Parallelizing The Matrix Multiplication
Serial version
\[
A_{md} \times X = B_{dn} = C_{mn}
\]

\[
c_{i,j} = \sum_{k=1}^{d} a_{i,k} \cdot b_{k,j}
\]
//Read and validate command line arguments
Read M,N,D from the command line

//Initialize the arrays
For all elements of A and B
initial value = function ( i , j )

// Matrix multiplication
For each C (i,j)
// Take the inner product of row i of A and column j of B
For each A (i,k) and B(k,j)
C (i,j) = C (i,j) + A (i,k) * B(k,j)

\[
a_{i,j} = i + j \\
b_{i,j} = i \times j
\]
// Read and validate command line arguments
Read M,N,D from the command line

// Initialize the arrays
For all elements of A and B
  initial value = function ( i , j )

// Matrix multiplication
For each C (i,j)
  // Take the inner product of row i of A and column j of B
  For each A (i,k) and B(k,j)
    C (i,j) = C (i,j) + A (i,k) * B(k,j)

\[
a_{i,j} = i + j \\
b_{i,j} = i \times j
\]
//Read and validate command line arguments
Read M,N,D from the command line

//Initialize the arrays
For all elements of A and B
  initial value = function ( i , j )

// Matrix multiplication
For each C (i,j)
  // Take the inner product of row i of A and column j of B
  For each A (i,k) and B(k,j)
    C (i,j) = C (i,j) + A (i,k) * B(k,j)

How do we know the result is correct?
Result validation is very important!

\[
a_{i,j} = i + j \\
b_{i,j} = i \times j
\]
\[ c_{i,j} = \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j} \]

\[ = \sum_{k=1}^{N} (i + k) \cdot j \cdot k \]

Assuming \( A, B \) and \( C \) are all \( N \times N \) square matrices

\[ = \sum_{k=1}^{N} (ijk + jk^2) \]

\[ = ij \sum_{k=1}^{N} k + j \sum_{k=1}^{N} k^2 \]

\[ = ij \cdot \frac{N(N + 1)}{2} + j \cdot \frac{N(N + 1)(2N + 1)}{6} \]

We can use this formula to validate the result from the program.
//Read and validate command line arguments
Read M,N,D from the command line

//Initialize the arrays
For all elements of A and B
  initial value = function ( i , j )

// Matrix multiplication
For each C (i,j)
  // Take the inner product of row i of A and column j of B
    For each A (i,k) and B(k,j)
      C (i,j) = C (i,j) + A (i,k) * B(k,j)

//Validate the result
Print out an element of C and compare it to the value
  given by the formula

\[ a_{i,j} = i + j \]
\[ b_{i,j} = i \times j \]
Serial Program Details

• A, B and C are all N x N square matrices
• Take input arguments
  – N: dimension of the matrices
  – \( i_{\text{peek}}, j_{\text{peek}} \): indices of the element used for result validation
• Values of A and B:

\[
a_{i,j} = i + j \\
b_{i,j} = i \times j
\]

• Output
  – How much time it takes (performance measurement)
  – The value of \( C(i_{\text{peek}}, j_{\text{peek}}) \) and the value given by the formula
Parallel version
Writing a parallel program step by step

• Step 1. Start from serial programs as a baseline
  – Something to check correctness and efficiency against
• Step 2. Analyze and profile the serial program
  – Identify the “hotspot”
  – Identify the parts that can be parallelized
• Step 3. Parallelize code incrementally
• Step 4. Check correctness of the parallel code
• Step 5. Iterate step 3 and 4
/\Read and validate command line arguments
Read M,N,D from the command line

//Initialize the arrays
For all elements of A and B
  initial value = function (i, j)

// Matrix multiplication
For each C(i,j)
  // Take the inner product of row i of A and column j of B
  For each A(i,k) and B(k,j)
    C(i,j) = C(i,j) + A(i,k) * B(k,j)

//Validate the result
Print out an element of C and compare it to the value given by the formula

Which parts can and should be parallelized?
//Read and validate command line arguments
Read M,N,D from the command line

//Initialize the arrays
For all elements of A and B
  initial value = function (i, j)

// Matrix multiplication
For each C(i,j)
  // Take the inner product of row i of A and column j of B
  For each A(i,k) and B(k,j)
    C(i,j) = C(i,j) + A(i,k) * B(k,j)

//Validate the result
Print out an element of C and compare it to the value
  given by the formula

Which parts can and should be parallelized?
\[ \mathbf{A}_{nn} \times \mathbf{X} = \mathbf{B}_{nn} = \mathbf{C}_{nn} \]

\[ c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j} \]
Some Considerations

• Decomposition
  – 1-D or 2-D

• Data distribution
  – Each process owns the entire matrices, but only performs calculation on a part of them, or
  – Each process only owns the sub-matrices that it is going to process
\[
A_{nn} \times X = B_{nn} = C_{nn}
\]

\[
c_{i,j} = \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j}
\]
\[
\begin{align*}
A_{nn} \times X &= B_{nn} = C_{nn} \\

c_{i,j} &= \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j}
\end{align*}
\]
Optimization Considerations

• Avoid data movement as much as possible (i.e. increase the amount of computation done relative to the amount of data moved)
  – True for both serial and parallel programs
  – For parallel programs, data movement means data communication between host and device (GPU) or between different nodes (distributed memory systems)

• Reduce the memory footprint

• When developing a parallel program, it’s important to know what to expect in terms of speedup and scaling
Cannon’s Algorithm
Cannon’s Algorithm

• Assume
  – the number of processes $p$ is a perfect square
  – the matrices are $N \times N$ square
• Arrange the processes into a 2-D $\sqrt{p} \times \sqrt{p}$ process grid
  – For each matrix, each process is assigned with a block of $N/\sqrt{p} \times N/\sqrt{p}$
\[
\begin{align*}
A_{nn} \times B_{nn} &= C_{nn} \\
\sum_{k=1}^{N} a_{i,k} \cdot b_{k,j} &= c_{i,j}
\end{align*}
\]

Working by elements
C(1,2) = A(1,0) X B(0,2) + A(1,1) X B(1,2) + A(1,2) X B(2,2)

Working by blocks
The multiplication is completed in $\sqrt{p}$ phases.

$$C(1,2) = A(1,0) \times B(0,2) + A(1,1) \times B(1,2) + A(1,2) \times B(2,2)$$

Three phases: 
- phase1
- phase2
- phase3

The multiplication is completed in $\sqrt{p}$ phases.
Cannon’s Algorithm

- Two stages
  - Skew the matrices so everything aligns properly
    - Shift row i of A by i columns to the left
    - Shift column j of B by j rows to the up
  - Shift and multiply
    - Each process calculate the local product and add into the accumulated sum
    - Shift A by 1 column to the left
    - Shift B by 1 row to the up
    - Repeat $\sqrt{p}$ times
- All shifts wrap around (circular)
Process $P(1,2)$ owns

After initialization: $A(1,2), B(1,2), C(1,2)$

After skewing: $A(1,0), B(0,2), C(1,2)$

Shifting once: $A(1,1), B(1,2), C(1,2)$

Shifting twice: $A(1,2), B(2,2), C(1,2)$
Cannon’s Algorithm – Pseudo Code

\[
\text{For } i = 0 \text{ to } \sqrt{p} - 1 \\
\quad \text{Shift } A(i,:) \text{ to the left by } i \\
\text{For } j = 0 \text{ to } \sqrt{p} - 1 \\
\quad \text{Shift } B(:,j) \text{ to the up by } j \\
\text{For } k = 0 \text{ to } \sqrt{p} - 1 \\
\quad \text{For } i = 0 \text{ to } \sqrt{p} - 1 \text{ and } j = 0 \text{ to } \sqrt{p} - 1 \\
\quad \quad \quad C(i,j) += A(i,j) \times B(i,j) \\
\quad \quad \text{Shift } A(i,:) \text{ to the left by 1} \\
\quad \text{Shift } B(:,j) \text{ to the up by 1}
\]