Introducing Parallel Prefix-Sums Algorithm and Its Applications in An Undergraduate Course

Sukhamay Kundu
Computer Science Department
Louisiana State University, Baton Rouge, LA 70803
kundu@csc.lsu.edu

Abstract—The parallel prefix-sums algorithm and its variations are often used as key substeps in more complex parallel computations. We have successfully introduced the parallel prefix-sum algorithm and its key ideas in a senior level undergraduate Computer Science course. To improve the understanding of these ideas, students were required to solve several problems using simple variations of the basic prefix-sums algorithm. They were also required to solve several non-trivial problems, including their 2-dimensional generalizations. We briefly discuss here four problems of this second category.

Index Terms—prefix-sums, parallel algorithm, applications

I. THE PREFIX-SUMS

Let \( x[0..(N-1)] \), in short \( x[i] \) or simply \( x \) when no confusion is likely, be an array of numbers. We call the sum \( s[i] = \sum_{i=0}^{j} x[i] \) the \( j \)th (inclusive) prefix-sum of \( x \). The prefix-sums of \( x = [3, 12, -4, 8, 2] \) then equals \( s = [3, 15, 11, 19, 21] \). Many computation problems involve prefix-sums in some form [1]-[4]. For example, if \( x[j] \) is the probability of \( \#(\text{heads}) = j \) in \( N-1 \) tosses of a coin, then \( s[j] \) is the probability of \( \#(\text{heads}) \leq j \). An optimal sequential algorithm to compute the prefix-sums is to let \( s[0] = x[0] \) and \( s[j] = s[j-1] + x[j] \) for \( 1 \leq j < N \). This takes \( O(N) \) time. A parallel algorithm for computing the prefix-sums takes \( O(\log N) \) time using \( O(N) \) CPU’s or computing agents [1]-[2].

We define prefix-mins of \( x \) to be the array \( m \), where \( m[j] = \min\{x[i]: 0 \leq i \leq j\} \). The prefix-maxs of \( x \) is defined similarly, and each of these can be computed in time \( O(\log N) \) as well using \( O(N) \) agents. In what follows, we use the convention \( z[-1] = 0 \) for any array \( z[0..(N-1)] \).

II. THE MAXIMUM CONSECUTIVE SUM PROBLEM: MCS

For a given \( x \), we want to find a subarray \( x[i..j] \) of \( x \) with maximum sum of items \( s(i,j) = \sum_{k=i}^{j} x[k] = s[j] - s[i-1] \), where \( s \) is the prefix-sums of \( x \). Let \( MCS(x) \) be the "leftmost" subarray \( x[i..j] \) having the maximum \( s(i,j) \), with smallest possible \( j \) and the largest \( i \leq j \) for that \( j \). Also, let \( \sigma MCS(x) \) be the sum of items in \( MCS(x) \). Let \( m \) be the prefix-mins of \( s \) and \( m[j] = s[j] - m[j-1] \), with \( m[0] = s[0] = x[0] \). Then, \( \sigma MCS(x) = \max\{s(i,j): i \leq j\} = \max\{s_m[j]: 0 \leq j < N\} \). Thus, \( \sigma MCS(x) \) can be computed in \( O(\log N) \) time. It is not hard to see that \( MCS(x) \) can also be computed in \( O(\log N) \) time.

Example 1. Fig. 1 shows an \( x \) and its associated \( s, m, \) and \( s_m \). Here, \( MCS(x) = x[4..7] = [11, -2, -6, 12] \) and \( \sigma MCS(x) = s[7] - m[6] = 10 - (-5) = 15 \). Fig. 1 also shows the array items \( indx[j] = \max\{i: 0 \leq i < j \text{ and } s[i] = m[j]\} \).

This is needed to find \( i \) in \( x[i..j] = MCS(x) \). We compute \( indx[j] \)'s as we compute \( m[j] \)'s. Note that \( j \) in \( x[i..j] = MCS(x) \) equals \( \min\{j': s_m[j'] = \sigma MCS(x)\} \).

\[
\begin{array}{cccccccccccc}
 s: & -2 & -1 & 2 & -5 & 6 & 4 & -2 & 10 & 7 & 6 \\
 s_m: & -2 & 1 & 4 & -3 & 11 & 9 & 3 & 15 & 12 & 11 \\
 indx: & 0 & 0 & 0 & 3 & 3 & 3 & 3 & 3 & 3 & 3
\end{array}
\]

Fig. 1. Illustration of the arrays \( s, m, s_m \) and \( indx \) for a given array \( x \).

III. LARGEST BLOCK OF CONSECUTIVE 1’S: LBO

Assume that each \( x[i] = 0 \) or 1, at least one \( x[i] = 0 \), and at least one \( x[i] = 1 \), i.e., \( 0 < s[N-1] < N \), where \( s \) is the prefix-sums of \( x \). We can thus test the above property by computing \( s \) in \( O(\log N) \) time. We want to find a largest length subarray \( x[i..j] \), \( i \leq j \), of \( x \) consisting of only 1’s. Let \( LBO(x) \) denote the "leftmost" such a subarray \( x[i..j] \), with \( j \) as small as possible and the largest \( i \leq j \) for that \( j \). Also, let \( \sigma LBO(x) \) denote length \( \sigma LBO(x) = j - i + 1 \) the sum of items in \( LBO(x) \). Thus, \( x = [1, 0, 1, 1, 1, 0, 1, 1, 1] \) gives \( LBO(x) = x[2..4] \) and \( \sigma LBO(x) = 3 \). If we write \( x' \) for the array obtained by replacing each 0 in \( x \) by \(-N \), then it is easy to see that \( MCS(x') = x'[i..j] \) if and only if \( LBO(x) = x[i..j] \) and hence \( \sigma MCS(x') = \sigma LBO(x) \). Using \( O(N) \) agents, we can determine \( x' \) from \( x \) in \( O(1) \) time and thus we can compute \( LBO(x) \) and \( \sigma LBO(x) \) in time \( O(\log N) \) using \( O(N) \) agents.

IV. TWO DIMENSIONAL MCS: MCS2

This is a generalization of the \( MCS \)-problem in §II to 2-dimensional arrays. For an \( M \times N \) matrix \( x \), \( 0 \leq r_1 \leq r_2 < M \), and \( 0 \leq c_1 \leq c_2 < N \), let \( x(r_1, r_2, c_1, c_2) \) be the submatrix of \( x \) consisting of rows \( r_1 \) to \( r_2 \) and columns \( c_1 \) to \( c_2 \). We want to find an \( x(r_1, r_2, c_1, c_2) \) such that its sum of items \( s(r_1, r_2, c_1, c_2) = \sum_{i=r_1}^{r_2} \sum_{j=c_1}^{c_2} x[i..j] \) is maximum. Let \( MCS2(x) \) denote an optimum submatrix of \( x \) and as usual we want to find an \( x(r_1, r_2, c_1, c_2) = MCS2(x) \) with the smallest \( r_2 \), the largest \( r_1 \) for that \( r_2 \), the smallest \( c_2 \) for that
We solve the MCS2 problem by reducing it to a set of 1-dimensional MCS problems by considering groups of consecutive rows of $x$ and then taking the best solution of those 1-dimensional MCS problems. For $0 \leq r_1 \leq r_2 < M$, let $x_{r_1, r_2}$ be the 1-dimensional array of column-wise sums of items in rows $r_1$ to $r_2$, i.e., $x_{r_1, r_2}[j] = \sum_{i=r_1}^{r_2} x[i,j]$. Clearly, $\sigma M C S (x_{r_1, r_2}) = \sigma M C S (x_{r_1, r_2})$ if and only if $\sigma M C S (x_{r_1, r_2}) = \max \{ \sigma M C S (x_{r_1, r'_2}) : 0 \leq r'_2 \leq r_2 < M \}$. To compute $M C S (x_{r_1, r_2})$, we simply choose smallest $r'_2$ and largest $r_1 \leq r_2$ such that $\sigma M C S (x_{r_1, r'_2}) = \sigma M C S (x_{r_1, r_2})$. We can efficiently compute all $x_{r_1, r_2}$ as follows. Let $y_c$ be the column $c$ of $x$, i.e., $y_c[i] = x[i, c]$ and let $s_c$ be the prefix-sums of $y_c$. Also, let $x'$ denote the $M \times N$ matrix whose columns are $y'_c = s_c$ and let $x''$ be the row $r$ of $x''$. Thus, $x''[j] = \sum_{i=0}^{r-1} x[i, j] = x_{0, r}[j]$ and hence $x_{r_1, r_2} = x''[j] - x''[j]$, where $x''[j] = 0$ by convention. See Fig. 2.

Using $O(MN)$ agents, we can compute in parallel all prefix-sums $s_c, 0 \leq c < N$, and hence the matrix $x''$ in $O(\log(M))$ time. For each fixed $r_1$, we can compute all $x_{r_1, r_2}$ in parallel in time $O(1)$ and then compute all the related $M C S (x_{r_1, r_2})$ and $\sigma M C S (x_{r_1, r_2})$ in time $O(\log(N))$ based on the results in §II. This gives a total time $O(M \log(N))$ as we vary $r_1$. Finally, the number of $(r_1, r_2)$-combinations is $M(M+1)/2 \leq MN$ and hence the best of all $\sigma M C S (x_{r_1, r_2})$ and its associated $M C S (x_{r_1, r_2})$ can be computed in time $O(\log(MN))$. This gives the total time $O(M \log(N))$ for $M C S (x)$ and $\sigma M C S (x)$ using $O(MN)$ agents.

\begin{center}
\begin{tabular}{c|c}
\hline
$r_1$ & $r_2$ & $x_{r_1, r_2}$ & $M C S (x_{r_1, r_2})$ & $\sigma M C S (x_{r_1, r_2})$
\hline
0 & 0 & $[1, 0, 2, 1]$ & 10 & 10
1 & 1 & $[3, 1, 4, -4]$ & 9 & 11
2 & 2 & $[4, 4, 5, 4]$ & 13 & 13
\hline
1 & 1 & $[2, 1, 2, -5]$ & 6 & 9
2 & 2 & $[3, 4, 3, -5]$ & 5 & 7
\hline
2 & 2 & $[1, 3, 1, 0]$ & 5 & 7
\hline
\end{tabular}
\end{center}

(iii) $\sigma M C S (x) = \max \{ \sigma M C S (x_{r_1, r_2}) : 0 \leq r_1 \leq r_2 \}$ and $M C S (x) = (0, 2, 0, 2)$ because $M C S (x_{0, 2}) = 0, 2, 0, 2$.

\section{V. An Image Processing Application of MCS2}

Fig. 3(i) shows an $8 \times 8$ black-and-white image. We want to find a largest size rectangular black area in such an image. The rows 3 to 6 and columns 1 to 3 gives a largest black rectangular area of size 12 in Fig. 3(i).

An $N \times N$ image can be represented by an $N \times N$ matrix $x$, where $x[i, j] = 1$ or 0 according as the pixel (unit square) in row $i$ and column $j$ is black or white. One way to solve this problem is by converting it to a 2-dimensional version of LBO problem in §V. Hence, we can call this problem LBO2. Let $x'$ be the matrix obtained by replacing each 0 in $x$ by $-N^2$. Assuming that at least one pixel in $x$ equals 1, a solution of $M C S (x')$ would not involve a 0-pixel of $x$. Thus, $M C S (x')$ gives a solution of LBO2($x$), i.e., a largest area black rectangle of $x$. This takes $O(N \log(N))$ time using $N^2$ agents based on the results in §IV.

\section{VI. Conclusion}

We have presented here four problems, involving the applications of the basic prefix-sums computation, that we successfully used for introducing parallel computation in a senior-level undergraduate Computer Science course. The MCS-problem is the core of these problems, and the students needed several hints to arrive at its solution. The students could easily solve the LBO-problem given the hint to convert it to the MCS-problem by modifying the original input array. The students could solve the MCS2-problem, which is a 2-dimensional generalization of the MCS-problem, by applying MCS to groups of rows of the input matrix. Finally, the image processing problem is a 2-dimensional generalization of the LBO problem and the students could easily solve it.

\begin{thebibliography}{9}


\end{thebibliography}