Vol. 37 No. 9 Transactions of Information Processing Society of Japan Sep. 1996

Regular Paper

A Parallel Algorithm for the Longest Path Problem on Acyclic Graphs with Integer Arc Lengths

QIAN-PING GU† and TADAO TAKAOKA††

This paper presents a parallel algorithm that computes the single source longest path problem on acyclic graphs $G(V, A)$ with integer arc lengths on SIMD-SM-EREW parallel computers. The algorithm solves the problem in $O(\log^2(ln))$ time, $O((ln)^{2.376})$ processors, and $O((ln)^2)$ memory space, where $n = |V|$ and the arc lengths are integers in the set $\{1, 2, \ldots, l\}$. For any constant l, our algorithm solves the single source longest path problem in $O(\log^2 n)$ time, $O(n^{2.376})$ processors, and $O(n^2)$ memory space. Our algorithm is then used to derive $O(\log^2 n)$ time, $O(n^{2.376})$ processors, and $O(n^2)$ memory space parallel algorithms for a number of problems, such as minimum coloring, maximum clique, and so on, of permutation graphs on an SIMD-SM-EREW computer.

1. Introduction

Let $G(V, A)$ be an acyclic graph, where V and A denote the set of vertices and the set of arcs in graph G, respectively. Let $|V| = n$. The arc lengths of G are drawn from the integer set $\{1, 2, \ldots, l\}$. The single source longest path problem of graph G is to find the longest distance from a source vertex s to all other vertices in G. This problem can be solved in $O(n^2)$ time by a sequential algorithm [1]. A naive parallel algorithm for this problem is to repeatedly square the distance matrix of graph $G(V, A)$ over the semi-ring $(N \bigcup \{-\infty\}, \max, +)$, where N denotes the set of natural numbers, until the transitive closure of the distance matrix is obtained. This algorithm solves the problem in $O(\log^2 n)$ time, $O(n^3/\log n)$ procesors, and $O(n^2)$ memory space on an SIMD-SM-EREW computer (Single Instruction, Multiple Data, Shared Memory, Exclusive Read and Exclusive Write).

It is known that the matrix multiplication can be performed more efficiently over a ring than over a closed semi-ring. Applying the matrix multiplication over the integer ring, Gazit and Miller [3] gave an algorithm for the single source shortest path problem in graphs with unit length edges. The algorithm of Ref. 3) solves the above shortest path problem in $O(\log^2 n)$ time, $M(n)$ processors, and $O(n^2 \log n)$ memory space, where $M(n)$ denotes the number of processors needed to multiply two $n \times n$ integer matrices over the integer ring $(Z, +, \times)$ in $O(\log n)$ time. The best known upper bound of $M(n)$ is $O(n^{2.376})$ [2]. In this paper, following a similar approach of Ref. 3), we present a parallel algorithm for the single source longest path problem on acyclic graphs with arc lengths drawn from the integer set $\{1, 2, \ldots, l\}$. Our algorithm solves the problem in $O(\log^2(ln))$ time, $M(ln)$ processors, and $O((ln)^2)$ memory space on an SIMD-SM-EREW computer. For any constant l, our algorithm solves the single source longest path problem in $O(\log^2 n)$ time, $O(n^{2.376})$ processors, and $O(n^2)$ memory space. Our algorithm is then used to derive algorithms for several problems, such as minimum coloring, maximum clique, and so on, of permutation graphs. The derived algorithms solve those problems in $O(\log^2 n)$ time, $M(n)$ processors, and $O(n^2)$ memory space on an SIMD-SM-EREW computer. Our algorithm can also be used for the single source shortest path problem.

The rest of the paper is organized as follows. In Section 2, we give a brief review of Gazit and Miller algorithm. Our algorithm for the longest path problem is given in Section 3. Applications of our algorithm to problems in permutation graphs is given in Section 4. Finally, Section 5 concludes this paper.

2. Gazit and Miller Algorithm

Let $G(V, A)$ be a directed graph with unit arc length and $V = \{1, 2, ..., n\}$. The shortest distance from vertex u to vertex v, denoted by

† Department of Computer Software, The University of Aizu
†† Department of Computer Science, University of Ibaraki

Procedure *Find_Distances*$(M_0, ..., M_{\lceil \log n \rceil})$;
begin
  for $(1 \leq u \leq n)$ **do in parallel**
    if $(u = s)$ **then** $D[u] := 0$ **else** $D[u] := \infty$;
  for $(i := 0)$ **to** $\lceil \log n \rceil$ **do**
    for $(1 \leq u \leq n)$ **do in parallel**
      $D[u] := \min_{w \in V}\{D[u], D[w] + M_i[w, u]\}$;
end.

Fig. 1 Gazit and miller algorithm.

$d(u, v)$, is the number of edges in the shortest path from u to v, and equals infinity (∞) if no path exists from u to v. Gazit and Miller proposed an algorithm which computes the shortest distances from a given source vertex s to all the other vertices of $G(V, A)$ by applying the matrix multiplication over the integer ring [3]. The algorithm works as follows.

Let B_0 be a Boolean matrix with $B_0[u, v] = 1$ if $(u, v) \in A$ or $u = v$, $B_0[u, v] = 0$ otherwise. The algorithm first computes the Boolean matrices $B_1, ..., B_{\lceil \log n \rceil}$, where $B_{i+1} = B_i^2$ over the Boolean ring. Obviously, $B_i[u, v] = 1$ if $d(u, v) \leq 2^i$, $B_i[u, v] = 0$ otherwise.

Next, the algorithm substitutes new values into the entries of each matrix B_i to get matrices $M_0, M_1, ..., M_{\lceil \log n \rceil}$ over the semiring $(N \cup \{\infty\}, \min, +)$, where

$$M_i[u, v] = \begin{cases} 0 & \text{if } u = v \\ 2^i & \text{if } d(u, v) \leq 2^i \text{ and } u \neq v \\ \infty & \text{otherwise} \end{cases}$$

$M_i[u, v]$ gives an approximation of the shortest distance from u to v. $M_i[u, v] \geq d(u, v)$ and the least i with $M_i[u, v] \neq \infty$ implies that $2^{i-1} < d(u, v) \leq 2^i$.

Finally, the algorithm computes the shortest distances from the given source vertex s to all the other vertices by the procedure *Find_Distances* given in **Fig. 1**. The correctness of the algorithm follows from the following lemma.

Lemma 1 Let $D_i[u]$ be the value of $D[u]$ at the end of the ith iteration. If $d(s, u) < 2^{i+1}$, then $D_i[u] = d(s, u)$.

Proof: To prove the lemma, we first show that $D_i[u] \geq d(s, u)$ for every i and $u \in V$. Obviously, $D_0[u] = M_0[s, u] \geq d(s, u)$. Assume $D_{i-1}[u] \geq d(s, u)$ and we prove $D_i[u] \geq d(s, u)$. Note that $D_i[u] = \min_{w \in V}\{D_{i-1}[u], D_{i-1}[w] + M_i[w, u]\}$. If $D_{i-1}[u] \leq D_{i-1}[w] + M_i[w, u]$ for all $w \in V$ then $D_i[u] = D_{i-1}[u] \geq d(s, u)$. So, we assume $D_{i-1}[u] > D_{i-1}[w] + M_i[w, u]$ and $D_i[u] = D_{i-1}[w] + M_i[w, u]$ for some $w \in V$. From $d(s, w) + d(w, u) \geq d(s, u)$, $D_{i-1}[w] \geq d(s, w)$, and $M_i[w, u] \geq d(w, u)$, we have $D_i[u] \geq d(s, u)$.

Next, we prove that $D_i[u] \leq d(s, u)$ if $d(s, u) < 2^{i+1}$. Obviously, the claim holds for $i = 0$. Assume the claim is true for $i - 1$ and we prove it for i. If $d(s, u) < 2^i$ then from the induction hypothesis $D_{i-1}[u] \leq d(s, u)$. Therefore, $D_i[u] \leq D_{i-1}[u] \leq d(s, u)$. So, we may assume that $2^i \leq d(s, u) < 2^{i+1}$. Then, there is a vertex $w \in V$ such that $d(s, w) < 2^i$, $d(w, u) = 2^i$, and $d(s, u) = d(s, w) + d(w, u)$. From the induction hypothesis and the definition of M_i, $D_{i-1}[w] \leq d(s, w)$ and $M_i[w, u] = 2^i$. Therefore, $D_i[u] \leq D_{i-1}[w] + M_i[w, u] \leq d(s, u)$. □

The Boolean matrix $B_{i+1} = B_i^2$ can be computed over the integer ring as follows:

$$B_{i+1} = 1 \text{ iff } \sum_{k=1}^{n} B_i[u, k] \times B_i[k, v] \geq 1.$$

Let $M(n)$ be the number of processors needed to multiply two $n \times n$ integer matrices over the integer ring $(Z, +, \times)$ in $O(\log n)$ time. It takes $O(\log^2 n)$ time and $M(n)$ processors to compute $B_1, ..., B_{\lceil \log n \rceil}$. The best known upper bound of $M(n)$ is $O(n^{2.376})$ [2]. $M_0, ..., M_{\lceil \log n \rceil}$ can be computed in $O(1)$ time by at most n^2 processors. It is easy to see that Procedure *Find_Distances* can be computed in $O(\log^2 n)$ time by at most $O(n^2 / \log n)$ processors. Thus, the algorithm solves the single source shortest path problem in $O(\log^2 n)$ time and $M(n)$ processors. Note that the original Gazit and Miller algorithm [3] computes D_i from $i = \log n$ to 0 (i decreasing) which takes $O(n^2 \log n)$ memory space and makes its proof complicated. The revised version described in this section is simpler and takes $O(n^2)$ memory space.

3. Algorithm for the Longest Path Problem

In this section, let $G(V, A)$ be an acyclic graph with unit length arcs. Assume $V = \{1, 2, ..., n\}$. The single source longest path problem of G is to compute the longest distances from a source vertex s to all other vertices in G. Let $D(u, v)$ denote the longest distance from u to v, i.e., $D(u, v)$ is the number of edges in the longest path from u to v, $D(u, u) = 0$, and $D(u, v) = -\infty$ if there is no path from u to v.

Following a similar approach of the previous section, we give an algorithm for the single source longest path problem of $G(V, A)$. To

```
Procedure Longest_Path
begin
   /* C is the adjacent matrix of G(V, A). */
   for (1 ≤ u ≤ n) do in parallel
      if (s = u) then E[u] := 0 else E[u] := −∞;
   for (1 ≤ u ≤ n) do in parallel P[u, u] := 0;
   for (i := 0) to ⌈log n⌉ do
   begin
      for (1 ≤ u, v ≤ n and u ≠ v) do in parallel
         if (C[u, v] = 1) then P[u, v] := 2^i else P[u, v] := −∞;
      for (1 ≤ u ≤ n) do in parallel E[u] := max_{w∈V}{E[u], E[w] + P[w, u]};
      C := C × C;
   end;
end.
```

Fig. 2 A parallel algorithm for the single source longest path problem.

make the idea of Gazit and Miller work on the longest path problem, the key point is how to construct the approximation matrix for the longest distances. The idea here is as follows: Suppose that the longest path from u to v has the length l with $2^i \leq l < 2^{i+1}$. Then, there is vertex w in the path such that $D(u,w) < 2^i$, $D(w,v) = 2^i$, and $D(u,v) = D(u,w) + D(w,v)$. Our starting point is then to construct the matrices C_i whose (u,v)-entry is 1 if and only if there is a path from u to v of length 2^i.

Let C_0 be the adjacent matrix of graph $G(V,A)$, i.e.,

$$C_0[u,v] = \begin{cases} 1 \text{ (true)} & u \neq v \text{ and } (u,v) \in A \\ 0 \text{ (false)} & \text{otherwise} \end{cases}$$

and $C_i = C_{i-1} \times C_{i-1}$ be defined as:

$$C_i[u,v] = \vee_{w=1}^{n} C_{i-1}[u,w] \wedge C_{i-1}[w,v].$$

Then $C_i[u,v] = 1$ if and only if there is path of length 2^i from u to v which implies $2^i \leq D(u,v)$. Similarly, we substitute new values into the entries of C_i to get matrices P_i over the semiring $(N \cup \{-\infty\}, \max, +)$, where

$$P_i[u,v] = \begin{cases} 0 & \text{if } u = v \\ 2^i & \text{if } C_i[u,v] = 1 \\ -\infty & \text{otherwise} \end{cases}$$

Obviously, $P_i[u,v] \leq D(u,v)$ for every i. Let E be a row vector of length n with $E[s] = 0$ and $E[u] = -\infty$ for $u \neq s$. We finally computes the longest distances from s to u by multiplying E and P_i over the semiring $(N \cup \{-\infty\}, \max, +)$. The algorithm *Longest_Path* for the single source longest path problem of an acyclic graph with unit-length arcs is given in **Fig. 2**. In the algorithm, $-\infty + (a \in N) = -\infty$. Obviously, the matrix P obtained in the i-th iteration of the algorithm is equal to P_i. We now show the correctness of algorithm *Longest_Path*.

Theorem 2 Let $E_i[u]$ be the value of $E[u]$ at the end of the ith iteration in algorithm Longest_Path. If $D(s,u) < 2^{i+1}$, then $E_i[u] = D(s,u)$.

Proof: The proof is a dual argument of that for Lemma 1. We first show that $E_i[u] \leq D(s,u)$ for every i and $u \in V$. Obviously, $E_0[u] = P_0[s,u] \leq D(s,u)$. Assume $E_{i-1}[u] \leq D(s,u)$ and we prove $E_i[u] \leq D(s,u)$. From $E_i[u] = \max_{w \in V}\{E_{i-1}[u], E_{i-1}[w] + P_i[w,u]\}$, if $E_{i-1}[u] \geq E_{i-1}[w] + P_i[w,u]$ for all w then $E_i[u] = E_{i-1}[u] \leq D(s,u)$. So, we assume $E_{i-1}[u] < E_{i-1}[w] + P_i[w,u]$ and $E_i[u] = E_{i-1}[w] + P_i[w,u]$ for some w. From $E_{i-1}[w] \leq D(s,w)$, $P_i[w,u] \leq D(w,u)$, and $D(s,w) + D(w,u) \leq D(s,u)$, we have $E_i[u] \leq D(s,u)$.

Now we show that $E_i[u] \geq D(s,u)$ if $D(s,u) < 2^{i+1}$. Obviously, the claim holds for $i = 0$. Assume the claim is true for $i-1$ and we prove it for i. If $D(s,u) < 2^i$ then $E_i[u] \geq E_{i-1}[u] \geq D(s,u)$. So, we may assume that $2^i \leq D(s,u) < 2^{i+1}$. Then, there is a vertex $w \in V$ such that $D(s,w) < 2^i$, $D(w,u) = 2^i$, and $D(s,u) = D(s,w) + D(w,u)$. From the induction hypothesis and the definition of P_i, $E_{i-1}[w] \geq D(s,w)$ and $P_i[w,u] = 2^i$. Therefore, $E_i[u] \geq E_{i-1}[w] + P_i[w,u] \geq D(s,u)$. □

In algorithm *Longest_Path*, matrices C and P take $O(n^2)$ memory space. Matrix E is a $1 \times n$ matrix. However, we need n copies of E on an SIMD-SM-EREW computer. Therefore, the algorithm uses only $O(n^2)$ memory space. The computation time of the algorithm is clearly $O(\log n) \times t(n)$, where $t(n)$ is the time of multiplying two $n \times n$ integer matrices

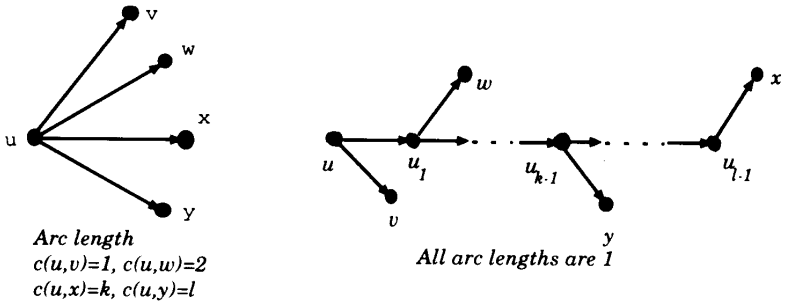

Fig. 3 Replacing arcs of length k in $G(V, A)$ with paths of length k in $G'(V', A')$.

over the integer ring of $(Z, \times, +)$. We have an upper bound of $t(n) = O(\log n)$ by $O(n^{2.376})$ processors on an SIMD-SM-EREW computer. Thus, the computation time of the algorithm is $O(\log^2 n)$ using $O(n^{2.376})$ processors and n^2 memory space. From this and Theorem 2 we have

Theorem 3 Algorithm *Longest_Path* solves the single source longest path problem of an acyclic graph with unit-length arcs in $O(\log^2 n)$ time, $O(n^{2.376})$ processors, and $O(n^2)$ memory space.

Algorithm *Longest_path* can be extended into one which solves the longest path problem on acyclic graphs with arc lengths drawn from the integer set $\{1, 2, \ldots, l\}$. Given an acyclic graph $G(V, A)$ with arc lengths drawn from $\{1, 2, \ldots, l\}$, we transform $G(V, A)$ into a new acyclic graph $G'(V', A')$ with unit-length arcs which preserves the distance between every pair of vertices in $G(V, A)$. The vertex set V' of G' is constructed as:

$V' := V$.

For $u \in V$, $V' := V \cup \{u_1, u_2, \ldots, u_{l-1}\}$.

The arc set A' of G' is constructed as:

Step 1, $A' := \emptyset$. For $u \in V$, $A' := A' \bigcup \{(u, u_1), (u_1, u_2,), \ldots, (u_{l-2}, u_{l-1})\}$.

Step 2, For $u \in V$ and $(u, v) \in A$, if $d(u, v) = k$ then $A' := A' \bigcup \{(u_{k-1}, v)\}$, where $u_0 = u$. Obviously, $|V'| = ln$. **Figure 3** gives an example of the transform. Applying algorithm *Longest_Path* to graph $G(V', A')$, we have

Theorem 4 The single source longest path problem on an acyclic graph with arc lengths drawn from the integer set $\{1, 2, \ldots, l\}$ can be solved in $O(\log^2(ln))$ time, $O((ln)^{2.376})$ processors, and $((ln)^2)$ memory space.

4. Applications to Permutation Graphs

An undirected graph $G(V, E)$ with $V = \{1, 2, \ldots, n\}$ is called a permutation graph if there exists a permutation P on set V such that

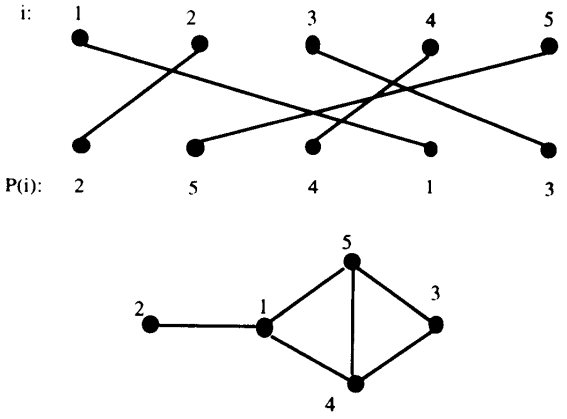

Fig. 4 A permutation graph and its matching diagram.

for any $i, j \in V$,

$$(i - j)(P^{-1}(i) - P^{-1}(j)) < 0$$

if and only if $(i, j) \in E$. Pictorially, draw the vertices $1, 2, \ldots, n$ in order on a line, and $P(1), P(2), \ldots, P(n)$ on a parallel line in such a way that i is directly above $P(i)$ for all $i \in V$. Next, for each $i \in V$, draw a line segment from i on the upper line to i on the lower line. There is an edge $(i, j) \in E$ if and only if the line segment for i intersects the line segment for j. An example of permutation graph (with $V = \{1, 2, 3, 4, 5\}$ and $P = (2, 5, 4, 1, 3)$) and its "matching diagram" is given in **Fig. 4.**

Permutation graphs have numerous applications in solving intersection-free layout problem, optimal schedules for memory reallocation, modeling, and so on[4),5)]. In this section, we consider minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set problems on permutation graphs. Many sequential algorithms have been developed for those problems. Supowit[6)] proposed an $O(n \log n)$ algorithm for the above problems. The idea in the algorithm of Ref. 6) is transforming the problems of permutation graphs into the longest chain problem on a two dimen-

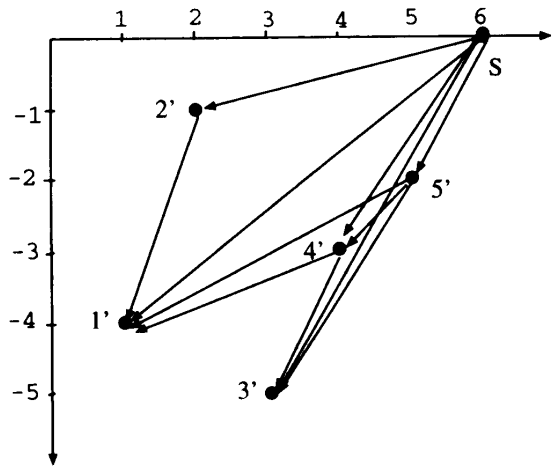

Fig. 5 The acyclic graph G' for the graph of Fig. 2.

sional real space R^2. Let $\leq$ be the partial ordering on R^2 defined by $(x_1, y_1) \leq (x_2, y_2)$ if and only if $x_1 \leq x_2$ and $y_1 \leq y_2$. A chain is a set of points in R^2 that are pairwise comparable under this ordering. For a permutation graph $G(V, E)$ and the corresponding permutation P, if the vertex $i \in V$ is mapped to R^2 by the mapping function $f(i) = (i, P^{-1}(i))$, then it is easy to show that the minimum coloring of G can be solved by partitioning $\{f(1), f(2), \ldots, f(n)\}$ into a minimum number of chains. Based on the idea of Ref. 6), we define a new mapping function from V to R^2. Then the minimum coloring problem, maximum clique problem, and so on, of a permutation graph can be solved by finding the single source longest path problem of a graph consisting of mapped vertices in R^2.

Let $G(V, E)$ be a permutation graph and P be a corresponding permutation. Let the mapping function for $i \in V$ to R^2 be defined by $f(i) = (i, -P^{-1}(i))$. Let the point $f(i)$ in R^2 be denoted by i'. For the graph G, let $G'(V', A)$ be a directed graph defined as:

$$V' = \{i' | i \in V\} \bigcup \{s\}$$

and

$$A = \{(i', j') | i', j' \in V' \text{ and } i' \leq j'\},$$

where $s = (n+1, 0)$ and $\leq$ is the partial ordering on R^2 defined by $(x_1, y_1) \leq (x_2, y_2)$ if and only if $x_1 \leq x_2$ and $y_1 \leq y_2$. Clearly, G' is an acyclic graph. The distance associated to the arcs in G' is one. **Figure 5** shows the graph G' for the graph G in Fig. 4.

We now show that the minimum coloring problem of G is equivalent to the single source (with source s) longest distance problem of

Procedure *Minimum_Coloring*()
begin

/* Construct Graph G' */
$V' := \emptyset$; $A := \emptyset$;
for $1 \leq i \leq n$ **do in parallel** $V' := V' \bigcup \{i'\}$;
$s := (n+1, 0)$; $V' := V' \bigcup \{s\}$;
for $1 \leq i, j \leq n$ **do in parallel**
  if $i' \leq j'$ **then** $A := A \bigcup \{(i', j')\}$;
for $1 \leq i \leq n$ **do in parallel** $A := A \bigcup \{(s, i')\}$;

Call *Logest_Path*() for Graph G' with source s;
end.

Fig. 6 A parallel algorithm for the minimum coloring problem of a permutation graph.

G'. By the definition of a permutation graph, $(i - j)(P^{-1}(i) - P^{-1}(j)) < 0$ if and only if $(i, j) \in E$. Therefore, $(i', j') \in A$ or $(j', i') \in A$ if and only if $(i, j) \in E$. From this, the longest distances from vertex s to vertices i' and j' are the same, i.e., $d(s, i') = d(s, j')$ if and only if $(i, j) \notin E$. If there is no arc from i' to j' or j' to i' in G' then we can give the same color to i and j in G. Thus, i and j can be given the same color if $d(s, i') = d(s, j')$. So it is easy to see that $\max\{d(s, 1'), d(s, 2'), \ldots, d(s, n')\}$ is the minimum number of colors for graph G. This number is also the size of the maximum clique of G. We now give our parallel algorithm for the minimum coloring problem of G in **Fig. 6**.

Theorem 5 The minimum coloring problem of a permutation graph can be solved by algorithm *Minimum_Coloring*() in $O(\log^2 n)$ time, $O(n^{2.376})$ processors, and $O(n^2)$ memory space.

Similar results can be obtained for other problems mentioned above for permutation graphs.

5. Conclusion

This paper gave a parallel algorithm for the single source longest path problem on acyclic graphs with integer arc lengths. For an acyclic graph with arc lengths drawn from the integer set $\{1, 2, \ldots, l\}$, the algorithm solves the problem in $O(\log^2(ln))$ time, $M(ln)$ processors, and $O((ln)^2)$ memory space on an SIMD-SM-EREW computer, where $M(n)$ is the number of processors needed to multiply two $n \times n$ integer matrices in $O(\log n)$ time. The best known upper bound of $M(n)$ is $O(n^{2.376})$ [2]. For any constant l, the algorithm solves the longest path problem on acyclic graphs with integer arc lengths in $O(\log^2 n)$ time, $O(n^{2.376})$ proce-

sors, and $O(n^2)$ memory space. If Strassen's method [1], which is more practical, is used then the algorithm can solve the above problems in $O(\log^2(ln))$ time, $O((ln)^{2.81})$ processors, and $O((ln)^2)$ memory space. The algorithm in this paper is then used to solve several problems, such as minimum coloring, maximum clique, and so on, of permutation graphs in $O(\log^2 n)$ time, $M(n)$ processors, and $O(n^2)$ memory space on an SIMD-SM-EREW computer.

Acknowledgement

The authors would like to thank the reviewers for the constructive comments.

References

1) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).
2) Coppersmith, D. and Winograd, S.: Matrix Multiplication via Arithmetic Progressions, *Proc. 19th ACM Symposium on Theory of Computing*, pp.1–6 (1987).
3) Gazit, H. and Miller, G.: An Improved Parallel Algorithm that Computes the bfs Numbering of a Directed Graph, *Information Processing Letters*, Vol.28, pp.61–65 (1988).
4) Knuth, D.E.: *The Art of Computer Programming*, Vol.1, Addison-Wesley, Reading, MA (1968).
5) Liu, C.L.: *Introduction to Combinatorial Mathematics*, McGraw-Hill (1968).
6) Supowit, K.J.: Decomposing a Set of Points into Chains, with Applications to Permutation Graphs, *Information Processing Letters*, Vol.21, pp.249–252 (1985).

(Received November 30, 1995)
(Accepted June 6, 1996)

Qian-Ping Gu received his B.S., M.S. and Ph.D. degrees, all in computer science, from Shandong University, China, Ibaraki University, Japan, and Tohoku University, Japan, in 1982, 1985, and 1988, respectively. He is currently Associate Professor in the Department of Computer Software, the University of Aizu, Japan. He was with the Institute of Software, Chinese Academy of Sciences, Beijing China. He visited Ibaraki University, Japan from 1990 to 1991, and the Department of Electrical and Computer Engineering, the University of Calgary, Canada, from 1991 to 1993. His research interets include algorithms, computational complexity, machine learning, parallel processing, and optimization. He is a member of ACM, IEEE Computer Society, and IEICE of Japan.

Tadao Takaoka was born in 1943. He received his M.S. and Ph.D. degrees from Kyoto University in 1968 and 1971. He worked in the NTT Laboratory since 1971 and was engaged in theoretical research. Since 1974 he has been at Ibaraki University as an associate professor and later a professor. His current research interests include analysis of algorithms and program verification. He taught at the University of Canterbury and the University of Alabama at Birmingham. He is a member of IPSJ, IEICE, IEEE and ACM.