CP6 PRESENTATION

TOPIC

Distributed-memory parallel version
of Sobel filter with the Message Passing Interface (MPI) utilizing
Varied Grid Decomposition Strategies

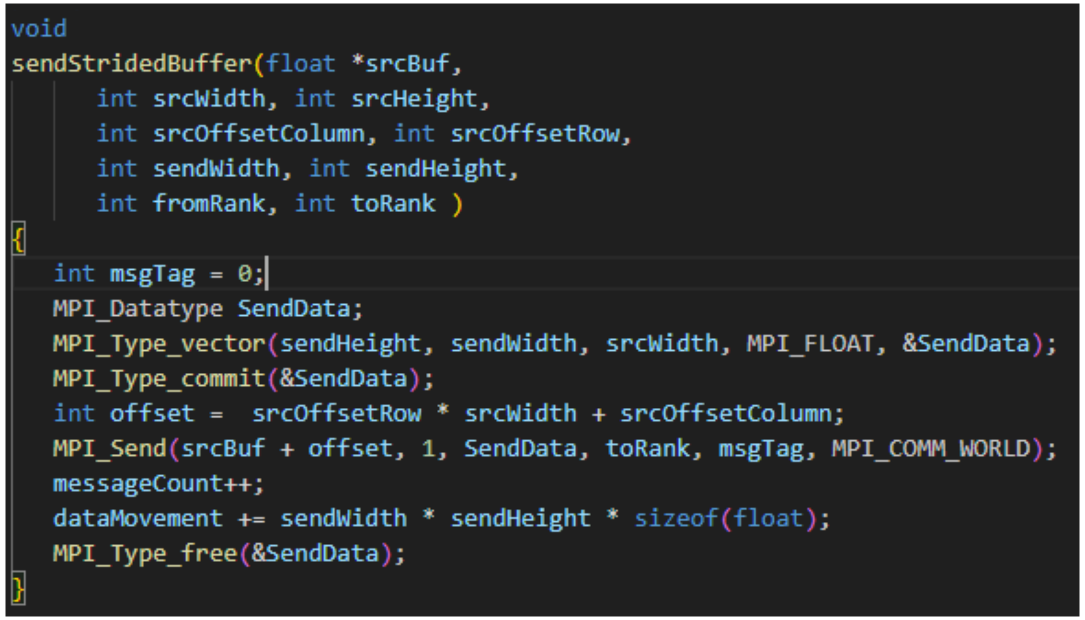
CODE HARNESS

- sobelAllTiles
    - This function designed to integrate the Sobel edge detection algorithm into a parallel computing framework using MPI. It ensures that each MPI process works on a specific part of the image, thereby leveraging the power of distributed computing to efficiently process large images.
- scatterAllTiles
    - This function is instrumental in distributing segments of an image (tiles) across different MPI ranks for parallel processing. If rank = 0 it going to send the data to another rank, otherwise it going to receive the tile from rank 0

CODE HARNESS

- gatherAllTiles
  - This function in the plays a crucial role in aggregating processed image segments (tiles) from various MPI ranks. If rank is other than 0, it going to send the processed tile to rank 0. If the rank is 0, it going to receive processed data from other rank.

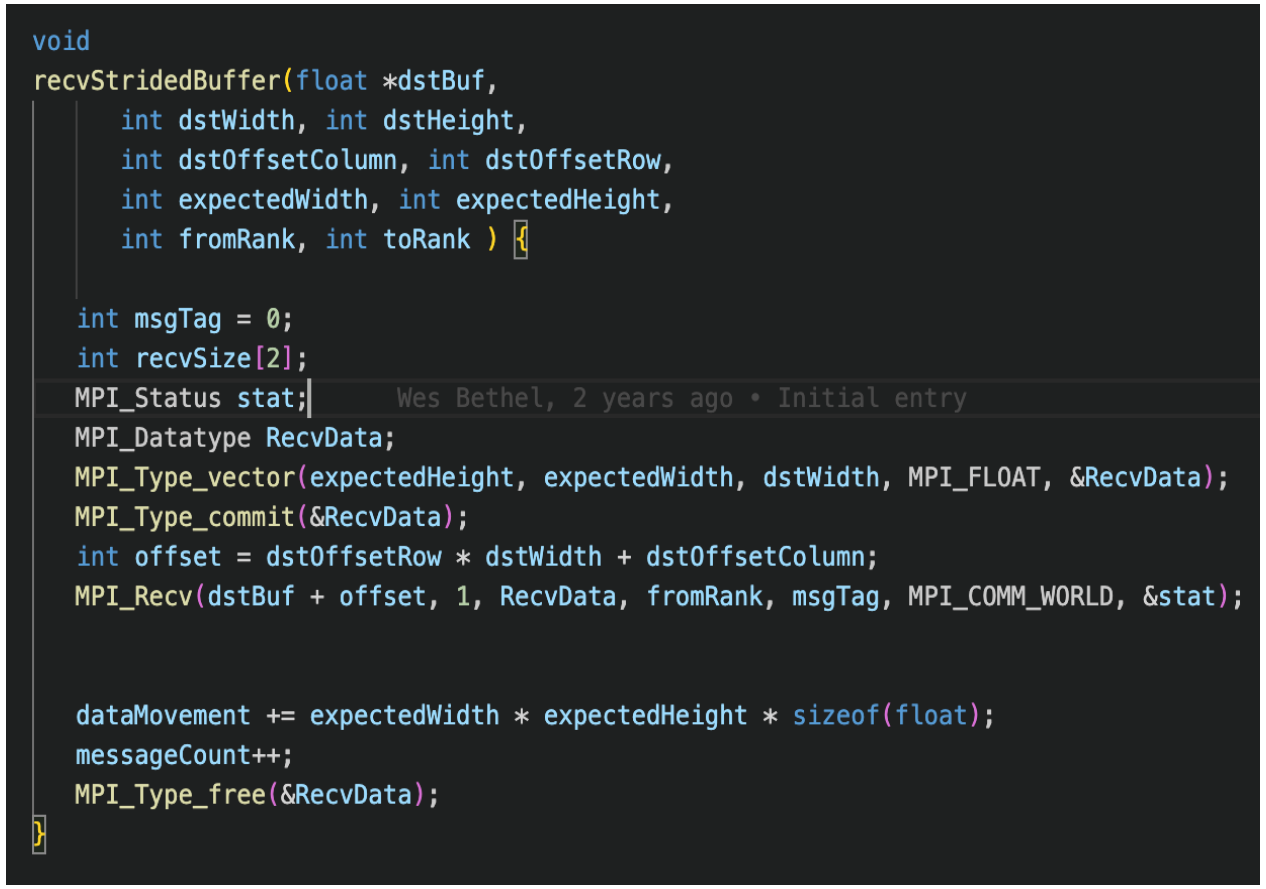
CODE IMPLEMENTATION



```
void
sendStridedBuffer(float *srcBuf,
        int srcWidth, int srcHeight,
        int srcOffsetColumn, int srcOffsetRow,
        int sendWidth, int sendHeight,
        int fromRank, int toRank )
{
    int msgTag = 0;
    MPI_Datatype SendData;
    MPI_Type_vector(sendHeight, sendWidth, srcWidth, MPI_FLOAT, &SendData);
    MPI_Type_commit(&SendData);
    int offset =  srcOffsetRow * srcWidth + srcOffsetColumn;
    MPI_Send(srcBuf + offset, 1, SendData, toRank, msgTag, MPI_COMM_WORLD);
    messageCount++;
    dataMovement += sendWidth * sendHeight * sizeof(float);
    MPI_Type_free(&SendData);
}
```

- sendStridedBuffer
- function call is responsible for sending a specifically defined subregion of a data buffer from one MPI process to another, using a custom data type to accurately describe the shape and size of the data segment being transmitted
- Also keep track on message count and how much data is transferred for further analysis

CODE IMPLEMENTATION



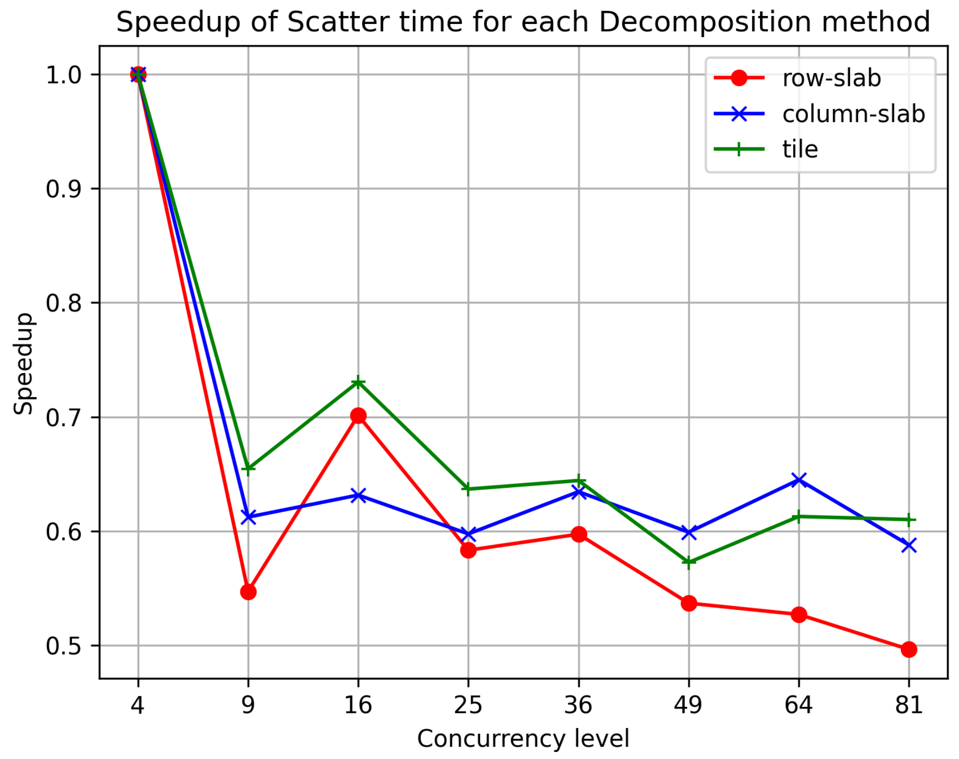
```
void
recvStridedBuffer(float *dstBuf,
        int dstWidth, int dstHeight,
        int dstOffsetColumn, int dstOffsetRow,
        int expectedWidth, int expectedHeight,
        int fromRank, int toRank ) {

   int msgTag = 0;
   int recvSize[2];
   MPI_Status stat;      Wes Bethel, 2 years ago • Initial entry
   MPI_Datatype RecvData;
   MPI_Type_vector(expectedHeight, expectedWidth, dstWidth, MPI_FLOAT, &RecvData);
   MPI_Type_commit(&RecvData);
   int offset = dstOffsetRow * dstWidth + dstOffsetColumn;
   MPI_Recv(dstBuf + offset, 1, RecvData, fromRank, msgTag, MPI_COMM_WORLD, &stat);


   dataMovement += expectedWidth * expectedHeight * sizeof(float);
   messageCount++;
   MPI_Type_free(&RecvData);
}
```

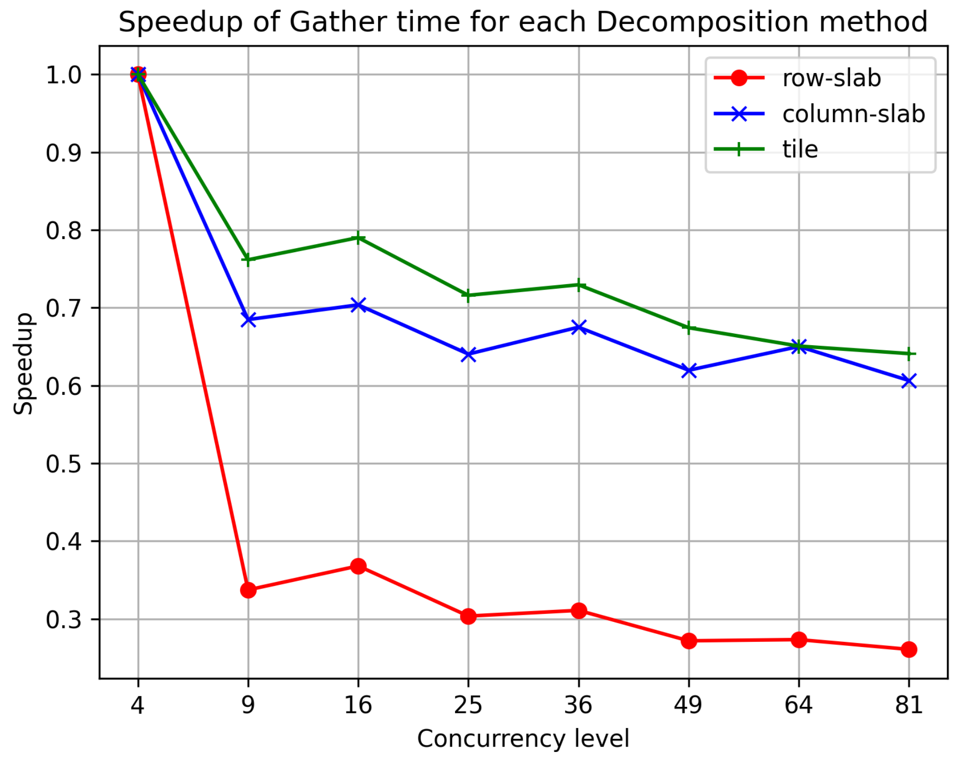
- recvStridedBuffer
- Similar to sendStridedBuffer but for receive end

RESULT



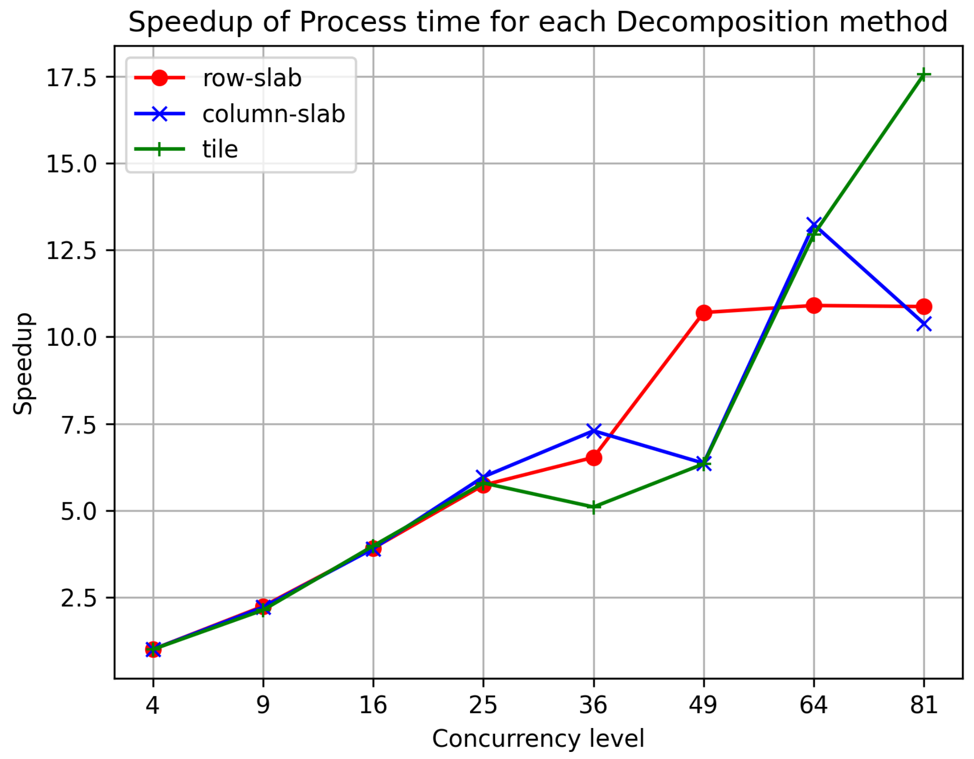
- Scatter Speedup
- In the scatter stage, all strategies showed a decrease in speedup as concurrency increased, with the row-slab strategy exhibiting the most decline. The tile strategy maintained a superior speedup at lower concurrency levels but experienced a significant reduction at a concurrency level of 9

RESULT



- Gather stage, the tile strategy achieved the highest speedup at all concurrency levels, suggesting a more efficient method. Conversely, the row-slab strategy demonstrated a consistent decrease in speedup as concurrency levels rose, indicating potential data collection bottlenecks

RESULT



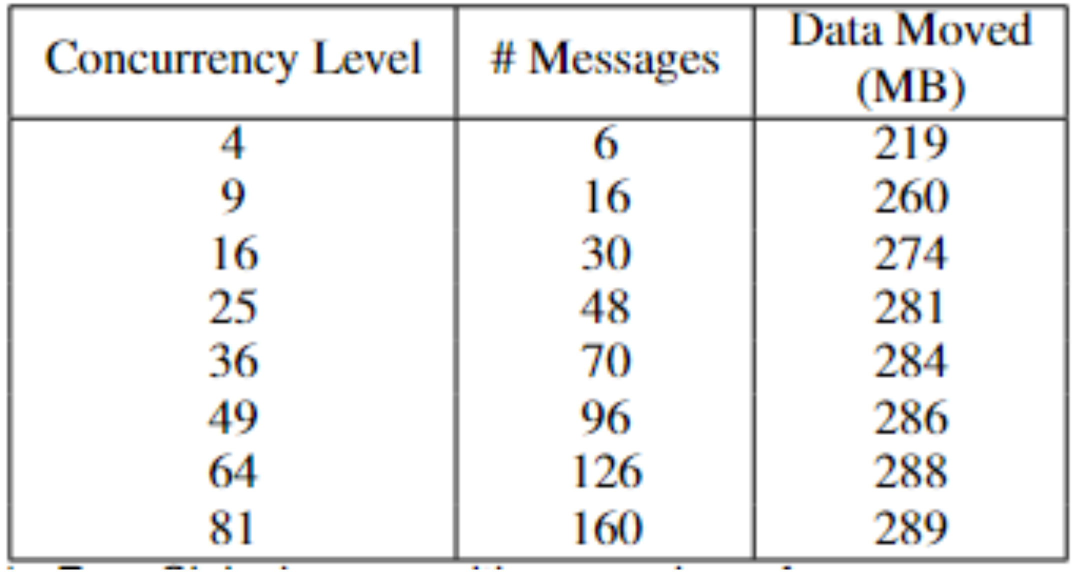
- Process speed up
- The process stage revealed a marked improvement in speedup for all strategies with increasing concurrency levels, particularly noticeable for the tile strategy at the highest concurrency level of 81. Row-slab and column-slab strategies displayed similar performance until a concurrency level of 64, after which the tile strategy's speedup surged ahead

SPEEDUP ANALYSIS

- In scatter and gather stages, a decrease in speedup with increased concurrency levels was observed across all strategies. This pattern suggests that communication overhead becomes more significant at higher concurrency levels

- In the process stage, the rise in speedup at higher concurrency levels across all strategies indicates effective parallel processing. Notably, the tiled strategy excelled at the highest concurrency, suggesting superior scalability and efficient utilization of parallel computation resources.

RESULT



Concurrency Level	# Messages	Data Moved (MB)
4	6	219
9	16	260
16	30	274
25	48	281
36	70	284
49	96	286
64	126	288
81	160	289

- Communication efficiency
- As concurrency grows, so does the communication load, regardless of the decomposition method. This increase is linear for message counts and data volume.

COMMUNICATION ANALYSIS

- As concurrency grows, so does the communication load, regardless of the decomposition method. This increase is linear for message counts and less so for data volume, hinting at the fixed size of the dataset being processed.

- However, when we consider how this overhead affect the scatter, process, and gather phases of execution, we anticipate that the increased messaging and data movement lead to longer runtimes for the scatter and gather phases. However, the processing phase gain the increasing in runtime thanks to the benefits of parallel processing.