Wed., 05/16/12:

HW6-7: some example numbers: time in HH:MM

$N \times N = 4096 \times 4096$:

4 cores	1 node
1 process	1:37
2	0:55
4	0:29

p by node	1 node
1	0:57
2	0:39
4	0:32
8	0:16

Ch. 10 Design of Parallel Algorithms

Part 1: Jacobi method = iterative method for $Ax = b$

→ idea of matrix-free method

→ We did already CG, which is much better mathematically

Issue is that Jacobi needs millions of iterations for moderately large N, such as 8,192. ⇒ There is no way to run this to convergence, even on many nodes

⇒ It is always more important to use a good numerical method than a parallel one!

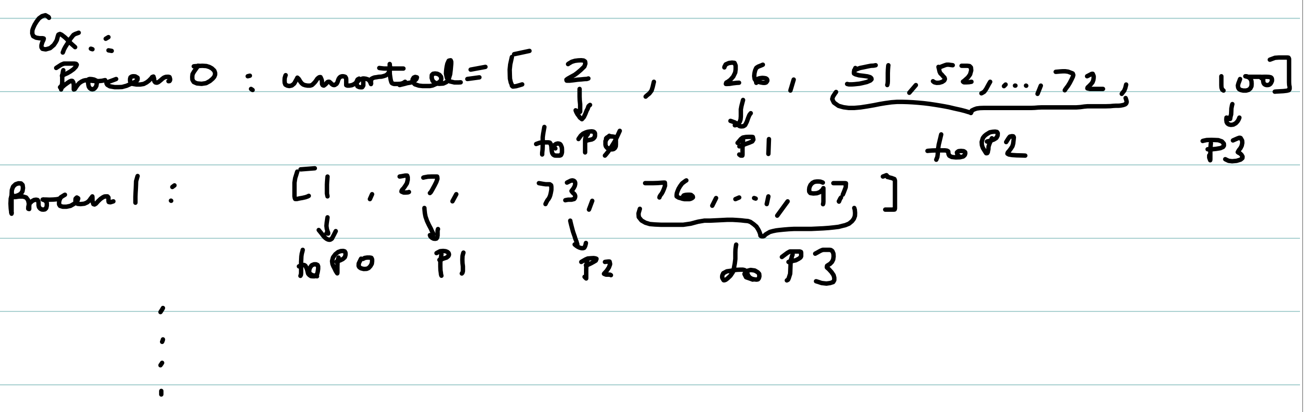
Part 2 of Ch. 10: Parallel sorting, really: introduction of some new MPI commands

Introduce MPI_Alltoall and MPI_Alltoallv,
then also know about MPI_Gatherv, etc.

→ Be aware of Appendix A.

Example: here a little different from Pacheco.
Assume $p = 4$ processes, problem: Have a total of 100 integers $1, 2, \ldots, 100$ that are given in local vectors "unsorted" of length 25 across all processes.
Want to obtain: $1, 2, \ldots, 25$ on Process 0, $26, \ldots, 50$ on Process 1, etc., in local vector "sorted".
Critical: Never assemble any vector of length > 25 anywhere!

Simplified from Pacheco: We do not allow for duplicate (or missing) numbers (like $1, 1, 3, 4, \ldots$)
$\Rightarrow$ We can allocate all vectors once, since we know their lengths.



Need communication, so that we arrive at

Proc 0 : sorted = [2, 1, ...
1 : [26, 27, ...]
2 [51, 52, ...
3 [100, 76, 77, ...]

⇒ "sorted" will <u>not</u> be sorted after the communication, even if unsorted on each process are sorted locally.

MPI_Alltoall (		above example ↓
void	*send_buffer	unsorted
int	send count	①
MPI_Datatype	send_type	MPI_INT
void	*recv_buffer	sorted
int	recv_count	②
MPI_Datatype	recv_type	MPI_INT
MPI_Comm	comm)	MPI_COMM_WORLD

① This needs to be <u>same</u> number to <u>each</u> process! That is just <u>not</u> the case. But we can determine the needed numbers from the data locally.

② We cannot determine this!

For ① and ②, need variable version

MPI_Alltoallv (
void *send_buffer
int *send_counts
int *send_displacements
MPI_Datatype send_type
void *recv_buffer
int *recv_counts
int *recv_displacements
MPI_Datatype recv_type
MPI_Comm comm)

only output is recv_buffer ; everything else is input

How long are vectors send_counts, send_displac
recv_counts, recv_displacements? np
=> need to allocate dynamically

Write code :

```
int *unsorted, *send_counts, *send_displacements
int *sorted, *recv_counts, *recv_displacements
/* np divides n = 100 assumed ! */
l_n = n/np
unsorted = allocate_int_vector ( l_n )
sorted = . . . .
send_counts = allocate_int_vector(np)
recv_counts =        - " -
send_displacements =   -||-
recv_displacements =   -||-
```

Perform local sorting => now have unsorted
sorted, as in example above.

```
for (i=0 ; i < np ; i++)  send_counts [i] = 0
for ( j=0 ; j < l_n ;  j++ )
        (send_counts [ (unsorted[j]-1)/l_n ])++
send_displacements [0] = 0
for  (i = 1 ; i < np ; i++)
     send_displacements [i] =
        send_displacements [i-1] + send_counts [i-1]
```

② We need to get the recv_counts !

```
MPI_Alltoall ( send_counts , 1, MPI_INT,
               recv_counts , 1, MPI_INT,
               MPI_COMM_WORLD )
```

recv_displacements [0]

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

Perform local search on "sorted" !

Bigger message: There are "V" version of other collective commands like MPI_Gatherv MPI_Scatterv , etc.

See man page for syntax

See also Appendix A , for instance to locate command that is nearly the right one, then see if other ones exist in that section that are better.

Ch. 6 Derived Datatypes = Grouping Data

This will explain, why send_type can be different from recv_type.

Let $A \in \mathbb{R}^{n \times n}$ be in 1-D column-oriented array storage (like A in power method)

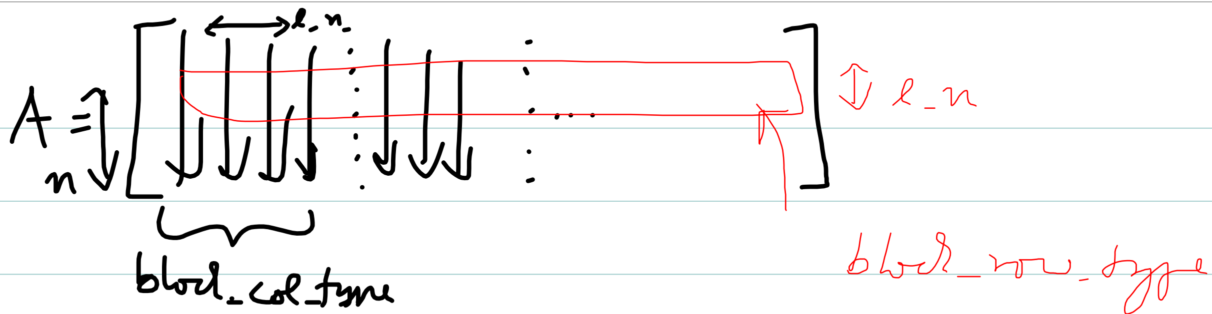
Let A be given on Process 0 and want to get $l_A \in \mathbb{R}^{n \times l_n}$ on each process.

You can do this with

```
MPI_Scatter ( A , 1, block_col_type,
              l_A, (n*l_n) , MPI_DOUBLE,
              0 , MPI_COMM_WORLD )
            root
```

Here, block_row_type is defined <u>at run time</u>

MPI_Type_vector(		this example
int	count	l_n
int	blocklength	n
int	stride	1
MPI_Datatype	el_type	MPI_DOUBLE
MPI Datatype	*new_mpi_t)	&block_col_type



MPI_Type_vector (n , l_n , n ,

MPI_DOUBLE, & block_row_type)

each block needs to be consecutive

Above block_col_type has all data consecutive
=> could have done as 1 block of length n*l_n
Or in fact of course, we could avoid derived datatype altogether, if data is consecutive!

But for MPI_Scatter (A, 1, block_row_type
the data is not consecutive and this derived datatype useful → Unfortunately need to fix the block_row_type as in Sec. 8.4.5 with MPI_UB !

Generally, regarding use of different send_type and recv_type : The criterion is that their type signatures have to match