Lecture 13
Matrix Multiplication
Announcements

• Project Progress report, due next Weds 11/28
Today’s lecture

- Cannon’s Matrix Multiplication Algorithm
- 2.5D “Communication avoiding”
- SUMMA
Parallel matrix multiplication

• Assume $p$ is a perfect square
• Each processor gets an $n/\sqrt{p} \times n/\sqrt{p}$ chunk of data
• Organize processors into rows and columns
• Assume that we have an efficient serial matrix multiply (dgemm, 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 $i=1$, $j=2$:

\[
\]
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 \( B[2,2] \)
Skewing the matrices


- We first skew the matrices so that everything lines up
- Shift each row $i$ by $i$ columns to the left using sends and receives
- Communication wraps around
- Do the same for each column
### 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 \( \sqrt{p} - 1 \)

\( \text{CShift-left } A[i; :] \) by \( i \) // \( T = \alpha + \beta n^2 / p \)

forall \( j = 0 \) to \( \sqrt{p} - 1 \)

\( \text{Cshift-up } B[; , j] \) by \( j \) // \( T = \alpha + \beta n^2 / p \)

for \( k = 0 \) to \( \sqrt{p} - 1 \)

forall \( i = 0 \) to \( \sqrt{p} - 1 \) and \( j = 0 \) to \( \sqrt{p} - 1 \)

\( C[i,j] += A[i,j]*B[i,j] \) // \( T = 2*n^3/p^{3/2} \)

\( \text{CShift-left } A[i; :] \) by 1 // \( T = \alpha + \beta n^2 / p \)

\( \text{Cshift-up } B[; , j] \) by 1 // \( T = \alpha + \beta n^2 / p \)

end for all

end for

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

```c
MPI_Comm rowComm;
MPI_Comm_split(MPI_COMM_WORLD, myRank / √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

©2012 Scott B. Baden /CSE 260/ Fall 2012
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

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

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

MPI_Comm_rank(rowComm,&myidRing);
MPI_Comm_size(rowComm,&nodesRing);
int next = (myidRng + 1) % nodesRing;
MPI_Send(&X,1,MPI_INT,next,0,rowComm);
MPI_Recv(&XR,1,MPI_INT,
          MPI_ANY_SOURCE,
          MPI_ANY_SOURCE,0,
          rowComm,&status);

Processes 0, 1, 2 are in one communicator because they share the same value of key (0)
Processes 3, 4, 5 are in another (1), and so on
Today’s lecture

• Cannon’s Matrix Multiplication Algorithm
• 2.5D “Communication avoiding”
• SUMMA
Motivation

• Relative to arithmetic speeds, communication is becoming more costly with time
• Communication can be data motion on or off-chip, across address spaces
• We seek algorithms that increase the amount of work (flops) relative to the amount of data they move
Communication lower bounds on Matrix Multiplication

• Assume we are using an $O(n^3)$ algorithm
• Let $M =$ Size fast memory (cache/local memory)
• Sequential case: # slow memory references
  $\Omega \left(\frac{n^3}{\sqrt{M}}\right)$ [Hong and Kung ’81]
• Parallel, $p =$ # processors,
  $\mu =$ Amount of memory needed to store matrices
  - Refs to remote memory
    $\Omega \left(\frac{n^3}{(p\sqrt{\mu})}\right)$ [Irony, Tiskin, Toledo, ’04]
  - If $\mu = 3n^2/p$ (one copy of A, B, C) $\Rightarrow$
    lower bound = $\Omega \left(\frac{n^2}{\sqrt{p}}\right)$ words
  - Achieved by Cannon’s algorithm (“2D algorithm”)
  - $T_p = 2n^3/p + 4\sqrt{p} (\alpha + \beta n^2/p)$
Canon’s Algorithm - optimality

- General result
  - If each processor has $M$ words of local memory …
  - … at least 1 processor must transmit $\Omega \left( \frac{\# \text{ flops}}{M^{1/2}} \right)$ words of data

- If local memory $M = O(n^2/p)$ …
  - at least 1 processor performs $f \geq n^3/p$ flops
  - … lower bound on number of words transmitted by at least 1 processor

\[ \Omega \left( \frac{(n^3/p)}{\sqrt{(n^2/p)}} \right) = \Omega \left( \frac{(n^3/p)}{\sqrt{M}} \right) = \Omega \left( \frac{n^2}{\sqrt{p}} \right) \]
New communication lower bounds – direct linear algebra [Ballard & Demmel ’11]

• Let $M =$ amount of fast memory per processor
• Lower bounds
  ◆ # words moved by at least 1 processor
    $\Omega \left( \frac{\text{# flops}}{M^{1/2}} \right)$
  ◆ # messages sent by at least 1 processor
    $\Omega \left( \frac{\text{# flops}}{M^{3/2}} \right)$
• Holds not only for Matrix Multiply but many other “direct” algorithms in linear algebra, sparse matrices, some graph theoretic algorithms
• Identify 3 values of $M$
  ◆ 2D (Cannon’s algorithm)
  ◆ 3D (Johnson’s algorithm)
  ◆ 2.5D (Ballard and Demmel)
Johnson’s 3D Algorithm

- 3D processor grid: $p^{1/3} \times p^{1/3} \times p^{1/3}$
  - Bcast A (B) in $j$ ($i$) direction ($p^{1/3}$ redundant copies)
  - Local multiplications
  - Accumulate (Reduce) in $k$ direction
- Communication costs (optimal)
  - Volume = $O(\frac{n^2}{p^{2/3}})$
  - Messages = $O(\log(p))$
- Assumes space for $p^{1/3}$ redundant copies
- Trade memory for communication

Source: Edgar Solomonik
2.5D Algorithm

• What if we have space for only \(1 \leq c \leq p^{1/3}\) copies?
• \(M = \Omega(c \cdot n^2/p)\)
• Communication costs: lower bounds
  - Volume = \(\Omega(n^2 / (cp)^{1/2})\); Set \(M = c \cdot n^2/p\) in \(\Omega\) (# flops / \(M^{1/2}\))
  - Messages = \(\Omega(p^{1/2} / c^{3/2})\); Set \(M = c \cdot n^2/p\) in \(\Omega\) (# flops / \(M^{3/2}\))
• 2.5D algorithm “interpolates” between 2D & 3D algorithms

Source: Edgar Solomonik
2.5D Algorithm

- Assume can fit $cn^2/P$ data per processor, $c>1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid

Example: $P = 32$, $c = 2$

Source Jim Demmel
2.5D Algorithm

- Assume can fit $cn^2/P$ data per processor, $c>1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid

Initially $P(i,j,0)$ owns $A(i,j)$ & $B(i,j)$ each of size $n(c/P)^{1/2} \times n(c/P)^{1/2}$

1. $P(i,j,0)$ broadcasts $A(i,j)$ and $B(i,j)$ to $P(i,j,k)$
2. Processors at level $k$ perform $1/c$-th of SUMMA, i.e. $1/c$-th of $\sum_m A(i,m) B(m,j)$
3. Sum-reduce partial sums $\sum_m A(i,m) B(m,j)$ along $k$-axis so that $P(i,j,0)$ owns $C(i,j)$

Source Jim Demmel
Performance on Blue Gene P

Matrix multiplication on 16,384 nodes of BG/P

95% reduction in comm

Execution time normalized by 2D

C=16

Source Jim Demmel
2.5D Algorithm

• Interpolate between 2D (Cannon) and 3D
  ✓ $c$ copies of A & B
  ✓ Perform $p^{1/2}/c^{3/2}$ Cannon steps on each copy of A&B
  ✓ Sum contributions to C over all $c$ layers

• Communication costs (not quite optimal, but not far off)
  ✓ Volume:
    $$O(n^2/(cp)^{1/2})$$
    $$\Omega(n^2/(cp)^{1/2})$$
  ✓ Messages:
    $$O(p^{1/2}/c^{3/2} + \log(c))$$
    $$\Omega(p^{1/2}/c^{3/2})$$
Today’s lecture

• Cannon’s Matrix Multiplication Algorithm
• 2.5D “Communication avoiding”
• SUMMA
Outer product formulation of matrix multiply

• Limitations of Cannon’s Algorithm
  ♦ P is must be a perfect square
  ♦ A and B must be square, and 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)

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

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

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

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

• Recall that when we multiply
  an m×n matrix by an n × p matrix…
  we get an m × p matrix

• Outer product of column vector $a^T$ and vector $b = \text{matrix } C$
  an m × 1 times a 1 × n

\[
a[1,3] \cdot x[3,1]
\]

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

Multiplication table with rows formed by $a[\cdot]$ and the columns by $b[\cdot]$

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

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

\[
\text{for } k := 0 \text{ to } n-1
\]

\[
C[:,:] += A[:,k] \cdot B[k,:]
\]
The new algorithm computes \( n \) partial outer products:

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

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

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

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

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

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

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

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

for $k := 0$ 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 $= b << N/\max(px,py)$

$$\text{for } k := 0 \text{ to } n-1 \text{ by } b$$

- Owner of $A[:,k:k+b-1]$ Bcasts to ACol  // Along processor rows
- Owner of $B[k:k+b-1,:]$ Bcasts BRow  // Along processor columns
- $C += \text{Serial Matrix Multiply}(ACol,BRow)$

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