ISSN1000-9825 Journal of Software 软件学报 2000,11(7):889~898

An Efficient Parallel Minimum Spanning Tree Algorithm on Message Passing Parallel Machine*

WANG Guang-rong GU Nai-jie

(Dept of Computer Science and Technology Univ. of Science and Technology of China Hefei 230027)
E-mail: john@mail.ustc.edu.cn/grwang@263.net

Abstract An efficient parallel minimum spanning tree is proposed based on the classical Borüvka's algorithm on message passing parallel machine. Three methods were used to improve its efficiency, including two-phase union and packaged contraction for reducing communication costs, and the balanced data distribution for computation balance in each processor. The computation and communication costs of the algorithm are $O(n^2/p)$ and $O((t_s p+t_w n)n/p)$. On Dawning-1000 parallel machine, it gets a speedup of 12 on 16 processors with a sparse graph of 10 000 vertices.

Key words MPP (message passing parallel), MST (minimum spanning tree), parallel algorithm, communication, disjoint set.

Given a connected, undirected graph $G=(V,E)$, $T=(V,E')$ is called the minimum spanning tree of G if and only if T is a spanning tree of G and $W(T)=\sum_{e\in T} w(e)$ is minimized. The problem of finding such a tree is called the minimum spanning tree problem. Finding minimum spanning tree of a given graph arises in many applications such as design of electronic circuits[1], design of communication networks, etc.

The histroy of MST can go back at least to Borüvka's work in 1926[2] with running time of $O(n^2)$ (we use n and m to denote the numbers of vertices and edges of a graph in the paper), and the other two classical MST (minimum spanning tree) algorithms are Kruskal's algorithm[1,3] and Prim's algorithm[1,4], which have a running time of $O(m\log n)$ and $O(m+n\log n)$ respectively. Much progress has been made on this probelm in the last decade. Fredman and Tarjan[5] established an $O(m\beta(n,m))$ bound algorithm where $\beta(n,m)$ is the number of logarithmic iterations necessary to map n to a number less than m/n. Gabow *et al.*[6] reduced it to $O(m\log\beta(n,m))$, while Bernard[7] published his algorithm of $O(m\alpha\log\alpha)$ where α is the inverse function of Ackerman's function. To the parallel algorithm, with CRCW PRAM model, Tsin *et al.*[8] proposed an $O(\log^2 n)$ algorithm with $O(n^2/\log^2 n)$ processors; with EREW PRAM model, Nath *et al.*[9] proposed an $O(\log^3 n)$ algorithm using $O(n^2/\log n)$ processors; and Huang[10] proposed an $O(n^2/p)$ time algorithm using p processors. To the MPP model, little progress has been made since the nature of heavy communication attribute of the problem, Sun Chung and Anne Condon[11] proposed an $O(m\log n/p)$ algorithm, with the speedup of 4 on 16-processor CM-5

* This research is supported by the Ph. D. Foundation of State Education Commission of China (国家教育部博士点基金, No. 9703825). **WANG Guang-rong** was born in 1973. He is an M. S. student at the Department of Computer Science and Technology, University of Science and Technology of China. He received a B. S. degree in computer science from the University of Science and Technology of China in 1997. His research interests are parallel processing, parallel algorithm and computer architecture. **GU Nai-jie** was born in 1961. He is an associate professor at the Department of Computer Science and Technology, University of Science and Technology of China. His current research areas include communication algorithm in high-performance computing, parallel algorithm and parallel processing.

Manuscript received 1999-01-21, accepted 1999-07-20.

machine.

We proposed a parallel algorithm on the MPP model, based on the classical sequential algorithm of Borůvka[2]. It has a linear speedup in principle, but due to communication cost and imbalance graph distribution, we cannot get it in practice. We develop some methods to conquer the problems. To the first problem, we developed the two-phase union and the packaged contraction algorithm to reduce communication complexity. And to the second, we discuss two new distribution methods to balance the graph distribution. The algorithm fits well on the machine which has much less processors than the number of vertices of the graph. Our experiments show that reducing communication cost gets good result while balancing the graph distribution improves little.

The algorithm is discribed in Section 1 and is analyzed the improved algorithm in Section 2. The experimental results are given in Section 3.

1 The Algorithm

The parallel algorithm is based on Borůvka's sequential MST algorithm[2], so we introduce this algorithm first.

1.1 Borůvka's MST algorithm

Given a graph G, select the lightest weighted edges incident on each vertex, and add the edges to the edge set of MST. Then, each connected component formed by the selected edges of graph G is a tree. Contract each connected part into one vertex, delete the self-loops and delete the redundant edges by deleting those edges with bigger weight. Thus, we get a new graph G'. Apply the above process to G' repeatedly, until there's only one vertex left in the graph. Then we get the edge set of the minimum spanning tree of G. For there are at most half of the vertices left in the graph after one iteration, the algorithm can be finished in at most $\lceil \log n \rceil$ iterations. Hence, the complexity of this algorithm is

$$\underbrace{O\left(n^2+\left(\frac{n}{2}\right)^2+\left(\frac{n}{4}\right)^2+\ldots\right)}_{\lceil \log n \rceil}=O(n^2). \tag{1}$$

1.2 The parallel algorithm

On the message passing parallel machine, we can parallelize the above algorithm based on the following idea. First, we distribute the vertices of the graph evenly among the processors, each processor finds the lightest weighted edges incident on every vertex belonging to it, then, compute the connected components of the graph formed by these edges, and contract each connected component into one vertex. If there is only one vertex left, we have found the result, otherwise, repeat the above procedure. Now, we give out the formal description of the algorithm:

Algorithm 1. Parallelization of Borůvka's MST Algorithm

Input. The weight matrix w of connected undirected graph G

Output. The edge set of the minimum spanning tree of G

Begin

(1) $T \leftarrow \varnothing$

(2) While $|T| < n-1$ Do

(2.1) For each processor PE_i Par-Do

For each $v \in V(G)$ and v in PE_i Do

$key[v] \leftarrow -\infty$ /* $key[v]$ is the lightest weight of all edges incident on v */

$\pi[v] \leftarrow v$ /* $(v\pi[v])$ is the lightest weighted edge incident on v */

```
            Init-Set(v)
        Endfor
    Endfor
    /* find connected part */
(2.2) For each processor PE_i Par_Do
        For each u∈V(G) and u in PE_i Do
        (i) For each v∈adj(u) Do
                If w[u,v]<key(u) Then
                  key[u]←w[u,v]
                  π[u]←v
                Endif
            Endfor
        (ii) T←T∪{u,π[u]}
            Synchronize
        (iii) Union(u,π[u])
        Endfor
    Endfor
    /* contract the connected part */
(2.3) For each processor PE_i Par_Do
        For each u∈V(G) Do
            If Find(u)≠u Then
              For each v∈V(G)&v in PE_i Do
              (i) If Find(u)=Find(v) Then /* self loop */
                    Set_Value(w[Find(u)][Find(v)],∞)*
                    Continue
                  Endif
              (ii) If w[Find(u),v]>w[u,v] Then /* redundant edge */
                    Set_Value(w[Find(u),Find(v)],w[u,v])
                    Set_Value(w[Find(v),Find(u)],w[u,v])
                  Endif
              Endfor
            Endif
        Endfor
    Endfor
Endwhile
End
```

The union procedure in Step 2.2 (iii) is a parallel version of serial disjoint set union operation[3], and so is the find operation. So, when two parts that are not in the same processor are unioned by the procedure, it

* Set_Value($w[x][y],w'$) is a procedure that compares the values of $w[x][y]$ and w'. If $w[x][y]$ is bigger, set it to w'. In the case that $w[x][y]$ is not in the host processor, send it to the processor which $w[x][y]$ belongs to, and do the same thing.

requests that one processor should set (or get) the parent value a vertex that is not in the same processor. It does so by sending the value (request) to that processor, waiting for the acknowledgement (answer) message from that processor. This intuitive implementation method seems easy, but the waiting process may cause deadlock in practice. So, we use the two-phase union procedure that will be given in the next section when we write the algorithm, but while analyzing the running time complexity, we still use the intuitive method to make analyses easy.

In the contraction Step 2.3, when a processor wants to set the new value of an entry of weight matrix w, while the entry is not in the host processor, we must send the new value to that processor. Waiting for the acknowledgement message has to be applied to make sure the value reached its destination and for synchronization.

1.3 Running time

We'll analyze the computation and communication complexities separately.

• Computation complexity:

First, consider the computation complexity of one iteration. Assume the graph has n vertices. Step (2.1) can be finished in $O(n^2/p)$ time to initialize the value of *key*, π and initiate the set of each vertex. In Step (2.2): (i) needs $O(n)$ time to be finished; (ii) can be finished in a constant time; (iii) can be finished in no more than $O(n)$ time (in fact, for we must have more find operations than init-set operations, define f to be the number of find operations, and the average time of an operation is $O(\log_{(1+f/n)}n/p)$[1]). So Step (2.2) can be finished in $O(n^2/p)$ time. After that, the find procedure has been executed on every vertex of the graph, so, all the find procedure called in Step (2.3) can be finished in a constant time, thus we know that Step (2.3) can be finished in $O(n^2/p)$ time.

To the further iterations, assume that the graph is still evenly distributed among the processors after the kth iteration, and consider the worst case after the kth iteration, in which all vertices converge into one processor PE_0. For there are at most n/p vertices in one processor, we can use serial Borůvka's algorithm to compute it, so we could get the complexity of the algorithm:

$$O\Big(\underbrace{\frac{n^2}{p}+\frac{(n/2)^2}{p}+\frac{(n/4)^2}{p}+\dots}_{k}\Big)=O\Big(\Big(\frac{n}{p}\Big)^2\Big)=O(n^2/p). \tag{2}$$

• Communication complexity:

For the communication cost, we still consider the complexity of one iteration. Step (2.1) requires no communication. In Step (2.2): (i) needs no communication; using the standard pointer jump techniques, (ii) can be finished in a constant time of communication; (iii) requires $O((t_s+t_w)\log p)$[12](for we define only those vertices with minor label which can be used as *parent*, there must be no cycle among processors) time of communication. So, Step (2.2) can be finished with communication cost of $O((t_s+t_w)n\log p/p)$. And in Step (2.3), for there are at most half vertices being used as leader of set, so, at most half of all the values of matrix w are changed. In the worst case, they all require communication, the communication cost would be $(t_s+t_w)n^2/2p$. So, the communication complexity of one iteration is $O((t_s+t_w)n^2/p)$.

For further iterations, use the same analysis method as Eq. (2), and we can get the communication complexity of the above algorithm:

$$O\Big((t_s+t_w)\underbrace{\Big(\frac{n^2}{p}+\frac{n^2}{2p}+\frac{n^2}{4p}+\dots\Big)}_{\lceil\log p\rceil}\Big)<O((t_s+t_w)n^2/p). \tag{3}$$

Now, we can see that both the computation complexity and the communication complexity of the above algorithm are $O(n^2/p)$. We get a linear speed-up.

2 Some Improvements on the Algorithm

As we have mentioned in introduction, the major obstacles to the algorithm are communication cost and imbalance data distribution.

We know that in the distributed memory environment, the startup time (t_s) of a transfer is much bigger than the time to transport a word (t_w)[13] through the network. So, to reduce communication time, thus to transfer as much data as possible in a single transfer step is a good method to improve algorithms' performance. We give two-methods based on this idea: one is two-phase union, which reduces the factor of t_s in the union step from $n\log p/p$ to $\log p$ (but increases the factor of t_w from $n\log p/p$ to $n\log p$); the other is packaged contraction which reduces the communication cost of contraction from $(t_s+t_w)n^2/2p$ to $t_s p+t_w n^2/4p$. In practice, we found these methods improve the algorithm's performance greatly.

To the imbalance data distribution, we give out a practical useful algorithm. We know, in practice, people usually name the vertices by eyes or by graph traveling algorithms. Both cause the connected vertices to be named adjacently. So, we give out two distribution methods to reduce data imbalance as computation goes on. But we found these methods contribute only little progress on the algorithm's performance, and we'll discuss the reason later.

2.1 Two-phase union

We first describe the union and find algorithm:

Algorithm 2. Serial Union Algorithm

```
Union (u,v)
Begin
    If Find(u)≠Find(v) Then
      If Find(u)< Find(v) Then
        parent [Find(v)]←Find(u)
      Else
        parent [Find(u)]←Find(v)
      Endif
    Endif
End
```

Algorithm 3. Path Compression Find Algorithm

```
Find(u)
Begin
    If u≠parent[u] Then
      parent[u]=Find(u)
    Endif
    Return parent[u]
End
```

In distributed memory environment, when PE_i wants to get parent value of vertex u while u stands in PE_j ($i\neq j$), PE_i must query it from PE_j and wait for the answer from PE_j, which causes twice communication, and the same number of transfers occurs when PE_i wants to set the parent value of u, for it must send the new value to PE_j and wait for an acknowledge message from PE_j to assure the value has reached its destination. And a more serious problem is that this straight forward method may cause a dead lock when there exists a request

chain among the processors. Although this could be avoided by adding query while waiting, it is still a time costing procedure. So, we introduce a new two-phase algorithm to seperate local union and inter union.

We give a brief description of the procedure. Give a full parent array parent $[0,1,\ldots,n-1]$ to each processor, and use serial Set-Union algorithm in each processor to locally find connected part, and union of the connected part by union of $parent_{PE_{0\sim(p-1)}}[0]$, $parent_{PE_{0\sim(p-1)}}[1],\ldots,parent_{PE_{0\sim(p-1)}}[n/p-1]$ respectively. This could be implemented by applying the serial union procedure to parent array two by two, thus we can get the final connected part of the graph. Assume the processors are hypercube* connected, we can achieve the final result in $\log p$ steps. So the communication complexity of Step (2.2) of the parallel algorithm is reduced to $O((t_s\log p+t_w n)n/p)$. The detailed description of the 2nd phase of the new union algorithm is,

Algorithm 4. Parallel Union Algorithm

```
Parallel Union (u,v)
Begin
    For i=0 To logp-1 Do
        /* iam is the binary representation of current processor number */
        peer← label of the processor that differs only at ith bit from iam
        Send (peer, parent, n)
        Recv (peer, tparent, n)
        For j=0 To n-1 Do
            Union(parent[j], tparent[j])
        Endfor
    Endfor
End
```

As an example, in Fig. 1 there are 8 processors that are hypercube connected, we can finish the job in 3 iterations. Though we have to send extra data than what we do with the original algorithm, it is worth the cost to reduce coefficient of t_s from $O(n)$ to $O(\log p)$.

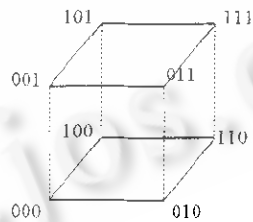


Fig. 1 Hypercube with 8 processors (all to all communication could be finished in 3 steps: 1st: 000-001 010-011 100-101 110-111, 2nd: 000-010 000-011 100-110 101-111, 3rd: 000-100 001-101 010-110 011-111)

2.2 Packaged contraction

Now, let's turn to the contraction procedure of the algorithm. In the straight forward algorithm, the communication cost is $(t_s+t_w)n^2/2p$. This is extremely large. The big coefficient of t_s is caused by the fact that a processor requests the value of parent immediately from other processors once it cannot find the value in itself. To reduce the cost, we store all the data sent to a processor to a specified buffer, send the buffer only after we

* The procedure can be applied for machines that are not hypercube connected without any change, using the standard embedding techniques[14].

have collected all the data. So, each processor sends at most $p-1$ packages in one iteration. That means, the cost of t_s is charged at most p times in one iteration, which is far less than $n^2/2p$.

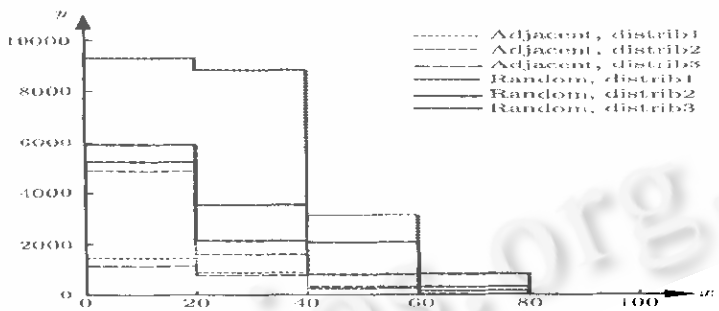
Further more, we can still reduce the factor of t_w. For there may be redundant-edges between two contracted vertices, thus the weight value in the weight matrix w may change multi-times, and some value may come from the same processor. So, instead of sending them separately, we can contract them in the host processor first and then send a single value, although this will cause an increase of computation about n, but it can reduce the coefficient of t_w from $n^2/2p$ to $n^2/4p$.

Table 1 Running time of different contraction methods (ms)

Method	Random graph $d=3$			Adjacent graph $d=3$		
	4	8	16	4	8	16
Origional contra	186.94	95.21	51.81	125.13	56.18	33.84
Packaged contra	123.51	67.23	37.5	111.53	60.2	35.6

Table 1 shows the experiment results. Random graph and adjacent graph are two graph types we used to test the algorithm. Random graph means that every vertex is randomly connected to other vertices. Adjacent graph means that it is more probable an edge exists between two vertices that are named adjacently than those are named separately. This is more factual for usual graph named by eyes or by graph travel algorithm, both will cause the case.

From the table, we can see that the packaged contraction method does improve the algorithm's performance greatly, especially for the random graph. The reason is that there are more communication requests for the random graph than those for the adjacent graph, which we can see in Fig. 2.



x: percent of computation, y: communication time

Fig. 2 Communication time

From Fig. 2, we can see that despite of the distribution method, the communication time of random graph is much bigger than the adjacent graph).

2.3 Distribution method

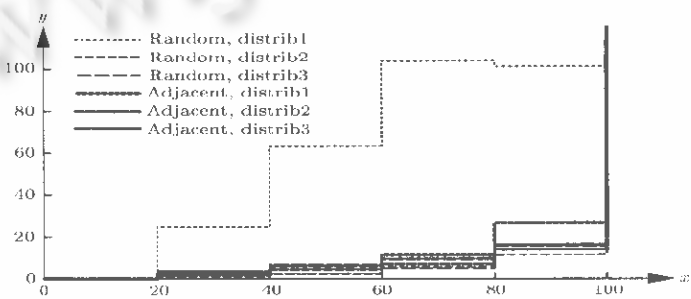
The intuitive graph distribution method is distributing the graph serially, that is: vertex $0,1,\ldots,n/p-1$ on PE_0, vertex $n/p,n/p+1,\ldots,2n/p-1$ on $PE_1,\ldots$, vertex $(p-1)n/p,(p-1)n/p+1,\ldots,n-1$ on PE_{p-1}. This distribution method will cause an imbalance data distribution for we know that when we union two connected parts, we choose the one with minior parent label as the unioned parent. This cause all the vertices to converge to the processor with minior label, thus may cause the data imbalance distribution among the processors in the later steps of the algorithm.

To avoid this imbalance, we distribute the graph cyclely, that is: vertex $0,n/p,\ldots,(p-1)n/p$ on PE_0, vertex $1,n/p+1,\ldots,(p-1)n/p+1$ on $PE_1,\ldots$, vertex $n/p-1,2n/p-1,\ldots,n-1$ on PE_{p-1}. By this

method, the graph is still balanced when the algorithm is nearly finished. But in practice, we found it improved the random graph only, while to the adjacent graph, it gave even worse effect. We analyzed the reason and found that to the adjacently graph, the communication cost increased greatly with the cycle distribution method. Since we always distribute the adjacent numbered vertices to different processors, but it is more probable for them to be in one connected part, it is obvious that the communication cost will increase. So we still need a better method to balance it.

We develop the third method. Here, we still distribute the graph as what we do in the first method, but make some change to the union procedure, instead of choosing the one with minior label as new parent, we choose the one with minior value of a function of vertex label to be the new parent, and the function is $f(x)=(x \bmod p)$. For those with the same return value, we still choose the one with minior label to be the new parent. Thus we got a solution to the distribution problem.

In Fig. 3 and Table 2, we give a comparison of data balancing and computation time of three distribution methods for the random graph and the adjacent graph. From the figure, we can see that the random graph using the first distribution method is very imbalance, so the computation time is much bigger than the others as Table 2 shows.



x: percent of computation, y: percent of imbalance

Fig. 3 Imbalance in the distribution of vertices among 16 processors

Table 2 Running time of different distribution methods (ms)

Method	Random graph $d=3$			Adjacent graph $d=3$		
	4	8	16	4	8	16
1	123.51	67.23	37.5	103.8	54.15	28.57
2	111.53	60.2	35.6	106.33	57.85	37.34
3	111.1	58.05	33.14	104.64	53.9	33.15

With all the above improved methods, we can reduce the communication cost from $O((t_s+t_w)n^2/p)$ to $O((t_s p+t_w n)n/p)$.

3 Experimental Result

We measured the total running time of our algorithm on Dawning-1000 parallel machine, using the two-phase union, packaged contraction, and the third distribution method, for random graph and adjacent graph with different degrees. The results are given in Figs. 4 and 5. From the figures, we can see that we got a speedup of 12 on random graph with an average degree of 3 on 16 processors, and the result differs a little for different graph types and for the graph with different degrees. This is much better than any distributed algorithm ever known[11].

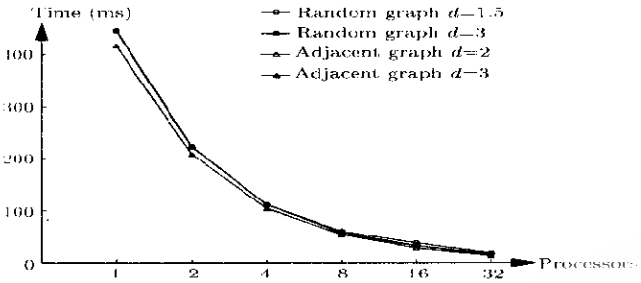


Fig. 4 Running time of the improved algorithm

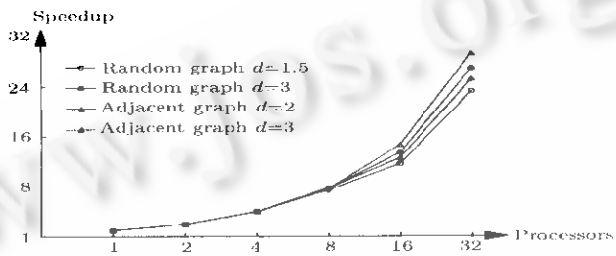


Fig. 5 Speed up of the improved algorithm

The computation results using 1 and 2 processors are given by linear probing because we cannot get it for memory constraint. But we still draw it for the consecutive of the graph.

4 Conclusions

The message passing model parallel machine is developing rapidly these years, but programming on it is difficult, and how to improve its efficiency is more difficult. We give an efficient MST algorithm on such model, analyze its efficiency and the slow downs, give an improved high-efficiency algorithm and get good result. The computation and communication complexities of the algorithm are $O(n^2/p)$ and $O((t_s p+t_w n)n/p)$ respectively. We got a speed-up of about 12 on 16 processors for the sparse graph with the average degree of 1.5 and 3. Our algorithm is suitable for many models of machine, such as mesh linear-array, hypecube etc.

The minimum spanning tree problem itself is very important. We give an $O(n^2/p)$ algorithm on MPP machine. The speed-up we got is the best for this problem on MPP machine.

References

1 Lormen T H, Leiserson C R, Rivert R L. Introduction to Algorithms. Massachusetts: The MIT Press, 1990. 77~81

2 Borůvka O. O jistem problemu minimalnúm. Práce Mor. Přírodověd Spol v Brně (Acta Societ Scient Natur Moravicae), 1926,3:37~58

3 Kruskal J B. On the shortest subtree of a graph and the travelling salesman problem. In: Proceedings American Math Society, 1956,7(1):48~50

4 Prim R C. Shortest connection networks and some generalizations. Bell System Technology Journal, 1951,B. 6(6):1389~1401

5 Fredman M L, Tarjan R E. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of ACM, 1987,34:596~615

6 Gabow H N, Galil Z, Spiencer T *et al*. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica, 1986,6:109～122

7 Bernard Chazelle. A fast deterministic algorithm for minimum spanning trees. In: Proceedings of the 38th Annual Symposium on Foundation of Computer Science. 1997. 20～22

8 Tsin Y H, Chin F Y. Efficient parallel algorithm for a class of graph theoretic problems. SIAM Journal of Computer, 1984,13(3):580～590

9 Nath R, Maheshwari S N. Parallel algorithms for the connected components and minimal tree problems. Information Proceedings Letter, 1982,14(1):7～11

10 Huang M D. Solving some graph problems with optimal or near-optimal speedup on mesh-of-tree problems. In: Proceedings of the 26th Annual. IEEE Symposium, Portland, 1985. 232～240

11 Sun Chung, Anne Condon. Parallel implementation of Borüvka's minimum spanning tree algorithm. In: Proceedings of IPPS'96. Hawii,1996. 302～308

12 Cheng Guo-liang. The Design and Analyses of Parallel Algorithm. Beijing: Advanced Education Publishing House, 1994. 30～32

13 Vipin Kumar, Ananth Grama, Anshul Gupta *et al*. Introduction to Parallel Computing. San Francisco: The Benjamin/Cummings Publishing Company, 1994. 45～49

在消息传递并行机上的高效的最小生成树算法

王光荣　顾乃杰

(中国科学技术大学计算机科学技术系　合肥　230027)

摘要　基于传统的 Borů vka 串行最小生成树算法,提出了一个在消息传递并行机上的高效的最小生成树算法.并且采用 3 种方法来提高该算法的效率,即通过两趟合并及打包收缩的方法来减少通信开销,通过平衡数据分布的办法使各个处理器的计算量平衡.该算法的计算和通信复杂度分别为 $O(n^2/p)$ 和 $O((tsp+twn)n/p$. 在曙光-1000 并行机上运行的实际效果是,对于有 10 000 个顶点的稀疏图,通过 16 个节点的运行加速比是 12.

关键词　MPP (message passing parallel), MST (minimum spanning tree),并行算法,通信,非关联图.

中图法分类号　TP301