

Monday, 02/06/12

Ch. 10 Design and Coding of Parallel Programs:

- 2 examples:
- Jacobi method for linear systems → next semester
- Sorting

Example of a sorting problem: State concretely with 4 processes here!

Have 100 integers from 1, 2, ..., 100 given across 4 processes.

Want to obtain these numbers sorted and equally distributed across the 4 processes, that is, 1..25 on Process 0, 26 to 50 on P1, etc.

This is modified from Padilla to make clearer and to make more realistic.

Think of these as indices into some list of large object (like of FEM triangles that are to be distributed into 4 subdomains after refinement/coarsening for purpose of load balancing).

Critical idea: Never store all integers on one process.

Very basic algorithm:

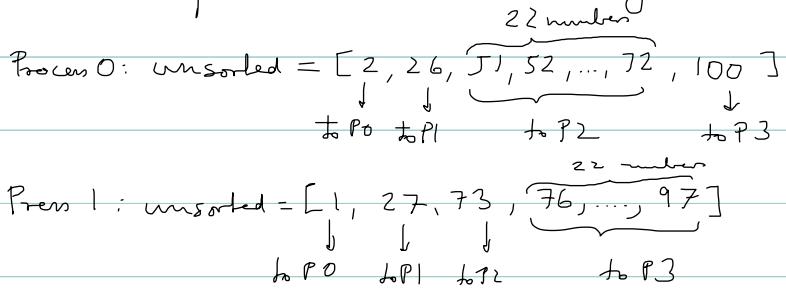
- Perform local sort of the 25 local integers
- On each process, send the integers in the range 1 to 25 to Process 0, in range 26 to 50 to P1, etc.

On each process, receive these numbers, but we do not know which process the numbers will come from.

Notice: We do know that there are only 25 numbers to be received.

- Notice that these numbers are not sorted, so need to do local sort

Example: Variable names are "unsorted" for the original 25 integers on each process and "sorted" for the 25 at the end.



Need communication among the 4 processes such that we receive:

on Process 0: sorted = [2, 1, ...]

on Process 1: sorted = [26, 27, ...]

on Process 2: sorted = [51, ..., 72, 73, ...]

on Process 3: sorted = [100, 76, ..., 97, ...]

MPI_Alltoall (for this example:

void * send-buffer,
 int send-count,

MPI_Datatype send-type, MPI_INT
 void *recv-buffer,
 int recv-count,

MPI_Datatype recv-type, MPI_INT

MPI_Comm comm)

This command is only good if you send the same number of data to each process and receive that same number from all processes!

⇒ There are variable version of all MPI Commands exactly for this purpose, here MPI_Alltoall_v, but others are

MPI_Scatter_v, MPI_Gather_v

⇒ See Appendix A in Pacheco → HW on power method

```

MPI_Alltoallv (
    void *send_buffer,
    int *send_counts,
    int *send_displacements, }  

    MPI_Datatype send_type, } array of length np,  

    void *recv_buffer, } dynamically allocated  

    int *recv_counts,
    int *recv_displacements, }  

    MPI_Datatype recv_type,
    MPI_Comm comm)

```

int *unsorted, *send_count, *send_displacement;
 int *sorted, *recv_count, *recv_displacement;
 Let n be total number of integers.

unsorted = (int) malloc ((n/np) * sizeof(int))

Sorted = — “ —

send_counts = (int) malloc (np * sizeof(int));

send_displacement = — “ —

recv_counts = — “ —

— Perform local sort \Rightarrow "unsorted" is now a sorted list of $\frac{n}{np}$ int.

for (i=0; i < np; i++) send_counts[i] = 0

for (j=0; j < n/np; j++) {

$k = (\text{unsorted}[j] - 1) / (n/np)$ /* = process id to send to */
 (send_counts[k])++

}

send_displacemt[0] = 0
for (i = 1; i < np; i++)

$$\text{send_displacemt}[i] = \text{send_displacemt}[i-1] + \text{send_counts}[i-1]$$

Now, we need the recv counts. But there is not enough info locally. Rather, the info is in all the send_counts on the other processes \Rightarrow

MPI_Alltoall (

send_counts, 1, MPI_INT,

recv_counts, 1, MPI_INT, MPI_COMM_WORLD)

recv_displacemt[0] = 0

for (i = 1; i < np; i++)

$$\text{recv_displacemt}[i] = \text{recv_displacemt}[i-1] + \text{recv_counts}[i-1]$$

MPI_Alltoallv (

unsorted, send_counts, send_displacemt, MPI_INT,

sorted, recv_counts, recv_displacemt, MPI_INT,

MPI_COMM_WORLD)

- Perform local sort of "sorted"

(In example of load-balancing of TEM: Now you commutes the actual, large data)

Contrast to Pacheco example:

(1) He allows for missing and repeated numbers \Rightarrow

Sorted might not have length $n/np (= 25)$ \Rightarrow

need to do dynamic memory re-allocating based

$$\text{on } \sum_{i=0}^{np-1} \text{recv_counts}[i] = \text{recv_displacemt}[np-1]$$

$$+ \text{recv_counts}[np-1]$$

(2) Memory as large as n might now be needed on some process \Rightarrow contradicts spirit of load-balancing