CSE 160
Lecture 18

Parallel Matrix Multiplication
Working with communicators
Announcements

• Assign 4 deadline is extended by 6 hours
  Weds, Dec 3 @ 11.59PM
Today’s lecture

• Quiz #4 return
• Cannon’s Parallel Matrix Multiplication Algorithm
• Working with communicators
Quiz #4

• What is the significance of the values $\alpha$ and $\beta_\infty$ in the formula $T(n) = \alpha + \beta_\infty \times N$?

• What are the values of $s$ & $t$ at the end?

<table>
<thead>
<tr>
<th>Process P0</th>
<th>Process P1</th>
<th>Process P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Send}(x,P2)$</td>
<td>$\text{Send}(z,P2)$</td>
<td>$\text{Recv}(s,*)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{Recv}(t,*)$</td>
</tr>
</tbody>
</table>
Matrix Multiplication

• An important core operation in many numerical algorithms

• Given two *conforming* matrices $A$ and $B$, form the matrix product $A \times B$
  
  $A$ is $m \times n$
  
  $B$ is $n \times p$

• Operation count: $O(n^3)$ multiply-adds for an $n \times n$ square matrix
Simplest Serial Algorithm

“ijk”

for i := 0 to n-1
  for j := 0 to n-1
    for k := 0 to n-1
      \[ C[i,j] += A[i,k] \times B[k,j] \]
Parallel matrix multiplication

• Assume \( p \) is a perfect square
• Each processor gets an \( \frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}} \) chunk of data
• Organize processors into rows and columns
• Process rank is an ordered pair of integers
• Assume that we have an efficient serial matrix multiply (\text{dgemm}, \text{sgemm})
Canon’s algorithm

- Move data incrementally in $\sqrt{p}$ phases
- Circulate each chunk of data among processors within a row or column
- In effect we are using a ring broadcast algorithm
- Consider iteration $i=1, j=2$:


<table>
<thead>
<tr>
<th>$A(0,0)$</th>
<th>$A(0,1)$</th>
<th>$A(0,2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(1,0)$</td>
<td>$A(1,1)$</td>
<td>$A(1,2)$</td>
</tr>
<tr>
<td>$A(2,0)$</td>
<td>$A(2,1)$</td>
<td>$A(2,2)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B(0,0)$</th>
<th>$B(0,1)$</th>
<th>$B(0,2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B(1,0)$</td>
<td>$B(1,1)$</td>
<td>$B(1,2)$</td>
</tr>
<tr>
<td>$B(2,0)$</td>
<td>$B(2,1)$</td>
<td>$B(2,2)$</td>
</tr>
</tbody>
</table>
Canon’s algorithm


- We want \( A[1,0] \) and \( B[0,2] \) to reside on the same processor initially.

- Shift rows and columns so the next pair of values \( A[1,1] \) and \( B[1,2] \) line up.

- And so on with \( A[1,2] \) and \( B[2,2] \).
Canon’s algorithm – 1 element of C


- We want \( A[1,0] \) and \( B[0,2] \) to reside on the same processor initially.
- Shift rows and columns so the next pair of values \( A[1,1] \) and \( B[1,2] \) line up.
- And so on with \( A[1,2] \) and \( B[2,2] \).
Skewing the matrices


- We want \(A[1,0]\) and \(B[0,2]\) to reside on the same processor initially

- Shift rows and columns so the next pair of values \(A[1,1]\) and \(B[1,2]\) line up

- And so on with \(A[1,2]\) and \(B[2,2]\)
Shift and multiply

\[
\]

- Takes $\sqrt{p}$ steps
- Circularly shift
  - each row by 1 column to the left
  - each column by 1 row to the left
- Each processor forms the product of the two local matrices adding into the accumulated sum
Cost of Cannon’s Algorithm

forall  i=0 to √p -1
    CShift-left A[i; :) by i  // T= α+βn²/p
forall  j=0 to √p -1
    Cshift-up B[; : , j] by j  // T= α+βn²/p
for k=0 to √p -1
   forall  i=0 to √p -1 and j=0 to √p -1
        C[i,j] += A[i,j]*B[i,j]  // T = 2*n³/p³/2
        CShift-leftA[i; :) by 1  // T= α+βn²/p
        Cshift-up B[; : , j] by 1  // T= α+βn²/p
end forall
end for

\[ T_P = \frac{2n^3}{p} + 2(\alpha(1+\sqrt{p}) + \beta n^2/(1+\sqrt{p})/p) \]
\[ E_P = \frac{T_1}{(pT_P)} = (1 + \alpha p^{3/2}/n^3 + \beta \sqrt{p}/n)^{-1} \]
\[ \approx (1 + O(\sqrt{p}/n))^{-1} \]
\[ E_P \to 1 \text{ as (n/√p) grows [sqrt of data / processor]} \]
Implementation
Communication domains

- Cannon’s algorithm shifts data along rows and columns of processors
- MPI provides communicators for grouping processors, reflecting the communication structure of the algorithm
- An MPI communicator is a name space, a subset of processes that communicate
- Messages remain within their communicator
- A process may be a member of more than one communicator
Establishing row communicators

- Create a communicator for each row and column
- By Row

\[
\text{key} = \text{myRank div } \sqrt{P}
\]
Creating the communicators

• Create a row communicator  \[ \text{key} = \text{myRank} \div \sqrt{P} \]

```c
MPI_Comm rowComm;
MPI_Comm_split(MPI_COMM_WORLD, "myRank / \sqrt{P}\), myRank, &rowComm);
MPI_Comm_rank(rowComm, &myRow);
```

• Each process obtains a new communicator
• Each process’ rank relative to the new communicator
• Rank applies to the respective communicator only
• Ordered according to myRank
More on Comm_split

MPI_Comm_split(MPI_Comm comm,  int splitKey,
                int rankKey,  MPI_Comm* newComm)

• Ranks assigned arbitrarily among processes sharing the same rankKey value
• May exclude a process by passing the constant MPI_UNDEFINED as the splitKey
• Return a special MPI_COMM_NULL communicator
• If a process is a member of several communicators, it will have a rank within each one
Circular shift

• Communication with columns (and rows
More on Comm_split

MPI_Comm_split(MPI_Comm comm, int splitKey, int rankKey, MPI_Comm* newComm)

• Ranks assigned arbitrarily among processes sharing the same **rankKey** value
• May exclude a process by passing the constant **MPI_UNDEFINED** as the **splitKey**
• Return a special **MPI_COMM_NULL** communicator
• If a process is a member of several communicators, it will have a rank within each one
Circular shift

• Communication with columns (and rows

\[
\begin{array}{|c|c|c|}
\hline
p(0,0) & p(0,1) & p(0,2) \\
\hline
p(1,0) & p(1,1) & p(1,2) \\
\hline
p(2,0) & p(2,1) & p(2,2) \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
B(0,0) & B(1,1) & B(2,2) \\
\hline
B(1,0) & B(2,1) & B(0,2) \\
\hline
B(2,0) & B(0,1) & B(1,2) \\
\hline
\end{array}
\]
Circular shift

- Communication with columns (and rows)

\[
\text{MPI\_Comm\_rank}(\text{rowComm}, \&\text{myidRing});
\]
\[
\text{MPI\_Comm\_size}(\text{rowComm}, \&\text{nodesRing});
\]
\[
\text{int next} = (\text{myidRng} + 1) \% \text{nodesRing};
\]
\[
\text{MPI\_Send}(\&X, 1, \text{MPI\_INT}, \text{next}, 0, \text{rowComm});
\]
\[
\text{MPI\_Recv}(\&XR, 1, \text{MPI\_INT},
              \text{MPI\_ANY\_SOURCE},
              0, \text{rowComm}, \&\text{status});
\]

- Processes 0, 1, 2 in one communicator because they share the same key value (0)

- Processes 3, 4, 5 are in another (key=1), and so on