HPC-19-2

February 21, 2024

1 Assignment 11 (Canon's Matrix Multiplication)

1. Describe Canon's Matrix Multiplication algorithm.

A. Cannon's algorithm is a distributed matrix multiplication algorithm that works particularly well with two-dimensional meshes. Lynn Elliot Cannon described it for the first time in 1969.

```
Algorithm Overview:

* When multiplying two n × n matrices A and B, we need n × n processing nodes
arranged in a 2D.
* Each processing node (PE) performs the following steps:
1. Initialize: k = (i + j) mod n (to ensure different processors access distinct data).
2. Compute: c[i][j] += a[i][k] × b[k][j] concurrently.
3. Communicate:
    * Send a to PE (i, (j + N - 1) mod n).
    * Send b to PE ((i + N - 1) mod n, j).
    * Receive a’ from PE (i, (j + 1) mod n).
    * Receive b’ from PE ((i + 1) mod n, j).
4. Update: a = a’ and b = b’.

* Repeat the above steps for n iterations, ensuring that each processor accesses
different matrix elements.
* The storage requirements remain constant and independent of the number of processors.
```

2. Implement Canon's Matrix Multiplication using collective communication.
3. Analyze the efficiency of the code.

[1]:
```python
from mpi4py import MPI
import numpy as np
```

[2]:
```python
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

[3]:
```python
import time
import matplotlib.pyplot as plt
```

[4]:
```python
N = 2000
```

[5]:
```python
if N % size != 0:
    raise ValueError("Matrix size N must be divisible by the number of␣
↪processes (size)")

block_size = N // size

print(f"Rank {rank}: Starting execution")

if rank == 0:
    print(f"Rank {rank}: Generating matrices A and B")
    A = np.random.randint(0, 10, (N, N))
    B = np.random.randint(0, 10, (N, N))
    print(f"Rank {rank}: Matrices A and B generated")
else:
    A = None
    B = None

print(f"Rank {rank}: Broadcasting matrices A and B")
```

```
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
```

[6]:
```python
start_time = time.time()
A = comm.bcast(A, root=0)
B = comm.bcast(B, root=0)
end_time = time.time()
print(f"Rank {rank}: Matrices A and B broadcasted")
```

```
Rank 0: Matrices A and B broadcasted
```

[7]:
```python
A_rows = np.zeros((block_size, N), dtype=int)
comm.Scatter(A, A_rows, root=0)

start_time_multiplication = time.time()
C_rows = np.dot(A_rows, B)
end_time_multiplication = time.time()

C = None
if rank == 0:
    C = np.zeros((N, N), dtype=int)

comm.Gather(C_rows, C, root=0)

if rank == 0:
    print("Resultant Matrix C:")
    print(C)
```

```
    print("Broadcasting time:", end_time - start_time, "seconds")
    print("Matrix multiplication time:", end_time_multiplication -␣
 ↪start_time_multiplication, "seconds")
```

```
Resultant Matrix C:
[[40747 41349 41354 … 41260 41169 39328]
 [39955 41279 41428 … 40482 42471 40155]
 [40196 41163 40919 … 40635 39846 39888]
 …
 [40674 41117 40256 … 40439 39831 39460]
 [39447 40800 40591 … 40467 40723 40124]
 [40330 40734 40834 … 41727 41303 40117]]
Broadcasting time: 0.039728403091430664 seconds
Matrix multiplication time: 20.929131031036377 seconds
```

[8]:
```
!mpiexec -n 4 python mpi_scatter_gather.py
```

```
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[41048 39762 40853 … 41543 41611 39830]
 [40264 38648 41718 … 40252 40400 39149]
 [41054 40522 41801 … 41724 41400 41007]
 …
 [41452 39388 41118 … 41178 40805 40147]
 [39720 38856 40107 … 40178 40473 39127]
 [40218 38532 40695 … 40379 40558 39088]]
Broadcasting time: 0.03373289108276367 seconds
Matrix multiplication time: 4.518505096435547 seconds
```

[9]:
```
!mpiexec -n 8 python mpi_scatter_gather.py
```

```
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
```

```
Rank 7: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39642 40083 40950 … 41662 41105 40729]
 [40628 40791 40643 … 41148 41905 39657]
 [40344 41142 40593 … 42146 42763 40476]
 …
 [39234 39860 40460 … 40807 40930 40160]
 [38715 39396 39736 … 41051 39867 39587]
 [40204 40537 40572 … 41915 41841 40698]]
Broadcasting time: 0.066436767578125 seconds
Matrix multiplication time: 2.638921022415161 seconds
```

[10]:
```
!mpiexec -n 16 python mpi_scatter_gather.py
```

```
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 15: Starting execution
Rank 15: Broadcasting matrices A and B
Rank 15: Matrices A and B broadcasted
Rank 9: Starting execution
Rank 9: Broadcasting matrices A and B
Rank 9: Matrices A and B broadcasted
Rank 11: Starting execution
Rank 11: Broadcasting matrices A and B
Rank 11: Matrices A and B broadcasted
Rank 10: Starting execution
Rank 10: Broadcasting matrices A and B
```

```
Rank 10: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 13: Starting execution
Rank 13: Broadcasting matrices A and B
Rank 13: Matrices A and B broadcasted
Rank 14: Starting execution
Rank 14: Broadcasting matrices A and B
Rank 14: Matrices A and B broadcasted
Rank 12: Starting execution
Rank 12: Broadcasting matrices A and B
Rank 12: Matrices A and B broadcasted
Rank 8: Starting execution
Rank 8: Broadcasting matrices A and B
Rank 8: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39895 40041 38242 … 39939 39545 40169]
 [39345 39674 38612 … 39800 40560 38977]
 [40013 39937 38095 … 40518 41096 40771]
 …
 [40008 39913 39012 … 39942 41684 39822]
 [41780 41378 40286 … 41451 42157 41894]
 [40301 40239 39599 … 39864 40844 40837]]
Broadcasting time: 0.22203373908996582 seconds
Matrix multiplication time: 3.5083491802215576 seconds
```

[11]:
```python
N = 2000
if N % size != 0:
    raise ValueError("Matrix size N must be divisible by the number of␣
↪processes (size)")

block_size = N // size

print(f"Rank {rank}: Starting execution")

if rank == 0:
    print(f"Rank {rank}: Generating matrices A and B")
    A = np.random.randint(0, 10, (N, N))
    B = np.random.randint(0, 10, (N, N))
    print(f"Rank {rank}: Matrices A and B generated")
else:
    A = None
    B = None

print(f"Rank {rank}: Broadcasting matrices A and B")
```

```
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
```

[12]:
```python
start_time = time.time()
A = comm.bcast(A, root=0)
B = comm.bcast(B, root=0)
end_time = time.time()
print(f"Rank {rank}: Matrices A and B broadcasted")

A_rows = np.zeros((block_size, N), dtype=int)
comm.Scatter(A, A_rows, root=0)

start_time_multiplication = time.time()
C_rows = np.dot(A_rows, B)
end_time_multiplication = time.time()

start_time_gather = time.time()
C_all = np.zeros((N, N), dtype=int)
comm.Allgather(C_rows, C_all)
end_time_gather = time.time()

if rank == 0:
    print("Resultant Matrix C:")
    print(C_all)
    print("Broadcasting time:", end_time - start_time, "seconds")
```

```
    print("Gathering time:", end_time_gather - start_time_gather, "seconds")
    print("Matrix multiplication time:", end_time_multiplication -␣
↪start_time_multiplication, "seconds")
```

```
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40262 41419 41149 … 40627 41185 39118]
 [40512 40635 39829 … 40602 41313 39061]
 [41364 40627 40990 … 40016 41141 40741]
 …
 [40798 40859 41008 … 39674 40921 40082]
 [40201 40859 40919 … 40100 40550 39726]
 [41429 40674 40897 … 39753 41346 39610]]
Broadcasting time: 0.036681175231933594 seconds
Gathering time: 0.007696866989135742 seconds
Matrix multiplication time: 21.168152570724487 seconds
```

[13]:
```
!mpiexec -n 4 python MPI_Allgather.py
```

```
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39268 39927 39869 … 39620 39577 39546]
 [40260 41715 40449 … 41332 40204 40137]
 [39132 40005 39775 … 40137 39549 40140]
 …
 [40205 40717 40398 … 40964 40263 40094]
 [40544 41437 40248 … 42028 40126 40540]
 [40018 40157 40004 … 41190 39445 40415]]
Broadcasting time: 0.031401872634887695 seconds
Gathering time: 0.011379480361938477 seconds
Matrix multiplication time: 4.781134605407715 seconds
```

[14]:
```
!mpiexec -n 8 python MPI_Allgather.py
```

```
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
```

```
Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40351 39324 40644 … 40178 40722 38738]
 [41633 41139 41735 … 41205 41598 40359]
 [42168 40904 41892 … 40341 42459 40579]
 …
 [41025 40337 40735 … 41337 41649 40040]
 [39452 38465 39429 … 39067 39286 38344]
 [41148 40563 41671 … 41142 41364 40395]]
Broadcasting time: 0.06987953186035156 seconds
Gathering time: 0.05123305320739746 seconds
Matrix multiplication time: 2.8613476753234863 seconds
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
```

[15]:
```
!mpiexec -n 16 python MPI_Allgather.py
```

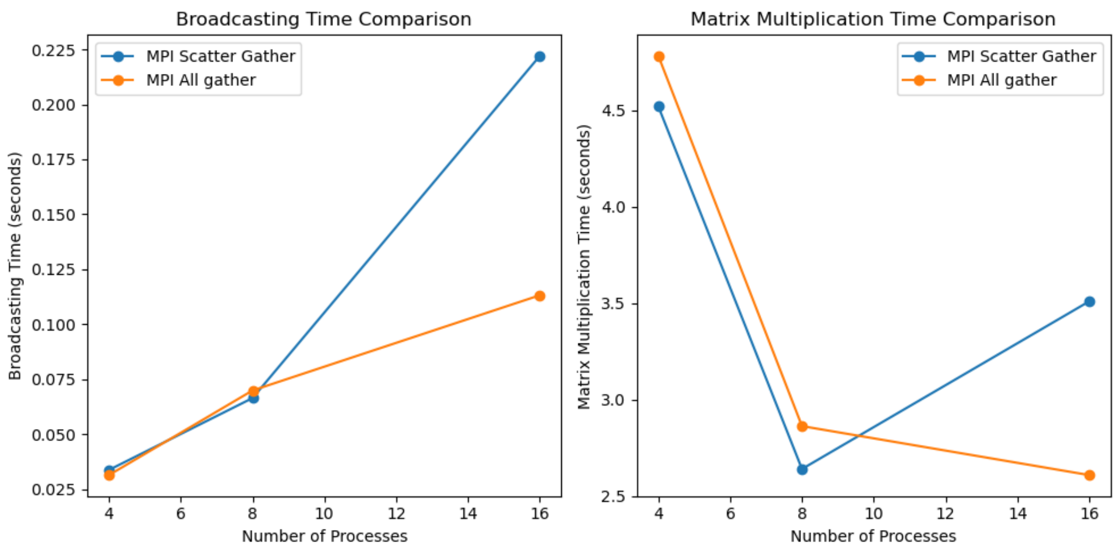
```
Rank 9: Starting execution
Rank 9: Broadcasting matrices A and B
Rank 9: Matrices A and B broadcasted
Rank 10: Starting execution
Rank 10: Broadcasting matrices A and B
Rank 10: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 8: Starting execution
```

```
Rank 8: Broadcasting matrices A and B
Rank 8: Matrices A and B broadcasted
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 11: Starting execution
Rank 11: Broadcasting matrices A and B
Rank 11: Matrices A and B broadcasted
Rank 13: Starting execution
Rank 13: Broadcasting matrices A and B
Rank 13: Matrices A and B broadcasted
Rank 12: Starting execution
Rank 12: Broadcasting matrices A and B
Rank 12: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40098 40513 39370 … 40675 40116 40372]
 [40373 40444 39136 … 40684 39919 40297]
 [40017 40324 39607 … 39830 40199 39921]
 …
 [40336 40603 39674 … 40866 39942 40451]
 [40586 40233 39441 … 40566 39837 40573]
 [40127 41018 40006 … 41142 40468 40251]]
Broadcasting time: 0.11317896842956543 seconds
Gathering time: 0.7809598445892334 seconds
Matrix multiplication time: 2.607423782348633 seconds
Rank 14: Starting execution
Rank 14: Broadcasting matrices A and B
Rank 14: Matrices A and B broadcasted
Rank 15: Starting execution
Rank 15: Broadcasting matrices A and B
Rank 15: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
```

```
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
```

[22]:

```
scatter_gather_processes = [4, 8, 16]
scatter_gather_broadcasting_time = [0.03373289108276367, 0.066436767578125, 0.
↪22203373908996582]
scatter_gather_multiplication_time = [4.518505096435547, 2.638921022415161, 3.
↪5083491802215576]

allgather_processes = [4, 8, 16]
allgather_broadcasting_time = [0.031401872634887695,0.06987953186035156, 0.
↪11317896842956543]
allgather_multiplication_time = [4.781134605407715 ,2.8613476753234863, 2.
↪607423782348633]

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(scatter_gather_processes, scatter_gather_broadcasting_time,␣
↪marker='o', label='MPI Scatter Gather')
plt.plot(allgather_processes, allgather_broadcasting_time, marker='o',␣
↪label='MPI All gather')
plt.xlabel('Number of Processes')
plt.ylabel('Broadcasting Time (seconds)')
plt.title('Broadcasting Time Comparison')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(scatter_gather_processes, scatter_gather_multiplication_time,␣
↪marker='o', label='MPI Scatter Gather')
plt.plot(allgather_processes, allgather_multiplication_time, marker='o',␣
↪label='MPI All gather')
plt.xlabel('Number of Processes')
plt.ylabel('Matrix Multiplication Time (seconds)')
plt.title('Matrix Multiplication Time Comparison')
plt.legend()

plt.tight_layout()
plt.show()
```


Broadcasting Time Comparison
Broadcasting Time (seconds)
Number of Processes
MPI Scatter Gather
MPI All gather
Matrix Multiplication Time Comparison
Matrix Multiplication Time (seconds)
Number of Processes
MPI Scatter Gather
MPI All gather