Matrix-vector multiplication
Overview

Chapter 8 from *Michael J. Quinn, Parallel Programming in C with MPI and OpenMP*

We want to calculate $c = Ab$, where $A$ is a $m \times n$ matrix, $b$ is a vector of length $n$, and $c$ is a vector of length $m$

Many MPI commands will be involved
Introduction

\[ A = \begin{bmatrix} a_{0,0} & a_{0,1}, & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1}, & a_{1,2} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1}, & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} \]

\[ b = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} \]

\[ c = Ab = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} \]

where \( c_i = a_{i,0}b_0 + a_{i,1}b_1 + a_{i,2}b_2 + \cdots + a_{i,n-1}b_{n-1} \)
Example

\[
A = \begin{bmatrix}
2 & 1 & 3 & 4 & 0 \\
5 & -1 & 2 & -2 & 4 \\
0 & 3 & 4 & 1 & 2 \\
2 & 3 & 1 & -3 & 0
\end{bmatrix}
\quad b = \begin{bmatrix}
3 \\
1 \\
4 \\
0 \\
3
\end{bmatrix}
\quad c = \begin{bmatrix}
19 \\
34 \\
25 \\
13
\end{bmatrix}
\]

For example, \(c_1 = 5 \times 3 + (-1) \times 1 + 2 \times 4 + (-2) \times 0 + 4 \times 3 = 34\).
Sequential algorithm

Matrix-vector multiplication

Input: $a[0..m-1, 0..n-1]$ — matrix with dimension $m \times n$

$b[0..n-1]$ — vector with dimension $n \times 1$

Output: $c[0..m-1]$ — result vector with dimension $m \times 1$

for $i \leftarrow 0$ to $m - 1$
    $c[i] \leftarrow 0$
    for $j \leftarrow 0$ to $n - 1$
        $c[i] \leftarrow c[i] + a[i, j] \times b[j]$
    endfor
endfor
Observations

- Value of $c_i$ is calculated by the an inner product (or dot product) between row $i$ of $A$ and vector $b$
  - needs $n$ multiplications and $n - 1$ additions
- Total computational complexity: $O(mn)$
- Value of $c_i$ does not depend on value of $c_k$
Decomposition of matrix $A$

- Data decomposition of matrix $A$ gives rise to parallelism
- Three strategies of decomposition
  - Rowwise block-striped decomposition
    - one process is responsible for a contiguous group of $\lfloor m/p \rfloor$ or $\lceil m/p \rceil$ rows of matrix $A$
  - Columnwise block-striped decomposition
    - one process is responsible for a contiguous group of $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ columns of matrix $A$
  - Checkerboard block decomposition
    - one process is responsible for a block of matrix $A$
Data distribution of vectors $b$ and $c$

- Two ways of distributing $b$ and $c$: 
  - replicated on all processes 
  - block decomposition 

- Why is it acceptable for each process to store the entire $b$ and/or $c$?
  - storage of $A$: $mn$ values 
  - storage of $b$: $n$ values 
  - storage of $c$: $m$ values
Three parallelization strategies

1. Rowwise block-striped decomposition of matrix $A$, replicated vectors
2. Columnwise block-striped decomposition of matrix $A$, block-decomposed vectors
3. Checkerboard block decomposition of matrix $A$, vectors block decomposed
Examples of decompositions

Rowwise block-striped

Columnwise block-striped

Checkerboard block
Rowwise block-striped decomposition

- Each row of matrix $A$ is a primitive task
- Vectors $b$ and $c$ are replicated among the primitive tasks
- Task $i$ has row $i$ and a copy of $b$
  - it can compute $c_i$ as inner product between row $i$ and vector $b$
- To let each task have a replicated entire vector $c$, communication is needed
  - an all-gather step
- Agglomeration $\Rightarrow$ a group of contiguous rows to a process
Complexity analysis

- Suppose \( m = n \), sequential computational complexity \( O(n^2) \)
- When \( p \) processes are used, each process is responsible for at most \( \lceil n/p \rceil \) rows of \( A \)
  - Computational complexity (without communication) per process: \( O(n^2/p) \)
  - An efficient all-gather asks each process to send \( \lceil \log_2 p \rceil \) messages
    - total number of elements sent per process is \( n(p - 1)/p \)
    - hence, communication complexity is \( O(n + \log_2 p) \)
- Overall complexity: \( O(n^2/p + n + \log_2 p) \)
Replicating a block-mapped vector

After each process finishes its computational part, it has calculated a block of vector $c$

Next objective: use communication to let each process replicate the entire $c$ vector

First, each process needs to allocate memory for the entire $c$ vector

Second, the processes must concatenate their pieces of $c$ into a complete vector $c$

Useful MPI command:

```c
int MPI_Allgatherv(void *sendbuf,
                   int sendcount, MPI_Datatype sendtype,
                   void *recvbuf, int *recvcounts, int *displs,
                   MPI_Datatype recvtype, MPI_Comm comm)
```
Example of MPI_Allgatherv

Process 0

send_buffer

<table>
<thead>
<tr>
<th>c o n</th>
</tr>
</thead>
</table>

send_cnt = 3

receive_cnt

| 3 4 4 |

receive_disp

| 0 3 7 |

⇒

receive_buffer

| c o n c a t e n a t e |

Process 1

send_buffer

| c a t e |

send_cnt = 4

receive_cnt

| 3 4 4 |

receive_disp

| 0 3 7 |

⇒

receive_buffer

| c o n c a t e n a t e |

Process 2

send_buffer

| n a t e |

send_cnt = 4

receive_cnt

| 3 4 4 |

receive_disp

| 0 3 7 |

⇒

receive_buffer

| c o n c a t e n a t e |

Matrix-vector multiplication – p. 14
Columnwise block-striped decomposition

- Each primitive gets a single column of matrix $A$
- Each primitive gets a single value of vector $b$
- Task $i$ first multiplies column $i$ of $A$ with value $b_i$
  - outcome: a vector of partial results
- Then, all tasks communicate with each other:
  - every partial result element $j$ on task $i$ is transferred to task $j$
    - MPI all_to_all communication
- Finally, each task sums up the incoming values
Complexity analysis

- Agglomeration ⇒ a group of contiguous columns to a process
  - Each process is responsible for at most \([n/p]\) columns of \(A\)
  - Each process is assigned with at most \([n/p]\) values of \(b\)
  - When done, each process has at most \([m/p]\) values of \(c\)

- We assume \(m = n\)

- Complexity of initial local multiplication: \(\mathcal{O}(n^2/p)\)

- Complexity of final local sum: \(\mathcal{O}(n)\)

- Complexity of the all-to-all communication:
  - Option 1: \([\log_2 p]\) substeps
    - during each substep, a process sends \(n/2\) values and receives \(n/2\) values
    - complexity as \(\mathcal{O}(n \log p)\)
  - Option 2: each process sends a message to all the \(p - 1\) processes
    - complexity as \(\mathcal{O}(n + p)\)

- Overall complexity: \(\mathcal{O}(n^2/p + n \log p)\) or \(\mathcal{O}(n^2/p + n + p)\)
int MPI_Alltoallv(void *sendbuf,
        int *sendcnts, int *sdispls,
        MPI_Datatype sendtype,
        void *recvbuf,
        int *recvcnts, int *rdispls,
        MPI_Datatype recvtype,
        MPI_Comm comm)

Every process exchanges values with every other process in a communicator
Checkerboard block decomposition

- Each primitive task: a single element of matrix $A$
- The task responsible for $a_{i,j}$ multiplies it by value $b_j$, which yields value $d_{i,j}$
- Each element $c_i$ of the result vector can be calculated as
  $$c_i = \sum_{j=0}^{n-1} d_{i,j}$$
  Communication is thus needed
- Agglomeration $\Rightarrow$ a rectangular block of $A$ to each process
Three principal steps

- A 2D task grid: each process is assigned with $A_{i,j}$, which is a block of $A$
- Suppose vector $b$ is initially distributed to the first column of the task grid
- Step 1: redistribute $b$ such that each process gets a suitable subvector $b_j$
- Step 2: multiply $A_{i,j}$ with $b_j$ on each process
- Step 3: each row of the task grid performs a sum-reduction
Redistribution of $b$

- Suppose the 2D task grid is of dimension $k \times l$
- Initially, vector $b$ is evenly distributed among the $k$ processes in the first column of the task grid
- Objective: vector $b$ should be redistributed among the $l$ tasks in each row of the task grid
  - If $k = l$, one-to-one communication plus broadcast
  - If $k \neq l$, gather, scatter, broadcast
- Need to create MPI communicators that encompass all processes on the same row/column of the task grid
Complexity analysis

Assume $m = n$, and $p$ is a square number.

Each process is responsible for a matrix block of size at most $\left\lceil \frac{n}{\sqrt{p}} \right\rceil \times \left\lceil \frac{n}{\sqrt{p}} \right\rceil$

- hence, the local matrix-vector multiplication has complexity $\mathcal{O}(n^2/p)$

Complexity of redistribution of vector $b$

- each process in the first column of the task grid sends its portion of $b$ to the process in the first row $\Rightarrow$ complexity: $\mathcal{O}(n/\sqrt{p})$

- each process in the first row of the task grid broadcasts its portion of $b$ to the other processes in the same column $\Rightarrow$ complexity: $\mathcal{O}(n \log_2 p / \sqrt{p})$

Complexity of the final reduction-sum: $\mathcal{O}(n \log_2 p / \sqrt{p})$
Exercise

Implement the checkerboard version of the parallel matrix-vector multiplication