Complexity and Advanced Algorithms
Monsoon 2011

Parallel Algorithms
Lecture 2

Trivia

- ISRO has a new supercomputer rated at 220 Tflops
  - Can be extended to Pflops.
  - Consumes only 150 KW of power.
- LINPACK is the standard benchmark used for these ratings.
  - Can get your computer rated too.
  - Visit http://www.netlib.org/benchmark/linpackjava/

Balanced Binary Tree – Prefix Sums

- Two traversals of a complete binary tree.
- The tree is only a visual aid.
  - Map processors to locations in the tree
  - Perform equivalent computations.
  - Algorithm designed in the PRAM model.
  - Works in logarithmic time, and optimal number of operations.

```
//upward traversal
1. for i = 1 to n/2 do in parallel
    $b_i = a_{2i-2}$ o $a_{2i}$
2. Recursively compute the prefix sums of B= ($b_1$, $b_2$, ..., $b_{n/2}$) and store them in C = ($c_1$, $c_2$, ..., $c_{n/2}$)
//downward traversal
3. for i = 1 to n do in parallel
       i is even : $s_i = c_i$
       i= 1 : $s_1 = c_1$
       i is odd : $s_i = c_{(i-1)/2}$ o $a_i$
```

Analysis of Parallel Algorithms

- To analyze parallel algorithms, we rely on asymptotics and recurrences.
- Each operation costs 1 unit, only sequential time needs to be counted. We assume as many processors as can be used are available.
- In the prefix sum example, let T(n) be the time in parallel for an input of size n.
  - Step 1 can use n/2 processors in parallel each taking 1 unit of time.
  - Step 2 is a recursive call and takes T(n/2) time.
  - Step 3 uses n processors each taking 1 unit of time.

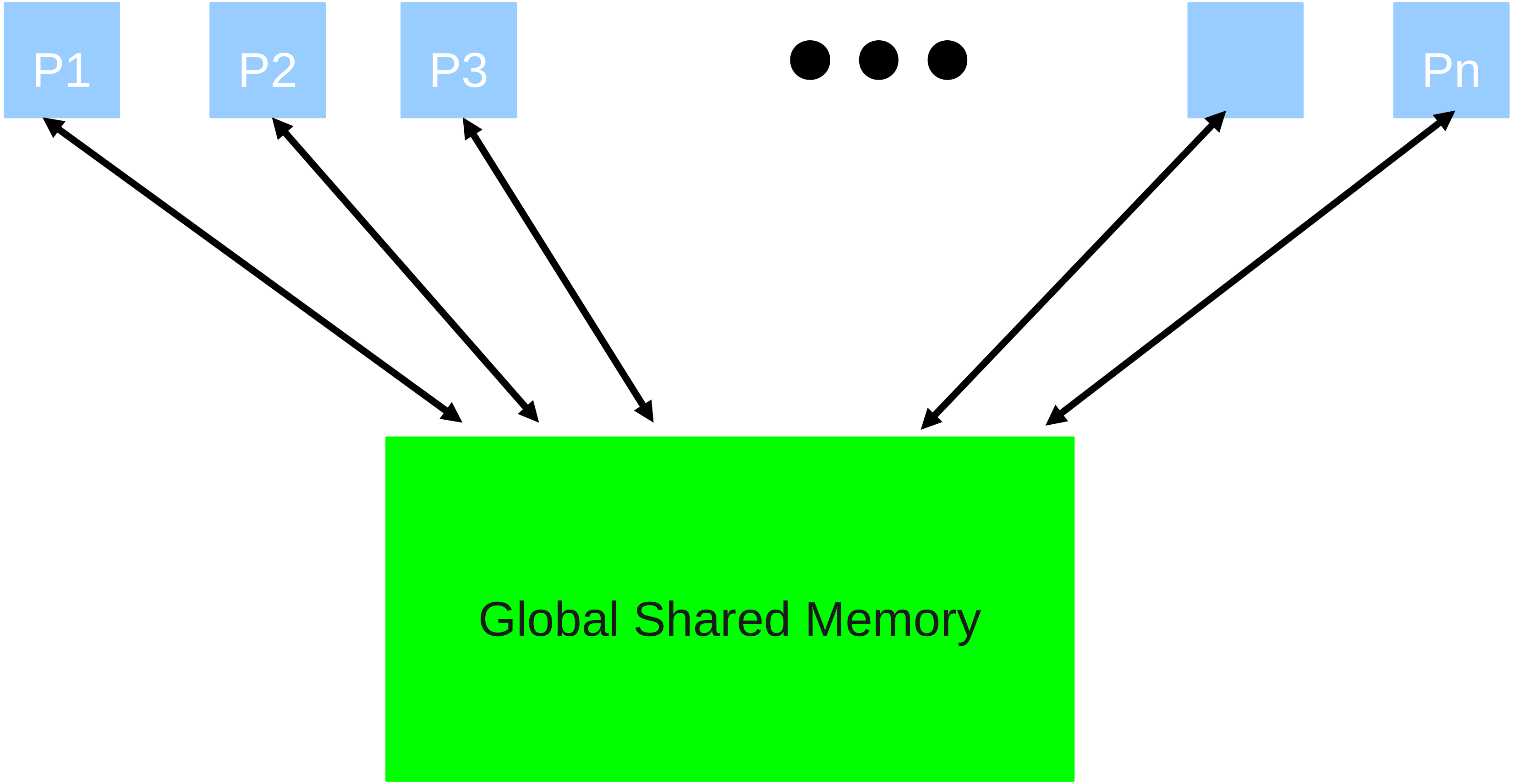
Analysis of Parallel Algorithms

- The recurrence relation is:
  - $T(n) = T(n/2) + O(1)$
  - Can ignore effects due to constant factors, such as the difference in the number of processors between steps 1 and 3.
- The solution to the above recurrence is $T(n) = O(\log n)$.
- Another parameter of interest in parallel algorithms is the work done.
- Can be stated as the sum of the works done by each of the processors.

Analysis of Parallel Algorithms

- The work done by the prefix algorithm can be expressed by the recurrence
  - $W(n) = W(n/2) + O(n)$.
  - The $O(n)$ accounts for the work in the first and the third steps.
  - Solution: $W(n) = O(n)$.
- Work done can indicate if the algorithm is doing about the same amount of operations as the best known sequential algorithm.

The PRAM Model



- An extension of the von Neumann model.

The PRAM Model

- A set of n identical processors
- A common access shared memory
- Synchronous time steps
- Access to the shared memory costs the same as a unit of computation.
- Different models to provide semantics for concurrent access to the shared memory
  - EREW, CREW, CRCW(Common, Aribitrary, Priority, ...)

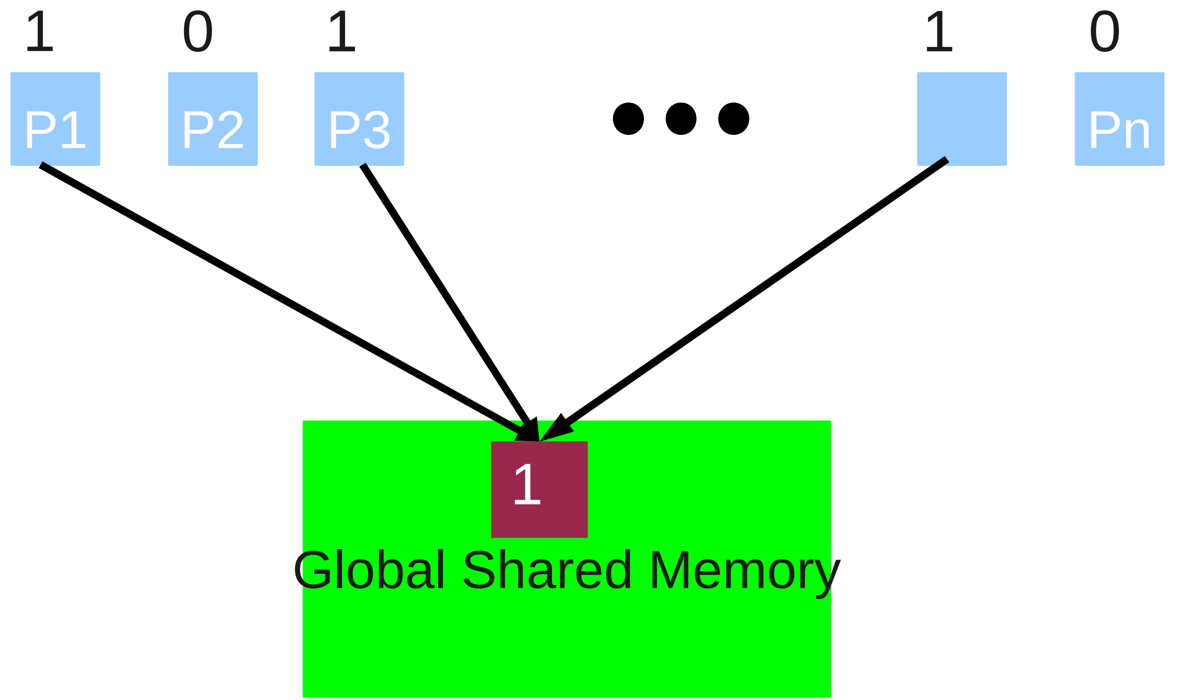
The Semantics

- In all cases, it is the programmer to ensure that his program meets the required semantics.
- EREW : Exclusive Read, Exclusive Write
  - No scope for memory contention.
  - Usually the weakest model, and hence algorithm design is tough.
- CREW : Concurrent Read, Exclusive Write
  - Allow processors to read simultaneously from the same memory location at the same instant.
  - Can be made practically feasible with additional hardware

The Semantics

- CRCW : Concurrent Read, Concurrent Write
  - Allow processors to read/write simultaneously from/to the same memory location at the same instant.
  - Requires further specification of semantics for concurrent write. Popular variants include

Concurrent Read/Write Models



- COMMON : Concurrent write is allowed so long as the all the values being attempted are equal.
- Example: Consider finding the Boolean OR of n bits.
  - Each Boolean bit with a processor.
  - Reserve a common cell in the memory
  - Every processor that holds a 1 writes 1 in the common cell.

Concurrent Read/Write Models

- ARBITRARY : In case of a concurrent write, it is guaranteed that some processor succeeds and its write takes effect.
  - Similar algorithms for Boolean AND can be designed.
  - Turns out that ARBITRARY is at least as powerful as COMMON.
    - If all values are equal, both match the semantics.
    - Otherwise, ARBITRARY may be more useful.

- PRIORITY : Assumes that processors have numbers that can be used to decide which write succeeds.
  - Difficult to place wrt ARBITRARY and COMMON.

- Other models: TOLERANT, COLLISION, ...

Revisiting our Prefix Sum Algorithm

- What model of PRAM does our algorithm require?
    - No concurrent writes.
    - May be concurrent reads.
    - Can however convert to an exclusive read algorithm. How?
    - Can this be done in all cases?

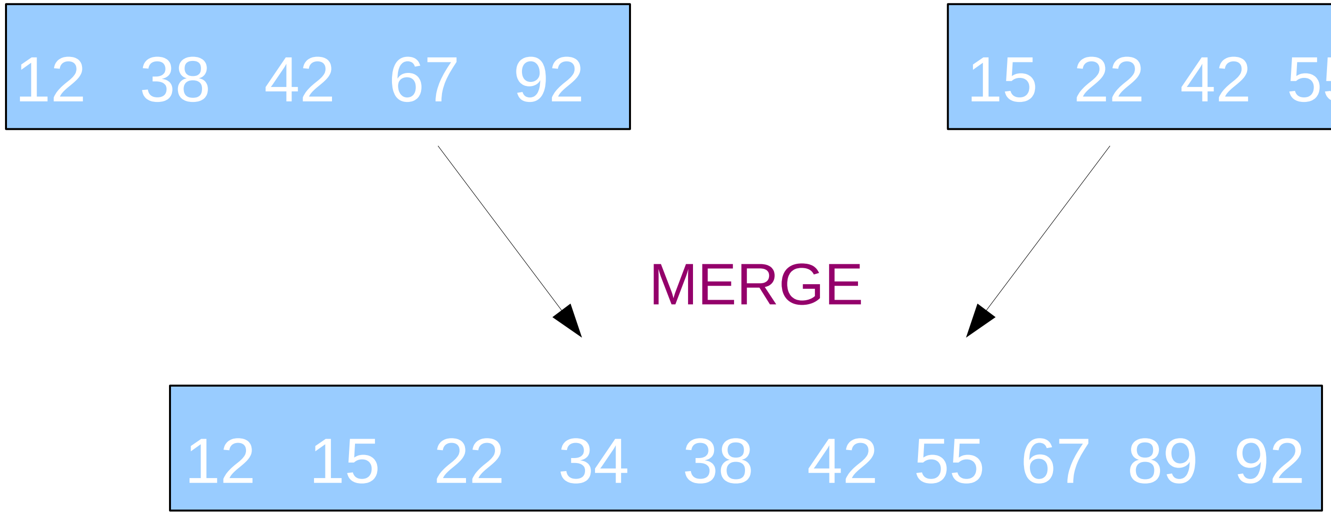
Other Design Paradigms

- Partitioning
  - Similar to divide and conquer
  - But no need to combine solutions
  - Can treat problems independently and solve in parallel.
  - Example: Parallel merging, searching.

Coming Up...

- From now on, we will also care about the model of the PRAM when designing algorithms.
- Aim for EREW algorithms because of their applicability.
- We will design algorithms for
  - Merging
  - Sorting
  - List ranking
  - Tree algorithms
  - Graph algorithms
  - ...

Parallel Algorithm for Merging



12 38 42 67 92

15 22 42 55 89

MERGE

12 15 22 34 38 42 55 67 89 92

- The problem:
  - Input: Two sorted arrays A and B of size n each.
  - Output: A sorted array C that contains all the elements of A and B.
  - Assume that all elements are distinct. Can be done away easily. (HW)

The Sequential Algorithm

→ 8 10 12 24 → 15 17 27 32

8

8 10

8 10 12

8 10 12 15

8 10 12 15 17

8 10 12 15 17 24

8 10 12 15 17 24 27

8 10 12 15 17 24 27 32

```
Procedure Merge(L, R, A)
   i = 1, j = 1
   for k = 1 to l + r do
     if L[i] < R[j] then
        A[k] = L[i];
        i = i + 1
    else
      A[k] = R[j];
      j = j + 1;
End.
```

Sequential Algorithm

- In the sequential algorithm, any element x waits for the positions of all lesser elements to be determined.
- The position of x is one plus the position of the largest element smaller than x.
- Looks inherently sequential.

Towards a First Parallel Algorithm

- Define Rank(x,A) to be the number of elements of A that are smaller than x in a sorted array A.
  - Example: A = (14, 18, 43, 58), Rank(49, A) = 3.
  - Notice that x need not be an element of A.
- Claim: Rank(x, C) = Rank(x, A) + Rank(x, B)
  - Proof immediate.
- Consider x in A.
  - Rank(x,A) = index(x) – 1.
  - Rank(x,B) can be found using binary search.
- This can be done in parallel for each x in A, and also each x in B.

Quick Example

A = [8 10 12 24]   B = [15 17 27 32]

Element	8	10	12	24	15	17	27	32
Rank in A	0	1	2	3	3	3	4	4
Rank in B	0	0	0	2	0	1	2	3
Rank in C	0	1	2	5	3	4	6	7

C = [8 10 12   15   17 24   27 32]

Analysis: Time and Work

- For each element of A and each element of B:
    - One binary search on n elements
    - Takes O(log n) time
    - Total time = O(log n) with O(n) processors.
- Total work done = O(n log n).
    - Higher than the sequential time taken.
    - Such a parallel algorithm is called non-optimal.
    - Optimal means that the work done by the parallel algorithm is asymptotically equal to the time complexity of the best known sequential algorithm.
        - In essence, can simulate the parallel algorithm as a sequential algorithm.

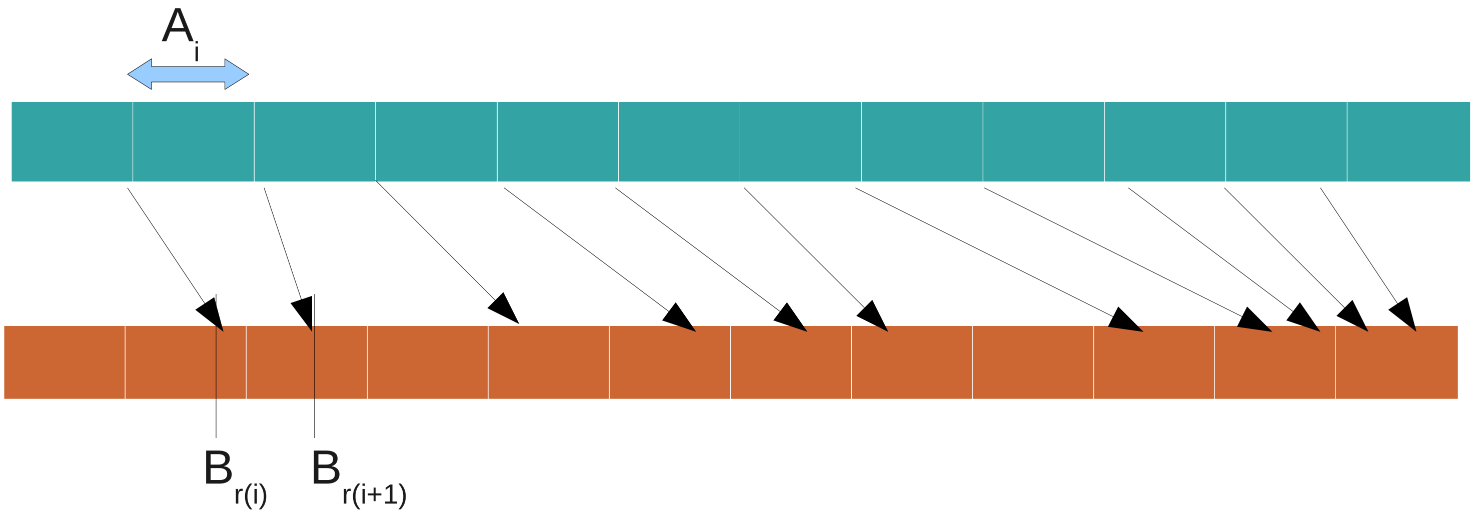
An Improved Optimal Algorithm

- General technique
  - Solve a smaller problem in parallel
  - Extend the solution to the entire problem.
- For the first step, the problem size to be solved is guided by the factor of non-optimality factor of an existing parallel algorithm.

An Improved Parallel Algorithm

- Our simple parallel algorithm is away from optimality by a factor of O(log n).
- So, we should solve a problem of size O(n/log n).
- For this purpose, we pick every log nth element of A, and similarly in B.
- Use the simple parallel algorithm on these elements of A and B.
    - Binary search however in the entire A and B.

An Improved Parallel Algorithm

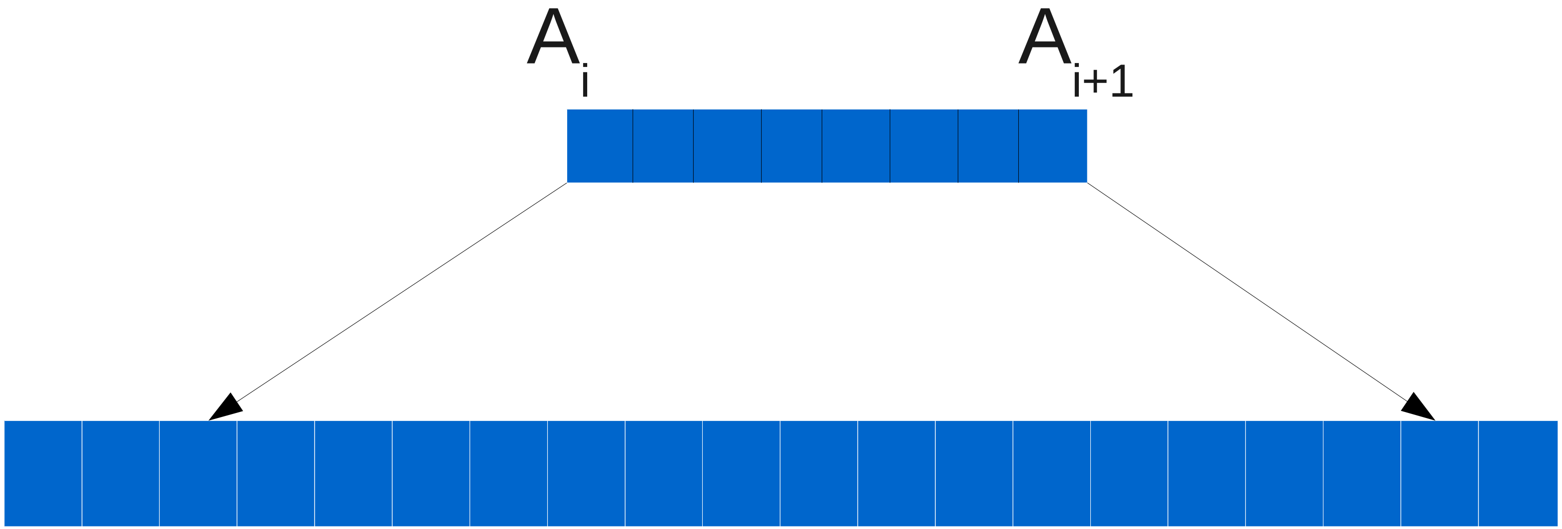


- Let A_1, A_2,…,$A_{n/\log n}$ be the elements of A ranked in B.
- These ranks induce partitions in B.
    - Define $[B_{r(i)} \ldots B_{r(i+1)}]$ as the portion of B so that [A(i)...A(i+1)] have ranks in.
- Can therefore merge [A(i)...A(i+1)] with $[B_{r(i)} \ldots B_{r(i+1)}]$ sequentially.

An Improved Parallel Algorithm

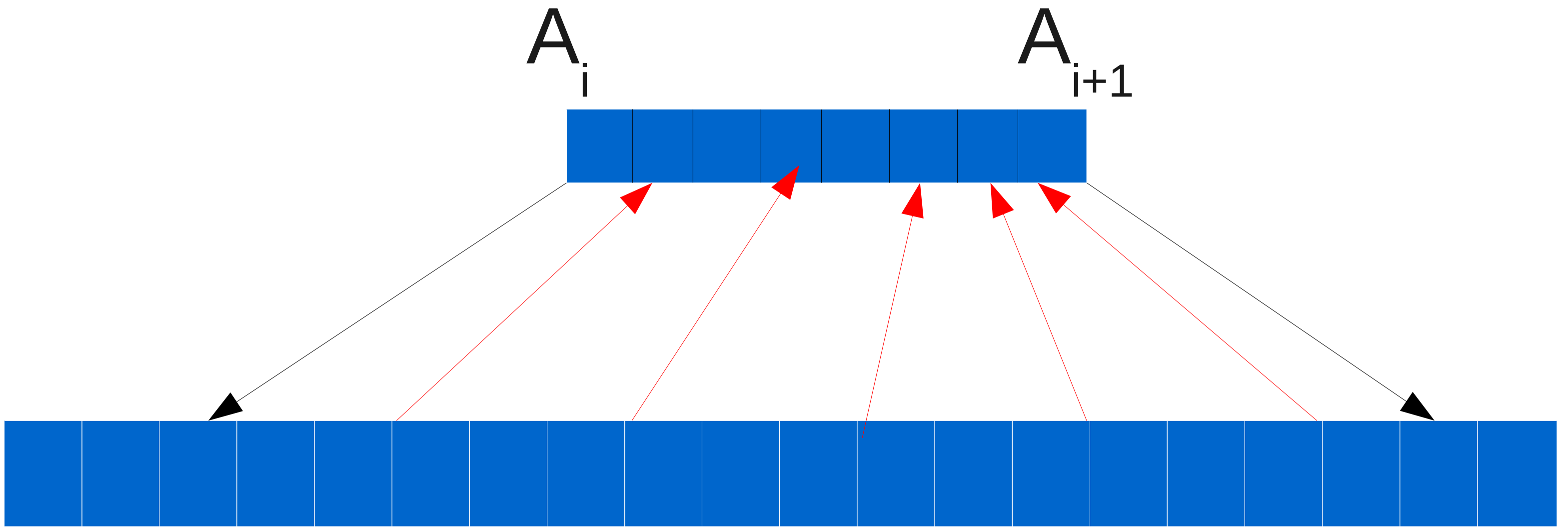
- Such sequential merges can happen in parallel, at each index of A[i].
- Time taken for the sequential merge is O(log n + $B_{r(i+1)} - B_{r(i)}$).
- Time:
  - Binary search: O(log n), with n/log n processors.
  - Sequential merge: O(log n), subject to certain conditions. There are also n/log n such merges in parallel.
- Work:
  - There are n/log n binary searches in parallel. Work = O(n).
  - For the sequential merges too, work = O(n).

An Improved Parallel Algorithm



- What if $[B_{r(i)}...B_{r(i+1)}]$ has a size of more than log n?
- The situation can be addressed
  - Pick equally spaced, no more than log n, spaced items in $[B_{r(i)}...B_{r(i+1)}]$.
  - Rank these in $[A_i...A_{i+1}]$.

An Improved Parallel Algorithm



- What if $[B_{r(i)} \ldots B_{r(i+1)}]$ has a size of more than log n?
- The situation can be addressed
  - Pick equally spaced, no more than log n, spaced items in $[B_{r(i)} \ldots B_{r(i+1)}]$.
  - Rank these in $[A_i \ldots A_{i+1}]$.

Final Result

- Can merge two sorted arrays of size n in time O(log n) with work O(n).
  - Need CREW model, for binary searches.
- Can improve further, we will see later.
- The technique to achieve optimality is a general technique, with several applications. We will see more applications of this later.

Some Administrative Issues

- HW 5 posted on the web today. Due in a week's time.
- HW4 grading will be returned by then.
- Revised mid1 scripts also will be returned by then.