Lecture 6: Parallel Matrix Algorithms (part 3)
A Simple Parallel Matrix-Matrix Multiplication

Let \( A = [a_{ij}]_{n \times n} \) and \( B = [b_{ij}]_{n \times n} \) be \( n \times n \) matrices. Compute \( C = AB \)

- Computational complexity of sequential algorithm: \( O(n^3) \)
- Partition \( A \) and \( B \) into \( p \) square blocks \( A_{i,j} \) and \( B_{i,j} \) \((0 \leq i, j < \sqrt{p})\) of size \((n/\sqrt{p}) \times (n/\sqrt{p})\) each.
- Use Cartesian topology to set up process grid. Process \( P_{i,j} \) initially stores \( A_{i,j} \) and \( B_{i,j} \) and computes block \( C_{i,j} \) of the result matrix.
- Remark: Computing submatrix \( C_{i,j} \) requires all submatrices \( A_{i,k} \) and \( B_{k,j} \) for \( 0 \leq k < \sqrt{p} \).
• Algorithm:
  – Perform all-to-all broadcast of blocks of A in each row of processes
  – Perform all-to-all broadcast of blocks of B in each column of processes
  – Each process $P_{i,j}$ perform $C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} B_{k,j}$
Performance Analysis

• $\sqrt{p}$ rows of all-to-all broadcasts, each is among a group of $\sqrt{p}$ processes. A message size is $\frac{n^2}{p}$, communication time: $t_s log \sqrt{p} + t_w \frac{n^2}{p} (\sqrt{p} - 1)$

• $\sqrt{p}$ columns of all-to-all broadcasts, communication time: $t_s log \sqrt{p} + t_w \frac{n^2}{p} (\sqrt{p} - 1)$

• Computation time: $\sqrt{p} \times (n/\sqrt{p})^3 = n^3 / p$

• Parallel time: $T_p = \frac{n^3}{p} + 2 \left( t_s log \sqrt{p} + t_w \frac{n^2}{p} (\sqrt{p} - 1) \right)$
Memory Efficiency of the Simple Parallel Algorithm

• Not memory efficient
  – Each process $P_{i,j}$ has $2\sqrt{p}$ blocks of $A_{i,k}$ and $B_{k,j}$
  – Each process needs $\Theta(n^2 / \sqrt{p})$ memory
  – Total memory over all the processes is $\Theta(n^2 \times \sqrt{p})$, i.e., $\sqrt{p}$ times the memory of the sequential algorithm.
Cannon’s Algorithm of Matrix-Matrix Multiplication

**Goal:** to improve the memory efficiency.

Let $A = [a_{ij}]_{n \times n}$ and $B = [b_{ij}]_{n \times n}$ be $n \times n$ matrices. Compute $C = AB$

- Partition $A$ and $B$ into $p$ square blocks $A_{i,j}$ and $B_{i,j}$ ($0 \leq i, j < \sqrt{p}$) of size $(n/\sqrt{p}) \times (n/\sqrt{p})$ each.
- Use Cartesian topology to set up process grid. Process $P_{i,j}$ initially stores $A_{i,j}$ and $B_{i,j}$ and computes block $C_{i,j}$ of the result matrix.
- Remark: Computing submatrix $C_{i,j}$ requires all submatrices $A_{i,k}$ and $B_{k,j}$ for $0 \leq k < \sqrt{p}$.
- **The contention-free formula:**

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\%\sqrt{p}}B_{(i+j+k)\%\sqrt{p},j}$$
Cannon's Algorithm

// make initial alignment
for i, j := 0 to \(\sqrt{p} - 1\) do
    Send block \(A_{i,j}\) to process \((i, (j - i + \sqrt{p}) \text{mod} \sqrt{p})\) and block \(B_{i,j}\) to process \(((i - j + \sqrt{p}) \text{mod} \sqrt{p}, j)\);
endfor;
Process \(P_{i,j}\) multiply received submatrices together and add the result to \(C_{i,j}\);

// compute-and-shift. A sequence of one-step shifts pairs up \(A_{i,k}\) and \(B_{k,j}\)
// on process \(P_{i,j}\). \(C_{i,j} = C_{i,j} + A_{i,k}B_{k,j}\)
for step := 1 to \(\sqrt{p} - 1\) do
    Shift \(A_{i,j}\) one step left (with wraparound) and \(B_{i,j}\) one step up (with wraparound);
    Process \(P_{i,j}\) multiply received submatrices together and add the result to \(C_{i,j}\);
Endfor;

Remark: In the initial alignment, the send operation is to: shift \(A_{i,j}\) to the left (with wraparound) by \(i\) steps, and shift \(B_{i,j}\) to the up (with wraparound) by \(j\) steps.
Cannon’s Algorithm for $3 \times 3$ Matrices

Initial A, B

A, B initial alignment

A, B after shift step 1

A, B after shift step 2
Performance Analysis

- In the initial alignment step, the maximum distance over which block shifts is $\sqrt{p} - 1$
  - The circular shift operations in row and column directions take time: $t_{comm} = 2(t_s + \frac{t_w n^2}{p})$

- Each of the $\sqrt{p}$ single-step shifts in the compute-and-shift phase takes time: $t_s + \frac{t_w n^2}{p}$.

- Multiplying $\sqrt{p}$ submatrices of size $\left(\frac{n}{\sqrt{p}}\right) \times \left(\frac{n}{\sqrt{p}}\right)$ takes time: $n^3/p$.

- Parallel time: $T_p = \frac{n^3}{p} + 2\sqrt{p} \left(t_s + \frac{t_w n^2}{p}\right) + 2\left(t_s + \frac{t_w n^2}{p}\right)$
int MPI_Sendrecv_replace( void *buf, int count,
MPI_Datatype datatype, int dest, int sendtag, int source,
int recvtag, MPI_Comm comm, MPI_Status *status );

• Execute a blocking send and receive. The same buffer is
used both for the send and for the receive, so that the
message sent is replaced by the message received.

• buf[in/out]: initial address of send and receive buffer
#include "mpi.h"
#include <stdio.h>

int main(int argc, char *argv[]) 
{
    int myid, numprocs, left, right;
    int buffer[10];
    MPI_Request request;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    right = (myid + 1) % numprocs;
    left = myid - 1;
    if (left < 0)
        left = numprocs - 1;

    MPI_Sendrecv_replace(buffer, 10, MPI_INT, left, 123, right, 123, MPI_COMM_WORLD, &status);

    MPI_Finalize();
    return 0;
}