TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 04/20 pp18-23
Volume 6, Number 1, March 2001

Communication Optimization for SMP Clusters*

LIN Weijian (林伟坚), CHEN Wenguang (陈文光)†,
LI Zhiguang (李志光), ZHENG Weimin (郑纬民)†

Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, China;
† Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract Shared Memory Processors (SMP) workstation clusters are becoming more and more popular. To optimize communication between the workstations, a new graph partition problem was developed to schedule tasks in SMP clusters. The problem is NP-complete and a heuristic algorithm was developed based on Lee, Kim and Park's algorithm. Experimental results indicate that our algorithm outperforms theirs, especially when the number of partitions is large. This algorithm can be integrated in a parallelizing compiler as a back end optimizer for the distributed code generator.

Key words SMP cluster; communication optimization; task scheduling

Introduction

Shared Memory Processors (SMP) workstation clusters are being used more and more. The communication cost inside an SMP machine is much less than the inter-machine communication cost. So tasks should be carefully allocated to minimize the inter-machine communication. Figure 1 shows a simple execution model for an SMP cluster. The execution model used here is simplified, with just one communication phase and we assume that the cost of the calculation phase is equal for all parallel tasks. For many parallel applications, this model is an acceptable simplification and it is a good starting point to solve the general task allocation problem for SMP workstation clusters.

Inter-node communication is separated from internal communication in Fig. 1 as a simplification but they are combined during actual execution. If the cost of the calculation phase of each task is the same, the whole program execution time would be minimized when the inter-node communication is minimized. A parallel program does not finish until the last task finished, the largest inter-node communication cost from one node to other nodes should be minimized. If each SMP node has the same number of processors, the tasks should also be allocated such that each SMP node has the same number of tasks.

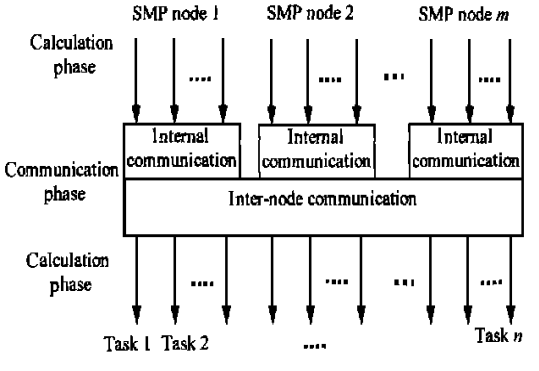

Fig. 1 A simple execution model for an SMP cluster

1 Problem Formulation

This section first defines some notation, then the notation is used to formulate a communication optimization problem into a graph partition problem.

Received: 1999-04-15; revised: 2000-07-17
* Supported by the State High-Tech Developments Plan of China (No. 863-306-ZD-02)

1.1 Definition

1.1.1 Size of nodes and subsets

Consider a graph $G(V, E)$ with N nodes $(v_1, v_2, \cdots, v_N)$ and E edges $(e_1, e_2, \cdots, e_E)$. Let the size of v_i be denoted by $S(v_i)$ and cost of e_j by $W(e_j)$. The size of each node and the cost of each edge are assumed to be non-negative integer values.

The size of node set V is denoted by $S(V)$, where $S(V)=\sum_{v\in V} S(v)$.

1.1.2 Balanced partition concept

Equal-sized partitions are generalized into balanced partitions since equality can not be achieved in general. Let a k-cut, C, partition the nodes of G into k subsets $P_1, P_2, \cdots, P_k$. We call C a balanced partition if $\sum_{1\leqslant i<j\leqslant k} |S(P_i)-S(P_j)|$ is a minimum.

1.1.3 Weighted sum between subsets

The weighted sum of edges between subsets P_i, P_j is denoted by $W_s(P_i, P_j)$, where

$$W_s(P_i, P_j) = \sum_{u\in P_i, v\in P_j} W((u, v)),$$

and the weighted sum of edges between one subset P_i and all other subsets is denoted by $S_{ws}(P_i)$, where

$$S_{ws}(P_i) = \sum_{1\leqslant j\leqslant k, j\neq i} W_s(P_i, P_j).$$

1.1.4 Goal functions

$$G_m(C) = \max_{1\leqslant i\leqslant k} S_{ws}(P_i),$$

$$G_s(C) = \sum_{1\leqslant i<j\leqslant k} W_s(P_i, P_j).$$

1.2 Problem statement

With the notation defined in Section 1.1, the communication optimization problem can be stated as:

Given a graph $G(V, E)$ with non-negative costs on its edges and sizes on its nodes, a k-cut, C, partitions the nodes of G into k subsets $P_1, P_2, \cdots, P_k$, of balanced size, such that C is a balanced partition and $G_m(C)$ is minimized. We denote this problem as MMP (Minimizing G_m Problem).

2 Related Work

A well-known similar problem is called the uniform k-way partitioning problem.

Given a graph $G(V, E)$ with non-negative costs on its edges and sizes on its nodes, a k-cut, C, partitions the nodes of G into k subsets $P_1, P_2, \cdots, P_k$, of balanced size, such that $G_s(C)$ is minimized. We denote this problem as MSP (Minimizing G_s Problem).

MSP has been extensively studied in the past and is NP-complete, so researchers have been focused on finding fast heuristic algorithms.

Kernighan-Lin's algorithm is the basis of these heuristic algorithms[1]. Their algorithm uses a pairwise-exchange scheme to transform an existing partition. The time complexity of this algorithm is greater than $O(N^2\log_2 N)$, where N is the number of nodes in the graph.

Many researchers have improved Kernighan-Lin's algorithm in various ways[2-4]. Fiduccia and Mattheyses proposed the one-move idea which means moving one element at a time as the basic technique to transform an existing partition to reduce the problem[2]. Krishnamurthy improved the algorithm by using more sophisticated heuristics[3], but both algorithms only improve the performance for the 2-way partitioning problem. Lee, Kim and Park transformed the MSP into a max k-cut problem and used the one-move technique[4]. (We denote their algorithm as the L-K-P algorithm.) The L-K-P algorithm also deals with nodes of various sizes without performance degradation and works well in k-way partitioning. Its computing time for a pass is $O(kN^2)$.

The MMP is defined for the first time in this paper. MMP is shown to be NP-complete and a heuristic algorithm is developed that extends the L-K-P algorithm to solve the MMP.

3 Heuristic Algorithm

3.1 NP-completeness of MMP

The Bisection Width problem is as follows. Given a graph $G(V, E)$, look for a cut S, $V-S$ of size M or less such that $|S| = |V-S|$. The Bisection Width problem is NP-complete[5].

For MMP, examine the special case where the sizes of all nodes are 1 and $k=2$. The MMP is the Bisection Width in this special case, which is NP-complete.

This proves that MMP is NP-complete, even for the special case $k=2$.

3.2 Heuristic algorithm definition

3.2.1 Goal functions

Define the goal functions,

$$C_s(C) = \sum_{1\leqslant i\leqslant k} S_{ws}^3(P_i),$$

$$T_{cs}(C) = \sum_{1\leqslant i<j\leqslant k} S(P_i)\cdot S(P_j)\cdot R - C_s(C),$$

where R is a predefined constant, $R \in Z^+$ and $R > \left(\sum_{i=1}^{|E|} W(e_i) \right)^3$.

3.2.2 Gain function

Define a gain function which indicates the gain in value of $T_{cs}(C)$ when a node v_r moves from one partition to another partition P_j, where C is the original cut and C^* is the new cut after v_r moves to P_j.

$$g(v_r, P_j) = T_{cs}(C^*) - T_{cs}(C).$$

3.2.3 Node to partition weighted sum

Define a node to partition weight sum to more quickly calculate the gain function,

$$W_{n2p}(v_r, P_j) = \sum_{u \in P_j} W((u, v_r)).$$

W_{n2p} is the weighted sum of the edges from a node to a subset. Another function, which represents the weighted sum of the edges from a node to all the subsets except the specified one, is

$$W'_{n2p}(v_r, P_j) = \sum_{u \notin P_j} W((u, v_r)).$$

3.3 Problem transformation

Since MMP is NP-complete, a fast heuristic algorithm is needed to solve it. Because the potential application of MMP is scheduling tasks for SMP clusters, the algorithm should have good performance when k is quite large. The L-K-P algorithm has good performance in dealing with large k, so the L-K-P algorithm was chosen as a starting point. The problem transformation and the goal function were modified to solve the MMP.

Naturally, G_m would be directly minimized in the heuristic algorithm to solve the MMP. But our experience shows that G_m is not a good candidate because it will be often trapped in a local optimal value. In most cases, it even fails to obtain a balanced partition. Therefore, C_s was used instead of G_m. Intuitively, the effect of larger S_{ws} value is enlarged in C_s. A cubic function was used instead of a square function or a higher order function to simultaneously evaluate the "enlarging effect", the precision and the computing complexity. Our experiments indicate that the cubic function is the most appropriate. The original MMP is transformed to the following problem.

Given a graph $G(V, E)$ with non-negative costs on its edges and sizes on its nodes, a k-cut, C, partitions the nodes of G into k subsets $P_1, P_2, \cdots, P_k$, of balanced size, such that $C_s(C)$ is minimized. We denote this problem as Minimizing Cubic Sum Problem (MCSP).

The MCSP still has two goals (1) Balanced partition and (2) Minimizing $C_s(C)$. Efficient heuristics are developed by transforming the problem to incorporate one goal into the other. The L-K-P algorithm transformed the original graph into another graph by reassigning the weight of each edge. We do not directly transform the edge weight to create a new graph in our algorithm, instead, we just transform the goal function. Our method simplifies the calculational process and is easy to combine with other destination functions. The transformed problem is as follows.

Given a graph $G(V, E)$ with non-negative costs on its edges and sizes on its nodes, a k-cut, C, partitions the nodes of G into k subsets $P_1, P_2, \cdots, P_k$, such that $T_{cs}(C)$ is maximized. We denote this problem as Transformed Maximizing Cubic Sum Problem (TMCSP).

Now we prove that the solution partition of TMCSP is the solution partition for MCSP.

Theorem 1 A solution partition of TMCSP is also a solution partition of MCSP.

Proof Let a solution partition of TMCSP, C, partition the nodes of G into k subsets $P_1, P_2, \cdots, P_k$. Assume that the partition C is not balanced. Let cut C_o be a balanced partition of G, which partitions G into k subsets $P_{1o}, P_{2o}, \cdots, P_{ko}$.

Since

$$\sum_{i=1}^{k} S(P_i) = \sum_{i=1}^{k} S(P_{io}) = \text{const}$$

and by definition of the balanced partition,

$$\sum_{1 \leqslant i < j \leqslant k} |S(P_{io}) - S(P_{jo})| < \sum_{1 \leqslant i < j \leqslant k} |S(P_i) - S(P_j)|,$$

then,

$$\sum_{1 \leqslant i < j \leqslant k} S(P_{io}) \cdot S(P_{jo}) \geqslant \sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) + 1.$$

By the definition of TMCSP,

$$T_{cs}(C) \geqslant T_{cs}(C_o).$$

So,

$$\sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \cdot R - C_s(C) \geqslant \sum_{1 \leqslant i < j \leqslant k} S(P_{io}) \cdot S(P_{jo}) \cdot R - C_s(C_o) \geqslant \sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \cdot R + R - C_s(C_o).$$

Then,

$$R \leqslant C_s(C_o) - C_s(C),$$

which is contradictory to the definition of R, which proves that C is a balanced partition.

Since $\sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \cdot R$ is constant for all balanced k-cut, for $T_{cs}(C)$ to be maximized, $C_s(C)$ must be minimized among balanced k-cuts.

This proves that C must be a solution of MCSP.

3.4 Algorithm

This section describes the heuristic algorithm to solve the TMCSP. In essence, the method starts with any arbitrary partition C and tries to increase $T_{cs}(C)$ by choosing one node from one subset and moving it to another. The node to be moved is chosen to obtain the maximum increase in the $T_{cs}(C)$. The moving of the node forms a new partition which is used to move the next node among the unmoved nodes. Moving all the nodes finishes a pass. If a pass can not improve $T_{cs}(C)$ any more, or the improvement is very small (i.e., less than a pre-defined value, V_{GATE_VALUE}), the algorithm ends. Otherwise another pass is begun based on the new partition. Convergence of this algorithm is easy to prove. Any starting partition has just a limited gain function value (V_{INIT_GAIN}) and any graph has a limited gain function upper bound (V_{MAX}). Increasing the gain function at least V_{GATE_VALUE} per pass would run at most (V_{MAX} - V_{INIT_GAIN}) / V_{GATE_VALUE} + 1 passes before stopping.

A key step in the algorithm is to choose the node movement by calculating the gain function of each possible move in a pass. The calculation of gain function should be as fast as possible.

Lemma 1 When a node v_r moves from P_{from} to another partition P_{to},

$$S_{ws}(P^*_{from}) = S_{ws}(P_{from}) + W_{n2p}(v_r, P_{from}) - W'_{n2p}(v_r, P_{from}),$$

$$S_{ws}(P^*_{to}) = S_{ws}(P_{to}) - W_{n2p}(v_r, P_{to}) + W'_{n2p}(v_r, P_{to}),$$

where P^*_i, $1 \leqslant i \leqslant k$ is the corresponding subset after the movement.

Proof The definitions of S_{ws}, W_{n2p}, and W'_{n2p} can be used to easily prove the lemma. The proof is omitted for simplification.

Theorem 2 When a node v_r moves from P_{from} to another partition P_{to},

$$g(v_r, P_{to}) = (S(P_{from})) - S(P_{to}) - 1)R + (S^3_{ws}(P_{from}) + S^3_{ws}(P_{to}) - S^3_{ws}(P^*_{from}) - S^3_{ws}(P^*_{to})),$$

where P^*_i, $1 \leqslant i \leqslant k$ is the corresponding subset after the movement.

Proof By definition

$$g(v_r, P_j) = T_{cs}(C^*) - T_{cs}(C) = \left(\sum_{1 \leqslant i < j \leqslant k} S(P^*_i) \cdot S(P^*_j) \cdot R - C_s(C^*) \right) - \left(\sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \cdot R - C_s(C) \right) = \left(\sum_{1 \leqslant i < j \leqslant k} S(P^*_i) \cdot S(P^*_j) - \sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \right) \cdot R + (C_s(C) - C_s(C^*)).$$

Since $S(P_i) = S(P^*_i)$ for all $i \neq$ from and $i \neq$ to, and

$$S(P_{from}) = S(P^*_{from}) + 1,\ S(P_{to}) = S(P^*_{to}) - 1,$$

then,

$$\sum_{1 \leqslant i < j \leqslant k} S(P^*_i) \cdot S(P^*_j) = \sum_{1 \leqslant i < j \leqslant k, i \neq from, i \neq to, j \neq from, j \neq to} S(P^*_i) \cdot S(P^*_j) + \sum_{1 \leqslant i \leqslant k, i \neq from} S(P^*_{from}) \cdot S(P^*_i) + \sum_{1 \leqslant i \leqslant k, i \neq to} S(P^*_{to}) \cdot S(P^*_i) - S(P^*_{from}) \cdot S(P^*_{to}) = \sum_{1 \leqslant i < j \leqslant k, i \neq from, i \neq to, j \neq from, j \neq to} S(P^*_i) \cdot S(P^*_j) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} S(P^*_{from}) \cdot S(P^*_i) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} S(P^*_{to}) \cdot S(P^*_i) + S(P^*_{from}) \cdot S(P^*_{to}) = \sum_{1 \leqslant i < j \leqslant k, i \neq from, i \neq to, j \neq from, j \neq to} S(P_i) \cdot S(P_j) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} (S(P_{from}) - 1) \cdot S(P_i) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} (S(P_{to}) + 1) \cdot S(P_i) + (S(P_{from}) - 1) \cdot (S(P_{to}) + 1) = \sum_{1 \leqslant i < j \leqslant k, i \neq from, i \neq to, j \neq from, j \neq to} S(P_i) \cdot S(P_j) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} S(P_{from}) \cdot S(P_i) + \sum_{1 \leqslant i \leqslant k, i \neq from, i \neq to} S(P_{to}) \cdot S(P_i) + S(P_{from}) \cdot S(P_{to}) + (S(P_{from}) - S(P_{to}) - 1) = \sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) + (S(P_{from}) - S(P_{to}) - 1),$$

So,

$$g(v_r, P_j) = \left(\sum_{1 \leqslant i < j \leqslant k} S(P^*_i) \cdot S(P^*_j) - \sum_{1 \leqslant i < j \leqslant k} S(P_i) \cdot S(P_j) \right) \cdot R + (C_s(C) - C_s(C^*)) = (S(P_{from}) - S(P_{to}) - 1) \cdot R + (C_s(C) - C_s(C^*)).$$

By definition,

$$C_s(C^*) = \sum_{1 \le i \le k} S^3_{ws}(P^*_i) = \sum_{1 \le i \le k, i \ne from, i \ne to} S^3_{ws}(P^*_i) + S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to}).$$

Since $S_{ws}(P_i) = S_{ws}(P^*_i)$ for all $i \ne$ from and $i \ne$ to,

then,

$$C_s(C^*) = \sum_{1 \le i \le k, i \ne from, i \ne to} S^3_{ws}(P^*_i) + S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to}) = \sum_{1 \le i \le k, i \ne from, i \ne to} S^3_{ws}(P_i) + S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to}) =$$
$$\sum_{1 \le i \le k} S^3_{ws}(P_i) + S^3_{ws}(P_{from}) + S^3_{ws}(P_{to}) - (S^3_{ws}(P_{from}) + S^3_{ws}(P_{to})) + S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to}) =$$
$$C_s(C) - (S^3_{ws}(P_{from}) + S^3_{ws}(P_{to})) + S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to}).$$

Since

$$g(v_r, P_j) = (S(P_{from}) - S(P_{to}) - 1) \cdot R + (C_s(C) - C_s(C^*)),$$

then,

$$g(v_r, P_j) = (S(P_{from}) - S(P_{to}) - 1) \cdot R + (C_s(C) - C_s(C^*)) =$$
$$(S(P_{from}) - S(P_{to}) - 1) \cdot R + (S^3_{ws}(P_{from}) + S^3_{ws}(P_{to})) - (S^3_{ws}(P^*_{from}) + S^3_{ws}(P^*_{to})).$$

Figure 2 shows the algorithm details. The MMP solver is very similar to the L-K-P algorithm. The main difference is the method to calculate $g(v_r, P_j)$, which in this method is based on Lemma 1 and Theorem 2, which means we have to calculate and update arrays S_{ws}, W_{n2p}, and W'_{n2p}.

Algorithm MMP-solver
Input Graph G
Output k-partition// PART[1..|V|]
Variables
PART[1..|V|]: integer; // PART[i]= partition number of v_i
GAIN[1..|V|][1..k]: real;// GAIN[i][j]= $g(v_i, P_j)$
STATE[1..|V|] boolean;// 0= unused, 1= used
HISTORY[1..|V|][1..2] integer; // HISTORY[i][1] saves the node to move
// HISTORY[i][2] saves the original partition this node
// belongs to.
TEMP[1..|V|]: integer; // temporary gain for move history
SWS[1..k][1..k] real; // SWS[i][j]= $S_{ws}(i, j)$
N2P[1..|V|][1..k] real; // N2P[i][j]= $W_{n2p}(v_i, P_j)$
N2P1[1..|V|][1..k]: real; // N2P[i][j]= $W'_{n2p}(v_i, P_j)$
S[1..k]: integer; // S[i]= $S(P_i)$
Begin
(1) Construct an initial partition; // PART[i]= 1 for all i
(2) Calculate S[1..k], WSW[1..k][1..k], N2P[1..|V|][1..k] and N2P[1..|V|][1..k] based on current partition
(3) for i:= 1 to |V| do
STATE[i]:= 0;// UNUSED
for j:= 1 to k do
GAIN[i][j]:= $g(v_i, P_j)$// Calculate GAIN[i][j] based on Lemma 1 and Theorem 2
endfor
endfor
(4) for i:= 1 to |V| do
(4.1) select unused v_d such that $g(v_d, P_n) = \max_{j,p}(GAIN[j][p])$// $v_d \in P_m$
(4.2) Recalculate S[1..k], WS[m][1..k], SWS[n][1..k], N2P[1..|V|][m], N2P[1..|V|][n], N2P1[1..|V|][m] and N2P1[1..|V|][n] after moving v_d to P_n
(4.3) STATE[d]:= 1// v_d is used
(4.4) PART[d]:= n// Change the partition
(4.5) HISTORY[i][1]:= d// Save the node to be moved
(4.6) HISTORY[i][2]:= m// Save the node's original partition number
(4.7) TEMP[i]:= GAIN[d][n]// Save gain of this moving
(4.8) for i:= 1 to |V| do// Update gain according to the new partition
for j:= 1 to k do

```
        GAIN[i][j]:= g(v_i, P_j) // Calculate GAIN[i][j] based on Lemma 1 and Theorem 2
      endfor
    endfor
  endfor
(5) Choose t to maximize G = Σ_{j=1}^{t} TEMP[j];
(6) if G > 0 then // complete one pass
    for j:= t+1 to |V| do
      PART[HISTORY[j][1]]:= HISTORY[j][2];
    endfor
  endif
end
```

Fig. 2 MMP solver algorithm

3.5 Algorithm complexity

First analyze the complexity of calculating $g(v_i, P_j)$. According to Lemma 1, $S_{ws}(P^*_{from})$ and $S_{ws}(P^*_{to})$ are calculated in $O(1)$ time if we maintain an array of current SWS[], N2P[], and N2P1[]. Then calculate $g(v_i, P_j)$ based on Theorem 2. The time complexity is also $O(1)$. Thus, calculating one $g(v_i, P_j)$ requires $O(1)$ time.

For the MMP solver algorithm, Step 1 needs $O(k)$ time. The complexity of Step 2 is $O(|V|^2)$. The complexity of Step 3 is $O(k|V|)$ because calculating one gain function needs $O(1)$ time. Each iteration of Step 4 needs $O(k|V|)$ time, which is mainly due to Step 4.8. So the complexity of Step 4 is $O(k|V|^2)$. The complexity of Step 5 and 6 is $O(|V|)$. Therefore, Step 2-step 6 need a total of $O(k|V|^2)$ computing time, which is the complexity for one algorithm pass.

4 Performance

The algorithm was implemented on a Sun Ultra5 workstation with the Solaris 2.5 operating system. The algorithm was compiled by GNU gcc 2.7.2 with the "-O2" option. We tested cases from 16 nodes to 128 nodes. For all cases, 50 graphs were generated randomly. Because the main potential application of this problem is scheduling SMP clusters, we set the size of all nodes to 1, which indicates that each processor in the SMP cluster has the same computational power. There are approximately $0.05 * |V|^2$ edges in the graph, where $|V|$ is the number of nodes in the graph. The result is shown in Table 1. In Table 1, "Mean" is the average value of $G_m(C)$ found by the algorithm for the 50 graphs, t is the average time for solving one graph partition problem.

Table 1 Experimental results of MMP solver

Number of nodes ($\|V\|$)	Number of partitions (k)	MMP-Solver		L-K-P's Algorithm	
		Mean	t/ms	Mean	t/ms
16	4	202	3.1	214	0.5
16	8	334	4.6	337	0.7
32	4	874	19.4	899	2.7
32	8	820	36.6	883	4.1
32	16	786	57.4	857	6.4
64	4	274	100.3	314	13.7
64	8	3157	189.8	3354	19.4
64	16	2150	361.4	2414	32.9
64	32	1585	566.5	1728	59.5
128	4	20 085	480.9	20 488	78.0
128	8	13 401	990.1	13 694	141.6
128	16	8234	1803.5	8772	187.9
128	32	5067	3332.3	5454	311.3
128	64	3186	3474.4	3487	552.7

(**Continued on page 41**)

the acquisition of the initial values for the leaf nodes and the method for processing uncertain information.

References

[1] Robert J Graves, Ashutosh Agrawal, Katharine Haberle. Estimating tools to support multipath agility in electronics manufacturing. IEEE Transaction on Components, Packaging, and Manufacturing, 1996, 19(1): 48-56.

[2] Elsayed A Orady, Osman T A. Capability study of robotics and manufacturing cell simulation software. Computers and Industry Engineering, 1997, 33(1/2), 83-86.

[3] Deng Hongchou, Hu Chunxiao, Liu Xianzhu. Research on agile manufacturing system. Computer Integrated Manufacturing System, 1997, (6): 24-29. (in Chinese)

[4] Qin Zheng, Lu Bingheng. Integrated decision of agile manufacturing. Mechanical Engineering of China, 1997, 8(6): 12-14. (in Chinese)

(**Continued from page 23**)

The data in Table 1 shows the following. (1) In all cases, the MMP solver determines a better mean solution than the L-K-P algorithm. (2) Generally speaking, for a fixed number of nodes, the larger k, MMP solver outperforms L-K-P algorithm more. For the scheduling problem for SMP clusters, an SMP usually has 2, 4 or 8 nodes. Therefore, k in the practical situations is quite large. (3) The computing time for MMP solver is longer than that for the L-K-P algorithm, but the complexity of the two algorithms is the same. The speed of the MMP solver is still acceptable as a backend in an optimizing compiler.

5 Conclusions

This paper defines a new graph partition problem (MMP) that can be used in scheduling tasks for SMP clusters. MMP is shown to be NP-complete and a heuristic algorithm is developed for it based on the L-K-P algorithm, which was originally used for solving MSP. Experimental results indicate that our algorithm outperforms the L-K-P algorithm, especially when the number of partitions is large.

Acknowledgement

The authors wish to thank Deng Xiaotie for his numerous and helpful suggestions and remarks.

References

[1] Kernighan B W, Lin S. An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J, 1970, 49(2): 291-307.

[2] Diduccia C M, Matteyses R M. A linear-time heuristic for improving network partitions. In: Proc 19-th Design Automation Conf. IEEE Press, 1982. 175-181.

[3] Krishnamurthy B. An improved Min-Cut algorithm for partitioning VLSI networks. IEEE Trans on Computers, 1984, 33(5): 438-446.

[4] Lee C H, Kim M, Park C I. An efficient K-way graph partitioning algorithm for task allocation in parallel computing systems. In: Ng P A, Ramamoorthy C V, Seifert L C, eds. Proc of System Integration 90. IEEE Press, 1990. 748-751.

[5] Christos H P. Computational Complexity. Addison-Wesley Publishing Company, 1994.