Parallel PROFIT/COST Algorithms Through Fast Derandomization[1]

*Yijie Han** and *Yoshihide Igarashi***

*Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA

**Department of Computer Science
Gunma University
Kiryu, 376 Japan

Summary

We present parallel algorithms for the PROFIT/COST problem with time complexity $O(\log n)$ using $O(m+n)$ processors. The design of these algorithms employ both the derandomization technique and the pipeline technique. They can be used to partition the vertices of a graph into two sets such that the number of edges incident with vertices in both sets is at least half of the total number of edges in the graph. Parallel algorithms for the PROFIT/COST problem have known applications in the design of parallel algorithms for several graph problems.

Keywords: Design of algorithms, parallel algorithms, graph algorithms, data structures, derandomization.

1 Introduction

The PROFIT/COST problems as formulated by Luby[16] can be described as follows.

Let $\overrightarrow{x} = < x_i \in \{0,1\} : i = 0, ..., n-1 >$. Each point $\overrightarrow{x}$ out of the 2^n points is assigned probability $1/2^n$. Given function $B(\overrightarrow{x}) = \sum_{i,j} f_{i,j}(x_i, x_j)$, where $f_{i,j}$ is defined as a function $\{0,1\}^2 \rightarrow \mathcal{R}$. The

[1]Preliminary version of the results in this paper was presented in [10][11].

PROFIT/COST problem is to find a good point $\vec{y}$ such that $B(\vec{y}) \geq E[B(\vec{x})]$. B is called the BENEFIT function and $f_{i,j}$'s are called the PROFIT/COST functions.

The size m of the problem is the number of nontrivial PROFIT/COST functions present in the input. The input is dense if $m = \theta(n^2)$ and is sparse if $m = o(n^2)$.

The vertex partition problem is a basic problem which can be modeled by the PROFIT/COST problem[16]. The vertex partition problem is to partition the vertices of a graph into two sets such that the number of edges incident with vertices in both sets is at least half of the number of edges in the graph. Let $G = (V, E)$ be the input graph. $|V|$ 0/1-valued uniformly distributed mutually independent random variables are used, one for each vertex. The problem of partitioning vertices into two sets is now represented by the 0/1 labeling of the vertices. Let x_i be the random variable associated with vertex i. For each edge $(i, j) \in E$ a function $f(x_i, x_j) = x_i \oplus x_j$ is defined, where $\oplus$ is the exclusive-or operation. $f(x_i, x_j)$ is 1 iff edge (i, j) is incident with vertices in both sets. The expectation of f is $E[f(x_i, x_j)] = (f(0, 0)+f(0, 1)+f(1, 0)+f(1, 1))/4 = 1/2$. Thus the BENEFIT function $B(x_0, x_1, ..., x_{|V|-1}) = \sum_{(i,j)\in E} f(x_i, x_j)$ has expectation $E[B] = \sum_{(i,j)\in E} E[f(x_i, x_j)] = |E|/2$. If we find a good point p in the sample space such that $B(p) \geq E[B] = |E|/2$, this point p determines the partition of vertices such that the number of edges incident with vertices in both sets is at least $|E|/2$.

The PROFIT/COST problem is a basic problem in the study of derandomization, *i.e.*, converting a randomized algorithm to a deterministic algorithm. The importance of the PROFIT/COST problem lies in the fact that it can be used as a basic subroutine in the derandomization of more complicated randomized algorithms[5][9][16][19].

Luby[16] gave a parallel algorithm for the PROFIT/COST problem with time complexity $O(\log^2 n)$ using $O(m + n)$ processors on the EREW(Exclusive Read Exclusive Write) PRAM[4][8]. He used a sample space with $O(n)$ sample points and designed $O(n)$ uniformly distributed pairwise independent random variables on the sample space. His algorithm was obtained through a derandomization process in which a good sample point is found by a binary search of the sample space.

We show that a derandomization scheme faster than Luby's [16] can be obtained. One of our observations leading to the faster derandomization scheme is to exploit the redundancy which is a consequence of a shrinking sample space. Initially a minimum sized sample space is chosen for the design of random variables of limited independence. When a binary search technique is used to search the sample space for a good sample point, the sample space is shrunken or reduced. In fact we observe the shrinkage of the sample space when conditional probability is considered. However, when the sample space is shrunken, the original assumption of independence among random variables can no longer hold. That is, dependency among random variables is expected. Such a dependency is a form of redundancy which can be exploited to the advantage of parallel algorithm design. We note that such redundancy has not been exploited before in the previous derandomization schemes [2][13][15][16].

Another idea used in our design is to exploit mutual independence. Previous derandomization schemes for parallel algorithms tried to stay away from mutual independence because a sample space containing n mutually independent random variables has an exponential number of sample points [2]. We show, on the contrary, that mutual independence can be exploited for the design of fast parallel algorithms. Our design provides a fast algorithm for the PROFIT/COST problem with time complexity $O(\log n)$ using no more than $O(m + n)$ processors.

Our first algorithm, which is a CREW(Concurrent Read Exclusive Write) algorithm, is designed through derandomization of the random variables by exploiting the redundancy and mutual independence of these random variables. The efficiency of this CREW algorithm is improved by using derandomization trees and tree contraction techniques. We then design an EREW algorithm for the PROFIT/COST problem. This EREW algorithm is obtained by using a pipeline to remove the concurrent read feature from the CREW algorithm and by building row and column trees to help disseminating the bit setting information. The pipeline used has inevitably made our EREW algorithm rather complicated. Since EREW PRAM is weaker than the CREW PRAM, the introduced complication is worthwhile for removing the concurrent read feature. To facilitate presentation we will first present an EREW algorithm for the dense case and then convert it to the general case by using row and column trees.

2 Exploiting Redundancy

We consider the scenario of 0/1-valued uniformly distributed pairwise independent random variables in the following setting.

A set of n 0/1-valued uniformly distributed pairwise independent random variables can be designed on a sample space with $O(n)$ points. The following design is given in [2][5][16]. Let $k = \lceil \log n \rceil$. The sample space is $\Omega = \{0, 1\}^{k+1}$. For each $a = a_0 a_1 ... a_k \in \Omega$, $Pr(a) = 2^{-(k+1)}$.

The value of random variables x_i, $0 \le i < n$, on point a is $x_i(a) = (\sum_{j=0}^{k-1}(i_j \cdot a_j) + a_k) \bmod 2$, where i_j is the j-th bit in the binary expansion of i.

Typical BENEFIT functions to be searched on have the form $B(x_0, x_1, ..., x_{n-1}) = \sum_{i,j} f_{i,j}(x_i, x_j)$, where $f_{i,j}$ is defined as a function $\{0,\ 1\}^2 \ \rightarrow \ \mathcal{R}$. Since B is the sum of functions each depends on two random variables, pairwise independent random variables can be used for the search of a good point. Luby's scheme uses binary search which fixes one bit of a at a time and evaluates the conditional expectations. His algorithm[16] is shown below.

```
**Algorithm** Convert1:
**for** $l := 0$ **to** $k$
    **begin**
        $F_0 := E[B(x_0,\ x_1,\ ...,\ x_{n-1}) \,|\, a_0 = r_0,\ ...,\ a_{l-1} = r_{l-1},\ a_l = 0]$;
        $F_1 := E[B(x_0,\ x_1,\ ...,\ x_{n-1}) \,|\, a_0 = r_0,\ ...,\ a_{l-1} = r_{l-1},\ a_l = 1]$;
        **if** $F_0 \ge F_1$ **then** $r_l := 0$ **else** $r_l := 1$;
    **end**
output($a_0,\ a_1,\ ...,\ a_k$);
```

It is guaranteed that the sample point $(a_0,\ a_1,\ ...,\ a_k)$ found is a good point, *i.e.*, the value of B evaluated at $(a_0,\ a_1,\ ...,\ a_k)$ is $\ge E[B(x_0,\ x_1,\ ...,\ x_{n-1})]$.

By linearity of expectation, the conditional expectation evaluated in the above algorithm can be written as $E[B(x_0,\ x_1,\ ...,\ x_{n-1}) \ \,|\, a_0 = r_0,\ ...,\ a_l = r_l] = \sum_{i,j} E[f_{i,j}(x_i,\ x_j) \,|\, a_0 = r_0,\ ...,\ a_l = r_l]$. We assume that the input is dense, *i.e.*, $m = \theta(n^2)$. We will drop this assumption in the next section. We also assume that constant operations(instructions) are required for a single processor to evaluate $E[f_{i,j}(x_i,\ x_j) \,|\, a_0 = r_0, ... a_l = r_l]$. Algorithm Convert1 uses $O(n^2 \log n)$ operations. The algorithm can be implemented with $n^2/\log n$ processors and $O(\log^2 n)$ time on the EREW PRAM model.

We observe that as the sample space is being partitioned and reduced, the pairwise independence can no longer be maintained among n random variables. Thus dependency among random variables is expected. Such dependency is a form of redundancy which can be exploited.

After bit a_0 is set, random variables x_i and $x_{i\#0}$ become dependent, where $i\#0$ is obtained by complementing the 0-th bit of i. If a_0 is set to 0 then in fact $x_i = x_{i\#0}$. If a_0 is set to 1 then

$x_i = 1 - x_{i\#0}$. Therefore we can reduce n random variables to $n/2$ random variables. Since the input is dense, we are able to cut the number of PROFIT/COST functions from m to about $m/4$.

The modified algorithm is shown below.

```
**Algorithm** Convert2:
for $l := 0$ to $k$
    begin
        $F_0 := \sum_{i,j} E[f_{i,j}(x_i, x_j) \mid a_0 = r_0, ..., a_{l-1} = r_{l-1}, a_l = 0]$;
        $F_1 := \sum_{i,j} E[f_{i,j}(x_i, x_j) \mid a_0 = r_0, ..., a_{l-1} = r_{l-1}, a_l = 1]$;
        if $F_0 \geq F_1$ then $r_l := 0$ else $r_l := 1$;
        combine($f_{i,j}(x_i, x_j)$, $f_{i\#l,j}(x_{i\#l}, x_j)$,
            $f_{i,j\#l}(x_i, x_{j\#l})$ and $f_{i\#l,j\#l}(x_{i\#l}, x_{j\#l})$; for all $i$, $j$);
    end
output($a_0$, $a_1$, ..., $a_k$);
```

Some remarks on algorithm Convert2 is in order. The difference between the scheme used in Convert2 and previous schemes is that previous derandomization schemes use a static set of random variables while the set of random variables used in Convert2 changes dynamically as the derandomization process proceeds. Thus our scheme is a dynamic derandomization scheme while previous schemes are static derandomization schemes.

The redundancy resulting from the shrinking sample space is being exploited resulting in a saving of $O(\log n)$ operations. Algorithm Convert2 can be implemented with $n^2/\log^2 n$ processors while still running in $O(\log^2 n)$ time, since its computing time is bounded by $(c \log^2 n)(1 + \frac{1}{2^2} + \cdots + \frac{1}{2^{2k}})$ for a constant c.

A bit more parallelism can be extracted from algorithm Convert2 by using idling processors to speed up the later steps of the algorithm. When there are $n^2/2^i$ PROFIT/COST functions left, we can extend a by i bits. That is, we can make 2^i copies of the PROFIT/COST functions and examine all 2^i patterns of i bits, one copy corresponds to one pattern. The number of iterations will be cut down to j which is the smallest i such that $\sum_{k=0}^{i} 2^k \geq \log n + 1$. j is $O(\log \log n)$. Thus we are able to obtain the time complexity $O(\log n \log \log n)$ using $n^2/(\log n \log \log n)$ processors.

3 Exploiting Mutual Independence

In this section we show how mutual independence can be exploited to the advantage of parallel algorithm design. Our idea is embedded in the design of the random variables which is particularly suited to the parallel searching of a good sample point.

In the previous derandomization schemes[2][13][15] the main objective of the design of random variables is to obtain a minimum sized sample space. Because a small sample space requires less effort to search. Original ideas of the design of small sample spaces for random variables of limited independence can be found in [3][12][14]. Luby's result[16] shows that considerations should be given that the design of random variables should facilitate parallel search. Our result presented here carries this idea further in that our design of the random variables facilitates the dynamic derandomization process.

For the problem of finding a good sample point for function $B(x_0, x_1, .., x_{n-1}) = \sum_{i,j} f_{i,j}(x_i, x_j)$, Luby's technique of derandomization yields a DNC algorithm with time complexity $O(\log^2 n)$ using $O(m+n)$ processors. When the input is dense or the PROFIT/COST functions are properly indexed, Luby's technique yields time complexity $O(\log n \log\log n)$ with $O(m+n)$ processors. However, indexing the functions properly requires $O(\log^2 n \log\log n)$ time with his algorithm[16].

Previous solutions[5][16][18] to the problem uses limited independence in order to obtain a small sample space. A small sample space is crucial to the technique of binary search if DNC algorithms are demanded. In this section we present a case where mutual independence can be exploited to achieve faster algorithms. Since our sample space has an exponential number of sample points, binary search can not be used in our situation to yield a DNC algorithm. What happens in our scheme is that the mutual independence helps us to fix random variables independently, thus resulting in a faster algorithm.

We use n 0/1-valued uniformly distributed mutually independent random variables $r_i, 0 \le i < n$. Assume without loss of generality n is a power of 2. Function B has n variables. We build a tree T which is a complete binary tree with n leaves plus a node which is the parent of the root of the complete binary tree (thus there are n interior nodes in T and the root of T has only one child). The n variables of B are associated with n leaves of T and the n random variables are associated with the interior nodes of T. The n leaves of T are numbered from 0 to $n-1$. Variable x_i is associated with leaf i.

Variables x_i, $0 \le i < n$, are chosen randomly as follows. Let r_{i_0}, r_{i_1}, ..., r_{i_k} be the random variables on the path from leaf i to the root of T, where $k = \log n$. Random variable x_i is defined to be $x_i = (\sum_{j=0}^{k-1} i_j \cdot r_{i_j} + r_{i_k}) \bmod 2$, where i_j is the j-th bit in the binary expansion of i.

Lemma 1: Random variables x_i, $0 \le i < n$, are uniformly distributed mutually independent random variables.

Proof: By flipping the random bit at the root of the random variable tree we see that each x_i is uniformly distributed in $\{0, 1\}$. To show the mutual independence we note that the mapping $m(\overrightarrow{r}) \mapsto \overrightarrow{x}$, where $x_i = (\sum_{j=0}^{\log n-1} i_j \cdot r_{i_j} + r_{i_{\log n}}) \bmod 2$, is a one to one mapping. Thus $Pr(x_{i_1} = a_1, x_{i_2} = a_2, ..., x_{i_k} = a_k) = 2^{n-k}/2^n = 1/2^k = Pr(x_{i_1} = a_1)Pr(x_{i_2} = a_2) \cdots Pr(x_{i_k} = a_k)$. $\Box$

We shall call tree T the random variable tree.

We are to find a sample point $\overrightarrow{a} = (a_0, a_1, ..., a_{n-1})$ such that $B(\overrightarrow{a}) \ge E[B] = \frac{1}{4}\sum_{i,j}(f_{i,j}(0,\ 0) + f_{i,j}(0,\ 1) + f_{i,j}(1,\ 0) + f_{i,j}(1,\ 1))$.

Our algorithm fixes random variables r_i (setting their values to 0's and 1's) one level in a step starting from the level next to the leaves (we shall call this level level 0) and going upward on the tree T until level k. Since there are $k+1$ interior levels in T all random variables will be fixed in $k+1$ steps.

Now consider fixing random variables at level 0. Since there are only two random variables x_j, $x_{j\#0}$ which are functions of random variable r_i (node r_i is the parent of the nodes x_j and $x_{j\#0}$) and x_j, $x_{j\#0}$ are not related to other random variables at level 0, and since random variables at level 0 are mutually independent, they can be fixed independently. This apparently saves computing time because random variables can be fixed locally, thus eliminating the needed time for collecting global status.

Consider in detail the fixing of r_i which is only related to x_j and $x_{j\#0}$. We simply compute $f_0 = f_{j,j\#0}(0,\ 0) + f_{j,j\#0}(1,\ 1) + f_{j\#0,j}(0,\ 0) + f_{j\#0,j}(1,\ 1)$ and $f_1 = f_{j,j\#0}(0,\ 1) + f_{j,j\#0}(1,\ 0) + f_{j\#0,j}(0,\ 1) + f_{j\#0,j}(1,\ 0)$. If $f_0 \ge f_1$ then set r_i to 0 else set r_i to 1. Our scheme will allow all random variables at level 0 be set in parallel in constant time.

Next we apply the idea of exploiting redundancy. This reduces the n random variables x_i, $0 \le i < n$, to $n/2$ random variables. PROFIT/COST functions $f_{i,j}$ can also be combined, whenever two functions have the same variables they can be combined into one function. In order for functions $f_{i,j}$, $f_{i\#0,j}$, $f_{i,j\#0}$, $f_{i\#0,j\#0}$ to find each other and to be combined, we use n^2 memory cells and associate function $f_{i,j}$ with cell (i, j). Now the combining can be done in constant time using

$O(m+n)$ processors and $O(n^2)$ space.

We now have a new function which has the same form of B but has only $n/2$ variables. A recursion on our scheme solves the problem in $O(\log n)$ time with $O(m+n)$ processors.

Theorem 2: A sample point $\vec{a} = (a_0, a_1, ..., a_{n-1})$ satisfying $B(\vec{a}) \geq E[B]$ can be found in $O(\log n)$ time using $O(m+n)$ processors and $O(n^2)$ space. □

4 Derandomization using Tree Contraction

In this section we outline further improvements on our derandomization algorithm. We show that the derandomization process can be viewed as a special case of tree contraction[17]. By using the RAKE operation[17] we are able to cut down the processor and space complexities further.

A close examination of the process of derandomization of our algorithm shows that functions $f_{i,j}$ are combined according to the so-called file-major indexing for the two dimensional array, as shown in Fig. 1. In the file-major indexing the $n \times n$ array A is divided into four subfiles $A_0 = A[0..n/2-1,\ 0..n/2-1]$, $A_1 = A[0..n/2-1,\ n/2..n-1]$, $A_2 = A[n/2..n-1,\ 0..n/2-1]$, $A_3 = A[n/2..n-1,\ n/2..n-1]$. Any element in A_i proceeds any element in A_j if $i < j$. The indexing of the elements in the same subfile is recursively defined in the same way. The indexing of function $f_{i,j}$ is the number at the i-th row and j-th column of the array. After the bits at level 0 are fixed by our algorithm, functions indexed $4k$, $4k+1$, $4k+2$, $4k+3$, $0 \leq k < n^2/4$, will be combined. After the combination of these functions they will be reindexed. The new index k will be assigned to the function combined from the original functions indexed $4k$, $4k+1$, $4k+2$, $4k+3$. This allows the recursion in our algorithm to proceed.

Obviously we want the input to be arranged by the file-major indexing. When the input has been arranged by the file-major indexing, we are able to build a tree which reflects the way input functions $f_{i,j}$'s are combined as the derandomization process proceeds. We shall call this tree the derandomization tree. This tree is built as follows.

We use one processor for each function $f_{i,j}$. These PROFIT/COST functions are stored in an array by the file-major indexing. Let f_{i_1,j_1} be the function stored immediately before $f_{i,j}$ and f_{i_2,j_2} be the function stored immediately after $f_{i,j}$. By looking at the file-major indexing of f_{i_1,j_1} and f_{i_2,j_2} the processor could easily figure out at which step of the derandomization $f_{i,j}$ should be combined with f_{i_1,j_1} or f_{i_2,j_2}. This information allows the tree to be built for the derandomization process. This tree has $\log n + 1$ levels. Functions at level i will be combined immediately after the random variables at level i in the random variable tree are fixed. Note that we use the term level in the derandomization tree to correspond to the level of the random variable tree presented in the last section.

Since the derandomization tree has $O(\log n)$ levels, it can be built in $O(\log n)$ time using m processors. If the input is arranged by the file-major indexing, the tree can be built in $O(\log n)$ time using an optimal $m/\log n$ processors as follows. Label each leaf with the level at which it is to be combined with other functions and the leaf must be in the right subtree in the combined tree. For example, if the input functions have the file-major indexing 0, 3, 4, 5, then they are labeled as 2, 0, 1, 0. After the labeling we process in step i of the tree construction only those leaves labeled with i. In our example, we process 3 and 5 in step 0, *i.e.* build the parent for 0 and 3 and the parent for 4 and 5. In Step 1 we process 4, *i.e.* build the parent for the tree containing 4 and the neighboring tree to the left (the tree containing 0 and 3). In step 2 we build the root of the whole tree. If there are n_i leaves labeled with i then step i takes $O(n_i \log n/m)$ time with $m/\log n$ processors. Summing for all i we get $O(\log n)$ time.

A derandomization tree is shown in Fig. 2.

The derandomization process can now be described in terms of the derandomization tree. Combine functions at level i of the derandomization tree immediately after the random variables at level i of the random variable tree are fixed. The combination can be accomplished by the RAKE[17] operation which rakes off leaves which are children of nodes at level i. The whole process of the derandomization can now be viewed as a process of tree contraction which uses only the rake operation without using the COMPRESS operation[17].

We can use known parallel sorting algorithms[1][7] to sort input functions into the file-major indexing and then build the derandomization tree. With the help of the derandomization tree the space requirement of our algorithm is reduced from $O(n^2)$ to $O(m)$.

Theorem 3: A sample point $\vec{a} = (a_0, a_1, ..., a_{n-1})$ satisfying $B(\vec{a}) \geq E[B]$ can be found on the CREW PRAM in $O(\log n)$ time using m processors and $O(m)$ space. $\Box$

Since the derandomization tree has only $O(m)$ tree nodes we show that, except the first sorting step which sorts input functions by the file-major indexing, it is possible to further reduce the processor requirement from m to $m/\log n$ (which is optimal) while maintaining time complexity $O(\log n)$ for our algorithm.

If a constant number of instructions are executed for each node in the derandomization tree we will spend only $O(m)$ operations for finding the good point. Let v be a node labeled (i, j) at level l having its parent at level $l + c$. The problem here is that when v is to be combined at level $l + c$ the random variables x_i and x_j have to be updated because there are c random bits for x_i and c random bits for x_j in the random variable tree which have already been fixed. If we are to read these random bits off the random variable tree we need spend $O(c)$ instructions for node v, instead of a constant number of instructions. To avoid this situation we keep updated value for all random variables $x_0, x_1, ..., x_{n-1}$ once random bits in a level of the random variable tree are fixed. Thus after random bits at level l are fixed, x_i is updated to $(\sum_{j=0}^{l} x_{i_j} \cdot a_{i_j})$ mod 2, where a_{i_j} are the fixed random bits on the path from x_i to the l-th level of the random variable tree. With these updated variables a node v in the derandomization tree can read off its updated value in constant time instead of using c instructions to update itself. Now we spend only a constant number of instructions for each node in the derandomization tree, the total number of instructions or operations is $O(m)$ for the derandomization tree. Since we update random variable at each level we need an extra $O(n \log n)$ operations. Thus we have

Theorem 4: A sample point $\vec{a} = (a_0, a_1, ..., a_{n-1})$ satisfying $B(\vec{a}) \geq E[B]$ can be found on the CREW PRAM in $O(m/p + n \log n/p + \log n)$ time using p processors and $O(m)$ space if the input is arranged by the file-major indexing. □

To remove the term $n \log n/p$ from the time complexity we first reduce the number of random variables to $n/\log n$ using a scheme to be described below. After that we may still have $O(m)$ PROFIT/COST functions left, if we apply the algorithm given by Theorem 4 the time complexity becomes $O(m/p + n/p + \log n) = O(m/p + \log n)$ because we can assume $m \geq n$.

We use a random variable tree as shown in Fig. 3. There are $n/\log n$ chains at the lowest $\log \log n$ levels of the random variable tree. When all random bits in these $\log \log n$ levels are fixed, the number of random variables is reduced to $n/\log n$. The $n/\log n$ chains represent $n/\log n$ independent PROFIT/COST problems. We consider functions $f_{i,j}$ such that the lowest common ancestor of x_i and x_j is at level $\leq \log \log n$. This level is defined to be the index l of $f_{i,j}$, denoted by $l(f_{i,j})$. All such functions associated with a chain is a PROFIT/COST problem containing $O(\log^2 n)$ PROFIT/COST functions. Such a PROFIT/COST problem is solved by using Luby's algorithm as shown below.

Algorithm Reduce:

```
for $i := 0$ to $\log\log n$
    begin
        $F_0 := \sum_{l(f_{i,j})=i} E[f_{i,j}(x_i, x_j) \mid a_0 = r_0, ..., a_{i-1} = r_{i-1}, a_i = 0]$;
        $F_1 := \sum_{l(f_{i,j})=i} E[f_{i,j}(x_i, x_j) \mid a_0 = r_0, ..., a_{i-1} = r_{i-1}, a_i = 1]$;
        if $F_0 \geq F_1$ then $r_i := 0$ else $r_i := 1$;
    end
output($a_0, a_1, ..., a_{\log\log n}$);
```

Because each PROFIT/COST function is evaluated only once in procedure Reduce the time complexity of Reduce is $O(m/p + (\log\log n)^2)$. After the execution of Reduce the n random variables $x_0, x_1, ..., x_{n-1}$ can be reduced to $O(n/\log n)$ random variables.

Theorem 5: A sample point $\vec{a} = (a_0, a_1, ..., a_{n-1})$ satisfying $B(\vec{a}) \geq E[B]$ can be found on the CREW PRAM in $O(\log n)$ time using optimal $m/\log n$ processors and $O(m)$ space if the input is arranged by the file-major indexing. $\Box$

The concurrent read is used in our algorithm when all functions $f_{i,j}$, $0 \leq j < n$, check the setting of the random variable r which is the parent of x_i and $x_{i\#0}$ in the random variable tree. In next section we show how to remove the concurrent read feature from our algorithm.

5 An EREW Algorithm

We show how to obtain an EREW PRAM algorithm for the PROFIT/COST problem with time complexity $O(\log n)$ using $O(m + n)$ processors.

Let $n = 2^k$ and A be an $n \times n$ array. Elements $A[i, j]$, $A[i, j\#0]$, $A[i\#0, j]$, $A[i\#0, j\#0]$ form a gang which is denoted by $g_A[\lfloor i/2 \rfloor, \lfloor j/2 \rfloor]$. All gangs in A form array g_A.

5.1 The Dense Case

To remove the concurrent read feature in the CREW algorithm we design a pipeline for disseminating the bit setting information.

We assume all n^2 PROFIT/COST functions $f_{i,j}$ are present in the input. We store these functions in an $n \times n$ array A_0. Function $f_{i,j}$ is stored in $A_0[i, j]$. The leaves of the derandomization tree D are the elements of array A_0. The l-th level of D is an $\frac{n}{2^l} \times \frac{n}{2^l}$ array A_l. The children of

$A_l[i,j]$ is $A_{l-1}[2i,2j]$, $A_{l-1}[2i+1,2j]$, $A_{l-1}[2i,2j+1]$ and $A_{l-1}[2i+1,2j+1]$. Assign one processor to each element in each array. The total number of processors used is $\sum_i \frac{n^2}{4^i} = O(n^2)$.

The problem in obtaining an EREW algorithm is how to combine functions without using concurrent read. Consider the processors working on array A_0. The random bits at level 0 of the random variable tree are set by the processors in the diagonal gang of A_0. There are a total of $O(n)$ processors in the diagonal gang. These processors know the bit setting information when they set the random bits. However, in order for functions in A_0 to be combined into functions in A_1 all processors in A_0 need to know the bit setting information. There are $O(n^2)$ processors in A_0. Therefore it takes $O(\log n)$ time for the processors in the diagonal gang to disseminate the bit setting information to all processors in A_0 if concurrent read is not allowed. If functions in A_0 are combined into functions in A_1 after all processors in A_0 get the bit setting information then no function in A_1 will be defined until processors in A_0 spend $O(\log n)$ steps for acquiring the bit setting information. It will lead to an $O(\log^2 n)$ time algorithm because level l requires $O(\log(n/2^l))$ steps to disseminate bit setting information.

We observe that functions in A_0 which are close to the diagonal need to be combined sooner than functions which are far away from the diagonal (those which are close to the upper right corner and lower left corner). This is because functions in the diagonal gang of A_1 are obtained from combined functions close to the diagonal in A_0, while functions which are far away from the diagonal will not be combined into a function in a diagonal gang until at a higher level in the derandomization tree. This observation enables us to use a pipeline to disseminate the bit setting information.

Consider the action of processors at level l (those working on A_l). We define $n/2^{l+1}$ groups for elements in A_l (this defines groups for processors at level l as well). $A_l[i,j]$ is in group $|\lfloor i/2 \rfloor - \lfloor j/2 \rfloor|$. Refer to Fig. 4. Define step 0 for level l as the step immediately after $A_l[i,i]$, $0 \le i < n/2^l$, are defined (they come from the combination of functions at level $l-1$). The algorithm for processors at level l is shown below.

Step 0.
(* At the beginning of this step elements (functions) in $A_l[i,i]$, $0 \le i < n/2^l$, are defined. *)
No action is taken in this step.
Step 1.

(* At the beginning of this step elements in $A_l[i,j]$, $|i-j|<2$, are defined. Therefore elements in group 0 are defined. *)
Processors in group 0 determine how the random variables at l-th level of the random variable tree are set and send the bit setting information (for levels 0 to l) to elements in group 1. They also combine functions in group 0 and send combined functions and the bit setting information to level $l+1$.
(* This action defines elements in $A_{l+1}[i,i], 0 \le i < n/2^{l+1}$. Note also that the bit setting information need not be sent to level $l+1$ in the dense case we are dealing with here, but it needs to be sent to level $l+1$ in the general case. *)
Step t (t >1).
(* At the beginning of this step elements in $A_l[i,j]$, $|i-j|<2^t$, are defined. Therefore elements in groups i, $0 \le i < 2^{t-1}$, are defined. Elements in groups i, $0 \le i < 2^{t-1}$, also have acquired the bit setting information. Besides, elements in groups i, $0 \le i < 2^{t-2}$, have already sent combined functions to level $l+1$. *)
Processors in groups i, $0 \le i < 2^{t-1}$, send the bit setting information to elements in groups i', $2^{t-1} \le i' < 2^t$. Processors in groups i, $2^{t-2} \le i < 2^{t-1}$, combine functions in these groups and send combined functions to level $l+1$.
(* This action defines elements in $A_{l+1}[i,j], 2^{t-2} \le |i-j| < 2^{t-1}$. *)

An example of the execution of the above algorithm is shown in Fig. 4.
Theorem 6: The PROFIT/COST problem can be solved in $O(\log n)$ time and $O(n^2)$ space with $O(n^2)$ processors on the EREW PRAM.
Proof: As can be seen in the algorithm, step t at level l is step $t-2c$ at level $l+c$ for $c \ge 0$. Thus step $2\log n+1$ at level 0 is step 1 at level $\log n$. Since there is only one function left at level $\log n$ it takes one step to finish the computation at that level. Therefore the time complexity of the algorithm is $O(\log n)$.

To prove the correctness of the algorithm we need to show, at step t of level 0, that the elements in groups i, $0 \le i < 2^{t-2l-1}$, at level l are defined and have acquired the bit setting information, that processors in groups i, $0 \le i < 2^{t-2l-2}$, have already sent combined functions to level $l+1$. These can be proved by induction as follows. For $t=1$, the elements in group 0 at level 0 are defined and no combined function has been sent to level 1. Assume that for t the hypothesis is true. At step $t+1$, at each level the number of groups defined and the number of processors which have already sent the combined functions to a higher level will double according to the algorithm, therefore at level l the elements in groups i, $0 \leq i < 2^{t-2l}$, are defined and the processors in groups i, $0 \leq i < 2^{t-2l-1}$, have already sent combined functions to level $l+1$. When $t = 2l$ the elements in group 0 at level l will be defined in step $t+1$.

Note that the number of groups which have received the bit setting information doubles in one step, therefore there is no need to use concurrent read to disseminate the bit setting information. $\square$

5.2 The General Case

The algorithm for the dense case cannot be used directly for the general case if we intend to use $O(m+n)$ processors and achieve $O(\log n)$ time. The reason is that many elements in A_l may not be present and therefore no processors are allocated to these elements, while the dissemination of the bit setting information in the algorithm for the dense case assumes that all these processors are available. Another problem is that the algorithm for the dense case uses $O(n^2)$ space which we want to avoid for the general case. As will be seen that $O(m \log n)$ space is sufficient for the general case. The saving in space is significant if m is much less than n^2.

In order to facilitate the dissemination of the bit setting information we use derandomization trees which contain nodes of single child. Thus the derandomization tree given in Fig. 2 should be modified. The modified tree is shown in Fig. 5. Such a derandomization tree contains $O(m \log n)$ nodes.

In the general case if each node in the derandomization tree receives the bit setting information no later than it would receive the information in the algorithm for the dense case, then the pipeline we designed will still be valid. We shall call this condition the synchronization condition. Therefore our algorithm for the dense case can be used for the general case if we have a scheme for disseminating the bit setting information which satisfies the synchronization condition. In this subsection we show how to adapt our algorithm for the dense case to the general case by using *row and column trees* to disseminate the bit setting information.

We first build the derandomization tree D. This is done by sorting the input into file-major indexing and constructing the tree bottom up. We then build row and column trees for nodes at each level of the derandomization tree for disseminating the bit setting information.

The row and column trees at level l are built for nodes at level l of D. The column trees are built for disseminating the bit setting information from gang $g_{A_l}[j,j]$ to all the gangs in the same column ($g_{A_l}[k,j], 0 \le k < n/2^{l+1}$) while the row trees are for disseminating the information from gang $g_{A_l}[i,i]$ to all the gangs in the same row ($g_{A_l}[i,k], 0 \le k < n/2^{l+1}$). Without loss of generality we may assume that all $A_l[i,i]$, $0 \le l \le \log n$, $0 \le i < n/2^l$, are present in the input. If some of them are not present we use zero functions to represent them. Because we are adding no more than $O(n)$ nodes to the derandomization tree, the time complexity of our algorithm will not be affected. The construction and the function of row and column trees are similar, so we only discuss how to construct and use row trees. For nodes in A_l we only consider how to build row trees for gangs in $g_{A_l}[i,j], j \ge i$, and disseminate information on the tree. Again the tree for gangs $g_{A_l}[i,j], j \le i$, can be constructed and used similarly.

Assign index value $index(i,j) = i * n + i + sign(j-i) * brv(|j-i|, \log(n/2^{l+1}))$ to $g_{A_l}[i,j]$ if it is not empty (*i.e.*, at least one node in the derandomization tree D is in $g_{A_l}[i,j]$), where $sign$ is the sign function and $brv(i,j)$ is the bit reversal function which takes j least significant bits from i and reverses these bits to get the function value. Now sort elements in g_{A_l} by the $index$ value. $i * n$ in the index ensures that after sorting elements in the same row are consecutive in the sorted array. i and $sign(j-i)$ in the index ensures that elements in row i are arranged by the sorting so that $g_{A_l}[i,j]$ is before $g_{A_l}[i,i]$ if $j < i$ and $g_{A_l}[i,j]$ is after $g_{A_l}[i,i]$ if $j > i$.

For each row i a row tree R is built for $g_{A_l}[i,j], i \le j < n/2^{l+1}$. The tree can be built bottom-up. $g_{A_l}[i,j], i \le j < n/2^{l+1}$, are stored at level 0, *i.e.*, leaves, of R. A node v at level k of R is created if both ranges, $n * 2^{k-1} \le index(i,j) < (n+1) * 2^{k-1}$ and $(n+1) * 2^{k-1} \le index(i,j) < n * 2^k$, are not empty (*i.e.*, there are input functions whose index value fall into these two ranges). The construction of R is straightforward after $g_{A_l}[i,j]$'s are sorted by the index values. An example of such a tree is shown in Fig. 6.

Each node in R is labeled. Leaf $g_{A_l}[i,j]$ is labeled with (i,j). A parent is labeled with the label of one of its children which has the smaller $|j-i|$ value. Thus the root is labeled with (i,i) which represents $g_{A_l}[i,i]$. The bit setting information propagates from the root down to the leaves. $g_{A_l}[i,j]$ gets the information at the smallest depth of the tree where label (i,j) appears. Thus $g_{A_l}[i,i]$ gets the bit setting information at depth 0 and the children of the root of R get the bit setting information at depth 1. Again we define step 0 for level l of D as the step immediately after $A_l[i,i]$, $0 \le i < n/2^l$, are defined. In step t for level l of D the bit setting information is transmitted from nodes at depth $t-1$ to nodes at depth t on the row tree R. It is easy to see that if all $g_{A_l}[i,j]$, $i \leq j < n/2^{l+1}$, are present, the bit setting information is disseminated on the row tree in exactly the same way as it is disseminated in the algorithm for the dense case.

Theorem 7: The PROFIT/COST problem can be solved in $O(\log n)$ time and $O(m \log n)$ space with $O(m+n)$ processors on the EREW PRAM.

Proof: The time complexity for building the derandomization tree and row and column trees is $O(\log n)$ with $O(m+n)$ processors. The correctness of the algorithm is now proved by an induction showing at step t at level 0, that the elements in groups i, $0 \leq i < 2^{t-2l-1}$, at level l are defined and have acquired the bit setting information, and that elements in groups i, $0 \leq i < 2^{t-2l-2}$, have already sent combined functions to level $l+1$. We note that the only difference between the dense case and the general case is that the bit setting information is transmitted on the row and column trees in the general case while there is no need to explicitly construct the row and column trees in the dense case. In the algorithm for the general case a function to be combined with other functions will be defined and receive the bit setting information at a step that is no later than the step it would be defined and receive the bit setting information in the algorithm for the dense case we described in the last subsection. Thus the action taken by a processor at a node in D is the same as that in the algorithm for the dense case except the transmission of the bit setting information is now carried on the row and column trees. □

6 Conclusions

The PROFIT/COST problem is an important problem for which best parallel algorithm ought to be sought. The algorithm presented in this paper achieves time $O(\log n)$ using $O(m+n)$ processors. It is not clear whether time $O(\log n)$ can be achieved on the EREW PRAM using an optimal number $O(m/\log n)$ processors. Since we have used a pipeline and row and column trees the structure of the algorithm obtained is rather complicated. It is desirable to obtain an EREW algorithm which has a simpler structure.

Several applications of the PROFIT/COST problem are outlined in [9][16]. We expect that many more important applications of the PROFIT/COST problem will be found.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi: An $O(N \log N)$ sorting network. Proc. 15th ACM Symp. on Theory of Computing, 1-9(1983).

[2] N. Alon, L. Babai, A. Itai: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. of Algorithms 7, 567-583(1986).

[3] S. Bernstein: "Theory of Probability," GTTI, Moscow 1945.

[4] R. A. Borodin and J. E. Hopcroft: Routing, merging and sorting on parallel models of computation. Proc. 14th ACM Symp. on Theory of Computing, 1982, pp. 338-344.

[5] B. Berger, J. Rompel: Simulating $(\log^c n)$-wise independence in NC. Proc. 1989 IEEE FOCS, 2-7.

[6] B. Berger, J. Rompel, P. Shor: Efficient NC algorithms for set cover with applications to learning and geometry. Proc. 1989 IEEE FOCS, 54-59.

[7] R. Cole: Parallel merge sort. 27th Symp. on Foundations of Comput. Sci., IEEE, 511-516(1986).

[8] S. Fortune and J. Wyllie: Parallelism in random access machines. Proc. 10th Annual ACM Symp. on Theory of Computing, San Diego, California, 1978, 114-118.

[9] Y. Han: A fast derandomization scheme and its applications. SIAM J. Comput. Vol. 25, No. 1, pp. 52-82, February 1996.

[10] Y. Han: A parallel algorithm for the PROFIT/COST problem. Proc. 1991 Int. Conf. on Parallel Processing, Vol. III, 107-114, (Aug. 1991).

[11] Y. Han and Y. Igarashi: Derandomization by exploiting redundancy and mutual independence. Proc. SIGAL 1990, LNCS 450, 328-337.

[12] A. Joffe: On a set of almost deterministic k-independent random variables. Ann. Probability 2(1974), 161-162.

[13] R. Karp, A. Wigderson: A fast parallel algorithm for the maximal independent set problem. JACM 32:4, Oct. 1985, 762-773.

[14] H. O. Lancaster: Pairwise statistical independence, Ann. Math. Stat. 36(1965), 1313-1317.

[15] M. Luby: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. 15:4, Nov. 1986, 1036-1053.

[16] M. Luby: Removing randomness in parallel computation without a processor penalty. Proc. 1988 IEEE FOCS, 162-173.

[17] G. L. Miller and J. H. Reif: Parallel tree contraction and its application. Proc. 1985 IEEE FOCS, 478-489.

[18] R. Motwani, J. Naor, M. Naor: The probabilistic method yields deterministic parallel algorithms. Proc. 1989 IEEE FOCS, 8-13.

[19] G. Pantziou, P. Spirakis, C. Zaroliagis: Fast parallel approximations of the maximum weighted cut problem through Derandomization. FST&TCS 9: 1989, Bangalore, India, LNCS 405, 20-29.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Fig. 1. File-major indexing.

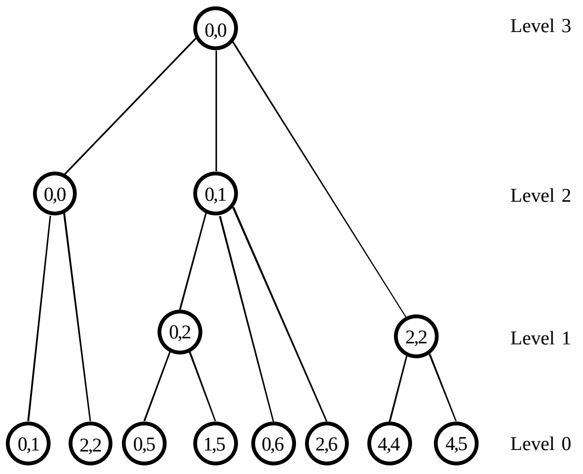


Fig. 2. A derandomization tree. Pairs in the circles are the subscripts of PROFIT/COST functions.

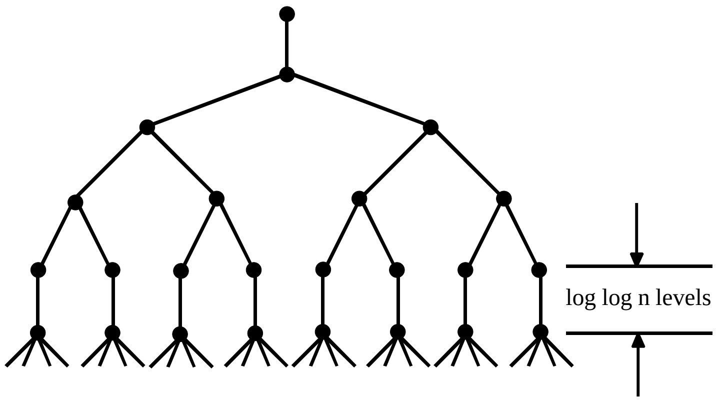
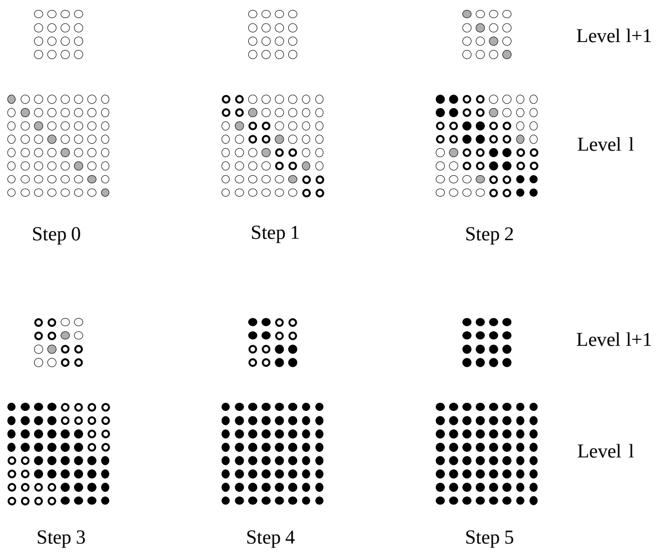


Fig. 3.



○ Undefined node.
◍ Defined node.
○ Node received bit setting information.
● Node which has been combined.

Fig. 4.

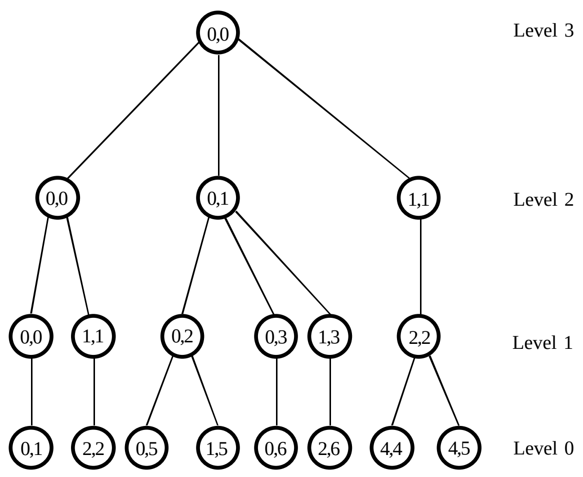

0,0
Level 3
0,0
0,1
1,1
Level 2
0,0
1,1
0,2
0,3
1,3
2,2
Level 1
0,1
2,2
0,5
1,5
0,6
2,6
4,4
4,5
Level 0

Fig. 5.

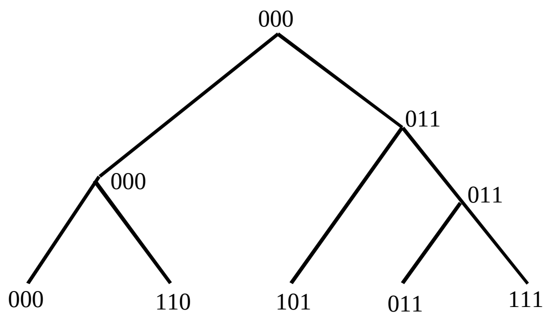


Fig. 6.