-PSO [10], the particles search the discrete space obeying the same rules to update their positions and velocities as the standard PSO, however when we find the solution we must obey the rules under the discrete space. So we redefined the operators. All the arithmetic operators in rule (5) and (6) are redefined as follows:

- *position X*: A position X is a solution to the problem. We define the k_{th} particle as X_k ($X_k \subseteq E$), in the j_{th} dimension of X_k we denote as $X_k^j \subseteq E^j \ (j = 1,2,\dots,D)$, where E can be divided into D dimensions, $E = E^1 \cup E^2 \cup \dots \cup E^D$.

- *velocity V*: In S-PSO the velocity is defined as a set with possibilities. A set with possibilities V is given by

$$V = \{e / p(e) \mid e \in E\}, \tag{7}$$

  $p(e)$ is the possibilities, based on the definition, in the j_{th} dimension, $V_k^j = \{e / p(e) \mid e \in E^j\}$.

- *constant* $\times$ *velocity*: the term constant is a parameter, in the S-PSO, the multiplication operator between a constant c($c \geq 0$) and a set with possibilities $V = \{e/p(e) \mid e \in E\}$ is defined as follows:

$$cV = \{e / p'(e) \mid e \in E\},$$
$$p'(e) = \begin{cases} 1, & if\ c \times p(e) > 0 \\ c \times p(e), & otherwise \end{cases} \tag{8}$$

- *position – position*: The minus operator between two sets A and B is given by

$$A - B = \{e \mid e \in A\ and\ e \notin B\} \tag{9}$$

- *Constant* $\times$ (*position* - *position*): The multiplication operator between a constant c and a set $E' \in E$ is defined as follows:

$$cE' = \{e / p'(e) \mid e \in E'\}, \tag{10}$$

$$p'(e) \begin{cases} 1, & if\ e \in E' and\ c > 1 \\ c, & if\ e \in E' and\ 0 \leq c \leq 1 \\ 0, & if\ e \notin E' \end{cases}$$

- *velocity + velocity*: given two set with possibilities $V_1 = \{e / p_1(e) \mid e \in E\}$ and $V_2 = \{e / p_2(e) \mid e \in E\}$, the plus operator between two sets with possibilities is defined as follows:

```
Position updating (X_i, V_i)
Step 1) Generate a cut set
  Generate a random number $\alpha \in (0,1)$ .
  For each dimension j
      $cut_\alpha(V_k^j) = \{e \mid e / p(e) \in V_k^j \text{ and } p(e) \geq \alpha\}$
  End for
Step 2) update the position
  $NEW\_X_k = \phi$
  For each dimension j
      $CandidateSet_k^j = \{e \mid e \in cut_\alpha(V_k^j) \text{ and } e \text{ satisfies } \Omega\}$
      While the construction of $NEW\_X_k^j$ is not finished and $CandidateSet_k^j \neq \phi$
          Select an element from $CandidateSet_k^j$ and add it to $NEW\_X_k^j$ ;
          Update $CandidateSet_k^j$ ;
      End while
      If the construction of $NEW\_X_k^j$ is not finished
          $CandidateSet_k^j = \{e \mid e \in X_k^j, \text{ and } e \text{ satisfies } \Omega\}$ ;
          While the construction of $NEW\_X_k^j$ is not finished and $CandidateSet_k^j \neq \phi$
              Select an element from $CandidateSet_k^j$ and add it to $NEW\_X_k^j$ ;
              Update $CandidateSet_k^j$ ;
          End while
      End if
      If the construction of $NEW\_X_k^j$ is not finished
          $CandidateSet_k^j = \{e \mid e \in E^j, \text{ and } e \text{ satisfies } \Omega\}$ ;
          Select an element from $CandidateSet_k^j$ and add it to $NEW\_X_k^j$
      End if
  End for
  $X_i = NEW\_X_i$
End procedure
```

Fig. 3. Pseudo code for the position updating procedure.

$$V_1+V_2 = \{e / max(p_1(e), p_2(e)) \mid e \in E\} \qquad (11)$$

According to the above definitions, we can update the velocity of the discrete PSO obey the rule (5).

After updating the *velocity*, particle *i* follows the rule (6) to update its current position X_i. However, the problems in discrete space are different from the continuous space, there are various constrains when we optimize the problem. In this case the positions' updating must satisfy the constraints Ω we considered above. So we defined a new set named *CandidateSet*, we put all the elements satisfied constrains in it. Every step we update the particles' position, we can only chose the element in the candidate set. To ensure the feasibility of the newly generated position NEW_X_i in S-PSO, we redefined the plus operator between a set with possibility and a crisp set. The procedure of the plus operator is given in Fig. 3.

IV. Proposed Approach

The general idea of our approach is to use the set-based PSO to find out a feasible workflow scheduling list. There are three key steps when we using the D-PSO to solve the mapping and scheduling problem. The first is how to encode the problem, that is, how to represented the solution. The second is the definition of the fitness function, according to the fitness function we can compare the optimal results can determine how to optimal the result in the next iteration. The last one is how to present the heuristic information, according the fitness function we should find out the heuristic information, the heuristic information are directly associate with the optimal results and the optimal time.

For the mapping and scheduling problem, we regard the particle's position *X* to represent the feasible solutions. The number of particle's dimension equal to the number of jobs in the application *G*, each dimension is composed by a key-value pair like *<j, i>*, it means job *j* executed in the implementation point *i*. In every generation, each practical will construct a new feasible solution. As an example, the particle depicted in Fig.4 represents a feasible solution under the condition *P*.

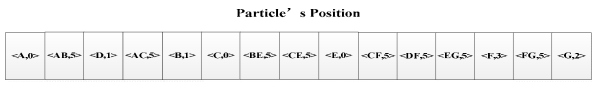

Fig. 4. Example of the particle's position. Each dimension is composed by a job and an implementation point.

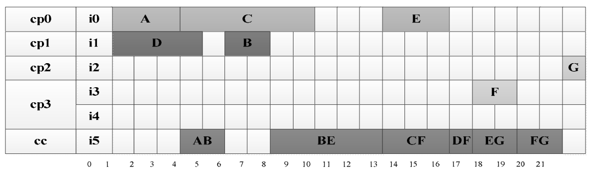

Fig. 5. The serial generation scheme schedule method to get the execution times.

For the problem we proposed, we need define a fitness function which is used to evaluate the potential solutions, in this case we try to find out the minima cost time for the application, so we define the fitness function to be the execution time of the problem. When we get a position *X*, we use the serial schedule generation scheme (SSGS) method to get the execution time. The SSGS construct the complete solution in *n* stages, where at each stage one feasible element is selected from the scheduling list. A simple schedule plan is shown Fig. 5.

According to the rule we defined above. When we update the position, we should define the heuristic information to help us make a good decision. In this question we define the job's execution time in the implementation point *i* as the heuristic information, we use the roulette wheel to chose an element from the candidate set, the less time needed the more probability it has.

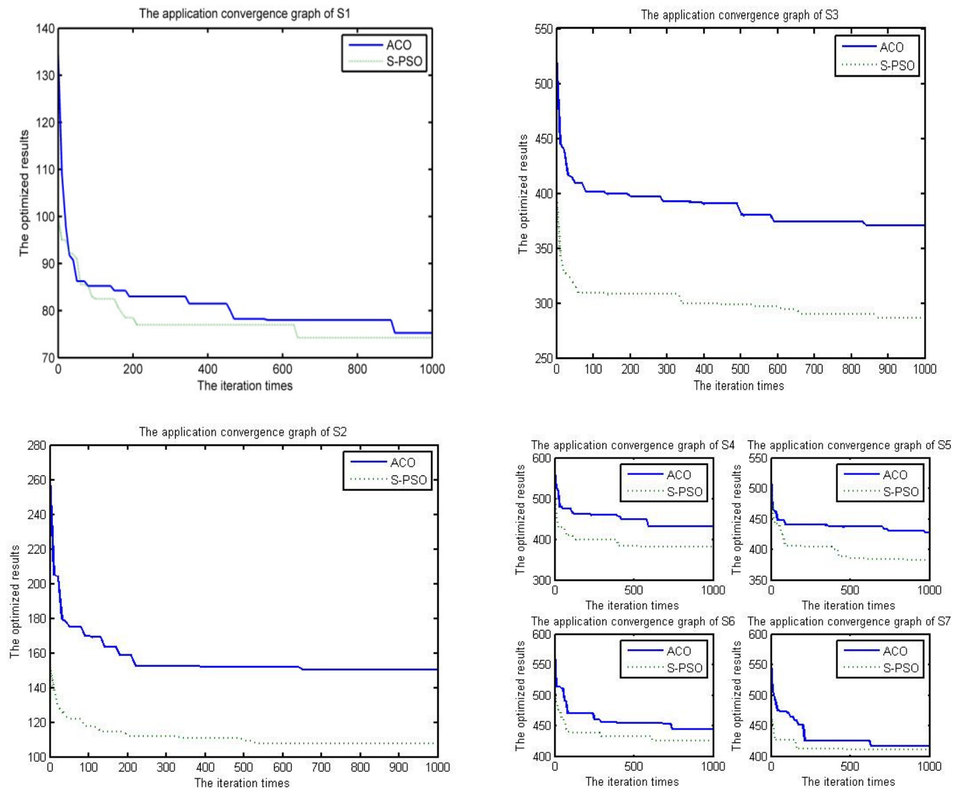

Fig. 6. The speed of convergence in small problem.

We formula the heuristic information as follows:

$$spt(<j, i>) = 1 / t_{ji}\ , <j, i> \in CandicateSet^{j} \qquad (12)$$

The structure of the S-PSO is similar to the standard PSO, but when we update the velocity and the position, we use the function we defined above. The update of the *pbest*$_k$ and the *gbest* are similar to the standard PSO, when each particle's position has been update, we use the new position to get the *pbest*$_k$ and the *gbest*, and the pseudo-code of the proposed methodology is shown Fig. 7.

```
Procedure S-PSO
  Initialization;
  While terminal condition not meet
      For each particle i (i=1,2,…,M)
          Velocity updating;                    (5)
          Position updating;                    (6)
      End for
      For each particle k (k=1,2,…,M)
          Update the pbest_k and gbest.
  End while
End procedure
```

Fig. 7. The pseudo-code for the proposed methodology.

V. Experimental Evaluation

In this section we implemented the proposed approach to deal with the mapping and scheduling problem and evaluated our methodology by optimizing several synthetic test cases on the abstract target architecture *H* and then we compare our approaches with other heuristics.

A. Experimental setup

First, we should initialize the number of the associated resources in the target architecture *H*. The number of resources is shown in Table II, when we deal with the problem, the mapping and scheduling solution should remain the resource constrains. Second, we use the task graph in [22] as the test DGA, we randomly generated the number of the time and resources consumed for each job *j* in the implementation point *i*.

To analyze the methodology in this paper, we compare our approaches with the ant colony optimization (ACO). We set the experimental condition as follows:

- ACO: The ACO [9] is a heuristic approach that has been applied to these problems with good result, in this experiment, we set $\alpha = \beta = 1$, where α is the weight for local heuristics and β is the weight for global heuristics. The evaporation rate has been set to ρ = 0.015

- S-PSO: this is the methodology we proposed in this paper and described above. We set the acceleration factor $c_1 = c_2 = 2$and the inertia weight $\omega = 0.3$.

In this experiment we defined the number of particles and ants are 50. We compare the optimized results under the same iteration numbers, in this experiment we set all the iteration number are 1000 times.

TABLE II. THE NUMBER OF RESUOURCES IN THE PLATFORM

Component	pe_0	pe_1	pe_2	pe_3	cc
Recourse number	30	30	30	60	8

pe^i (i = 0, 1, 2, 3) is the number of local memory and cc is the number of bandwidth.

B. Result

In the first experiment, we randomly generated several small benchmarks and real-life benchmarks to demonstrate our method's effectiveness, namely *S*1-*S*7 in Table III. In this experiment there are some applications have the same tasks and edges but they are not the same, because the applications have the different type of DGA. According to the experiment result, we compared the methods on the number of the optimum value and the speed of convergence. Because the value of the optimal results are directly related with the mount of the jobs and the execution time of each job, in order to compare the algorithm performance more objective, we also compute the ratio of the performance improved, we compute the performance as (*Avg.*(*ACO*)–*Avg.*(*PSO*)) / *Avg.*(*ACO*), Avg. is the average time of the experiments, Std. is the standard deviation of the experiments, the Diff. presents the percentage of optimization. In order to observe the optimal results directly, we also display the optimized procedure in the line chart, for the experiment we did in the small test set we can get the plot Fig. 6, from Fig. 6 we can compared the search methods on the number of evaluations.

TABLE III. THE SMALL-SCALE TEST SET

App.	#Tasks/ #Edges	ACO		PSO		Diff.
		Avg.	*Std.*	*Avg.*	*Std.*	
S1	7/8	75.25	2.17	76.25	3.90	1.33%
S2	15/20	150.50	8.90	107.75	3.11	28.41%
S3	31/40	371.00	17.56	286.50	6.58	22.78%
S4	48/32	432.50	15.20	382.00	11.41	11.68%
S5	48/32	427.00	8.87	383.00	14.34	10.30%
S6	48/32	443.50	17.54	426.00	6.38	3.95%
S7	48/32	416.00	22.41	410.50	11.68	1.32%

As the scale of the problems becomes larger, many algorithms fail to optimize the problem. In the second experiment, we compared our algorithm on the larger benchmarks described in Table IV, namely *S*8-*S*19. For the same size jobs, we also compared the optimized results for different type of DGA. For the large scale test set we composed the optimization results in Table IV and the speed of convergence in Fig. 8.

TABLE IV. THE LARGE-SCALE TEST SET

App.	#Tasks/ #Edges	ACO		PSO		Diff.
		Avg	*Std.*	*Avg*	*Std.*	
S8	93/62	1001.50	18.89	818.50	12.29	18.27%
S9	93/62	994.00	16.36	910.50	12.79	8.40%
S10	93/62	993.50	13.40	803.50	30.19	19.12%
S11	93/62	1016.00	21.52	947.50	11.30	6.74%
S12	138/92	1390.50	45.61	1323.50	12.21	4.82%
S13	138/92	1466.00	35.19	1254.00	58.00	14.46%
S14	138/92	1435.50	23.98	1257.50	16.15	12.40%
S15	138/92	1435.50	31.88	1321.00	6.00	7.98%
S16	183/122	1934.00	21.99	1894.00	17.00	2.07%
S17	183/122	1990.50	19.55	1674.00	11.00	15.90%
S18	183/122	1957.00	26.39	1752.00	54.00	10.48%
S19	183/122	1941.50	40.11	1827.00	48.00	5.90%

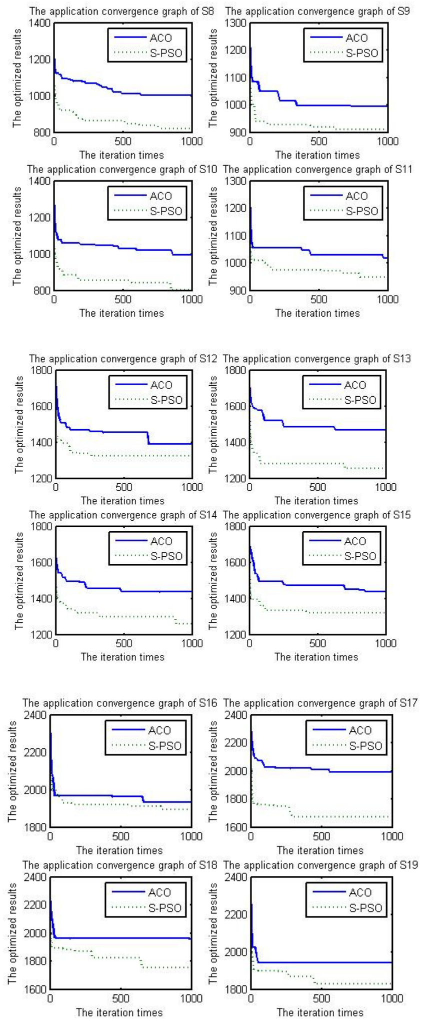

Fig. 8. The speed of convergence in large scale of problems .

VI. Conclusion

In this paper, we proposed a set-based PSO algorithm for optimizing the mapping and scheduling problem on the heterogeneous multiprocessor architectures. We use the execution time for each job in the candidate set as the heuristic information, so we can construct a new solution effectively under the constraints of the target. In the section V, we compare our method with the ACO which has been tested can solve the problem efficiently. We compare the two methods on the set of instances in the benchmark library PSPIB. The result in Table III and Table IV show that our method is more effective than ACO. For the same number of evaluations, we get the results 10.86% better on average. For the same problem our method can get the better optimum value. Moreover, from Fig.6 and Fig.8 show that our approach was able to reach the optimal solutions much faster than ACO. The results proved that our algorithm has higher efficiency to optimize both in the small and lager scale of problems.

References

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in Proc. 41st Assoc. Comput. Machinery/IEEE Design Automat. Conf. (DAC), 2004, pp. 681–685.

[2] J. Kim, S. Lee, H. Shin, Y. Lee, and Hwangsik Bae, "Effective task mapping and scheduling techniques for heterogeneous multi-core systems based on zone refinement," Computer Sciences and Convergence Information Technology (ICCIT), 6th International Conference. 2011, pp. 363 – 366.

[3] T. Wiangtong, P.Y.K. Cheung, and W.Luk, "Comparing three heuristic searchmethods for functional partitioning in hardware–software code-sign." Design Automation for Embedded Systems, 6(4):425–449, 2002.

[4] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search." Design Automation for Embedded Systems, 2:5–32, 1997.

[5] M. Grajcar, "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system." In DAC'99: the 36th ACM/IEEE conference on Design Automation, pages 280–285, 1999

[6] F. Vahid and T. D. Le, "Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning." Design Automation for Embedded Systems,2(2):237–261, March 1997.

[7] G. Wang, W. Gong, B. DeRenzi, and R. Kastner, "Ant colony optimizations for resource and timing constrained operation scheduling." IEEE Transactions on Computer-Aided Design of Integrated Circuitsand Systems, 26(6):1010–1029, June 2007.

[8] P.-C. Chang, I.-W. Wu, J.-J. Shann, and C.-P. Chung, "ETAHM: An energy aware task allocation algorithm for heterogeneous multiprocessor." In Proceedings of DAC '08, pages 776 –779, june 2008.

[9] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Antcolony heuristic for mapping and scheduling tasks and communicationson heterogeneous embedded systems." IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 29(6):911–924, June 2010.

[10] W.-N. Chen, J. Zhang, Chung, H.S.H., W.-L. Zhong, W.-G. Wu, and Y. Shi, "A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems," IEEE Trans. Evol. Comput. Vol. 14, no. 2, pp. 278-300, April 2010.

[11] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch,"Resource constrained project scheduling: Notation, classification, models, and methods," Eur. J. Operat. Res., vol.112, no.1, pp. 3–41, Jan.1999

[12] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming, " Design Automat. Embedded Syst., vol. 2, no. 2, pp. 125–163, Mar.1997.

[13] S. J. Beaty, "Genetic algorithms versus tabu search for instruction scheduling," in Proc. Int. Conf. Neural Netw. Genetic Algorithms, Feb. 1993, pp. 496–501.

[14] M. Grajcar, "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system," in Proc. 36th Assoc. Comput. Machinery/IEEE Conf. Design Automat. (DAC), 1999, pp. 280–285.

[15] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in Proc.2nd IEEE/Assoc. Comput. Machinery/IFIP Int. Conf. Hardware/Software Codesign Syst. Synthesis (CODES+ISSS), 2004, pp. 248–253

[16] J. Kennedy and R. C. Eberhart,"Particle swarm optimization," in Proc. IEEE Int. Conf. Neural Netw., 1995, pp. 1942–1948.

[17] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in Proc. IEEE Int. Conf. Syst. Man Cybern., 1997, pp. 4104–4109.

[18] W.-N. Chen, *et al*. "Particle Swarm Optimization with an Aging Leader and Challengers", IEEE Transactions on Evolutionary Computation, vol. 17, no. 2, pp. 241-258, 2013.

[19] B. Al-Kazemi and C. K. Mohan, "Discrete multi-phase particle swarm optimization," in Information Processing with Evolutionary Algorithms. Berlin, Germany: Springer, 2006, pp. 306–326

[20] G. Pampara, N. Franken, and A.P.Engelbrecht, "Combining particle swarm optimization with angle modulation to solve binary problems,"in Proc. 2005 IEEE Congr. Evol. Comput., vol. 1. pp. 89–96.

[21] Q. K. Pan, M. F. Tasgetiren, and Y. C. Liang, "A Discrete Particle Swarm Optimization Algorithm for the Permutation Flow shop Sequencing Problem with Makespan Criteria," In the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Cambridge, UK, 2006, pp. 19-31.

[22] R. Kolisch, C. Schwindt, and A. Sprecher, "Benchmark instance for project scheduling problem," in Handbook on Recent Advance in Project scheduling, J. Weglarz, Ed. Amsterdam, The Netherlands: Kluwer, 1999, pp.147-178.