Parallel Processing Letters Vol. 5 No. 3 (1995) 367–373


# PARALLEL ALGORITHMS FOR MAXIMUM SUBSEQUENCE AND MAXIMUM SUBARRAY

KALYAN PERUMALLA
*College of Computing, Georgia Institute of Technology*
*Atlanta, GA 30332-0280, USA*

and

NARSINGH DEO
*Department of Computer Science, University of Central Florida*
*Orlando, FL 32816-2362, USA*

Received January 1994
Revised May 1995
Accepted by J. Barker

ABSTRACT

Given a sequence $Q$ of $n$ numbers (positive and negative), the *maximum subsequence* of $Q$ is the contiguous subsequence that has the maximum sum among all contiguous subsequences of $Q$. Given a two-dimensional array $A$ of $n \times n$ numbers (positive and negative), the *maximum subarray* of $A$ is the contiguous subarray that has the maximum sum among all contiguous subarrays of $A$. We present two $O(\log n)$-time parallel algorithms — one for finding the *maximum subsequence sum* of a given sequence, and the other for finding the *maximum subarray sum* of a given array. The former is optimal on an EREW PRAM. The latter is optimal on a CREW PRAM, in the sense that the time-processor product matches the current sequential upperbound of $O(n^3)$.

*Keywords*: Subsequence, subarray, prefix sums, suffix sums, prefix maxima, suffix maxima.



## 1. Introduction

Given an $n \times n$ array $A$ of reals (positive and negative), the problem of finding a rectangular subarray with maximum sum arises in two-dimensional pattern matching [1]. Such a maximum-sum subarray corresponds to a maximum-likelihood estimator of a certain kind of pattern in a digitized picture. A simplification of the two-dimensional problem to a one-dimensional one is the following: Given a sequence $Q$ of $n$ reals (positive and negative), find a subsequence which has the maximum sum among all contiguous subsequences in $Q$.

A linear-time sequential algorithm for the one-dimensional problem, attributed to Jay Kadane, is given in [1]. Furthermore, as reported in [1], Ulf Grenander of Brown University, who originally in 1977 formulated the two-dimensional problem, "abandoned that approach to the pattern-matching problem," because no

reasonably fast algorithm (sequential) could be found. A formal design of a linear-time sequential algorithm and an analysis for the one-dimensional problem are given in [2] and [3], respectively. A unified approach to both the one-dimensional and two-dimensional problems can be found in [4] and [5]. A solution generalized to higher dimensions is presented in [4], with a sequential time-complexity of $O(N^{\frac{2d-1}{d}})$ for a $d$-dimensional matrix of $N$ elements. A divide-and-conquer approach to the problems is given in [5], where algorithms are presented that are amenable to parallelization in a natural way. Specifically, algorithms with sequential-time complexities of $O(n)$ and $O(n^3)$, and parallel-time complexities of $O(\log n)$ and $O(\log^2 n)$, corresponding to one- and two-dimensions, are presented.

We present two $O(\log n)$-time parallel algorithms — one for each of the two problems — on the *EREW PRAM* model. Our approach is substantially different from those employed earlier. Furthermore, since the number of processors we use in solving the one-dimensional problem is $O(n/\log n)$, and the best possible sequential algorithm is $O(n)$-time, the algorithm is cost-optimal. For the two-dimensional case, the number of processors used in our parallel algorithm is $O(n^3)$ on EREW PRAM, and $O(n^3/\log n)$ on CREW PRAM. Since the best known sequential time complexity for the two-dimensional problem is $O(n^3)$, the second parallel algorithm is optimal on CREW PRAM, in the sense that the time-processor product matches the current sequential upperbound of $O(n^3)$.

## 2. The Maximum Subsequence Problem

Given a sequence $Q = [q_1, q_2, \ldots, q_n]$, let $Q_{ij}$ be the subsequence $[q_i, q_{i+1}, \ldots, q_j]$, and let $Range(q_k)$ be the set of all the subsequences of $Q$ that include $q_k$ in them.

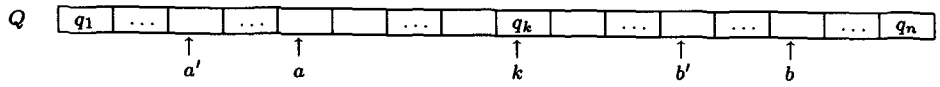
**Note:** If all $q_i$ are negative, then the maximum sum is defined to be the least negative number. This can be easily redefined to be zero, if desired.

Consider an element of the given sequence $Q$, say $q_k$. Let $l(q_k) = [q_1, q_2, \ldots, q_{k-1}, q_k]$ be the elements of $Q$ to the left of $q_k$ including $q_k$. Let $r(q_k) = [q_k, q_{k+1}, \ldots, q_n]$ be the elements of $Q$ to the right of $q_k$ including $q_k$. Let $S_l(q_k) = [s_1, s_2, \ldots, s_{k-1}, s_k]$ be the suffix sums[a] of $l(q_k)$, and $P_r(q_k) = [p_k, p_{k+1}, \ldots, p_n]$ be the prefix sums[a] of $r(q_k)$. Also, let $M_s^k$ be the maximum of $S_l(q_k)$, and $M_p^k$ be the maximum of $P_r(q_k)$.

**Lemma.** *The maximum among the sums of all subsequences that include $q_k$ in them is given by $Max(q_k) = M_s^k + M_p^k - q_k$.*

**Proof.** Let $SQ_{ij}$ denote the sum of elements in $Q_{ij}$. Let the subsequence corresponding to the sum given by the preceding equation be $Q_{ab}^k$. Such a subsequence exists since $M_s^k = SQ_{ak}$ and $M_p^k = SQ_{kb}$ for some $1 \leq a \leq k \leq b \leq n$. Let there exist another subsequence $Q_{a'b'}^k$ that includes $q_k$, such that its sum, $SQ_{a'b'}^k$, is greater than $SQ_{ab}^k$.

[a]See [6] for definitions of *suffix sums*, *prefix sums*, *suffix maxima* and *prefix maxima*.




Since the subsequence $Q^k_{a'b'}$ can be viewed as two subsequences — $Q_{a'k}$ and $Q_{kb'}$ — overlapping at $q_k$, the sum $SQ^k_{a'b'}$ can be written as $SQ^k_{a'b'} = SQ_{a'k} + SQ_{kb'} - q_k$. But, since by definition $M^k_s = SQ_{ak} \geq SQ_{ik}$ for all $1 \leq i \leq k$, it follows that $SQ_{ab'} \geq SQ_{a'b'}$. A similar argument holds for $b$ and $b'$, giving $SQ_{ab} \geq SQ_{ab'}$. Thus, $SQ^k_{ab} \geq SQ^k_{a'b'}$ for all $SQ^k_{a'b'} \in Range(q_k)$. □

EXAMPLE:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q$ | 3 | 2 | −7 | 11 | 10 | −6 | 4 | 9 | −6 | 1 | −2 | −3 | 4 | −3 | 0 | 2 |
| $l(q_7)$ | 3 | 2 | −7 | 11 | 10 | −6 | 4 | | | | | | | | | |
| $S_l(q_7)$ | 17 | 14 | 12 | 19 | 8 | −2 | 4 | | | | | | | | | |
| $r(q_7)$ | | | | | | | 4 | 9 | −6 | 1 | −2 | −3 | 4 | −3 | 0 | 2 |
| $P_r(q_7)$ | | | | | | | 4 | 13 | 7 | 8 | 6 | 3 | 7 | 4 | 4 | 6 |

$M^7_s = \text{Maximum}(S_l) = 19$, $M^7_p = \text{Maximum}(P_r) = 13$
$M^7_s + M^7_p - q_7 = 19 + 13 - 4 = 28 = \text{Max}(q_7)$

Suppose that $\text{Max}(q_k)$ is computed for all $q_k$, i.e. for each element, $q_k$, of the sequence $Q$, the maximum sum of all subsequences of $Q$ that include $q_k$ is computed. Then clearly, the maximum subsequence sum of the sequence $Q$ is the maximum of those maximums, i.e.

$$\text{MaxSeqSum} = \text{Maximum}(\text{Max}(q_k),\ 1 \leq k \leq n)\,.$$

Since the suffix sums of $q_{k-1}$ are related to the suffix sums of $q_k$ , it is enough to compute the suffix sums of the whole sequence only once. Similar reasoning holds for the prefix sums. In other words, if $S_l(q_k) = [s_1, s_2, \ldots, s_k]$, then $S_l(q_{k-1}) = [s_1 - q_k, s_2 - q_k, \ldots, s_{k-1} - q_k]$. Similar relation holds for prefix sums. Also, to find the maximum of $S_l(q_k)$ for all $k$, it is enough to compute the prefix maxima[a] of the suffix sums of $Q$. Similarly, to obtain the maximum of $P_r(q_k)$ for all $k$, it is enough to compute the suffix maxima[a] of the prefix sums of $Q$.

These observations lead to the following parallel algorithm for finding the maximum subsequence sum:

**Algorithm:** Maximum Subsequence Sum
**Input:** Sequence $Q[1..n]$ of numbers (positive and negative).

**Output:** Maximum Subsequence sum of $Q$.
**Begin Algorithm**

1. Compute in parallel the prefix sums of $Q$ into array $PSUM$.
2. Compute in parallel the suffix sums of $Q$ into array $SSUM$.
3. Compute in parallel the suffix maxima of $PSUM$ into array $SMAX$.
4. Compute in parallel the prefix maxima of $SSUM$ into array $PMAX$.
5. For $1 \leq i \leq n$ do **in parallel**
   (a) $M_s[i] := PMAX[i] - SSUM[i] + Q[i]$
   (b) $M_p[i] := SMAX[i] - PSUM[i] + Q[i]$
   (c) $M[i] := M_s[i] + M_p[i] - Q[i]$
4. Find the maximum of $M$ into $MSQ$.
5. Output $MSQ$.

**End Algorithm**

EXAMPLE:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q$ | 3 | 2 | −7 | 11 | 10 | −6 | 4 | 9 | −6 | 1 | −2 | −3 | 4 | −3 | 0 | 2 |
| $PSUM$ | 3 | 5 | −2 | 9 | 19 | 13 | 17 | 26 | 20 | 21 | 19 | 16 | 20 | 17 | 17 | 19 |
| $SSUM$ | 19 | 16 | 14 | 21 | 10 | 0 | 6 | 2 | −7 | −1 | −2 | 0 | 3 | −1 | 2 | 2 |
| $SMAX$ | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 21 | 21 | 20 | 20 | 20 | 19 | 19 | 19 |
| $PMAX$ | 19 | 19 | 19 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 21 |
| $M$ | 26 | 26 | 26 | 28 | 28 | 28 | 28 | 28 | 23 | 23 | 22 | 22 | 22 | 21 | 21 | 21 |

Maximum Subsequence Sum = Maximum(M) = 28 = Sum($Q_{4,8}$)

**Complexity.**

Steps 1 and 2 can be performed in $O(\log n)$ time using the standard parallel prefix and parallel suffix algorithms [7]. Steps 3 and 4 can be performed with $O(n/\log n)$ processors on an *EREW-PRAM* machine in $O(\log n)$ time using variations of parallel prefix and parallel suffix algorithms [6]. Similarly, Step 6 can be done optimally in $O(\log n)$ time. Step 5 can be easily scheduled to be done in the same time optimally. Thus, the total algorithm can be performed in $O(\log n)$ time.

Since the best sequential algorithm for the problem requires $O(n)$ time (which is also the lower bound), the algorithm is optimal.

The algorithm can be modified in a straightforward way if the actual maximum subsequence is needed instead of just the sum. This is done by computing the

indices of the prefix maximum and suffix maximum for each element, instead of just the values.

## 3. The Maximum Subarray Problem

Given an array $A = [a_{ij}]$, $1 \leq i, j \leq n$, let $A_{i_1 j_1 i_2 j_2}$ be the subarray $[a_{ij}]$, $1 \leq i_1 \leq i \leq i_2 \leq n$, $1 \leq j_1 \leq j \leq j_2 \leq n$.

**Note:** If all $a_{ij}$ are negative, then the maximum sum is defined to be the least negative number. This can be easily redefined to be zero, if desired.

Given an $n \times n$ array $A$, to find the maximum subarray sum. For a given $(g, h)$-pair, $1 \leq g \leq h \leq n$, construct the column (sequence) $C^{gh}$, of size $n$, by compressing (summing) all the columns of $A$ between $g$ and $h$ inclusive, i.e. $C_i^{gh} = \sum_{j=g}^{h} a_{ij}$.

Denote by $R^{gh}$, the set of all the subarrays of the array $A$ that start at the $g$th column and end at the $h$th column of the array, i.e. all $SA_{i_1 g i_2 h}$, $1 \leq i_1 \leq i_2 \leq n$.

It is clear that the maximum subsequence sum of $C^{gh}$ is the same as the maximum of the sums of all the subarrays belonging to $R^{gh}$.

EXAMPLE:

$A =$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | −1 | 12 | −9 | 12 | 8 | 6 | 1 |
| 5 | −4 | −12 | 10 | −1 | 1 | 2 | −9 |
| 3 | −8 | −9 | 11 | −10 | −2 | −8 | −7 |
| −7 | 9 | −11 | 9 | −2 | 10 | −2 | 9 |
| 0 | −8 | 8 | −1 | −1 | 5 | 3 | −6 |
| 6 | −5 | −9 | 7 | −2 | 11 | 4 | 8 |
| 2 | 6 | 7 | −8 | 2 | −3 | −6 | −8 |
| 4 | 5 | −1 | −8 | 1 | −7 | 10 | 6 |

$C^{3,6} =$

| | | | | | |
|---|---|---|---|---|---|
| 12 | −9 | +12 | +8 | = | 23 |
| −12 | +10 | −1 | +1 | = | −2 |
| −9 | +11 | −10 | −2 | = | −10 |
| −11 | +9 | −2 | +10 | = | 6 |
| 8 | −1 | −1 | +5 | = | 11 |
| −9 | +7 | −2 | +11 | = | 7 |
| 7 | −8 | +2 | −3 | = | −2 |
| −1 | −8 | +1 | −7 | = | −15 |

The Maximum Subsequence Sum of $C^{3,6} = 35$. This is the same as the maximum of the individual sums of all the subarrays of $A$ that start at the 3rd column and end at the 6th column of $A$. For example, one such subarray is $SA_{2,3,5,6}$ whose sum is 7, and another subarray is $SA_{3,3,7,6}$ whose sum is 14. The Maximum Subsequence Sum of $C^{3,6}$ of 35 corresponds to the subarray $SA_{1,3,6,6}$.

If the preceding sums are computed for all possible $(g, h)$-pairs, then all the subarrays of the given array would have been taken into account, and the maximum, $M$, of all the Maximum Subsequence Sums, $MSQ^{gh}$, thus calculated gives the Maximum Subarray Sum, $MSA$.

To compute the sequences $C^{gh}$ efficiently, the rows can be preprocessed. Replacing each row by its prefix sums allows the sum of the elements of a row between any two columns $g$ and $h$ to be computed in $O(1)$ time.

The foregoing observations lead to the following parallel algorithm for finding the Maximum Subarray Sum of an $n \times n$ array:

**Algorithm:** Maximum Subarray Sum
**Input:** Array $A[1..n,\ 1..n]$ of numbers (positive and negative).
**Output:** Maximum Subarray Sum of $A$.
**Begin Algorithm**

1. Replace each row of $A$ by its prefix sums.
2. Add a column of zeroes as the zeroth column of $A$.
3. For all $1 \leq g \leq h \leq n$ do **in parallel**
   (a) Compute the sequence $C^{gh}$ :
   - For all $1 \leq i \leq n$ do **in parallel**
     $C^{gh}[i] := A[i][h] - A[i][g-1]$
   (b) Compute the maximum subsequence sum of $C^{gh}$ into $M_{gh}$.
3. Find the maximum of all $M_{gh}$, $1 \leq g \leq h \leq n$ into $MSA$.
4. Output $MSA$.

**End Algorithm**

**Complexity.**

Assuming the availabilty of $n^3/\log n$ processors for a CREW PRAM, and $n^3$ processors for an EREW PRAM:

Step 1 can be completed in $O(\log n)$ time. Step 2 can be completed in $O(1)$ time. Step 3(a) can be computed in $O(\log n)$ time assigning $n/\log n$ processors on a CREW PRAM, and $n$ processors on an EREW PRAM, respectively, to each $(g, h)$-pair. Step 3(b) can be computed in $O(\log n)$ time by assigning $n/\log n$ processors to each $C^{gh}$, by using the parallel algorithm for finding Maximum Subsequence Sum previously presented. Step 4 can be completed in at most $O(\log n)$ time. Thus, the total time is $O(\log n)$, with $n^3/\log n$ processors on a CREW PRAM, and $n^3$ processors on an EREW PRAM.

The algorithm can be modified in a straightforward way to find the actual maximum subarray instead of just its sum. This can be done by keeping track of the maximum subsequence of each $(g, h)$-pair.

## 4. Remarks

The one-dimensional and two-dimensional problems can be generalized to $d$ dimensions as follows:

> Given a $d$-dimensional cube with sides of size $n$, of positive and negative numbers, find the sub-$d$-cube that has the maximum sum out of all sub-$d$-cubes.

An approach similar to the one for the two-dimensional version given in the preceding sections holds for the $d$-dimensional version also. Consider any principal set (not a diagonal set) of $n$ $\overline{d-1}$-dimensional planes that are parallel to each other in the $d$-cube. In each of these planes, there exist $\binom{n}{2}^{(d-1)}$ sub-$\overline{d-1}$-planes. For each of the sub-planes, computing the maximum subsequence of the sequence formed by compressing the "tube" along the perpendicular axis of the planes enclosed by the sub-plane gives the maximum subsequence sum out of all sub-$d$-cubes that are enclosed by the tube. Thus, the Maximum Sub-$d$-cube Sum of a $d$-cube with each side of length $n$ can be computed in $O(\log n)$ time, using $n \times \binom{n}{2}^{d-1}$ processors on an EREW PRAM.

## References


1. J. Bentley, *Programming Pearls* (Addison-Wesley, 1989) 69–78.
2. U. Manber, *Introduction to Algorithms* (Addison-Wesley, 1989) 106–107.
3. J. Bates and R. Constable, Proofs as Programs, *ACM Trans. Program. Lang. Syst.* **7**, 1 (1985) 113–136.
4. J. Jeuring, The derivation of hierarchies of algorithms on matrices, *Constructing Programs from Specifications* (North-Holland, 1991) 9–32.
5. D. R. Smith, Applications of a strategy for designing divide-and-conquer algorithms, *Sci. Comput. Program.* **8** (1987) 213–229.
6. J. Jájá, *An Introduction to Parallel Algorithms* (Addison-Wesley, 1992).
7. R. E. Ladner and M. J. Fischer, Parallel prefix computation, *JACM* **27**, 4 (1980) 831–838.