Parallelizability of some P-complete problems*

Akihiro Fujiwara[1], Michiko Inoue[2], and Toshimitsu Masuzawa[2]

[1] Kyushu Institute of Technology, JAPAN
fujiwara@cse.kyutech.ac.jp
[2] Nara Institute of Science and Technology, JAPAN
{kounoe, masuzawa}@is.aist-nara.ac.jp

Abstract. In this paper, we consider parallelizability of some P-complete problems. First we propose a parameter which indicates parallelizability for a convex layers problem. We prove P-completeness of the problem and propose a cost optimal parallel algorithm, according to the parameter. Second we consider a lexicographically first maximal 3 sums problem. We prove P-completeness of the problem by reducing a lexicographically first maximal independent set problem, and propose two cost optimal parallel algorithms for related problems. The above results show that some P-complete problems have efficient cost optimal parallel algorithms.

1 Introduction

In parallel computation theory, one of primary complexity classes is the class NC. Let n be the input size of a problem. The problem is in the class NC if there exists an algorithm which solves the problem in $T(n)$ time using $P(n)$ processors where $T(n)$ and $P(n)$ are polylogarithmic and polynomial functions for n, respectively. Many problems in the class P, which is the class of problems solvable in polynomial time sequentially, are also in the class NC. On the other hand, some problems in P seem to have no parallel algorithm which runs in polylogarithmic time using a polynomial number of processors. Such problems are called *P-complete*. A problem in the class P is P-complete if we can reduce any problem in P to the problem using NC-reduction. (For details of the P-completeness, see [9].) Although there are some efficient *probabilistic* parallel algorithms for some P-complete problems, it is believed that the P-complete problems are inherently sequential and hard to be parallelized.

Among many P-complete problems, only some graph problems are known to be asympotically parallelizable. Vitter and Simons[12] showed that the unification, path system accessibility, monotone circuit value and ordered depth-first search problems have cost optimal parallel algorithms if their input graphs are dense graphs, that is, the number of edges is $m = \Omega(n^{1+\epsilon})$ for a constant ϵ where the number of vertices is n.

* Research supported in part by the Scientific Research Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan (Scientific research of Priority Areas(B)10205218)

In this paper, we consider parallelizability of two P-complete problems. First we consider *a convex layers problem*. For the problem, we propose a parameter d which indicates parallelizability of the problem. Using the parameter, we prove that the problem is still P-complete if $d = n^\epsilon$ with $0 < \epsilon < 1$. Next we propose a parallel algorithm which runs in $O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$ time using p processors $(1 \leq p \leq d)$ on the EREW PRAM. From the complexity, the problem is in NC if $d = (\log n)^k$ where k is a positive constant, and has a cost optimal parallel algorithm if $d = n^\epsilon$ with $0 < \epsilon \leq \frac{1}{2}$.

Second P-complete problem is *a lexicographically first maximal 3 sums problem*. We prove the P-completeness of the problem, and propose a parallel algorithm, which runs in $O(\frac{n^2}{p} + n \log n)$ using p processors $(1 \leq p \leq n)$ on the CREW PRAM, for the problem. The above algorithm is cost optimal for $1 \leq p \leq \frac{n}{\log n}$. In addition, we propose a cost optimal parallel algorithm for a related P-complete problem. These results show that some P-complete problems have efficient cost optimal parallel algorithms.

2 Parameterized convex layers

First we give some definitions for convex layers.

Definition 1 (Convex layers). *Let S be a set of n points in the Euclidean plane. The convex layers is a problem to compute a set of convex hulls, $\{CH_0, CH_1, \ldots, CH_{m-1}\}$, which satisfies the following two conditions.*

(1) $CH_0 \cup CH_1 \cup \ldots \cup CH_{m-1} = S$.
(2) *Each CH_i $(0 \leq i \leq m-1)$ is a convex hull of a set of points $CH_i \cup CH_{i+1} \cup \ldots \cup CH_{m-1}$.* □

Dessmark et al.[5] proved P-completeness of the convex layers problem, and Chazelle[1] proposed an optimal sequential algorithm which runs in $O(n \log n)$ time. The sequential algorithm is time optimal because computation of a convex hull, which is the first hull of convex layers, requires $\Omega(n \log n)$ time[13].

In this paper, we consider an additional parameter d for the problem, and restrict its input points on d horizontal lines.

Definition 2 (Convex layers for d lines). *The convex layers for d lines is a convex layers problem whose input points are on d horizontal lines.* □

The parameter d is at most n if there is no restrictions for positions of input points. In the following, $CL(d)$ denotes the convex layers for d lines problem. We can solve the problem sequentially in $O(n \log n)$ time using the algorithm[1], and prove the lower bound $\Omega(n \log n)$ by reduction from the sorting.

We can prove the following theorem for the problem $CL(d)$. (We omit the proof because of space limitation. The proof is described in [7].)

Theorem 1. *The problem $CL(n^\epsilon)$ with $0 < \epsilon \leq 1$ is P-complete.* □

Next we propose a cost optimal parallel algorithm for $CL(d)$.

Algorithm for computing $CL(d)$

Input: A set of points $\{u_0, u_1, \ldots, u_{n-1}\}$ on lines $\{l_0, l_1, \ldots, l_{d-1}\}$.

Step 1: Set variables $TOP = 0$ and $BOT = d - 1$. (l_{TOP} and l_{BOT} denote top and bottom lines respectively.) Compute a set of points on each line l_i ($0 \leq i \leq d-1$), and store them in a double-ended queue Q_i in order of x coordinates.

Step 2: For each line l_i ($TOP \leq i \leq BOT$), compute the leftmost point u^i_{left} and the rightmost point u^i_{right}.

Step 3: Let U_{left} and U_{right} denote sets of points $\{u^{TOP}_{left}, u^{TOP+1}_{left}, \ldots, u^{BOT}_{left}\}$ and $\{u^{TOP}_{right}, u^{TOP+1}_{right}, \ldots, u^{BOT}_{right}\}$ respectively. Compute a left hull of U_{left} and a right hull of U_{right}, and store the obtained points on each hull in CH_{left} and CH_{right}, respectively. (The left hull of U_{left} consists of points on a convex hull of U_{left}, which are from u^{BOT}_{left} to u^{TOP}_{left} in clockwise order. The right hull of U_{right} is defined similarly.)

Step 4: Remove points in Q_{TOP}, Q_{BOT}, CH_{left} and CH_{right} as the outmost convex hull.

Step 5 Compute top and bottom lines on which there is at least one point. Set TOP and BOT to obtained top and bottom lines respectively.

Step 6: Repeat Step 2, 3, 4 and 5 until no point remains.

We discuss complexities of the above parallel algorithm on the EREW PRAM. We use at most p processor ($1 \leq p \leq d$) in the algorithm except for Step 1. Step 1 takes $O(\frac{n \log n}{p} + \log n)$ using Cole's merge sort[4] and primitive operations, and Step 2 takes $O(\frac{d}{p})$ time obviously. We can compute the left hull and the right hull in Step 3 using a known parallel algorithm[2, 3] for computing a convex hull of sorted points. The algorithm runs in $O(\frac{d}{p} + \log d)$ time for each hull. Step 4 takes $O(\frac{d}{p})$ time to remove the points. (Points in Q_{TOP}, Q_{BOT} are automatically removed by changing TOP and BOT in Step 5.) We can compute top and bottom lines in Step 5 in $O(\frac{d}{p} + \log d)$ time using a basic parallel algorithm computing the maximum and the minimum. Since the number of the repetition of Step 6 is $\lceil \frac{d}{2} \rceil$, we can compute $CL(d)$ in $O(\frac{n \log n}{p} + \log n + (\frac{d}{p} + \log d) \times \lceil \frac{d}{2} \rceil) = O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$, and obtain the following theorem.

Theorem 2. *We can solve $CL(d)$ in $O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$ time using p processors ($1 \leq p \leq d$) on the EREW PRAM.* □

We can show that the class of the problem changes according to the number of lines d from the above complexity. (Details are omitted.)

Corollary 1. *We can solve $CL((\log n)^k)$, where k is a positive constant, in $O(\log n \log \log n)$ time using n processors on the EREW PRAM, that is, $CL((\log n)^k)$ is in NC.* □

Corollary 2. *We can solve $CL(n^\epsilon)$ with $0 < \epsilon \leq \frac{1}{2}$ in $O(\frac{n \log n}{p})$ time using p processors ($1 \leq p \leq n^\epsilon$) on the EREW PRAM.* □

3 Lexicographically first maximal 3 sums

We first define the lexicographically first maximal 3 sums problem as follows.

Definition 3 (Lexicographically first maximal 3 sums). *Let I be a set of n distinct integers. The lexicographically first maximal 3 sums is a problem to compute the set of 3 integers $LFM3S = \{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_{m-1}, b_{m-1}, c_{m-1})\}$, which satisfies the following three conditions.*

1. *The set $S = \{a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_{m-1}, b_{m-1}, c_{m-1}\}$ is a subset of I.*
2. *Let $s_i = \{a_i, b_i, c_i\}$ $(0 \leq i \leq m-1)$. Then, (a_i, b_i, c_i) is the lexicographically first set of 3 integers which satisfies $a_i + b_i + c_i = 0$ for $I - (s_0 \cup s_1 \cup \ldots \cup s_{i-1})$.*
3. *There is no set of three integers (a', b', c') which satisfies $a', b', c' \in I - S$ and $a' + b' + c' = 0$.* □

Next we prove P-completeness of LFM3S. We show reduction from *the lexicographically first maximal independent set (LFMIS) problem* to LFM3S. Let $G = (V, E)$ be an input graph for LFMIS. We assume that all vertices in $V = \{v_0, v_1, \ldots, v_{n-1}\}$ are ordered, that is, v_i is less than v_j if $i < j$.

In [11], Miyano proved the following lemma for LFMIS.

Lemma 1. *The LFMIS restricted to graphs with degree at most 3 is P-complete.* □

Using the above lemma, we can prove the P-completeness of LFM3S. (Details are described in [7].)

Theorem 3. *The problem LFM3S is P-complete.*

(Outline of proof)

It is obvious that LFM3S is in P.

Let $G = (V, E)$ with $V = \{v_0, v_1, \ldots, v_{n-1}\}$ be an input graph with degree at most 3. First we define *a vertex value* $VV(i)$ for each vertex v_i. The vertex value is a negative integer and defined as $VV(i) = i - n$. Thus vertices $v_0, v_1, \ldots, v_{n-1}$ have vertex values $-n, -(n-1), \ldots, -1$ respectively. We also difine a key set of integers $Q = \{q_0, q_1, \ldots, q_{12}\} = \{-64, -61, -32, -31, -29, -15, -14, -13, -10, -8, 23, 46, 93\}$. Using the vertex value and the key set, we define the following 4-tuples for each vertex v_i in V $(0 \leq i \leq n-1)$ as inputs for LFM3S.

1. **Vertex tuple for** v_i: $VT(i) = [VV(i), q_0, VV(i), 0]$
2. **Auxiliary tuples for** v_i:
   (a) $AT_1(i) = [VV(i), q_1, 0, VV(i)]$ (b) $AT_2(i) = [VV(i), q_2, VV(i), 0]$
   (c) $AT_3(i) = [VV(i), q_3, 0, VV(i)]$ (d) $AT_4(i) = [VV(i), q_4, 0, VV(i)]$
   (e) $AT_5(i) = [VV(i), q_5, VV(i), 0]$ (f) $AT_6(i) = [VV(i), q_6, 0, VV(i)]$
   (g) $AT_7(i) = [VV(i), q_7, 0, VV(i)]$ (h) $AT_8(i) = [VV(i), q_8, VV(i), 0]$
   (i) $AT_9(i) = [VV(i), q_9, 0, VV(i)]$
   (j) $AT_{10}(i) = [2 * |VV(i)|, q_{10}, |VV(i)|, |VV(i)|]$
   (k) $AT_{11}(i) = [2 * |VV(i)|, q_{11}, |VV(i)|, |VV(i)|]$
   (l) $AT_{12}(i) = [2 * |VV(i)|, q_{12}, |VV(i)|, |VV(i)|]$

3. **Link tuples for** v_i: For each adjacent vertex v_j of v_i, which satisfies $i < j$, add one of the following tuples.
   (a) $LT_1(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_1|, |VV(j)|, |VV(i)|]$
   (b) $LT_2(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_3|, |VV(j)|, |VV(i)|]$
   (c) $LT_3(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_7|, |VV(j)|, |VV(i)|]$
   (In case that v_i has only one adjacent vertex v_j which satisfies $i < j$, add $LT_1(i,j)$ for v_j. In case that v_i has the two adjacent vertices v_{j_1}, v_{j_2}, add $LT_1(i,j_1)$ and $LT_2(i,j_2)$ for each vertex. In case that v_i has the three adjacent vertices, add all three tuples similarly.)

The above 4-tuples have the following special feature. Let $\{VT(i), AT_1(i), AT_2(i), \ldots, AT_{12}(i), LT_1(i,s), LT_2(i,t), LT_3(i,u), VT(s), VT(t), VT(u)\}$ be the input for LFM3S[1]. (We assume v_s, v_t and v_u are adjacent vertices which satisfy $i < s < t < u$.) Then the solution of LFM3S is as follows. (We call the solution *TYPE A sums*.)

$$\{(VT(i), AT_4(i), AT_{12}(i)),\ (AT_2(i), AT_6(i), AT_{11}(i)),\ (AT_5(i), AT_9(i), AT_{10}(i)),$$
$$(AT_1(i), VT(s), LT_1(i,s)),\ (AT_3(i), VT(t), LT_2(i,t)),\ (AT_7(i), VT(u), LT_3(i,u))\}$$

Note that vertex tuples, $VT(s)$, $VT(t)$ and $VT(u)$, are in the sums. In other words, the above vertex tuples are not in the remaining inputs after the computation.

Next, we consider the solution without $VT(i)$ in the input. (We call the solution *TYPE B sums*.)

$$\{(AT_1(i), AT_2(i), AT_{12}(i)), (AT_3(i), AT_5(i), AT_{11}(i)), (AT_7(i), AT_8(i), AT_{10}(i))\}$$

In this case, the vertex tuples, $VT(s)$, $VT(t)$ and $VT(u)$, remain in the inputs.

We give the above 4-tuples for all vertices in V of LFMIS, and compute LFM3S. Then the vertex $v_i \in V$ is in the solution of LFMIS if and only if there exists a sum of three 4-tuples (T_1, T_2, T_3) which satisfies $T_1 = VT(i)$ in the solution of LFM3S. (Proof of correctness is omitted.)

It is easy to see that the above reduction is in NC. Although we define that inputs of $LFM3S$ are distinct integers, inputs of the above reduction are 4-tuples. We can easily reduce each 4-tuple to an integer without loss of the features. Let $2^g \leq n < 2^{g+1}$ and $h = \max\{g, 6\}$. Then we can reduce each 4-tuple $[\alpha_0, \alpha_1, \alpha_2, \alpha_3]$ to $\alpha_0 * 2^{3(h+1)} + (\alpha_1 - 65) * 2^{2(h+1)} + \alpha_2 * 2^{h+1} + \alpha_3$. □

Finally, we consider a parallel algorithm for LFM3S on the CREW PRAM. We can propose a sequential algorithm which solves LFM3S in $O(n^2)$ by modifying an algorithm computing the 3 sum problem[8]. The algorithm is the known fastest sequential algorithm for LFM3S. Note that strict lower bound of LFM3S is not known. However the 3 sum has no $o(n^2)$ algorithm and has an $\Omega(n^2)$ lower bound on a weak model of computation[6].

Algorithm for computing $LFM3S$
Input: A set of n integers I.

[1] The sum of tuples $A = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]$ and $B = [\beta_0, \beta_1, \beta_2, \beta_3]$ is defined as $A + B = [\alpha_0 + \beta_0, \alpha_1 + \beta_1, \alpha_2 + \beta_2, \alpha_3 + \beta_3]$, and $A < B$ if A is lexicographically less than B. We assume that the sum is zero if the sum of tuples is $[0, 0, 0, 0]$.

Step 1: Sort all elements in I. (Let $S = (s_0, s_1, \ldots, s_{n-1})$ be the sorted sequence.)
Step 2: Repeat the following substeps from $i = 0$ to $i = n - 3$.
(2-1) Create the following two sorted sequences S' and S'_R from S.

$$S' = (s_{i+1}, s_{i+2}, \ldots, s_{n-1}), S'_R = (-s_{n-1} - s_i, -s_{n-2} - s_i, \ldots, -s_{i+1} - s_i)$$

(For $b \in S'$ and $c \in S'_R$ which satisfy $b = s_g$ and $c = -s_h - s_i$ respectively, $b = c$ if and only if $s_i + s_g + s_h = 0$.)
(2-2) Merge S' and S'_R into a sorted sequence $SS = (ss_0, ss_1, \ldots, ss_{2(n-i-1)-1})$.
(2-3) Compute the smallest element ss_j in SS which satisfies $ss_j = ss_{j+1}$.
(2-4) If the above ss_j is obtained, compute s_g and s_h in S such that $s_g = ss_j$ and $s_h = -s_g - s_i$, respectively. (It is obvious that $s_g \in S'$ and $-s_g - s_i \in S'_R$ since all elements in S are distinct.) Delete s_i, s_g, s_h from I, and output (s_i, s_g, s_h), whenever they exist.

We assume the number of processors p is restricted to $1 \leq p \leq n$. We can sort n elements in $O(\frac{n \log n}{p} + \log n)$ time using Cole's merge sort[4] in Step 1. In Step 2, we can compute a substep (2-1) in $O(\frac{n}{p})$ time easily. We can compute substeps (2-3) and (2-4) in $O(\frac{n^2}{p} + \log n)$ time using simple known algorithms and basic operations. In a substep (2-2), we can merge two sorted sequence in $O(\frac{n}{p} + \log \log n)$ time using a fast merging algorithm[10]. Since repetition of Step 2 is $O(n)$, we obtain the following theorem.

Theorem 4. *We can solve LFM3S in $O(\frac{n^2}{p} + n \log n)$ time using p processors $(1 \leq p \leq n)$ on the CREW PRAM.* □

In the case of $1 \leq p \leq \frac{n}{\log n}$, the time complexity becomes $O(\frac{n^2}{p})$. Therefore the above algorithm is cost optimal for $1 \leq p \leq \frac{n}{\log n}$.

As generalization of LFM3S, we can also obtain the similar results for the following problem.

Definition 4 (Lexicographically first maximal set of 3 arguments (LFMS3A)). *Let E be a totally ordered set of n elements. The lexicographically first maximal set of 3 arguments is a problem to compute the set of 3 elements $LFMS3A = \{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_m, b_m, c_m)\}$, which satisfies the following three conditions for a given function $f(x, y, z)$ whose value is $TRUE$ or $FALSE$.*

1. *The set $S = \{a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_m, b_m, c_m\}$ is a subset of E.*
2. *Let $e_i = \{a_i, b_i, c_i\}$ $(0 \leq i \leq m)$. Then, (a_i, b_i, c_i) is the lexicographically first set of 3 elements which satisfies $f(a_i, b_i, c_i) = TRUE$ for $I - (e_0 \cup e_1 \cup \ldots \cup e_{i-1})$.*
3. *There is no set of three elements (a', b', c') which satisfies $a', b', c' \in I - S$ and $f(a', b', c') = TRUE$.* □

Corollary 3. *The problem LFMS3A is P-complete.* □

Theorem 5. *We can solve $LFMS3A$ with an unresolvable function f in $O(\frac{n^3}{p} + n \log n)$ time using p processors $(1 \leq p \leq n^2)$ on the CREW PRAM.* □

4 Conclusions

In this paper, we proved that two problems are P-complete, and proposed cost optimal algorithms for the problems. The results imply that some P-complete problems are parallelizable within the reasonable number of processors.

In the future research, we investigate other parallelizable P-complete problems. The result may imply new classification of problems in P. Another future topic is proposition of fast parallel algorithms which run in $O(n^\epsilon)$ time where $0 < \epsilon < k$ for P-complete problems. Only a few P-complete problems are known to have such algorithms[12].

References

1. B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, IT-31(4):509–517, 1985.
2. D. Z. Chen. Efficient geometric algorithms on the EREW PRAM. *IEEE transactions on parallel and distributed systems*, 6(1):41–47, 1995.
3. W. Chen. *Parallel Algorithm and Data Structures for Geometric Problems*. PhD thesis, Osaka University, 1993.
4. R. Cole. Parallel merge sort. *SIAM Journal of Computing*, 17(4):770–785, 1988.
5. A. Dessmark, A. Lingas, and A. Maheshwari. Multi-list ranking: complexity and applications. In *10th Annual Symposium on Theoretical Aspects of Computer Science (LNCS665)*, pages 306–316, 1993.
6. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. In *34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, pages 528–536, 1993.
7. A. Fujiwara, M. Inoue, and M. Toshimitsu. Practical parallelizability of some P-complete problems. Technical Report of IPSF, Vol. 99, No. 72 (AL-69-2), September 1999.
8. A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry*, 5:165–185, 1995.
9. R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford university press, 1995.
10. C. Kruskal. Searching, merging and sorting in parallel computation. *IEEE Transactions on Computers*, C-32(10):942–946, 1983.
11. S. Miyano. The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. *Mathematical Systems Theory*, 22:47–73, 1989.
12. J.S. Vitter and R.A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Transactions of Computers*, C-35(5):403–418, 1986.
13. A. C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28(4):780–787, 1981.