Daoud’s Parallel prefix Applications:

The prefix problem can be easily seen in the application of carry look-ahead adders, and various implementations of these (1, pp154-160, 226-227). For instance, carry select adders can be seen as twin blocking processes with greater prefix algorithm size in order to reduce the depth of the calculation, and since they are elementary units in ALUs and hardwired it can be seen that this tradeoff in increase in size increases the speed of the computation, but also increases the number of gates in each ALU though the character of the implementation makes the data streams easy to implement with splitters to the select units.

Another application of the parallel prefix problem is in the computation of fluid dynamics problems – compressible flow (2, p1), 2 phase flow etc. which involve matrix intensive computations including the banded matrix problem examined in class and (1, pp374-379), (1, 338-339) which my second source indicates is easy for single processor machines but difficult to implement in parallel processors – though we are tackling the subject.

2) A Simple Prefix Algorithm for compact Finite Difference Schemes – Xian-He Sun, Ronald D. Joslin

Comments by Rukmini and Supriya on Daoud’s Review:

The applications presented by Daoud for parallel prefix which involve matrix intensive computations, were good. The references given were that of two books so could not find many details about the applications. Needs more information.

Jayanthi’s Parallel prefix Applications:

Applications for prefix problems:
1) Prefix Sums

Prefix Sums can be applied in mathematical operations when given a set X with operation @ defined on elements of X such that:

a) @ is binary and it applies to all element pairs of X

b) If Xi and Xj are elements of X, the n Xi @ Xj is also an element of X

c) @ is associative with respect to elements of X.

Applications:
1) Maximum Sum Subsequence Problem.
Definition: Given a sequence of numbers \( X = \{ X_0, X_1, \ldots, X_{n-1} \} \), find two indices \( u \) and \( v \) \((u < v)\) such that the subsequence \( \{X_u, X_{u+1}, \ldots, X_v\} \) has the largest possible sum among all subsequences of \( X \). This problem is difficult if negative numbers are present in sequence.

Analysis:
The sequential problem takes \( O(\log n) \) time.

The prefix algorithm takes \( O(n) \) time using the parallel solution as follows:

1) It first computes prefix sums of \( X_0 \) to \( X_{n-1} \)

2) For each \( i = 0 \) to \( n-1 \) obtain maximum \( S_j \) to the right, beginning with \( S_i : M_i = \max \{ S_j \mid I \leq j \leq n-1 \} \)

3) Assume \( A_i \) is the index at which \( M_i \) (max) is found.

4) For each \( i \), compute sum of max sum subsequence
   \[ B_i = M_i - S_i + X_i \]

5) Overall maximum sum subsequence \( \Rightarrow \)
   \[ L = \max \{ B_i \mid 0 \leq i \leq n-1 \} \] \( u \) is the index where maximum is found, i.e., \( v = A_u \).

If algorithm is implemented with \( n/\log n \) processors:

Step 1 and step 2 take \( O(\log n) \) time

Step 3 takes \( O(k) = O(\log n) \) time as each processor computes \( k \) values

Step 5 takes \( \log n \) time as max of \( \log n \) values is calculated.

Time taken by prefix computation based Maximum sum subsequence algorithm = \( O(\log n) \)

Cost of prefix computation based Maximum sum subsequence algorithm = \( O(n) \) which is optimal

2) Array Packing
   Definition: Given an array \( X \) of \( n \) elements, some of whose entries are labeled and which are scattered arbitrarily, bring the labeled elements into contiguous positions of the same array

Analysis:
Sequential algorithm takes \( O(n) \) time.
Parallel algorithm uses a secondary array of n elements $S = \{S_1, S_2, \ldots, S_n\}$ to compute destination of each labeled element of $X$.

Assuming array $X$ is created with $n / \log n$ processors in $O(\log n)$ time each processor filling $\log n$ positions of $S$

1) $S_i = 1$ if $X_i$ is labeled or $S_i = 0$ otherwise.

2) Prefix sum computation is performed on elements of $S$.

3) Each labeled element $X_i$ of $X$ is copied into position $S_i$ of array $Y$.

References:

http://cs.hiram.edu/~irina/parallel/Prefix.ppt

Comments by Rukmini and Supriya for Jayanthi:

The application for parallel prefix sum was addressed well with the analysis. The analysis showed the comparison of parallel algorithms with sequential algorithms. The reference provided was informative and contained prefix problems in more details.

Sheela’s Parallel prefix Applications:

1. “A Reconfigurable Network Architecture For Parallel Prefix Counting”

An efficient reconfigurable parallel prefix counting network of size $N-1$ is proposed by this paper. This proposed architecture is significantly faster than any known architectures, including tree of adders or a processor with the same structure as proposed in his paper. Here $N$-prefix sums are computed and output is done row by row with a simple PE or control unit per row. The control units run asynchronously, driven solely by enable signals, which simplifies the hardware requirement, and the full inherent speed of the computation can be utilized. This is a very interesting paper.

2. Optimum Parallel Approximation for Prefix Sums and Integer Sorting.

This paper introduces an approximate parallel prefix routine that can be used in a number of non-trivial applications. The most interesting application is for padded integer sorting where one wishes that the $n$ integers are sorted into an array of size $O(n)$. But this algorithm solves the problem in $O(\log n)$ time. Also, the paper presents several applications to integer-coordinate problems in computational geometry, such as convex hulls and hidden line elimination as well as for approximate selection.

3. Parallel Prefix on the NuMesh can be seen at http://www.incert.com/~metcalf/papers/parpref/
The parallel prefix routine is particularly useful in data-parallel models, where inventively used parallel prefix operations can make up substantial parts of programs. The prefix operation works as follows. For n nodes, label them x1 to xn. For some operation $\otimes$, when we're done we want each node i to hold the value $x_1 \otimes \cdots \otimes x_i$. Any associative operation at all can be used for prefix: add (integer or FP), OR, AND, XOR, MIN, MAX, etc., as well as any user-defined operation. (The carry-look ahead adder is itself an example of parallel prefix.)

Comments by Rukmini and Supriya for Sheela:
Sheela mentions the implementation of parallel prefix sums which was very informative but was not required. The application of Integer sorting is a good example where parallel prefix algorithms can be used. She also showed how it is useful in data parallel models. Good references.

Kuntal’s Parallel prefix Applications:

Application Area 1: Maximum Subarray Problem
References: [http://citeseer.nj.nec.com/perumalla95parallel.html](http://citeseer.nj.nec.com/perumalla95parallel.html)

Given a two-dimensional array A of n x 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.

Solving the above problem will solve the following problem.

Application of the above problem: Given an n x n array of reals, the problem of finding a rectangular subarray with maximum sum arises in 2-dimensional pattern matching. Such a maximum-sum subarray corresponds to a maximum-likelihood estimator of a certain kind of pattern in a digitized picture.

The above problem reduces to the solution of the problem given below which will use prefix algorithm for the solution to achieve parallelism.

Given a sequence Q on n numbers (positive and negative), the maximum subsequence of Q is the contiguous subsequence that has the maximum sum among all contiguous subsequence of Q.
The sequential time complexity of Maximum Subsequence and Maximum Subarray are $O(n)$ and $O(n^3)$ respectively. The parallel algorithm presented by Perumalla etc. all will solve the Maximum Subsequence problem and Maximum Subarray problem in $O(\log n)$ time using EREW PRAM model and CREW PRAM model. (EREW – exclusive read exclusive write, CREW- concurrent read and exclusive write, PRAM – shared memory random access machine). The number of processor used were $O(n/\log n)$ for Maximum Subsequence using both the models. The number of processors used for Maximum Subarray problem were $O(n^3)$ using EREW PRAM model and $O(n^3/\log n)$ for CREW PRAM model.

The authors give very good examples to explain the algorithm they presented. (See pages 3 and 4) Since I am attaching the paper itself with this assignment I would not go into the details of algorithm but I would definitely point out the steps at where we apply the prefix sum algorithms. This shown in the algorithm which is given as follows.

**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) $M[i] := PMAEX[i] - SSUM[i] + Q[i]$
   
   b) $M^p[i] := SMAX[i] - PSUM[i] + Q[i]$
   

6. Find the maximum of $M$ into $MSQ$.
7. Output $MSQ$.

**Application Area 2:** Finite automata problems

References: Pages 3 to 7 of this paper [http://citeseer.nj.nec.com/374523.html](http://citeseer.nj.nec.com/374523.html)

The prefix algorithm comes handy for lot of finite automata problems as discussed by Ravikumar in his paper. For more details the relevant pages of the paper are attached herewith. I will summarize the Membership problem for which the prefix algorithm is used.

**Problem definition:** Given a finite automaton $M$ for some language $L$ over some alphabets, determine if string $x$ is a member of $L(M)$. The following algorithm was presented by Ravikumar to solve the above problem.
**ALGORITHM** Prefix Sums

**Input:** An array of $n = 2^k$ elements $(x_1, x_2, ..., x_n)$ where $k$ is a nonnegative integer.

**Output:** The prefix sums $s_i = x_1 + ... + x_i$, for $i = 1, ..., n$.

**begin**

if ($n=1$) then set $s_1 = x_1$; exit.

for $i$ from 1 to $n/2$ pardo

Set $y_i = x_{2i-1} * x_{2i}$

Recursively compute the prefix sums of $\{y_1, y_2, ..., y_{n/2}\}$ and store them in $z_1, z_2, ..., z_{n/2}$.

for $i$ from 1 to $n$ pardo

\[ \text{case i:} \]

i even: set $s_i = z_i/2$

i = 1: set $s_1 = x_1$

i odd and $i > 1$: set $s_i = z_{(i-1)/2} * x_i$

**end**

The above problem requires $O(\log n)$ time and $O(n)$ total work. The sequential algorithm will take $O(n)$ time for such a problem. For more details see page 4 and 5

Comments by Rukmini and Supriya for Kuntal:

The two application areas showed by Kuntal were addressed well. He showed how the parallel prefix algorithms are useful for a lot of automata problems and also mentions the discussions done by Ravikumar in one of the papers. Good comparison with sequential algorithms with respect to time and total work done.

Sufficient references were given.

Comments by Rukmini for Supriya:

(soft copy not provided, based on hard copy I am submitting my comments)

The example of integer sorting was presented well with sufficient references given. Also Supriya covers one of the simplest and most useful parallel algorithms--prefix-sums operation.

**Rukmini’s Application of parallel prefix problem**

The following algorithms provide examples as how to analyze algorithms in terms of work and depth and how to use nested data-parallel constructs. The following examples are available at http://www-2.cs.cmu.edu/~scandal/cacm/node10.html.

1. **Examples of primes summary:** The example describes an algorithm that finds all the prime numbers less than $n$. This example demonstrates a common technique used in parallel algorithms--solving a smaller case of the same problem to speed the solution of the full problem and also that is work efficient. First, it demonstrates with simple sequential algorithms, derives depth, size, etc, are compared to size, depth derived using parallel algorithms.
2. **planar convex hull problem summary:** Given \( n \) points in a plane, find which of them lie on the perimeter of the smallest convex region that contains all points. This example shows use of nested parallelism for divide-and-conquer algorithms. Using the parallel prefix the depth, efficiency is derived.

**Comments by Supriya for Rukmini:**

Rukmini describes algorithm that finds all the prime numbers less than \( n \) and planar convex hull problem was addressed well. The reference provided was informative. But should have given more details.

Ahmad’s Application of parallel prefix problem:

A Family of Parallel Prefix Algorithms Embedded in Networks, this is the link:

http://www.computer.org/tpds/td1993/l1179abs.htm

This paper talks about several algorithms in an interconnected network of processors. Each one can be embedded in a switch in networks such as Omega and hypercube.

I found it also provides a pattern for each one to describe underlying operations.

**Prefix Sums And Their Applications**

http://www.dcs.qmul.ac.uk/SEL-HPC/Articles/GeneratedHtml/hpc.scan.html

This paper relies on the simplifications of the algorithms by using blocks and focus on their functionality in the whole view of the algorithm.

It starts with the simplest method of all-prefix-sum-operations and it’s applications such as the following:

**Lexical compare strings of characters**

A. To add multi-precision numbers, those that cannot be represented by a single machine word.

B. To evaluate polynomials.

C. To solve recurrences.

D. To implement radix sort.

E. To implement quick sort.
F. To solve traditional linear system.

G. To delete marked elements from an array.

H. To dynamically allocate processors.

I. To perform lexical analysis such as string parsing into tokens.

J. To search for regular expressions, such as UNIX group utility.

K. To implement some tree applications.

L. To label component in two-dimensional images.

I found this paper also map algorithms into the P-RAM model of processors. All-sum-prefix operations considered as a primitive, along with the minimum and maximum operations in some parallel machine components.

**Parallel Prefix and reduction Algorithms Using Coterie Structures**

http://www.dcs.qmul.ac.uk/SEL-HPC/Articles/GeneratedHtml/hpc.scan.html

In this paper the authors introduce a new model/ technique called coterie structure for simultaneous parallel processing of connected components on re-configurable broadcast meshes.

Comments by Rukmini and Supriya for Ahmad:

Parallel Prefix Algorithms Embedded in Networks and prefix sum are presented well. He also shows show these algorithms are applied in models. Very good references were provided.

Viraj’s Parallel prefix application:

1) PARALLEL PREFIX ALGORITHMS EMBEDDED IN NETWORKS

A Family of Parallel Prefix Algorithms Embedded in Networks
TAKESUE Masaru

? 1 NTT Software Laboratories
This paper presents a family of algorithms for producing, from $(?_0, ?_1,..., ?_{<n-1>})$, all initial prefixes $x_i = ?_0 ? ?_1 ? ... ? ?_i (i=0,1,...,n-1)$ in parallel in interconnection networks such as the omega network and hypercube, where $?$ is an associative binary operator. Each algorithm can be embedded in the switches and interconnections of the network, and can be executed in $O(\log_s n)$ time steps provided that the network connecting $n$ processors is constructed by using an $s \times s$ switch. The objective of these algorithms is thus not necessarily to improve the time and space required to execute them, but to attain a communication pattern that fits the topology of the network. Because one type of network can be made equivalent to, or can be embedded in, another type of network, a family of algorithms can be derived from one basic algorithm. In the basic algorithm, in principle, every processor $p_i$ multicasts $v_i$ to the processors $p_k$ ($k=i+1,i+2,...,n-1$). En route to $p_i$, $?_j (j=0,1,...,i-1)$ are combined with each other in the switches to calculate the $(i-1)$-th initial prefix $x_{<i-1>}$ that is received by $p_i$, which thus computes the $i$-th initial prefix $x_i = x_{<i-1>} ? ?_i$.

The above description says that the prefix algorithms can be used for embedded networks. These algorithms are embedded in switches and follow a certain communication pattern of that of the parallel algorithms.[1]

2 PREFIX COMPUTATIONS ON MULTI MESH ALGORITHMS

The diameter of a topological network plays a vital role in determining the lower bound of the running time in problems where data elements need to be routed between the diametrically opposite processors. The prefix problem is a good example of this kind. The multi-mesh topology offers a distinct advantage over the mesh in such cases because of its lower diameter. The parallel solutions to many real life problems such as job scheduling, knapsack problem, etc., are dependent on the efficiency at which prefix computations can be carried out on a set of values. The key idea is as follows. The computation is actually performed in two basic phases. In the first phase, we apply the mesh prefix computation on all the blocks in parallel and the partial results are stored which is local to the individual block. In phase two, we emulate the prefix computation with the partial result of phase one on the whole architecture. For this, we treat each block as a single virtual processor and compute the final result analogous to phase one. To implement phase two, we actually search some virtual links between different blocks to broadcast the partial results stored in phase one. [2]
Comments by Rukmini and Supriya for Viraj:

Parallel prefix embedded in network certainly seems to be a popular example. Viraj’s approach towards application were interesting especially parallel solution in multi meshes. References provided by Viraj were good and informative