A Permutation-Based Differential Evolution Algorithm Incorporating Simulated Annealing for Multiprocessor Scheduling with Communication Delays

Xiaohong Kong[1,2], Wenbo Xu[1], and Jing Liu[1]

[1] School of Information Technology, Southern Yangtze University, Wuxi 214122, China
nancykong@hist.edu.cn, xwb@sytu.edu.cn
[2] Henan Institute Of Science and Technology, Xinxiang, Henan 453003,China

Abstract. Employing a differential evolution (DE) algorithm, we present a novel permutation-based search technique in list scheduling for parallel program. By encoding a vector as a scheduling list and differential variation as s swap operator, the DE algorithm can generate high quality solutions in a short time. In standard differential evolution algorithm, while constructing the next generation, a greedy strategy is used which maybe lead to convergence to a local optimum. In order to avoid the above problem, we combine differential evolution algorithm with simulated annealing algorithm which relaxes the criterion selecting the next generation. We also use stochastic topological sorting algorithm (STS) to generate an initial scheduling list. The results demonstrate that the hybrid differential evolution generates better solutions even optimal solutions in most cases and simultaneously meet scalability.

1 Introduction

Given parallel program modelled by a directed acyclic graph (DAG), the objective of scheduling the tasks to multiprocessors is minimizing the completion time or makespan while satisfying the precedence constraints. The problem is NP-hard even simplified model with some assumptions and becomes more complex under realistic application such as arbitrary task execution and communication times. Due to the intractability, many classical heuristics have been proposed to find out sub-optimal solution of the problem, the idea behind these heuristic algorithms is to tradeoffs the solution quality and the complexity [1-5]. Recently meta-heuristics search approaches have also made some accomplishment on solving the problem [1][2][3].

Since DE was first introduced to minimizing possibly nonlinear and non-differentiable continuous space functions [6], it has been successfully applied in a variety of applications [7]. In this paper, we exploit a hybrid differential evolution algorithm to construct the solution for parallel program scheduling with the permutation-based solution presentation.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 514–521, 2006.
© Springer-Verlag Berlin Heidelberg 2006

2 The Multiprocessor Scheduling with Communication Delays

It is popular to model the multiprocessor scheduling using a directed acyclic graph (DAG), which can be defined by a tuple $G = (V, E, C, W)$, where $V = \{n_j, j = 1 : v\}$ is the set of task nodes and $v = |V|$ is the number of nodes, E is the set of communication edges and $e = |E|$ is the number of edges,C is the set of edge communication costs, and W is the set of node computation costs. The value $c(n_i, n_j) \in C$ corresponds to the communication cost incurred while the task n_i and n_j are scheduled to different processors, which is zero if both nodes are assigned on the same processor. The value $w(n_i) \in W$ is the execution time of the node $n_i \in V$. The edge $e_{i,j} = (n_i, n_j) \in E$ represents the partial order between tasks ni and n_j, which dictate that a task cannot be executed unless all its predecessors have been completed their execution.

The target system M is consisted of m identical or homogeneous processors with local memory connected by an interconnection network with a certain topology. When scheduling tasks to machines, we assume every task of a parallel program can be executed on any processor and only on one processor non-preemptively and the system executes computation and communication simultaneously.

Scheduling the graph G to M is to find out pairs of (task, processor) which optimize the scheduling length or completion time. Most scheduling algorithms are based on the so-called list scheduling strategy. The basic idea of list scheduling is to make a scheduling list (a sequence of nodes for scheduling) by assigning them some priorities, and then assign the tasks to processor according to some rule such as the earliest start time first.

3 Differential Evolution Algorithm

Differential evolution (DE) is one of the latest evolutionary optimization methods proposed by Storn and Price [6]. Like other evolution algorithms, mutant operator, Crossover operator, selection operator are introduced to generate a next generation, but DE has its advantages such as simple concept, immediately accessible for practical applications, simple structure, ease of use, speed to get the solutions, and robustness, parallel direct search method [6].

At the heart of the DE method is the strategy that the weighted difference between two vectors selected randomly is exerted on the perturbed vector to generate a trial vector, then the trial vector and the assigning vector exchange some elements according to probability, better individuals are selected as members of the generation G+1.

For example, one version DE/rand/2 updates according to the following formulates:

(1) Initial population,$X_{i,G}$, $i = 0, 1, 2, \cdots, NP - 1, NP$ is the number of population.
(2) Evolution operation, for every $X_{i,G}$, denote running vector.

Mutation
A mutation vector v is generated according to

$$V_{i,G+1} = X_{r1,G} + F1 * (X_{r2,G} - X_{r3,G}) + F_2 * (X_{r4,G} - X_{r5,G}) \quad (1)$$

$r1$,$r2$,$r3$,$r4$ and $r5 \in [0, NP-1]$ re mutually different integer and different from running index i; $F_1, F_2 \in [0, 2]$ are constant factor which controls the amplification of the differential variation $(X_{r2,G} - X_{r3,G})$ and $(X_{r4,G} - X_{r5,G})$.

Crossover
The trial vector is formed,

$$U_{i,G+1} = (u_{0i,G+1}, u_{1i,G+1}, \ldots, u_{(D-1)i,G+1}) \quad (2)$$

$$\text{where }, u_{ji} = \begin{cases} v_{ji,G+1} & j =< n >_D, < n+1 >_D, \ldots, < n+L-1 >_D \\ x_{ji,G} & \text{otherwise} \end{cases} \quad (3)$$

where $<>_D$ denote the modulo function with modulus D. The starting index n in (2) is a randomly chosen integer from the interval $[0, D-1]$. The integer L is drawn from the interval $[0, D-1]$ with the probability

$$Pr(L = v) = (CR)^v \quad (4)$$

$CR \in [0, 1]$ is the crossover probability and constitutes a control variable for the DE scheme. The values of both n, D and L can refer to literature [6].

Selection
In order to determine whichever of $U_{i,G+1}$ and $X_{i,G}$ is transferred into the next generation, the fitness values of the two are comparedand the better is preserved.

(3) Stop Criterion

This process is repeated until a convergence occurs.

4 Applying DE Heuristic to Scheduling Problem

The DE algorithm with few control variables is robust, easy to use and lends itself very well to parallel computation [6]. However, the continuous nature of the algorithm limited DE to apply to combinatorial optimization problems. In order to use it in parallel program scheduling problem, we must re-define the operations in following way as to take into account the precedence relations between tasks.

4.1 Redefining the DE

Defining the vector: Every vector in differential evolution algorithm is represented by a feasible permutation of tasks, a tasks list satisfying topology order.

Defining of differential variation: In our proposed algorithm, the differential variations $(X_{r2,G} - Xr3, G)$ and $(X_{r4,G} - X_{r5,G})$ is defined as a set of Swap

Operator on task nodes in scheduling list[8]. Consider a normal solution sequence of multiprocessor scheduling with n nodes, here we define Swap Operator $SO(n_i, n_j)$ as exchanging node n_i and node n_j in scheduling list. Then we define $\widetilde{S} = S + SO(n_i, n_j)$ as a new solution on which operator $SO(n_i, n_j)$ acts. For example,

$$\{n_1, n_2, n_3, n_4, n_5, n_6\} + SO(n_1, n_3) = \{n_3, n_2, n_1, n_4, n_5, n_6\} \tag{5}$$

Plus operation between two SOs: Swap Set SS is a set of Swap Operators.

$$SS = (SO_1, SO_2, SO_3, \cdots, SO_n) \tag{6}$$

When Swap Set acts on a solution, all the Swap Operators of the swap Set act on the solution in order. i.e.

$$SS = \{(n_i^k, n_j^k), i, j \in \{1, 2, \cdots, N\}, k \in \{1, 2, \cdots, \}\} \tag{7}$$

which represents that node n_i^k and n_j^k are swapped firstly, and n_i^2 and n_j^2 are swapped secondly, and so forth. Define plus operation between SO_1 and SO_2 as the union of the two swap operators, denote

$$SO_1 + SO_2 \tag{8}$$

so Swap Set operation can be described by the following formula[8]:

$$\begin{aligned} \widetilde{S} &= S + SS = S + (SO_1, SO_2, SO_3, \cdots, SO_n) \\ &= ((SO_1, SO \cdots) + SO_2) + \ldots + SO_n \end{aligned} \tag{9}$$

The plus sign "+" above means continuous swap operations on one solution.

Plus operation between two SSs: If several Swap Sets have the same results as a single Swap Set acting on one solution, we define the operator "$\oplus$" as merging several Swap Sets into a new swap Set[8]. For instance, there is two Swap Sets SS_1 and SS_2, SS_1 and SS_2 act on one solution$S$ in order, and there is another Swap Set $S\widetilde{S}$ acting on the same solution S, then get the same solution $\widetilde{S}$, call that $S\widetilde{S}$ is equivalent to $SS_1 \oplus SS_2$.

Minus operation between two vectors: Suppose there are two vectors, A and B, a Swap Set SS which can act on B to get vector A, i.e. we can swap the nodes in B according to A from left to right to get SS. So there must be an equation $A = B + SS$. We define the minus operation between vectors A and B as a SS, that is. $A = B + SS \Leftrightarrow A - B = SS$.

Updating: On the basis of above, Formula (1) has already no longer been suitable for the scheduling problem. We update it as follows:

$$V_{i,G+1} = X_{r1,G} + (X_{r2,G} - X_{r3,G}) \oplus (X_{r4,G} - X_{r5,G}) \tag{10}$$

4.2 Stochastic Topological Sorting Algorithm

On the basis of above, the initial vectors, initial scheduling list, must satisfy the precedence constrains. The topological sorting algorithm (TS) can serve the

purpose, but the TS has two fatal disadvantages, one of which is that it is based on the depth-first search so that the topological orders generated by the TS algorithm cannot cover the whole feasible solutions, the other of which is that the topological order is fixed because it is subject to the storage structure of the DAG in the computer. In [9], we devise a novel sorting algorithm, a stochastic topological sorting algorithm (STS). The STS algorithm will be used in this paper to generate an initialized population.

4.3 Crossover Operation

In our algorithm, crossover operators adopted do not exchange the values of the elements but the order the elements appear in vectors to avoid permutation infeasibility when using permutation-based DE for the scheduling. We lend this idea from partially mapped crossover (PMX) [10], so that the order the activities appear in the multidimensional vectors other than the values of the elements are changed during updating process. The strategy of PMX that performs swapping elements is illustrated in Fig.1.

Here, the mutation vector and the running vector, respectively, resemble parent 1 and parent 2 in PMX, except that the two vectors should not be incorporated into a vector. Element(called element 1) of mutation vector will be determined according to formula(3) to see if the activity represented by the element will be moved to another placement or if the element will be swapped with another one (called element 2), which is the element that element 1 maps in the running vector[11]. When the placements of the two elements satisfy the predecessors-successors constraints, the crossover operation takes place.

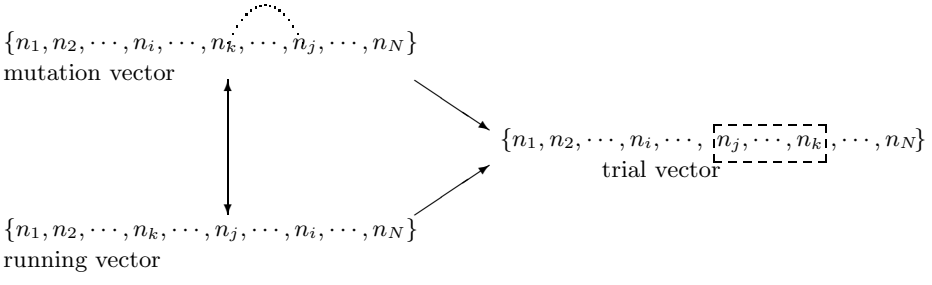

Fig. 1. An example of the partially mapped crossover operator in DE vectors

4.4 Simulated Annealing Selection Operation

Once a new solution is generated, a decision must be made whether or not to accept the newly derived vector as next generation. In standard DE, a greedy strategy is utilized to determine trial and running vector, which means the better of fitness of the two survive into next generation. The greedy criterion can converge fairly fast, but it runs the risk of becoming trapped by a local minimum[6].

The mechanism of Simulated Annealing is introduced into DE[12], which effectively optimize the objective function while moving both uphill and downhill. According to SA algorithm, selection strategy not only accepts better solutions (decreasing scheduling length) but also some worse solutions(increasing scheduling length). The Metropolis criterion decides on acceptance probability of worse solutions[12].

5 Experiment Results

Benchmark instances from website (http://www.mi.sanu.ac.yu/ tanjad/) are used to test the validity of our approach with three different selection strategies. In order to demonstrate the efficiency of the SA select operation, we test the benchmark instances using three strategies, SA selection operation (DE+SA), standard DE(STDE), the trial vector directing into next generation(DE+RAND) respectively.

In our experiments, we select the ogra100 problem with different edge densities , which is defined as the number of edges divided by the number of edges of a completely connected graph with the same number of nodes [5]. For each density, the mean deviations from the optimal schedule for 10 runs are recorded in Table1 for DE with and without SA selection strategy respectively. DE with SA always has comparable performance on the results of DE without SA whether using different processors or different densities.

Table 1. Results of the DE algorithm with different selection strategies compared against optimal solutions (% deviations) for the random task graphs with three densities using 2, 4, 6, and 8 processors.(amplification factor F1=F2=1, Iteration number equals 20).

	No. of processors											
	STDE				DE+RAND				DE+SA			
Graph density	2	4	6	8	2	4	6	8	2	4	6	8
60	3.75	8.00	9.25	10.88	2.88	7.13	12.88	11.25	2.88	6.75	8.63	6.79
70	2.75	6.38	7.25	7.25	2.63	8.50	12.13	9.38	1.63	6.75	6.79	6.75
80	1.63	5.00	7.63	4.63	1.38	8.13	10.25	7.38	1.00	4.00	4.75	4.75
90	1.00	1.12	5.48	5.50	1.13	3.50	5.50	5.50	0.50	1.63	4.00	4.00
Avg. Dev.	2.28	5.12	7.40	7.06	2.00	6.81	10.18	8.38	1.50	4.78	6.04	5.57

At the same time, we also investigate how the density of graphs affects the scheduling results. Because the lengths of optimal schedules are given depend onlyon task number, it appears that dense graphs spent less communication costs. Fig.2 is the result. Based on same tasks number and processors, less deviation is achieved for dense-task graphs.

Finally, we test the average deviation under different task numbers, and the result is illustrated in Fig.3. With the tasks number increasing, the communication cost between tasks assigned to different processors has a vital proportion to

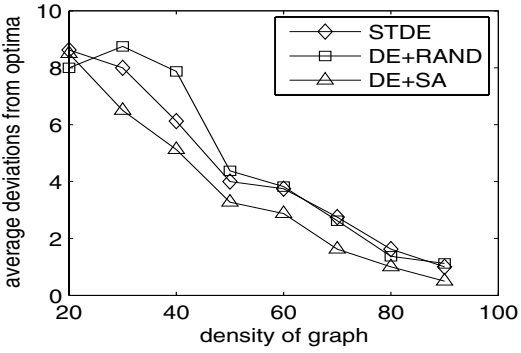

Fig. 2. The average % deviations from the optimal of the solutions generated by the DE algorithm for the random task graphs with different densities using 2 processors

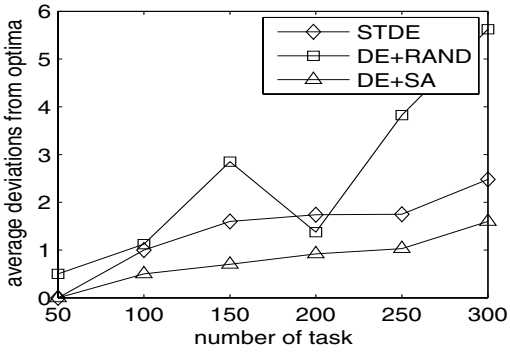

Fig. 3. The average % deviations from the optimal of the solutions generated by the DE algorithm for the random task graphs with different tasks using 4 processors

the overall cost. A conclusion can be observed that the DE algorithm with SA selection strategy for scheduling has a good scalability.

From the values above it is evident that a hybrid strategy of DE incorporating temperature-based acceptance criterion of SA is preferable to greedily selecting manipulation because the use of probability acceptance to inferior solutions in SA enhances the solution diversity in search process. The effect is always desirable due to the advantage of the DE with SA to avoid the local optima.

6 Summary

We have presented a differential evolution algorithm for multiprocessor DAG scheduling. As can be see from the previous results of the performance-testing experiment, in most cases the permutation-based DE method can find near-optimal schedule, especially the DE method with SA selection technique. In

practice, we combine the DE technique of searching in global space with the SA capacity to jump out of local optimum in selecting an optimal scheduling list.

References

1. I. Ahmad, M. K. Dhodhi: Multiprocessor Scheduling in a Genetic Paradigm. Parallel Computing,Vol. 22 (1996) 395-406
2. Y.-K. Kwok, I. Ahmad: Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm. J. Parallel and Distributed Computing,Vol 47,(1997) 58-77
3. T. Davidovic, P. Hansen, N. Mladenovic: Permutation Based Genetic, Tabu and Variable Neighborhood Search Heuristics for Multiprocessor Scheduling with Communication Delays. Asia-Pacific Journal of Operational Research,Vol. 22, (2005) 297-326
4. T. Davidovic, T. G. Crainic: New Benchmarks for Static Task Scheduling on Homogeneous Multiprocessor Systems with Communication Delays. Publication CRT-2003-04, Centre de Recherche sur les Transports, Universite de Montreal (2003)
5. T. Davidovic, P. Hansen, N. Mladenovic: Variable Neighborhood Search for Multiprocessor Scheduling Problem with Communication Delays. Proc. MIC2001, 4th Metaheuristic International Conference, J. P. de Sous, Porto, Portugal (2001)
6. R. Storn, K. Price: Differential Evolution-a Simple and Efficient Heuristic for Global Optimization over Continuous Space. Journal of Global Optimization, Vol. 11,(1997) 341-359
7. R. Storn: System Design by Constraint Adaptation and Differential Evolution. IEEE Transaction on Evolutionary Computation,vol. 3 (1999) 22-34
8. KANG-PING WANG, LAN HUANG,CHUN-GUANG ZHOU,WE1 PANG: Particle Swarm Optimization for Traveling Salesman Problem. Proceedings of the Second International Conference on Machine Learning and Cybernetics,vol. 5 (2003) 1583-1585
9. Wenbo Xu, Jun Sun: Efficient Scheduling of Task Graphs to Multiprocessors Using a Simulated Annealing Algorithm. DCABES 2004 Proceedings,Vol .1 (2004) 435-439
10. D.E. Goldberg: Genetic Algorithms in Search, Optimization, and Matching Learning. Addison-Wesley Publishing Company, Inc.,Reading (1989)
11. Hong Zhang, Xiaodong Li, Heng Li: Particle Swarm Optimization-Based Schemes for Resource-Constrained Project Scheduling. Automation in Construction,Vol. 14 (2005) 393-404
12. S.Kirkpatrick, C.Gelatt, M.Vecchi: Optimization by Simulated Annealing. Science,vol. 220 (1983) 671-680