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 Pacheco to make clearer and to make more realistic

Think of these as indices into some list of large object (like of FEM tringles that are to distributed into 4 subdomains afte 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 Proc 1, 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.

Process 0: unsorted = $[\underset{\text{to P0}}{\underset{\downarrow}{2}}, \underset{\text{to P1}}{\underset{\downarrow}{26}}, \underbrace{\overbrace{51, 52, \ldots, 72}^{\text{22 numbers}}}_{\text{to P2}}, \underset{\text{to P3}}{\underset{\downarrow}{100}}]$

Process 1: unsorted = $[\underset{\text{to P0}}{\underset{\downarrow}{1}}, \underset{\text{to P1}}{\underset{\downarrow}{27}}, \underset{\text{to P2}}{\underset{\downarrow}{73}}, \underbrace{\overbrace{76, \ldots, 97}^{\text{22 numbers}}}_{\text{to P3}}]$

⋮

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,	unsorted
int	send_count,	
MPI_Datatype	send_type,	MPI_INT
void	*recv_buffer,	sorted
int	recv_count,	
MPI_Datatype	recv_type,	MPI_INT
MPI_Comm	comm)	

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

⇒ There are variable version of <u>all</u> MPI Commands exactly for this purpose, here MPI_Alltoallv, but others are MPI_Scatterv, MPI_Gatherv

⇒ 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,
                                                dynamically allocated
    void            *recv_buffer,
    int             *recv_counts,          ]
    int             *recv_displacements,   ]

    MPI_Datatype          recv_type,

    MPI_Comm          comm)
```

```
int    *unsorted , *send_counts, *send_displacements ;
int    * sorted , *recv_counts, *recv_displacements ;
Let n be total mber of integers.
unsorted = (int) malloc ( (n/np) * sizeof(int ) )
Sorted  =       — " —

send_counts = (int) malloc ( np *sizeof (int));
send_displacements =     — " —
recv_counts         =   — " —
recv_displacements =   — " —
— Perform local sort => "unsorted" is now a sorted list of n/np int.

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

for ( j = 0 ; j < n/np ; j++)  {
        k   = (unsorted [j] - 1 ) / (n/np)   /* = process id to send to */
        (send_counts [k])++
    }
```

```
send_displacements[0] = 0
for ( i = 1 ; i < np ; i++ )
    send_displacements[ i ] = send_displacements[i-1] + 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 =>

```
MPI_Alltoall (
    send_counts, 1, MPI_INT,
    recv_counts, 1, MPI_INT, MPI_COMM_WORLD )
recv_displacements[0] = 0
for ( i = 1; i < np; i++ )
    recv_displacements[i] = recv_displacements[i-1] + recv_counts[ i-1 ]
MPI_Alltoallv (
    unsorted, send_counts, send_displacements, MPI_INT ,
    sorted , recv_counts, recv_displacements, MPI_INT ,
    MPI_COMM_WORLD )
```

- Perform local sort of "sorted"

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

Contrast to Pacheco Example:

(1) He allows for missing and repeated numbers => sorted might not have length n/np (= 25) => need to do dynamic memory re-allocating based on $\sum_{i=0}^{np-1}$ recv_counts[i] = recv_displacements[np-1] + recv_counts[np-1]

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