Lecture 14

SUMMA Matrix Multiplication Algorithm
Advanced Collectives
Today’s lecture

- SUMMA Matrix Multiplication Algorithm
- Advanced Communication
An improved matrix multiply

• Difficulties with Cannon’s Algorithm
  ✷ P is not a perfect square
  ✷ A and B are not square, and not evenly divisible by $\sqrt{p}$

• Interoperation with applications and other libraries difficult or expensive

• The SUMMA algorithm offers a practical alternative
  ✷ Uses a shift algorithm to broadcast
  ✷ A variant used in SCALAPACK by Van de Geign and Watts [1997]
Formulation

- The matrices may be non-square (kij formulation)

\[
\text{for } k := 0 \text{ to } n_3 - 1 \\
\quad \text{for } i := 0 \text{ to } n_1 - 1 \\
\quad \quad \text{for } j := 0 \text{ to } n_2 - 1 \\
\quad \quad \quad C[i,j] += A[i,k] \times B[k,j] \\
C[i,:] += A[i,k] \times B[k,:] 
\]

- The two innermost loop nests compute \( n_3 \) outer products

\[
\text{for } k := 0 \text{ to } n_3 - 1 \\
\quad C[ :, :] += A[ :, k] \times B[k, :] 
\]

where \( \times \) is outer product
Outer product

• Recall that when we multiply an \( m \times n \) matrix by an \( n \times p \) matrix… we get an \( m \times p \) matrix

• Outer product of *column vector* \( \mathbf{a}^T \) and *vector* \( \mathbf{b} = \text{matrix } \mathbf{C} \)
  
an \( m \times 1 \) times a \( 1 \times n \)

\[
\mathbf{a}[1,3] \cdot \mathbf{x}[3,1]
\]

\[
(a,b,c) \times (x,y,z)^T = \begin{pmatrix}
ax & ay & az \\
bx & by & bz \\
cx & cy & cz
\end{pmatrix}
\]

Multiplication table with rows formed by \( \mathbf{a}[:] \) and the columns by \( \mathbf{b}[:] \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>10</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>60</td>
<td>90</td>
</tr>
</tbody>
</table>

• The SUMMA algorithm computes \( n \) partial outer products:

\[
\text{for } k := 0 \text{ to } n-1 \\
\mathbf{C}[:,:] += \mathbf{A}[:,k] \cdot \mathbf{B}[k,:]
\]
Outer Product Formulation

• The new algorithm computes $n$ partial outer products:

\[
\text{for } k := 0 \text{ to } n-1 \\
C[:,:] += A[:,k] \cdot B[k,:] 
\]

"Inner product" formulation:
for $i := 0$ to $n-1$, $j := 0$ to $n-1$
\[
C[i,j] += A[i,:] \cdot B[:,j] 
\]
Serial algorithm

- Each row $k$ of B contributes to the $n$ partial outer products

$$\text{for } k := 0 \text{ to } n-1$$
$$\text{C}[:,:] += \text{A}[:,k] \cdot \text{B}[k,:]
$$
Animation of SUMMA

- Compute the sum of $n$ outer products
- Each row & column $(k)$ of $A$ & $B$ generates a single outer product
  - Column vector $A[:,k]$ $(n \times 1)$ & a vector $B[k,:]$ $(1 \times n)$

for $k := 0$ to $n-1$

$$C[:,:] += A[:,k] \cdot B[k,:]$$
Animation of SUMMA

• Compute the sum of $n$ outer products
• Each row & column $(k)$ of $A$ & $B$ generates a single outer product
  - $A[:,k+1] \cdot B[k+1,:]$ 

for $k := 0$ to $n-1$

$C[:,:] += A[:,k] \cdot B[k,:]$
Animation of SUMMA

- Compute the sum of $n$ outer products
- Each row & column (k) of A & B generates a single outer product
  - $A[:,n-1] \cdot B[n-1,:]$

\[
\text{for } k := 0 \text{ to } n-1 \\
C[:,,:] += A[:,k] \cdot B[k,:] 
\]
Parallel algorithm

- Processors organized into rows and columns, process rank an ordered pair
- Processor geometry $P = px \times py$
- Blocked (serial) matrix multiply, panel size $= b << N/\max(px,py)$

\[
\begin{align*}
\text{for } k := 0 \text{ to } n-1 \text{ by } b \\
\text{Owner of } A[:,k:k+b-1] \text{ Bcasts to } ACol & \quad \text{// Along processor rows} \\
\text{Owner of } B[k:k+b-1,:] \text{ Bcasts } BRow & \quad \text{// Along processor columns} \\
C += \text{Serial Matrix Multiply}(ACol,BRow ) \\
\end{align*}
\]

- Each row and column of processors independently participate in a panel broadcast
- Owner of the panel (Broadcast root) changes with $k$, shifts across matrix

What is the performance?

for k := 0 to n−1 by b

// Tree broadcast:  \( \lg \sqrt{p} (\alpha + b\beta n/\sqrt{p}) \)
// For long messages: \( 2((\sqrt{p} -1)/ p)b\beta n \)

\[
\text{multicast } A[:, k:k+b-1] \text{ along rows}
\]
\[
\text{multicast } B[k:k+b-1, :] \text{ along columns}
\]
// Built in matrix multiply: \( 2(n/\sqrt{p})^2 b \)

\[
C += A[:,k:k+b-1] * B[k:k+b-1,:,] 
\]

- Total running time: \( \sim 2n^3/p + 4\beta bn/ \sqrt{p} \)
- On triton.sdsc.edu Intel E5530, Gainstown 2.4 GHz
  - Panel size = 64
  - 16 cores: \((1792^2) \ 0.117 \text{ sec} [0.1163]; \ 98.5 \text{ GF} (6.15/core)\)
  - 64 cores \((3854^2) \ 0.259 \text{ sec} [0.258]; \ 354 \text{ GF} (5.54/core)\)

Today’s lecture

• SUMMA Matrix Multiplication Algorithm
• Advanced Collective Communication
Collective communication

• Diverse applications
  ✷ Fast Fourier Transform
  ✷ Sorting
• Collective operations are called by all processes within a communicator
• Basic collectives seen so far
  ✷ Broadcast: distribute data from a designated root process to all the others
  ✷ Reduce: combine data from all processes returning the result to the root process
• Other Useful collectives
  ✷ Scatter/gather
  ✷ All to all
  ✷ Allgather
Underlying assumptions

• Fast interconnect structure
  - All nodes are equidistant
  - Single-ported, bidirectional links

• Communication time is $\alpha + \beta n$ in the absence of contention
  - Determined by bandwidth $\beta^{-1}$ for long messages
  - Dominated by latency $\alpha$ for short messages
Inside MPI-CH

- Tree like algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
- Block size may be configured at installation time
- If there is hardware support, then it is given responsibility to carry out the broadcast
- Polyalgorithms apply different algorithms to different cases, i.e. long vs. short messages, different machine configurations
- We’ll use hypercube algorithms to simplify the special cases when $P=2^k$, $k$ an integer
Details of the algorithms

- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast
Scatter/Gather family

\( P_0 \quad P_1 \quad P_{p-1} \)

Gather

Scatter

Root
Scatter

- Simple linear algorithm
  - Root processor sends a chunk of data to all others
  - Reasonable for long messages

\[(p - 1)\alpha + \frac{p - 1}{p} n\beta\]

- Similar approach taken for Reduce and Gather
- For short messages, we need to reduce the complexity of the latency (\(\alpha\)) term
Minimum spanning tree algorithm

- Recursive hypercube-like algorithm with ⌈log P⌉ steps
  - Root sends half its data to process (root + p/2) mod p
  - Each receiver acts as a root for corresponding half of the processes
  - MST: organize communication along edges of a minimum-spanning tree covering the nodes
- Requires O(n/2) temp buffer space on intermediate nodes
- Running time:
  \[\lceil \lg P \rceil \alpha + \frac{p-1}{p} n \beta\]
Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
All to all

- Also called *total exchange or personalized communication*: a transpose
- Each process sends a different chunk of data to each of the other processes
- Used in sorting and the Fast Fourier Transform
Exchange algorithm

- \( n \) elements / processor \( (n \) total elements) \\
- \( p - 1 \) step algorithm
  - Each processor exchanges \( n/p \) elements with each of the others
  - In step \( i \), process \( k \) exchanges with processes \( k \pm i \)

\[
\text{for } i = 1 \text{ to } p-1 \\
\text{ src } = (\text{rank } - i + p) \mod p \\
\text{ dest } = (\text{rank } + i ) \mod p \\
\text{ sendrecv( from src to dest ) }
\]

- Good algorithm for long messages
- Running time:

\[
(p - 1)\alpha + (p - 1) \frac{n}{p} \beta \approx n\beta
\]
Recursive doubling for short messages

- In each of $\lceil \log p \rceil$ phases all nodes exchange $\frac{1}{2}$ their accumulated data with the others
- Only $P/2$ messages are sent at any one time

\[
D = 1
\]

\[
\text{while (D < p)} \\
\quad \text{Exchange & accumulate data with rank } \otimes D \\
\quad \text{Left shift D by 1}
\]

end while

- Optimal running time for short messages

\[
[\lg P] \alpha + nP \beta \approx [\lg P] \alpha
\]
Flow of information

A B C D

A B C D

10 11

00 01

Flow of information
Flow of information
Summarizing all to all

- Short messages \[ [\lg P] \alpha \]

- Long messages \[ \frac{P-1}{P} n \beta \]
“Vector” variants

- Generalize all-to-all, gather, etc.
- Processes supply varying length datum
- Vector all-to-all

```c
MPI_Alltoallv (  
    void *sendbuf, int sendcounts[], int sDispl [],  
    MPI_Datatype sendtype,  
    void* recvbuf, int recvcounts[], int rDispl[],  
    MPI_Datatype recvtype, MPI_Comm comm )
```
Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
Revisiting Broadcast

• P may not be a power of 2
• We use a binomial tree algorithm
• We’ll use the hypercube algorithm to illustrate the special case of $P=2^k$
• Hypercube algorithm is efficient for short messages
• We use a different algorithm for long messages
Strategy for long messages

• Based van de Geijn’s strategy
• Scatter the data
  ♦ Divide the data to be broadcast into pieces, and fill the machine with the pieces
• Do an Allgather
  ♦ Now that everyone has a part of the entire result, collect on all processors
• Faster than MST algorithm for long messages

\[ 2 \frac{p-1}{p} n\beta \ll \lceil \log p \rceil n\beta \]
Algorithm for long messages

The scatter step

$P_0$  $P_1$  $P_{p-1}$  Root

Scatter
Algorithm for long messages

AllGather step