Yan XH, He FZ, Chen YL. A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(2): 340–355 Mar. 2017. DOI 10.1007/s11390-017-1714-2

A Novel Hardware/Software Partitioning Method Based on Position Disturbed Particle Swarm Optimization with Invasive Weed Optimization

Xiao-Hu Yan, *Member, CCF*, Fa-Zhi He *, *Member, CCF*, and Yi-Lin Chen

State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
School of Computer Science, Wuhan University, Wuhan 430072, China

E-mail: yanxiaohupaul@126.com; fzhe@whu.edu.cn; 285813238@qq.com

Received July 27, 2016; revised December 24, 2016.

Abstract With the development of the design complexity in embedded systems, hardware/software (HW/SW) partitioning becomes a challenging optimization problem in HW/SW co-design. A novel HW/SW partitioning method based on position disturbed particle swarm optimization with invasive weed optimization (PDPSO-IWO) is presented in this paper. It is found by biologists that the ground squirrels produce alarm calls which warn their peers to move away when there is potential predatory threat. Here, we present PDPSO algorithm, in each iteration of which the squirrel behavior of escaping from the global worst particle can be simulated to increase population diversity and avoid local optimum. We also present new initialization and reproduction strategies to improve IWO algorithm for searching a better position, with which the global best position can be updated. Then the search accuracy and the solution quality can be enhanced. PDPSO and improved IWO are synthesized into one single PDPSO-IWO algorithm, which can keep both searching diversification and searching intensification. Furthermore, a hybrid NodeRank (HNodeRank) algorithm is proposed to initialize the population of PDPSO-IWO, and the solution quality can be enhanced further. Since the HW/SW communication cost computing is the most time-consuming process for HW/SW partitioning algorithm, we adopt the GPU parallel technique to accelerate the computing. In this way, the runtime of PDPSO-IWO for large-scale HW/SW partitioning problem can be reduced efficiently. Finally, multiple experiments on benchmarks from state-of-the-art publications and large-scale HW/SW partitioning demonstrate that the proposed algorithm can achieve higher performance than other algorithms.

Keywords hardware/software partitioning, particle swarm optimization, invasive weed optimization, communication cost, parallel computing

1 Introduction

Hardware/software (HW/SW) partitioning which involves multidisciplinary and collaborative design is a key step in the design of modern embedded products[1]. Implementation with software module has more flexibility and needs less cost, but costs more executing time, and vice versa in hardware case[2]. The target of HW/SW partitioning is to balance all the tasks to optimize some objectives of the system under some constraints[3]. Traditionally, HW/SW partitioning is carried out manually. However, with the development of the design complexity in embedded systems, HW/SW partitioning becomes a challenging problem.

Many approaches about HW/SW partitioning have been proposed. Based on the partitioning algorithm, exact and heuristic solutions can be differentiated[4]. The proposed exact algorithms include dynamic programming[5-6], Branch-and-Bound (B&B)[7-9] and Integer Linear Programming (ILP)[10]. Exact algorithms cannot provide a feasible solution for the large-scale HW/SW partitioning problem because most formulations of the HW/SW partitioning problem are NP-hard. Thus, many researchers have applied heuris-

Regular Paper

The work was supported by the National Natural Science Foundation of China under Grant No. 61472289 and the National Key Research and Development Project of China under Grant No. 2016YFC0106305.

*Corresponding Author

tic algorithms to HW/SW partitioning. The traditional heuristic algorithms include hardware-oriented and software-oriented ones. The hardware-oriented approach starts with a complete hardware solution and swaps parts to software until constraints are violated[11-12], while the software-oriented approach means that the initial implementation of the system is supposed to be a software solution. General-purpose heuristics include genetic algorithm (GA)[13-16], simulated annealing (SA)[17-18], greedy algorithm[19], tabu search (TS)[20-21], ant colony optimization (ACO)[22-23] and particle swarm optimization (PSO)[24-25]. The other group of partitioning-related heuristics is the Kernighan-Lin heuristic[26], which is substantially improved by many others[27-29]. All these approaches can work perfectly within their own co-design environments, but due to the enormous differences among them, it is not possible to compare the results obtained[30].

In this paper, a novel HW/SW partitioning method based on position disturbed particle swarm optimization with invasive weed optimization (PDPSO-IWO) is presented. To fight against premature convergence and avoid local optimum, the particles in PDPSO-IWO move away from the worst particle in the population, near which there is a potential predatory threat. To improve the search accuracy and the solution quality, the improved IWO is integrated to search a better position, with which the global best position is updated. To enhance the solution quality, a hybrid NodeRank (HNodeRank) algorithm is proposed to initialize the population. To accelerate HW/SW partitioning method based on PDPSO-IWO, the HW/SW communication cost runs in GPU parallel computing environment.

The rest of the article is organized as follows. Section 2 describes related work. Position disturbed particle swarm optimization with invasive weed optimization is presented in Section 3. A novel HW/SW partitioning method based on PDPSO-IWO is presented in Section 4. Section 5 analyzes the performance of PDPSO-IWO experimentally. Finally, conclusions are drawn in Section 6.

2 Related Work

2.1 Problem Definition

In this paper, the partitioning problem definition is based on the following notations. Given an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set, the HW/SW partitioning problem can be formulated to the minimization problem P as follows:

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{n} h_i(1 - x_i) \\ &\text{subject to} \sum_{i=1}^{n} s_i x_i + C(\boldsymbol{x}) \leqslant R, \\ &\qquad x_i \in \{0, 1\}, i = 1, 2, ..., n, \end{aligned} \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ indicates a solution of HW/SW partitioning problem. $x_i = 1(0)$ denotes that the node is partitioned to software(hardware). s_i and h_i indicate the software and the hardware cost of node v_i, respectively. $C(\boldsymbol{x})$ indicates the communication cost of $\boldsymbol{x}$, and R is the constraint. In [19], $C(\boldsymbol{x})$ is replaced with uR, where $0 \leqslant u \leqslant 1$, and then the problem P can be converted to Q':

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n} h_i x_i \\ &\text{subject to} \sum_{i=1}^{n} s_i x_i \leqslant (1 - u)R, \\ &\qquad x_i \in \{0, 1\}, i = 1, 2, ..., n. \end{aligned}$$

The HW/SW partitioning problem Q' is reduced to a variation of knapsack problem in [19]. To obtain the global best solution, the PDPSO-IWO is used to solve the problem P in (1).

2.2 Heuristic Algorithms

As a key challenge in HW/SW co-design, HW/SW partitioning has been studied for many years. In early studies, the HW/SW partitioning models and algorithms have been tested on systems with some dozens of components. The target architecture is supposed to consist of a single software and a single hardware unit[31-32]. The HW/SW partitioning problem has specific optimization objectives, such as minimizing power, hardware area and communication overhead[33-35]. But in recent research, the HW/SW partitioning model and algorithm have been tested on systems with hundreds or even thousands of components. The HW/SW partitioning problem is formalized as a task graph, or a set of task graphs. In this work, the HW/SW partitioning problem is based on the same assumptions and system model which are used in [4, 19]. This paper does not aim at partitioning for a given architecture, nor does it propose a complete co-design environment. The HW/SW partitioning definition is general enough so that the proposed algorithm can be used in different practical cases[4].

Many exact algorithms for HW/SW partitioning were proposed in 1990s, such as B&B and ILP. Exact algorithms have mathematic solution and can find high-quality solutions rapidly for the small-scale HW/SW

partitioning problem. Since most formulations of the HW/SW partitioning problem are NP-hard, exact algorithms fall victim to combinatorial explosion and tend to be quite slow when the scale of HW/SW partitioning problem becomes large. In recent years, researchers have applied many heuristic algorithms to HW/SW partitioning, such as GA, ACO and TS. The heuristic algorithm can find approximate global optimum in short time.

In [19], the partitioning problem is transformed into a one-dimensional search problem and three algorithms are proposed. Since Alg-new3 works the best among the three algorithms, Alg-new3 which is named as Base is compared in this paper. A heuristic algorithm which is named Heur was proposed in [21]. Experimental results show that Heur outperforms the algorithm in [19] by up to 28%. In [36], NodeRank that calculates the rank of each node iteratively was proposed to solve the HW/SW partitioning problem. Experimental results show that NodeRank outperforms the algorithm in [21].

PSO is attractive for the HW/SW partitioning problem as it offers reasonable coverage of the design space together with short execution time[25]. In [24], it was found that PSO outperforms GA in the cost function and the execution time for solving the HW/SW partitioning problem. In [37], the authors proposed a modified PSO restarting technique to avoid quick convergence, named the re-excited PSO algorithm, to solve the HW/SW partitioning problem. In [38], it is revealed that the performance of PSO outperforms that of ILP, GA and ACO for solving the HW/SW partitioning problem. In [39], Discrete Particle Swarm Optimization (DPSO) and B&B algorithms were presented to solve HW/SW partitioning problem, and DPSO was used to increase the speed of B&B. In [40], the algorithm which combines PSO and immune clone (IC) acquires better trade-off between partitioning time and optimization quality. However, the improved PSO strategies do not consider the searching diversification and searching intensification simultaneously. In this paper, the searching diversification is enhanced by simulating the squirrel behavior and the searching intensification is enhanced by integrating IWO. HNodeRank which makes use of problem-specific knowledge is used to initialize the population of PDPSO-IWO.

The disturbance approach in PDPSO-IWO is different from that in the algorithm of [41]. All particles move away from the global worst position in PDPSO-IWO. But random dimensions of the personal best positions produce disturbance for corresponding dimensions of the global best position in [41].

It has been proved that heuristic algorithms take too much time in addressing the large-scale HW/SW partitioning problem. In previous studies, PC clusters or super computers were used to solve the problem[21,36]. However, because of sequential computing method, the performance of such computer systems could not be easily promoted. With the rapid development of parallel technique, using parallel computing is an intuitive and simple way to speed up the algorithm for HW/SW partitioning problem. To accelerate the speed of HW/SW partitioning, parallel HW/SW partitioning method based on message passing Interface (MPI) was presented in [42-43]. In [44], the hardware accelerators such as field-programmable gate arrays (FPGAs), GPUs, and multicores were compared to solve HW/SW partitioning. In [45], GPU-based acceleration was used to solve two detailed cases of HW/SW partitioning. However, the proposed methods accelerate the whole process of HW/SW partitioning and the acceleration effect is not obvious. In this paper, we adopt the GPU parallel technique to accelerate the HW/SW communication cost computing which is the most time-consuming process for HW/SW partitioning problem.

2.3 Basic Particle Swarm Optimization

PSO, which was first introduced by Kennedy and Eberhart in 1995, emulates the social behavior of bird flocking and fish schooling[46]. Basic PSO has received significant interest from researchers studying in different research areas and has been applied to several real-world problems, such as image segmentation, travelling salesman problem and feature selection[47].

There are three parts in the evolution equation of basic PSO. The first part represents the previous velocity, which provides the necessary momentum for particles to roam across the search space. The second part, known as the cognitive component, represents the personal thinking of each particle. The third part is known as the social component, which represents the collaborative effect of particles[48].

3 Position Disturbed Particle Swarm Optimization with Invasive Weed Optimization

Basic PSO has many advantages, such as simply implementation and few parameters. However, PSO may easily get trapped into local optimum when solving complex multimodal optimization problems. To search

the global best solution, PDPSO-IWO is proposed to solve the HW/SW partitioning problem.

3.1 Position Disturbed Particle Swarm Optimization

It has been observed by biologists that social relationship is important for helping an animal to cope with its environment[49]. The ground squirrels warn conspecifics of a predator's presence through the production of alarm vocalizations[50]. And their peers can move away from the potential predatory threat[51]. The survival of the fittest is the law of nature[52]. The ground squirrels that cannot escape from predators will be eaten. To avoid the local optimum, the behavior is simulated in PDPSO. The worst particle in the population that cannot escape from predators is most likely to be eaten by the predator. Then the place near the worst particle in the population has potential predatory threat, and we can give following definitions.

Definition 1. *The particle which is the most likely to be eaten by the predator is the worst particle in the population.*

Definition 2. *The place from which the particles move away is the place where there is a potential predatory threat.*

Definition 3. *The place where there is a potential predatory threat is the global worst position in the population.*

In PDPSO, $xSize$ is the number of the particles and n is the number of nodes. $xSize$ particles search the global best position in the n-dimension search space. And each particle has the following attributes: a current position in the search space $\boldsymbol{X}_i$, a current velocity $\boldsymbol{V}_i$, and a personal best position $\boldsymbol{pbest}_i$ in the search space, then:

$$\begin{aligned}
&\boldsymbol{X}_i = (X_{i1}, \cdots, X_{id}, \cdots, X_{in}),\\
&\boldsymbol{V}_i = (V_{i1}, \cdots, V_{id}, \cdots, V_{in}),\\
&\boldsymbol{pbest}_i = (pbest_{i1}, \cdots, pbest_{id}, \cdots, pbest_{in}),
\end{aligned}$$

where $1 \leqslant i \leqslant xSize$, $1 \leqslant d \leqslant n$. $\boldsymbol{gbest} = (gbest_1, gbest_2, \cdots, gbest_n)$ is the global best position discovered by the whole population. $\boldsymbol{gworst} = (gworst_1, gworst_2, \cdots, gworst_n)$ is the global worst position. The algorithm may be trapped into the local optimum when the current global best value is almost the same as the previous one. Each particle approaches its personal best position and the global best position, and moves away from the global worst position, and then evolution equations of PDPSO are:

$$\begin{aligned}
V_{id}^{k+1} &= wV_{id}^k + c_1r_1(pbest_{id}^k - X_{id}^k) + \\
&\quad c_2r_2(gbest_d^k - X_{id}^k) - \\
&\quad c_3r_3(gworst_d^k - X_{id}^k), \qquad (2)\\
X_{id}^{k+1} &= X_{id}^k + V_{id}^{k+1}. \qquad (3)
\end{aligned}$$

w is the inertia factor used to balance the local and the global search abilities; c_1 and c_2 are learning coefficients; c_3 is the disturbed coefficient; r_1, r_2 and r_3 are random numbers uniformly distributed in the range $[0, 1]$; k is the number of iterations. The fourth part of (2) that particles move away from the global worst position can disturb particle distribution and make the population become various. Therefore, PDPSO can fight against premature convergence and avoid local optimum.

The direction that the i-th particle moves towards is shown in Fig.1. As shown in Fig.1, t_1 is the direction of the previous velocity. t_2 is the direction that the i-th particle approaches its personal best position. t_3 is the direction that the i-th particle approaches the global best position. t_4 is the direction that the i-th particle moves away from the global worst position. t_5 is the direction combined with t_1, t_2, t_3 and t_4. Therefore, t_5 is the final direction that the i-th particle moves towards.

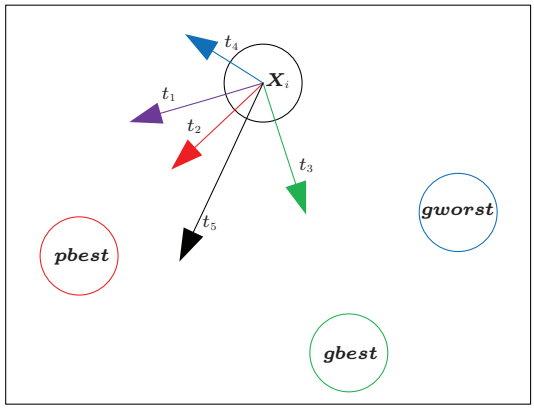

Fig.1. Direction that the i-th particle moves towards.

3.2 Convergence Analysis of PDPSO

The convergence of PDPSO is analyzed in this paper. According to (2), the velocity equation of PDPSO can be converted to:

$$\begin{aligned}
V_i^{k+1} &= wV_i^k + c_1r_1pbest_i^k + c_2r_2gbest^k - \\
&\quad c_3r_3gworst^k - (c_1r_1 + c_2r_2 - c_3r_3)X_i^k. \qquad (4)
\end{aligned}$$

When $k \to \infty$, V_i^k and V_i^{k+1} are equal to zero. And (4) can be converted to:

$$\lim_{k\to\infty} \boldsymbol{X}_i = \frac{c_1 r_1 \boldsymbol{pbest} + c_2 r_2 \boldsymbol{gbest} - c_3 r_3 \boldsymbol{gworst}}{c_1 r_1 + c_2 r_2 - c_3 r_3}.$$

Since r_1, r_2, r_3 are random numbers uniformly distributed in the range $[0, 1]$, the mean of $\boldsymbol{X}_i$ in PDPSO is:

$$\begin{aligned} &\lim_{k\to\infty} E(\boldsymbol{X}_i) \\ &= E(\frac{c_1 r_1 \boldsymbol{pbest} + c_2 r_2 \boldsymbol{gbest} - c_3 r_3 \boldsymbol{gworst}}{c_1 r_1 + c_2 r_2 - c_3 r_3}) \\ &= \frac{c_1 \boldsymbol{pbest} + c_2 \boldsymbol{gbest} - c_3 \boldsymbol{gworst}}{c_1 + c_2 - c_3} \\ &= (1 - \alpha + \beta)\boldsymbol{pbest} + \alpha \boldsymbol{gbest} - \beta \boldsymbol{gworst}, \end{aligned} \quad (5)$$

where $\alpha = \frac{c_2}{c_1+c_2-c_3}$, $\beta = \frac{c_3}{c_1+c_2-c_3}$. In the same way, the mean of $\boldsymbol{X}_i$ in PSO is:

$$\begin{aligned} \lim_{k\to\infty} E(\boldsymbol{X}_i) &= E(\frac{c_1 r_1 \boldsymbol{pbest} + c_2 r_2 \boldsymbol{gbest}}{c_1 r_1 + c_2 r_2}) \\ &= \frac{c_1 \boldsymbol{pbest} + c_2 \boldsymbol{gbest}}{c_1 + c_2} \\ &= (1 - \delta)\boldsymbol{pbest} + \delta \boldsymbol{gbest}, \end{aligned} \quad (6)$$

where $\delta = \frac{c_2}{c_1+c_2}$. According to (6), the diversification and the convergence of PSO are excellent in early iterations. However, the particles approach from ***pbest*** to ***gbest*** in late iterations. Then the diversification is reduced and the algorithm is trapped into local optimal[53]. According to (5) and (6), convergence values of PDPSO and PSO are different. Therefore, PDPSO can increase population diversity, and avoid premature convergence and local optimal. Then the global search ability can be enhanced.

3.3 Updating the Global Best Position

Invasive weed optimization is a population-based optimization algorithm inspired from invasive and robust nature of weeds in growth and colonizing[54]. IWO has been applied to solve many optimization problems, such as linear antenna array design, unit commitment problem and nonlinear equations systems[55].

IWO has a small amount of calculation and can enhance the searching intensification in PDPSO-IWO. Therefore, IWO is integrated to search a better position, by which the global best position is updated. The algorithm can be described in the following steps.

Step 1: *Initializing a Population.* A population of initial solutions is filled with the personal best position ***pbest***.

Step 2: *Reproduction.* The production of seeds by a weed is dependent on its own fitness and the fitness of its colony. The higher the weed's fitness, the more the seeds it produces. The formula of weeds producing seeds is:

$$weed_g = \frac{f - f_{\min}}{f_{\max} - f_{\min}}(w_{\max} - w_{\min}) + w_{\min},$$

where f is the current weed's fitness, $f_{\max}$ and $f_{\min}$ represent the maximum and the minimum fitness of the current population respectively, and $w_{\max}$ and $w_{\min}$ represent the maximum and the minimum value of a weed respectively.

Step 3: *Spatial Dispersal.* The generated seeds are randomly distributed in the search space such that they abode near to the parent plant by the normal distribution with the mean equal to zero and varying variance. However, the standard deviation (σ) of the random function is reduced in each iteration from an initial value σ_{init} to its final value σ_{final} by means of a nonlinear equation presented as follows:

$$\sigma_{\text{cur}} = \frac{(iter_{\max} - iter)^t}{(iter_{\max})^t}(\sigma_{\text{init}} - \sigma_{\text{final}}) + \sigma_{\text{final}},$$

where $iter_{\max}$ is the maximum number of iterations, σ_{cur} is the standard deviation in the current iteration, and t is the nonlinear modulation index.

Step 4: *Competitive Exclusion.* After passing some iterations, the number of produced plants in a colony reaches its maximum (P_{MAX}). In this step, a competitive mechanism is activated for eliminating undesirable plants with poor fitness and allowing fitter plants to reproduce more seeds.

Step 5: *Check Termination Criterion.* If the maximum number of iterations is reached, the best fitted plant is considered as the optimal solution; otherwise, return to step 2.

In step 1, to enhance the solution quality, initial solutions are the personal best positions ***pbest***. Hence, the population can remain diversiform and the search accuracy can be improved. To reduce the computational burden, the production of seeds by each weed in step 2 is set to the same value. The improved formula of weeds producing seeds is:

$$weed_g = l.$$

The number of seeds that each weed produces is set to l. Then the process of reproduction is simplified, and the calculation and runtime can be reduced. In each iteration, the global best position is updated by the improved IWO. Therefore, there is a greater possibility of escaping from local optimum in PDPSO-IWO.

4 Novel Hardware/Software Partitioning Method Based on PDPSO-IWO

4.1 Population Initialized by HNodeRank

In NodeRank, Adjust_Out and Adjust_In which are used to search a feasible solution are computed in each iteration. Since the time complexity of Adjust_Out and Adjust_In is $O(\log n(n+m))$, the computation is large in NodeRank. To reduce the computational burden, the HNodeRank algorithm which is combined with NodeRank and Base is proposed to solve the HW/SW partitioning problem. The binary search approach which is proposed in Base is used to search a feasible solution in HNodeRank. The formal description of the algorithm HNodeRank is shown in Algorithm 1.

In HNodeRank, a greedy solution of Q' in lines 6 and 15 is the solution produced by the algorithm Alg-greedy which is proposed in [19]. In line 23, the neighboring value of x' is the partitioning solution obtained from reversing t bits of x' which is proposed in [21].

Theorem 1. *The time complexity of HNodeRank is* $O(N(n\log n+(d+\log d)(n+m)))$, *where* n *is the number of nodes,* m *is the number of edges, and* $d = 1/\Delta u$, *and* N *is the maximum iteration time of HNodeRank.*

Proof. In HNodeRank, line 3 runs in $O(n\log n)$ time, and Alg-new3 runs in $O(n\log n + (d + \log d)(n + m))$ time in the worst case[19]. In lines 2∼35, the repeat-until runs in N time. Therefore, the time complexity of HNodeRank is $O(N(n\log n + (d + \log d)(n + m)))$. □

The solution quality of HNodeRank is similar to that of NodeRank while the time complexity is lower. The solution of PDPSO-IWO is initialized by HNodeRank, which can make use of problem-specific knowledge. Since PDPSO-IWO is an improved PSO, the proposed PDPSO-IWO can make use of the strong global search ability of general-purpose heuristic simultaneously. Therefore, the solution quality is enhanced and the runtime is reduced in PDPSO-IWO.

Algorithm 1. HNodeRank

Input: communication graph G and the constraint R
Output: a partitioning solution $best\_so\_far = (x_1, x_2, ..., x_n)$

1: $best\_so\_far = (x_1, x_2, ..., x_n)$, $E(c) = (x_1, x_2, ..., x_n)$, k=0, u=1, ε=0.02, $left$=0, $right$=1
2: **repeat**
3: Sort nodes $\{v_i\}_{i\leqslant n}$ according to $\frac{h_1}{s_1+E(c_1)} \geqslant \frac{h_2}{s_2+E(c_2)} \geqslant \cdots \geqslant \frac{h_n}{s_n+E(c_n)}$
4: **repeat**
5: $middle$=$\frac{left+right}{2}$
6: x'=a greedy solution of Q' with $middle$
7: **if** x' is a feasible solution of P **then**
8: $right$=$middle$
9: **else**
10: $left$=$middle$
11: **end if**
12: **until** $right - left < \Delta u$
13: u=$right$
14: **repeat**
15: x'=a greedy solution of Q' with u
16: **if** x' is a feasible solution of P **then**
17: **if** x' is better than $best\_so\_far$ **then**
18: $best\_so\_far$=x'
19: **end if**
20: Reset Δu
21: **else**
22: **for** $i = 1$ to n **do**
23: y = the i-th neighbor of x'
24: **if** y is a feasible solution of P and y is better than $best\_so\_far$ **then**
25: $best\_so\_far$=y
26: **end if**
27: **end for**
28: Δu=$(1+\varepsilon)\Delta u$
29: **end if**
30: $u = u - \Delta u$
31: **until** $u < 0$
32: $P_S^k(i) = \begin{cases} x_i P_S + (1 - x_i)P_H, & \text{if } k = 0 \\ \alpha P_S^{k-1}(i) + (1 - \alpha)x_i P_S + (1 - \alpha)(1 - x_i)P_H, & \text{if } k \geqslant 1 \end{cases}$
33: $E(c_i) = \sum_{j\in V,(i,j)\in E} ((P_S^k(i) \times P_H^k(j)) + (P_S^k(j) \times P_H^k(i))) \times c_{ij}$
34: $k = k + 1$
35: **until** $k > N$

4.2 Parallel HW/SW Partitioning Method Based on PDPSO-IWO

Since the HW/SW partitioning is treated as a discrete optimization problem, the discrete PDPSO-IWO is adopted in this paper. In discrete PDPSO-IWO, the position of each particle in each iteration is defined as follows:

$$X_{id}^k = \begin{cases} 1, \text{ if } r_4 < \text{sig}(V_{id}^k), \\ 0, \text{ otherwise,} \end{cases} \tag{7}$$

where the sigmoid function $\text{sig}(V_{id}^k) = \frac{1}{1+e^{-V_{id}^k}}$, and r_4 is a random number uniformly distributed in the range [0, 1]. If r_4 is smaller than $\text{sig}(V_{id}^k)$, then X_{id}^k is 1; otherwise, X_{id}^k is 0. The position of particle is the HW/SW partitioning solution and the fitness of particle is the hardware cost. The discrete PDPSO-IWO is used to optimize the minimization problem P in (1). Then the HW/SW partitioning method based on position disturbed particle swarm optimization with invasive weed optimization is named PDPSO-IWO. And the flowchart of PDPSO-IWO is shown in Fig.2.

In Fig.2, to reduce the computation, the algorithm may be trapped into the local optimum when the previous and current values of the global best position $fgbest$ are almost the same. The description of PDPSO-IWO is shown in Algorithm 2.

In PDPSO-IWO, X is initialized by the partition solution which is obtained by HNodeRank. $xSize$ is the number of particles, and n is the number of nodes. In line 3, $fpbest$ is the fitness of ***pbest***, $fgbest$ is the fitness of ***gbest*** and $fgworst$ is the fitness of ***gworst***. In lines 15∼19, the velocity and the position of each particle are updated by (2) and (3) respectively when the algorithm is trapped into the local optimum. Otherwise, they are updated by the evolution equation of basic PSO. *Adjust_In* is used to search a better solution in line 20. The global best position is updated by IWO in line 21, and the description of IWO is shown in Algorithm 3.

In PDPSO-IWO, the calculation of the HW/SW communication cost $C(\boldsymbol{x})$ in line 10 is computed by:

$$C(\boldsymbol{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}|x_i - x_j|, \tag{8}$$

where c_{ij} indicates the communication cost between node v_i and node v_j if they are in different contexts.

Theorem 2. *The time complexity of the HW/SW communication cost $C(\boldsymbol{x})$ is $O(n^2)$, where n is the number of nodes. The time complexity of PDPSO-IWO is $O(iter\_num \times xSize \times n^2)$, where n is the number of nodes, $xSize$ is the number of particles in the population, and $iter\_num$ is the maximum iteration time of PDPSO-IWO.*

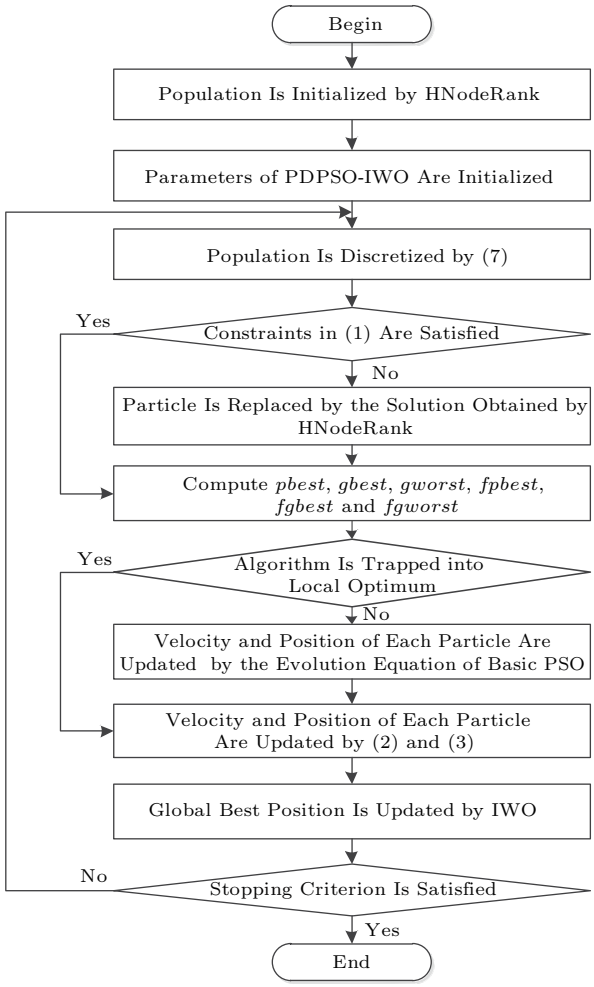

Fig.2. Flowchart of PDPSO-IWO.

Proof. In (8), there are two layers of iteration, and each iteration is implemented in $O(n)$. Therefore, the time complexity of $C(\boldsymbol{x})$ is $O(n^2)$. In PDPSO-IWO, the time complexity of HNodeRank is $O(N(n\log n + (d + \log d)(n + m)))$. In lines 9∼13, the time complexity is $O(xSize \times n^2)$. It takes $O(xSize \times n)$ to update ***pbest***, ***gbest***, $fpbest$ and $fgbest$ in line 14. *Adjust_In* in line 20 can be implemented in $O(\log n(n + m))$ time[36]. Since IWO is a simple optimization algorithm, we set the time complexity of IWO lower than $O(xSize \times n^2)$. Compared with $\log n(n+m)$, $xSize \times n^2$ is larger. The maximum iteration of PDPSO-IWO is $iter\_num$. Therefore, the time complexity of PDPSO-IWO is $O(iter\_num \times xSize \times n^2)$. □

Algorithm 2. PDPSO-IWO

Input: communication graph G and the constraint R
Output: global best partition solution $\boldsymbol{gbest}$ and its hardware cost $fgbest$
1: X_nr = the partition solution obtained by HNodeRank
2: $\boldsymbol{X} = (X\_nr, X\_nr, \cdots, X\_nr)$
3: $\boldsymbol{pbest} = \boldsymbol{X}, fpbest = 0, \boldsymbol{gbest} = (0, 0, \cdots, 0), fgbest = 0, \boldsymbol{gworst} = (0, 0, \cdots, 0), fgworst = inf, fgbestold = 1, iter = 0$
4: Initialize other parameters, such as w, c_1, c_2, c_3 and $epsx$, the individual number $xSize$ and the maximum iteration time $iter\_num$
5: **repeat**
6: $iter = iter+1$
7: $\boldsymbol{X}$ is discretized by (7)
8: $sx = S(\boldsymbol{X}), hx = H(\boldsymbol{X})$
9: **for** $i = 1$ to $xSize$ **do**
10: **if** $sx(i) + C(X_i) > R$ **then**
11: $X_i = X\_nr, hx(i) = H(X_i)$
12: **end if**
13: **end for**
14: Update $\boldsymbol{pbest}, \boldsymbol{gbest}, \boldsymbol{gworst}, fpbest, fgbest$ and $fgworst$
15: **if** $|fgbestold - fgbest| > epsx$ **then**
16: The velocity and the position of particles are updated by the evolution equation of basic PSO
17: **else**
18: The velocity and the position of particles are updated by (2) and (3), respectively
19: **end if**
20: $\boldsymbol{gbest} = Adjust\_In(\boldsymbol{gbest})$, $fgbest = H(\boldsymbol{gbest})$
21: The global best position is updated by IWO
22: $fgbestold = fgbest$
23: **until** $(iter > iter\_num)$

Algorithm 3. IWO

Input: personal best position $\boldsymbol{pbest}$
Output: updated global best position $\boldsymbol{gbest}$
1: Initialize the parameters, such as t, P_{MAX}, l, $iter_{\max}$, $k = 0$
2: solutions are initialized by $\boldsymbol{pbest}$
3: **repeat**
4: The standard deviation in the current iteration σ_{cur} is computed
5: Each weed in the population produces l weeds with the mean equal to zero and the variance equal to σ_{cur}
6: All seeds are ranked together with their parents
7: Weeds with lower fitness are eliminated to reach the maximum allowable population in the colony
8: k=k+1
9: **until** $k > iter_{\max}$

HW/SW partitioning method based on basic PSO is named PSO in this paper. The computing steps of PSO and PDPSO-IWO are almost the same. In PSO, the velocity and the position of particles are updated by the evolution equation of basic PSO. The global best position is not updated by IWO in PSO.

Theorem 3. *The space complexity of PDPSO-IWO is $O(xSize \times n)$, which is the same as that of PSO, where n is the number of nodes and $xSize$ is the number of the particles.*

Proof. In PDPSO-IWO, each particle gets close to the global best position, each particle's individual best position, and moves away from the global worst position in the population. The space complexity of the global best position and the global worst position is $O(n)$. The space complexity of each particle's individual best position is $O(xSize \times n)$. Therefore, the space complexity of PDPSO-IWO is $O(xSize \times n)$. In PSO, each particle approaches the global best position, each particle's individual best position. Therefore, the space complexity of PSO is $O(xSize \times n)$, which is the same as that of PDPSO-IWO. □

In PDPSO-IWO, the HW/SW communication cost is considered in each iteration. Since the time complexity of the HW/SW communication cost $C(\boldsymbol{x})$ is $O(n^2)$, the calculation of $C(\boldsymbol{x})$ is excessive. In order to accelerate the speed of PDPSO-IWO, the HW/SW communication cost computing which is the most time-consuming process for HW/SW partitioning method runs in an ordinary GPU parallel platform. In PDPSO-IWO, the communication cost computing for $xSize$ particles is in lines 9∼13. And the computing process runs in an ordinary GPU parallel platform. The parallelization of the communication cost in PDPSO-IWO is shown in Fig.3[56].

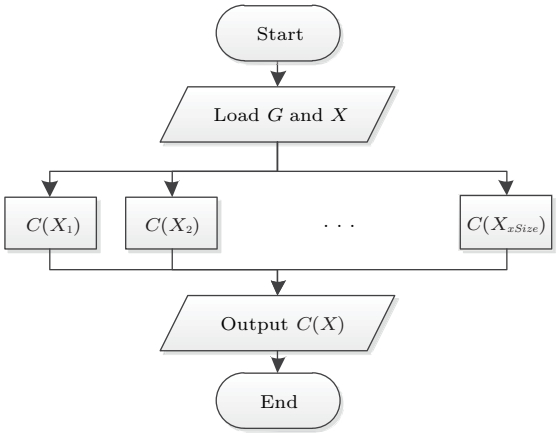

Fig.3. Parallelization of the communication cost.

As shown in Fig.3, the communication costs $C(X_1), C(X_2), ..., C(X_{xSize})$ are computed simultaneously in a typical GPU parallel platform, which saves a lot of computation time. All of communication costs are joined together and the HW/SW communication cost computing result is obtained[57]. Therefore, the runtime of PDPSO-IWO can be reduced efficiently.

5 Experimental Results and Analysis

In order to evaluate the performance of PDPSO-IWO, the algorithm is applied to solve several acknowledged benchmarks of HW/SW partitioning problem. The experiment environment is as follows: CPU: i7-4770 @3.4 GHz; physical memory: 16 GB; software: Matlab R2014b. The GPU platform running the parallel procedures is NVIDIA GTX 780, consisting of 12 SMs (streaming multiprocessors), 192 SPs (streaming processors) per SM. The clock frequency of each SP is 1.059 GHz. The size of global memory is 3 GB.

We adopt the same experimental parameters as [19]. Software costs s_i are generated as uniform random numbers from interval [1, 100]. Hardware costs h_i are generated as random numbers from a normal distribution with expected value $k \times s_i$ and a given standard deviation. The value of k only corresponds to the choice of units for software and hardware costs. The communication costs are generated as uniform random numbers from interval $[0, 2\rho s_{\max}]$, where $s_{\max}$ is the highest software cost. ρ is named communication to computation ratio (CCR), and it is taken as 0.1, 1, and 10, corresponding to computation-intensive case, intermediate case, and communication-intensive case, respectively. Constraint R is randomly generated as a uniform random number from interval $[0, 0.5 \sum s_i]$ or $[0.5 \sum s_i, \sum s_i]$. The two cases are indicated as R = low and R = high, respectively. There are six cases for different values of CCR and R, which are shown in Table 1.

Table 1. Cases for Different Values of CCR and R

Case	CCR	R
1	0.1	Low
2	0.1	High
3	1.0	Low
4	1.0	High
5	10.0	Low
6	10.0	High

For testing, we use benchmarks from MiBench[58] and task graphs in [4, 19, 21, 36]. The characteristics of the test benchmarks are summarized in Table 2. In Table 2, *size* and *number* are the scale size and the number of the HW/SW partitioning problems, respectively. For examples, *crc*32 is 32-bit cyclic redundancy check and *dijkstra* is the algorithm which computes

Table 2. Summary of Used Benchmarks

Name	n	m	*size*	*number*	Description
*crc*32	25	34	152	1	32-bit cyclic redundancy check
patricia	21	50	192	2	Routine to insert values into patricia tries, which are used to store routing tables
dijkstra	26	71	265	3	Compute the shortest paths in a graph
clustering	150	333	1 299	4	Image segmentation algorithm in a medical application
*rc*6	329	448	2 002	5	RC6 cryptographic algorithm
*random*1	1 000	1 000	5 000	6	Random graph
*random*2	1 000	2 000	8 000	8	Random graph
*random*3	1 000	3 000	11 000	10	Random graph
*random*4	1 500	1 500	7 500	7	Random graph
*random*5	1 500	3 000	12 000	11	Random graph
*random*6	1 500	45 000	16 500	13	Random graph
*random*7	2 000	2 000	10 000	9	Random graph
*random*8	2 000	4 000	16 000	12	Random graph
*random*9	2 000	6 000	22 000	14	Random graph

the shortest path in a graph. In Table 2, n and m denote the number of nodes and edges in the communication graph, respectively. The formula $size = 2n + 3m$ is utilized for the size evaluation of the graph. This is because each node is assigned two values (its hardware and software costs) and each edge is assigned three numbers (the identities of its endpoints and its communication cost).

The algrithms of Base, Heur, NodeRank, PSO and PDPSO-IWO are compared to solve the acknowledged benchmarks in Table 2. As important parameters, the learning coefficients c_1 and c_2, and the disturbed coefficient c_3 have great effect on the performance of PDPSO-IWO. In this paper, the appropriate values of c_1, c_2 and c_3 are obtained by experiments. In PDPSO-IWO, $c_1 = 1.5$, $c_2 = 1.5$, $c_3 = 1$, $epsx = 1$, $m = 3$, $P_{\mathrm{MAX}} = 50$, $l = 4$, and the maximum iteration number of IWO is 30. In PSO, $c_1 = 2$, and $c_2 = 2$. The parameters of NodeRank are set as the same values in [36]. In PDPSO-IWO and PSO, inertia factor (w) is decreasing from 0.9 to 0.7 linearly with the iterations, the individual number $xSize$ is 40 and the maximum iteration number is 50. Without loss of generality, we choose 100 as the number of the random instances for statistical comparison in our empirical study. The solution quality and the runtime of algorithms are compared.

5.1 Solution Quality

We define A and B as the HW/SW partitioning algorithms. HA is the average hardware cost of algorithm A and HB is the average hardware cost of algorithm B. In order to compare the solution quality of the algorithm, we define the improvement of algorithm A over algorithm B as:

$$imp = \left(1 - \frac{HA}{HB}\right) \times 100\%. \tag{9}$$

In (9), $imp > 0$, $imp = 0$ and $imp < 0$ reflect that the performance of algorithm A is better than, the same as, and worse than that of algorithm B, respectively. Fig.4 shows the improvement of different algorithms over Base, averaged over 100 instances on different cases. Abscissa represents the problem size while the vertical axis indicates the improvement of different algorithms over Base.

In Fig.4, generally, the solutions found by PDPSO-IWO are better than or similar to the solutions found by the other algorithms. For the case of the smaller communication cost, e.g., as shown in Fig.4(a), the solutions found by NodeRank, PSO and PDPSO-IWO

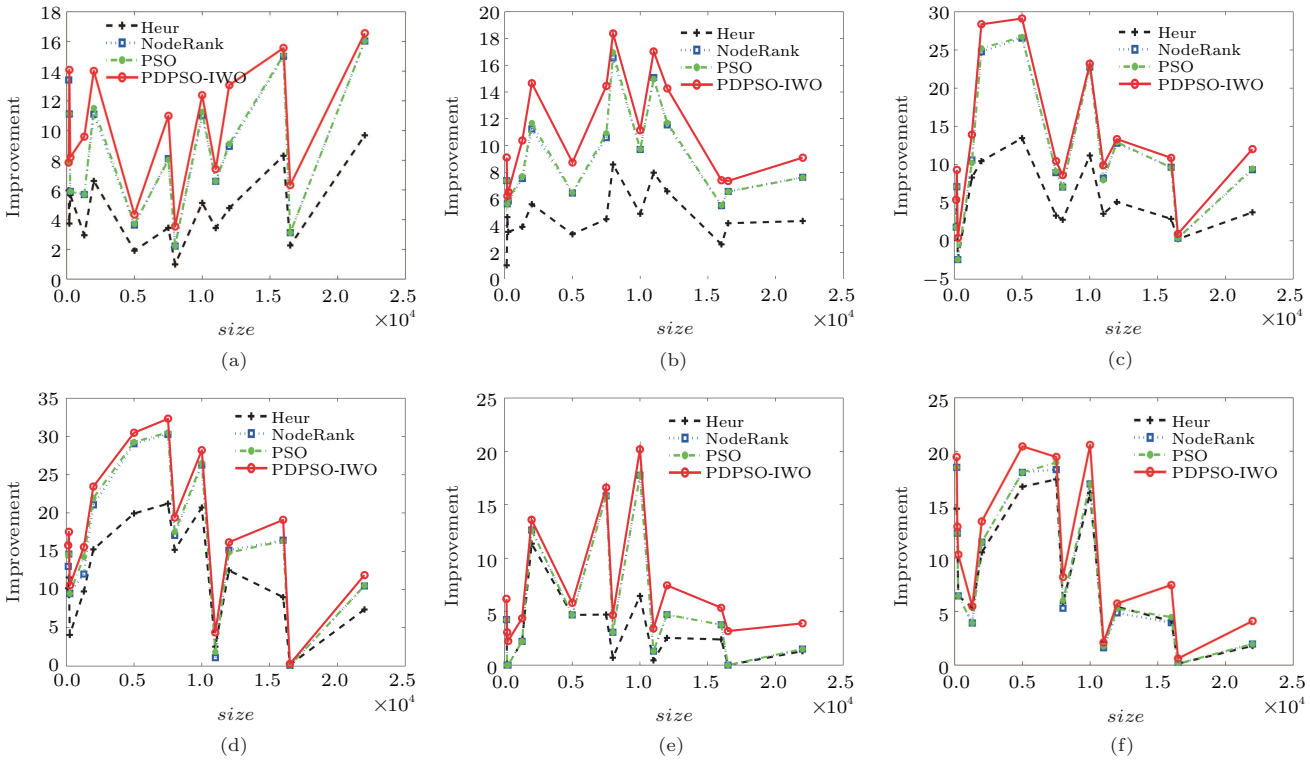

Fig.4. Improvements of different algorithms over Base, averaged over 100 instances on different cases. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Case 5. (f) Case 6.

are almost the same. This is because the HW/SW partitioning problem with small communication cost does not have a significant impact on the constraint. The optimal solutions can be easily found by other algorithms. On the other hand, with the increase of the communication cost, the constraints of the partitioning problem become far different from those of the standard knapsack problem. And it becomes harder to search the optimal solution. Since the global search ability of PDPSO-IWO is stronger, the improvement differences among PDPSO-IWO, PSO and NodeRank become greater, e.g., as shown in Fig.4(f).

In order to explore the worst-case performance of PDPSO-IWO, we investigate the distribution of the improvement over the algorithm Base for 100 random instances. Without loss of generality, the *imp* values are collected from −50% to 50%. This corresponds to the distribution interval [−50, −45, ..., −10, −5, 0, 5, 10, ..., 45, 50] with the unit length of 5 in x-axis. We choose the representative benchmark *Random*2 with size 8 000. And the distribution of improvements over Base on different cases with the benchmark is shown in Fig.5. The ordinate represents the count of solution instances while the abscissa indicates the corresponding distribution interval.

From Fig.5, the corresponding improvements are distributed in a relatively large interval for the cases of small communication cost, e.g., as shown in Fig.5(a). And the interval becomes smaller with the increasing communication cost, e.g., as shown in Fig.5(f). PDPSO-IWO can produce higher quality solutions than the other algorithms for most instances especially when the communication cost is relatively large.

5.2 Runtime of Algorithms

The speedup S_p is used to evaluate the GPU parallel efficiency of the HW/SW communication cost $C(\boldsymbol{x})$:

$$S_p = \left(\frac{T_{\text{seq}}}{T_{\text{par}}}\right) \times 100\%,$$

where T_{seq} is the runtime of communication cost in sequential computing environment and T_{par} is the runtime in parallel computing environment. Fig.6 shows the speedup of HW/SW communication cost, averaged over 100 instances on different cases. The abscissa represents the problem number while the ordinate indicates the average speedup.

As shown in Fig.6, generally, the speedup becomes larger and the advantage of parallel computing becomes more obvious as the problem size and the communication cost increase. The speedup is greater than 1 when the problem size is larger than 5 000. The speedup is

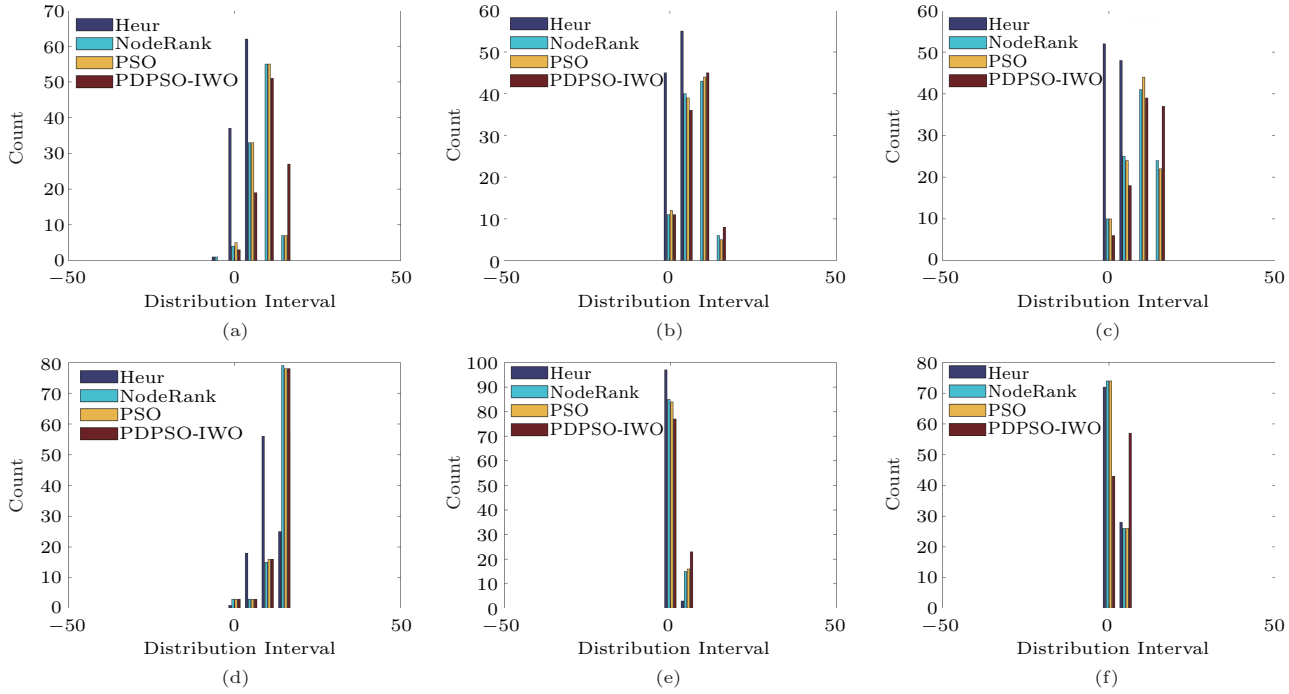

Fig.5. Distribution of improvements over Base, 100 random instances on different cases with size 8 000. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Case 5. (f) Case 6.

smaller than 1 when the problem size is equal to or less than 5 000. This is because there are extra operations in GPU parallel computing, such as thread creation, task creation and data communication. The operations take extra runtime. When the problem size is small, the extra runtime takes up much runtime of the HW/SW communication cost computing. Then the runtime of $C(\boldsymbol{x})$ in parallel computing environment is longer. When the problem size is large, the calculation of $C(\boldsymbol{x})$ is great. Compared with the runtime of $C(\boldsymbol{x})$, the extra runtime is small. Then the runtime of $C(\boldsymbol{x})$ in parallel computing environment is shorter.

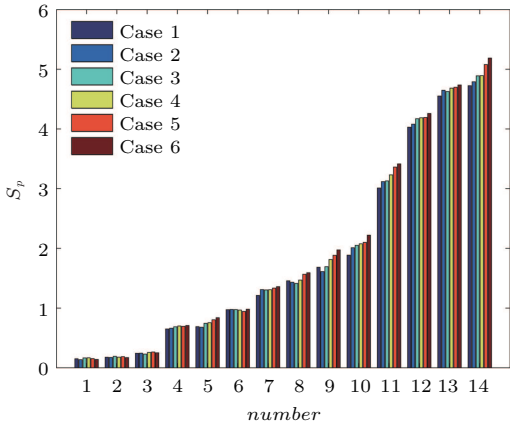

Fig.6. Speedup of HW/SW communication cost, averaged over 100 instances on different cases.

The communication cost computing of PDPSO-IWO runs in GPU parallel platform, while the computing of other algorithms runs in sequential computing environment. Table 3 and Table 4 show the runtime for case 1 and case 6, respectively.

Table 3. Comparisons in Runtime (in Second) for Case 1 (Averaged over 100 Instances)

size	Base	Heur	NodeRank	PSO	PDPSO-IWO
152	0.002 1	0.001 5	0.001 2	0.022 2	0.842 1
192	0.002 2	0.002 2	0.001 4	0.020 0	0.960 9
265	0.002 3	0.002 3	0.001 9	0.023 2	1.001 8
333	0.002 7	0.002 9	0.023 5	0.123 2	1.132 4
2 002	0.004 1	0.007 2	0.074 1	0.260 5	1.362 3
5 000	0.009 4	0.034 9	0.442 8	1.077 5	2.165 0
7 500	0.011 0	0.090 2	1.022 4	2.587 1	3.666 3
8 000	0.011 0	0.081 5	0.893 2	2.235 3	3.087 7
10 000	0.016 7	0.151 3	1.741 5	4.018 1	4.820 9
11 000	0.016 3	0.114 6	1.311 9	3.140 1	3.766 3
12 000	0.017 2	0.193 0	1.673 0	4.986 4	5.510 8
16 000	0.022 0	0.311 0	3.528 8	8.127 7	8.010 7
16 500	0.022 7	0.424 1	4.682 2	10.560 8	9.756 4
22 000	0.024 5	0.510 2	5.463 4	13.546 2	12.125 2

Table 4. Comparisons in Runtime (in Second) for Case 6 (Averaged over 100 Instances)

size	Base	Heur	NodeRank	PSO	PDPSO-IWO
152	0.002 8	0.001 5	0.002 3	0.021 0	1.013 0
192	0.003 5	0.001 3	0.001 7	0.020 7	1.024 7
265	0.003 5	0.002 4	0.002 4	0.024 3	0.998 3
333	0.004 0	0.003 5	0.030 1	0.131 5	1.164 5
2 002	0.004 4	0.009 3	0.086 0	0.289 0	1.304 8
5 000	0.012 8	0.050 0	0.516 6	1.320 7	2.453 8
7 500	0.014 0	0.109 7	1.072 8	2.620 0	3.614 5
8 000	0.014 8	0.101 9	0.981 7	2.524 5	3.425 2
10 000	0.025 8	0.194 7	1.952 2	4.538 2	5.340 4
11 000	0.025 9	0.154 6	1.469 8	3.741 0	4.396 8
12 000	0.026 4	0.233 5	2.222 0	5.406 9	5.954 9
16 000	0.030 2	0.418 6	4.027 0	9.412 8	9.102 5
16 500	0.030 5	0.308 2	4.502 2	12.452 1	11.376 9
22 000	0.032 7	0.619 8	5.983 0	14.145 8	12.368 3

As shown in Table 3 and Table 4, the runtime of algorithms becomes longer when the problem size becomes larger. Generally, the runtime of algorithms in Table 4 is longer than the corresponding value in Table 3. This is because the computational burden becomes greater when the HW/SW communication cost becomes larger. The runtime of PDPSO-IWO is longer than that of PSO when the size is smaller than 16 000. This is because the computational burden of PDPSO-IWO is larger than that of PSO and the advantage of the parallel computing is not obvious when the size is small. On the other hand, the HW/SW communication cost which runs in an ordinary GPU parallel platform becomes excessive when the size is large. And the runtime of PDPSO-IWO is shorter than that of PSO when the size is equal to or greater than 16 000.

5.3 Large-Scale HW/SW Partitioning

As the devolvement of modern embedded systems, the scale of HW/SW partitioning becomes larger. In order to analyze the performance of PDPSO-IWO further, the algorithm is used to solve the large-scale HW/SW partitioning problem which is shown in Table 5.

The re-excited PSO (RPSO) algorithm proposed in [37] is used to solve the HW/SW partitioning problem. The parameters of RPSO are set to the same values of PSO. The algorithms of Base, Heur, NodeRank, PSO, RPSO and PDPSO-IWO are compared to solve the large-scale HW/SW partitioning problem on the typical case 3. Fig.7 shows the improvement and runtime of different algorithms, averaged over 100 instances on case 3.

Table 5. Summary of Large-Scale HW/SW Problem

Name	n	m	$size$	$number$
*random*10	5 000	5 000	25 000	15
*random*11	5 000	10 000	40 000	17
*random*12	6 000	12 000	48 000	18
*random*13	7 000	7 000	35 000	16
*random*14	7 000	14 000	56 000	20
*random*15	8 000	16 000	64 000	21
*random*16	9 000	18 000	72 000	22
*random*17	10 000	10 000	50 000	19
*random*18	10 000	20 000	80 000	23

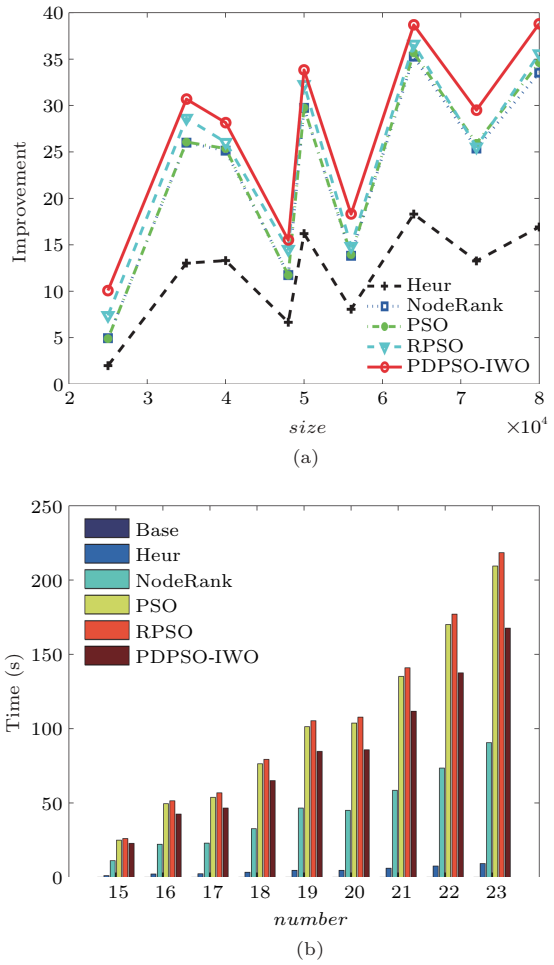

Fig.7. Performance of different algorithms, averaged over 100 random instances on case 3. (a) Improvement. (b) Runtime.

As shown in Fig.7, the differences of solution quality among PDPSO-IWO, NodeRank, PSO and RPSO become larger when the size increases. PDPSO-IWO which enhances the HW/SW partitioning solution distinctly has stronger search ability for the large-scale HW/SW partitioning problem. The solution quality of RPSO is higher than that of PSO, but is lower than that of PDPSO-IWO. When the size is 80 000, the runtime of PDPSO-IWO is shorter than that of PSO by 41.759 2 s, and is shorter than that of RPSO by 50.767 2 s. The advantage of the parallel computing is obvious when the scale is large. Therefore, the GPU parallel PDPSO-IWO is an efficient method to solve the large-scale HW/SW partitioning problem.

6 Conclusions

A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization was proposed. In PDPSO-IWO, the particles move away from the worst particle in the population, and the searching population remains varied. To improve the initialization and reproduction strategies, the IWO was integrated to search a better position, with which the global best position is updated. The search accuracy and the solution quality are improved. HNodeRank was proposed to initialize the population, and the solution quality was enhanced further. The HW/SW communication cost ran in GPU parallel computing environment, and the runtime was reduced efficiently in the experiments.

In future, we will explore at least four directions but not limited. 1) We will adopt the similar modification strategy from the improved optimization methods[59], and therefore we should expect that the performance of PDPSO-IWO can be enhanced substantially. 2) GPU has been a popular acceleration platform in scientific computation[60-61]. In our current research, we simply use the Matlab parallel programming on GPU. How to explore the GPU power with lower-level programming (such as thread/warp/block parallel computing with CUDA) to accelerate the PDPSO-IWO algorithm for HW/SW partitioning should be discussed. 3) As the optimization approaches are widely used in CAD&CG[62-65], videos[66-67] and images[68-70], the PDPSO-IWO will be extended and used in these areas. 4) The algorithm proposed in this paper will be used to solve more HW/SW partitioning models and real projects such as MPSoC, coupled CPU-GPU and high-level VLSI synthesis[71].

References

[1] Trappey A J C, Shen W, Cha J J. Special issue editorial on advances in collaborative systems engineering for product design, production and service network. *Journal of Systems Science and Systems Engineering*, 2016, 25(2): 139-141.

[2] Zhang Y G, Luo W J, Zhang Z M, Li B, Wang X F. A hardware/software partitioning algorithm based on artificial immune principles. *Applied Soft Computing*, 2008, 8(1): 383-391.

[3] Hidalgo J I, Lanchares J. Functional partitioning for hardware-software codesign using genetic algorithms. In *Proc. the 23rd EUROMICRO Conference on New Frontiers of Information Technology*, Sept. 1997, pp.631-638.

[4] Arató P, Mann Z A, Orbán A. Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems*, 2005, 10(1): 136-156.

[5] Wu J G, Srikanthan T. Low-complex dynamic programming algorithm for hardware/software partitioning. *Information Processing Letters*, 2006, 98(2): 41-46.

[6] Madsen J, Grode J, Knudsen P V, Petersen M E, Haxthausen A. LYCOS: The Lyngby co-synthesis system. *Design Automation for Embedded Systems*, 1997, 2(2): 195-235.

[7] Strachacki M. Speedup of branch and bound method for hardware/software partitioning. In *Proc. the 1st International Conference on Information Technology*, May 2008.

[8] Chatha K S, Vemuri R. Hardware-software partitioning and pipelined scheduling of transformative applications. *IEEE Transactions on Very Large Scale Integration Systems*, 2002, 10(3): 193-208.

[9] Wu J G, Thambipillai S. A branch-and-bound algorithm for hardware/software partitioning. In *Proc. the 4th International Symposium on Signal Processing and Information Technology*, Dec. 2004, pp.526-529.

[10] Niemann R, Marwedel P. An algorithm for hardware/software partitioning using mixed integer linear programming. *Design Automation for Embedded Systems*, 1997, 2(2): 165-193.

[11] Gupta R K, Coelho Jr C N, De Micheli G. Synthesis and simulation of digital systems containing interacting hardware and software components. In *Proc. the 29th ACM/IEEE Design Automation Conference*, June 1992, pp.225-230.

[12] Gupta R K, De Micheli G. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 1993, 10(3): 29-41.

[13] Wiangtong T, Cheung P Y K, Luk W. Comparing three heuristic search methods for functional partitioning in hardware-software codesign. *Design Automation for Embedded Systems*, 2002, 6(4): 425-449.

[14] Purnaprajna M, Reformat M, Pedrycz W. Genetic algorithms for hardware-software partitioning and optimal resource allocation. *Journal of Systems Architecture*, 2007, 53(7): 339-354.

[15] Dick R P, Jha N K. MOGAC: A multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1998, 17(10): 920-935.

[16] Janakiraman N, Kumar P N. Multi-objective module partitioning design for dynamic and partial reconfigurable system-on-chip using genetic algorithm. *Journal of Systems Architecture—Embeded Systems Design*, 2014, 60(1): 119-139.

[17] Henkel J, Ernst R. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Transactions on Very Large Scale Integration Systems*, 2001, 9(2): 273-289.

[18] Liu P, Wu J G, Wang Y J. Hybrid algorithms for hardware/software partitioning and scheduling on reconfigurable devices. *Mathematical and Computer Modelling*, 2013, 58(1/2): 409-420.

[19] Wu J G, Srikanthan T, Chen G. Algorithmic aspects of hardware/software partitioning: 1D search algorithms. *IEEE Transactions on Computers*, 2010, 59(4): 532-544.

[20] Eles P, Peng Z, Kuchcinski K, Doboli A. System level hardware/software partitioning based on simulated annealing and tabu search. *Design Automation for Embedded Systems*, 1997, 2(1): 5-32.

[21] Wu J G, Wang P, Lam S K, Srikanthan T. Efficient heuristic and tabu search for hardware/software partitioning. *The Journal of Supercomputing*, 2013, 66(1): 118-134.

[22] Wang G, Gong W, Kastner R. A new approach for task level computational resource bi-partitioning. In *Proc. the 15th IASTED International Conference on Parallel and Distributed Computing and Systems*, June 2003, pp.439-444.

[23] Xiong Z, Li S, Chen J. Hardware/software partitioning based on ant optimization with initial pheromone. *Computer Research and Development*, 2005, 42(12): 2176-2183. (in Chinese)

[24] Abdelhalim M B, Salama A E, Habib S E D. Hardware software partitioning using particle swarm optimization technique. In *Proc. the 6th International Workshop on System-on-Chip for Real-Time Applications*, Dec. 2006, pp.189-194.

[25] Abdelhalim M B, Habib S E D. An integrated high-level hardware/software partitioning methodology. *Design Automation for Embedded Systems*, 2011, 15(1): 19-50.

[26] Kernighan B W, Lin S. An efficient heuristic procedure for partitioning graphs. *Bell Labs Technical Journal*, 1970, 49(2): 291-307.

[27] Fiduccia C M, Mattheyses R M. A linear-time heuristic for improving network partitions. In *Proc. the 19th Design Automation Conference*, June 1982, pp.175-181.

[28] Saab Y G. A fast and robust network bisection algorithm. *IEEE Transactions on Computers*, 1995, 44(7): 903-913.

[29] Vahid F, Gajski D D. Clustering for improved system-level functional partitioning. In *Proc. the 8th International Symposium on System Synthesis*, Sept. 1995, pp.28-35.

[30] López-Vallejo M, López J C. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems*, 2003, 8(3): 269-297.

[31] Vahid F, Le T D. Extending the Kernighan/Lin heuristic for hardware and software functional partitioning. *Design Automation for Embedded Systems*, 1997, 2(2): 237-261.

[32] Arató P, Juhász S, Mann Z A, Orban A, Papp D. Hardware-software partitioning in embedded system design. In *Proc. IEEE International Symposium on Intelligent Signal Processing*, Sept. 2003, pp.197-202.

[33] Grode J, Knudsen P V, Madsen J. Hardware resource allocation for hardware/software partitioning in the LYCOS system. In *Proc. the Conference on Design, Automation and Test in Europe*, Feb. 1998, pp.22-27.

[34] Ernst R, Henkel J, Benner T. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, 1993, 10(4): 64-75.

[35] Wolf W H. An architectural co-synthesis algorithm for distributed, embedded computing systems. *IEEE Transactions on Very Large Scale Integration Systems*, 1997, 5(2): 218-229.

[36] Chen Z, Wu J G, Song G Z, Chen J L. NodeRank: An efficient algorithm for hardware/software partitioning. *Chinese Journal of Computers*, 2013, 36(10): 2033-2040. (in Chinese)

[37] Abdelhalim M B, Salama A E, Habib S E D. Constrained and unconstrained hardware-software partitioning using particle swarm optimization technique. In *Proc. IIESS*, May 30-June 1, pp.207-220.

[38] Bhattacharya A, Konar A, Das S, Grosan C, Abraham A. Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. In *Proc. the 2nd International Conference on Complex, Intelligent and Software Intensive Systems*, March 2008, pp.171-176.

[39] Eimuri T, Salehi S. Using DPSO and B&B algorithms for hardware/software partitioning in co-design. In *Proc. the 2nd International Conference on Computer Research and Development*, May 2010, pp.416-420.

[40] Luo L, He H, Liao C, Dou Q, Xu W X. Hardware/software partitioning for heterogeneous multicore SoC using particle swarm optimization and immune clone (PSO-IC) algorithm. In *Proc. IEEE International Conference on Information and Automation* (*ICIA*), June 2010, pp.490-494.

[41] Xue J, Jin Y. Position disturbed particle swarm optimization. *Computer Engineering and Design*, 2014, 35(3): 1037-1040. (in Chinese)

[42] Wu Y, Zhang H, Yang H. Research on parallel HW/SW partitioning based on hybrid PSO algorithm. In *Proc. the 9th International Conference on Algorithms and Architectures for Parallel Processing*, June 2009, pp.449-459.

[43] Farahani A F, Kamal M, Salmani-Jelodar M. Parallel-genetic-algorithm-based HW/SW partitioning. In *Proc. the 5th IEEE International Symposium on Parallel Computing in Electrical Engineering*, Sept. 2006, pp.337-342.

[44] Cooke P, Fowers J, Brown G, Stitt G. A tradeoff analysis of FPGAS, GPUS, and multicores for sliding-window applications. *ACM Transactions on Reconfigurable Technology and Systems*, 2015, 8(1): 2:1-2:24.

[45] Bordoloi U D, Chakraborty S. GPU-based acceleration of system-level design tasks. *International Journal of Parallel Programming*, 2010, 38(3/4): 225-253.

[46] Du H Z, Xia N, Jiang J G, Xu L N, Zheng R. A monte carlo enhanced PSO algorithm for optimal QoM in multi-channel wireless networks. *Journal of Computer Science and Technology*, 2013, 28(3): 553-563.

[47] Inbarani H H, Azar A T, Jothi G. Supervised hybrid feature selection based on PSO and rough sets for medical diagnosis. *Computer Methods and Programs in Biomedicine*, 2014, 113(1): 175-185.

[48] Ratnaweera A, Halgamuge S K, Watson H C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 2004, 8(3): 240-255.

[49] Herzenstein M, Dholakia U M, Andrews R L. Strategic herding behavior in peer-to-peer loan auctions. *Journal of Interactive Marketing*, 2011, 25(1): 27-36.

[50] Swan D C, Hare J F. The first cut is the deepest: Primary syllables of Richardson's ground squirrel, Spermophilus richardsonii, repeated calls alert receivers. *Animal Behaviour*, 2008, 76(1): 47-54.

[51] Thompson A B, Hare J F. Neighbourhood watch: Multiple alarm callers communicate directional predator movement in Richardson's ground squirrels, Spermophilus richardsonii. *Animal Behaviour*, 2010, 80(2): 269-275.

[52] Patel A. Survival of the fittest and zero sum games. *Fluctuation and Noise Letters*, 2002, 2(4): 279-284.

[53] Clerc M, Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 2002, 6(1): 58-73.

[54] Mehrabian A R, Lucas C. A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics*, 2006, 1(4): 355-366.

[55] Pourjafari E, Mojallali H. Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering. *Swarm and Evolutionary Computation*, 2012, 4: 33-43.

[56] Yang Y, Li C, Zhou H Y. CUDA-NP: Realizing nested thread-level parallelism in GPGPU applications. *Journal of Computer Science and Technology*, 2015, 30(1): 3-19.

[57] Qi R Z, Wang Z J, Li S Y. A parallel genetic algorithm based on spark for pairwise test suite generation. *Journal of Computer Science and Technology*, 2016, 31(2): 417-427.

[58] Guthaus M R, Ringenberg J S, Ernst D, Austin T M, Mudge T, Brown R B. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. the 4th IEEE Annual Workshop on Workload Characterization, Workload Characterization*, Dec. 2001, pp.3-14.

[59] Yan X H, He F Z, Chen Y L, Yuan Z Y. An efficient improved particle swarm optimization based on prey behavior of fish schooling. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 2015, 9(4): JAMDSM0048:1-JAMDSM0048:11.

[60] Zhou Y, He F Z, Qiu Y M. Optimization of parallel iterated local search algorithms on graphics processing unit. *The Journal of Supercomputing*, 2016, 72(6): 2394-2416.

[61] Zhou Y, He F Z, Qiu Y M. Dynamic strategy based parallel ant colony optimization on GPUs for TSPs. *Science China Information Sciences*, 2016, doi 10.1007/s11432-015-0594-2. (to be appeared)

[62] Wu Y Q, He F Z, Zhang D J, Li X X. Service-oriented feature-based data exchange for cloud-based design and manufacturing. *IEEE Transactions on Services Computing*, 2016, pp(99), doi 10.1109/TSC.2015.2501981.

[63] Zhang D J, He F Z, Han S H, Li X X. Quantitative optimization of interoperability during feature-based data exchange. *Integrated Computer-Aided Engineering*, 2016, 23(1): 31-50.

[64] Cheng Y, He F Z, Wu Y Q, Zhang D J. Meta-operation conflict resolution for human-human interaction in collaborative feature-based CAD systems. *Cluster Computing*, 2016, 19(1): 237-253.

[65] Lv X, He F Z, Cai W W, Cheng Y. A string-wise CRDT algorithm for smart and large-scale collaborative editing systems. *Advanced Engineering Informatics*, 2016, doi 10.1016/j.aei.2016.10.005.

[66] Li K, He F Z, Chen X. Real-time object tracking via compressive feature selection. *Frontiers of Computer Science*, 2016, 10(4): 689-701.

[67] Sun J, He F Z, Chen Y, Chen X. A multiple template approach for robust tracking of fast motion target. *Applied Mathematics—A Journal of Chinese Universities*, 2016, 31(2): 177-197.

[68] Ni B, He F Z, Yuan Z Y. Segmentation of uterine fibroid ultrasound images using a dynamic statistical shape model in HIFU therapy. *Computerized Medical Imaging and Graphics*, 2015, 46(part3): 302-314.

[69] Ni B, He F Z, Pan Y T, Yuan Z Y. Using shapes correlation for active contour segmentation of uterine fibroid ultrasound images in computer-aided therapy. *Applied Mathematics—A Journal of Chinese Universities*, 2016, 31(1): 37-52.

[70] Yu H P, He F Z, Pan Y T, Chen X. An efficient similarity-based level set model for medical image segmentation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 2016, 10(8): JAMDSM0100.

[71] Jiang G, Wu J, Lam S K, Srikanthan T, Sun J. Algorithmic aspects of graph reduction for hardware/software partitioning. *The Journal of Supercomputing*, 2015, 71(6): 2251-2274.

Xiao-Hu Yan received his B.S. degree in information and computing science from Huazhong Agricultural University, Wuhan, in 2008, and his M.S. degree in computer application from North China Electric Power University, Beijing, in 2010. Currently, he is a Ph.D. candidate of the School of Computer Science in Wuhan University, Wuhan. His research interests include hardware/software partitioning and optimization algorithm.

Fa-Zhi He received his B.S., M.S. and Ph.D. degrees in computer science from Wuhan University of Technology, Wuhan. He was a postdoctoral researcher of Zhejiang University, Hangzhou, a visiting researcher of Korea Institute of Science and Technology and a visiting faculty member of the University of North Carolina at Chapel Hill. Now he is a professor of the School of Computer Science in Wuhan University, Wuhan. His research interests are hardware/software partitioning, computer-aided design, computer graphics, and collaborative computation.

Yi-Lin Chen is currently a Ph.D. candidate at the School of Computer Science in Wuhan University, Wuhan. His research interests are pattern recognition, image processing, and computer graphics.