Wednesday 02/08/12:

Ch. 7, Sec. 7.2, 7.8 Fox's Algorithm

Fox: $C = AB$, $A, B, C \in \mathbb{R}^{n \times n}$

with all matrices in block-cyclic storage = block matrices

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \cdots \\ A_{10} & A_{11} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

each block is $l\_n \times l\_n$

$l\_n = \frac{n}{q}$, $p = q^2$ parallel process

Goal: Never assemble any matrix larger than $l\_n \times l\_n$ on any process.

Pacheco $n = 4$, $q = 2$

$$AB = \left[\begin{array}{cc|cc} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & \cdots & \\ a_{20} & a_{21} & & \\ a_{30} & a_{31} & & \end{array}\right] \left[\begin{array}{cc|cc} b_{00} & b_{01} & \cdots & \\ b_{10} & & & \\ \hline \cdots & & \cdots & \end{array}\right] = \left[\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right] \left[\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right]$$

$$= \left[\begin{array}{c|c} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} + a_{03}b_{30} & \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} + a_{13}b_{30} & \\ \hline & \end{array}\right]$$

$$= \left[\begin{array}{c|c} A_{00}B_{00} + A_{01}B_{10} & \\ \hline & \end{array}\right]$$

$$\left[\begin{array}{cc} a_{00}b_{00} + a_{01}b_{10} & \cdots \\ \cdots & \cdots \end{array}\right] \qquad \left[\begin{array}{cc} a_{02}b_{20} + a_{03}b_{30} & \cdots \\ \cdots & \cdots \end{array}\right]$$

$$= C = \left[\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right]$$

⇒ Message: block-matrix arithmetic works as expected.

⇒ Viewpoint is that each parallel process in a Cartesian process grid holds block matrices A_{ij}, B_{ij}, C_{ij} of size $l\_n \times l\_n$.

Idea of Fox: Compute C_{ij} in place on the local process by moving/copying needed data of A_{ik}, B_{kj}, $0 \le k < n'$ with $n' = n/l\_n$ $\Leftrightarrow$ A, B, C have $n' \times n'$ blocks of size $l\_n \times l\_n$

$C_{ij} = A_{i0}B_{0j} + A_{i1}B_{1j} + \cdots + A_{ii}B_{ij} + \cdots + A_{in'-1}B_{n'-1j}$

Idea is to compute each of these terms on Process (i,j) in the Cartesian process grid. We start with the term $A_{ii}B_{ij}$, that is, we start such that B_{ij} is available on the process and then proceed like

$C_{ij} = A_{ii}B_{ij} + A_{ii+1}B_{i+1j} + \cdots + A_{in'-1}B_{n'-1j} + A_{i0}B_{0j} + \cdots + A_{ii-1}B_{i-1j}$

Here, I used (i,j) to denote Process id in the Cartesian grid.

Moreover, in code, l_A, l_B, l_C that denote the data stored on Process (i,j).

There l_A, l_B are A_{ij}, B_{ij} initially and also at the end of the algorithm, but in the intermediate stages, we will overide with A_{ik}, B_{kj} $0 \le k < n'$.

Assume we use BLAS function dgemm for the local products $l\_C \mathrel{+}= l\_A * l\_B$ with the accumulation in "+=" $\rightarrow$ Notice that dgemm accumulates, so we will use that.

At start, $l\_A = A_{ij}$, $l\_B = B_{ij}$.

We need A_{ii} on Procesr (i,j) to start the first local product $l\_C \mathrel{+}= l\_A * l\_B \Leftrightarrow C_{ij} = A_{ii} B_{ij}$.

Notice that A_{ii} is need in this first stage on all processes in the i-th row of the process grid.

$\Rightarrow$ this is a broadcast within each row communicator. The root process is i = row id in the Cartesian process grid.

MPI_Bcast(l_A, (l_n*l_n), MPI_DOUBLE, i, row_comm)

Then perform local product

dgemm ('N', 'N', l_n, l_n, l_n, 1.0, l_A, l_n, l_B, l_n, 1.0, l_C, l_n); $\Leftrightarrow$

$l\_C = \alpha\, lA\, lB + \beta\, l\_C$ with $\alpha = \beta = 1.0$

We realize now that there needs to be some index for the stage of the algorithm, so that we can express A_{ik} and B_{kj} somehow.

$$C_{ij} = A_{ii+0} B_{i+0,j} + A_{ii+1} B_{i+1,j} + \cdots + A_{ik} B_{kj} + \cdots$$

produce these from mod operation $\%q$

$\Rightarrow$ root of Bcast $= (i+s) \% q$

Now s = stage index with `for(s=0; s<q; s++)`
we look at B_{ij} communication. In stage s,
Procesr (i,j) needs $B_{i+s\,j}$. $\Rightarrow$ All there B_{kj}
live in the same column communicator.
$\Rightarrow$ Realise Procesr (i,j) needs $B_{i+s\,j}$ which is
'below' it in the same column, and the procesr
'above' needs the current data in l_B.
$\Rightarrow$ So, throughout the q stages, we need
to 'shift' l_B up within the column
communicator and wrap around from top
to bottom (like in the ringsend on HW)
Most convenient MPI command for this
is MPI_Sendrecv_replace(B, (l_n*l_n),
MPI_DOUBLE, dest, source, Col_Comm)

$(i-1+q)\%q$ $(i+1)\%q$

↑ Since (i,j) needs $B_{i+1\,j}$, $B_{i+2\,j}$, etc.

Looking back at Bcast of l_A, we notice
that Stage 0 Bcast of A_{ii} destroys the data in all
other l_A that are need in future $\Rightarrow$ So, we
need to save original l_A and recover it in each stage

```
l_Aoriginal = l_A
l_C = 0   (initialized)
for (s=0; s < q; s++) {
    MPI_Bcast(l_A, ll_u*ll_u, MPI_DOUBLE, (i+s)%q, row_comm)
    dgemm(..., 1.0, l_A, ..., l_B, ..., 1.0, l_C, ...) <=> l_C += l_A * l_B
    MPI_Sendrecv_replace(l_B, ..., (i-1+q)%q, (i+1)%q, col_comm)
    l_A = l_Aoriginal  /* recover original value */
}
```

Since we do q moves of l_B upwards in col_comm with a wraparound from top to bottom, B will be stored as before the algorithm.

Note: Pacheco on p. 127 does not have l_Aoriginal, but rather uses a temporary variable to program Bcast in an if statement:

```
if (j = (i+s)%q)   <=> if I am the sending process
                       (j = root of Bcast)
    MPI_Bcast(l_A, ....)
else  <=> if I am receiving data
    MPI_Bcast(l_Atmp, ....)
    dgemm(..., l_Atmp, ....)
```