The Journal of Systems and Software 84 (2011) 985–992

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm

Qinma Kang[a,b,*], Hong He[b], Huimin Song[c]

[a] Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 201804, China
[b] School of Information Engineering, Shandong University at Weihai, 180 Wenhua Xilu, Weihai 264209, China
[c] School of Mathematics and Statistics, Shandong University at Weihai, Weihai 264209, China

ARTICLE INFO

Article history:
Received 12 October 2010
Received in revised form 15 January 2011
Accepted 15 January 2011
Available online 31 January 2011

Keywords:
Iterated greedy algorithm
Task assignment
Task interaction graph
Heterogeneous computing
Meta-heuristics

ABSTRACT

A fundamental issue affecting the performance of a parallel application running on a heterogeneous computing system is the assignment of tasks to the processors in the system. The task assignment problem for more than three processors is known to be NP-hard, and therefore satisfactory suboptimal solutions obtainable in an acceptable amount of time are generally sought. This paper proposes a simple and effective iterative greedy algorithm to deal with the problem with goal of minimizing the total sum of execution and communication costs. The main idea in this algorithm is to improve the quality of the assignment in an iterative manner using results from previous iterations. The algorithm first uses a constructive heuristic to find an initial assignment and iteratively improves it in a greedy way. Through simulations over a wide range of parameters, we have demonstrated the effectiveness of our algorithm by comparing it with recent competing task assignment algorithms in the literature.

1. Introduction

Heterogeneous computing (HC) systems have been widely deployed for executing computationally intensive applications with diverse computing requirements. An HC system generally consists of a suite of geographically distributed dissimilar processors interconnected via communication networks. In such a system, a parallel application can be decomposed into a number of cooperating tasks that are distributed to the various processors for parallel execution. In reality, however, the performance of a parallel application running on a HC system heavily depends on the mapping of tasks partitioned from the application onto the available processors in the system, referred to as the task assignment problem which, if not properly handled, can nullify the benefits of parallelization.

Task assignment can be performed statically or dynamically (Casavant and Kuhl, 1988). Static task assignments take place during compile time before running the application and remain unchanged until the end of the execution. In contrast, dynamic task assignments are performed at run time. Since static mapping does not incur overheads on the execution time of the mapped application, more complex mapping algorithms than the dynamic ones can be adopted.

When all information needed for the assignment, such as the structure of the parallel application, the execution costs of tasks, the amount of data to be transferred among tasks, the computing nodes and the communication network, is known before the application execution, static mapping can be exploited. In the general form of static mapping, a parallel application is modeled using a task interaction graph (TIG). In the TIG model, the vertices represent application tasks and the edges represent inter-task communications. There are no precedence relations between tasks. A task incurs an execution cost that may vary from one processor to another, and two interacting tasks that are not assigned to the same processor incur a communication cost. Certain resource constraints, such as memory and processing load constraints, may be present at each processor. The goal of the task assignment is to minimize the sum of the total execution and communication costs by appropriately allocating the tasks to the processors without violating any of the constraints.

Due to its key importance on performance, the task assignment problem has been extensively studied and numerous methods have been reported in the literature. These allocation schemes can be classified into two categories. First, there are the exact methods that try to find the optimal allocation for the given objective. The existing approaches are developed using different strategies such as graph theoretic techniques (Stone, 1977), integer programming (Ernst et al., 2006), and state space search (Chern et al., 1989;

* Corresponding author at: School of Information Engineering, Shandong University at Weihai, 180 Wenhua Xilu, Weihai 264209, China. Tel.: +86 631 5688066; fax: +86 631 5688338.
E-mail addresses: qmkang@sina.com, qmkang@sdu.edu.cn (Q. Kang).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.01.051

Sinclair, 1987; Tom and Murthy, 1999). However, as the problem is NP-hard for more than three processors (Chern et al., 1989), these methods are limited by the amount of time and memory needed to obtain an optimal solution since they grow as exponential function of the problem order. On the other hand, heuristic algorithms provide fast and effective means for obtaining suboptimal solutions. These techniques require less computation time than exact methods. They are useful in applications where an optimal solution is not obtainable within a critical time limit. They are also applicable to large-size problems. Therefore, development of effective heuristic procedures is gaining importance among researchers. Different algorithms are used for developing heuristic methods such as greedy heuristic (Shatz et al., 1992), genetic algorithm (GA) (Chockalingam and Arunkumar, 1995; Hadj-Alouane et al., 1999), simulated annealing (SA) (Hamam and Hindi, 2000), hybrid particle swarm optimization (HPSO) (Yin et al., 2006), and harmony search (HS) (Zou et al., 2010).

Because of the intractable nature of the task assignment problem, new efficient techniques are always desirable to obtain the best-possible solution within a reasonable amount of computation time. The iterated greedy (IG) heuristic is an effective stochastic local search algorithm recently developed for combinatorial optimization problems which has exhibited state-of-the-art performances for several problems from computer science and engineering, such as set covering problems (Jacobs and Brusco, 1995; Marchiori and Steenbeek, 2000), flow shop scheduling problems (Pan et al., 2008; Ruiz and Stützle, 2007, 2008), Sequencing single-machine tardiness problems (Ying et al., 2009), multi-objective optimization problem (Dubois-Lacoste et al., in press), just to name a few. Thus, we intend to further extend the application of IG to the task assignment problem in the heterogeneous distributed computing systems. To the best of our knowledge, this study is the first to pioneer the use of IG heuristic for the problem considered.

The remainder of this paper is organized as follows. After formulating the problem in Section 2, the proposed IG heuristic is elaborated in Section 3. The computational results of applying the proposed algorithm to the task assignment problem are provided in Section 4, as well as comparisons of the performance against state-of-the-art meta-heuristics from the relevant literature. Finally, some concluding remarks are made in Section 5.

2. Problem formulation

There exists a large body of the literature covering many task and heterogeneous computing models. In this paper, we consider the task assignment problem with the following characteristics.

A distributed application is characterized by a task interaction graph (TIG) $G(V, E)$, where V is a set of N nodes indicating the N tasks of the application, and E is a set of edges specifying the communication requirements among these tasks. A weight c_{ij} associated with the edge between tasks i and j represents the amount of data to be transferred between the two tasks. The processors in the system are heterogeneous. Hence, a task will incur different execution costs if it is executed on different processors. Let K be the number of processors in the HC systems and $EEC = \{x_{ik}\}_{N\times K}$ be the estimated execution cost matrix where x_{ik} denotes the execution cost of task i on processor k. On the other hand, all of the communication channels are assumed to be non-uniform. That is, an identical amount of data, if transmitted through different communication channels, will incur different communication costs. Define d_{kl} as distance-related communication cost associated with one unit of data transferred from processor k to processor l, such that if tasks i and j are executed on processors k and l respectively, then a communication cost of $c_{ij}d_{kl}$ is incurred. The distance metric is symmetric, i.e., $d_{kl} = d_{lk}$. Furthermore, we assume that no communication cost is incurred if two interacting tasks are assigned to the same processors.

The allocation constraints depend on the characteristics of both the application involved (resource requirements by the tasks) and on the available resource capacities of the processors in the system. To describe the allocation constraints, let r_i denote the resource requirement of task i and let R_p denote the available resource capacity of processor p.

A particular task assignment can be represented by an integer vector φ of size N which is a mapping from the set of tasks to the set of processors. It contains the indices of the processors to which each task is allocated, i.e., $\varphi[i] = k$, if task i is allocated to processor k.

Let Ω be the set of all mappings from the set of tasks to the set of processors. Our objective is to minimize the total execution and communication costs incurred by the task assignment subject to the resource constraint. Hence, the considered task assignment problem can be formulated as

$$\text{minimize} \quad cost(\varphi) = \sum_{i=1}^{N} x_{i\varphi[i]} + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} c_{ij}d_{\varphi[i]\varphi[j]} \quad \forall \varphi \in \Omega \tag{1}$$

$$s.t. \quad \sum_{i:\varphi[i]=k} r_i \le R_k \quad k = 1, 2, \cdots, K \tag{2}$$

In the above formulation, objective function (1) consists of two parts. The first is the sum of the execution costs and the second the sum of the communication costs incurred between interacting tasks residing on different processors. Constraint (2) ensures that the total resource requirements of the tasks assigned to each processor must not exceed its resource availability.

3. Iterated greedy algorithm

The iterated greedy (IG) algorithm starts from a heuristically constructed solution to the given problem. Then following an iteration-based approach, it seeks to reach better solutions from one generation to the next. The algorithm has three basic steps: destruction, construction and acceptance criterion. In the destruction phase, some elements are randomly extracted from the incumbent solution to obtain a partial solution. Afterwards, a complete candidate solution is reconstructed by using a greedy constructive heuristic in the construction phase. Once a newly reconstructed solution has been obtained, an acceptance criterion is used to decide whether it will replace the incumbent solution or not. These steps are carried out repetitively until a stopping criterion is satisfied. Besides, an optional local search procedure can be easily added to improve the initial solution and the reconstructed solutions, respectively. A generic IG algorithm (Ruiz and Stützle, 2007) is outlined in Fig. 1, while each of its components customized to the task assignment problem will be detailed in the following subsections.

```
procedure IteratedGreedy
{   φ0 = GenerateInitialSolution;
    φ = LocalSearch(φ0);                % optional
    repeat
        φp = Destruction (φ);
        φc = Construction (φp);
        φ* = LocalSearch (φc);          % optional
        φ = AcceptanceCriterion (φ, φ*);
    until termination condition met
}
```

Fig. 1. General outline of an iterated greedy algorithm.

3.1. Initial solution

The proposed algorithm starts off by generating an initial feasible solution as the starting point of search by the local search procedure. We use the greedy constructive heuristic (GCH) adapted from Shatz et al. (1992) to rapidly obtain a solution. Based on the objective function (1), The algorithm below first sorts the tasks in decreasing order of task communication costs (Step 1) and then assigns all tasks one by one such that each time a task is assigned, the system costs incurred is minimized (Steps 2 and 3). The time complexity of this algorithm is $O(N^2K)$.

Step 1: Order all tasks in $S=(t_l,\ldots,t_N)$ so that the task communication costs are in decreasing order.

Step 2: Assign t_l to the processor p_k for which x_{1k} is minimum over all processors that are capable of processing t_l without violating system constraints.

Step 3: For $i=2$ to N do.

Assign t_i to the processor p_k such that

$$f(t_i) = x_{ik} + \sum_{j=1}^{i-1} c_{ij} d_{k\varphi[j]} \tag{3}$$

is minimum over all processors which are capable of processing t_i without violating system constraints. Apparently, $f(t_i)$ is the costs incurred by task t_i under the current partial assignment.

3.2. Destruction and construction phases

Let the incumbent assignment be φ. In the destruction phase, a given number d of tasks, chosen at random and without repetition, are removed from their current processors, and remaining tasks construct a partial assignment φ_p. The removed d tasks construct a sequence π_d in the order in which they were chosen. The tasks in π_d have to be reassigned to yield a complete candidate solution. The computational complexity of the destruction phase is $O(dN)$.

The construction phase consists in the application of Step 3 of the GCH given in Section 3.1. The construction phase starts with the partial assignment φ_p and performs d steps in which the tasks in π_d are reassigned. In particular, we start with φ_p and reassign the first task of π_d, $\pi_d[1]$, onto all possible processors that are capable of processing the task without violating system constraints. The best processor for $\pi_d[1]$ is the one that yields the smallest costs according to Eq. (3). This process is repeated until a complete assignment is obtained. The construction phase has a computational complexity of $O(dNK)$.

3.3. Local search

To further improve the performance of the basic IG algorithm, a local search procedure may be applied to the constructed solution to generate a new solution. There are many different alternatives for the local search algorithm that can be considered. We choose a rather straightforward local search algorithm that is based on the transfer neighborhood. The transfer neighborhood of an assignment is defined to be the set of assignments that can be obtained by removing one task from its current processor and reassigning it to another.

Following the idea of having a simple and easily implementable algorithm, we use a first-improvement type pivoting rule (Ruiz and Stützle, 2007, 2008; Pan et al., 2008). One single pass of this local search heuristic has a computational complexity of $O(N^2K)$. The outline of this procedure is shown in Fig. 2.

3.4. Acceptance criterion

Another step in the IG algorithm is to consider whether the new assignment is accepted or not as the incumbent solution for the next iteration. One of the simplest acceptance criteria is to accept solutions with better objective function values. However, an IG algorithm using this acceptance criterion is prone to stagnation due to insufficient diversification. Hence, we considered a simple SA type acceptance criterion. A candidate solution that is better or of equal quality to the incumbent solution is deterministically accepted; a worse solution is accepted with a probability given by $\exp\{-\Delta/T\}$, where Δ is the cost difference between the new and the current solution. T is a control parameter called temperature and it is computed following the suggestion of Stützle (2006) as follows: After the initial solution φ_0 is locally optimized to yield solution φ, we set the initial temperature $T_{\text{init}} = 0.025 \times cost(\varphi)$. The temperature is lowered every iteration according to a geometric cooling scheme, by setting $T_{i+1} = 0.9 \times T_i$.

3.5. Termination condition

Various conditions may be used to stop the algorithm: a maximal number of iterations, a computation time limit, etc. Since our algorithm will be compared to other meta-heuristics, the termination condition is the computation time here. The computation time is usually proportional to the problem size, and it can be determined in the preliminary tests based on the observation that all algorithms can converge before the cutoff time (see Section 4).

3.6. The IG for task assignment problem

The details of the proposed IG algorithm are presented in Fig. 3. The algorithm starts with an initial solution and then iterates through a main loop. In the loop a partial candidate solution is obtained firstly by removing some tasks from a complete solution, and next these tasks are reassigned one by one in a greedy way until a complete solution is reconstructed. To expedite the convergence speed, a local search heuristic is applied to further improve the constructed solution. An acceptance criterion decides whether the improved solution will become the new incumbent solution before entering the next iteration. This process is repeated until some stopping criterion is met. When the algorithm is terminated, the incumbent best solution and the corresponding objective function value are output and considered as the optimal task assignment and the minimum costs.

4. Experimental evaluation

To evaluate the performance of the proposed algorithm, we carry out an extensive computational study. Three existing meta-heuristics from the literature for the problem considered are also implemented for comparison purpose. These competing algorithms include the SA of Hamam and Hindi (2000), the HPSO of Yin et al. (2006), and the HS of Zou et al. (2010). We do not compare IG with a GA since the earlier studies of Yin et al. (2006) showed that the GA based algorithm for task assignment problem perform poorly in comparison with the HPSO algorithm. All the four algorithms are coded in MATLAB6.5 and executed on a 1.86 GHz Pentium Dual processor with 1GB main memory running under Windows XP environment.

For detailed information about their implementations of the SA, the HPSO, and the HS, we refer to (Attiya and Hamam, 2006; Yin et al., 2006; Zou et al., 2010). To be specific, a brief description and the parameter values that control each algorithm are given as follows:

```
procedure LocalSearch(φ)
{   improved = true;
    while (improved) do
        improved = false;
        generate a permutation π of tasks randomly;
        for i = 1 to N do
            φ* = best assignment obtained by reassigning task π [i] to any possible processors;
            if cost(φ*) < cost(φ)
                φ = φ*;
                improved = true;
            end if
        end for
    end while
    return φ
}
```

Fig. 2. The local search procedure based on the transfer neighborhood.

(1) SA:
   (a) The initial solution is generated randomly.
   (b) Cooling factor = 0.9.
   (c) The number of inner loop repetitions = four times the number of tasks.
   (d) Increasing factor = 1.05.
(2) HPSO:
   (a) $c_1 = c_2 = 2$.
   (b) Population size = 80.
   (c) The initial solutions are generated randomly.
   (d) The algorithm is implemented by hybridizing PSO with a hill-climbing local search heuristic.
(3) HS:
   (a) Population size = 10.
   (b) The initial solutions are generated randomly.
   (c) Mutation probability = 0.2.
   (d) The algorithm is implemented by embedding a hill-climbing local search heuristic.

As discussed above, the GCH heuristic is not used in the SA, the HPSO, and the HS; and therefore the competing meta-heuristics start from different initial solutions which are randomly generated from a uniform distribution. In addition, the local search used by the SA and the hill-climbing local search heuristic used by the HPSO and the HS are not the same used by the IG.

4.1. Experimental scenarios

Due to a lack of generally accepted standard benchmarks for the evaluation of assignment algorithms in the heterogeneous distributed computing systems, we have chosen to test the proposed IG algorithm on a variety of randomly generated task and processor graphs, which are similar to those used by other researchers (Shatz et al., 1992; Yin et al., 2006; Zou et al., 2010). The main parameters that characterize the generated instances are described below:

1) Number of tasks comprising the application TIG (N).
2) Number of processors in the system (K).
3) Task interaction density (D). It is the probability that there is an interaction between two tasks. Therefore, the task interaction density quantifies the ratio of the inter task communication demands for a TIG and can serve as one of the key factors that affect the problem complexity.
4) Communication to computation time ratio (CCR). It is the ratio of the average communication cost to the average computation cost. Thus the mean communication cost between tasks is set

```
procedure IteratedGreedy_for_TaskAssignment
{   φ0 = GCH;                          %initialization
    φ = LocalSearch(φ0);
    φb = φ;                            % best solution found so far
    T = 0.025 × cost(φ);
    while (termination criterion not satisfied)
        φp = φ;
        for i = 1 to d do
            φp = remove one task at random from φp and insert it in πd;
        end for
        for i = 1 to d do
            φc = best assignment obtained by reassigning task πd[i] onto all possible processors;
        end for
        φ* = LocalSearch (φc);
        if cost(φ*) < cost(φ) then     % Acceptance criterion
            φ = φ*;
            if cost(φ) < cost(φb) then
                φb = φ;
            end if
        elseif (random < exp {-( cost(φ*) - cost(φ)) / T}) then
            φ = φ*;
        end if
        T = 0.9 × T;
    end while
    return φb
}
```

Fig. 3. Iterated greedy algorithm for the task assignment problem.

to the average execution cost of tasks multiplied by the *CCR*. By using a range of *CCR* values, different types of applications can be accommodated. That is, if a TIG has a high *CCR*, it is considered as a data intensive application; if *CCR* is low, it is a computation intensive application.

The expected execution costs of all tasks taking heterogeneity into account are generated using the coefficient of variation (*COV*) based method described in (Ali et al., 2000) because the method provides a great control over the dispersion of the execution cost values than the common range based method (Braun et al., 2001). The *COV* is defined as the ratio of the standard deviation of task execution costs to the mean, and its values are used to quantify task heterogeneity and machine heterogeneity (Ali et al., 2000). Let $EEC = \{x_{ik}\}_{N\times K}$ be the expected execution cost matrix where x_{ik} denotes the execution cost of task i on processor k. First, a task vector of the expected execution costs which size equals the number of tasks is generated following gamma distribution with a mean of 30 and a *COV* of 0.9 (task heterogeneity). Then the *EEC* values for each task on all the machines are produced following gamma distribution using each element of the task vector as the mean and a *COV* of 0.9 (machine heterogeneity). The *EEC* matrix is of "inconsistent" type, i.e., there is no special structure in the execution cost matrix.

The transmission costs of links associated with one unit of data are generated from a uniform random distribution $U[1,10]$. Three levels of task interaction density (0.3, 0.5, and 0.8) are used for modeling the communication requirements among tasks. The volumes of data to be transmitted among tasks are randomly generated such that the *CCR* is 0.5, 1.0, or 2.0. The task resource requirements are generated from a uniform distribution $U[10,50]$, while the processor capacities are generated as $r \times \phi/M$, where r is a uniform random number in $U[1,2]$ and $\phi = \sum_{i=1}^{N} r_i$ is the total resource requirement across all the tasks.

In our experiments, the values of (N, K) are set to (60, 8), (80, 10), and (100, 12), respectively, to verify the algorithm with different problem scales. Thus, for each pair of (N, K), nine different TIGs are generated with three different task interaction density values and three different *CCR* values. For each problem set, 10 problem instances are generated randomly. As such, we obtain a testing data set of 270 problem instances for evaluating the comparative performances of the competing methods.

4.2. Choice of the destruction level

The destruction level d (number of tasks removed from a solution) may have an impact on the performance of the proposed algorithm. To identify an appropriate setting for the parameter, we have carried out a full factorial experiment with d as a controllable factor and N, K, D, and *CCR* as non-controllable factors. The destruction level is varied from $0.1N$ to $0.9N$ with an increment of $0.1N$. For the experiments we use a set of randomly generated instances. These instances have been generated using the *COV* based method given above. More precisely, we have considered 18 different combinations of N, K, D, and *CCR*, with $(N, K) \in \{(70, 8), (90, 10)\}$, $D \in \{0.3, 0.5, 0.8\}$, and $CCR \in \{0.5, 1.0, 2.0\}$. Due to the randomness of the proposed method, we have generated five instances for each of the 18 combinations and performed 10 runs per instance. The computation time limit for each run was set to $N \times K \times 20$ ms. Setting the time limit in this way allows more computation time as the problem size increases.

The experimental results were analyzed by means of a one-way analysis of variance (ANOVA) where N, K, D, and *CCR* were non-controllable factors. To check the ANOVA model hypothesis (normality, homoscedasticity and independence), the residuals resulting from the experimental data were analyzed and no major departure from the assumption was found. In this preliminary experiment, we focus only on the controllable factor. The computational results for most instances are similar, and therefore, we illustrate the main findings giving two example results. The means plots for instances of (N, K, D, CCR) being equal to (70, 8, 0.8, 0.5) and (90, 10, 0.5, 1) are depicted in Figs. 4 and 5, respectively.

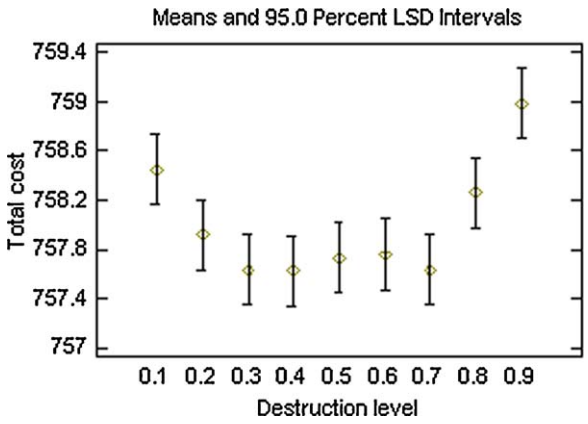

Fig. 4. Means plot and LSD intervals with 95% confidence level for various setting of the destruction level, $(N, K, D, CCR) = (70, 8, 0.8, 0.5)$.

It can be observed from the figures that very low and very high destruction levels do result in statistically worse performances. Recall that overlapping Least Significant Difference (LSD) intervals mean statistically equivalent levels for the studied factor. It seems that the setting of $d = 0.3N$ is much appropriate, so this value was adopted for all further experiments in this study.

4.3. Convergence analysis

To analyze the convergence behaviors and determine a computation time limit for all algorithms, we have recorded the variations of the objective function value obtained by every algorithm at each iteration step. For the population based algorithms, such as HPSO and HS, their best objective function values at each iteration step are recorded. Figs. 6–8 display the typical runs of the algorithms for solving three problem instances with a computation time limit set as $N \times K \times 20$ ms. We observe that the objective function value of the incumbent solution obtained by the IG algorithm decreases rapidly at the early stage of evolution, and the solution with inferior objective value could be accepted due to the SA type acceptance criterion, which helps IG escape from local optima. At the later stage, the IG algorithm exploits the neighborhoods around the high-quality

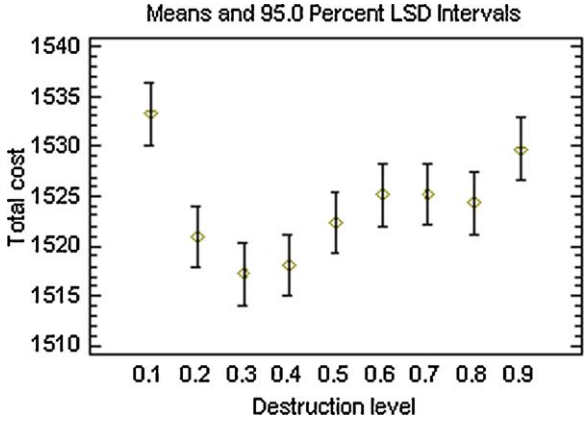

Fig. 5. Means plot and LSD intervals with 95% confidence level for various setting of the destruction level, $(N, K, D, CCR) = (90, 10, 0.5, 1)$.

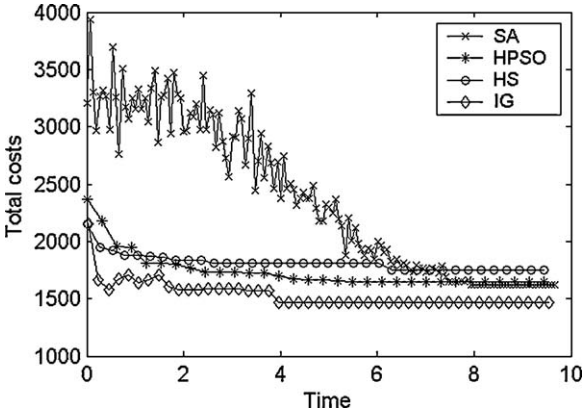

Fig. 6. The typical convergence behaviors of the SA, HPSO, HS, and IG for an instance $(N, K, D, CCR) = (60, 8, 0.3, 2)$.

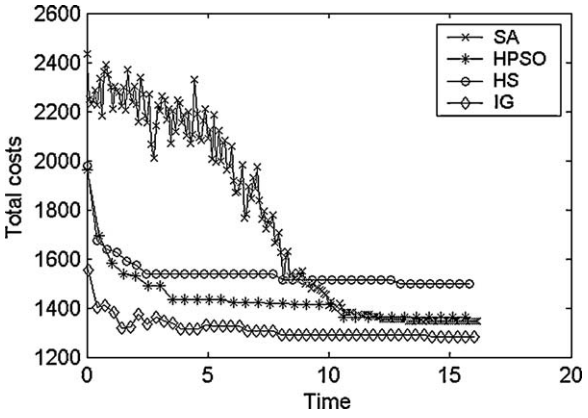

Fig. 7. The typical convergence behaviors of the SA, HPSO, HS, and IG for an instance $(N, K, D, CCR) = (80, 10, 0.5, 1)$.

solutions; and therefore, the decreasing rate becomes gentle. It is also observed that all algorithms converge before termination so their best performances are reached. Based on the observations, we set the computation time limit to $N \times K \times 20$ ms for the following experiments.

4.4. Experimental results

The performance measure considered is the quality of the solutions (total execution and communication costs). Moreover, all the

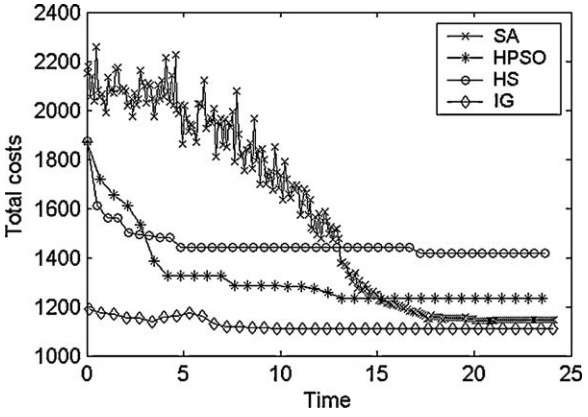

Fig. 8. The typical convergence behaviors of the SA, HPSO, HS, and IG for an instance $(N, K, D, CCR) = (100, 12, 0.8, 0.5)$.

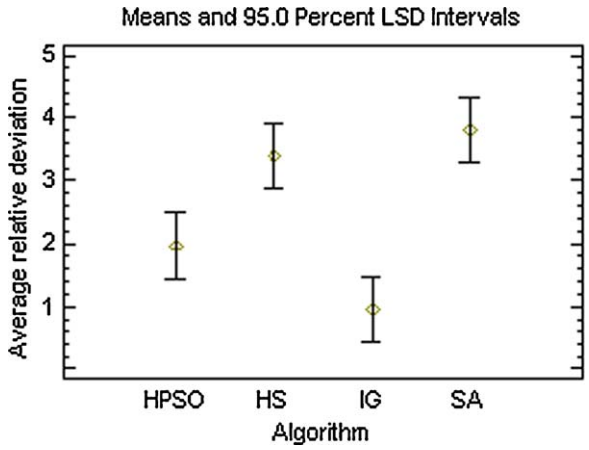

Fig. 9. Means plot and LSD intervals with 95% confidence level for all tested algorithms.

algorithms are stochastic approaches and each independent run of the same algorithm on a particular testing instance may yield a different result, we thus run each algorithm 10 times for every problem instance and report the statistical results.

Table 1 reports the comparison results obtained using the four algorithms. Each cell belonging to the caption 'C_{ARD}' reports the average relative percentage deviation obtained by the corresponding algorithm. The C_{ARD} is computed as follows:

$$C_{ARD} = \frac{C_{avg} - Best_{sol}}{Best_{sol}} \times 100 \quad (4)$$

where C_{avg} is the average objective function value on a given instance over 10 independent runs of the corresponding algorithm, and $Best_{sol}$ is the objective function value of the best solution obtained by all the algorithms for the same instance. In the resulting table, the caption 'C_{std}' denotes the standard deviation of objective function values obtained by the corresponding algorithm.

The comparative results presented in Table 1 show that IG algorithm performs best in terms of the mean relative percentage deviation values for all the experiments, followed by HPSO. A more careful examination of the experimental data shows that IG yields better results than HPSO in 19 out of 27 instances, and for only eight instances HPSO gives slightly better results than IG. Also, the IG algorithm shows the best performance in terms of standard deviation of the assignment costs, since the mean standard deviation generated by IG is smaller than those by other algorithms. These results reveal that the proposed IG algorithm is more effective than SA, HPSO, and HS for the task assignment with the goal of minimizing the total execution and communication costs in the heterogeneous distributed computing environments.

So far we have just shown average results. We need to carry out some statistical testing in order to validate the statistical significance of the observed differences in the average results. We performed an analysis of variance where we study a single factor, i.e., the type of algorithm at 4 levels and the average relative percentage deviation is the response variable. Fig. 9 shows the means plot and LSD intervals at 95% confidence level for the different algorithms. Recall that overlapping intervals indicates that no statistically significant difference exists among the overlapped means.

As can be seen, IG statistically outperforms all the other tested methods. HPSO works better than HS and SA, and there is no clear statistically significant difference between the HS and the SA at a 95% confidence level. Of course, this applies to the overall average.

Table 1
Comparison of IG with SA, HPSO, and HS.

Instances			IG		SA		HPSO		HS		
(N, K)	D	CCR	C_{ARD}	C_{std}	C_{ARD}	C_{std}	C_{ARD}	C_{std}	C_{ARD}	C_{std}	$Best_{sol}$
		0.5	**0.36**	1.44	0.86	2.74	0.65	0.73	1.51	2.58	712.1
	0.3	1.0	2.65	13.76	5.31	22.81	**0.50**	4.00	5.35	11.69	1050.1
		2.0	**1.19**	12.82	7.87	52.78	2.49	6.57	5.40	10.82	1574.9
		0.5	**0.13**	0.75	1.98	17.95	1.27	2.97	1.39	4.07	734.0
(60,8)	0.5	1.0	**0.15**	1.38	0.99	8.12	4.65	12.64	3.51	9.06	915.6
		2.0	**0.74**	11.94	7.25	35.00	4.57	25.29	3.95	28.18	1645.0
		0.5	**0.09**	0.84	0.67	4.09	0.68	2.27	0.25	0.87	648.8
	0.8	1.0	2.56	2.73	4.49	11.95	**0.35**	2.40	3.37	16.07	936.9
		2.0	**2.72**	23.95	3.94	47.30	3.07	15.42	3.48	14.65	1931.9
		0.5	1.74	9.40	7.78	19.44	**1.67**	12.74	4.79	9.41	1057.4
	0.3	1.0	**0.32**	1.95	2.64	30.08	1.95	15.76	4.75	5.60	1221.8
		2.0	1.14	16.49	7.26	54.29	**1.10**	19.23	2.81	25.35	2026.2
		0.5	**0.44**	3.21	3.82	17.98	3.11	3.97	4.58	7.67	888.6
(80,10)	0.5	1.0	**0.87**	11.01	9.70	20.31	2.15	26.89	4.51	9.24	1420.1
		2.0	**0.37**	12.33	8.15	37.26	9.47	22.54	9.47	19.55	2081.5
		0.5	**1.27**	3.09	2.73	7.61	0.24	1.47	1.55	1.91	793.1
	0.8	1.0	**0.44**	4.62	4.46	60.96	2.74	9.41	2.83	6.49	1319.8
		2.0	**1.34**	25.83	4.12	61.30	1.41	12.99	1.98	24.40	2293.1
		0.5	1.53	4.90	2.51	9.88	**0.61**	6.19	2.55	4.41	1147.5
	0.3	1.0	**0.61**	9.71	3.92	23.04	1.26	4.02	3.46	8.57	1717.9
		2.0	**0.92**	17.77	1.83	38.55	3.62	7.56	4.15	12.61	2422.7
		0.5	**0.59**	2.56	1.32	10.22	0.94	1.08	1.58	2.14	1130.1
(100,12)	0.5	1.0	0.72	3.69	2.57	16.92	**0.47**	7.18	3.67	8.53	1560.6
		2.0	**0.53**	15.68	2.03	25.05	1.91	23.56	2.91	19.51	3049.4
		0.5	0.10	0.36	0.22	2.11	**0.07**	0.45	0.46	1.52	912.7
	0.8	1.0	2.34	20.27	1.46	20.64	**0.90**	10.01	4.27	7.76	1616.5
		2.0	**0.33**	9.57	3.26	32.98	1.05	19.29	2.98	35.21	2767.5
Average			0.97	8.96	3.82	25.61	1.96	10.25	3.39	11.40	

Table 2
Results of IG with varying computation time.

Instances			Normal runs	Short runs		Long runs	
(N, K)	D	CCR	C_{normal}	C_{short}	C_{loss}	C_{long}	$C_{benefit}$
		0.5	1394.5	1435.3	2.93	1392.8	0.12
	0.3	1.0	2232.5	2264.2	1.42	2223.4	0.41
		2.0	3255.8	3358.7	3.16	3251.5	0.13
		0.5	1245.2	1285.7	3.25	1243.2	0.16
(128,16)	0.5	1.0	1943.9	1972.8	1.49	1934.5	0.48
		2.0	4150.3	4268.7	2.85	4125.3	0.60
		0.5	1050.9	1089.6	3.68	1050.0	0.08
	0.8	1.0	1854.3	1862.9	0.46	1849.9	0.24
		2.0	3226.0	3252.6	0.82	3213.4	0.39
Average					2.23		0.29

4.5. Variations on computational time

In this subsection, we are interested in the behavior of the proposed IG algorithm when shorter or longer computational time is provided. For this purpose, we first ran the IG algorithm with cutoff time set as $N \times K \times 20$ ms. And then we halved the initial cutoff time in the second experiment. Finally we doubled it in the third experiment. The average results over 10 independent runs on instances with (N, K) equaling (128, 16) are presented in Table 2. In the resulting table, the captions 'C_{normal}', 'C_{short}', and 'C_{long}' denote the average total costs obtained by normal runs, short runs, and long runs of the IG algorithm, respectively; the loss and the benefit in mean costs are computed according to Eqs. (5) and (6), respectively, as follows:

$$C_{loss} = \frac{C_{short} - C_{normal}}{C_{normal}} \times 100 \quad (5)$$

$$C_{benefit} = \frac{C_{normal} - C_{long}}{C_{normal}} \times 100 \quad (6)$$

This table first shows that IG still produces high-quality solutions within short computational times. Compared to the reference runs, we observe the loss in mean costs ranges from 0.46% to 3.68% with an average of 2.23%. This is an interesting feature in practical situations where efficient algorithm is needed. On the other hand, this table also confirms that IG benefits from more computational time. Indeed, long runs lead always to solutions of better quality. The improvement is important for some contexts where the resources should be utilized as optimally as possible.

5. Conclusions

To the best of our knowledge, this is the first report on the application of an iterated greedy algorithm to the task assignment problem in heterogeneous computing systems. We used the greedy constructive heuristic (GCH) by Shatz et al. (1992) to generate an initial solution with a certain quality, and employed the construction procedure based on the GCH heuristic and the SA type acceptance criterion to avoid the search getting trapped in local minima. We then applied the simple local search to stress local exploitation and to improve the effectiveness. The performance of the proposed algorithm is evaluated in comparison with three competing algorithms on a number of randomly generated mapping problem instances. Computational results demonstrated the superiority of the proposed iterated greedy algorithm in terms of

effectiveness. Furthermore, the IG has the advantages that it has fewer parameters that need to be tuned than the competing algorithms, and it is a rather simple, easily implementable algorithm compared to HPSO algorithm, which is the state-of-the-art method for the problem considered.

We are currently extending the application of the proposed IG algorithm to another version of the task assignment problem where each processor and each communication link has a failure ratio and the goal is to maximize the system reliability for accomplishing the task execution.

Acknowledgements

The authors thank the anonymous referees and the editor for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (NSFC), under Grant Number 60803065, and partially supported by the National High Technology Research and Development Program of China (863 Program) Grant No. 2009AA01Z141, the National Basic Research Program of China (973 Program) Grant 2010CB328100 and the Program for Changjiang Scholars and Innovative Research Team in University.

References

Ali, S., Siegel, H.J., et al., 2000. Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering 3 (3), 195–207.

Attiya, G., Hamam, Y., 2006. Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. Journal of Parallel and Distributed Computing 66, 1259–1266.

Braun, T.D., Siegel, H.J., et al., 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system. Journal of Parallel and Distributed Computing 6, 810–837.

Casavant, T., Kuhl, J.G., 1988. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transaction on Software Engineering 14 (2), 141–154.

Chern, M.S., Chen, G.H., Liu, P., 1989. An LC branch-and-bound algorithm for module assignment problem. Information Processing Letters 32, 61–71.

Chockalingam, T., Arunkumar, S., 1995. Genetic algorithm based heuristics for the mapping problem. Computer and Operations Research 22, 55–64.

Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T. A hybrid TP + PLS algorithm for bi-objective flow-shop scheduling problems. Computers and Operations Research, in press.

Ernst, A., Jiang, H., Krishnamoorthy, M., 2006. Exact solutions to task allocation problems. Management Science 52, 1634–1646.

Hadj-Alouane, A.B., Bean, J.C., Murty, K.G., 1999. A hybrid genetic/optimization algorithm for a task allocation problem. Journal of Scheduling 2, 189–201.

Hamam, Y., Hindi, K.S., 2000. Assignment of program modules to processors: a simulated annealing approach. European Journal of Operational Research 122, 509–513.

Jacobs, L.W., Brusco, M.J., 1995. A local-search heuristic for large set-covering problems. Naval Research Logistics Quarterly 42, 1129–1140.

Marchiori, E., Steenbeek, A., 2000. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. Lecture Notes in Computer Science 1803, 367–381.

Pan, Q.K., Wang, L., Zhao, B.H., 2008. An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. International Journal of Advanced Manufacturing Technology 38, 778–786.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177 (3), 2033–2049.

Ruiz, R., Stützle, T., 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 187 (3), 1143–1159.

Shatz, S.M., Wang, J.P., Goto, M., 1992. Task allocation for maximizing reliability of distributed computer systems. IEEE Transactions on Computers 41, 1156–1168.

Sinclair, J.B., 1987. Efficient computation of optimal assignments for distributed tasks. Journal of Parallel and Distributed Computing 4, 342–361.

Stone, H.S., 1977. Multiprocessor scheduling with the aid of network flow algorithms. IEEE Transactions on Software Engineering SE 3 (1), 85–93.

Tom, A.P., Murthy, C.S.R., 1999. Optimal task allocation in distributed systems by graph matching and state space search. Journal of Systems and Software 46 (1), 59–75.

Stützle, T., 2006. Iterated local search for the quadratic assignment problem. European Journal of Operational Research 174, 1519–1539.

Yin, P.Y., Yu, S.S., Wang, P.P., Wang, Y.T., 2006. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. Computer Standard and Interface 28, 441–450.

Ying, K.C., Lin, S.W., Huang, C.Y., 2009. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Systems with Applications 36, 7087–7092.

Zou, D.X., Gao, L.Q., Li, S., Wu, J.H., Wang, X., 2010. A novel global harmony search algorithm for task assignment problem. Journal of Systems and Software 83 (10), 1678–1688.