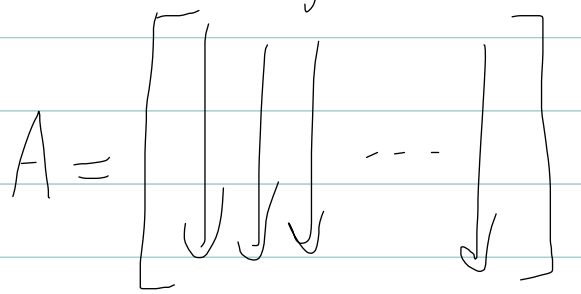
Monday, 01/30/12:

Today: The datastructure from homework, i.e., with matrix A stored as one-dimensional array in C, with storage organized in column-oriented storage $\Leftrightarrow$ first index is consecutive in memory

$A[i+n*j] = A_{ij}$ is next to $A[(i+1)+n*j] = A_{i+1\,j}$



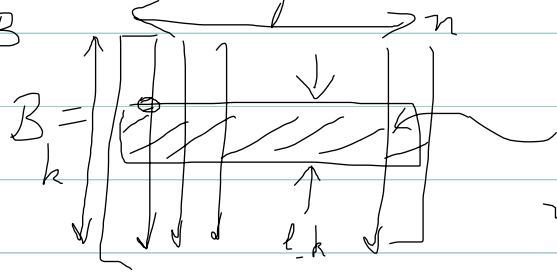
HW7 #2(6): $B \in \mathbb{R}^{k \times n}$

Question: What structure and size is $l\_B \in \mathbb{R}^{?\times ?}$?

$C = AB = \sum_{q=0}^{k-1} a_q b_q^T$, b_q^T is q-th row of $B \in \mathbb{R}^{k \times n}$

Idea of algorithm is to split this sum onto p parallel processes, so we need $l\_k = k/p$.

$\Rightarrow$ So, on each process, we need several, l_k many, rows of B



the rows b_q^T that we need on this process

Have B on Procs 0, so we want to program MPI_Send that sends these rows to another procs.

```
MPI_Type_vector(
        count
        blocklength
        stride
        old_type
        &new_mi_t )
MPI_Type_vector(n, l_k, k,
        MPI_DOUBLE, &block_row_t)
```

Solution 1: use MPI_Send / MPI_Recv like Ch. 3/4

Question: $l\_B \in \mathbb{R}^{l\_k \times n}$

```
if (id == 0) {
    for (j = 0; j < n; j++)
        for (l_q = 0; l_q < l_k; l_q++)
            l_B[l_q + l_k*j] = B[l_q + k*j];
    for (dest = 1; dest < npe; dest++)
        MPI_Send(&(B[ ]), 1, block_row_t,
                 dest, 0, MPI_COMM_WORLD)
                          dest * l_k
} else {
    MPI_Recv(l_B, l_k*n, MPI_DOUBLE,
             0, 0, MPI_COMM_WORLD)
}
```

For the case, recall that we need the type signature to match, namely l_k * n doubles. These are exactly the data for l_B on each process.

Solution 2: Make this simpler.

Really want to program Ch. 5 style collective call using MPI_Scatter

```
MPI_Scatter ( B, 1, block_row_t,
              l_B, (l_k*n), MPI_DOUBLE
              0, MPI_COMM_WORLD )
```

But this will not work!?

Reason: MPI_Scatter sends l_k*n doubles to each process here. The MPI_Scatter automatically uses the next-following element in B as the first element sent to the next process, instead of using &(B[dest*l_k]) which it should be.

In MPI terms, the sending to Process 1 uses as starting index the extent of block_row_t, which is l_k*n. But we need l_k as starting index for sending to Process 1. And need dest*l_k in general for all processes.

In other words, MPI_Scatter is designed for consecutive data, but block_row_t is not consecutive.

Solution in Pacheco Sec. 8.4.5

= there is a special datatype that overwrites the extent of a datatype manually, we will put l_k there, and then MPI_Scatter will work.

So, starting point is block_row_t as above.

Now define a doubly derived datatype based on that with a modified extent. Use MPI_Type_struct with first variable of type block_row_t and second one MPI_UB = upper bound = modifies the extent

```
int count = 2
int bl[2]
MPI_Aint dl[2]
MPI_Datatypes tl[2], ub_block_row_t
bl[0] = bl[1] = 1
dl[0] = 0
dl[1] = l_k * sizeof(double)
tl[0] = block_row_t
tl[1] = MPI_UB
MPI_Type_struct(count, bl, dl, tl, & ub_block_row_t)
MPI_Type_commit (& ub_block_row_t)
```

then can use

```
MPI_Scatter(B, 1, ub_block_row_t,
            l_B, (l_k*n), MPI_DOUBLE, 0, MPI_COMM_WORLD)
```

Load balancing on p parallel processes = HW2

trapezoidal rule, if p does not divide n

Idea: If p does not divide n, that is, n/p is not an integer, then there is a remainder of the integer division, namely rem = n % np.

So, add 1 trapezoid to rem many processes

```
rem = n % np
if (id < rem) {
    l_n = n/np + 1
} else {
    l_n = n/np        /* integer division! */
}
```

for trap. rule, also need l_a and l_b

⇒ Think and debug integers!

Idea: Think of indices l_ia and l_ib only

$l\_a = a + h * l\_ia$; $l\_b = a + h * l\_ib$ } same code on all processes!

$l\_ib = l\_ia + l\_n$

$$l\_ia = \begin{cases} l\_n * id = (n/np + 1) * id & \text{if } (id < rem) \\ (l\_n + 1) * rem + l\_n * (id - rem) & \text{if } (id >= rem) \\ = (n/np + 1)\, rem + (n/np) * (id - rem) & \end{cases}$$