

CSCE 5160 Parallel Processing

NO CLASS Tuesday October 27

HW#6 (Due Oct 29) 8.7, 8.25, 9.1

Review

Solving linear equations (dense matrices)

Gaussian Elimination technique

Iterative methods

Jacobi

Conjugate Gradient method

Sorting algorithms

Quick Sort

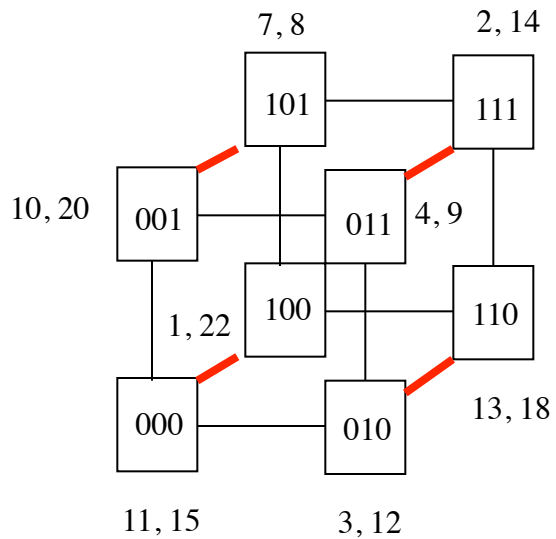
Parallelization (recursively double number of processes)

Using CRCW model

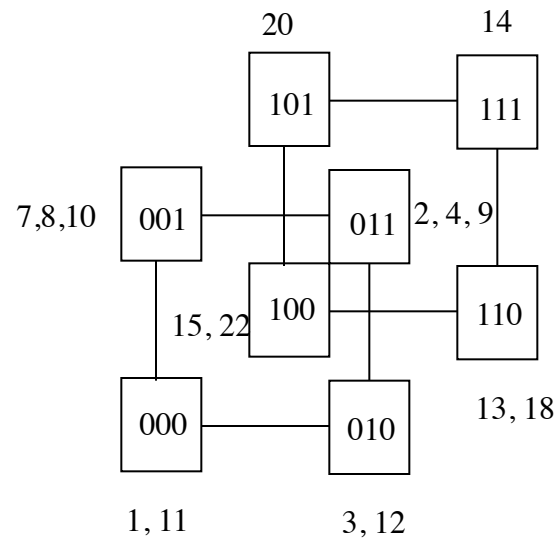
$O(1)$ complexity for partitioning lists

Implementation using Hypercubes

CSCE 5160 Parallel Processing

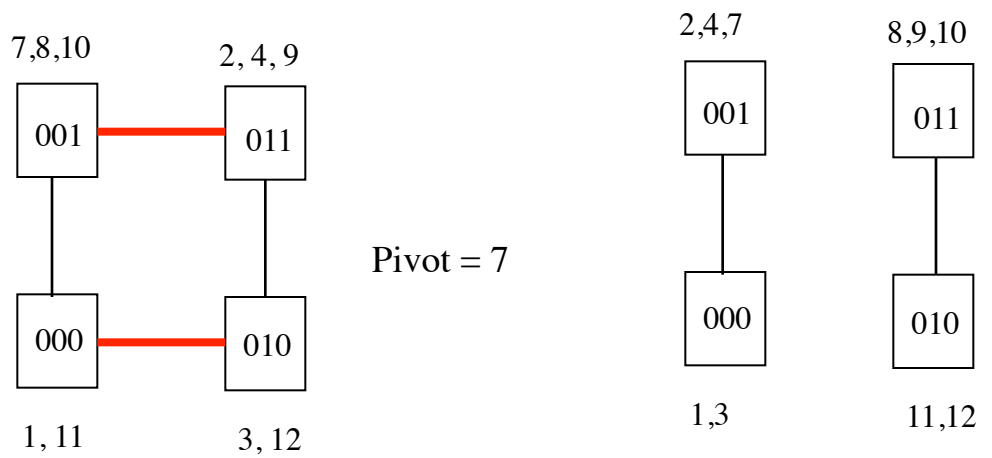
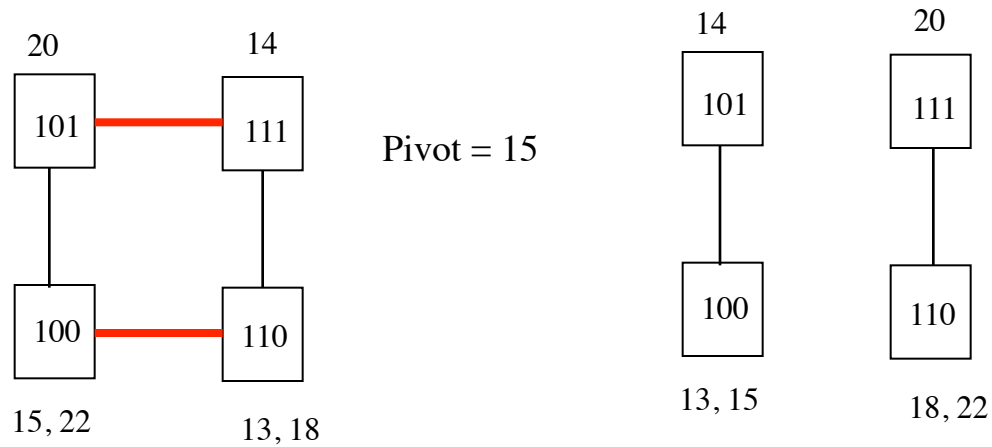


Pivot = 12



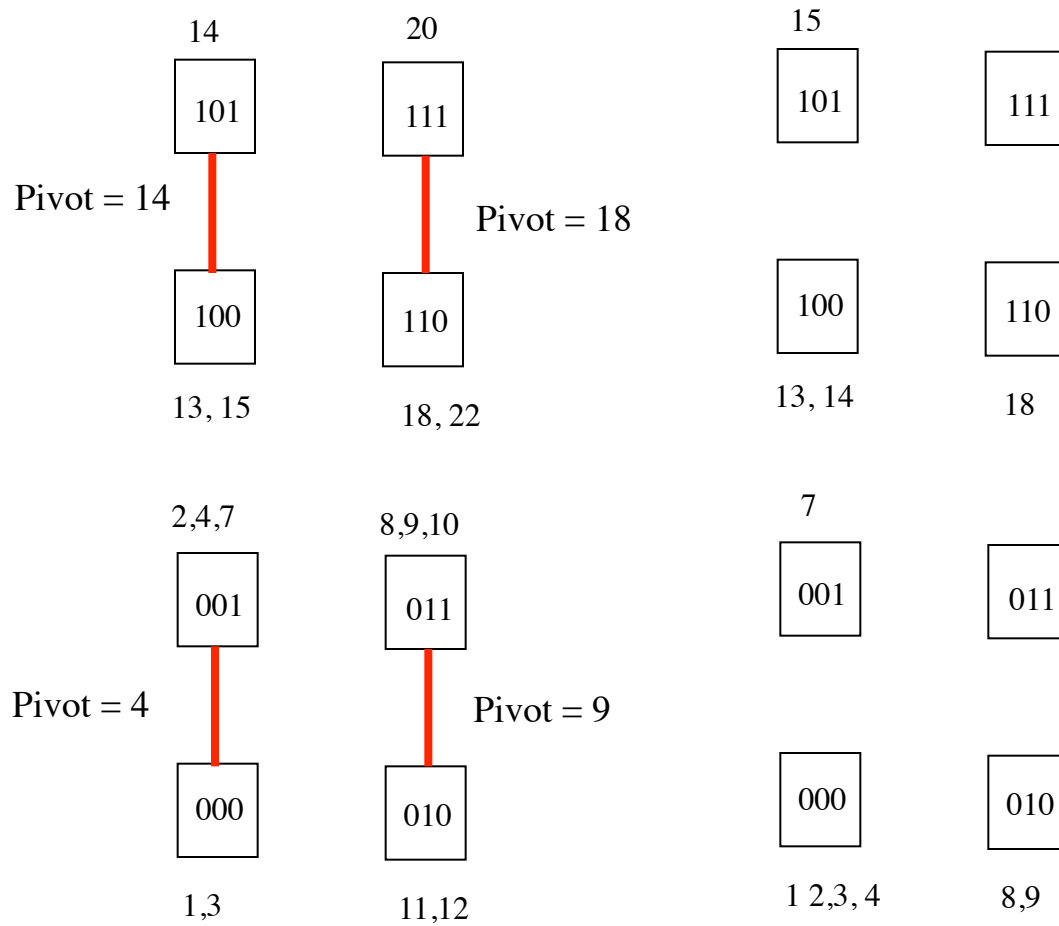
Now we have two sub-cubes; all processors in one sub-cube have elements smaller than or equal to the pivot, and the other sub-cube has all elements greater than the pivot.

CSCE 5160 Parallel Processing



Now we have 4 sub-cubes
each with two elements

CSCE 5160 Parallel Processing



Now, we have sorted list by reading the elements with processors according to their binary labels.

CSCE 5160 Parallel Processing

Selecting a Pivot at each step is very critical to the performance.

Otherwise, some processor may be idle. Consider the case where we selected the largest number as pivot in the first stage?

Half of the processors will be idle after that step.

If numbers are distributed uniformly, we can take a median as our pivot at each step.

How do we find a median? Each processor maintains its n/p elements in sorted order -- we can use sequential quick sort for this.

After first step we just use merge the elements exchanged by messages.

Complexity?

The algorithm performs d iterations (where d is the dimension of hypercube).

In each iteration we have

- (a) pivot selection
- (b) Broadcast the pivot
- (c) split the numbers

CSCE 5160 Parallel Processing

Communication: broadcasting the pivot in step $i = O(d-(i-1))$

So total cost for all $d = \log p$ steps

Total cost of this broadcast $= O(\log^2 p)$

Each processor must divide its list into two lists. Let us assume that each local list will be sorted, so we have a computation cost of $n/p \log(n/p)$ -- for sorting local lists

Then dividing the local list into two lists is simple (one unit of computation).

And then exchange these lists with its neighbor in the i th dimension (in i th iteration)

This communication costs $O(n/p)$ — since each processor has at most n/p elements and all of these elements may have to be exchanged

And since we have d iterations, total communication cost for this step

$= O(n/p * \log p)$

So total complexity = $O[(n/p) * \log(n/p)] + O[(n/p) * \log p] + [O(\log^2 p)]$

CSCE 5160 Parallel Processing

We now look at another common sequential algorithm and see if we can parallelize it.

Simplest: Bubble sort

```
for (i = 0; i < n-1, i++)
  for (j = i + 1, J < n, J++)
  {
    if (A(j) < A(i))
    {
      Temp = A(i);
      A(i) = A(j);
      A(j) = Temp;
    }
  }
```

How do we parallelize this algorithm?

Sequential Complexity = $O(n^2)$

CSCE 5160 Parallel Processing

ODD-EVEN Bubble sort.

Assume that we have n processors to sort n elements. We will alternate between the ODD and EVEN phase.

In ODD phase, all **odd numbered** processors **compare & exchange** elements with their “right” neighbor (that is processor i exchanges with processor $i+1$)

In EVEN phase, all **even numbered** processors **compare & exchange** elements with their right neighbor (again, processor i exchanges with processor $i+1$)

We need a total of n steps ($n/2$ odd and $n/2$ even steps)

How much does the “compare & exchange” cost? 1 unit if no communication

Otherwise, depends on how processors can communicate.

If we assume that the i and $i+1$ are next to each other and if we can communicate in parallel, the communication should be $O(n)$

CSCE 5160 Parallel Processing

What about implementing ODD-EVEN sort on a ring?

At both ODD and EVEN phase, processor i communicates with processor $i+1$. If the ring is bi-direction, this costs only 1 unit in communication (assuming one element per processor).

What about if we are using a Hypercube?

How do we assure that at each step (ODD and EVEN), the processors in Hypercube are neighbors?

Gray-coding for mapping rings on hypercubes?

Gray codes encode consecutive integers as binary where consecutive numbers differ in only one bit

0	0000	1	0001
2	0011	3	0010
4	0110	5	0111
6	0101	7	0100

CSCE 5160 Parallel Processing

Gray-coding for mapping rings on hypercubes?

This way, processor numbered i will always be a neighbor of processor $i+1$ on the hypercube.

What about 2-D Mesh?

More difficult to assure and processor i will always be a neighbor of processor $i+1$.

What if we have $n > p$? Each processor is assigned n/p elements.

We start with a local sort of the n/p elements on each processor using Quick Sort

$$O(n/p \log(n/p))$$

The compare-and-exchange now requires a comparison with the two sorted lists

$$O(n/p) \text{ comparisons.}$$

How many ODD-EVEN steps are needed = p ($p/2$ odd and $p/2$ even).

$$\text{So total cost of comparisons} = O(p * n/p) = O(n)$$

CSCE 5160 Parallel Processing

What about communication? Each step involves n/p elements being communicated over one communication link -- $O(n/p)$

Since we have p steps, total cost of communication = $O(n)$.

Total cost = $O[n/p * \log (n/p)] + O(n) + O(n)$

Is this cost optimal?

only if $p = O(\log n)$ -- in other word, we can use only very small number of processors

Consider a shared memory implementation

All array elements in shared memory.

We can still consider Odd – Even but now processor i (with $a[i]$) will compare with $a[i+1]$ and exchange $a[i]$ with $a[i+1]$

Do we need barriers?

Yes, to make sure all processors are done with “compare and exchange” in each ODD and EVEN phase

CSCE 5160 Parallel Processing

Other Sorting Algorithms

Enumeration Sort: -- similar to midterm problem from previous semester

For each a_i find how many elements are smaller than a_i . This is known as the rank of a_i

Can we find out for any element a_i , its position (or **rank**) in the final sorted order, without performing a compare and exchange?

Consider $n*n (=n^2)$ processors to sort n elements.

In order to find the rank of one element $a[i]$, we use n processors.

Each of these n processors will compare $a[i]$ with one other element $a[j]$.

Using atomic (or reduction) we “sum” on all the n processors together count how many other elements are smaller than $A[i]$.

Thus, with n^2 processors, we can obtain the rank for each of the n elements in $O(1)$

Let us consider coding this algorithm

CSCE 5160 Parallel Processing

```
/* initialize
```

```
  #pragma parallel for  
  for (j=0;j<n; j++) c[j]=0;
```

```
/* Note we use one row of processors here
```

```
/* find the rank of a[i]
```

```
  #pragma parallel for  
  for (i=0; i<n; i++)  
    #pragma parallel for  
    for (j=0;j<n; j++)  
      if (a[i]<= a[j])  
        #prgma omp atomic  
        c[j]= c[j]+1;
```

```
/* if atomic doesn't work use reduce  
/* Note c[j] is the rank of a[j]
```

```
  #pragma parallel for  
  for (j=0;j<n; j++)  
    a[c[j]]=a[j];
```

```
/* Store a[i] at correct place
```

CSCE 5160 Parallel Processing

Bucket Sort.

Suppose we know the total range of the values we are dealing with.

Identify m sub-ranges, **called buckets**

Place all elements into these buckets (should take no more than n time units for sequential algorithm)

Then sort each bucket. -- **We can sort each bucket using bucket sort again**

Complexity; Placing in buckets = $O(n)$,

Sorting each bucket = $n/m \log (n/m)$ if we use quick sort

= $O(\log(n/m))$ if we use bucket sort recursively

If $m = O(n)$, then serial Bucket sort has a complexity of $O(n)$.

How to parallelize this?

CSCE 5160 Parallel Processing

Parallelization of Bucket Sort. Consider p processors. Each processor is assigned n/p elements.

Step 1: We identify p sub-ranges for our elements (or p buckets).

Step 2: Each processor divides its n/p elements into p buckets

Step 3: All processors send their p buckets to appropriate processors

At the end of this communication step, each processor has a single bucket to deal with

Step 4: All processors sort their buckets -- we can use sequential version of bucket sort at each processor

Parallel Complexity;

Let us compute the complexity of this algorithm on Hypercube. The communication needed for Step 3 is all-all-gather messages.

Step 2; $O(n/p)$

Step 3: $O((n/p + (p \log p)))$

Step 4; $O(n/p)$

CSCE 5160 Parallel Processing

Radix Sort: If we assume that our elements are represented as binary, we can examine r -bits at a time, starting with most significant r -bits.
Sort the elements using r -bits only.

We can consider this similar to bucket sort.

The most significant r -bits first define 2^r buckets.

Within each bucket, using next r -significant bits we create 2^r buckets again.

We repeat this until we do this for all bits.

How to parallelize Radix Sort

Let us consider the case with n processors sorting n elements.

Consider an example

CSCE 5160 Parallel Processing

An Example.

consider 4 numbers and 4 processors

0100 0001	P1
0001 0001	P2
0100 0100	P3
0010 0100	P4

Let us look at 4-bits at a time, from least significant to most significant

We examine the elements at each processor in parallel for $j = 0..3$

for $j = 0$, two processors P1 and P2 set flag. Use **prefix sum**,

P1 has an index value of 1, and 2 has an index value of 2

P1's rank is 1 and P2's rank is 2

For $j = 1$, no processors set a flag

for $j = 2$, two processors (P3 and P4) set flags. prefix sum of P3 = 1, and prefix sum of P4=2

P3's rank = 3 and P4's rank = 4

Exchanging elements based on ranks, in this example changes no elements.

CSCE 5160 Parallel Processing

0100 0001	P1
0001 0001	P2
0100 0100	P3
0010 0100	P4

Continuing with next significant 4 bits,

for $j=0$, only processor P2 sets a flag; its prefix and rank = 1

for $j=1$, only processor P4 sets a flag; its prefix sum=1 and rank =2

for $j=3$, both P1 and P3 set flags; prefix sum of P1 = 1, rank of P1 = 3

prefix sum of P3 = 2, rank of P4 = 4

for $j=4$, no processor sets a flag

Now exchanging elements based on ranks will yield

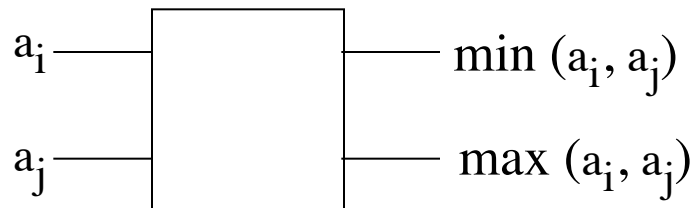
0001 0001	P1
0010 0100	P2
0100 0001	P3
0100 0100	P4

CSCE 5160 Parallel Processing

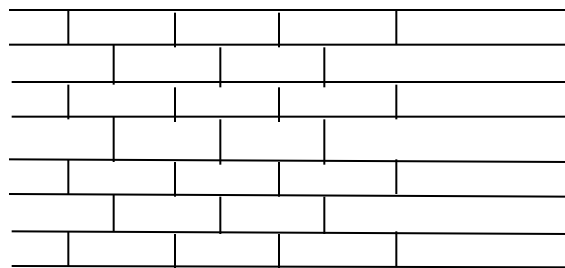
Sorting Networks

Consider how we used ODD-EVEN sort method, We can think of building a sorting network based on this idea. In a sorting network, we have hardware comparators.

For example we can build a simple comparator that takes 2 inputs and produces two outputs as below.



Now let us build a sorting network for 8 elements.



How many comparators are needed?

We have $n/2$ per column and $O(n)$ columns)

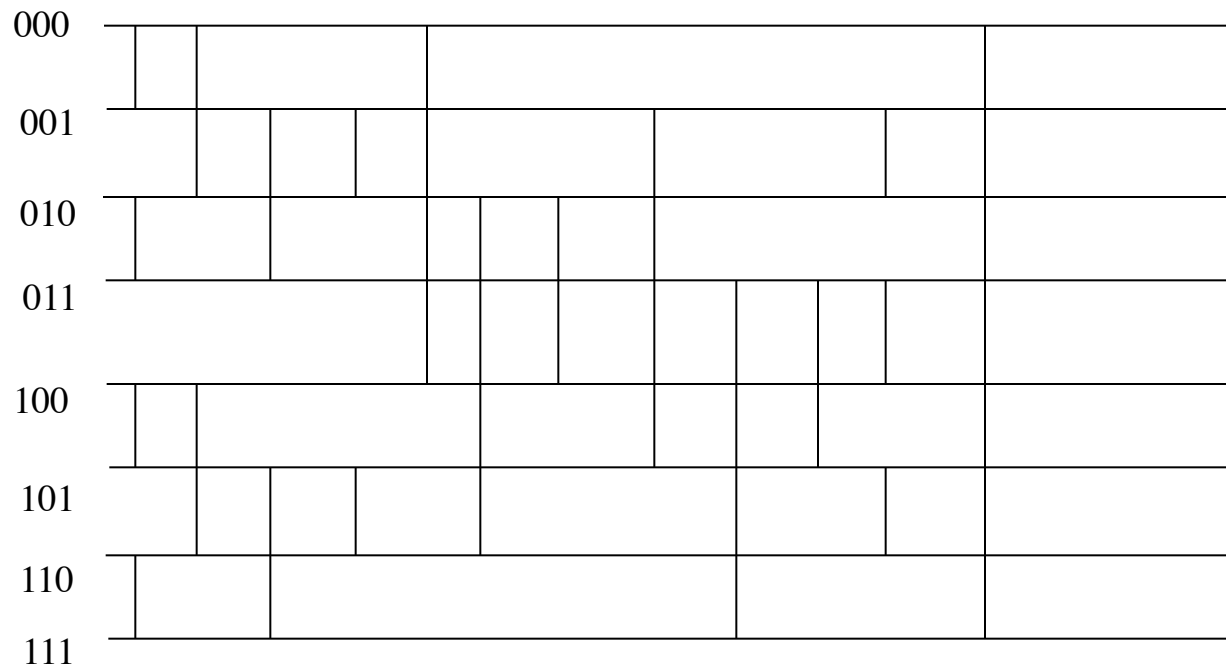
So we need a total of $O(n^2)$ Comparators

Can we come up with other ways of building sorting networks?

CSCE 5160 Parallel Processing

Is it possible to think of other ways of constructing Sorting networks?

Yes. Consider the Batcher's sorting network.



In general we can come with any number of combinations -- people have used Genetic algorithms to find out what is the minimum number of comparisons needed.