Monday 12/19/11:

Last time: Pacheco Ch. 3

MPI_Bcast, MPI_Reduce, MPI_Allreduce

Today: example of Allreduce that divides up vector on several parallel processes.

Sec. 3.5 Dot Product:

Mathematically, $x, y \in \mathbb{R}^n$, column vectors

How to split them onto parallel processes?

One option: cyclic distribution

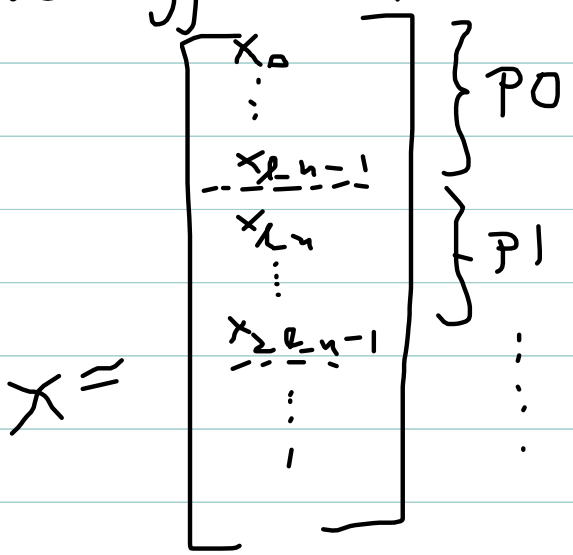
$$x = \begin{bmatrix} x_0 \leftarrow \text{on } P0 \\ x_1 \leftarrow P1 \\ x_2 \leftarrow P2 \\ x_3 \leftarrow P0 \\ x_4 \leftarrow P1 \\ x_5 \leftarrow P2 \\ \vdots \quad \vdots \\ \end{bmatrix}$$

for np = 3 processes

Note: Consecutive access to memory is the key to good performance on today's computers that have (L1, L2, L3) caches.

Caching means that the CPU assumes that it will need x_{i+1} next, if you accessed x_i now; so it pre-loads x_{i+1} ("caching").

Therefore, block distribution is both easier and more efficient.



$l\_n = \frac{n}{p}$ = local_n elements of $x \in \mathbb{R}^n$ on each process.

Issues: $x, y \in \mathbb{R}^n$ => We really only want to define their local portions $l\_x, l\_y \in \mathbb{R}^{l\_n}$

Static:

```
double l_x[l_n], l_y[l_n];
```

Dynamic:

```
double *l_x, *l_y;
l_x = malloc( l_n * sizeof(double) )
```

(l_n * sizeof(double)) = number of bytes to be allocated

malloc does not initialize the memory.
≡
calloc sets it to 0 and has different syntax:

```
l_x = calloc( l_n, sizeof(double) )
```

→ check man pages
→ also notice any header files to be #include'd ?

Then use either of these vectors as l_x[l_i]

for dynamically allocated vectors, remember to free them:

free(l_x)

Problem: $x, y \in \mathbb{R}^n$, want $d = x^T y = \sum_{i=0}^{n-1} x_i y_i$

Serial code:

```
double serial_dot(double *x, double *y, int n) {
    double d;
    int i;
    d = 0.;
    for (i=0; i<n; i++) {
        d += x[i] * y[i];
    }
    return d;
}

double parallel_dot(double *l_x, double *l_y, int l_n)
    double l_d, d;
    l_d = serial_dot(l_x, l_y, l_n);
    d = MPI_Allreduce(&l_d, &d, 1, MPI_DOUBLE,
                      MPI_SUM, MPI_COMM_WORLD)
    return d;
}
```

Sec. 5.7 Gather, Scatter:

Have $l\_x \in \mathbb{R}^{l\_n}$ locally, want to assemble x on Process 0 for test output.

```
MPI_Gather(
    void *  send_data
    int     send_count
    MPI_Datatype  send_type
    void *  recv_data
    int     recv_count
    MPI_Datatype  recv_type
    int     root          <- process to send to
    MPI_Comm      Comm )
```

```
MPI_Gather( l_x, l_n, MPI_DOUBLE,
            x, (l_n), MPI_DOUBLE, 0,
            MPI_COMM_WORLD )
```

Notes:

- receive counts = number of elements from each of the processes
  here: l_n (and not n, despite $x \in \mathbb{R}^n$)
- This is a collective communication $\Rightarrow$ All processes must execute this command $\Rightarrow$ x must be defined on all processes, even though it is used and valid on the root process $\Rightarrow$ only allocate memory for x on Rank 0:

```
double *x;
if (id == 0)
    x = malloc( n * sizeof(double) )
```

MPI_Scatter is inverse operation of Gather ⇒

calling sequence same with reversed roles of send, recv.

What if you want to assemble l_x into x and have it available on all processes?

Option 1: MPI_Gather to P0, as above.

MPI_Bcast(x, n, MPI_DOUBLE, 0, MPI_COMM_WORLD)

Option 2:

MPI_Allgather → just like Gather, but without root process, since result goes to all processes

Next time: How to define and use matrices, not following Pacheco, since we want to interface with other software (BLAS, LAPACK) and take full control of memory allocation ourselves.

Lab 3 tomorrow = HW 4 and 5 :

Make utility : Unix command "make" to compile,

link, etc., of all your code to get an executable.

→ Download files with HW 4 !!

How to compile and link if your code is contained in 2 files:

```
mpicc file1.c -c
mpicc file2.c -c
/* -c = compile only.
   output file name file1.o, file2.o automatically */
mpicc file1.o file2.o -o executable-name
```