An Improved Parallel Prefix Computation on 2D-Mesh Network

Sudhanshu Kumar Jha*

Department of Computer Applications,
National Institute of Technology, Jamshedpur – 831 014 (INDIA)

Abstract

Parallel prefix is an important technique that has been widely accepted in many area of scientific and engineering research. In this paper we propose an improved parallel prefix computation algorithm on $n \times n$ mesh network that requires $2n + 5$ times. Our proposed algorithm can be compare with the traditional parallel prefix algorithm [1-4, 9] that requires $3n + 2$ time on same architecture.

Keywords: prefix computation, parallel prefix computation, modified prefix, 2D-mesh network

1. Main text

Parallel prefix computation is a basic tool that have been researched extensively for its wide application in various fields such as data-parallel programming, knapsack problem [5], sorting, parsing, combinator reduction, region labeling [6], medial axis transformation [7] etc. For a given sequence of data items say $x_0, x_1, \ldots, x_{k-1}$ and a binary associative operator say $\otimes$, the process of computing the value $p_i = x_0 \otimes x_i \otimes x_{i+1} \otimes \cdots \otimes x_{k-1}$ is called prefix computation.

A rich material is available on parallel prefix in the literature. Cole and Vishkin [8] developed an algorithm for prefix computation on a CRCW-PRAM model that requires $O(\log n / \log \log n)$ time using $(n \log \log n) / \log n$
processors. Egecioglu and Srinivasan [9] presented an algorithm on $\sqrt{n} \times \sqrt{n}$ mesh network that requires $2\tau \sqrt{n} + O(\log n)$ time, where $\tau$ is the time for a single routing step. Akl [2] showed that a specialized network with $O(n \log n)$ processors requires $O(\log n)$ time. Lin and Lin [10] presented a parallel prefix computation on a fully connected message passing system having $n$ processors require maximum $\lceil 1.44 \log n \rceil + 1$ communication steps. Meijer and Akl [11] presented an optimal-cost algorithm on a binary tree with $p$ processors in $O((n/p) + \log p)$ time for a cost of $O(n + p \log p)$. Ranka and Sahni [12] had shown that for $2^k$ window size hypercube requires $O(k)$ time to compute the prefix sum. Li, Peng and Chu [13] presented an algorithm for parallel prefix computation for a dual-processor [4].

High speed. A 2D mesh can be laid out on a VLSI chip in an area that increase the linearly with the number of processors. Egecioglu and Srinivasan [9] presented an algorithm on $n \times n$ mesh network that requires $4 \log n$ time. Lin and Lin [10] presented a parallel prefix computation on a fully connected network. The benefit of mesh network includes their simplicity, regularity and good scalability. A number of large research and commercial multicomputer systems have been built based on mesh architectures that include Illiac IV, Tera Computer System [21], Intel Paragon SP/S and Paragon XP/E [22], Cray system [23], Blue Gene Supercomputer [24] and InfiniBand architecture [25]. Mesh topology have been used to solve many engineering and scientific applications include sorting, matrix multiplication and inversion, Fourier transformation, convolution, signal and image processing, speech recognition, finite element analysis, polynomial interpolation and many more.

It is clear that because of the short, local connection among the processors, the area consumed by the wires in mesh network is negligible and also realistic and fair to equate the complexity or implementation cost of a 2D mesh network. The signal propagation delay among the adjacent processor is quite small making a communication at very high speed. A 2D mesh can be laid out on a VLSI chip in an area that increase the linearly with the number of processors [4].

In this paper we propose an improved parallel algorithm for prefix computation on 2D mesh network. The proposed parallel prefix algorithm is shown to be run in $2n + 5$ times which is comparable with the algorithm presented in [1-5, 9].

The rest of the paper is organized as follows. Section 2 describes the topological structure of the mesh architecture. In section 3 we present the proposed parallel prefix algorithm on two dimensional mesh network followed by conclusion in section 4.

2. Topological Structure of Mesh Network

An $n \times n$ mesh network is a simple and popular topology. It consists of $n^2$ processors arranged into form a 2-dimensional lattice. An $n \times n$ mesh has diameter $2n - 2$ and bisection width $n$ or $n + 1$. The processors in 2-dimensions mesh can be indexed into number of various ways. The most convenient and acceptable way is to identify the processor by its particular row and column using origin $(1, 1)$, thus the processor $(1, 1)$ is reside at the
upper left corner. Therefore, the processor \((1, n)\) is at the upper right corner, the processor \((n, 1)\) is at lower left corner and the processor \((n, n)\) at the lower right corner of an \(n^2\)-processors square 2-dimensional mesh. If each node is addressed by a pair of indices say \((x, y)\) where \(1 \leq x, y \leq n\), then two processors say \((x, y)\) and \((x', y')\) are connected if and only if \(x' = x \pm 1\) and \(y' = y\) or \(y' = y \pm 1\) and \(x' = x\). As an example \(7 \times 7\) 2D mesh is shown in Fig. 1 in which the processors and links are represented by small circles and solid lines respectively.

![Fig. 1 Mesh network consists of 7×7 processors](image)

We assume here that all links are bi-directional therefore the data movements can be done in both directions and boldfaces adjacent to some processors represent the indices for particular processor.

### 3. Proposed Parallel Prefix Computation

In literature, traditional parallel prefix computation [1-5] requires three basic steps 1) perform row-wise parallel prefix computation in parallel 2) perform a modified parallel prefix computation in the rightmost column 3) broadcast the computed modified prefix value from the rightmost column processor to all of the processors in the respective rows and combine with the initially computed row prefix value. A modified prefix [3] or diminished prefix [4] is an important technique for parallel prefix computation in parallel architectures such as mesh [3], mesh of trees [20], OTIS mesh [14], OTIS mesh of trees [20], multi mesh [26], multi mesh of trees [27] etc.

Our basic observation is as follows: during the modified prefix at the rightmost column processors all other processors remains idle which is the wastage of precious computation power of the processing elements. In an interconnection network where all processors are independent to each other to process their task this can be done at the midlist column processor instead of the rightmost column processor. While the midlist column processors are engaged to perform prefix computation, simultaneously, some of the processors can also perform a prefix computation in same way. By this, we can reduce the overall time complexity of the parallel prefix computation in 2D mesh network.

We can see that, in modified prefix the computation of prefixes shift down by one processor, which requires \(n - 1\) communication/ combining steps. Therefore, we cannot perform parallel prefix computation and modified prefix computation at a same time in SIMD architecture. This can also be improving in slightly different way. Instead of performing of modified prefix, we perform 1) a parallel prefix computation 2) now shift down the computed prefix value by one processor that will require \(n\) steps.

We assume here all processors have two register in which one should hold the temporarily computed value. The proposed algorithm is based on the SIMD architecture where all active processors perform a same task.
**Initialization:** We assume here $n^2$ data elements $x_1, x_2, x_3, \ldots, x_{n^2}$ are stored in $A$ register of mesh network in row major order as shown in Fig. 2 for $n = 5$.

**Note:** In each figure the square represents the processing elements (PE). The upper halves of the square denote the contents at $A$ register and lower half of the square represents the contents at register $B$. The ‘-’ denotes the don’t care value.

**Algorithm** `parallel_prefix_MESH()`

**Step 1:** For all processor $p(i, j)$ $\forall i, j, 1 \leq i, j \leq n$ do step 1.1 and 1.2 in parallel

1.1 Row $i$, $\forall i, 1 \leq i \leq \lfloor \frac{n}{2} \rfloor$ computes the prefixes of its $\lfloor \frac{n}{2} \rfloor$ elements and store it at $A$ register of $p(i, j)$, $\forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq \lfloor \frac{n}{2} \rfloor$. After this step the register $A$ of processor $p(i, j)$, $\forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq \frac{n}{2}$, holds the value

$$\sum_{q=1}^{\lfloor \frac{n}{2} \rfloor} x_{i,q}$$

as shown in Fig. 3.

1.2 In each row processor $p(i,j)$, $\forall i, 1 \leq i \leq n, \forall j, \lceil \frac{n}{2} \rceil + 1 \leq j \leq n$ compute the prefixes of its $\lfloor \frac{n}{2} \rfloor$ elements starting from $(i, n)$ $\forall i, 1 \leq i \leq n$ therefore in this the step the direction of prefixes in each row is $p(i,n) \otimes p(i,n-1) \otimes p(i,n-2) \otimes \ldots \otimes p(i,\lceil \frac{n}{2} \rceil + 1)$ and store the result at $B$ register. After this step the register $B$ of processor $p(i, j)$ $\forall i, 1 \leq i \leq n, \forall j, \lceil \frac{n}{2} \rceil + 1 \leq j \leq n$, holds the prefixes value

$$\sum_{q=\lceil \frac{n}{2} \rceil + 1}^{\lfloor n \rfloor} x_{i,q}$$

as shown in Fig. 3.

**Step 2:** For all processor $p(i, \lceil \frac{n}{2} \rceil)$ $\forall i, 1 \leq i \leq n$ do in parallel

The processor $p(i, \lceil \frac{n}{2} \rceil)$ $\forall i, 1 \leq i \leq n$ received the data sent from $p(i, \lceil \frac{n}{2} \rceil + 1), \forall i, 1 \leq i \leq n$ and store into $B$ register. The situation is shown in fig. 4.
Step 3: For all processor $p(i, [i] + \frac{1}{2}) \forall 1 \leq i \leq n$ do in parallel

Perform the prefix computation \( \otimes \) on the contents of $A$ and $B$ registers and save it into $B$ register. The result after this step is shown in fig. 5.

$$B \text{ register data} \leftarrow A \text{ register value} \otimes B \text{ register value}$$

Fig. 4 Situation after step 2

Fig. 5 Situation after step 3

Step 4: For all processor $p(i, [i] + 1), \forall i, 1 \leq i \leq n$ do in parallel

In each row $i$, $\forall i, 1 \leq i \leq n$, the processor $p(i, [i] + 1), \forall i, 1 \leq i \leq n$ receive the $A$ register data sent from $p(i, [i] + \frac{1}{2}), \forall i, 1 \leq i \leq n$. Compute the prefixes on received data with own $A$ register data and store in $A$ register as shown in fig. 6.

Fig. 6 Situation after step 4

Fig. 7 Situation after step 5
Step 5: For all processor \( p(i, j) \forall 1 \leq i \leq n, \forall \left[ \frac{j}{q} \right] \leq j \leq n \) step 5.1 and 5.2 do in parallel

5.1 For all processor \( p(i, j) \forall 1 \leq i \leq n , \forall \left[ \frac{j}{q} \right] + 1 \leq j \leq n \) do in parallel

In each row \( i \), computes the prefixes of its remaining \( \left[ \frac{j}{q} \right] - 1 \) elements. At the end the \( A \) register of processor \( p(i, j), \forall 1 \leq i, j \leq n \), holds \( y_{i,j} = \sum_{q=i}^{j} x_{i,q} \) prefixes value.

5.2 Only column \( \left[ \frac{j}{q} \right] \) processor computes the prefixes of sums on \( B \) register data computed in step 3. Thus at the end the \( B \) register of processor \( p(i, j) \forall 1 \leq i \leq n \) has \( y_{i,j} = \sum_{q=i}^{j} x_{i,q} \).

The result after the step 5.1 and 5.2 is shown in fig. 7.

Step 6: for all processors \( p(i, \left[ \frac{j}{q} \right]), \forall 1 \leq i \leq n \) do in parallel

The computations of prefixes shift them down by one processor and store in \( B \) register i.e., the processor \( p(i, \left[ \frac{j}{q} \right]) \forall 1 \leq i \leq n \) send the \( B \) register data to processor \( p(i + 1, \left[ \frac{j}{q} \right]) \forall 1 \leq i \leq n \) as shown in fig. 8.

Step 7: For all processor \( p(i, \left[ \frac{j}{q} \right]) \forall 1 \leq i \leq n \) do in parallel

Broadcast the contents of \( B \) register to all processor in same row. All receiving processor store the value in corresponding \( B \) register as shown in fig. 9.

Step 8: For all processor \( p(i, j) \forall 1 \leq i, j \leq n \) do in parallel

Perform the operation \( \odot \) on the contents of \( A \) register and \( B \) register and store in \( A \) register result shown in Fig. 10.

\[
A\ register\ data \quad \longleftrightarrow \quad A\ register\ value \quad \odot \quad B\ register\ value
\]

Fig. 8 Situation after step 6

Fig. 9 Situation after step 7

Step 9: Stop
**Time Complexity:**

Step 1, 7: Perform prefixes of $\left\lceil \frac{n}{2} \right\rceil$ elements in $\left\lceil \frac{n}{2} \right\rceil$ time.

Step 2, 3, 4, 6, 8: Each will take constant time to perform receive, store, prefix operation, shifting of data values etc.

Step 5 will take $n$ time to perform prefixes computation.

Therefore, the proposed algorithm requires $2n + 5$ steps to compute the prefix computation of $n^2$ data elements on $n \times n$ two-dimensional mesh network.

4. **Conclusions**

In this paper we have shown an improved parallel prefix algorithm on 2D mesh network which requires $2n + 5$ times. Our proposed parallel prefix algorithm can be comparable with the parallel prefix algorithm on mesh network presented in [1-5, 9].

**References**