Received September 28, 2017, accepted November 14, 2017, date of publication November 22, 2017,
date of current version February 14, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2776295

A Parallel Genetic Algorithm With Dispersion Correction for HW/SW Partitioning on Multi-Core CPU and Many-Core GPU

NENG HOU[1], FAZHI HE[1,2], YI ZHOU[3], YILIN CHEN[1], AND XIAOHU YAN[1]
[1]State Key Laboratory of Software Engineering, School of Computer Science, Wuhan University, Wuhan 430072, China
[2]State Key Laboratory of Digital Manufacturing Equipment and Technology, Wuhan 430074, China
[3]Engineering Research Center of Metallurgical Automation and Measurement Technology, School of Information Science and Engineering, Wuhan University of Science and Technology, Wuhan 430081, China

Corresponding author: Fazhi He (fzhe@whu.edu.cn)

This work was supported in part by the National Science Foundation of China under Grant 61472289 and in part by the Open Project Program of State Key Laboratory of Digital Manufacturing Equipment and Technology, HUST, under Grant DMETKF2017016.

ABSTRACT In hardware/software (HW/SW) co-design, hardware/software partitioning is an essential step in that it determines which components to be implemented in hardware and which ones in software. Most of HW/SW partitioning problems are NP hard. For large-size problems, heuristic methods have to be utilized. This paper presents a parallel genetic algorithm with dispersion correction for HW/SW partitioning on CPU-GPU. First, an enhanced genetic algorithm with dispersion correction is presented. The under-constraint individuals are marched to feasible region step by step. In this way, the intensification can be enhanced as well as the constraint problem can be handled. Second, the individuals performing costs computation and dispersion correction are run in parallel. For a given problem size, the overall run-time can be reduced while the diversity of genetic algorithm can be kept. Third, especially when a number of under-constraint individuals should be corrected in an irregular way, the computation process is complicated and the computation overhead is large. Therefore, we present a novel parallel strategy by leveraging the parallel power of a multi-core CPU and that of a many-core GPU. The proposed strategy computes the costs of each individual in parallel on GPU and corrects the under-constraint individuals in parallel on the multi-core CPU. In this way, a highly efficient parallel computing can be achieved in which dozens of irregular correction computing steps are mapped to the multi-core CPU and thousands of regular cost computing steps are mapped to the many-core GPU. Fourth, at each iteration of the hybrid parallel strategy, the solution vectors of individuals are transferred to the GPU and their costs are transferred back to the CPU. In order to further improve the efficiency of proposed algorithm, we propose an asynchronous transfer pattern (stream concurrency pattern) for CPU-GPU, in which the transfer process and computation process are overlapped and eventually the overall run-time can be reduced further. Finally, the experiments show that the solution quality obtained by our method is competitive with existing heuristic methods in reasonable time. Furthermore, by combining with the multi-core CPU and many-core GPU, the running time of the proposed method is efficiently reduced.

INDEX TERMS Hardware/software co-design, heuristic method, genetic algorithm, multi-core CPU, many-core GPU.

I. INTRODUCTION

In most embedded systems, a hardware platform is made up of the predominant digital components which execute software application programs. Due to the increasingly need of raising the potential quality and shortening the development time of electronic products, plus the emergence of computer aided design (CAD) tools, hardware/ software co-design has become a hot topic since 1990s [1]–[3]. In hardware/software co-design, hardware/ software partitioning (HW/SW partitioning) is an essential procedure because it determines which components to be implemented on hardware and which ones on software. HW/SW partitioning can improve the overall performance of modern embedded systems. Over the past two decades, a number of works have been done for HW/SW partitioning [4], [5].

The target architecture of embedded systems generally consists of software component and hardware component. The software component usually refers to RISC CPU.

The hardware component refers to *Field Programmable Gate Array* (FPGA) or *Application Specification Integrated Circuit* (ASIC). When the application is implemented on hardware, it is significantly faster and more power-efficient. However, the spending is very high. Relatively, when the application is implemented on software, it is power-consuming and the spending is small, but the speed is slow. HW/SW partitioning can ensure an optimal trade-off between cost, performance and power.

At the theory and methodology level, HW/SW partitioning was taken a more theoretical description. The application to be partitioned is given in the form of a task graph, or a set of task graphs. For example, the system to be partitioned was modeled as an undirected communication graph [6]. Based on the model, Arató *et al* categorized two different versions of HW/SW partitioning. One can be solved optimally in polygon time complexity, while the other was NP-hard in the strong sense [7].

On algorithmic aspects of HW/SW partitioning, two categories of algorithms are utilized, namely exact methods and approximate methods (widely implemented with heuristic algorithms). The exact algorithm is used to obtain an exact solution for the small size problem. The typical exact methods include dynamic programming [8], linear programming [9] and branch and bound [10]. When the problem size becomes large and the solution space of HW/SW partitioning increases exponentially, exploring the exact solution in reasonable time is impractical. Heuristic algorithms therefore become popular approximate alternatives due to their superior ability to obtain good quality solutions within the limited computing time.

In the early stage of HW/SW partitioning, domain-specific heuristics, including hardware-oriented heuristic algorithm and software-oriented heuristic algorithm, were proposed [11], [12]. The former starts with a complete hardware solution and iteratively moves parts of the system to the software, while the latter starts with a software program moving pieces to hardware. The hardware-oriented approach is treated as a performance-constraints method. The software-oriented approach is treated as a time-constraint method.

After then, many general heuristics or metaheuristics were also adopted, such as genetic algorithm [6], [7], [13], ant colony algorithm [14], [15], artificial bees [16], particle swarm optimization (PSO) [17], [18], simulated annealing [19], [20], artificial immune algorithm [21], tabu search [8] and the hybridization of these methods [22]–[26].

HW/SW partitioning is a problem of constraint optimization. No matter what kind of heuristic is adopted, handling with infeasible solutions is an important issue. For example, in [7], genetic algorithm was outperformed by a problem-specific method. The main reason is the genetic algorithm did not consider the problem-specific issues. Therefore, how to combine genetic algorithm with problem-specific exploitation is the key.

However, a genetic algorithm with problem-specific exploitation is time-consuming. On the one hand, multi-core CPU and many-core GPU have become the mainstream of

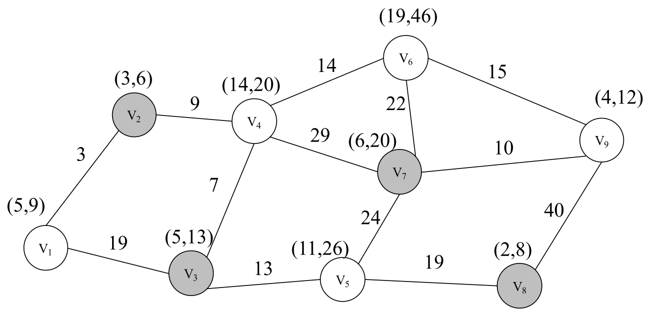


FIGURE 1. An example of task graph for HW/SW partitioning. White nodes are implemented on software and gray nodes are implemented on hardware.

personal computer systems as cheap alternate of high performance computing. On the other hand, how to leverage the computing resources of both multi-core CPU and many-core GPU for HW/SW partitioning and to reduce the run-time of genetic algorithm with problem-specific exploitation are unknown.

This paper presents a novel parallel genetic algorithm on CPU-GPU with dispersion correction for HW/SW partitioning. Firstly, a novel genetic algorithm for constraint optimization is presented. The under-constraint individuals are marched into feasible region step by step. Therefore, the search intensity can be enhanced as well as the constraint problem can be handled. Secondly, because a number of infeasible individuals are needed to be corrected and the computation overhead is high, we devise a parallel strategy and asynchronous transfer pattern to match well the architecture of multi-core CPU and many-core GPU. Finally, the experiments testify the effectiveness of our method.

The rest of this article is organized as follows. In section 2, we formulize the problem. In section 3, we firstly propose our method of serial implementation for HW/SW partitioning. We secondly propose the CPU-GPU parallel strategy. In section 4, computational experiments are conducted to evaluate the effectiveness of proposed method and the efficiency of hybrid parallel strategy. Conclusion and future work are given in the final section.

II. RELATED WORK

A. HW/SW PARTITIONING MODEL

Formally, the application to be partitioned is represented as an undirected graph G (V, E), s, h: V $\rightarrow$ R+, and c: E $\rightarrow$ R+. V $=$ $\{v_1, v_2, \ldots, v_n\}$ indicates the task nodes. Each node includes hardware cost $h(v_i)$ and software cost $s(v_i)$. E indicates the set of edges between the nodes. The weights $c(v_i, v_j)$ on the edges indicate the communication cost when the two adjacent nodes are separated implemented (hardware or software). $P = \{V_H, V_S\}$ is called a HW/SW partitioning, if it satisfies $V_H \cap V_S = \Phi$ and $V_H \cup V_S = V$. Accordingly, the edge set of P is defined as $Ep = \{(v_i, v_j)|v_i \in V_H, v_j \in V_S \text{ or } v_i \in V_S, v_j \in V_H\}$. Figure 1 gives an example of task graph for HW/SW partitioning.

As in [7] described, a partition is characterized by three metrics, namely hardware cost H_P, software cost S_P, and communication cost C_P, which are formulated as follows.

$$\mathrm{H}_P = \sum_{v_i \in V_H} h_i \tag{1}$$

$$\mathrm{S}_P = \sum_{v_i \in V_S} s_i \tag{2}$$

$$\mathrm{C}_P = \sum_{(v_i, v_j) \in E_P} \mathrm{c}\left(v_i, v_j\right) \tag{3}$$

The total cost of partition P is defined as $\mathrm{T}_P = \alpha \mathrm{H}_P + \beta \mathrm{S}_P + \gamma \mathrm{C}_P$. In T_P, α, β, γ are weights, which are non-negative constants, reflecting the relative importance of the three costs. Hence, two versions of partitioning problem are defined as.

Problem $\boldsymbol{P_0}$. Given a graph G with the cost function s, h, c and the constant α, β, $\gamma \geq 0$, finding a hardware/software partitioning P with minimum T_P.

Problem $\boldsymbol{P}$. Given a graph G with the cost function s, h, c and $\mathrm{R} \geq 0$. Finding a hardware/software partitioning P with $\mathrm{S}_\mathrm{P} + \mathrm{C}_\mathrm{P} \leq \mathrm{R}$ that minimize H_P.

The Problem $\boldsymbol{P_0}$ can be solved in polynomial-time, and the Problem P is NP-hard. Therefore, many challenging methods for HW/SW partitioning focus on Problem $\boldsymbol{P}$ [7], [27]–[31].

B. EXISTING METHODS FOR PROBLEM P

Problem $\boldsymbol{P}$ can be directly solved or indirectly solved with different algorithms.

Among direct methods [4], genetic algorithm was firstly proposed [7]. However, the solution quality and run-time of this method is not good.

In order to address the deficiencies of above method, an indirect method based on 2D search was proposed to search the solution space of $\boldsymbol{P}$. The search process of $\boldsymbol{P}$ was guided by the solution of Problem $\boldsymbol{P_0}$ [7]. To search the 2D space more precisely and efficiently, Tahaee and Jahangir [27] analyzed the search space and improved the search process.

The second indirect methods tried to transform Problem $\boldsymbol{P}$ into a variation of standard 0-1 knapsack problem. Specially, Wu *et al.* [28] proposed three new algorithms, named *1D search method* to obtain better quality in shorter execution time, by comparing against the previous 2D search method. However, on a variation of the standard 0-1 knapsack problem, there was a defect in the theoretical description, which was checked and perfected by Quan *et al.* [31].

Instead of a variation of standard 0-1 knapsack problem, Wu *et al* [29] proposed HEUR method. He treated problem P as a standard 0-1 knapsack problem without consideration of communication cost. The temporary solution without communication cost was adjusted into a feasible solution by considering the impact of the communication cost. Experiments showed that Wang's method could produce better solution quality than *1D search method* [28].

After introducing the idea of PageRank, Chen *et al.* [30] proposed NodeRank algorithm, an iteration-based HEUR, to further improve the solution quality in some cases.

In a short, there was no one method of absolute advantage in both quality and run-time for problem $\boldsymbol{P}$. Therefore, this manuscript proposes a novel approach to solve problem $\boldsymbol{P}$.

-- We combine the advantages of direct method and indirect method. Firstly, we search the solution space of problem P using genetic algorithm. Secondly, we also adopt the correction idea from the second indirect methods for problem-specific constraint issue.
-- Especially, our correction strategy is different from that of existing methods. In existing methods, such as in [29] and [30], only one infeasible solution is corrected. In our method, a number of infeasible solutions are corrected. In this way, our strategy can enhance the search intensity and eventually improve the solution quality as well as address the constraint issue.
-- Furthermore, a novel parallel pattern for our correction strategies is proposed to match well the architecture of CPU/GPU hardware and eventually to reduce the run-time of about steps, otherwise it would be time-consuming.

The experiments testify the effectiveness of our method.

C. ARCHITECTURE AND PROGRAMMING OF MULTI-CORE CPU AND MANY-CORE GPU

On the one hand, there are some pioneering publications on parallel method for other HW/SW partitioning. The known works include parallel genetic algorithm [32] and PSO algorithm [33]. Both works were implemented on the cluster platforms.

On the other hand, multi-core CPU and many-core GPU has become a popular parallel computing platform with low power-consuming and high ratio of performance to price. Multi-core CPU and many-core GPU are available on very general PC systems. They are playing an important role in science and engineering domains [34], [35]. Therefore, how to use the parallel power of multi-core CPU and many-core GPU on personal computers for HW/SW partitioning is an interesting topic. The manuscript proposes a new CPU-GPU parallel genetic algorithm with dispersion correction for HW/SW partitioning.

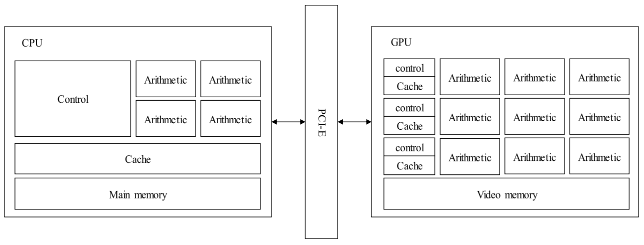


FIGURE 2. Difference of architecture between CPU and GPU. Data transfer between CPU and GPU is through PCI-E.

The architecture comparison between CPU and GPU is shown in Figure 2. The architecture of CPU is latency-oriented. When there are complex logic judgements in application programs, the architecture of CPU shows a significant advantage over that of GPU. The architecture of GPU is throughput-oriented [34]. When there are computing-intensive parts in application programs, the architecture of GPU shows more advantages over that of CPU. The main reason of the difference is that in CPU, logic control units and cache take up most of space, while in a single chip of GPU, arithmetic units take up most of area.

At present, there are various parallel programming languages supporting multi-core CPU and many-core GPU. This paper focus on OpenMP and Compute Unified Device Architecture (CUDA).

-- OpenMP is an industry standard for parallel programming of shared memory system, which uses Fork/Join parallel execution model. It provides a simple and easy-to-use mechanism of multi-threading on multi-core CPU. When parallelizing a serial program in OpenMP, there is no need to make a big change to resource code. Adding a simple directive statement before the loop body is enough to unroll the loop.

-- CUDA is a programming model to provide a programming interface to GPU devices. It is released by NVIDIA in the purpose of popularizing GPU computing. The functions writing in CUDA C and running on GPU are kernels. All the threads running the same kernel are organized into a thread grid. In a thread grid, all of the threads execute the same kernel on different data, which is known as *Single Program Multi Data* (SPMD). The threads in a grid are divided into equally sized thread blocks and dispatched on the *Stream Multi-Processors* (SMs) of the GPU in a cyclic manner. Within a thread block, the continuous 32 threads are organized into a warp and the warps are dispatched by the warp dispatcher on the SM. This way hides the latency result from memory accessing because there are other idle warps waiting for being executed within a thread block.

-- In PC platforms, GPU is an independent computing device. The data between CPU and GPU is transferred through PCI-E bus. Therefore, an efficient algorithm should minimize the transfer overhead between GPU and CPU.

In this manuscript, we proposed a novel GPU-CPU parallel pattern to support genetic algorithm with dispersion correction for HW/SW partitioning.

III. THE PROPOSED METHOD

A. OVERVIEW OF PROPOSED PARALLEL GENETIC ALGORITHM WITH DISPERSION CORRECTION

As a population-based metaheuristic, standard genetic algorithm is based on the principle of natural genetics and natural selection. The core elements in the genetic search include reproduction, crossover and mutation.

In our method, these core elements are adapted for HW/SW partitioning. We stop our algorithm when a given max number of generation is reached. In addition, if the global hardware cost was not improved in a given number of iterations, we stop the algorithm as well. If the global hardware cost is improved, we reset the no-improvement counter as 0.

In overview, the proposed genetic algorithm with dispersion correction for HW/SW partitioning is shown in Figure 3.

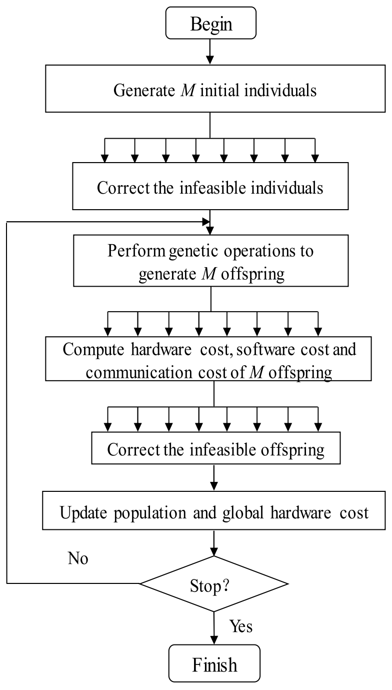

Begin
Generate *M* initial individuals
Correct the infeasible individuals
Perform genetic operations to generate *M* offspring
Compute hardware cost, software cost and communication cost of *M* offspring
Correct the infeasible offspring
Update population and global hardware cost
No
Stop?
Yes
Finish

FIGURE 3. Flowchart of proposed method.

B. PROPOSED GENETIC ALGORITHM WITH DISPERSION CORRECTION STRATEGY

1) INDIVIDUAL REPRESENTATION

In standard genetic algorithm, an initial codification of individuals is the starting point to the following steps. For HW/SW partitioning, let $\boldsymbol{x}$ be $(x_1, x_2, \ldots, x_n)$. Here, $\boldsymbol{x}$ denotes a HW/SW partitioning of problem $\boldsymbol{P}$. $x_i = 1$ ($x_i = 0$) indicates that the node v_i is assigned to software (hardware), $1 \leq i \leq$ n. Hence, an individual is an expression of 01…01. The length of expression equals to number of nodes in the task graph. Figure 4 shows an example of randomized individuals.

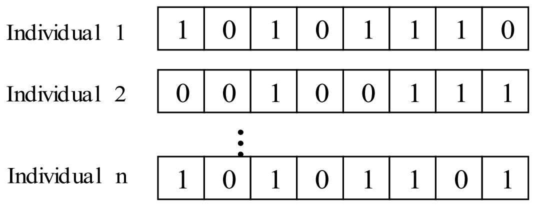


FIGURE 4. An initialized population for HW/SW partitioning.

In our method, we initialize one individual by the solution of NODERANK [30]. The remaining individuals are generated by randomly flipping two genes in the first individual. Comparing with random population, this ensures the initial population of high quality.

2) OBJECTIVE FUNCTION

The goal of problem ***P*** is to minimize the hardware cost while satisfying the constraint of software cost plus communication cost. Here, h_i denotes the hardware cost of v_i and s_i denotes the software cost of v_i. C($\boldsymbol{x}$) indicates the communication cost in which c_ij denotes the communication cost between v_i and v_j if the two nodes are in different context. Hence, the hardware cost, software cost and communication cost of problem ***P*** are formulized as.

$$\mathrm{H}(\boldsymbol{x}) = \sum_{i=1}^{n} h_i (1 - x_i) \tag{4}$$

$$\mathrm{S}(\boldsymbol{x}) = \sum_{i=1}^{n} s_i \cdot x_i \tag{5}$$

$$\mathrm{C}(\boldsymbol{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left| x_i - x_j \right| \tag{6}$$

According to definition, HW/SW partitioning is formulized as.

$$P \begin{cases} \min H(\boldsymbol{x}) \\ S(\boldsymbol{x}) + C(\boldsymbol{x}) \le R \end{cases}$$

The inequality in ***P*** is extended by the equation (4) to (6) as.

$$\sum_{i=1}^{n} s_i \cdot x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left| x_i - x_j \right| \le R \tag{7}$$

In summary, the final formulation of problem ***P*** is as follows.

$$P \begin{cases} \min \sum_{i=1}^{n} h_i (1 - x_i) \\ \sum_{i=1}^{n} s_i \cdot x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left| x_i - x_j \right| \le R \end{cases}$$

Although the costs of randomized individuals are obtained by the above formulas, it is necessary to further judge whether each individual satisfies the constraint. If it does, the individual is a feasible HW/SW partitioning. Otherwise, it is an infeasible partitioning. The following section will describe the method we propose for handling with the infeasible individuals.

3) DISPERSION CORRECTION STRATEGY

In evolutionary computation, how to solve constrained optimization problems is an important research topic. Among different techniques, penalty function-based method is very practical [36]. For HW/SW partitioning, it is also a concern to find an effective penalty function to handle with these infeasible individuals [6], [24]. However, penalty function-based method does not change the nature of infeasible individuals. In our problem, when the number of infeasible individuals is much more than that of feasible solutions, whether such method is still effective is doubtful. Although function-based method can search for the solution by enhance the diversity of search process, it fails to enhance the intensification of search process.

In this paper, we propose a dispersion correction method, which involves two procedures. Specially, when some individual does not satisfy the constraint, it means one of the tasks assigned to software component is needed to be switched to implementing on hardware component. That is to say, setting x_i as 0. The process of correction involves multiple nodes assigned to software component. The goal of correction is to make under-constraint individuals feasible.

When node i is assigned to software component, the sum of communication cost of all its adjacent nodes assigned to hardware is given as

$$C_{old} = \sum_{j \in V_H} c(i, j) \tag{8}$$

If node i is corrected, the sum of communication cost of its adjacent nodes is changed as well. After it is changed to implementing on hardware component, there will be communication cost between all its adjacent nodes assigned to software component. The sum of communication cost of v_i becomes as.

$$C_{new} = \sum_{k \in V_s} c(i, k) \tag{9}$$

Let Δc_i denotes the change of communication cost of node i. It is obtained by the following formula.

$$\Delta c_i = C_{old} - C_{new} \tag{10}$$

Combining Δc_i with the hardware cost and software cost of node i, we select the node assigned to software component with the minimum ratio and set it as 0 (hardware component). At each step, only one node is corrected and the hardware cost H($\boldsymbol{x}$), software cost S($\boldsymbol{x}$) and communication cost C($\boldsymbol{x}$) are updated. The procedure does not stop until the HW/SW partitioning satisfies the constraint, as in algorithm1 shows.

Algorithm 1 One-Step Correction

Input: Task graph G, constraint R and HW/SW partitioning $\boldsymbol{x}$
Output: corrected HW/SW partitioning $\boldsymbol{x}$

```
1. while $S(\boldsymbol{x}) + \mathrm{C}(\boldsymbol{x}) > R$
2.   for each $x_i = 1$ do
3.     compute $\Delta c_i$ and $h_i/(s_i + \Delta c_i)$
4.     choose node $k$ with minimum $h_i/(s_i + \Delta c_i)$
5.   end for
6.   set $x_k$ as 0
7.   update $H(\boldsymbol{x})$, $S(\boldsymbol{x})$, $C(\boldsymbol{x})$
8. end while
```

After the first correction, the under-constraint individual becomes feasible. In order to optimize its solution, it is further corrected by setting the nodes assigned to hardware component as the one implemented on software component.

The process is called intensification correction. Likewise, the process involves the change of communication cost. Similar to equation (8) to equation (10), when the node v_i assigned to hardware component is changed into implementing on software component, the change of communication cost of node v_i is given as.

$$\Delta c_i = \sum_{k \in V_s} c(i,k) - \sum_{j \in V_H} c(i,j) \tag{11}$$

Although the intensification correction involves selecting node of minimum value ratio, it is different from the first correction. After the first correction, by constraint R subtracting the sum of software cost and communication cost of feasible individual, there exists a residual value. The process of intensification correction is based on that residual value. Therefore, the process of intensification correction does not stop until the update of residual is less than 0. Besides, we select the node of maximum ratio value to be implemented on software component, just as Algorithm 2 shows.

Algorithm 2 Intensification Correction

```
Input: task graph G, constraint R and HW/SW partitioning
$x$ in the first correction
Output: HW/SW partitioning $x$ of two-step correction

1. $residual = R - (S(x) + C(x))$
2. while $residual \geq 0$
3.     for each $x_i = 0$ do
4.         assume $x_i = 1$
5.         compute $\Delta c_i$ and $h_i/(s_i + \Delta c_i)$
6.         choose node $k$ with maximum $h_i/(s_i + \Delta c_i)$ in
       the condition of $(\Delta c_i + h_i/(s_i + \Delta c_i)) \leq residual$
7.     end for
8.     if exists node $k$ then
9.         $x_k = 1$
10.        $residual = residual - (s_k + \Delta c_k)$
11.        update $H(x)$, $S(x)$, $C(x)$
12.    else
13.        break
14. end if
15. end while
```

In algorithm 1, line 3 and line 4 are implemented in O (n). In line 7, H(x), S(x) can be implemented in O (1) and C(x) is implemented in O (m_i), in which m_i denotes the number of edges associated with node i. At the worst case, m_1 equals to that of edges in the task graph, namely m. Therefore, the time complexity of algorithm 1 is O $(k_1 * (n + m))$, in which k_1 denotes the while loop in line 1. Similarly, the time complexity of algorithm 2 is O $(k_2 * (n + m))$, in which k_2 denotes the while loop in line 2. Let k be max $\{k1, k2\}$, the time complexity of dispersion correction strategy is O $(k * (n+m))$.

4) GENETIC OPERATIONS FOR HW/SW PARTITIONING

At each generation, standard genetic algorithm selects two individuals in the current population as parents. The algorithm either passes these parents directly to the new population or generate two offspring individuals from the parents. Hence, how to select the two individuals from current population is an important issue. Among different selection strategies, roulette wheel selection is a general option. It is a probability-based approach in which the individuals with higher fitness have more priority. However, roulette wheel selection can not handle a minimization problem directly [37].

In our problem, the goal is minimizing the hardware cost. For HW/SW partitioning, the smaller the hardware cost, the more the saved hardware cost. Based on the fact, we adapt the roulette wheel selection based on saved hardware cost. The individual with more saved hardware cost has higher probability to be selected. Meanwhile, the probabilistic characteristics of roulette wheel selection makes individuals with small saved hardware cost have a chance to be selected. However, it is possible for the same individual to be selected again as a parent individual if directly using roulette wheel selection twice.

To overcome the shortcoming, we propose a random sampling and combine it with roulette wheel selection. Specially, we still use the roulette wheel selection to select the first parent individual and the first parent individual's position is obtained. Next, we set it as a start point and generate a random offset to select the second parent individual. The modular operation keeps the result within the range of population. This random sampling ensures that the two selected parents are always different. The formula of random sampling is given as follows.

$$pos_2 = \left(pos_1 + rand\,(1, pop\_size - 1)\right) mod\ pop\_size \tag{12}$$

To generate the new individuals, standard genetic algorithm generates offspring individuals either by crossover operation to combine the vector entries of a pair of parents or by mutation operation to make random changes to a single parent.

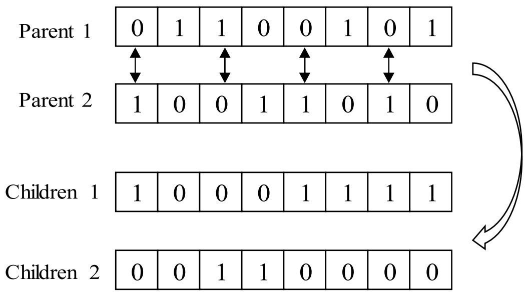


FIGURE 5. Crossover based on component.

In our work, we perform the crossover operation as follows. For each component in the parent individuals, every two of them are swapped. After that, two new individuals are generated, as Figure 5 shows.

Standard genetic algorithm further increases the diversity of population by mutation operation. In HW/SW partitioning, we perform the mutation on some component of offspring. Whether performing the mutation or not depends on a given probability. After mutation operation, the nodes assigned to hardware component is changed into software component, and vise versa.

5) UPDATE STRATEGY

After the genetic operations, a new population is created. It is noteworthy that genetic operation only generate the solution vectors of individuals. Next, the hardware cost, software cost and communication cost of each new individual are obtained according to formula (4)∼(6). Besides, the process of genetic operations does not check whether the new individuals satisfy the constraint or not, resulting in under-constraint individuals in the new population. Dispersion correction mentioned before is utilized to make the infeasible solutions feasible.

Finally, the old population is replaced with the new population. Meanwhile, an individual with the optimal hardware cost is obtained. If its hardware cost is better than the global hardware cost, then updating the global hardware cost.

6) ALGORITHM COMPLEXITY

At each iteration, for each individual, the complexities of computing costs is O(n^2), which means the worst case is dominated by computing the communication cost. At the stage of dispersion correction, the time complexity is O ($k * (n + m)$). The time complexity of genetic operation is O ($\mathrm{M} * n$), in which M denotes the individual number. Therefore, the overall time complexity at each iteration is O ($\mathrm{M} * (n^2 + k * (n + m))$).

C. CPU-GPU PARALLEL STRATEGY

1) HIGHLY EFFICIENT PARALLEL APPROACH FOR ENHANCED GENETIC ALGORITHM

On the one hand, modern personal computer systems consist of multi-core CPU and many-core GPU, which form a heterogeneous computing platform. How to combine advantages of both CPU and GPU to solve real-world applications is challenging [38].

On the other hand, most of exiting methods for HW/SW partitioning focus on sequential implementations which do not leverage the available computing resources on personal computer systems. There are scarce publications on parallel method for HW/SW partitioning. A few works include parallel genetic algorithm [32] and parallel PSO algorithm for HW/SW partitioning [33], which were implemented on the cluster platform.

Based on above enhanced genetic algorithm with dispersion correction for HW/SW partitioning, this section devises a highly efficient parallel approach for the algorithm, which addresses several key issues. The first one is how to parallelize the algorithm components. The second one is how to map these algorithm components on CPU-GPU. The mapping should match well the architecture of CPU-GPU. The third one is how to transfer the data between CPU and GPU efficiently.

Therefore, we present a parallel genetic algorithm with dispersion correction for HW/SW partitioning, in which the individual processing are run in parallel on CPU and GPU.

Especially, a number of under-constraint individuals are corrected in an irregular way, in which the computation process is complicated and the computation overhead is large. We present a CPU-GPU parallel strategy to take the advantage of the parallel power of both multi-core CPU and many-core GPU. The proposed strategy computes the costs of each individual in parallel on GPU and corrects the under-constraint individuals in parallel on multi-core CPU as follows.

2) COMPUTING INDIVIDUAL'S COSTS IN A WAY OF DATA-PARALLEL ON GPU

In our method, the GPU will be utilized to compute each individuals' hardware cost, software cost and communication cost. In order to perform the roulette wheel selection, GPU will also be used calculate the saved hardware cost of each individual.

Although the thread running on GPU is the basic unit and one thread can be mapped to one individual logically. However, this configuration has following drawbacks.

- The number of individual in genetic algorithm is generally less than that of arithmetic units on modern GPU. Therefore, this configuration does not leverage the available computing resources of GPU.
- Furthermore, this configuration fails to make fully use of the data-parallel characteristics in the process of computing each individual's costs.

Equation (4) to (6) show that the process of hardware cost, software cost and communication cost mainly involves summation operation in GPU. It is the first choice to implement the operation with GPU reduction in data-parallel pattern. However, due to the peculiarity of HW/SW partitioning problem, this reduction process is different from standard reduction.

Therefore, the computation method of hardware cost is proposed as follows.

- According to the partitioning of each node, the intra-block threads firstly perform the product of hardware cost of nodes in the task graph and the hardware/software partitioning of individuals in parallel.
- After the product, each intra-block thread performs the partial summation in parallel. This is different from the standard reduction in which the input data is directly used to perform the partial summation.
- Next, the partial summations are uploaded in the shared memory. The finally summations are finished by the intra-block threads in a cooperative way. Computing software cost and saved hardware cost follow the same way.

-- After the summation, the hardware cost and software costs are written into global memory. The saved hardware costs are also written into global memory.

In our method, a thread block is mapped to an individual and the intra-block threads are mapped to the genes of an individual. For HW/SW partitioning, a gene of individual represents the partition of a node in the task graph.

Figure 6 shows the difference between the process of standard reduction and that of computing hardware cost in context of HW/SW partitioning.

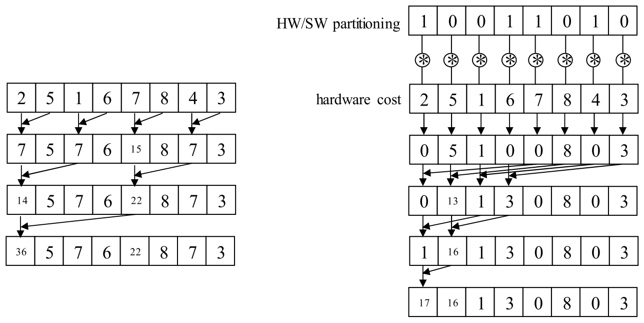


FIGURE 6. The difference between the process of standard reduction and that of computing hardware cost for HW/SW partitioning.

For computing the software cost and the saved hardware cost, they follow the same way as the computation of the hardware cost.

However, computing communication cost is quite different from above computation methods.

For each individual, the efficiency of computing the communication cost depends on the representation of task graph on GPU. It is noteworthy that the weight on each edge denotes the communication cost between a pair of nodes if they are in different context. In the original formula (6), the task graph is represented as an adjacent matrix in which the complexity of sequential procedure of communication cost is $O(n^2)$.

Therefore, our method of computing communication cost is as follows.

-- When computing communication cost of each individual, only the information of edge in the task graph is necessary. Therefore, the complexity of original sequential procedure is reduced to $O(m)$ in our method.
-- When porting the process of communication cost to GPU, we configure a thread block as an individual. Furthermore, different from above computing of hardware cost, a thread within a block is mapped to an edge of task graph.
-- Because two nodes of each edge is accessed by their own threads, the data-placement of solution vector is located on shared memory. Likewise, the summation of each edge is finished by the intra-block cooperative threads.

Figure 7 shows the process of computing communication cost on GPU.

In summary, this section propose a data-parallel method to calculate the four types of costs. When computing the costs of each individual in parallel on GPU, our strategy exhibits three levels of parallelism. These are the parallelism between the new individuals, the parallelism among the nodes in each individual and the parallelism among the edges in each individual, respectively.

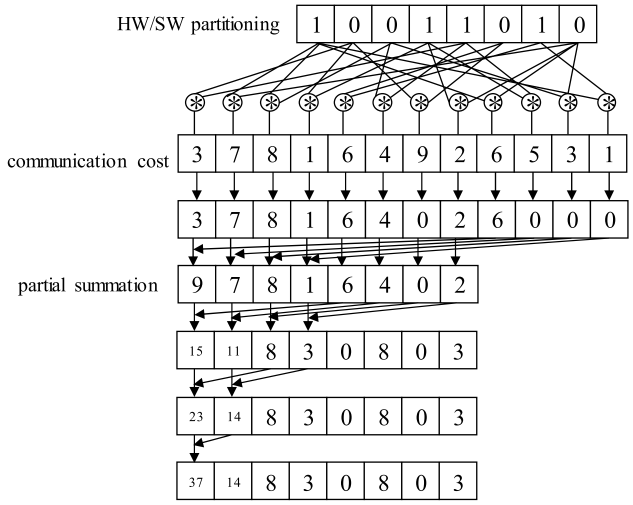


FIGURE 7. The process of computing communication cost on GPU.

In implementation, we assign two kernels running on GPU. The first kernel is to compute each individuals' hardware cost, software cost and saved hardware cost. The second kernel is to compute each individual's communication cost.

3) MULTI-CORE CPU BASED DISPERSION CORRECTION OF INFEASIBLE INDIVIDUALS

After each new individual obtains the hardware cost, software cost and communication cost, the feasibility of individual will be checked based on constraint R. If an individual does not satisfy the constraint, a dispersion correction strategy will be invoked.

However, the procedures of dispersion correction inside *Algorithm1* and *Algorithm2* are extremely irregular.

-- Firstly, at each iteration, when the two algorithms stop depend on how far the under-constraint individuals is away from the constraint.
-- Secondly, for example of *Algorithm1*, line 2 to line 5 are the main procedures, but the number of loop depends on the number of nodes assigned to software.
-- Lastly, in order to correctly realize the two algorithms, two data structures of managing the partitioned nodes are created and the data structures are accessed frequently in the whole procedure. The same data structures are also used in the *Algorithm2*.

Therefore, this section proposes a multi-start strategy of parallel dispersion correction on multi-core CPU for our algorithm with following reasons.

-- The irregular dispersion correction processes inside each individual is more suitable for CPU than for GPU.
-- Furthermore, for overall individuals, the processes of dispersion correction are independent of each other,

meaning that they can be parallelized. Multi-start of correcting the under-constraint individuals can be accelerated by multi-core CPU.

4) AN ASYNCHRONOUS TRANSFER PATTERN FOR REDUCING THE TRANSFER OVERHEAD

At each iteration, solution vector of each individual is transferred from CPU to GPU. After computing costs, the results are transferred back to CPU side. Hence, the transfer overhead is unavoidable.

In order to further improve the efficiency of proposed algorithm, we proposed an asynchronous transfer pattern for CPU-GPU computation in which the processes of data transfer and cost computation run in a way of pipeline. This way can minimize the transfer overhead. Figure 8 shows an example in which the number of individuals is 4 and number of stream is 2 respectively.

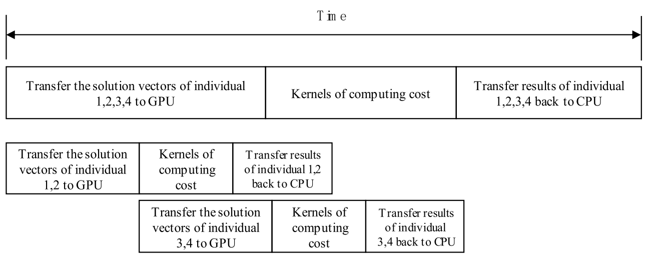

Time
Transfer the solution vectors of individual 1,2,3,4 to GPU
Kernels of computing cost
Transfer results of individual 1,2,3,4 back to CPU
Transfer the solution vectors of individual 1,2 to GPU
Kernels of computing cost
Transfer results of individual 1,2 back to CPU
Transfer the solution vectors of individual 3,4 to GPU
Kernels of computing cost
Transfer results of individual 3,4 back to CPU

FIGURE 8. Asynchronous transfer pattern between data transfer and kernel execution.

In summary, the proposed flowchart of CPU-GPU parallel genetic algorithm with dispersion correction strategy for HW/SW partitioning is shown in Figure 9.

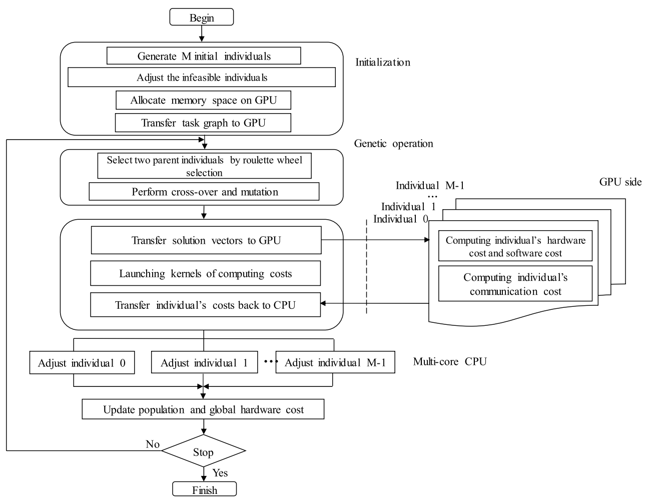

Begin
Generate M initial individuals
Adjust the infeasible individuals
Allocate memory space on GPU
Transfer task graph to GPU
Initialization
Genetic operation
Select two parent individuals by roulette wheel selection
Perform cross-over and mutation
Individual M-1
Individual 1
Individual 0
GPU side
Transfer solution vectors to GPU
Launching kernels of computing costs
Transfer individual's costs back to CPU
Computing individual's hardware cost and software cost
Computing individual's communication cost
Adjust individual 0
Adjust individual 1
Adjust individual M-1
Multi-core CPU
Update population and global hardware cost
No
Stop
Yes
Finish

FIGURE 9. Flow chart of CPU-GPU parallel genetic algorithm with dispersion correction for HW/SW partitioning.

IV. EXPERIMENT

A. BENCHMARK AND PLATFORM

Because the proposed method is population-based metaheuristic, empirical evaluation is widely used to test the performance and effectiveness in many researches [7], [27]–[30]. So does in this manuscript.

We also follow the same benchmark according to the method mentioned in [7] and [28]–[30]. TABLE 1 shows the number of node n and that of edge m in each task graph respectively, the total size is given by $2 \times n + 3 \times m$. 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).

TABLE 1. Benchmark.

no.	name	n	m	size	description
1	crc32	25	34	152	32-bit cyclic redundancy check. From the telecommunications category of MiBench [39]
2	patricia	21	50	192	Routine to insert values into patricia tries, which are used to store routing tables. From the Network category of MiBench [39]
3	dijkstra	26	71	265	Computes shortest paths in a graph. From the Network category of MiBench [39]
4	clustering	150	333	1299	Image segmentation algorithm in a medical application
5	rc6	329	448	2002	RC6 cryptographic algorithm
6	random1	1000	1000	5000	random graph
7	random2	1000	2000	8000	random graph
8	random3	1000	3000	11000	random graph
9	random4	1500	1500	7500	random graph
10	random5	1500	3000	12000	random graph
11	random6	1500	4500	16500	random graph
12	random7	2000	2000	10000	random graph
13	random8	2000	4000	16000	random graph
14	random9	2000	6000	22000	random graph

The software cost is generated as uniform random numbers from the interval [1,100]. The hardware cost is generated as random numbers from a normal distribution with expected value $k \cdot s_i$ and a given standard deviation $k \cdot \lambda \cdot s_i$, where s_i is the software cost of the given node. The value of k denotes the choice of units for software and hardware cost. The value of λdenotes the correlation between a node's hardware cost and software cost. In [7], [28], and [29], the author pointed out that the value of k has no algorithmic implications and that of λdid not have any impact on the performance of the algorithm.

The communication costs were generated as uniform random numbers from the interval $[0, 2 \cdot \rho \cdot s_{max}]$, where s_{max} is the highest software cost. Thus, communicat-ion cost have an expected value of $\rho \cdot s_{max}$, and ρ is the so-called communication to computation ratio (CCR). ρ was taken as 0.1, 1 and 10, corresponding to computation-intensive

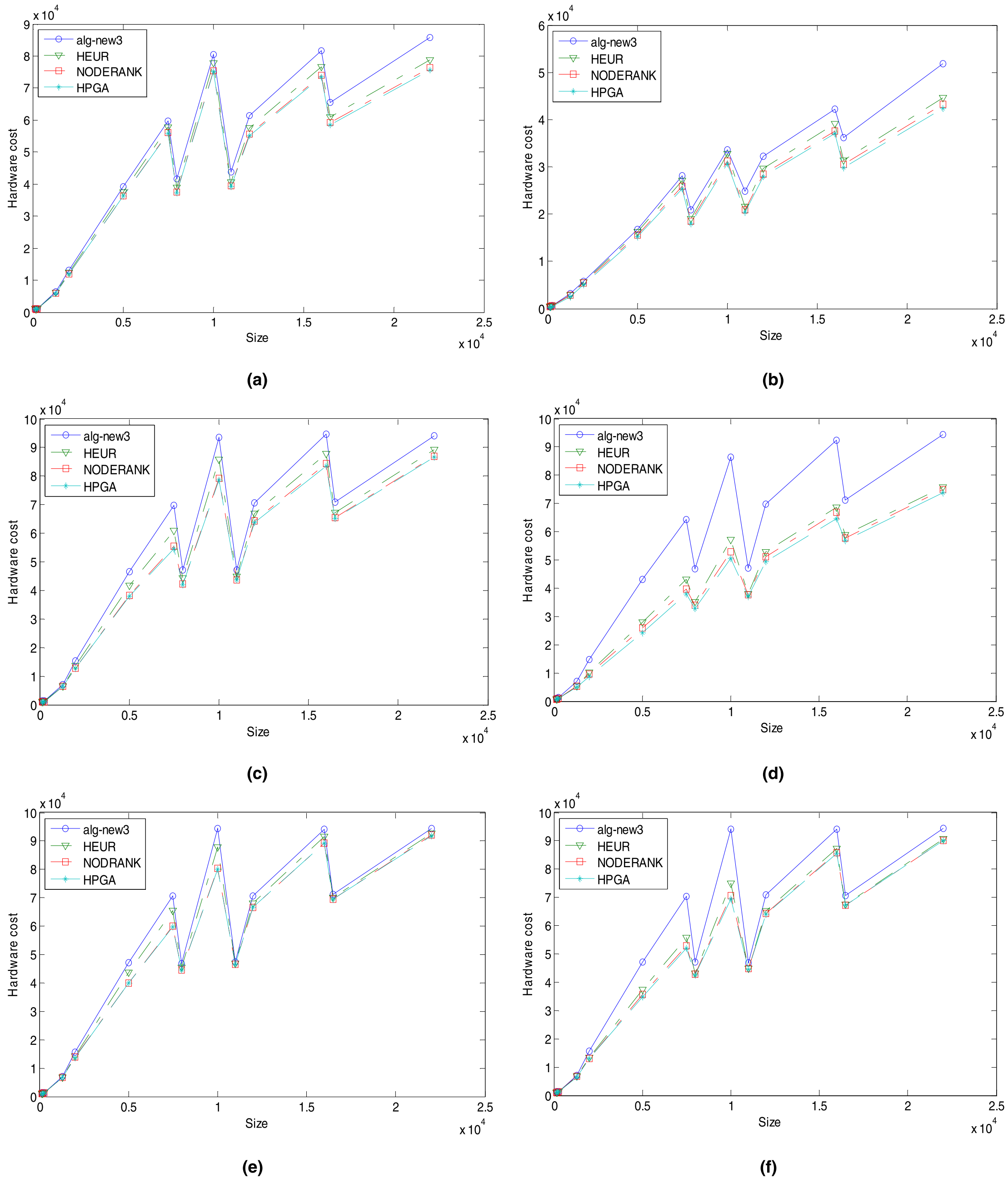


FIGURE 10. **Solution quality averaged on 30 random instances in the 6 cases. (a) CCR = 0.1, R = low. (b) CCR = 0.1, R = high. (c) CCR = 1.0, R = low. (d) CCR = 1.0, R = high.**

case, intermediate case, and communication-intensive case, respectively.

R was randomly generated as a uniform random number (1) from the interval $[0, 1/2\times \sum s_i]$, corresponding to the strict real-time constraints, (2) from the interval $[1/2\times \sum s_i, \sum s_i]$, corresponding to the loose real-time constraint. The two cases are indicated as $R = low$ and $R = high$, respectively.

The development tool is Visual Studio2012, programming in C++. The computing platform running multi-core parallel procedure is Intel Pentium (R)Dual-Core CPU E5300 with 2.6GHz clock frequency that has four cores. The size of CPU main memory is 16GB. The computing platform running many-core parallel procedure is NVIDIA GTX 780, consisting of 12 SMs, 192 SPs per SM. The clock frequency of each SP is 1.059GHZ. The size of GPU global memory is 3GB. We implement our method in OpenMP and CUDA, respectively.

B. RESULT ANALYSIS

According the section 2, a convincing test should consider following reasons.

-- The proposed approach solve the Problem P. The compared algorihtms should be existing methods which also sovle Problem P.

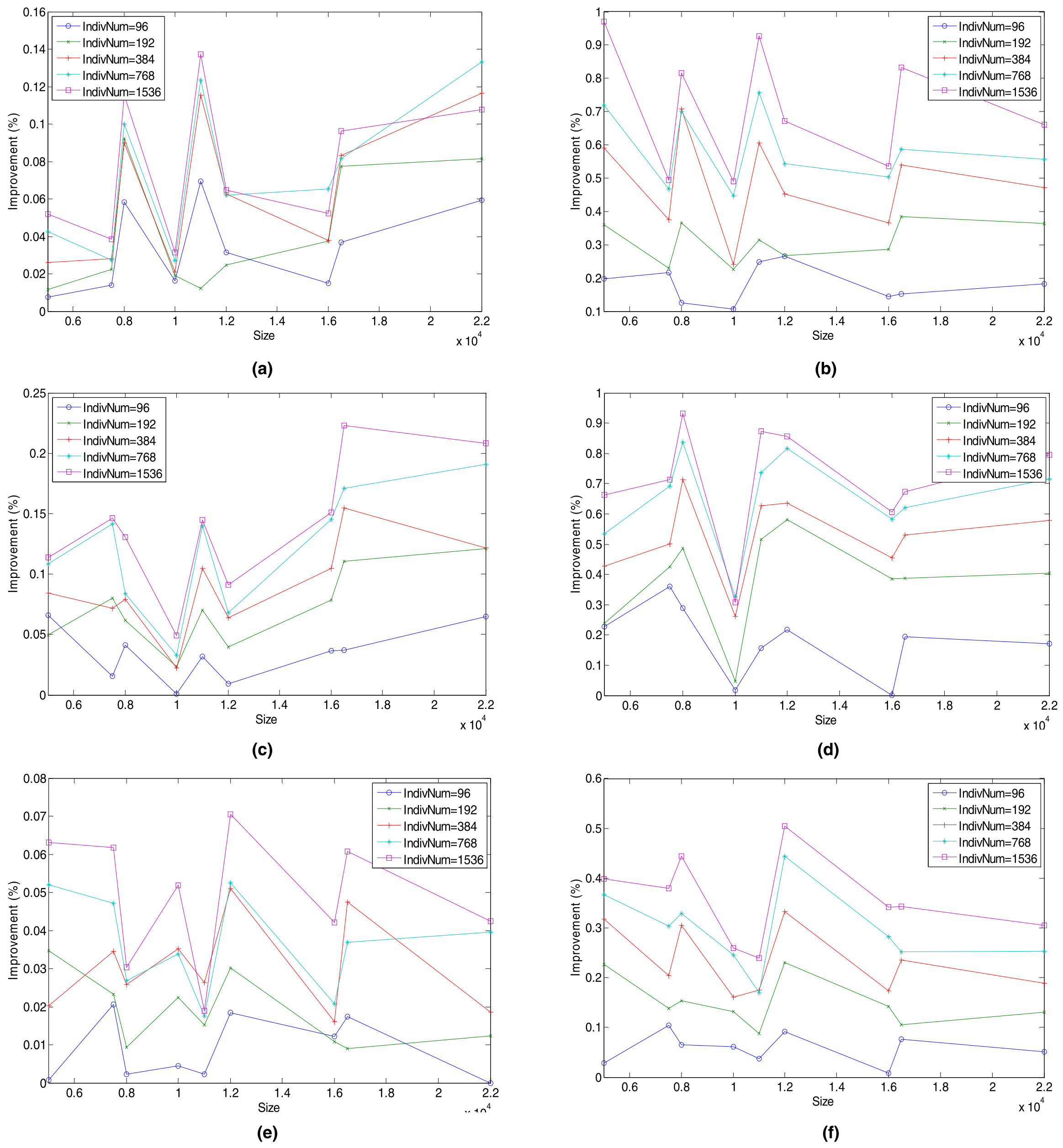


FIGURE 11. The improvement of solution quality by doubling the individual number step by step. (a) CCR = 0.1, R = low. (b) CCR = 0.1, R = high. (c) CCR = 1.0, R = low. (d) CCR = 1.0, R = high. (e) CCR = 10.0, R = low.

-- The proposed approach combine the advantages of direct methods and indirect methods. The existing indirect methods outperformed existing direct method in [6] and [7]. It is reasonable to compare with existing indirect methods.

According to the existing indirect methods for problem P, we compare the solution quality of our method with that of *Alg-new3* in [28], HEUR in [29] and NODERANK in [30]. For *Alg-new3*, the search solution space d_x was set as 20, the same as in [28]. For NODERANK, the iteration number is 4, the same as in [30]. We tested the proposed method in different values of CCR and constraint R.

Our proposed CPU-GPU parallel method is name as HPGA, while our method of serial inplementation is name as SGA. The parameters in our method is shown in TABLE 2.

TABLE 2. Parameters.

Parameter	Value
Size of population	48
Cross-over probability	0.8
Mutation probability	0.2
Max generation	200
No improvement count	50

Figure 10 shows the solution quality by the methods in the six cases. In our problem, minimizing the hardware cost is our goal. The smaller the quality, the better. The figure shows that the solution quality by our method is the best at most cases. Besides, since the existing methods is superior to the proposed genetic algorithm in [7], this indirectly reflects that our proposed HPGA is superior to it.

TABLE 3 shows the average improvement of each method over *alg-new3* on all benchmarks in six cases. Formally, the improvement of algorithm *A*over algorithm *B*is defined as

$$\text{improvement} = \frac{hw\_cost_A - hw\_cost_B}{hw\_cost_A} \times 100 \qquad (13)$$

TABLE 3. Average improvement over ALG-NEW3 IN 6 CASES

	CCR	R	HEUR	NODERANK	HPGA
case1	0.1	low	5.2	7.9	**8.6**
case2	0.1	high	5.3	11.3	**15.3**
case3	1	low	7.3	11.5	**11.6**
case4	1	high	24.4	26.6	**30**
case5	10	low	3.9	6.2	**7.2**
case6	10	high	9.7	11.1	**11.7**

In the table, the highest improvements are bolded. It can be seen that the improvements of our method are the highest in the six cases.

In our method, the individual number is fixed at 48 even in large task graphs such as *random*1 to *random* 9. The main reason is the hardware specifications of multi-core CPU and GPU are considered. Also, the run-time of our method is considered. In order to further testify the effect of individual number, we concentrate on solution quality in large task graphs by doubling the inidivdual number step by step. Let the solution quality of HPGA with 48 inidivduals be baseline, Figure 11 shows the improvements of different inidivdual number on radom1∼ random 9 in the 6 cases.

As the results show, increasing the individual number does not improve the solution quality significantly.

In our method, both multi-core CPU and GPU are utilized. To illustrate the efficiency more clearly, we need to test the speedups separately.

Firstly, for computing each individual's costs, two kernels are launched on GPU. Figure 12 shows their speedups between the two kernels and their corresponding sequential implementations. Kernel1 is computing each individual's hardware cost, software cost and the saved hardware cost and kernel2 is computing each individual's communication cost. On the whole, the efficiency is very high. When the task graph becomes large, the speedup becomes large as well. Especially, the speed up of kernel2 is more significant. The reason is kernel1 needs to compute three costs and involves much more global memory accesses to store these costs. It is noteworthy that in task graph 11, 13 and 14, the speedups of kernel2 grows no more. The reason is when the number of edge in the task graph is more than 4000, GPU is saturated.

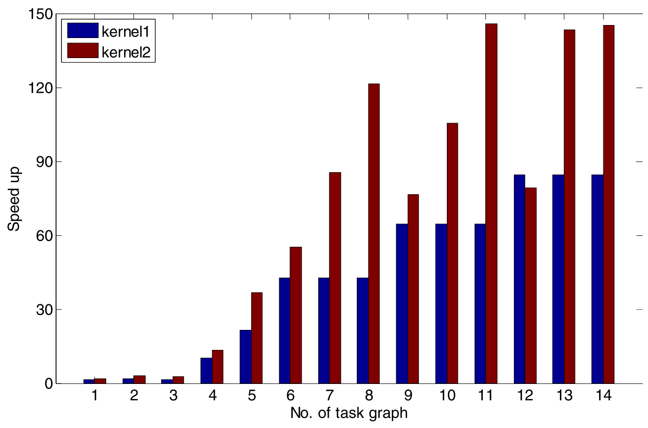


FIGURE 12. Speed-ups between two kernels on GPU and the corresponding sequential implementations.

Secondly, to improve the efficiency of dispersion correction strategy, each individual is parallelized on multi-core CPU. In our platform, the Hyper-Threading is utilized which means one physical core consist of two logical cores. The total number of logical core and that of physical core are 8 and 4, respectively. To analyze the parallel scalability of OpenMP implementation, let the runtime of dispersion correction strategy running on single logical core be baseline, the speedups are recorded by increasing the number of logical core till 8. In OpenMP, this is realized by setting the thread number from 1 to 8. Figure 13 shows the results by changing the number of logical core.

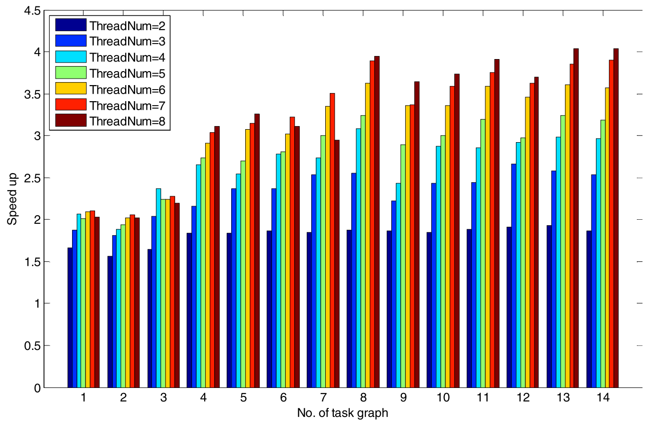


FIGURE 13. The parallel efficiency of OpenMP implementation of dispersion correction strategy.

As the figure shows, for the same task graph, increasing the number of threads leads to the increment of speedup. Meanwhile, when the number of threads is fixed, the larger the size of task graph, the higher the speedup.

After testing the efficency of GPU implementation and multi-core implementation seperately, the overall efficiency will be tested. In our method, the most time-consuming part is the process of genetic search for optimum solution. We divide the running time of sequential implementation into three

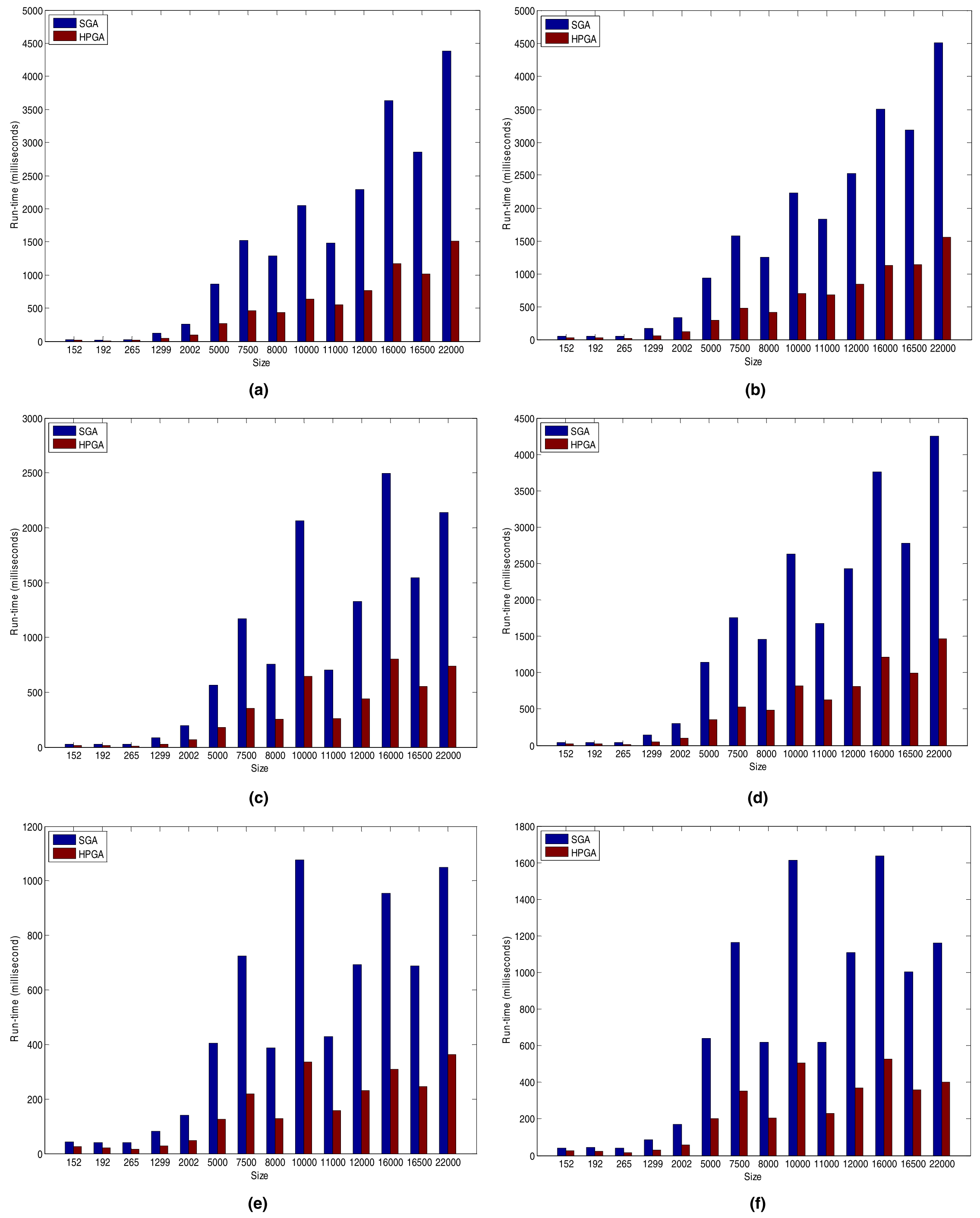


FIGURE 14. Runtime of proposed method between serial implementation and hybrid-parallel implementation. (a) CCR = 0.1, R = low. (b) CCR = 0.1, R = high. (c) CCR = 1.0, R = *low*. (d) CCR = 1.0, R = high. (e) CCR = 10.0, R = *low*. (f) CCR = 10.0, R = high.

parts, namely computing costs, dispersion correction of infeasible solutions and other procedure. Other procedure includes genetic operations, population update, roulette wheel array update, etc. TABLE 4 shows the percentage of each part. The table shows that dispersion correction of infeasible individuals improves the solution quality at the cost of taking up a large amount of time. Besides, when the number of nodes in the task graph remains unchanged, and the ratio of node to edge is 1:2:3, the proportion of dispersion correction of infeasible individuals increases accordingly. It reflects that when the task graph becomes complex, it is more difficult to generate an feasible offspring only by genetic operation.

Table 5 further shows the reduction of running time step by step through CPU-GPU parallel strategy. The contents of

TABLE 4. Percentage of time cost (%).

No	Compute costs	dispersion correction	other
6	27%	65%	8%
7	24%	69%	7%
8	18%	73%	9%

TABLE 5. Reduction of time after step-by-step parallelization (million second).

No	Serial implementation	Multi-core CPU	Many-core GPU	Asynchronous transfer
6	4060	2413	1803	1550
7	4970	2915	2130	1710
8	6732	3155	2415	2065

table include the time of sequential implementation, implementation of mutli-core CPU, implementation of many-core GPU and asynchronous transfer pattern. Since in our method, we configure one individual to one thread blcok, means the number of total thread block is 48. For each individual, the size of thread block is 512. At the stage of stream concurent, the number of stream is given as 4, namely 12 individuals per stream.

Finally, figure 14 shows the run-time of our method between sequential implementation and CPU-GPU parallel implementation in the 6 cases.

From the above analysis, we can see that the solution quality of our method is competitive with existing heuristic methods in reasonable time. After combining with multi-core CPU and many-core GPU, the run-time is greatly reduced.

V. CONCLUSION AND FUTURE WORK

This paper presents a CPU-GPU parallel genetic algorithm with dispersion correction for HW/SW partitioning. The contributions of this work are as follows. Firstly, an enhanced genetic algorithm with dispersion correction is presented. The under-constraint individuals are marched to feasible region with dispersion correction step by step. Secondly, the individuals processing including costs computation and dispersion correction are run in parallel. For a given problem size, the overall running time can be reduced while keeping the diversity of genetic algorithm. Thirdly, we present a novel parallel strategy by leveraging the parallel power of multi-core CPU and that of many-core GPU. The proposed strategy computes the costs of each individual in parallel on GPU and corrects the under-constraint individuals in parallel on multi-core CPU. Fourthly, in order to further improve the efficiency of proposed algorithm, we propose an asynchronous transfer pattern for CPU-GPU, in which the transfer process and computation process are overlapped and eventually the overall run-time is reduced further.

The above technique details and strategies have two benefits. The intensification of exploitation is enhanced while the diversity of exploration is maintained. An efficient parallel approach is constructed to match well the architecture of CPU/GPU hardware to greatly reduce the run-time when considering the computation workload for improving the solution quality. Numerous experiments demonstrate the effectiveness of the proposed approach.

It is also very interesting that the proposed ideas have a general significance to guide how to accelerate other types of HW/SW co-design [40], [41], as well as other applications in CAD and graphics [42]–[46], Computer-Supported Cooperation Work [47], [48], image and video processing [49]–[54].

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable and constructive comments.

REFERENCES

[1] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proc. IEEE*, vol. 82, no. 7, pp. 967–989, Jul. 1994.
[2] W. H. Wolf, "A decade of hardware/software codesign," *Computer*, vol. 36, no. 4, pp. 38–43, 2003.
[3] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proc. IEEE*, vol. 100, pp. 1411–1430, May 2012.
[4] I. Mhadhbi, S. Ben Othman, and S. Ben Saoud, "A comprehensive survey on hardware/software partitioning process in co-design," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 3, p. 263, 2016.
[5] R. Wang, W. N. N. Hung, G. Yang, and X. Song, "Uncertainty model for configurable hardware/software and resource partitioning," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3217–3223, Oct. 2016.
[6] P. Arató, S. Juhász, Z. A. Mann, A. Orbán, and D. Papp, "Hardware-software partitioning in embedded system design," presented at the IEEE Int. Symp. Intell. Signal Process., Budapest, Hungary, Sep. 2003, pp. 197–202.
[7] P. Arató, Z. A. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Trans Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, 2005.
[8] W. Shi, J. Wu, S.-K. Lam, and T. Srikanthan, "Algorithms for bi-objective multiple-choice hardware/software partitioning," *Comput. Elect. Eng.*, vol. 50, pp. 127–142, Feb. 2016.
[9] A. B. Trindade and L. C. Cordeiro, "Applying SMT-based verification to hardware/software partitioning in embedded systems," *Des. Autom. Embedded Syst.*, vol. 20, no. 1, pp. 1–19, 2016.
[10] Z. A. Mann, A. Orbán, and P. Arató, "Finding optimal hardware/software partitions," *Formal Methods Sys. Des.*, vol. 31, no. 3, pp. 241–263, 2007.
[11] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Des. Test Comput.*, vol. 10, no. 3, pp. 29–41, Sep. 1993.
[12] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Des. Test Comput.*, vol. 10, no. 4, pp. 64–75, Dec. 1993.
[13] N. Janakiraman and P. N. Kumar, "Multi-objective module partitioning design for dynamic and partial reconfigurable system-on-chip using genetic algorithm," *J. Syst. Archit.*, vol. 60, no. 1, pp. 119–139, 2014.
[14] G. Wang, W. Gong, and R. Kastner, "Application partitioning on programmable platforms using the ant colony optimization," *J. Embedded Comput.*, vol. 2, no. 1, pp. 119–136, 2006.
[15] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony optimization for mapping, scheduling and placing in reconfigurable systems," presented at the IEEE NASA/ESA Conf. Adapt. Hardw. Syst., Turin, Italy, Jun. 2013, pp. 47–54.
[16] M. Koudil, K. Benatchba, A. Tarabet, and EI B. Sahraoui, "Using artificial bees to solve partitioning and scheduling problems in codesign," *Appl. Math. Comput.*, vol. 186, no. 2, pp. 1710–1722, 2007.
[17] M. B. Abdelhalim and S. E.-D. Habib, "An integrated high-level hardware/software partitioning methodology," *Des. Autom. Embedded Syst.*, vol. 15, no. 1, pp. 19–50, 2011.
[18] X. Yan, F. He, N. Hou, and H. Ai, "An efficient particle swarm optimization for large-scale hardware/software co-design system," *Int. J. Cooperat. Inf. Syst.*, vol. 9, no. 4, p. 59, 2017, doi: 10.1142/S0218843017410015.
[19] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 2, pp. 273–289, Apr. 2001.

[20] K. Garg, Y. L. Aung, S.-K. Lam, and T. Srikanthan, "KnapSim—Run-time efficient hardware-software partitioning technique for FPGAs," presented at the 28th IEEE Int. Conf. Syst.-Chip, Beijing, China, 2015, pp. 64–69.

[21] Y. Zhang, W. Luo, Z. Zhang, B. Li, and X. Wang, "A hardware/software partitioning algorithm based on artificial immune principles," *Appl. Soft Comput.*, vol. 8, no. 1, pp. 383–391, 2008.

[22] Y. Jiang *et al.*, "Uncertain model and algorithm for hardware/software partitioning," presented at the IEEE Comput. Soc. Annu. Symp. VLSI, Amherst, MA, USA, 2012, pp. 243–248.

[23] G. Li, J. Feng, C. Wang, and J. Wang, "Hardware/software partitioning algorithm based on the combination of genetic algorithm and tabu search," *Eng. Rev.*, vol. 34, no. 2, pp. 151–160, 2014.

[24] G. Lin, W. Zhu, and M. M. Ali, "A tabu search-based memetic algorithm for hardware/software partitioning," *Math. Problems Eng.*, vol. 2014, pp. 1–15, Jul. 2014.

[25] T. Zhang, X. Zhao, X. An, H. Quan, and Z. Lei, "Using blind optimization algorithm for hardware/software partitioning," *IEEE Access*, vol. 5, pp. 1353–1362, 2017.

[26] X.-H. Yan, F.-Z. He, and Y.-L. Chen, "A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization," *J. Comput. Sci. Technol.*, vol. 32, no. 2, pp. 340–355, 2017.

[27] S. A. Tahaee and A. H. Jahangir, "A polynomial algorithm for partitioning problems," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 9, no. 4, p. 34, 2010.

[28] J. G. Wu, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: 1D search algorithms," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 532–544, Apr. 2010.

[29] J. G. Wu, P. Wang, S. K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," *J. Supercomput.*, vol. 66, no. 1, pp. 118–134, 2013.

[30] Z. Chen, J.-G. Wu, G.-Z. Song, and J.-L. Chen, "NodeRank: An efficient algorithm for hardware/software partitioning," *Chin. J. Comput.*, vol. 36, no. 10, pp. 2033–2040, 2013.

[31] H. Quan, T. Zhang, Q. Liu, J. Guo, X. Wang, and R. Hu, "Comments on 'algorithmic aspects of hardware/software partitioning:1D search algorithms,'" *IEEE Trans. Comput.*, vol. 4, no. 63, pp. 1055–1056, Apr. 2014.

[32] A. F. Farahani, M. Kamal, and M. Salmani-Jelodar, "Parallel genetic algorithm based HW/SW partitioning," presented at the Int. Symp. Parallel Comput. Electr. Eng., Bialystok, Poland, 2006, pp. 337–342.

[33] Y. Wu, H. Zhang, and H. Yang, "Research on parallel HW/SW partitioning based on hybrid PSO algorithm," presented at the Int. Conf. Algorithms Architectures Parallel Process., 2009, pp. 449–459.

[34] Y. Zhou, F. He, and Y. M. Qiu, "Optimization of parallel iterated local search algorithms on graphics processing unit," *J. Supercomput.*, vol. 72, no. 6, pp. 2394–2416, 2016.

[35] Y. Zhou, F. He, and Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSP," *Sci. China, Inf. Sci.*, vol. 60, no. 6, p. 068102, 2017.

[36] G. Jia, Y. Wang, Z. Cai, and Y. Jin, "An improved ($\mu + \lambda$)-constrained differential evolution for constrained optimization," *Inf. Sci.*, vol. 222, pp. 302–322, Feb. 2013.

[37] R. A. Rahman, R. Ramli, Z. Jamari, and K. R. Ku-Mahamud, "Evolutionary algorithm with roulette-tournament selection for solving aquaculture diet formulation," *Math. Problems Eng.*, vol. 2016, pp. 1–10, May 2016.

[38] M. A. Alsmirat, Y. Jararweh, M. Al-Ayyoub, M. A. Shehab, and B. B. Gupta, "Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU-GPU implementations," *Multimedia Tools Appl.*, vol. 76, no. 3, pp. 3537–3555, 2017.

[39] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," presented at the IEEE Int. Workshop Workload Characterization, Austin, TX, USA, 2001, pp. 3–14.

[40] W. Zuo *et al.*, "Accurate high-level modeling and automated hardware/software co-design for effective SoC design space exploration," presented at the 54th Annu. Design Autom. Conf., Austin, TX, USA, 2017, p. 78.

[41] S. Ben Haj Hassine, M. Jemai, and B. Ouni, "Power and execution time optimization through hardware software partitioning algorithm for core based embedded system," *J. Optim.*, vol. 2017, Feb. 2017, Art. no. 8624021.

[42] D. J. Zhang, F. Z. He, S. H. Han, and X. X. Li, "Quantitative optimization of interoperability during feature-based data exchange," *Integr. Comput.-Aided Eng.*, vol. 23, no. 1, pp. 31–50, 2016.

[43] J. Xue, G. Zhao, and W. Xiao, "Efficient GPU out-of-core visualization of large-scale CAD models with voxel representations," *Adv. Eng. Softw.*, vol. 99, pp. 73–80, Sep. 2016.

[44] Y. Wu, F. He, D. Zhang, and X. Li, "Service-oriented feature-based data exchange for cloud-based design and manufacturing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2015.2501981.

[45] Y. L. Chen, F. Z. He, Y. Q. Wu, and N. Hou, "A local start search algorithm to compute exact Hausdorff distance for arbitrary point sets," *Pattern Recognit.*, vol. 67, pp. 139–148, Jul. 2017.

[46] R. Li, Q. Hou, and K. Zhou, "Efficient GPU path rendering using scanline rasterization," *ACM Trans Graph.*, vol. 35, no. 6, p. 228, 2016.

[47] X. Lv, F. He, W. Cai, and Y. Cheng, "A string-wise CRDT algorithm for smart and large-scale collaborative editing systems," *Adv. Eng. Informat.*, vol. 19, pp. 397–409, Aug. 2016.

[48] Y. Cheng *et al.*, "Meta-operation conflict resolution for human–human interaction in collaborative feature-based CAD systems," *Cluster Comput.*, vol. 19, no. 1, pp. 237–253, 2016.

[49] B. Ni, F.-Z. He, Y.-T. Pan, and Z.-Y. Yuan, "Using shapes correlation for active contour segmentation of uterine fibroid ultrasound images in computer-aided therapy," *Appl. Math.-A, J. Chin. Univ.*, vol. 31, no. 1, pp. 37–52, 2016.

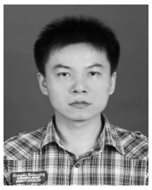
[50] K. Li, F. Z. He, H. P. Yu, and Y. T. Pan, "A parallel and robust object tracking approach synthesizing adaptive Bayesian learning and improved incremental subspace learning," *Frontiers Comput. Sci.*, to be published, doi: 10.1007/s11704-018-6442-4.

[51] K. Li, F. Z. He, and X. Chen, "Real-time object tracking via compressive feature selection," *Frontiers Comput. Sci.*, vol. 10, no. 4, pp. 689–701, 2016.

[52] K. Li, F.-Z. He, H.-P. Yu, and X. Chen, "A correlative classifiers approach based on particle filter and sample set for tracking occluded target," *Appl. Math.-A, J. Chin. Univ.*, vol. 32, no. 3, pp. 294–312, 2017.

[53] G. M. Rao and C. M. Rao, "GPU based video tracking system," presented at the IEEE 10th Int. Conf. Semantic Comput., Laguna Hills, CA, USA, 2016, pp. 170–171.

[54] J. Sun, F.-Z. He, Y.-L. Chen, and X. Chen, "A multiple template approach for robust tracking of fast motion target," *Appl. Math.-A, J. Chin. Univ.*, vol. 31, no. 2, pp. 177–197, 2016.



NENG HOU received the B.S. degree from the Huazhong University of Science and Technology in 2008 and the M.S. degree from Guangxi University in 2012, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University.

His research interests include hardware/software co-design, intelligent optimization algorithms, and parallel computing.

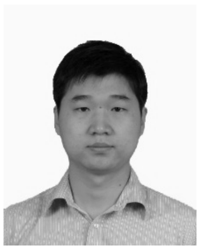
FAZHI HE received the B.S., M.S., and Ph.D. degrees from the Wuhan University of Technology 1990, 1996, and 2000, respectively.

He has been a Post-Doctoral Scholar with Zhejiang University, a Visiting Researcher with the Korea Institute of Science and Technology, and a Visiting Faculty Member with the University of North Carolina Chapel Hill.

He is currently a Professor with the State Key Laboratory of Software Engineering, Wuhan University. His research interests include collaborative computation, computer-aided design, and computer graphics.

YI ZHOU was born in 1983. He received the Ph.D. degree from Wuhan University in 2016. He is currently a Lecturer with the Engineering Research Center of Metallurgical Automation and Measurement Technology, School of Information Science and Engineering, Wuhan University of Science and Technology.

His research interests include artificial intelligent and parallel computing.



YILIN CHEN is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University. His research interests include GPGPU in computer graphics.

XIAOHU YAN is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University. His research interests include evolutionary computing and hardware/software co-design.