Optimal Parallel Prefix on Mesh Architectures

Ömer Eğecioğlu and *Ashok Srinivasan*
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

Algorithms for efficient implementation of computation of prefix products on mesh-connected processor arrays are presented. Assuming that an arithmetic operation takes unit time and communication/computation ratio for a single input item is τ, we show that the prefixes of n items can be computed in time $2\tau\sqrt{n} + O(\log n)$ on a square mesh with n processors. If n processors are configured as a disc with respect to the Manhattan metric, then the parallel time for the problem becomes $\sqrt{2}\tau\sqrt{n} + O(\sqrt{\tau}\sqrt[4]{n})$. We show that both of these algorithms are asymptotically optimal.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures – *parallel processors*; F.1.2 [**Computation by Abstract Devices**]: Modes of computation – *parallelism*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – *computations on discrete structures.*

Keywords: Parallel prefix, mesh-connected processor arrays, communication complexity.

1 Introduction

Given an ordered n-tuple $(x_1, x_2, \ldots, x_n)$ of elements of a set X closed under an associative binary operation $*$, the *prefix problem* is the computation of the partial products $= x_1 * x_2 * \cdots * x_i$ for $1 \leq i \leq n$. Problems of this nature arise in various settings including circuit design where $*$ is a simple boolean operation, and numerical problems where $*$ may be floating-point matrix multiplication. As examples, parallel algorithms for computing the Newton and Hermite interpolating polynomials make use of parallel prefix algorithms where the x_i's are floating-point numbers and $*$ is a floating-point addition or multiplication [5]. Solution of kth order linear recurrences can be obtained by a parallel prefix algorithm where $*$ is $k \times k$ matrix multiplication [10, 9, 8]. Tridiagonal systems can be solved with Stone's recursive doubling algorithm by computing the prefixes of 2×2 matrices [17].

Parallel prefix circuits have applications in the design of optimal-area adders [2] and the simulation of sequential circuits by combinational circuits [12]. Fich's paper [7] contains a review of the literature on parallel prefix circuits along with further applications. In a parallel prefix circuit, the concern is to reduce the depth and the size of the circuit. Size and depth bounds and trade-offs for prefix circuits appear in [7] and [16].

Prefixes of n elements can be computed trivially in $n - 1$ steps sequentially where at each step a single $*$ operation is performed. There are several parallel prefix algorithms [12, 2, 11, 13], given either in the arithmetic circuit or PRAM model of parallel computation. Asynchronous algorithms [14] and implementation on various ensemble architectures [15, 3, 4, 1, 6] have also been considered.

We assume that we are given n identical processors with a routing mechanism to send an operand from one processor to any other processor. An *arithmetic step* (or a *multiplication step*) is defined as the time required to perform a $*$ operation by a single processor. A *routing step* is the time required to transfer an operand from one processor to a neighboring one. We assume that a routing step requires τ units of time and an arithmetic step takes unit time. In this paper, we focus on the performance of parallel prefix algorithms on mesh-connected processor arrays.

In §2 we derive a lower bound for the time complexity of prefix computations on a mesh-connected system with the assumption that initially exactly one data item is assigned to each processor. We first present a suboptimal parallel prefix algorithm (Algorithm A) on a rectangular mesh in §3 to motivate the rest of the paper. In §4, we describe an improved algorithm (Algorithm B) for prefix computations on a rectangular mesh, which is asymptotically optimal. Subsequently in §5 we construct an algorithm (Algorithm C) for the prefix problem on the disc which uses Algorithm B as a subprocedure and which is asymptotically optimal. This is followed by conclusions in §6.

2 Lower bounds

Consider the infinite grid of lattice points in the plane where a point (a_1, b_1) is connected to a point (a_2, b_2) iff

$$|a_1 - a_2| + |b_1 - b_2| = 1 \ .$$

The Manhattan metric between the points $P = (a_1, b_1)$ and $Q = (a_2, b_2)$ is given by

$$d(P, Q) = |a_1 - a_2| + |b_1 - b_2| \ . \tag{1}$$

We say that a multiprocessor network with N processors forms a *mesh* if the processors are arranged as a connected, induced subgraph of the lattice grid. The mesh is *rectangular* if it consists of all lattice points in some $a \times b$ rectangle of lattice points as in Figure 1 (a). Figure 1 (b) shows *disc* of radius $r = 2$. This latter mesh can be identified with the set of points P with $d(P, Q) \leq r$ for some fixed center point Q.

Figure 1: Two meshes: (a): 3×4 rectangle (b) disc of radius 2.

Consider the list $X = (x_1, x_2, \ldots, x_n)$ and suppose $*$ is an associative binary operation on X. For $1 \leq i \leq j \leq n$, we denote the product $x_i * x_{i+1} * \cdots * x_j$ by the symbol $x[i:j]$. The prefix problem on

X is the computation of the initial products

$$\begin{aligned} x[1:1] &= x_1 \\ x[1:2] &= x_1 * x_2 \\ &\vdots \\ x[1:n] &= x_1 * x_2 * \cdots * x_n \ . \end{aligned}$$

We linearly order the prefix products by putting $x[1:i]$ to be smaller than $x[1:j]$ when $i < j$.

A lower bound for the time required for the computation of the prefix products of X on a mesh with n processors is

$$\log n + \tau c(n) \tag{2}$$

where $c(n)$ is a lower bound on the number of routing steps. This can be proved by showing that (2) is a lower bound for the computation of the single term $x[1:n]$. We assume that

1. initially each processor is assigned a single item x_i,
2. the communication distance between two processors P and Q in the mesh is $d(P,Q)$.

A lower bound for $c(n)$ is given by

$$\min_{M} \max_{P,Q} d(P,Q) \tag{3}$$

where the minimum is taken over all possible meshes of n processors. In other words $c(n)$ is the minimum diameter of all meshes with n points. We observe that the value of (3) is the diameter of the smallest disc (with respect to the Manhattan metric) which contains all n processors. Next we determine the radius of this smallest disc as a function of n.

Lemma 1 *The number of lattice points at a distance $k > 0$ from a given lattice point is $4k$.*

Proof Without loss of generality, we can assume that the lattice point we are considering is $Q = (0,0)$. Point $P = (a_1, b_1)$ is at a distance $|a_1| + |b_1|$ from Q. Therefore, the number of points k away from Q is equal to four times the number of solutions of

$$a_1 + b_1 = k, \quad a_1 \geq 0, b_1 > 0 \ .$$

Since this number is k, the total number of points at distance exactly k from Q is $4k$. □

It follows that in any mesh, the total number of processors at a distance at most r from Q is bounded from above by the number of lattice points in a disc of radius r. By Lemma 1 this number is

$$1 + \sum_{k=1}^{r} 4k = 1 + 2r(r+1) \ . \tag{4}$$

For example, the disc of radius 2 in Figure 1 (b) contains $1 + 2 * 2 * (2+1) = 13$ lattice points. If r is the radius of the smallest disc that contains n processors, then from (4)

$$n = 1 + 2r(r+1) \ \Rightarrow \ r = \frac{1}{2}(-1 + \sqrt{2n-1}) \ .$$

For $n \geq 1$,

$$\frac{1}{2}\sqrt{2n} - 1 \ \leq \ r \ \leq \ \sqrt{\frac{n}{2}} \ , \qquad (5)$$

and therefore

$$c(n) \ \geq \ \sqrt{2n} - 2 \ \ .$$

Consequently a lower bound on the communication time required to compute prefixes of n items on any mesh with n processors is $\tau(\sqrt{2n} - 2)$. Since $\log n$ is a lower bound for the computation of $x[1:n]$ we obtain that a lower bound for the computation of the prefixes of X on any mesh-connected system is

$$\tau(\sqrt{2n} - 2) + \log n \ .$$

When we have an $a \times b$ rectangular mesh with a total of n processors, the lower bound for $c(n)$ becomes $2\sqrt{n} - 2$, which is achieved for $a = b = \sqrt{n}$. Thus

Lemma 2 *A lower bound for the computation of prefixes of* $(x_1, x_2, \ldots, x_n)$ *on*

(a) an arbitrary mesh with n processors is $\sqrt{2}\tau\sqrt{n} + \log n - 2\tau$,

(b) on a rectangular mesh with n processors is $2\tau\sqrt{n} + \log n - 2\tau$.

3 A simple algorithm

In this section we describe a simple but suboptimal algorithm for the prefix problem on a rectangular mesh. For simplicity, we describe the algorithm on the $\sqrt{n} \times \sqrt{n}$ square mesh. The generalization to an arbitrary rectangular mesh is a straightforward extension of this case.

We number the processors so that the processor at the lower left corner is $(1,1)$, and the one at the top right is $(\sqrt{n}, \sqrt{n})$. The initial assignment of $x_1, x_2, \ldots, x_n$ to the processors is as follows: processor (i,j) in row i and column j is assigned the item $x_{(j-1)\sqrt{n}+i}$. An example is shown in Figure 2 (a) for $n = 16$.

For $i < j$, each processor in column i will calculate a smaller prefix than any processor in column j. Also, if $i < j$, then for any given column, the processor at row i will calculate a smaller prefix than the processor at row j. This means that at the end of the algorithm, processor (i,j) will contain the prefix $x[1 : (j-1)\sqrt{n} + i]$. The algorithm (called Algorithm A) consists of the following four steps:

ALGORITHM A

Input: Items $X = (x_1, x_2, \ldots, x_n)$. Initially processor (i,j) is assigned $x_{(j-1)\sqrt{n}+i}$.
Output: Prefix products of X. Processor (i,j) contains $x[1 : (j-1)\sqrt{n} + i]$.

1. Calculate the partial prefixes over each column using the $\log n$ arithmetic step algorithm (given as procedure prefix.independent) in Kruskal, Rudolph, and Snir [11]. At the end of this step processor (i,j) contains the value $x[(j-1)\sqrt{n} + 1 : (j-1)\sqrt{n} + i]$. The contents of the processors after Step 1 is executed is shown in Figure 2 (b). The number of arithmetic steps required is $\log \sqrt{n}$, and the number of routing steps required is

$$\sum_{k=0}^{\log(\sqrt{n})-1} 2^k = \sqrt{n} - 1 \ \ .$$

Therefore, the total time required for Step 1 is

$$\tau(\sqrt{n}-1)+\log\sqrt{n} \quad .$$

2. Calculate the partial prefixes for the top row using the $\log n$ algorithm as in Step 1. At the end of this step, processor $(\sqrt{n}, j)$ will have the value $x[1 : j\sqrt{n}]$. The time required is the same as in Step 1:
$$\tau(\sqrt{n}-1)+\log\sqrt{n} \quad .$$
The contents of the processors immediately after the completion of Step 2 are shown in Figure 2 (c).

3. The value of $x[1 : j\sqrt{n}]$, now available in $(\sqrt{n}, j)$, is broadcast to each processor in column $j+1$ for $1 \leq j < \sqrt{n}$. The dotted lines in Figure 2 (d) show these parallel routing steps. The time required required is
$$\tau\sqrt{n} \quad .$$

4. Processor (i, j) multiplies the value $x[1 : (j-1)\sqrt{n}]$ received in Step 3, along with the value $x[(j-1)\sqrt{n}+1 : (j-1)\sqrt{n}+i]$ computed in Step 1, to compute its target product $x[1 : (j-1)\sqrt{n}+i]$. Since this requires a single parallel multiplication, the time required is
$$1 \quad .$$
The final contents of the processors are shown in Figure 2 (d).

END ALGORITHM A

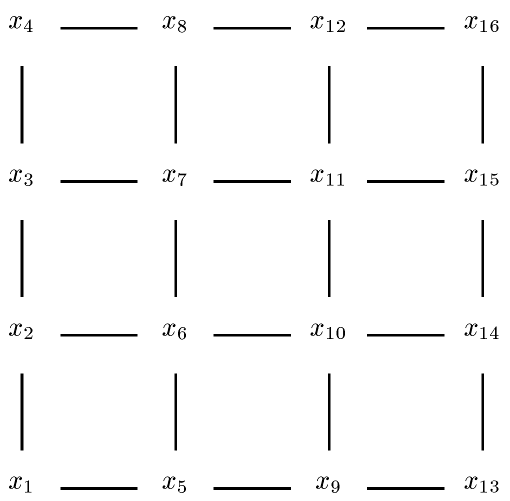
The total time T_A required by Algorithm A is the sum of the time required for each step. This is found to be

$$T_A = 3\tau\sqrt{n} + \log n - 2\tau + 1 \quad .$$

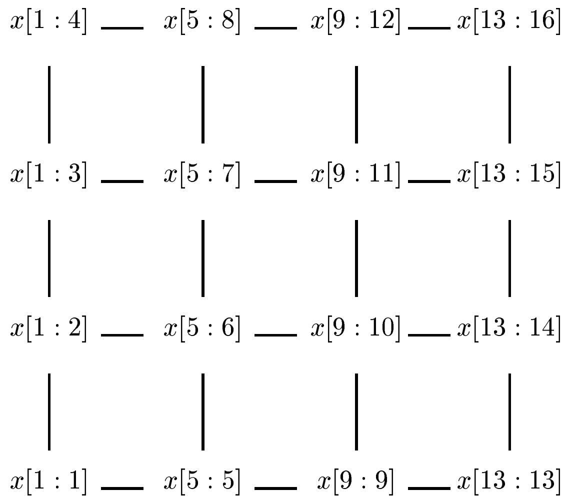
The parallel time T_A exceeds the lower bound given in lemma 2 (b) by $\tau\sqrt{n}+1$. In the next section we present an improved algorithm for the square mesh which exceeds the lower bound only by a constant amount.

4 An optimal algorithm for the square

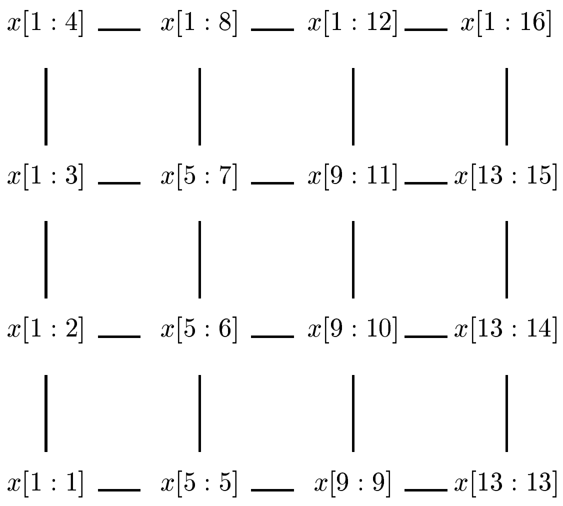
The algorithm we present in this section is based on Algorithm A, but has near optimal performance achieved by reducing the communication complexity of Algorithm A. Here we imagine that the $\sqrt{n}\times\sqrt{n}$ square as made up of two $\frac{1}{2}\sqrt{n}\times\sqrt{n}$ rectangles, an upper rectangle, and a lower rectangle, as separated by a dotted line in Figure 3 (a) for $n = 16$. The idea is to limit the communication between the upper and the lower rectangles to communicating across the horizontal boundary only. As before, for $i < j$, the prefix computed in column i will be smaller than the one computed in column j. In the lower rectangle, the prefixes will be computed as in Algorithm A; however in the upper rectangle, the computation will proceed in the reverse order, so that the large products will be computed closer to the boundary between the two rectangles.


x_4 x_8 x_{12} x_{16}
x_3 x_7 x_{11} x_{15}
x_2 x_6 x_{10} x_{14}
x_1 x_5 x_9 x_{13}

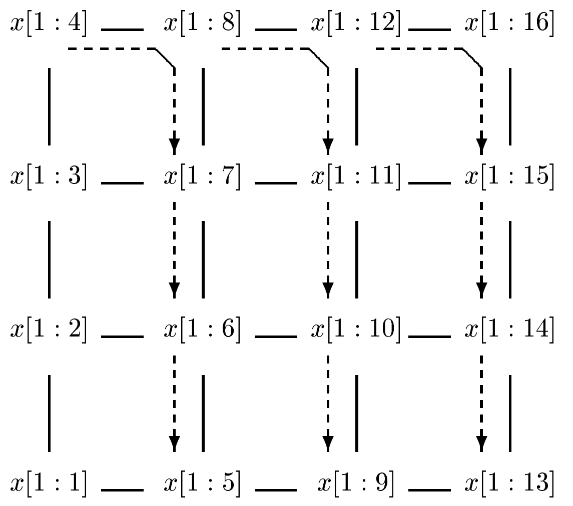
(a) Initial assignment.


$x[1:4]$ $x[5:8]$ $x[9:12]$ $x[13:16]$
$x[1:3]$ $x[5:7]$ $x[9:11]$ $x[13:15]$
$x[1:2]$ $x[5:6]$ $x[9:10]$ $x[13:14]$
$x[1:1]$ $x[5:5]$ $x[9:9]$ $x[13:13]$

(b) Step 1:
Parallel columnwise prefixes.


$x[1:4]$ $x[1:8]$ $x[1:12]$ $x[1:16]$
$x[1:3]$ $x[5:7]$ $x[9:11]$ $x[13:15]$
$x[1:2]$ $x[5:6]$ $x[9:10]$ $x[13:14]$
$x[1:1]$ $x[5:5]$ $x[9:9]$ $x[13:13]$

(c) Step 2:
Top row prefixes.


$x[1:4]$ $x[1:8]$ $x[1:12]$ $x[1:16]$
$x[1:3]$ $x[1:7]$ $x[1:11]$ $x[1:15]$
$x[1:2]$ $x[1:6]$ $x[1:10]$ $x[1:14]$
$x[1:1]$ $x[1:5]$ $x[1:9]$ $x[1:13]$

(d) Steps 3 & 4:
Column broadcast and single multiplication.

Figure 2: (a)-(d): Algorithm A.

The initial assignment of values are as shown in Figure 3 (a): processor (i,j) in row i and column j is assigned the value $x_{(j-1)\sqrt{n}+i}$ if $i \leq \sqrt{n}/2$, i.e., the indices increase as we go up a column in the lower rectangle; processor (i,j) has the value $x_{j\sqrt{n}+\sqrt{n}/2+1-i}$ if $i > \sqrt{n}/2$, i.e., the indices increase as we go down a column in the upper rectangle. The algorithm (which is called Algorithm B) consists of seven basic steps as follows:

ALGORITHM B

Input: Items $X = (x_1, x_2, \ldots, x_n)$. Initially processor (i,j) is assigned

$$\begin{array}{rll} x_{(j-1)\sqrt{n}+i} & \text{for} & i \leq \sqrt{n}/2, \\ x_{j\sqrt{n}+\sqrt{n}/2+1-i} & \text{for} & i > \sqrt{n}/2 \ . \end{array}$$

Output: Prefix products of X. Processor (i,j) contains

$$\begin{array}{rll} x[1:(j-1)\sqrt{n}+i] & \text{for} & i \leq \sqrt{n}/2 \ , \\ x[1:j\sqrt{n}+\sqrt{n}/2+1-i] & \text{for} & i > \sqrt{n}/2 \ . \end{array}$$

1. Calculate the partial prefixes over each half-column in parallel, using the same method as Step 1 of Algorithm A. After this step, processor (i,j) has the partial product

$$\begin{array}{rl} x[(j-1)\sqrt{n}+1:(j-1)\sqrt{n}+i], & \text{if } \ i \leq \sqrt{n}/2 \\ x[(j-1)\sqrt{n}+\sqrt{n}/2+1:j\sqrt{n}+\sqrt{n}/2+1-i], & \text{if } \ i > \sqrt{n}/2 \end{array}$$

   as indicated in Figure 3 (b). Similar to Step (1) of Algorithm A, the time taken for this computation is

$$\tau(\frac{\sqrt{n}}{2}-1)+\log(\sqrt{n}/2) \quad .$$

2. Send the values computed in Step (1) from

   processor $(\sqrt{n}/2, j)$ to processor $(\sqrt{n}/2+1, j)$ directly for $1 \leq j \leq \sqrt{n}$,

   processor $(\sqrt{n}/2+1, j)$ to $(\sqrt{n}/2, j+1)$ through processor $(\sqrt{n}/2, j)$ in two steps for $1 \leq j < \sqrt{n}$.

   The time required for this step of the algorithm is:

$$2\tau \quad .$$

   The communication pattern used by the two central rows in Step 2 is indicated as dotted lines in Figure 3 (c).

3. By a single multiplication,

   processor $(\sqrt{n}/2, j)$ calculates

$$x[(j-2)\sqrt{n}+\sqrt{n}/2+1:(j-1)\sqrt{n}+\sqrt{n}/2] \ \text{ for } \ 1 \leq j \leq \sqrt{n} \ ,$$

   processor $(\sqrt{n}/2+1, j)$ calculates $x[(j-1)\sqrt{n}+1:j\sqrt{n}]$ for $1 \leq j < \sqrt{n}$.

After Step 3, the processors contain the partial products in Figure 3 (d). The time required is

$$1 \quad .$$

4. Now using the $\log n$ algorithm of Step 1 of Algorithm A, processors in row $\sqrt{n}/2+1$ calculate their partial prefix products across their row. In the same way, the processors in row $\sqrt{n}/2$ calculate their partial prefix products, arriving at the configuration in Figure 3 (e). At the end of this step,

   processor $(\sqrt{n}/2, j)$ has $x[1:(j-1)\sqrt{n}+\sqrt{n}/2]$, and
   processor $(\sqrt{n}/2+1, j)$ has $x[1:j\sqrt{n}]$.

   The time required for this step is

$$\tau(\sqrt{n}-1)+\log\sqrt{n} \quad .$$

5. Send the values computed in the previous step from

   processor $(\sqrt{n}/2, j)$ to processor $(\sqrt{n}/2+1, j)$ directly, and from
   processor $(\sqrt{n}/2+1, j)$ to $(\sqrt{n}/2, j+1)$ through processor $(\sqrt{n}, j)$ for $1 \le j < \sqrt{n}$.

   These routing lines are shown in Figure 3 (f). The time required is

$$2\tau \quad .$$

6. The processors $(\sqrt{n}/2, j)$ and $(\sqrt{n}/2+1, j)$ now broadcast the values received in the previous step to all of the processors in their side of column j. The parallel time required is

$$\tau(\sqrt{n}/2-1) \quad .$$

7. Each processor calculates its target prefix product by at most one more multiplication using the values received at the end of Step 5. The time required is

$$1 \quad .$$

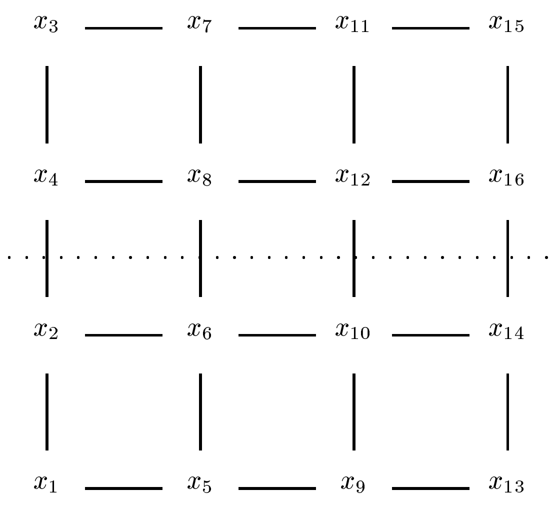
   The final configuration of values is shown in Figure 3 (f).

END ALGORITHM B

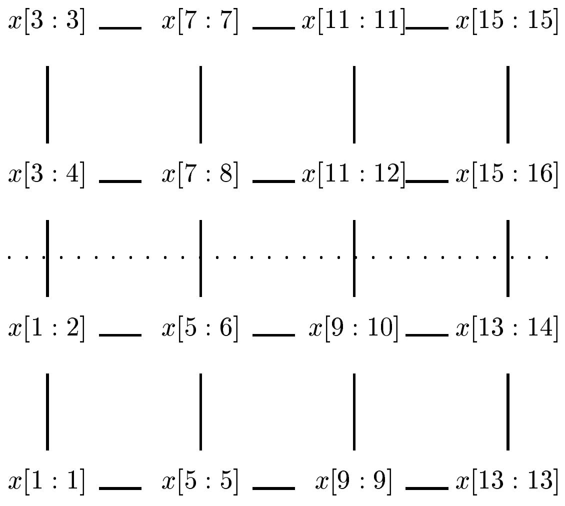
Let T_B denote the total time required by algorithm B. T_B is the sum of the seven individual steps of the algorithm, which is found to be

$$T_B = 2\tau\sqrt{n}+\log n+\tau+1 \quad . \tag{6}$$

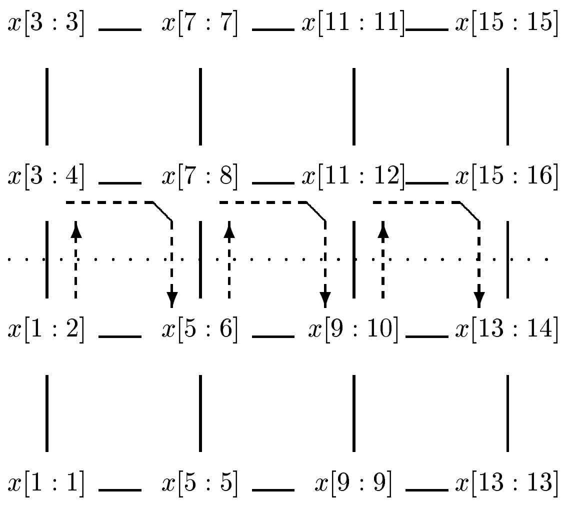
By lemma 2 (b), a lower bound on the time required for the prefix computation on a rectangular mesh is $2\tau\sqrt{n}+\log n-2\tau$. From the expression in (6) for T_B we see that the running time of Algorithm B exceeds the lower bound only by the constant amount $3\tau+1$.


x_3 x_7 x_{11} x_{15}
x_4 x_8 x_{12} x_{16}
x_2 x_6 x_{10} x_{14}
x_1 x_5 x_9 x_{13}

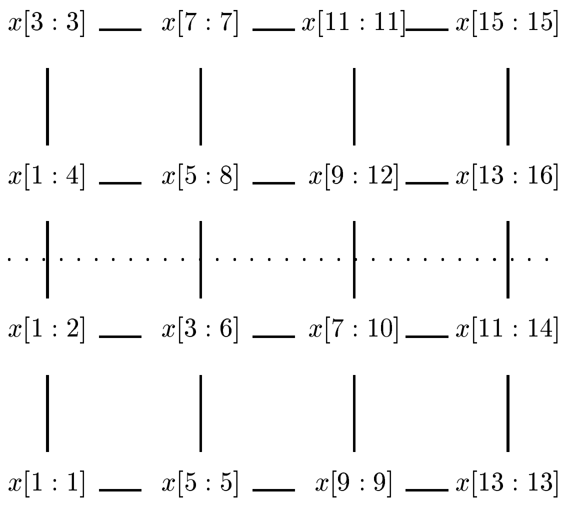
(a) Initial assignment.


$x[3:3]$ $x[7:7]$ $x[11:11]$ $x[15:15]$
$x[3:4]$ $x[7:8]$ $x[11:12]$ $x[15:16]$
$x[1:2]$ $x[5:6]$ $x[9:10]$ $x[13:14]$
$x[1:1]$ $x[5:5]$ $x[9:9]$ $x[13:13]$

(b) Step 1:
Parallel columnwise prefixes.

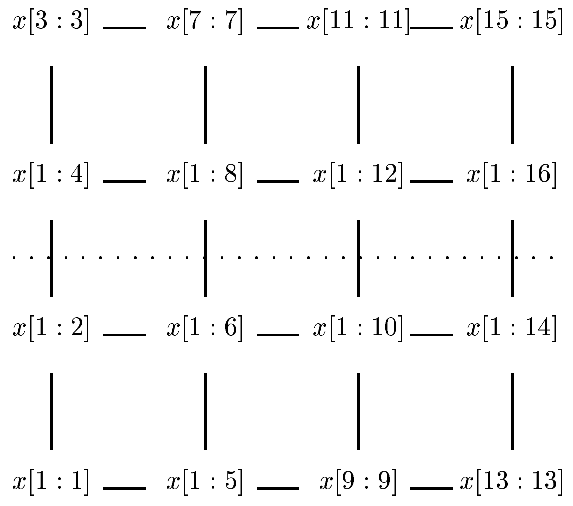

$x[3:3]$ $x[7:7]$ $x[11:11]$ $x[15:15]$
$x[3:4]$ $x[7:8]$ $x[11:12]$ $x[15:16]$
$x[1:2]$ $x[5:6]$ $x[9:10]$ $x[13:14]$
$x[1:1]$ $x[5:5]$ $x[9:9]$ $x[13:13]$

(c) Step 2:
Communication pattern.

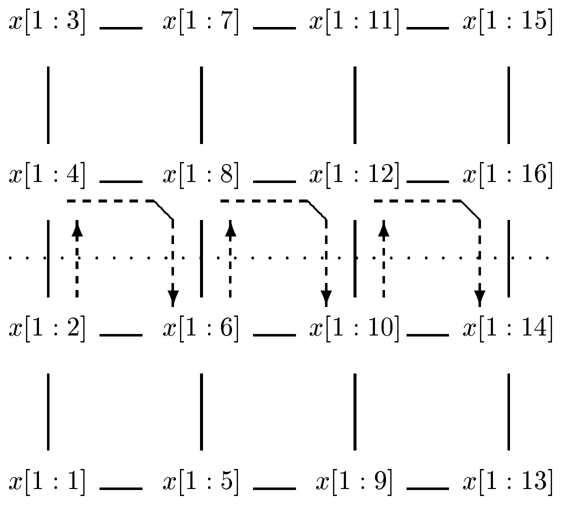

$x[3:3]$ $x[7:7]$ $x[11:11]$ $x[15:15]$
$x[1:4]$ $x[5:8]$ $x[9:12]$ $x[13:16]$
$x[1:2]$ $x[3:6]$ $x[7:10]$ $x[11:14]$
$x[1:1]$ $x[5:5]$ $x[9:9]$ $x[13:13]$

(d) Step 3:
Single multiplication.

Figure 3: (a)-(d): Algorithm B.



(e) Step 4:
Parallel prefix along central rows.



(f) Steps 5, 6 & 7:
Communication pattern and final configuration.

Figure 3: (e)-(f): Algorithm B.

5 Prefix on a disc

In this section we consider the prefix computation on mesh of processors arranged as a disc with respect to the metric in (1), as in Figure 1 (b). For simplicity, we assume that n is of the form $n = 1+2r(r+1)$.

The algorithm is as follows: we label the processors by their lattice coordinates, the center being $O = (0,0)$. The disc is conceptually divided into four quadrants Q_1, Q_2, Q_3, and Q_4 as shown in Figure 4 (a), where

$$\begin{aligned}
Q_1 &= \{ \text{ processors } (s,t) \mid s \leq 0,\ t \geq 0\},\\
Q_2 &= \{ \text{ processors } (s,t) \mid s > 0,\ t \geq 0\},\\
Q_3 &= \{ \text{ processors } (s,t) \mid s > 0,\ t < 0\},\\
Q_4 &= \{ \text{ processors } (s,t) \mid s \leq 0,\ t < 0\}.
\end{aligned}$$

For any positive integer L, we can further divide the quadrants into cells of size $L \times L$ as shown in Figure 4 (b) for $L = 2$. For example, in Q_1 processors (s,t) with $-L < s \leq 0$, $0 \leq t < L$ are in one cell denoted by $C_{0,0}$, processors (s,t) with $-2L < s \leq -L$, $L \leq t < 2L$ will be in one cell denoted by $C_{-1,1}$. In general we label the cells in Q_1 as $C_{i,j}$ with $-r/L \leq i \leq 0$ and $0 \leq j \leq r/L$ where $C_{i,j}$ consists of all processors (s,t) with $-(i+1)L < s \leq -iL$ and $jL \leq t \leq (j+1)L$. A similar notation is used in the other quadrants. For instance the cells in Q_2 are labeled $C_{i,j}$ with $0 < i \leq \lceil r/L \rceil$, and $0 \leq j \leq (r-1)/L$, with $C_{i,j}$ consisting of all processors (s,t) with $(i-1)L < s \leq iL$, and $jL \leq t \leq (j+1)L$. See Figure 4 (b) for an example of this decomposition with $r = 4$ and $L = 2$.

The assignment of initial values to the processors is as follows: first of all if $i < j$ with $x_u \in Q_i$ and $x_v \in Q_j$, then $u < v$. Within a quadrant, the items are assigned to cells so that $x_u \in C_{i,j}$ and $x_v \in C_{k,l}$ implies $u < v$, whenever $|i| > |k|$, or $i = k$, and $|j| > |l|$. Thus essentially, in each quadrant the columns of cells farther from the origin are assigned items with smaller indices, and within a given

Figure 4: (a) Quadrants Q_1, Q_2, Q_3, Q_4 on a disc D with center O and radius $r = 4$, (b) Decomposition of D into cells of size $L = 2$.

column in a quadrant, the indices of the items increase as we get closer to the horizontal axis. This bulk assignment is shown in Figure 5 (c) for $r = 7$ and $L = 4$, in which the of the list shown in each cell is the set of items that will be assigned to processors in the cell.

Within a cell, the items are assigned as in algorithm B, but with two provisos:

1. The highest indexed item that appears in a cell is assigned to the processor on the vertical side of the cell closer to the center. Thus in Q_1 and Q_4 the items are assigned exactly as in algorithm B, whereas in Q_2 and Q_3 the assigned values are reversed along the vertical axis of each cell. Consequently, in quadrants 2 and 3, the largest indexed item is on the left hand side of the cell.

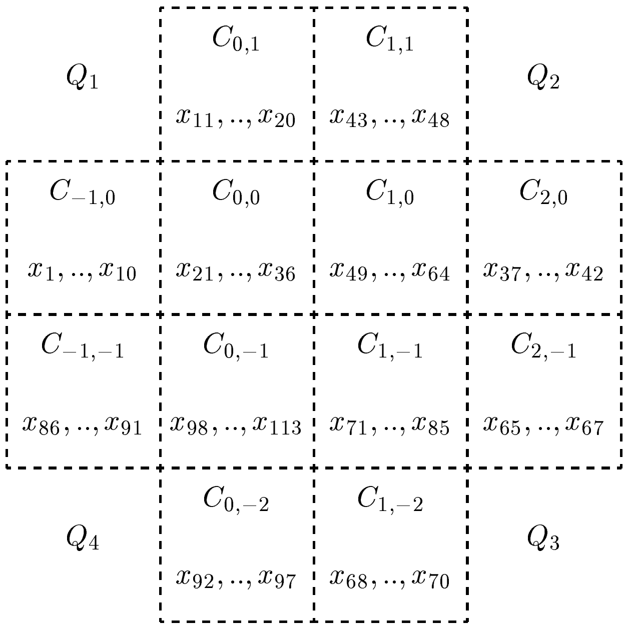
2. The cell may not be complete. In this case we respect the order given by the assignment of algorithm B in our assignment to the existing processors in the incomplete cell.

For $r = 7$ and $L = 4$, the assignments of items to individual processors in each of the cells $C_{0,1}$, $C_{0,0}$, and $C_{1,-1}$ are shown in Figure 6 (a), (b), and (c), respectively.

ALGORITHM C

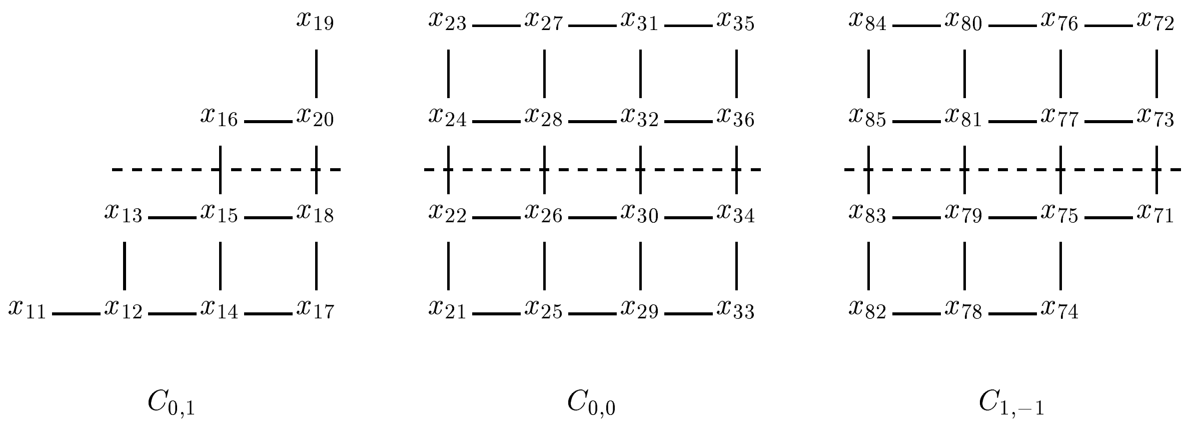
Input: Items $X = (x_1, x_2, \ldots, x_n)$. Initially processor (s, t) is assigned item x_i, as described above. We assume that $n = 1 + 2r(r + 1)$ for some r.
Output: Prefix products of X.

1. Calculate in parallel the prefixes of the assigned items in each cell by using Algorithm B given in the previous section. Let $c_{i,j}$ denote the processor in cell $C_{i,j}$ that is closest to the origin and let $p_{i,j}$ be the product of all of the items assigned to processors in $C_{i,j}$. The quantity $p_{i,j}$ is



Q_1
Q_2
Q_3
Q_4
$C_{0,1}$ $x_{11},..,x_{20}$
$C_{1,1}$ $x_{43},..,x_{48}$
$C_{-1,0}$ $x_1,..,x_{10}$
$C_{0,0}$ $x_{21},..,x_{36}$
$C_{1,0}$ $x_{49},..,x_{64}$
$C_{2,0}$ $x_{37},..,x_{42}$
$C_{-1,-1}$ $x_{86},..,x_{91}$
$C_{0,-1}$ $x_{98},..,x_{113}$
$C_{1,-1}$ $x_{71},..,x_{85}$
$C_{2,-1}$ $x_{65},..,x_{67}$
$C_{0,-2}$ $x_{92},..,x_{97}$
$C_{1,-2}$ $x_{68},..,x_{70}$

Figure 5: Bulk distribution of items to cells: $r = 7$, $n = 113$, and $L = 4$.



x_{19}
x_{16} x_{20}
x_{13} x_{15} x_{18}
x_{11} x_{12} x_{14} x_{17}
$C_{0,1}$
x_{23} x_{27} x_{31} x_{35}
x_{24} x_{28} x_{32} x_{36}
x_{22} x_{26} x_{30} x_{34}
x_{21} x_{25} x_{29} x_{33}
$C_{0,0}$
x_{84} x_{80} x_{76} x_{72}
x_{85} x_{81} x_{77} x_{73}
x_{83} x_{79} x_{75} x_{71}
x_{82} x_{78} x_{74}
$C_{1,-1}$

Figure 6: Assignment of items to processors in cells $C_{0,1}$, $C_{0,0}$, and $C_{1,-1}$: $r = 7$, $L = 4$.

the largest prefix computed in $C_{i,j}$ at the completion of Algorithm B. Send $p_{i,j}$ to processor $c_{i,j}$. Figure 7 (a) and Figure 7 (b) show the resulting values in the processors in cell $C_{0,1}$, and the values in each of the special processors $c_{i,j}$ over the entire mesh after this step, respectively.

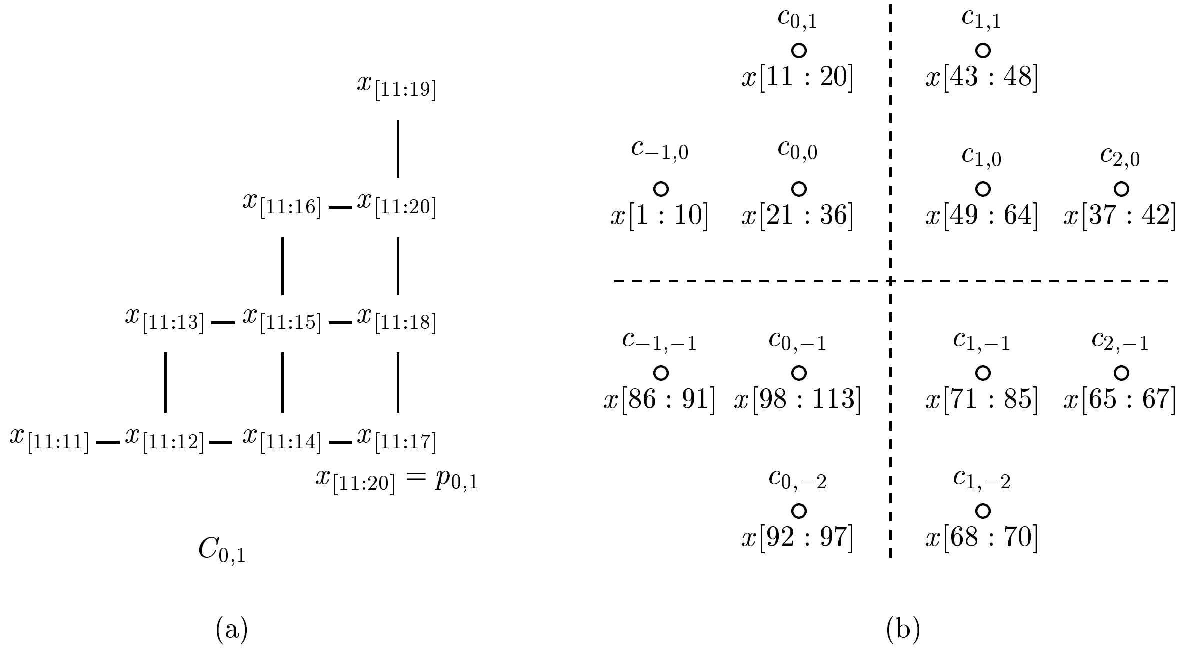


Figure 7: After Step 1 (a): Prefixes in $C_{0,1}$, (b) Contents of processors $c_{i,j}$.

The time required for this computation is the time (6) required by Algorithm B on an $L \times L$ square plus $\tau L/2$ units of communication time required to send $p_{i,j}$ to processor $c_{i,j}$. The total is computed to be

$$\frac{5}{2}\tau L + 2\log L + \tau + 1$$

2. In this step, prefixes of the $p_{i,j}$'s themselves along each column of cells in each quadrant, as well as the product of all $p_{i,j}$ in a given quadrant will be computed at the special processors $c_{i,j}$. Let q_1, q_2, q_3, q_4 denote the product of all $p_{i,j}$ in the quadrants Q_1, Q_2, Q_3, Q_4, respectively. In each quadrant, the computation in this step of the algorithm is independent of the values computed in the other quadrants, and is performed in parallel. Denote the prefix computed by $c_{i,j}$ during this step by $a_{i,j}$. Initially $a_{i,j} = p_{i,j}$ for every processor $c_{i,j}$. We will give a high level description of this step of the algorithm on quadrant Q_1. A sequence of operations similar to the code in Figure 8 are executed in quadrants Q_2, Q_3, and Q_4 also. In these quadrants indices of the code Figure 8 need to be suitably modified.

   In Q_1 we perform the sequence of operations given in Figure 8:

   An example of this step with $r/L = 2$ is shown in parts (a), (b), and (c) of Figure 9. In (a), the initial send operation together with the computation of $a_{-1,0} := p_{-1,1} * p_{-1,0}$ and $a_{0,1} := p_{0,1} * p_{0,2}$ are shown. The next two figures in parts (b) and (c) correspond to the execution of the **for** loop in Figure 8 with $k = 1$ and $k = 0$, respectively. In (b), computation of $a_{-1,0} := a_{-2,0} * a_{-1,0}$ and

```
for  each $i, j$ with $(j - i = r/L)$ and $(j > 0)$ parallel do                /* initial step */
  begin
    send $a_{i,j}$ from $c_{i,j}$ to $c_{i,j-1}$;                              /* send down parallel */
    $a_{i,j-1} := p_{i,j-1} * a_{i,j}$;                                        /* multiply parallel */
  end

for $k = r/L - 1$ downto  0 do                                                 /* loop */
  begin
    for  each $i, j$ with $(j - i = k)$ parallel do
      begin
        if  $j > 0$ then send $a_{i,j}$ from $c_{i,j}$ to $c_{i,j-1}$            /* send down parallel */
          else send $a_{-k-1,0}$ from $c_{-k-1,0}$ to $c_{-k,0}$;    /* send right by one processor */

        if  $j > 0$ then  $a_{i,j-1} := p_{i,j-1} * a_{i,j}$;                   /* multiply parallel */
          else $a_{-k,0} := a_{-k-1,0} * a_{-k,0}$;                  /* multiply by one processor */
      end
  end
```

Figure 8: Step 2 of Algorithm C: Code for Q_1.

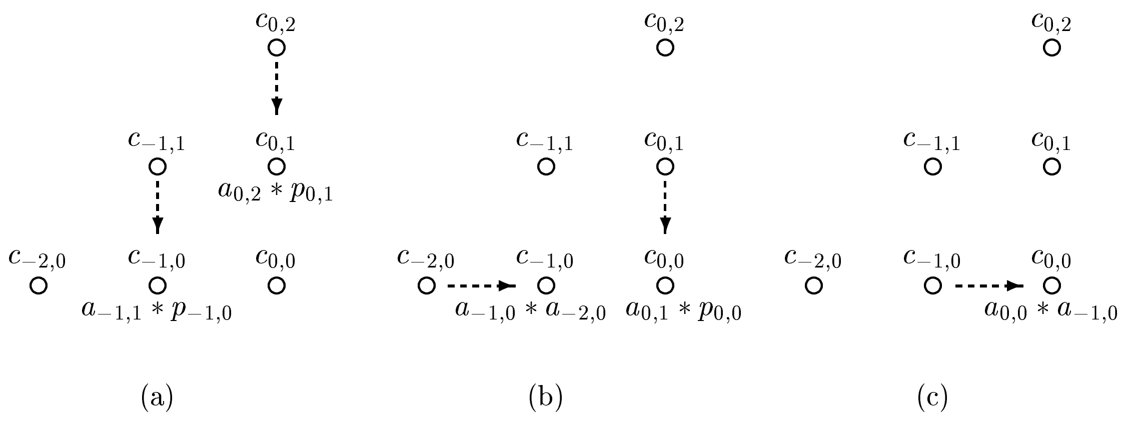


Figure 9: Execution of Step 2 in Q_1 with $r/L = 2$: (a) initial, (b) $k = 1$, (c) $k = 0$.

$a_{0,0} := a_{0,1} * p_{0,0}$ after the parallel send is shown. Finally in part (c), the current $a_{-1,0}$ is sent to $c_{0,0}$ and consequently the final value of $a_{0,0} := a_{-1,0} * a_{0,0}$ is computed. Note that the final values of $a_{i,j}$ in Q_1 are

$a_{-2,0}$ = product of items assigned to cell $C_{-2,0}$ (available at $c_{-2,0}$ and $c_{-1,0}$),
$a_{-1,0}$ = product of items assigned to cells $C_{-2,0}$, $C_{-1,1}$, and $C_{-1,0}$ (available at $c_{-1,0}$ and $c_{0,0}$),
$a_{0,0}$ = product of items assigned to cells $C_{-2,0}$, $C_{-1,1}$, $C_{-1,0}$, $C_{0,2}$, $C_{0,1}$, and $C_{0,0}$ (available at $c_{0,0}$).

In particular, after the execution of the above code processor $c_{0,0}$ contains the product of all items assigned to Q_1, in other words $a_{0,0} = q_1$. Similarly, at the end of Step 2, processor $c_{1,0}$ contains $q_2 = a_{1,0}$, processor $c_{1,-1}$ contains $q_3 = a_{1,-1}$, and processor $c_{0,-1}$ contains $q_4 = a_{0,-1}$. For the example given in Figure 7 (b), these products are

$$q_1 = x[1:36]\ ,\quad q_2 = x[37:64]\ ,\quad q_3 = x[65:85]\ ,\quad q_4 = x[86:113]\ .$$

An individual **send** operation performed in the code Figure 8 requires τL steps. The time taken for the initial **parallel do** is $\tau L + 1$. The **for** loop takes $(r/L)(\tau L + 1)$ units of time, giving a total parallel time of $(r/L + 1)(\tau L + 1)$ for this step. Using the upper bound on r in terms of n given in (5) we find that the total time taken by Step 2 is at most

$$\tau\sqrt{\frac{n}{2}} + \frac{1}{L}\sqrt{\frac{n}{2}} + \tau L + 1 \ .$$

3. Calculate and forward the prefixes of (q_1, q_2, q_3) using the available values at the four processors $c_{0,0}$, $c_{1,0}$, $c_{1,-1}$, and $c_{0,-1}$ in clockwise manner as shown in Figure 10: First send q_1 to processor

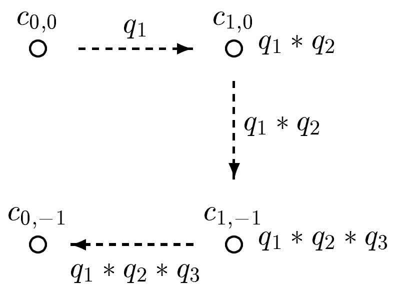


Figure 10: Execution of Step 3.

$c_{1,0}$ and compute $q_1 * q_2$. Next send $q_1 * q_2$ to processor $c_{1,-1}$ and compute $q_1 * q_2 * q_3$. Finally send this last value to processor $c_{0,-1}$.

The time required is:

$$2 + 3\tau$$

Going back to the example Figure 7 (b), at the end of Step 3, the prefixes $x[1,36]$, $x[1,64]$, and $x[1,85]$ are available at processors $c_{1,0}$, $c_{1,-1}$, and $c_{0,-1}$, respectively.

4. In Step 4 we combine two types of broadcast operations: broadcast to special processors that lie along the horizontal axis in each quadrant, and from each such processor to all special processors along its column. Again, these operations are carried on independently and in parallel in each quadrant. More precisely

   In Q_1, broadcast the value 1 from $c_{0,0}$ to each processor $c_{i,0},\ i \leq 0$.

   In Q_2, broadcast q_1 (obtained in Step 3) from processor $c_{1,0}$ to processors $c_{i,0},\ i > 0$.

   In Q_3, broadcast $q_1 * q_2$ (computed in Step 3) from processor $c_{1,-1}$ to processors $c_{i,-1},\ i > 0$.

   In Q_4, broadcast $q_1 * q_2 * q_3$ (computed in Step 3) from processor $c_{0,-1}$ to processors $c_{i,-1},\ i \leq 0$.

   The special processors that lie along the horizontal axis compute the smallest prefix that is needed in each cell in its column (we can call this the smallest prefix for the column) by a single multiplication as soon as they receive the horizontally broadcast value. Consequently they initiate a broadcast to the special processors in each cell that lie in their column. For example, in Q_1, processor $c_{i,0},\ i < 0$,

   1. receives 1 from $c_{i+1,0}$,
   2. sends 1 to $c_{i-1,0}$,
   3. computes $1 * a_{i-1,0}$ and initiates a broadcast to the special processors in column i by sending this value to $c_{i,1}$.

   Similarly, in quadrant Q_2, each $c_{i,0},\ i > 1$,

   1. receives q_1 from $c_{i-1,0}$,
   2. sends q_1 to $c_{i+1,0}$,
   3. computes $q_1 * a_{i+1,0}$ and initiates a broadcast to the special processors in column i by sending this value to $c_{i,1}$.

   This step is shown in Figure 11 in quadrant Q_2.

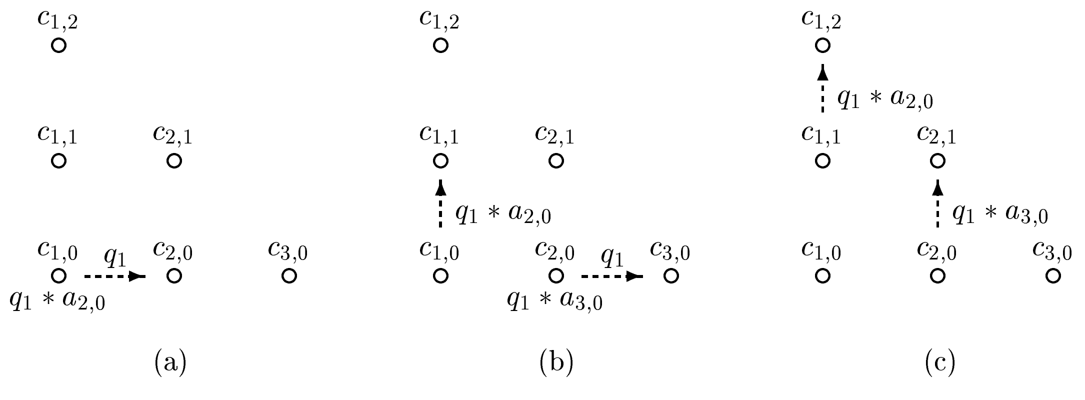
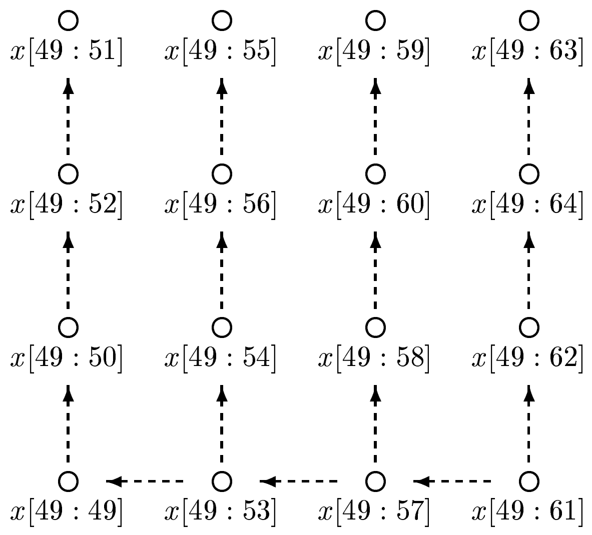


Figure 11: Execution of Step 4 in Q_2 with $r/L = 2$.

Each special processor receives the smallest prefix for its column computed above in no more than r communication steps and a single multiplication. Therefore the total time required is:

$$\tau\sqrt{\frac{n}{2}} + 1 .$$

5. In each cell $C_{i,j}$, compute the product of the smallest column prefix (which is available in $c_{i,j}$ after Step 4) and the value obtained in Step 2 from the closest cell in the same column. For example, in the first quadrant, the processor $c_{i,j}$ computes the product of the smallest column prefix and $a_{i,j+1}$, and in Q_3 processor $c_{i,j}$ computes the product of the smallest column prefix it received in Step 4 with $a_{i,j-1}$ that it obtained in Step 2. Each $c_{i,j}$ broadcasts this product to each processor in the cell as shown in Figure 12.



$x[43:48] = a_{1,1}$
$x[1:42] = a_{0,0} * a_{2,0}$
$x[1:48] = a_{1,1} * a_{0,0} * a_{2,0}$

Figure 12: Step 5 in $C_{1,0}$ with $r = 7$, $n = 117$, and $L = 4$: Broadcast $x[1:48]$.

The time required for this step is

$$1 + 2\tau L .$$

6. Calculate the actual prefix in each processor, using the value obtained in Step 5. The time required is

$$1 .$$

END ALGORITHM C

The total time T_C required by Algorithm C is the sum of the time required for Steps 1 through 6. This is found to be

$$2\tau\sqrt{\frac{n}{2}} + \frac{1}{L}\sqrt{\frac{n}{2}} + \frac{11}{2}\tau L + 2\log L + 4\tau + 7 . \quad (7)$$

To minimize time, we differentiate equation (7) with respect to L and equate it to 0. This gives

$$11\sqrt{2}\,\tau\, L^2 + 4\sqrt{2}\,L - 2\sqrt{n} = 0 \ .$$

Therefore,

$$L = \frac{-2 + \sqrt{4 + 11\,\sqrt{2}\,\tau\,\sqrt{n}}}{11\tau} \ ,$$

and for large n

$$L \approx \frac{\sqrt[4]{2n}}{\sqrt{11\tau}} \ . \tag{8}$$

Substituting this value in (8) into the expression (7) gives

$$T_C \approx \sqrt{2}\,\tau\,\sqrt{n} + \frac{\sqrt{44}}{\sqrt[4]{8}}\,\sqrt{\tau}\,\sqrt[4]{n} + 2\log\frac{\sqrt[4]{2n}}{\sqrt{11\tau}} + 4\tau + 7 \ .$$

Therefore the running time of Algorithm C is

$$T_C = \sqrt{2}\tau\sqrt{n} + O(\sqrt{\tau}\sqrt[4]{n}) \ . \tag{9}$$

By the lower bound given in part (a) of Lemma 2, algorithm C is asymptotically optimal.

6 Remarks

Under the assumptions that each processor is assigned a single item x_i, and the communication cost of sending a single input item between processors P and Q is $\tau d(P,Q)$, algorithms for the computation of prefix products of a list X of n items on mesh-connected multiprocessor systems are constructed. Initial loading of the items to the processors is not taken into account. For rectangular meshes two algorithms are provided: Algorithm A is simple to implement but suboptimal, whereas Algorithm B is asymptotically optimal but harder to implement. By partitioning the disc into variable size square meshes and using Algorithm B as a subprocedure on these squares, it is possible to construct an asymptotically optimal algorithm (Algorithm C) for the prefix problem on a disc with n processors. This latter algorithm assumes that n is of the form $n = 1 + 2r(r+1)$. If this is not the case, then list X can be augmented by adding necessary number of 1's to make it so. This new list can not be longer than the original by more than $O(\sqrt{n})$, therefore the asymptotic complexity of Algorithm C given in (9) remains valid.

References

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, 31(3):260–264, March 1982.

[3] D. A. Carlson. Modified mesh-connected parallel computers. *IEEE Transactions on Computers*, 37(10):1315–1321, October 1988.

[4] Ö. Eğecioğlu, Ç. K. Koç, and A. J. Laub. A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors. *Journal of Computational and Applied Mathematics*, 27(1+2):95–108, 1989.

[5] Ö. Eğecioğlu, E. Gallopoulos, and Ç. K. Koç. Parallel Hermite interpolation: An algebraic approach. *Computing*, 42(4):291–307, 1989.

[6] Ö. Eğecioğlu and Ç. K. Koç. Parallel prefix computation with few processors. *Computers and Mathematics with Applications*, Vol. 24, No. 4:77–84, 1992.

[7] F. E. Fich. New bounds for parallel prefix circuits. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 100–109, 1983.

[8] A. G. Greenberg, R. E. Ladner, M. Paterson, and Z. Galil. Efficient parallel algorithms for linear recurrence computation. *Information Processing Letters*, 15(1):31–35, 1982.

[9] L. Hyafil and H. T. Kung. The complexity of parallel evaluation of linear recurrences. *Journal of the ACM*, 24(3):513–521, July 1977.

[10] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, 22(8):786–792, August 1973.

[11] C. P. Kruskal, L. Rudolph, and M. Snir. The power of parallel prefix. *IEEE Transactions on Computers*, 34(10):965–968, October 1985.

[12] R. Ladner and M. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[13] S. Lakshmivarahan, C. Yang, and S. K. Dhall. On a new class of optimal parallel prefix circuits with (SIZE + DEPTH) $= 2n - 2$ and $\lceil \log n \rceil \leq$ DEPTH $\leq (2\lceil \log n \rceil - 3)$. In *Proceedings of the International Conference on Parallel Processing*, pages 58–65, August 17–21 1987.

[14] B. D. Lubachevsky and A. G. Greenberg. Simple, efficient asynchronous parallel prefix algorithms. In *Proceedings of the International Conference on Parallel Processing*, pages 66–69, August 17–21 1987.

[15] H. Meijer and S. G. Akl. Optimal computation of prefix sums on a binary tree of processors. *International Journal of Parallel Programming*, 16(2):127–136, 1987.

[16] M. Snir. Depth-size trade-offs for parallel prefix computation. *Journal of Algorithms*, 7(2):185–201, 1986.

[17] H. S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM*, 20(1):27–38, January 1973.