Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 404375, 23 pages
http://dx.doi.org/10.1155/2014/404375

Research Article

A DAG Scheduling Scheme on Heterogeneous Computing Systems Using Tuple-Based Chemical Reaction Optimization

Yuyi Jiang, Zhiqing Shao, and Yi Guo

College of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

Correspondence should be addressed to Zhiqing Shao; zshao@ecust.edu.cn

Received 1 April 2014; Revised 5 May 2014; Accepted 8 May 2014; Published 24 June 2014

Academic Editor: Yu-Bo Yuan

A complex computing problem can be solved efficiently on a system with multiple computing nodes by dividing its implementation code into several parallel processing modules or tasks that can be formulated as directed acyclic graph (DAG) problems. The DAG jobs may be mapped to and scheduled on the computing nodes to minimize the total execution time. Searching an optimal DAG scheduling solution is considered to be NP-complete. This paper proposed a tuple molecular structure-based chemical reaction optimization (TMSCRO) method for DAG scheduling on heterogeneous computing systems, based on a very recently proposed metaheuristic method, chemical reaction optimization (CRO). Comparing with other CRO-based algorithms for DAG scheduling, the design of tuple reaction molecular structure and four elementary reaction operators of TMSCRO is more reasonable. TMSCRO also applies the concept of constrained critical paths (CCPs), constrained-critical-path directed acyclic graph (CCPDAG) and super molecule for accelerating convergence. In this paper, we have also conducted simulation experiments to verify the effectiveness and efficiency of TMSCRO upon a large set of randomly generated graphs and the graphs for real world problems.

1. Introduction

Modern computer systems with multiple processors working in parallel may enhance the processing capacity for an application. The effective scheduling of parallel modules of the application may fully exploit the parallelism. The application modules may communicate and synchronize several times during the processing. The limitation of the overall application performance may be incurred by a large communication cost on heterogeneous systems with a combination of GPUs, multicore processors and CELL processors, or distributed memory systems. And an effective scheduling may greatly improve the performance of the application.

Scheduling generally defines not only the processing order of application modules but also the processor assignment of these modules. The concept of makespan (i.e., the schedule length) is used to evaluate the scheduling solution quality including the entire execution and communication cost of all the modules. On the heterogeneous systems [1–4], searching optimal schedules minimizing the makespan is considered as a NP-complete problem. Therefore, two classes of scheduling strategies have been proposed to solve this problem by finding the suboptimal solution with lower time overhead, such as heuristic scheduling and metaheuristic scheduling.

Heuristic scheduling strategies try to identify a good solution by exploiting the heuristics. An important subclass of heuristic scheduling is list scheduling with an ordered task list for a DAG job on the basis of some greedy heuristics. Moreover, the ordered tasks are selected to be allocated to the processors which minimize the start times in list scheduling algorithms. In heuristic scheduling, the attempted solutions are narrowed down by greedy heuristics to a very small portion of the entire solution space. And this limitation of the solution searching leads to the low time complexity. However, the higher complexity DAG scheduling problems have, the harder greedy heuristics produce consistent results on a wide range of problems, because the quality of the found solutions relies on the effectiveness of the heuristics, heavily.

Metaheuristic scheduling strategies such as ant colony optimization (ACO), genetic algorithms (GA), Tabu search (TS), simulated annealing (SA), and so forth take more

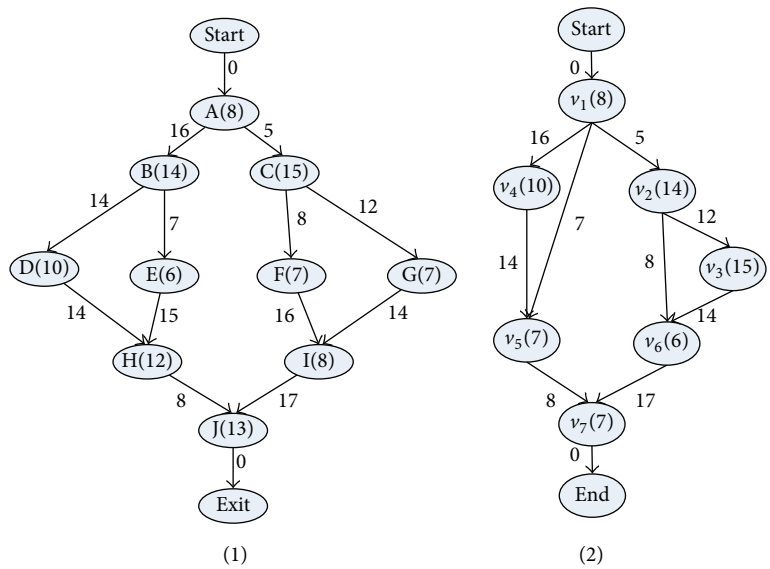

Figure 1: Two simple DAG models with 7 and 10 tasks.

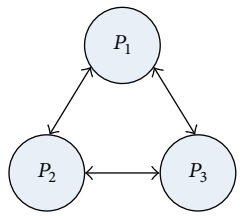

Figure 2: A fully connected parallel system with 3 heterogeneous processors.

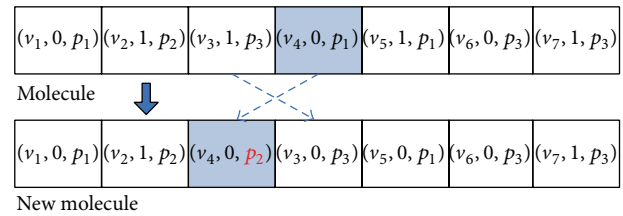

Figure 4: Illustration of molecular structure change for on-wall ineffective collision.

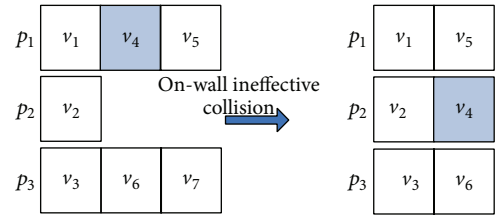

Figure 5: Illustration of the task-to-computing-node mapping for on-wall ineffective collision.

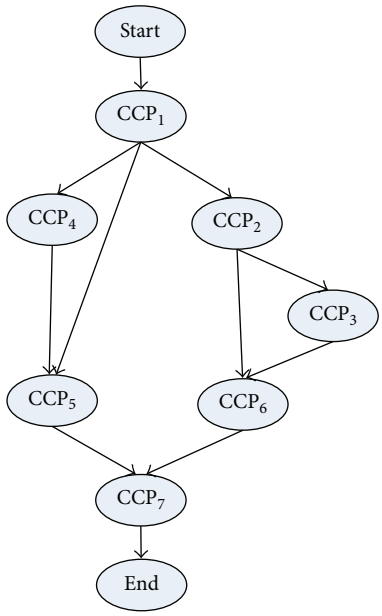

Figure 3: CCPDAG corresponding to the DAG as shown in Figure 1 and the CCP as indicated in Table 1.

time cost than heuristic scheduling strategies, but they can produce consistent results with high quality on the problems with a wide range by directed searching solution spaces.

Chemical reaction optimization (CRO) is a new metaheuristic method proposed very recently and has shown its power to deal with NP-complete problem. There is only one CRO-based algorithm called double molecular structure-based CRO (DMSCRO) for DAG scheduling on heterogeneous system as far as we know. DMSCRO has a better performance on makespan and convergence rate than genetic algorithm (GA) for DAG scheduling on heterogeneous systems. However, the rate of convergence of DMSCRO as a metaheuristic method is still defective. This paper proposes a

(1) //PHASE 1: Find the constrained critical paths (CCPs)
(2) Find set of critical paths CP according to the description in the second paragraph of Section 3.1.
(3) $j = 1$
(4) **for** $i = 1$ to $|\mathrm{CP}|$ **do**
(5) **while** there exist ready nodes in CP_i **do**
(6) Insert ready node v_k into constrained critical path Queue (Q_j).
(7) **end while**
(8) $j \leftarrow j + 1$
(9) $i \leftarrow i\%\,|\mathrm{CP}|$
(10) **end for**
(11) //PHASE 2: Assign and schedule tasks
(12) **for** $j = \{1, 2, \ldots, |Q|\}$ do
(13) **for** each processor $P_r \in P$ do
(14) **for** each node $w \in Q_j$ do
(15) Find the start time of node k, which is the predecessor of w
$$\mathrm{ST}_{P_r}(w,k) = \max\left(\left(\left(\mathrm{AEFT}_k\right) + \mathrm{CM}\left(w, P_r, k, P_x\right)\right), \mathrm{AT}_{P_r}\right)$$
(16) Find the finish time of the node
$$\mathrm{EFT}_{P_r}(w) = \max\left(\mathrm{ST}_{P_r}(w,k)\right)_{\forall k \in \mathrm{Pred}(w)} + \mathrm{EC}_{P_r}(w)$$
(17) **end for**
(18) Find the finish time of the CCP Q_j
$$\mathrm{CEFT}_{P_r}(Q_j) = \max\left(\left(\mathrm{EFT}_{P_r}(w)\right)_{\forall w \in Q_j}\right)$$
(19) **end for**
(20) Assign the processor to CCP Q_j which minimizes $\mathrm{CEFT}_{P_r}(Q_j)$.
(21) Let P_x be assigned, update AEFT_w of each task w in Q_j
$$(\mathrm{AEFT}_w)_{\forall w \in Q_j} = (\mathrm{EFT}_{P_x}(w))_{\forall w \in Q_j}$$
(22) **end for**

Algorithm 1: CEFT.

(1) **for** each $E_i = (ev_s, ev_e, w_{s,e})$ in E
(2) $\mathrm{CCP}_s = \mathrm{BelongCCP}\,(ev_s)$;
(3) $\mathrm{CCP}_e = \mathrm{BelongCCP}\,(ev_e)$;
(4) **if** $(\mathrm{CCP}_s \neq \mathrm{CCP}_e)$ & $(\mathrm{CCPE}\,(\mathrm{CCP}_s, \mathrm{CCP}_e))$ does not exist
(5) create $\mathrm{CCPE}\,(\mathrm{CCP}_s, \mathrm{CCP}_e)$
(6) **end if**
(7) add Start and End
(8) add edges among Start and CCP nodes
(9) add edges among End and CCP nodes
(10) **end for**

Algorithm 2: Gen_CCPDAG(DAG, CCP) generating CCPDAG.

new CRO-based algorithm, tuple molecular structure-based CRO (TMSCRO), for the mentioned problem, encoding the two basic components of DAG scheduling, module execution order and module-to-processor mapping, into an array of tuples. Combining this kind of molecular structure with the elementary reaction operator designed in TMSCRO has a better capability of intensification and diversification than DMSCRO. Moreover, in TMSCRO, the concept of constrained critical paths (CCPs) [5] and constrained-critical-path directed acyclic graph (CCPDAG) are applied to creating initial population in order to speed up the convergence of TMSCRO. In addition, the first initial molecule, InitS, is also considered to be a super molecule [6] for accelerating convergence, which is converted from the scheduling result of the algorithm constrained earliest finish time (CEFT).

In theory, a metaheuristic method will gradually approach the optimal result if it runs for long enough, based on No-Free-Lunch Theorem, which means the performances of the search for optimal solution of each metaheuristic algorithm are alike when averaged over all possible fitness functions. We have conducted the simulation experiments over the graphs abstracted from two well-known real applications: Gaussian elimination and molecular dynamics application and also a large set of randomly generated graphs. The experiment results show that the proposed TMSCRO can achieve similar performance as DMSCRO

```
(1)  InitS = ConvertMole(InitCCPS);
(2)  update each f_i in molecule InitS as defined in the last paragraph of Section 5.1.1
(3)  MoleN = 1;
(4)  while MoleN ≤ PopSize − 1 do
(5)      for each CCP_i in CCP molecule CCPS
(6)          find the first successor Succ(i) in CCPDAG from i to the end;
(7)          for each CCP_j, j ∈ (i, Succ(i))
(8)              find the first predecessor Pred(j) from Succ(i) to the begin in CCP molecule CCPS;
(9)                  if Pred(j) < i
(10)                     interchanged position of (CCP_i, sp_i) and (CCP_j, sp_j) in CCP molecule CCPS;
(11)                 end if
(12)         end for
(13)     end for
(14)     Generate a new CCP molecule CCPS′;
(15)     S = ConvertMole(CCPS′)
(16)     update each f_i in reaction molecule S as defined in the last paragraph of Section 5.1.1
(17)     MoleN ← MoleN + 1;
(18) end while
```

Algorithm 3: InitTMolecule(InitCCPS) generating the initial population.

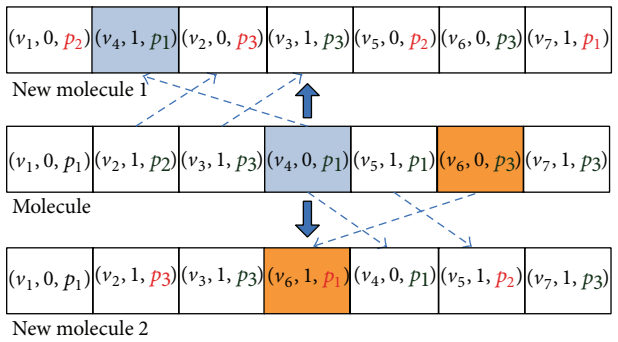

Figure 6: Illustration of molecular structure change for decomposition.

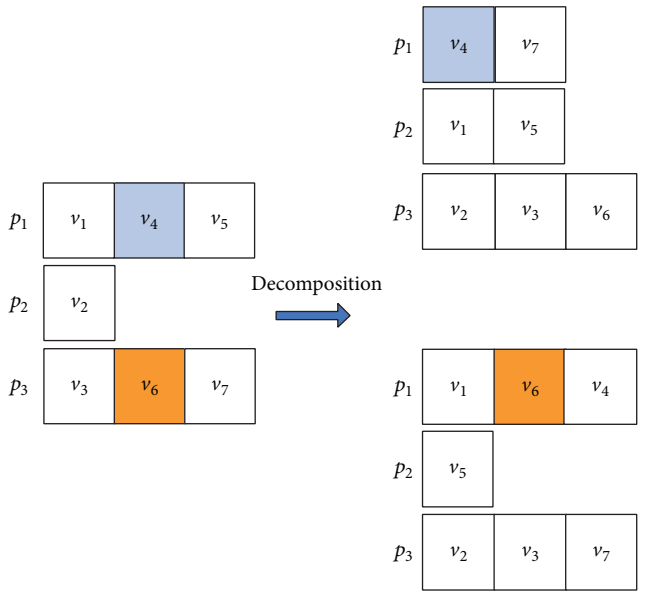

Figure 7: Illustration of the task-to-computing-node mapping for decomposition.

in the literature in terms of makespan and outperforms the heuristic algorithms.

There are three major contributions of this work.

(1) Developing TMSCRO based on CRO framework by designing a more reasonable molecule encoding method and elementary chemical reaction operators on intensification and diversification search than DMSCRO.

(2) For accelerating convergence, applying CEFT and CCPDAG to the data pretreatment, utilizing the concept of CCPs in the initialization, and using the first initial molecule, InitS, to be a super molecule in TMSCRO.

(3) Verifying the effectiveness and efficiency of the proposed TMSCRO by simulation experiments. The simulation results of this paper show that TMSCRO is able to approach similar makespan as DMSCRO, but it finds good solutions faster than DMSCRO by 12.89% on average (by 26.29% in the best case).

2. Related Work

Most of the scheduling algorithms can be categorized into heuristic scheduling (including list scheduling, duplication-based scheduling, and cluster scheduling) and metaheuristic (i.e., guided-random-search-based) scheduling. These strategies are to generate the scheduling solution before the execution of the application. The approaches adopted by these different scheduling strategies are summarized in this section.

2.1. Heuristic Scheduling. Heuristic methods usually provide near-optimal solutions for a task scheduling problem in less

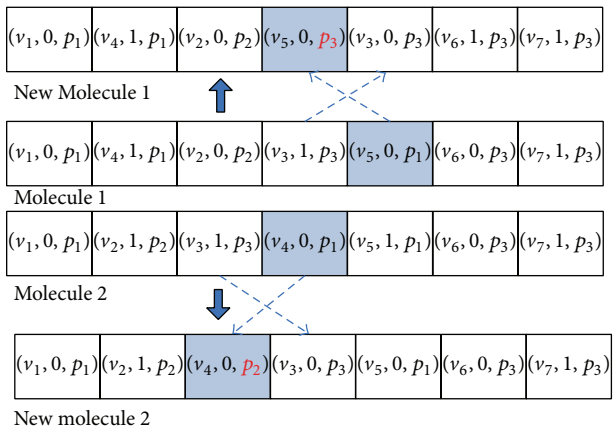

Figure 8: Illustration of molecular structure change for intermolecular ineffective collision.

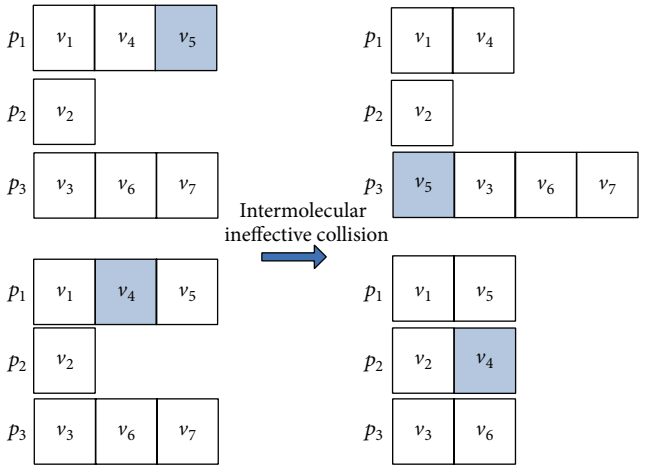

Figure 9: Illustration of the task-to-computing-node mapping for intermolecular ineffective collision.

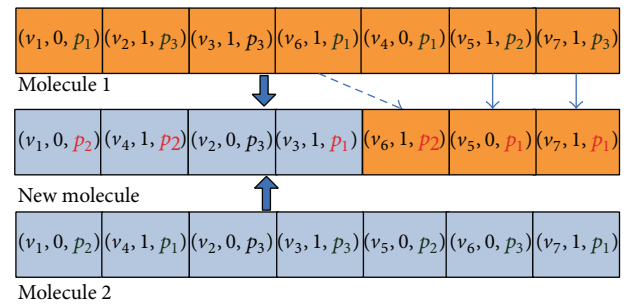

Figure 10: Illustration of molecular structure change for synthesis.

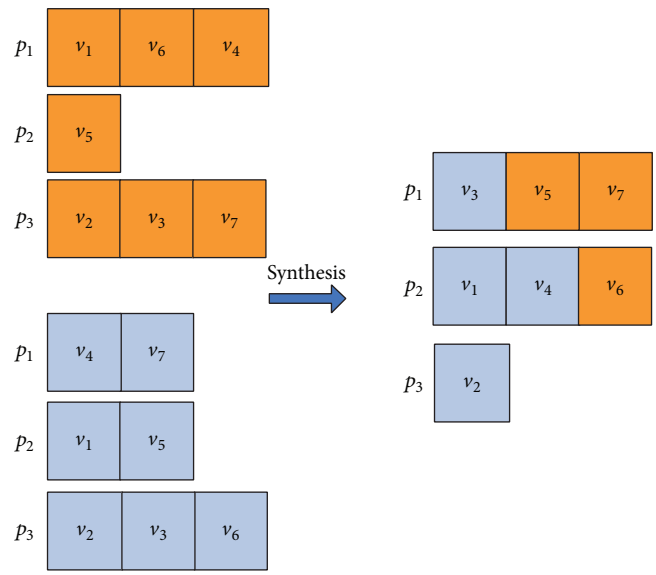

Figure 11: Illustration of the task-to-computing-node mapping for synthesis.

than polynomial time. The approaches adopted by heuristic method search only one path in the solution space, ignoring other possible ones [7]. Three typical kinds of algorithms based on heuristic scheduling for the DAG scheduling problem are discussed as below, such as list scheduling [7, 8], cluster scheduling [9, 10], and duplication-based scheduling [11, 12].

The list scheduling [7, 13–21] generates a schedule solution in two primary phases. In phase 1, all the tasks are processed in a sequence order by their assigned priorities, which are normally based on the task execution and communication costs. There are two attributes used in most list scheduling algorithms, such as b-level and t-level, to assign task priorities. In a DAG, b-level of a node (task) is the length of the longest path from the end node to the node; however, t-level of a node is the length of the longest path from the entry node to the node. In phase 2, the processors are assigned to each task in the sequence.

The heterogeneous earliest finish time (HEFT) scheduling algorithm [16] assigns the scheduling task priorities based on the earliest start time of each task. HEFT allocates a task to the processor which minimizes the task's start time.

The modified critical path (MCP) scheduling [22] considers only one CP (critical path) of the DAG and assigns the scheduling priority to tasks based on their latest start time. The latest start times of the CP tasks are equal to their t-levels. MCP allocates a task to the processor which minimizes the task's start time.

Dynamic-level scheduling (DLS) [23] uses the concept of the dynamic level, which is the difference between the b-level and earliest start time of a task on a processor. Each time the (task, processor) pair with the largest dynamic-level value is chosen by DLS during the task scheduling.

Mapping heuristic (MH) [24] assigns the task scheduling priorities based on the static b-level of each task, which is the b-level without the communication costs between tasks. Then, a task is allocated to the processor which gives the earliest start time.

Levelized-min time (LMT) [17] assigns the task scheduling priority in two steps. Firstly, it groups the tasks into different levels based on the topology of the DAG, and then in each level, the task with the highest priority is the one with

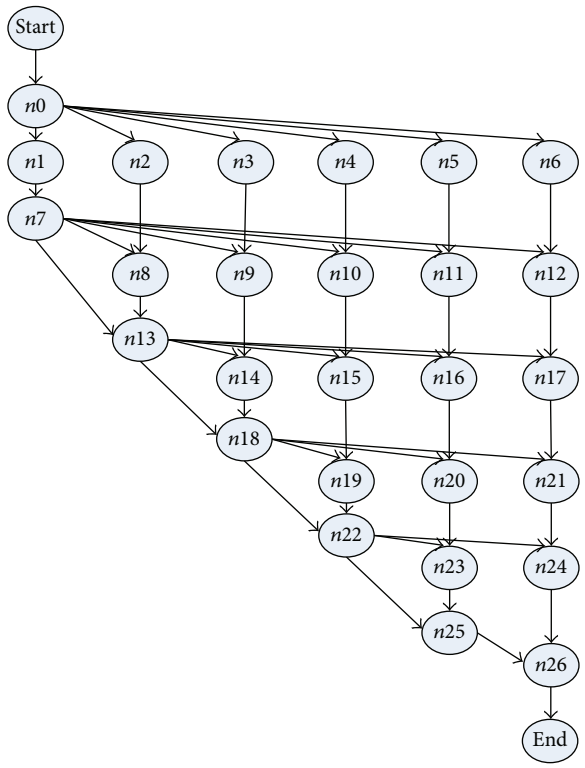

Figure 12: Gaussian elimination for a matrix of size 7.

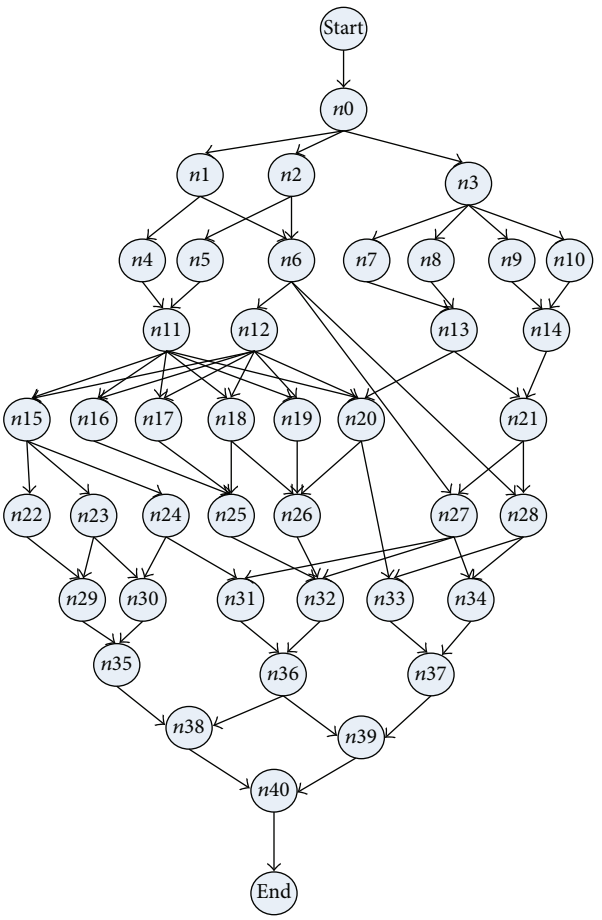

Figure 13: A molecular dynamics code.

the largest execution cost. A task is allocated to the processor which minimizes the sum of the total communication costs with the tasks in the previous level and the task's execution cost.

There are two heuristic algorithms for DAG scheduling on heterogeneous systems proposed in [8]. One algorithm named HEFT_T uses the sum of t-level and b-level to assign the priority to each task. In HEFT_T, the critical tasks are attempted to be on the same processor, and the other tasks are allocated to the processor that gives earliest start time. The other algorithm named HEFT_B applies the concept of b-level to assign the priority (i.e., scheduling order) to each task. After the priority assignment, a task is allocated to the processor that minimizes the start time. The extensive experiment results in [8] demonstrate that HEFT_B and HEFT_T outperform (in terms of makespan) other representative heuristic algorithms in heterogeneous systems, such as DLS, MH, and LMT.

Comparing with the list scheduling algorithms, the duplication-based algorithms [23, 25–29] attempt to duplicate the tasks to the same processor on heterogeneous systems, because the duplication may eliminate the communication cost of these tasks and it may effectively reduce the total schedule length.

The clustering algorithms [8, 11, 30–32] regard task collections as clusters to be mapped to appropriate processors. These algorithms are mostly used in the homogeneous systems with unbounded number of processors and they will use as many processors as possible to reduce the schedule length. Then, if the number of the processors used for scheduling is

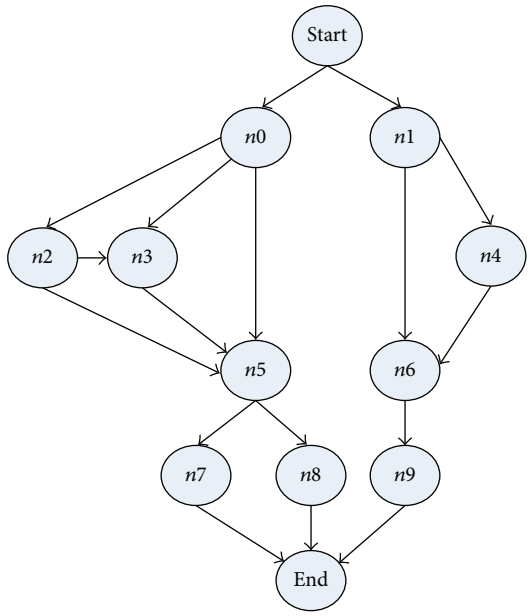

Figure 14: A random graph with 10 nodes.

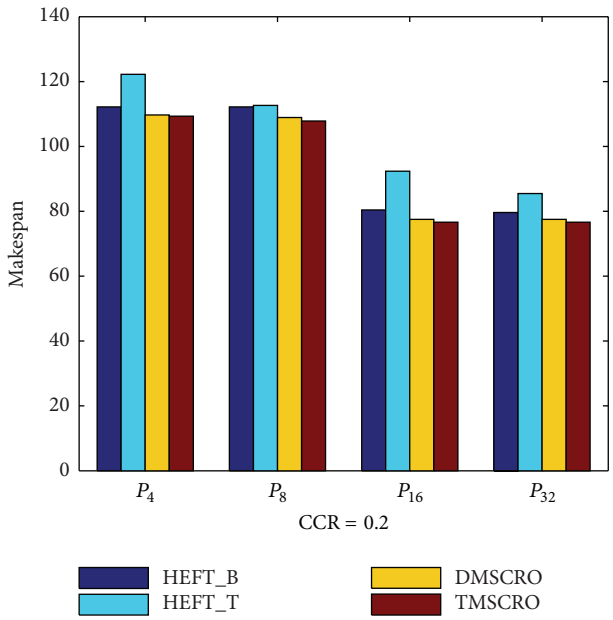

Figure 15: Average makespan for Gaussian elimination.

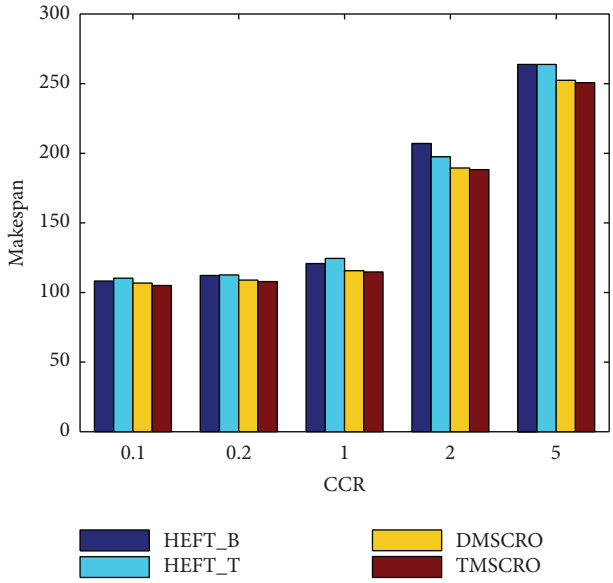

Figure 16: Average makespan for Gaussian elimination; the number of processors is 8.

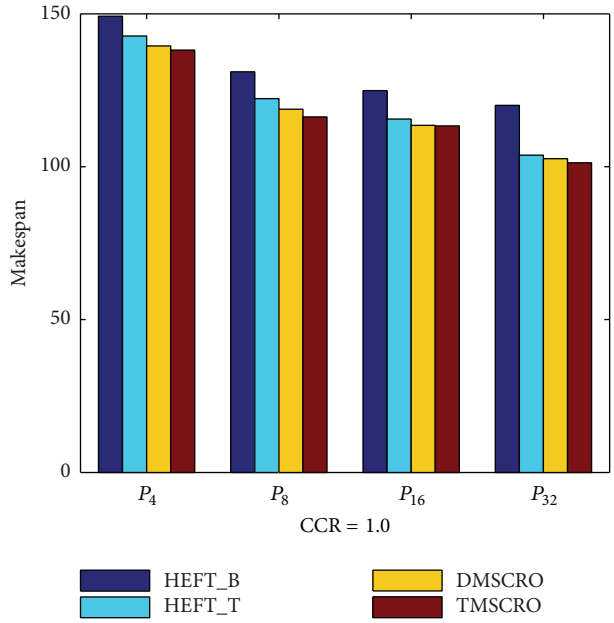

Figure 17: Average makespan for the molecular dynamics code.

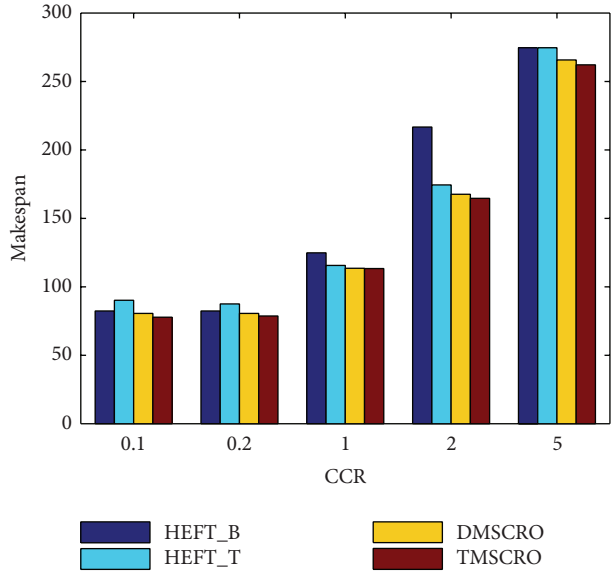

Figure 18: Average makespan for the molecular dynamics code; the number of processors is 16.

more than that of the available processors, the task collections (clusters) are processed further to fit in with a limited number of processors.

2.2. Metaheuristic Scheduling. In comparison with the algorithms based on heuristic scheduling, the metaheuristic (guided-random-search-based) algorithms use a combinatorial process for solution searching. In general, with robust performance on many kinds of scheduling problems, the metaheuristic algorithms need sampling candidate solutions in the search space, sufficiently. Many metaheuristic algorithms have been applied to solve the task scheduling problem successfully, such as GA, chemical reaction optimization (CRO), energy-efficient stochastic [33], and so forth.

GA [15, 31, 34–36] is the mostly used metaheuristic method for DAG scheduling. In [15], a solution for scheduling is encoded as one-dimensional string representing an ordered list of tasks to be allocated to a processor. In each string of two parent solutions, the crossover operator selects a crossover point randomly and then merges the head portion of one parent with the tail portion of the other. Mutation operator

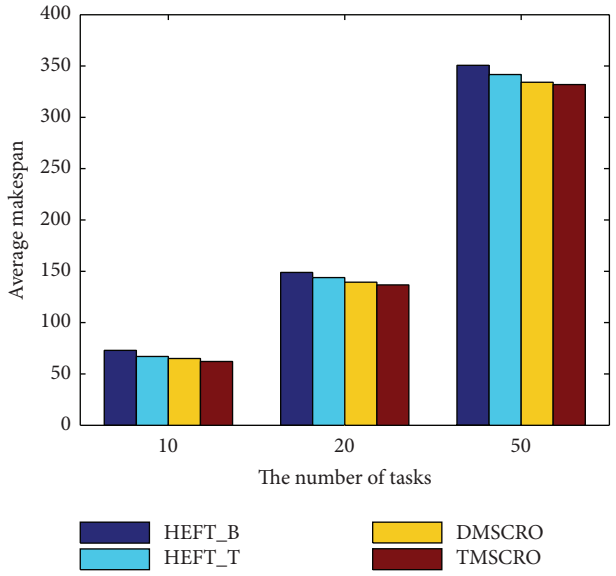

Figure 19: Average makespan of different task numbers, CCR = 10; the number of processors is 32.

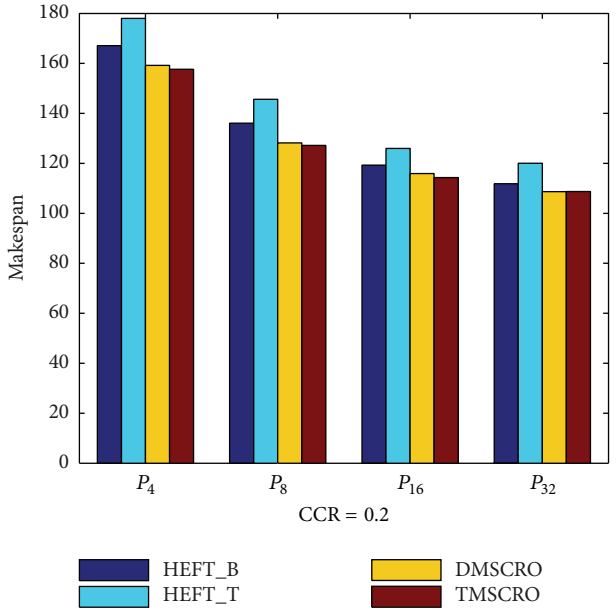

Figure 20: Average makespan of four algorithms under different processor numbers and the low communication costs; the number of tasks is 50.

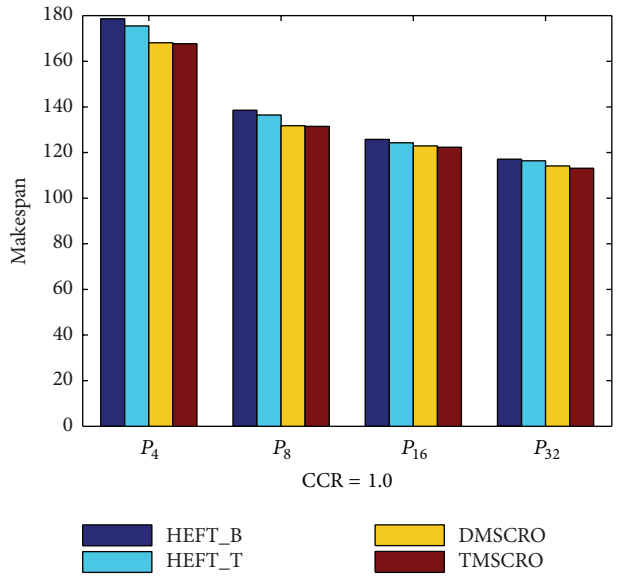

Figure 21: Average makespan of four algorithms under different processor numbers and the low communication costs; the number of tasks is 50.

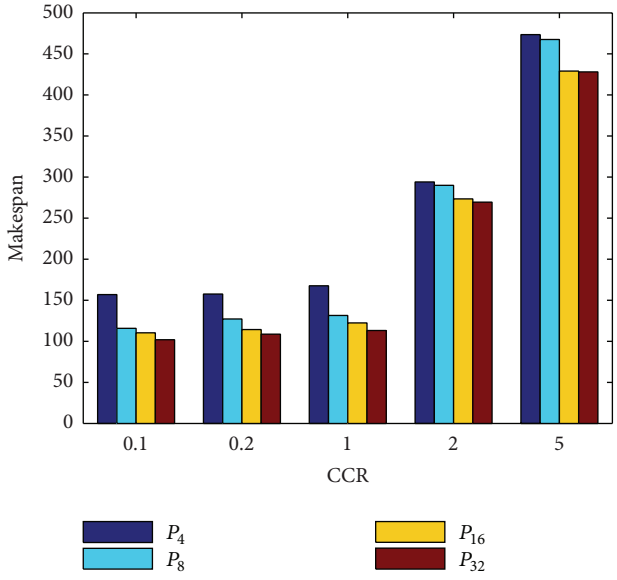

Figure 22: Average makespan of TMSCRO under different values of CCR; the number of tasks is 50.

exchanges two tasks in two solutions, randomly. The concept of makespan is used to evaluate the scheduling solution quality by fitness function.

Chemical reaction optimization (CRO) was proposed very recently [20, 30, 37–39]. It mimics the interactions of molecules in chemical reactions. CRO has good performance already in solving many problems, such as quadratic assignment problem (QAP), resource-constrained project scheduling problem (RCPSP), channel assignment problem (CAP) [39], task scheduling in grid computing (TSGC) [40], and 0-1 knapsack problem (KP01) [41]. So far as we know, double molecular structure-based chemical reaction optimization (DMSCRO) recently proposed in [37] is the only one CRO-based algorithm with two molecular structures for DAG scheduling on heterogeneous systems. CRO-based algorithm (just DMSCRO) mimics the chemical reaction process in a closed container and accords with energy conservation. In DMSCRO, one solution for DAG scheduling including two essential components, task execution order and task-to-processor mapping, corresponds to a double-structured

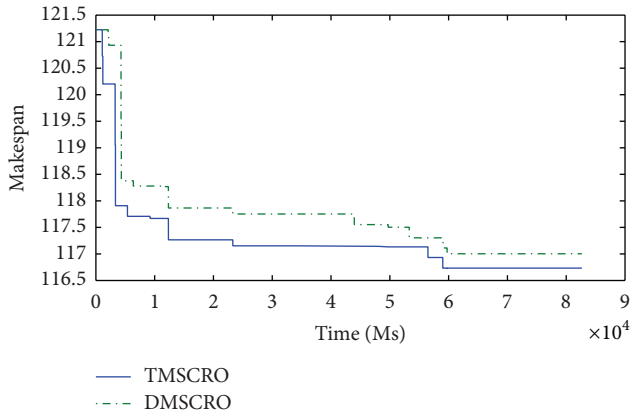

Figure 23: The convergence trace for Gaussian elimination; ccr = 0.2; the number of processors is 8.

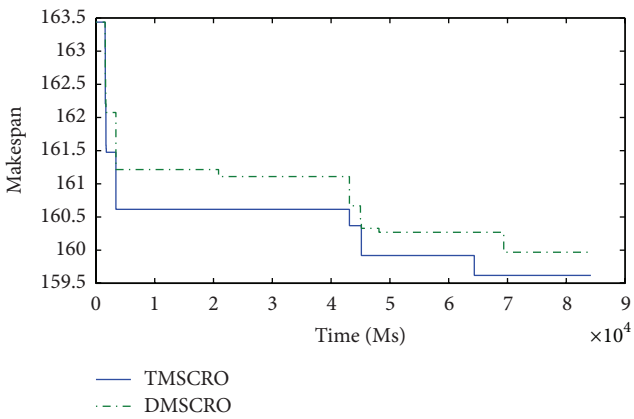

Figure 24: The convergence trace for the molecular dynamics code; ccr = 1; the number of processors is 16.

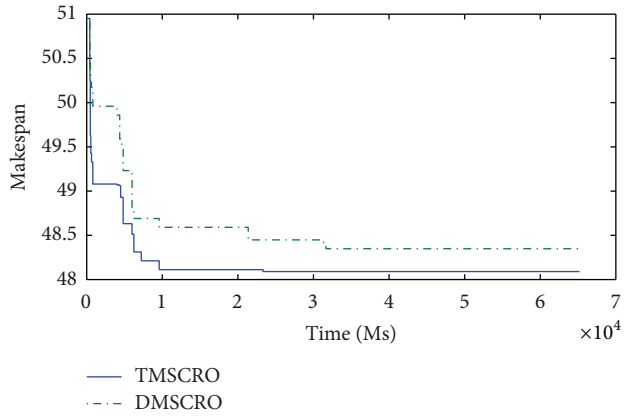

Figure 25: The convergence trace for the randomly generated DAGs with each containing 10 tasks.

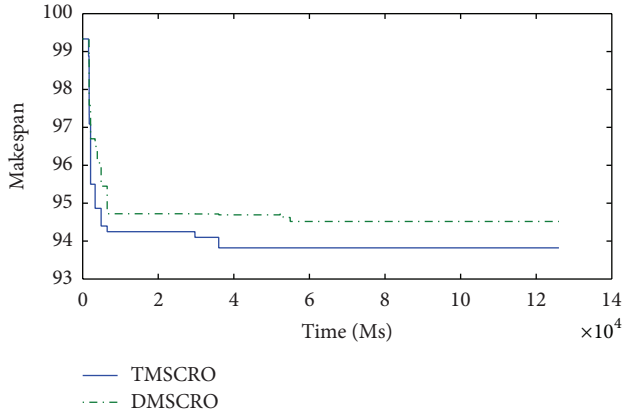

Figure 26: The convergence trace for the randomly generated DAGs with each containing 20 tasks.

molecule with two kinds of energy, potential energy (PE) and kinetic energy (KE). The value of PE of a molecule is just the fitness value (objective value), makespan, of the corresponding solution, which can be calculated by the fitness function designed in DMSCRO, and KE with a nonnegative value is to help the molecule escape from local optimums. There are four kinds of elementary reactions used to do the intensification and diversification search in the solution space to find the solution with the minimal makespan, and the principle of the reaction selection is in detail presented in Section 3.2. Moreover, a central buffer is also applied in DMSCRO for energy interchange and conservation during the searching progress. However, as a metaheuristic method for DAG scheduling, DMSCRO still has very large time expenditure and the rate of convergence of this algorithm needs to be improved. Comparing with GA, DMSCRO is similar in model and workload to TMSCRO proposed in this paper.

Our work is concerned with the DAG scheduling problems and the flaw of CRO-based method for DAG scheduling, proposing a tuple molecular structure-based chemical reaction optimization (TMSCRO). Comparing with DMSCRO, TMSCRO applies CEFT [5] to data pretreatment to take the advantage of CCPs as heuristic information for accelerating convergence. Moreover, the molecule structure and elementary reaction operators design in TMSCRO are more reasonable than those in DMSCRO on intensification and diversification of searching the solution space.

3. Background

3.1. CEFT. Constrained earliest finish time (CEFT) based on the constrained critical paths (CCPs) was proposed for heterogeneous system scheduling in [5]. In contrast to other approaches, the CEFT strategy takes account of a broader view of the input DAG. Moreover, the CCPs can be scheduled efficiently because of their static generation.

The constrained critical path (CCP) is a collection with the tasks ready for scheduling only. A task is ready when all its predecessors were processed. In CEFT, a critical path (CP) is generally the longest path from the start node to the end node for scheduling in the DAG. The DAG is initially traversed and critical paths are found. Then it is pruned off the nodes that constitute a critical path. The subsequent traversals

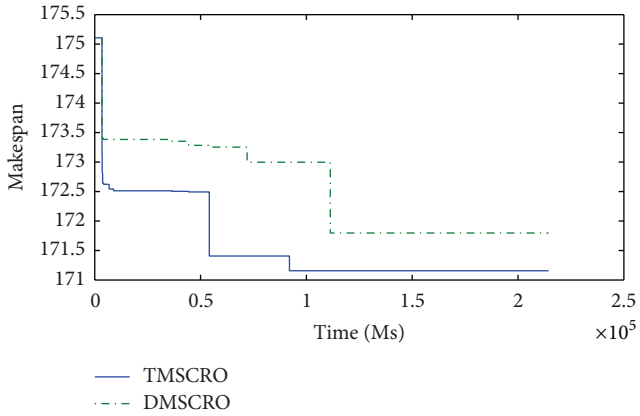

FIGURE 27: The convergence trace for the randomly generated DAGs with each containing 50 tasks.

of the pruned graph produce the remaining critical paths. While the nodes are being removed from the task graph, a pseudo-edge to the start or end node is added if a node has no predecessors or no successors, respectively. The CCPs are subsequently formed by selecting ready nodes in the critical paths in a round-robin fashion. Each CCP may be assigned a single processor which has the minimum finish time of processing all the tasks in the CCP. All the tasks in a CCP not only reduce the communication cost, but also benefit from a broader view of the task graph.

Consider the CEFT algorithm generates schedules for n tasks with $|P|$ heterogeneous processors. Some specific terms and their usage are indicated in Table 1.

The CEFT scheduling approach (Algorithm 1) works in two phases. (1) The critical paths are generated according to the description in the second paragraph of Section 3.1. The critical paths are traversed and the ready nodes are inserted into the constrained critical paths (CCPs) $\mathrm{CCP}_j, \forall j = 1, 2, \ldots, |Q|$. If no more ready nodes are in a critical path, the constrained critical path takes nodes from the next critical path following round-robin traversal of the critical paths. (2) All the CCPs are traversed in order (line 12). Then, $\mathrm{ST}_{P_r}(w, k)$, the maximum of AT_{P_r} and the start time of the predecessors of each node w, is calculated (1). $\mathrm{EFT}_{P_r}(w)$ is computed as the sum of $\mathrm{ST}_{P_r}(w, k)$ and $\mathrm{EC}_{P_r}(w)$ (2). $E_{P_r}(Q_j)$ is the maximum of the finish times of all the CCP nodes on the same processor P_r (3). The processor is then assigned to constrained-critical-path CCP_j which minimizes the $\mathrm{CEFT}_{P_r}(\mathrm{CCP}_j)$ value (line 20). After the actual finish time AEFT_w of each task w in CCP_j is updated, the processor assignment continues iteratively.

3.2. CRO. Chemical reaction optimization (CRO) mimics the process of a chemical reaction where molecules undergo a series of reactions between each other or with the environment in a closed container. The molecules are manipulated agents with a profile of three necessary properties of the molecule, including the following. (1) The molecular structure S: S actually structure represents the positions of atoms in a molecule. Molecular structure can be in the form of a number, a vector, a matrix, or even a graph which is independent of the problem, (2) (Current) potential energy (PE): PE is the objective function value of the current molecular structure ω, that is, $\mathrm{PE}_\omega = f(\omega)$. (3) (Current) kinetic energy (KE): KE is a nonnegative number and it helps the molecule escape from local optimums. There is a central energy buffer implemented in CRO. The energy in CRO may accord with energy conservation and can be exchanged between molecules and the buffer.

Four kinds of elementary reactions may happen in CRO, which are defined as below.

(1) On-wall ineffective collision: on-wall ineffective collision is a unimolecule reaction with only one molecule. In this reaction, a molecule ω is allowed to change to another one ω', if their energy values accord with the following inequality:

$$\mathrm{PE}_\omega + \mathrm{KE}_\omega \geq \mathrm{PE}_{\omega'}; \tag{1}$$

after this reaction, KE will be redistributed in CRO. The redundant energy with the value $\mathrm{KE}_{\omega'} = (\mathrm{PE}_\omega + \mathrm{KE}_\omega - \mathrm{PE}_{\omega'}) \times t$ will be stored in the central energy buffer. Parameter t is a random number from KELossRate to 1 and KELossRate, a system parameter set during the CRO initialization, is the KE loss rate less than 1.

(2) Decomposition: decomposition is the other unimolecule reaction in CRO. A molecule ω may decompose into two new molecules, ω_1' and ω_2', if their energy values accord with inequality (2), in which buf denotes the energy in the buffer, representing the energy interactions between molecules and the central energy buffer:

$$\mathrm{PE}_\omega + \mathrm{KE}_\omega + \mathrm{buf} \geq \mathrm{PE}_{\omega_1'} + \mathrm{PE}_{\omega_2'}; \tag{2}$$

after this reaction, buf is updated by (3) and the KEs of ω_1' and ω_2' are, respectively, computed as (4) and (5), where $\mathrm{Edecomp} = (\mathrm{PE}_\omega + \mathrm{KE}_\omega) - (\mathrm{PE}_{\omega_1'} + \mathrm{PE}_{\omega_2'})$ and $\mu 1$, $\mu 2$, $\mu 3$, $\mu 4$ is a number randomly selected from the range of $[0, 1]$. Consider

$$\mathrm{buf} = \mathrm{Edecomp} + \mathrm{buf} - \left(\mathrm{PE}_{\omega_1'} + \mathrm{PE}_{\omega_2'}\right), \tag{3}$$

$$\mathrm{KE}_{\omega_1'} = \left(\mathrm{Edecomp} + \mathrm{buf}\right) \times \mu 1 \times \mu 2, \tag{4}$$

$$\mathrm{KE}_{\omega_2'} = \left(\mathrm{Edecomp} + \mathrm{buf} - \mathrm{KE}_{\omega_1'}\right) \times \mu 3 \times \mu 4. \tag{5}$$

(3) Intermolecular ineffective collision: intermolecular ineffective collision is an intermolecule reaction with two molecules. Two molecules, ω_1 and ω_2, may change to two new molecules, ω_1' and ω_2', if their energy values accord with the following inequality:

$$\mathrm{PE}_{\omega_1} + \mathrm{PE}_{\omega_2} + \mathrm{KE}_{\omega_1} + \mathrm{KE}_{\omega_2} \geq \mathrm{PE}_{\omega_1'} + \mathrm{PE}_{\omega_2'}; \tag{6}$$

after this reaction, the KEs of ω_1' and ω_2', $\mathrm{KE}_{\omega_1'}$ and $\mathrm{KE}_{\omega_2'}$, will share the spare energy Eintermole calculated by (7). $\mathrm{KE}_{\omega_1'}$ and $\mathrm{KE}_{\omega_2'}$ are computed as (8)

and (9), respectively, where $\mu 1$ is a number randomly selected from the range of [0, 1]. Consider

$$\text{Eintermole} = \left(\text{PE}_{\omega_1} + \text{PE}_{\omega_2} + \text{KE}_{\omega_1} + \text{KE}_{\omega_2}\right) - \left(\text{PE}_{\omega_1'} + \text{PE}_{\omega_2'}\right), \tag{7}$$

$$\text{KE}_{\omega_1'} = \text{Eintermole} \times \mu 1, \tag{8}$$

$$\text{KE}_{\omega_2'} = \text{Eintermoler} \times (1 - \mu 1). \tag{9}$$

(4) Synthesis: synthesis is also an intermolecule reaction. Two molecules, ω_1 and ω_2, may be combined to a new molecule, ω', if their energy values accord with inequality (10). The KE of ω' is computed as (11):

$$\text{PE}_{\omega_1} + \text{PE}_{\omega_2} + \text{KE}_{\omega_1} + \text{KE}_{\omega_2} \geq \text{PE}_{\omega'}, \tag{10}$$

$$\text{KE}_{\omega'} = \text{PE}_{\omega_1} + \text{PE}_{\omega_2} + \text{KE}_{\omega_1} + \text{KE}_{\omega_2} - \text{PE}_{\omega'}. \tag{11}$$

The canonical CRO works as follows. Firstly, the initialization of CRO is to set system parameters, such as PopSize (the size of the molecules), KELossRate, InitialKE (the initial energy of molecules), buf (initial energy in the buffer), and MoleColl (MoleColl is a threshold value to determine whether to perform a unimolecule reaction or an intermolecule reaction). Then the CRO processes a loop. In each iteration, whether to perform a unimolecule reaction or an intermolecule reaction is first decided in the following way. A number ε is randomly selected from the range of [0, 1]. If ε is bigger than MoleColl, a unimolecule reaction will be chosen, or an intermolecular reaction is to occur. If it is a unimolecular reaction, a parameter θ as a threshold value is used to guide the further choice of on-wall collision or decomposition. NumHit is the parameter used to record the total collision number of a molecule. It will be updated after a molecule undergoes a collision. If the NumHit of a molecule is larger than θ, a decomposition will then be selected. Similarly, a parameter ϑ is used to further decide selection of an intermolecule collision reaction or a synthesis reaction. ϑ specifies the least KE of a molecule. Synthesis reaction will be chosen when both KEs of the molecules ω_1 and ω_2 are less than ϑ, or intermolecular ineffective collision reaction will take place. When the stopping criterion satisfies (e.g., a better solution cannot be found after a certain number of consecutive iterations), the loop will be stopped and the best solution is just the molecule that possesses the lowest PE.

4. Models

This section discusses the system, application, and task scheduling model assumed in this work. The definition of the notations can be found in the Notations section.

4.1. System Model. In this paper, there are multiple heterogeneous processors in the target system, which are presented by $P = \{p_i \mid i = 1, 2, 3, \ldots, |P|\}$. They are fully interconnected with high speed network. Each task in a DAG can only be executed on one processor on heterogeneous system. The edges of the graph are labeled with communication cost that should be taken into account if its start and end tasks are executed on different processors. The communication cost is zero when the same processor is assigned to two communicating modules.

We assume a static computing system model in which the constrained relations and the execution costs of tasks are known a priori and the execution and communication can be performed simultaneously by the processors. In this paper, the heterogeneity is represented by $\text{EC}_{P_r}(w)$, which means the execution cost of a node w using processor P_r. As the assumption of the MHM model, the heterogeneity in the simulations is set as follows to make a processor have different speed for different tasks. The value of each $\text{EC}_{P_r}(w)$ is randomly chosen within the scope of $[1 - g\%, 1 + g\%]$ by using a parameter g ($g \in (0, 1)$). Therefore, the heterogeneity level can be formulated as $(1 + g\%)/(1 - g\%)$. g is set as the value that makes the heterogeneity level 2 in this paper unless otherwise specified.

4.2. Application Model. In DAG scheduling, finding optimal schedules is to find the scheduling solution with the minimum schedule length. The schedule length encompasses the entire execution and communication cost of all the modules and is also termed as makespan. In this paper, the task scheduling problem is to map a set of tasks to a set of processors, aiming at minimizing the makespan. It takes as input a directed acyclic graph DAG = (V, E), with $|V|$ nodes representing tasks, and $|E|$ edges representing constrained relations among the tasks. $V = (v_1, v_2, \ldots, v_i, \ldots, v_{|V|})$ is a node sequence in which the hypothetical entry node (with no predecessors) v_1 and end node (with no successors) $v_{|V|}$, respectively, represent the beginning and the end of execution. The execution cost value of v_i on processor p_k is denoted as $\text{EC}_{p_k}(v_i)$, and the average computation cost of v_i, denoted as $\overline{W(v_i)}$, can be calculated by (12). The parameter for the amounts of computing power available at each node in a heterogeneous system and its heterogeneous level value is given in the 5th paragraph of Section 6 and Table 1.

$E = \{E_i \mid i = 1, 2, 3, \ldots, |E|\}$ is an edge set in which $E_i = (\text{ev}_s, \text{ev}_e, \text{ew}_{s,e})$, with ev_s & $\text{ev}_e \in \{v_1, v_2, \ldots, v_{|V|}\}$ representing its start and end nodes, and the value of communication cost between ev_s and ev_e is denoted as $\text{ew}_{s,e}$. The DAG topology of an exemplar application model and system model is shown in Figures 1 and 2, respectively.

Consider

$$\overline{W(v_i)} = \sum_{k=1}^{|P|} \frac{\text{EC}_{p_k}(v_i)}{|P|}. \tag{12}$$

The constrained-critical-path sequence of DAG = (V, E) is denoted as $\text{CCP} = (\text{CCP}_1, \text{CCP}_2, \ldots, \text{CCP}_{|\text{CCP}|})$ with $\text{CCP}_i = (\text{cv}_{i,1}, \text{cv}_{i,2}, \ldots, \text{cv}_{i,|\text{CCP}_i|})$ in which the set $\{\text{cv}_{i,1}, \text{cv}_{i,2}, \ldots, \text{cv}_{i,|\text{CCP}_i|}\} \subseteq \{v_1, v_2, \ldots, v_{|V|}\}$.

The start time of the task v_i on processor p_k is denoted as $\text{ST}_{p_k}(v_i)$, which can be calculated using (13), where $\text{Pred}(v_i)$ is the set of the predecessors of the task v_i. And the earliest finish

```
(1)  for i = 1; i ≤ |V|; i++
(2)        for each CCP_j in molecule CCPS
(3)            for each cv_k in CCP_j
(4)                v_i = cv_k;
(5)                f_i = 0;
(6)                p_i = sp_j;
(7)                Generate a new tuple (v_i, f_i, p_i)
(8)            end for
(9)        end for
(10) end for
(11) Generate a new reaction molecule S = ((v_1, f_1, p_1), (v_2, f_2, p_2), ..., (v_|V|, f_|V|, p_|V|));
(12) for each (v_i, f_i, p_i) in reaction molecule S
(13)       find the first successor Succ(v_i) in DAG from i to the end;
(14)       for each v_j ∈ (v_i, Succ(v_i))
(15)           find the first predecessor v_k = Pred(v_j) from Succ(v_i) to the begin in reaction molecule S;
(16)               if k < i
(17)                   interchanged position of (v_i, f_i, p_i) and (v_j, f_j, p_j) in reaction molecule S;
(18)               end if
(19)       end for
(20) end for
(21) for each p_i in reaction molecule S to randomly change;
(22)        change p_i randomly
(23) end for
(24) return S;
```

Algorithm 4: ConvertMole(CCPS) converting a CCPS to an S.

```
(1) slength = 0;
(2) for each node v in S = ((v_1, f_1, p_1), (v_2, f_2, p_2), ..., (v_|V|, f_|V|, p_|V|)) do
(3)    Calculate the start time of predecessor node pv of v
       ST_{p_v}(v, pv) = max((EFT_pv + CM(v, p_v, pv, p_pv)), AT_{P_r});
(4)    Find the finish time of v
       EFT_{p_v}(v) = max((ST_{p_v}(v, pv)_{∀pv∈Pred(v)}) + EC_{p_v}(v));
(5)    if slength < EFT_{p_v}(v)
(6)        update scheduling length
           slength = EFT_{p_v}(v);
(7)    end if
(8) end for
(9) return slength;
```

Algorithm 5: Fit(S) calculating the fitness value of a molecule and the processor allocation optimization.

time of the task v_i on processor p_k is denoted as $\mathrm{EFT}_{p_k}(v_i)$, which can be calculated using (14):

$$\mathrm{ST}_{p_k}(v_i) = \begin{cases} 0, & v_i = v_1 \\ \max\limits_{v_j \in \mathrm{Pred}(v_i)} \mathrm{EFT}_{p_k}(v_i), & p_k = p_m \\ \max\limits_{v_j \in \mathrm{Pred}(v_i)} \left(\mathrm{EFT}_{p_m}(v_j) + \mathrm{ew}_{j,i}\right), & p_k \neq p_m \end{cases} \tag{13}$$

$$\mathrm{EFT}_{p_k}(v_i) = \mathrm{ST}_{p_k}(v_i) + \mathrm{EC}_{p_k}(v_i). \tag{14}$$

The communication to computation ratio (CCR) can be used to indicate whether a DAG is communication intensive or computation intensive. For a given DAG, it is computed by the average communication cost divided by the average computation cost on a target computing system. The computation can be formulated as follows:

$$\mathrm{CCR} = \frac{\sum_{(v_i, v_j, \mathrm{ew}_{i,j}) \in E} \mathrm{ew}_{i,j}}{\overline{W(v_i)}}. \tag{15}$$

5. Design of TMSCRO

TMSCRO mimics the interactions of molecules in chemical reactions with the concepts of molecule, atoms, molecular structure, and energy of a molecule. The structure of a molecule is unique, which represents the atom positions in a molecule. The interactions of molecules in four kinds of basic

```
(1)  Initialize PopSize, KELossRate, MoleColl and InitialKE, θ and ϑ;
(2)  Call Algorithm 2 to generate the initial population of TMSCRO, CROPop;
(3)  Call Algorithm 3 to calculate PE of each molecule in CROPop;
(4)  while the stopping criteria is not met do
(5)     Generate ε ∈ [0, 1];
(6)     if ε > MoleColl
(7)        Select a reaction molecule S from CROPop randomly;
(8)        if ((NumHit_S − MinHit_S) > θ) & (S ≠ InitS)
(9)           Call DecompT to generate new molecules S'_1 and S'_2;
(10)          Call Algorithm 3 to calculate PE_{S'_1} and PE_{S'_2};
(11)          if Inequality (2) holds
(12)             Remove S from CROPop;
(13)             Add S'_1 and S'_2 to CROPop;
(14)          end if
(15)       else
(16)          Call OnWallT to generate a new molecules S';
(17)          Call Algorithm 3 to calculate PE_{S'};
(18)          If (S = InitS)
(19)             InitS = S';
(20)          end if
(21)          Remove S from CROPop;
(22)          Add S' to CROPop;
(23)       end if
(24)    else
(25)       Select two molecules S_1 and S_2 from CROPop randomly;
(26)       if (KE_{S_1} < ϑ) & (KE_{S_2} < ϑ) & (S_1 ≠ InitS) & (S_2 ≠ InitS)
(27)          Call SynthT to generate a new molecule S';
(28)          Call Algorithm 3 to calculate PE_{S'};
(29)          if Inequality (10) holds
(30)             Remove S_1 and S_2 from CROPop;
(31)             Add S' to CROPop;
(32)          end if
(33)       else
(34)          Call IntermoleT to generate two new molecules S'_1 and S'_2;
(35)          Call Algorithm 3 to calculate PE_{S'_1} and PE_{S'_2};
(36)          if (S_1 = InitS)
(37)             InitS = S'_1;
(38)          else if (S_2 = InitS)
(39)             InitS = S'_2;
(40)          end if
(41)          Remove S_1 and S_2 from CROPop;
(42)          Add S'_1 and S'_2 to CROPop;
(43)       end if
(44)    end if
(45) end while
(46) return the molecule with the lowest PE in CROPop;
```

Algorithm 6: TMSCRO(DAG) The TMSCRO outline(framework).

chemical reactions, on-wall ineffective collision, decomposition, intermolecular ineffective collision, and synthesis, aim to transform to the molecule with more stable states which has lower energy. In DAG scheduling, a scheduling solution including a task and processor allocation corresponds to a molecule in TMSCRO. This paper also designs the operators on the encoded scheduling solutions (tuple arrays). These designed operators correspond to the chemical reactions and change the molecular structures. The arrays with different tuples represent different scheduling solutions, and we can calculate the corresponding makespan of the scheduling solution. A scheduling solution makespan corresponds to the energy of a molecule.

In this section, we first present the data pretreatment of the TMSCRO. After the presentation of the encoding of scheduling solutions and the fitness function used in the TMSCRO, we present the design of four elementary chemical reaction operators in each part of the TMSCRO. Finally, we outline the framework of the TMSCRO scheme and discuss a few important properties in TMSCRO.

Table 1: Specific terms and their usage for the CEFT algorithm.

$EC_{P_r}(w)$	Execution cost of a node w using processor P_r
$CM(w, P_r, v, P_x)$	Communication cost from node v to w, if P_x has been assigned to node v and P_r is assigned to node w
$ST_{P_r}(w, v)$	Possible start time of node w which is assigned the processor P_r with the v node being any predecessor of w which has already been scheduled
$EFT_{P_r}(w)$	Finish time of node w using processor P_r
$AEFT_w$	Actual finish time of node w
$CEFT_{P_r}(CCP_j)$	Finish time of the constrained critical path Q_j when processor P_r is assigned to it
AT_{P_r}	Availability time of P_r
Pred(w)	Set of predecessors of node w
Succ(w)	Set of successors of node w
AEC(w)	Average execution cost of node w

Table 2: CCP corresponding to the DAG as shown in Figure 1(1).

i	CCP_i
1	A-B-D
2	C-G
3	F
4	E
5	H
6	I
7	J

Table 3: Configuration parameters for the simulation of TMSCRO.

Parameter	Value
InitialKE	1000
θ	500
ϑ	10
Buffer	200
KELossRate	0.2
MoleColl	0.2
PopSize	10
g	0.33
Number of runs	50

Table 4: Configuration parameters for the Gaussian elimination graphs.

Parameter	Possible values
CCR	{0.1, 0.2, 1, 2, 5}
Number of processors	{4, 8, 16, 32}
Number of tasks	27

5.1. Molecular Structure, Data Pretreatment, and Fitness Function. This subsection first presents the encoding of scheduling solutions (i.e., the molecular structure) and data pretreatment, respectively. Then we give the statement of the fitness function for optimization designed in TMSCRO.

5.1.1. Molecular Structure and Data Pretreatment. A reasonable initial population in CRO-based methods may increase the scope of searching over the fitness function [20] to support faster convergence and to result in a better solution. Constrained critical paths (CCPs) can be seen as the classification of task sequences constructed by constrained earliest finish time (CEFT) algorithm, which takes into account all factors in DAG (i.e., the average of each task execution cost, the communication costs, and the graph topology). Therefore, TMSCRO utilizes the CCPs to create a reasonable initial population based on a broad view of DAG.

The data pretreatment is to generate the CCPDAG from DAG and to construct CCPS for the initialization of TMSCRO. The CCPDAG is a directed acyclic graph with |CCP| nodes representing constrained critical paths (CCP_s), two virtual nodes (i.e., start and end) representing the beginning and exit of execution, respectively, and |CE| edges representing dependencies among the nodes. The edges of CCPDAG are not labeled with communication overhead which is different from DAG. The data pretreatment includes two steps.

(1) The CCP and the processor allocation of each element of CCP in DAG can be obtained by executing CEFT and the first initial CCP solution, InitCCPS = $((CCP_1, sp_1), (CCP_2, sp_2), \ldots, (CCP_{|CCP|}, sp_{|CCP|}))$, can also be got, in which $((CCP_i, sp_i))$ is sorted as the generated order of CCP_i and sp_i is processor assignment of CCP_i after executing CEFT. Consider the graph as shown in Figure 1; the resulting CCPs are indicated in Table 2.

(2) After the execution of CEFT for DAG, the CCPDAG is generated with the input of CCP and DAG. A detailed description is given in Algorithm 2.

As shown in Algorithm 1, the edge E_i of DAG with the start node CCP_s and the end node CCP_e is obtained in each loop (line 1). BelongCCP(v_i) represents which CCP_j in CCPv_i belongs to (line 2 and line 3). If CCP_s and CCP_e are different CCPs and there is no edge between them (line 4), then the edge between CCP_s and CCP_e is generated (line 5). Finally, the nodes, start and end, and the edges among them and CCP nodes are added (line 7, line 8, and line 9). Consider the DAG as shown in Figure 1 and the CCP as indicated in Table 1. The resulting CCPDAG is shown in Figure 3.

In this paper, there are two kinds of molecular structures of TMSCRO, CCPS, and S. CCP molecular structure CCPS is just used in the initialization of TMSCRO, which can be formulated as in (16). Whereas the reaction molecular structure S converted from CCPS is used to participate in the elementary reaction of TMSCRO. In CCPS, $((CCP_i, sp_i))$s are sorted as the topology of CCPDAG in which CCP_i is constrained critical path (CCP), and sp_i is the processor assigned to CCP_i. $|CCP| \leq |V|$ because the number of elements in each $SCCP_i$ is greater than or equal to one. A reaction molecule S can be formulated as in (17), which consists of an array of atoms (i.e., tuples) representing a solution of DAG scheduling problem. A tuple includes three integers v_i, f_i, and p_i. The reaction molecular structure S is

Table 5: The experiment results for the Gaussian elimination graph under different processors, CCR = 0.2.

The number of processors	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
4	112.2	122.227	109.9	109.31	109.2	109.9	0.2473
8	112.2	112.648	108.9	107.83	107.1	108.9	0.9613
16	80.4	92.354	77.5	76.62	76.3	78.9	1.6696
32	79.64	85.454	77.5	76.62	76.1	78.9	1.7201

Table 6: The experiment results for the Gaussian elimination graph under different CCRs; the number of processors is 8.

CCR	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
0.1	108.2	110.312	106.78	105.04	104.76	106.6	1.7271
0.2	112.2	112.648	108.9	107.83	107.1	108.9	0.9613
1	120.752	124.536	115.63	114.717	114.3	115.4	0.3787
2	207.055	197.504	189.4	188.303	188.1	188.75	0.1522
5	263.8	263.8	252.39	250.671	250.3	251.79	0.9178

encoded with each integer in the permutation representing a task in DAG, the constraint relationship between a tuple and the one before it, and the processor p_i. In each reaction molecular structure S, v_i represents a task in DAG and $(v_1, v_2, \ldots, v_{|V|})$ is a topological sequence of DAG. In S, if v_A of the tuple A, which is before tuple B, is the predecessor of v_B of tuple B in DAG, the second integer of tuple B, f_B, will be 1, or it will be 0. p_i represents the processor allocation of each v_i in the tuple. The sequence of the tuples in a reaction molecular structure S represents the scheduling order of each task in DAG:

$$\begin{aligned}\text{CCPS} &= \left((\text{CCP}_1, \text{sp}_1), (\text{CCP}_2, \text{sp}_2), \ldots, \left(\text{CCP}_{|\text{CCP}|}, \text{sp}_{|\text{CCP}|}\right)\right), \quad (16)\\ S &= \left((v_1, f_1, p_1), (v_2, f_2, p_2), \ldots, (v_{|V|}, f_{|V|}, p_{|V|})\right). \quad (17)\end{aligned}$$

5.1.2. Fitness Function. The initial molecule generator is used to generate the initial solutions for TMSCRO to manipulate. The first molecule InitS is converted from InitCCPS. Part three sp_i of each tuple is generated by a random perturbation in the first InitCCPS. A detailed description is given in Algorithms 3 and 4 and presents how to convert a CCPS to an S.

Potential energy (PE) is defined as the objective function (fitness function) value of the corresponding solution represented by S. The overall schedule length of the entire DAG, namely, makespan, is the largest finish time among all tasks, which is equivalent to the actual finish time of the end node in DAG. For the DAG scheduling problem by TMSCRO, the goal is to obtain the scheduling that minimizes makespan and ensure that the precedence of the tasks is not violated. Hence, each fitness function value is defined as

$$\text{PE}_S = \text{makespan} = \text{Fit}(S). \quad (18)$$

Algorithm 5 presents how to calculate the value of the optimization fitness function Fit(S).

Table 7: Configuration parameters for the molecular dynamics code graphs.

Parameter	Possible values
CCR	{0.1, 0.2, 1, 2, 5}
Number of processors	{4, 8, 16, 32}
Number of tasks	41

5.2. Elementary Chemical Reaction Operators. This subsection presents four elementary chemical reaction operators for sequence optimization and processor allocation optimization designed in TMSCRO, including on-wall collision, decomposition, intermolecular collision, and synthesis.

5.2.1. On-Wall Ineffective Collision. In this paper, the operator, OnWallT, is used to generate a new molecule S' from a given reaction molecule S for optimization. OnWallT works as follows. (1) The operator randomly chooses a tuple (v_i, f_i, p_i) with $f_i = 0$ in S and then exchanges the positions of (v_i, f_i, p_i) and $(v_{i-1}, f_{i-1}, p_{i-1})$. (2) f_{i-1}, f_i and f_{i+1} in S are modified as defined in the last paragraph of Section 5.1.1. (3) The operator changes p_i randomly. In the end, the operator generates a new molecule S' from S as an intensification search. Figures 4 and 5 show the example which is the molecule corresponding to the DAG as shown in Figure 1(2).

Table 8: The experiment results for the molecular dynamics code graph under different processors, CCR = 1.0.

The number of processors	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
4	149.205	142.763	139.51	138.13	137.87	138.6	0.1749
8	131.031	122.265	118.8	116.9	116.2	117.33	0.2764
16	124.868	115.584	113.52	113.36	113.1	113.43	0.0237
32	120.047	103.784	102.617	101.29	101.023	101.47	0.0442

Table 9: The experiment results for the molecular dynamics code graph under different CCRs; the number of processors is 16.

CCR	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
0.1	82.336	90.136	80.53	77.781	77.3	78.9	0.9459
0.2	82.356	87.504	80.53	78.704	78.21	79.13	0.2002
1	124.868	115.584	113.52	113.36	113.1	113.43	0.0237
2	216.735	174.501	167.612	164.7	164.32	164.91	0.0742
5	274.7	274.7	265.8	262.173	262.022	262.6	0.1344

Table 10: Configuration parameters for random graphs.

Parameter	Possible values
CCR	{0.1, 0.2, 1, 2, 5, 10}
Number of processors	{4, 8, 16, 32}
Number of tasks	{10, 20, 50}

5.2.2. Decomposition. In this paper, the operator, DecompT, is used to generate new molecules S_1' and S_2' from a given reaction molecule S. DecompT works as follows. (1) The operator randomly chooses two tuples (tuples) (v_i, f_i, p_i) with $f_i = 0$ and (v_t, f_t, p_t) with $f_t = 0$ in S and then finds the tuple with the first predecessor of (v_i, f_i, p_i), such as (v_j, f_j, p_j), from the selection position to the beginning of reaction molecule S. (2) A random number $k \in [j+1, i-1]$ is generated, and the tuple (v_i, f_i, p_i) is stored in a temporary variable temp, and then from the position $i-1$, the operator shifts each tuple by one place to the right position until a position k. (3) The operator moves the tuple temp to the position k. The rest of the tuples in S_1' are the same as those in S. (4) f_i, f_{i+1} and f_k in S are modified as defined in the last paragraph of Section 5.1.1. (5) The operator generates the other new molecule S_2' as the former steps. The only difference is that, in step 2, we use (v_t, f_t, p_t) instead of (v_i, f_i, p_i). (6) The operator keeps the tuples in S_1', which is at the odd position in S, and retains the tuples in S_2', which is at the even position in S, and then changes the remaining $p_x s$ of tuples in S_1', and S_2', randomly. In the end, the operator generates two new molecules S_1' and S_2' from S as a diversification search. Figures 6 and 7 show the example which is the molecule corresponding to the DAG as shown in Figure 1(2).

5.2.3. Intermolecular Ineffective Collision. In this paper, the operator, IntermoleT, is used to generate new molecules S_1' and S_2' from given molecules S_1 and S_2. This operator first uses the steps in OnWallT to generate S_1' from S_1, and then the operator generates the other new molecule S_2' from S_2 in similar fashion. In the end, the operator generates two new molecules S_1' and S_2' from S_1 and S_2 as an intensification search. Figures 8 and 9 show the example which is the molecule corresponding to the DAG as shown in Figure 1(2).

5.2.4. Synthesis. In this paper, the operator, SynthT, is used to generate a new molecule S' from given molecules S_1 and S_2 for optimization. SynthT works as follows. (1) If $|V|$ is plural, then the integer $i = |V|/2$; else $i = (|V| + 1)/2$. (2) S_1 and S_2 are cut off at the position i to become the left and right segments. (3) The left segments of S' are inherited from the left segments of S_1, randomly. (4) Each tuple in the right segments of S' comes from the tuples in S_2 that do not appear in the left segment of S', with their f_x modified as defined in the last paragraph of Section 5.1.1 as well. (5) The operator keeps the tuples in S', which are at the same position in S_1 and S_2 with the same $p_x s$, and then changes the remaining $p_y s$ in S', randomly. As a result, the operator generates S' from S_1 and S_2 as a diversification search. Figures 10 and 11 show the example which is the molecule corresponding to the DAG as shown in Figure 1(2).

5.3. The Framework and Analysis of TMSCRO. The framework of TMSCRO is shown as an outline to schedule a DAG job in Algorithm 6 and the output of Algorithm 6 is just the resultant near-optimal solution for the corresponding DAG scheduling problem. In this framework, TMSCRO first

Table 11: The experiment results for the random graph under different task numbers, CCR = 10; the number of processors is 32.

The number of tasks	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
10	73	67	65.1	62.2
20	148.9	143.9	139.421	136.8
50	350.7	341.7	334.17	331.9

Table 12: The experiment results for the random graph under different processors, CCR = 0.2; the number of tasks is 50.

The number of processors	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
4	167.12	178.023	159.234	157.63	157.12	158.3	0.3923
8	136.088	145.649	128.17	127.178	127.06	127.7	0.1949
16	119.292	125.986	115.9	114.33	114.1	115.2	0.4753
32	111.866	120.065	108.7	108.71	108.31	108.9	0.0733

initializes the process. Then, the process enters a loop. In each iteration, one of the elementary chemical reaction operators for optimization is performed to generate new molecules and PE of newly generated molecules will be calculated. The whole working of TMSCRO for DAG scheduling on heterogeneous problem is as presented in the last paragraph in Section 3.2. However, InitS is considered to be a super molecule [6], so it will be tracked and only participates in on-wall ineffective collision and intermolecular ineffective collision to explore as much as possible the solution space in its neighborhoods and the main purpose is to prevent InitS from changing dramatically. The iteration repeats until the stopping criteria are met. The stopping criteria may be set based on different parameters, such as the maximum amount of CPU time used, the maximum number of iterations performed, an objective function value less than a predefined threshold obtained, and the maximum number of iterations performed without further performance improvement. The stopping criterion of TMSCRO in the experiments of this paper is that the makespan is not changed after 5000 consecutive iterations in each loop. The time complexity of TMSCRO is $O(\text{iters} \times [2 \times (|V|^2 + |E| \times |P|)]$, where iters is the number of iterations in TMSCRO, respectively.

It is very difficult to theoretically prove the optimality of the CRO (as well as DMSCRO and TMSCRO) scheme [37]. However, by analyzing the molecular structure, chemical reaction operators, and the operational environment in TMSCRO, it can be shown to some extent that TMSCRO scheme has the advantage of three points in comparison with GA, SA, and DMSCRO.

First, just like DMSCRO, TMSCRO enjoys the advantages of GA and SA to some extent by analyzing the chemical reaction operators designed in TMSCRO and the operator environment of TMSCRO: (1) the OnWallT and IntermoleT in TMSCRO exchange the partial structure of two different molecules like the crossover operator in GA. (2) The energy conservation requirement in TMSCRO is able to guide the searching of the optimal solution in a similar way as the Metropolis Algorithm of SA guides the evolution of the solutions in SA. Second, constrained earliest finish time (CEFT) algorithm constructs constrained critical paths (CCPs) by taking into account a broader view of the input DAG [5]. TMSCRO applies CEFT and CCPDAG to the data pretreatment and utilizes CCPs in the initialization of TMSCRO to create a more reasonable initial population than DMSCRO for accelerating convergence, because a wide distributed initial population in CRO-based methods may increase the scope of searching over the fitness function [20] to support faster convergence and to result in a better solution. Moreover, to some degree, InitS is also similar to the super molecule in super molecule-based CRO or the "elite" in GA [6]. However, the "elite" in GA is usually generated from two chromosomes, while InitS is based on the whole input DAG by executing CEFT. Third, the operators with the molecular structure in TMSCRO are designed more reasonably than DMSCRO. In CRO-based algorithm, the operators of on-wall collision and intermolecular collision are used for intensifications, while the operators of decomposition and synthesis are for diversifications. The better the operator can get the better the search results of intensification and diversification are. This feature of CRO is very important, which gives CRO more opportunities to jump out of the local optimum and explore the wider areas in the solution space. In TMSCRO, the operators of OnWallT and IntermoleT every time only exchange the positions of one tuple and its former neighbor in the molecule with better capability of intensification on sequence optimization than DMSCRO, of which the reaction operators, OnWall (ω_1) and Intermole (ω_1, ω_2) [37] (ω_1 and ω_2 are big molecules in DMSCRO), may change the task sequence(s) dramatically. Moreover, under the consideration that the optimization includes not only sequence but also processor assignment optimization,

Table 13: The experiment results for the random graph under different processors, CCR = 1.0; the number of tasks is 50.

The number of processors	HEFT_B (the average makespan)	HEFT_T (the average makespan)	DMSCRO (the average makespan)	TMSCRO (the average makespan)	TMSCRO (the best makespan)	TMSCRO (the worst makespan)	TMSCRO (the variance of resultant makespans)
4	178.662	175.52	168.12	167.703	167.42	168	0.0857
8	138.572	136.47	131.8	131.451	131.1	131.9	0.178
16	125.772	124.31	122.91	122.32	122.1	122.432	0.0233
32	117.11	116.4	114.124	113.127	112.9	113.54	0.1348

Table 14: The experiment results for the random graph under different task CCRs, the number of tasks is 50.

CCR	The number of processors is 4	The number of processors is 8	The number of processors is 16	The number of processors is 32
0.1	156.97	115.724	110.3	101.87
0.2	157.63	127.178	114.33	108.71
1	167.703	131.451	122.32	113.127
2	294.042	289.878	273.375	269.514
5	473.5	467.61	429.13	428.13

all reaction operators in TMSCRO can change the processor assignment, but DMSCRO has only two reactions, on-wall and synthesis [37], for processor assignment optimization. On the one hand, TMSCRO has 100% probability of searching the processor assignment solution space by four elementary reactions, with better capability of diversification and intensification on processor assignment optimization than DMSCRO, of which the chance to search this kind of solution space is only 50%. On the other hand, the division of diversification and intensification of four reactions in TMSCRO is very clear; however, this is not in DMSCRO. In each iteration, the diversification and intensification search in TMSCRO have the same probability to be conducted, whereas the possibility of diversification or intensification search in DMSCRO is uncertainty. This design enhances the ability to get better rapidity of convergence and search result in the whole solution space, which is demonstrated by the experimental results in Section 6.3.

6. Simulation and Results

The simulations have been performed to test TMSCRO scheduling algorithm in comparison with heuristic (HEFT_B and HEFT_T) [8] for DAG scheduling and with two metaheuristic algorithms, double molecular structure-based chemical reaction optimization (DMSCRO) [37], by using two sets of graph topology such as the real world application (Gaussian elimination and molecular dynamics code) and randomly generated application. The task graph for Gaussian elimination for input matrix of size 7 is shown in Figure 12, whereas a molecular dynamics code graph is shown in Figure 13. Figure 14 shows a random graph with 10 nodes. The baseline performance is the makespan obtained by DMSCRO.

Considering that HEFT_B and HEFT_T have better performance than other heuristics algorithms for DAG scheduling on heterogeneous computing systems, as proposed in the 8th paragraph in Section 2.1, these two algorithms are used to be the representatives of heuristics in the simulation. There are three reasons why we regard the makespan performance of DMSCRO [37] scheduling as the baseline performance. (1) So far as we know, DMSCRO is the only one CRO-based algorithm for DAG scheduling which takes into account the searching of the task order and processor assignment. (2) As discussed in the 3rd paragraph of Section 2.2, DMSCRO [37] has the closest system model and workload to that of TMSCRO. (3) In [37], CRO-based scheduling algorithm is considered as absorbing the strengths of SA and GA. However, the underlying principles and philosophies of SA are very different from DMSCRO, and because the DMSCRO is also proved to be more effective than genetic algorithm (GA) [15] as presented in [37], we just use DMSCRO to represent the metaheuristic algorithms. We propose to make a comparison between TMSCRO and DMSCRO to validate the advantages of TMSCRO over DMSCRO.

The performance has been evaluated by the parameter makespan. The makespan values plotted in the bar graph of makespan and the chart of converge trace are, respectively, the average result of 50 and 25 independent runs to validate the robustness of TMSCRO. The communication cost is calculated by using computation costs and the computation cost ratio (CCR) values. The computation can be formulated as in (17):

$$\text{Communication Cost} = \text{CCR} * \text{Computation Cost}. \quad (19)$$

All the suggested values for the other parameters of the simulation of TMSCRO and their values are listed in Table 3. These values are proposed in [20].

6.1. Real World Application Graphs. The real world application set is used to evaluate the performance of TMSCRO, which consists of two real world problem graph topologies, Gaussian elimination [22] and molecular dynamics code [19].

6.1.1. Gaussian Elimination. Gaussian elimination is a well-known method to solve a system of linear equations. Gaussian elimination converts a set of linear equations to the upper triangular form by applying elementary row operators on them systematically. As shown in Figure 12, the matrix size of the task graph of Gaussian elimination algorithm is 7, with 27 tasks in total. In [37], this DAG has been used for the simulation of DMSCRO, and we also apply it to the evaluation of TMSCRO in this paper. Under the consideration that graph structure is fixed, the variable parameters are only 22 the communication to computation ratio (CCR) value and the heterogeneous processor number. In the simulation, CCR values were set as 0.1, 0.2, 1, 2, and 5, respectively. Considering the identical operator is executed on each processor and the information communicated between heterogeneous processors is the same in Gaussian elimination, the execution cost of each task is supposed to be the same and all communication links have the same communication cost.

The parameters and their values of the Gaussian elimination graphs performed in the simulation are given in Table 4.

The makespan of TMSCRO, DMSCRO, HEFT_B, and HEFT_T under the increasing processor number is shown in Figure 15. As shown in Figure 15, it can also been seen that as the processor number increases, the average makespan declines, and the advantage of TMSCRO and DMSCRO over HEFT_B and HEFT_T also decreases, because when more computing nodes are contributed to run the same scale of tasks, less intelligent scheduling algorithms are needed in order to achieve good performance.

As the intelligent random search algorithms, TMSCRO and DMSCRO search a wider area of the solution space than HEFT_B, HEFT_T, or other heuristic algorithms, which narrow the search down to a very small portion of the solution space. This is the reason why TMSCRO and DMSCRO are more likely to obtain better solutions and outperform HEFT_B and HEFT_T.

The simulation results show that the performance of TMSCRO and DMSCRO is very similar to the fundamental reason that these algorithms are metaheuristic algorithms. Based on No-Free-Lunch Theorem in the field of metaheuristics, the performances of all well-designed metaheuristic search algorithms for optimal solution are the same, when averaged over all possible objective functions. The optimal solution will be gradually approached by a well-designed metaheuristic algorithm in theory, if it runs for long enough. The DMSCRO developed in [37] is well-designed, and we use it in the simulations of this paper. Therefore similar simulation results of the performances of TMSCRO and DMSCRO indicate that TMSCRO we developed is also well-designed. The detailed experiment result is shown in Table 5.

In Figure 15, the figure shows that TMSCRO is superior to DMSCRO slightly. There will be only one reason for it: the stopping criteria set in this simulation are that the makespan stays unchanged for 5000 consecutive iterations in the search loop. As discussed in the last paragraph of Section 5, all metaheuristic methods that search for optimal solutions are the same in performance when averaged over all possible objective functions. And these experimental stopping criteria make TMSCRO and DMSCRO run for long enough to gradually approach the optimal solution. Moreover, better convergence of TMSCRO makes it more efficient in searching good solutions than DMSCRO by running much less iteration times. More detailed experiment results in this regard will be presented in Section 6.3.

Figure 16 shows that the average makespan of these four algorithms increases rapidly under the CCR increasing. The reason for it is because as CCR increases, the application becomes more communication intensive, making the heterogeneous processors in the idle state for longer. As shown in Figure 16, TMSCRO and DMSCRO outperform HEFT_B and HEFT_T with the advantage being more obvious as CCR becomes larger. These experimental results suggest that, for communication-intensive applications, TMSCRO and DMSCRO can deliver more consistent performance and perform more effectively than heuristic algorithms, HEFT_B and HEFT_T, in a wide range of scenarios for DAG scheduling. The detailed experiment result is shown in Table 6.

6.1.2. Molecular Dynamics Code. Figure 13 shows the DAG of a molecular dynamics code as presented in [19]. As the experiment of Gaussian elimination, the structure of graph and the number of processors are fixed. The varied parameters are the number of heterogeneous processors and the CCR values which are used in our simulation are 0.1, 0.2, 1, 2, and 5.

The parameters and their values of the molecular dynamics code graphs performed in the simulation are given in Table 7.

As shown in Figures 18 and 19, under different heterogeneous processor number and different CCR values, the average makespans of TMSCRO and DMSCRO are over HEFT_B and HEFT_T, respectively. In Figure 17, it can be observed that, with the number of heterogeneous processors increasing, the average makespan decreases. The average makespan with respect to different CCR values is shown in Figure 18. The average makespan increases with the value of CCR increasing. The detailed experiment results are shown in Tables 8 and 9, respectively.

6.2. Random Generated Application Graphs. An effective mechanism to generate random graph for various applications is proposed in [42]. By using the probability for an edge between any two nodes, it can generate a random graph without incline towards a specific topology.

In the random graph generation of this mechanism, the topological order is used to guarantee the precedence constraints; that is, an edge exists between two nodes v_1 and v_2 only if $v_1 < v_2$. For probability pb, $\lfloor |V| * \text{pb} \rfloor$ edges are created from every node m to another node $(N_1 + (1/\text{pb}) * i) \bmod |V|$, where $1 \leq i \leq \lfloor |V| * \text{pb} \rfloor$, and $\lfloor V \rfloor$ is the total account of task nodes in DAG.

Table 15: Configuration parameters of convergence experiment for the Gaussian elimination graph.

Parameter	Value
CCR	0.2
Number of processors	8
Number of tasks	27

Table 16: Configuration parameters of convergence experiment for the molecular dynamics graph.

Parameter	Value
CCR	1
Number of processors	16
Number of tasks	41

The parameters and their values of the random graphs performed in the simulation are given in Table 10.

Figure 19 shows that TMSCRO always outperforms HEFT_B, HEFT_T, and DMSCRO with the number of tasks in a DAG increasing. The comparison of the average makespan of four algorithms under the increase of heterogeneous processor number is shown in Figures 20 and 21. As can be seen from these figures, the performance of TMSCRO is better than the other three algorithms in all cases. The reasons for these two figures are the same as those explained in Figure 15. The detailed experiment results are shown in Tables 11, 12, and 13, respectively.

As shown in Figure 22, it can be observed that the average makespan approached by TMSCRO increases rapidly with CCR values increasing. This may be because as CCR increases, the application becomes more communication intensive, making the heterogeneous processors in the idle state for longer. The detailed experiment results are shown in Table 14.

6.3. Convergence Trace of TMSCRO. The result of the experiments in the previous subsections is the final makespan obtained by TMSCRO and DMSCRO, showing that TMSCRO can obtain similar makespan performance as DMSCRO. Moreover, in some cases the final makespan achieved by TMSCRO is even better than that by DMSCRO after the stop criteria are satisfied. In this section, the change of makespan in the experiments as TMSCRO and DMSCRO progress during the search is demonstrated by comparing the convergence trace of these two algorithms. These experiments help further reveal the better performance of TMSCRO on convergence and can also help explain why the TMSCRO sometimes outperforms DMSCRO in some cases.

The parameters and their values of the Gaussian elimination, molecular dynamics code, and random graphs performed in the simulation are given in Tables 15, 16, and 17, respectively.

Figures 23 and 24, respectively, plot the convergence traces for processing Gaussian elimination and the molecular dynamics code. Figures 25, 26, and 27 show the convergence traces when processing the sets of randomly generated DAGs and each set contains the DAGs of 10, 20, and 50 tasks, respectively. These figures demonstrated that the makespan performance decreases quickly as both TMSCRO and DMSCRO progress and that the decreasing trends tail off when the algorithms run for long enough. These figures also show that, in most cases, the convergence traces of both algorithms are rather different even though the final makespans obtained by them are almost the same.

Table 17: Configuration parameters of convergence experiment for the random graphs.

Parameter	Values
CCR	{0.2, 1}
Number of processors	{8, 16}
Number of tasks	{10, 20, 50}

Table 18: The results of the statistical analysis over the average coverage rate at different sampling times of all the experiments (the threshold of **P** is set as **0.05**).

DAG	The value of P after Friedman test	Average convergence acceleration ratio
Gaussian elimination	7.10×10^{-8}	4.23%
Molecular dynamics code	2.54×10^{-8}	7.21%
Random graph with 10 tasks	4.26×10^{-8}	23.27%
Random graph with 20 tasks	3.48×10^{-8}	16.41%
Random graph with 50 tasks	2.58×10^{-8}	13.32%

The statistical analysis results over the average coverage rate at 5000 ascending sampling points from start time to end time of all the experiments are shown in Table 18 (the threshold of P is set as 0.05), which are obtained by Friedman test, and each experiment is carried out 25 times. We can find that the differences between two algorithms in performance are significant from a statistical point of view. The reason of it is because the super molecule makes TMSOCRO have a stronger convergence capability, especially early in each run. Moreover, the performance of TMSCRO on convergence is better than DMSCRO. Quantitatively, our records show that TMSCRO converges faster than DMSCRO by 12.89% on average in all the cases (by 23.27% on average in the best case).

In these experiments, the stopping criteria of the algorithms are that the algorithm stops when the makespan performance remains unchanged for a preset number of consecutive iterations in the search loop (in the experiments, it is 5000 iterations). In reality, the algorithms can also stop when the total processing time of it reaches a preset value (e.g., 180s). Moreover, both of TMSCRO and DMSCRO have the same initial population. In this case, the fact that TMSCRO outperforms DMSCRO on convergence means that the makespan achieved by TMSCRO could be much better than that by DMSCRO when the stopping criteria of the algorithm are satisfied. The reason for this can be

explained by the analysis presented in the last paragraph of Section 5.3.

7. Conclusion

In this paper, we developed a TMSCRO for DAG scheduling on heterogeneous systems based on chemical reaction optimization (CRO) method. With a more reasonable reaction molecular structure and four designed elementary chemical reaction operators, TMSCRO has a better ability on intensification and diversification search than DMSCRO, which is the only one CRO-based algorithm for DAG scheduling on heterogeneous systems as far as we know. Moreover, in TMSCRO, the algorithm constrained earliest finish time (CEFT) and constrained-critical-path directed acyclic graph (CCPDAG) are applied to the data pretreatment, and the concept of constrained paths (CCPs) is also utilized in the initialization. We also use the first initial molecule, InitS, to be a super molecule for accelerating convergence. As a metaheuristic method, the TMSCRO algorithm can cover a much larger search space than heuristic scheduling approaches. The experiments show that TMSCRO outperforms HEFT_B and HEFT_T and can achieve a higher speedup of task executions than DMSCRO.

In future work, we plan to extend TMSCRO by applying synchronous communication strategy to parallelize the processing of TMSCRO. This kind of design will divide the molecules into groups and each group of molecules is handled by a CPU or GPU. So, multiple groups can be manipulated simultaneously in parallel and molecules can also be exchanged among the CPUs or GPUs from time to time in order to reduce the time cost.

Notations

DAG = (V, E):	Input directed acyclic graph with $\|V\|$ nodes representing tasks, and $\|E\|$ edges representing constrained relations among the tasks
$V = (v_1, v_2, \ldots, v_{\|V\|})$:	Node sequence in which the hypothetical entry node (with no predecessors) v_1 and end node (with no successors) $v_{\|V\|}$, respectively, represent the beginning and end of execution
$E = \{E_i \mid i = 1, 2, 3, \ldots, \|E\|\}$:	Edge set in which $E_i = (\mathrm{ev}_s, \mathrm{ev}_e, \mathrm{ew}_{s,e})$, with ev_s & $\mathrm{ev}_e \in \{v_1, v_2, \ldots, v_{\|V\|}\}$ representing its start and end nodes, and the value of communication cost between ev_s and ev_e denoted as $\mathrm{ew}_{s,e}$
$P = \{p_i \mid i = 1, 2, 3, \ldots, \|P\|\}$:	Set of multiple heterogeneous processors in target system
$\mathrm{CCP} = (\mathrm{CCP}_1, \mathrm{CCP}_2, \ldots, \mathrm{CCP}_{\|\mathrm{CCP}\|})$:	Constrained-critical-path sequence of DAG = (V, E)
$\mathrm{CCP}_i = (\mathrm{cv}_{i,1}, \mathrm{cv}_{i,2}, \ldots, \mathrm{cv}_{i,\|\mathrm{CCP}_i\|})$:	Constrained critical path in which the set $\{\mathrm{cv}_{i,1}, \mathrm{cv}_{i,2}, \ldots, \mathrm{cv}_{i,\|\mathrm{CCP}_i\|}\} \subseteq \{v_1, v_2, \ldots, v_{\|V\|}\}$
CCPDAG:	Directed acyclic graph with |CCP| nodes representing CCPs, two virtual nodes (i.e., start and end) representing the beginning and exit of execution, respectively, and |CE| edges representing dependencies among all nodes
$\mathrm{CCPS} = ((\mathrm{CCP}_1, \mathrm{sp}_1), (\mathrm{CCP}_2, \mathrm{sp}_2), \ldots, (\mathrm{CCP}_{\|\mathrm{CCP}\|}, \mathrm{sp}_{\|\mathrm{CCP}\|}))$:	A CCP molecule used in the initialization of TMSCRO, in which sp_i is the processor assigned to the constrained-critical-path CCP_i
$S = ((v_1, f_1, p_1), (v_2, f_2, p_2), \ldots, (v_{\|V\|}, f_{\|V\|}, p_{\|V\|}))$:	A reaction molecule (i.e., solution) in TMSCRO
(v_i, f_i, p_i):	Atom (i.e., tuple) in S
InitCCPS:	The first CCP molecule for the initialization of TMSCRO
InitS:	The first molecule in TMSCRO
BelongCCP(w):	CCP_i that node w belongs to
CCPE(CCPs, CCP_e):	Edge between CCPs and CCPe
$\overline{W(v)}$:	Average computation cost of node v
$\mathrm{EC}_{P_r}(w)$:	Execution cost of a node w using processor P_r
$\mathrm{CM}(w, P_r, v, P_x)$:	Communication cost from node v to w, if P_x has been assigned to node v and P_r is assigned to node w
$\mathrm{ST}_{P_r}(w, v)$:	Possible start time of node w which is assigned the processor P_r with the v node being any predecessor of w which has already been scheduled
$\mathrm{EFT}_{P_r}(w)$:	Finish time of node w using processor P_r
AT_{P_r}:	Availability time of P_r
Pred(w) :	Set of predecessors of node w
Succ(w) :	Set of successors of node w
CCR :	Communication to computation ratio
g :	The parameter to adjust the heterogeneity level in a heterogeneous system
PE:	Current potential energy of a molecule
KE:	Current kinetic energy of a molecule
InitialKE:	Initial kinetic energy of a molecule
θ:	Threshold value guiding the choice of on-wall collision or decomposition
ϑ:	Threshold value guiding the choice of intermolecule collision or synthesis
Buffer:	Initial energy in the central energy buffer

KELossRate: Loss rate of kinetic energy
MoleColl: Threshold value to determine whether to perform a unimolecule reaction or an intermolecule reaction
PopSize: Size of the molecules
NumHit: Total collision number of a molecule.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. L. R. L. Graham, E. L. Lawler, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[2] C. Papadimitriou and M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC '88)*, pp. 510–513, 1988.

[3] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*, The MIT Press, Cambridge, Mass, USA, 1989.

[4] P. Chrétienne, "Task scheduling with interprocessor communication delays," *European Journal of Operational Research*, vol. 57, no. 3, pp. 348–354, 1992.

[5] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Computing*, vol. 38, no. 4-5, pp. 175–193, 2012.

[6] J. Xu, Y. S. Albert Lam, and O. K. Victor Li, "Stock portfolio selection using chemical reaction optimization," in *Proceedings of the International Conference on Operations Research and Financial Engineering (ICORFE '11)*, pp. 458–463, 2011.

[7] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[8] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[9] A. Amini, T. Y. Wah, M. R. Saybani, and S. R. A. S. Yazdi, "A study of density-grid based clustering algorithms on data streams," in *Proceedings of the 8th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '11)*, pp. 1652–1656, Shanghai, China, July 2011.

[10] H. Cheng, "A high efficient task scheduling algorithm based on heterogeneous multi-core processor," in *Proceedings of the 2nd International Workshop on Database Technology and Applications (DBTA '10)*, pp. 1–14, Wuhan, China, November 2010.

[11] T. Tsuchiya, T. Osada, and T. Kikuno, "A new heuristic algorithm based on gas for multiprocessor scheduling with task duplication," in *Proceedings of the 3rd International Conference on Algorithms and Architectures for Parallel Processing (ICAPP '97)*, pp. 295–308, Melbourne, Australia, December 1997.

[12] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107–118, 2004.

[13] H.-W. Ge, L. Sun, Y.-C. Liang, and F. Qian, "An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics A: Systems and Humans*, vol. 38, no. 2, pp. 358–368, 2008.

[14] N. B. Ho and J. C. Tay, "Solving multiple-objective flexible job shop problems by evolution and local search," *IEEE Transactions on Systems, Man and Cybernetics C: Applications and Reviews*, vol. 38, no. 5, pp. 674–685, 2008.

[15] E. S. H. Hou, N. Ansari, and H. Ren, "Genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, 1994.

[16] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM Journal on Computing*, vol. 18, no. 2, pp. 244–257, 1989.

[17] M. Iverson, F. Özgüner, and G. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in *Proceedings of the IEEE International Conference on Heterogeneous Computing Workshop (HCW '95)*, pp. 93–100, 1995.

[18] M. H. Kashani and M. Jahanshahi, "Using simulated annealing for task scheduling in distributed systems," in *Proceedings of the International Conference on Computational Intelligence, Modelling, and Simulation (CSSim '09)*, pp. 265–269, Brno, Czech Republic, September 2009.

[19] S. Kim and J. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in *Proceedings of the International Conference on Parallel Processing*, vol. 3, pp. 1–8, 1988.

[20] A. Y. S. Lam and V. O. K. Li, "Chemical-reaction-inspired metaheuristic for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 381–399, 2010.

[21] H. Li, L. Wang, and J. Liu, "Task scheduling of computational grid based on particle swarm algorithm," in *Proceedings of the 3rd International Joint Conference on Computational Sciences and Optimization (CSO '10)*, vol. 2, pp. 332–336, Huangshan, China, May 2010.

[22] M.-Y. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.

[23] G. C. Sih and E. A. Lee, "Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[24] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

[25] F.-T. Lin, "Fuzzy job-shop scheduling based on ranking level (lambda, 1) interval-valued fuzzy numbers," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 4, pp. 510–522, 2002.

[26] B. Liu, L. Wang, and Y.-H. Jin, "An effective PSO-based memetic algorithm for flow shop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, vol. 37, no. 1, pp. 18–27, 2007.

[27] F. Pop, C. Dobre, and V. Cristea, "Genetic algorithm for DAG scheduling in Grid environments," in *Proceedings of the IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP '09)*, pp. 299–305, Cluj-Napoca, Romania, August 2009.

[28] R. Shanmugapriya, S. Padmavathi, and S. M. Shalinie, "Contention awareness in task scheduling using tabu search," in *Proceedings of the IEEE International Advance Computing Conference (IACC '09)*, pp. 272–277, Patiala, India, March 2009.

[29] L. Shi and Y. Pan, "An efficient search method for job-shop scheduling problems," *IEEE Transactions on Automation Science and Engineering*, vol. 2, no. 1, pp. 73–77, 2005.

[30] P. Choudhury, R. Kumar, and P. P. Chakrabarti, "Hybrid scheduling of dynamic task graphs with selective duplication for multiprocessors under memory and time constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 7, pp. 967–980, 2008.

[31] S. Song, K. Hwang, and Y.-K. Kwok, "Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 703–719, 2006.

[32] D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd, "Performance-aware workflow management for grid computing," *The Computer Journal*, vol. 48, no. 3, pp. 347–357, 2005.

[33] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[34] J. Wang, Q. Duan, Y. Jiang, and X. Zhu, "A new algorithm for grid independent task schedule: genetic simulated annealing," in *Proceedings of the World Automation Congress (WAC '10)*, pp. 165–171, Kobe, Japan, September 2010.

[35] L. He, D. Zou, Z. Zhang, C. Chen, H. Jin, and S. Jarvis, "Developing resource consolidation frameworks for moldable virtual machines in clouds," *Future Generation Computer Systems*, vol. 32, pp. 69–81, 2012.

[36] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014.

[37] Y. Xu, K. Li, L. He, and T. K. Truonga, "A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[38] J. Xu, A. Lam, and V. Li, "Chemical reaction optimization for the grid scheduling problem," in *Proceedings of the IEEE International Conference on Communications (ICC '10)*, pp. 1–5, Cape Town, South Africa, May 2010.

[39] B. Varghese, G. Mckee, and V. Alexandrov, "Can agent intelligence be used to achieve fault tolerant parallel computing systems?" *Parallel Processing Letters*, vol. 21, no. 4, pp. 379–396, 2011.

[40] J. Xu, A. Lam, and V. Li, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 10, pp. 1624–1631, 2011.

[41] T. K. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem," *Applied Soft Computing Journal*, vol. 13, no. 4, pp. 1774–1780, 2013.

[42] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Arabe, and D. A. Menasce, "Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems," in *Proceedings of the ACM/IEEE Conference on Supercomputing (Supercomputing '92)*, pp. 683–691, IEEE Computer Society Press, Los Alamitos, Calif, USA, 1992.