ECE 432/532
Programming for Parallel Processors

Sorting

- n keys and p = comm sz processes.
- n/p keys assigned to each process.
- No restrictions on which keys are assigned to which processes.
- When the algorithm terminates:
  - The keys assigned to each process should be sorted in (say) increasing order.
  - If $0 \leq q < r < p$, then each key assigned to process q should be less than or equal to every key assigned to process r.

Serial bubble sort

```
void Bubble_sort(
      int   a[]  /* in/out */,
      int   n    /* in     */) {
   int list_length, i, temp;

   for (list_length = n; list_length >= 2; list_length--)
      for (i = 0; i < list_length-1; i++)
         if (a[i] > a[i+1]) {
            temp = a[i];
            a[i] = a[i+1];
            a[i+1] = temp;
         }

}  /* Bubble_sort */
```

6 5 3 1 8 7 2 4

Serial bubble sort

- Is it good for parallel implementation?

- Example, you have the numbers 9, 5, 7
  - How would you sort them in serial way?
  - How would you sort them in parallel way?

- The order in which the “compare-swaps” take place is essential to the correctness of the algorithm

Odd-even transposition sort

- A sequence of phases.
- Even phases, compare swaps:

$$(a[0], a[1]), (a[2], a[3]), (a[4], a[5]), \ldots$$

- Odd phases, compare swaps:

$$(a[1], a[2]), (a[3], a[4]), (a[5], a[6]), \ldots.$$

Example

Start: 5, 9, 4, 3

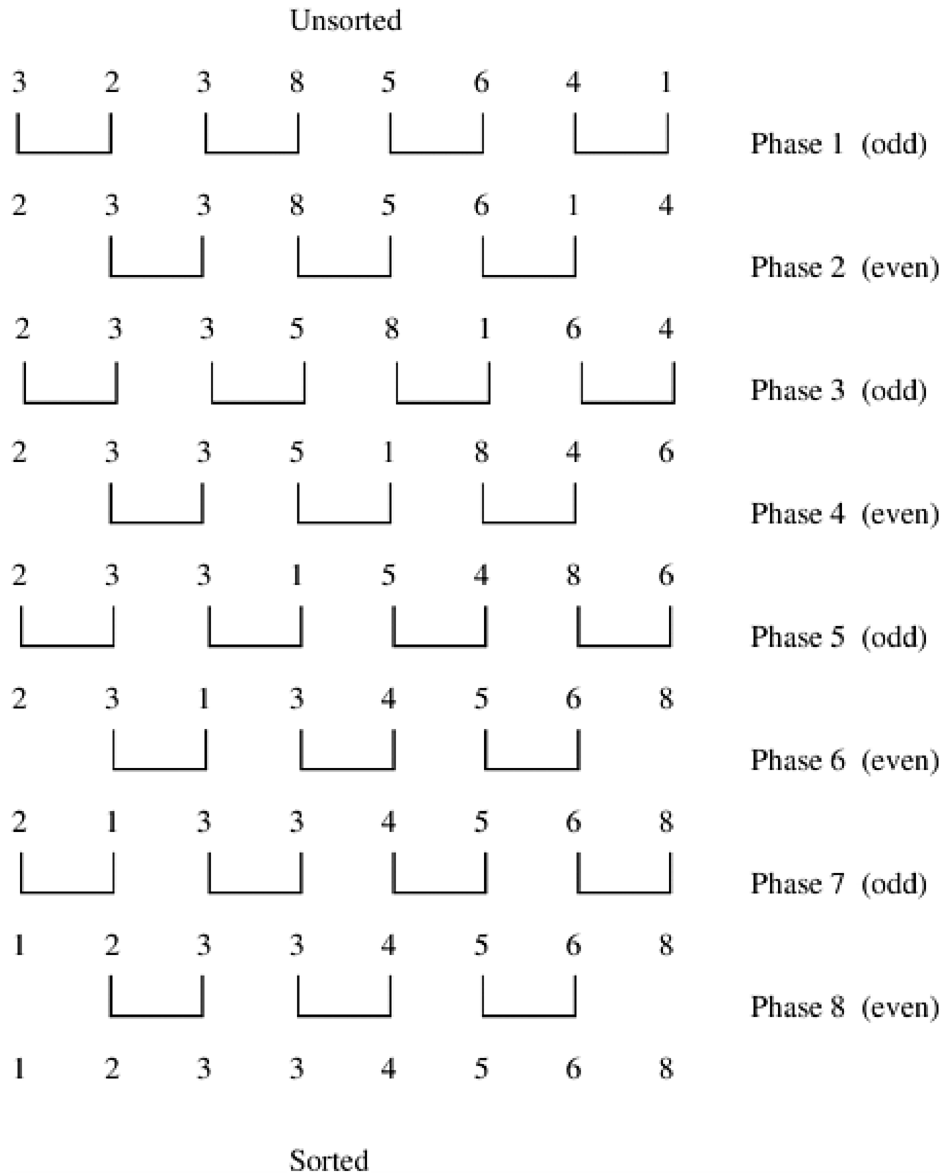
Even phase: compare-swap (5,9) and (4,3)
getting the list 5, 9, 3, 4

Odd phase: compare-swap (9,3)
getting the list 5, 3, 9, 4

Even phase: compare-swap (5,3) and (9,4)
getting the list 3, 5, 4, 9

Odd phase: compare-swap (5,4)
getting the list 3, 4, 5, 9

Example


Unsorted
3 2 3 8 5 6 4 1
Phase 1 (odd)
2 3 3 8 5 6 1 4
Phase 2 (even)
2 3 3 5 8 1 6 4
Phase 3 (odd)
2 3 3 5 1 8 4 6
Phase 4 (even)
2 3 3 1 5 4 8 6
Phase 5 (odd)
2 3 1 3 4 5 6 8
Phase 6 (even)
2 1 3 3 4 5 6 8
Phase 7 (odd)
1 2 3 3 4 5 6 8
Phase 8 (even)
1 2 3 3 4 5 6 8
Sorted

Serial odd-even transposition sort

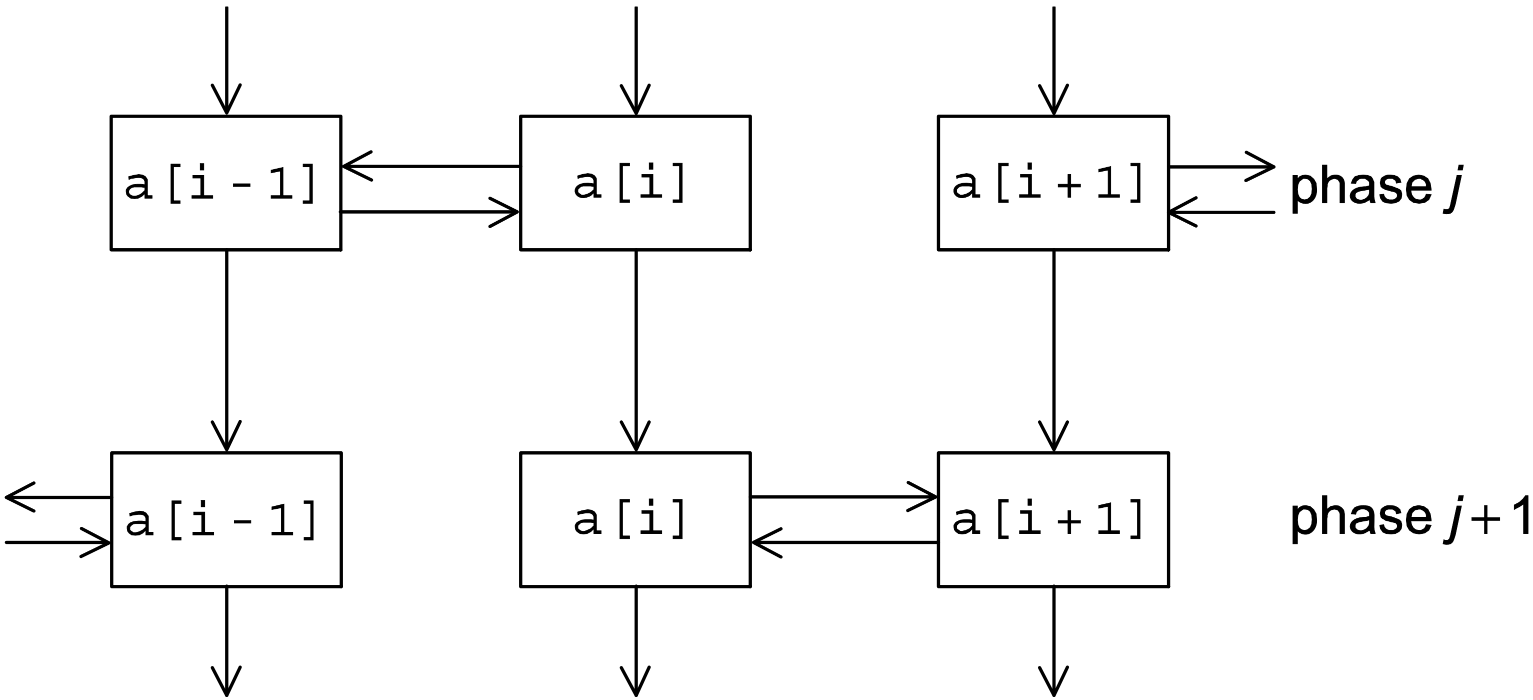
```
void Odd_even_sort(
      int  a[]  /* in/out */,
      int  n    /* in     */) {
   int phase, i, temp;

   for (phase = 0; phase < n; phase++)
      if (phase % 2 == 0) { /* Even phase */
         for (i = 1; i < n; i += 2)
            if (a[i-1] > a[i]) {
               temp = a[i];
               a[i] = a[i-1];
               a[i-1] = temp;
            }
      } else { /* Odd phase */
         for (i = 1; i < n-1; i += 2)
            if (a[i] > a[i+1]) {
               temp = a[i];
               a[i] = a[i+1];
               a[i+1] = temp;
            }
      }
}  /* Odd_even_sort */
```

Parallel odd-even transposition sort

- The odd-even transposition sort has considerably more opportunities for parallelism than bubble sort
- All of the compare-swaps in a single phase can happen simultaneously.
- Foster's methodology:
  - *Tasks:* Determine the value of a[i] at the end of phase *j*.
  - *Communications:* The task that's determining the value of a[i] needs to communicate with either the task determining the value of a[i-1] or a[i+1]. Also the value of a[i] at the end of phase *j* needs to be available for determining the value of a[i] at the end of phase *j*+1.

Communications among tasks in odd-even sort



n=p

Parallel odd-even transposition sort

Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

When will the list be sorted? After how many phases?

Pseudo-code

```
Sort local keys;
for (phase = 0; phase < comm_sz; phase++) {
   partner = Compute_partner(phase, my_rank);
   if (I'm not idle) {
      Send my keys to partner;
      Receive keys from partner;
      if (my_rank < partner)
         Keep smaller keys;
      else
         Keep larger keys;
   }
}
```

Compute_partner

```
if (phase % 2 == 0)          /* Even phase */
   if (my_rank % 2 != 0)      /* Odd rank */
      partner = my_rank − 1;
   else                       /* Even rank */
      partner = my_rank + 1;
else                         /* Odd phase */
   if (my_rank % 2 != 0)      /* Odd rank */
      partner = my_rank + 1;
   else                       /* Even rank */
      partner = my_rank − 1;
if (partner == −1 || partner == comm_sz)
   partner = MPI_PROC_NULL;
```

MPI PROC NULL is a constant defined by MPI. When it’s used as the source or destination rank in a point-to-point communication, no communication will take place and the call to the communication will simply return