PARALLEL IMAGE FILTERING

Presented by John Hunter

University at Buffalo The State University of New York

Problem

Apply an image filter using parallel processing

- Averaging Filter
- Modifying Sequential Strategy
- Performing Parallel Matrix Multiplication

https://studynewyork.us/schools/university-at-buffalo-suny/

Typical Sequential Strategy

1. Apply Padding
2. Creating a kernel
3. Begin at row 0 and column 0
4. Apply the kernel
5. Divide by the kernel size
6. Set the pixel
7. Move to the next location
8. Repeat

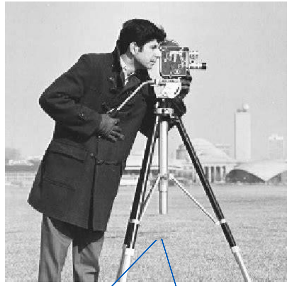
$$\begin{bmatrix} 128 & 205 & 81 \\ 179 & 204 & 211 \\ 189 & 196 & 146 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \frac{1}{9} = 171$$

$$(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 + a_6x_6 + a_7x_7 + a_8x_8 + a_9x_9) \times \frac{1}{9}$$

Modified Sequential Strategy

1. Apply Padding
2. Slightly Alter the kernel
3. Begin at row 0 and column 0
4. Perform matrix multiplication at the point.
5. Sum one column in the resulting matrix.
6. Set the pixel
7. Move to the next location
8. Repeat

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \frac{1}{9} \Rightarrow \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$



$$\begin{bmatrix} 128 & 205 & 81 \\ 179 & 204 & 211 \\ 189 & 196 & 146 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 46 & 46 & 46 \\ 66 & 66 & 66 \\ 59 & 59 & 59 \end{bmatrix}$$

$$46 + 66 + 59 = 171$$

Sequential Matrix Multiplication

Input: 2 N×N matrices A, B

Outputs: 1 N×N matrix C

```
For(x=0; x < N; x++)
        For(y=0; j < N; y++)
                For (i=0; i < N; i++)
                        C[x][y] = C[x][y] + A[x][i] × B[i][y]
                EndFor
        EndFor
EndFor
```

Parallel Strategy

1. Apply Padding
2. Slightly Alter the kernel
3. Begin at row 0 and column 0
4. Perform matrix multiplication at the point in parallel.
5. Sum one column in the resulting matrix.
6. Set the pixel
7. Move to the next location
8. Repeat

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \frac{1}{9} \Rightarrow \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

$$\begin{bmatrix} 128 & 205 & 81 \\ 179 & 204 & 211 \\ 189 & 196 & 146 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 46 & 46 & 46 \\ 66 & 66 & 66 \\ 59 & 59 & 59 \end{bmatrix}$$

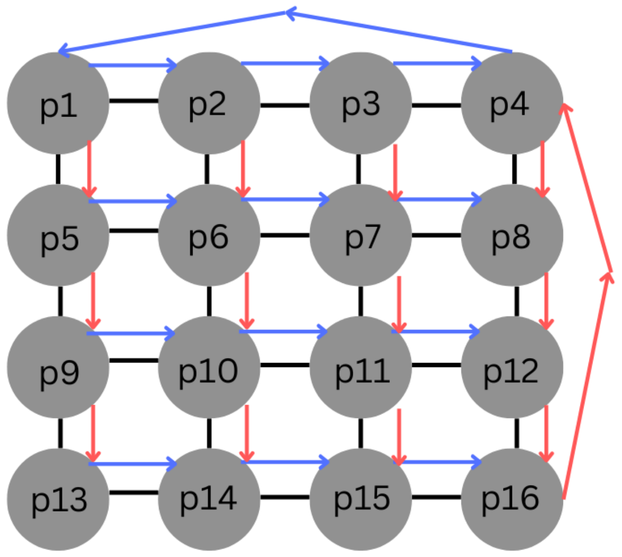
Parallel Matrix Multiplication

Utilize Cannon's Algorithm

Input: 2 N×N matrices A, B

Outputs: 1 N×N matrix C

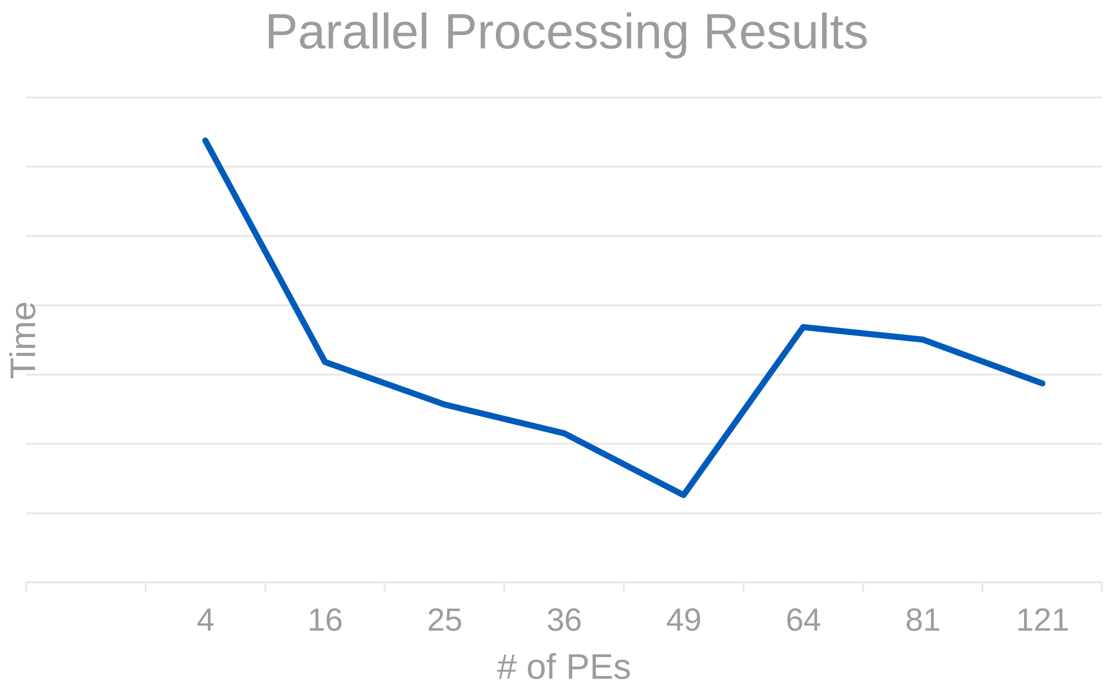
1. Align P processors in a mesh of size sqrt(P) × sqrt(P)
2. Divide A and B into P sub-matrices
3. Assign each submatrices A_i and B_i to processor p_i
4. All processors p_i performs sequential matrix multiplication on A_i and B_i to create C_i
5. All processors p_i pass A_i horizontally to their neighboring processor p
6. All processors p_i pass B_i vertically to their neighboring processor p
7. All processors p_i performs sequential matrix multiplication on the newly acquired A_i and B_i and add the value to C_i
8. Repeat steps 5-7 until every processor p has seen all the data



Parallel Implementation Results
Kernel Size = 1,024 x 1,024

# of PEs	Data Per Processor	Single Matrix Multiply	Overall Image Filter Time
4	262,144	12.75s	336.21s
16	65,536	6.36s	176.33s
25	41,943	5.13s	145.4s
36	29,127	4.3s	122.43s
49	21,399	2.52s	80.32s
64	16,384	7.37s	201.25s
81	12,945	7.01s	191.25s
121	8,665	5.74s	158.5s

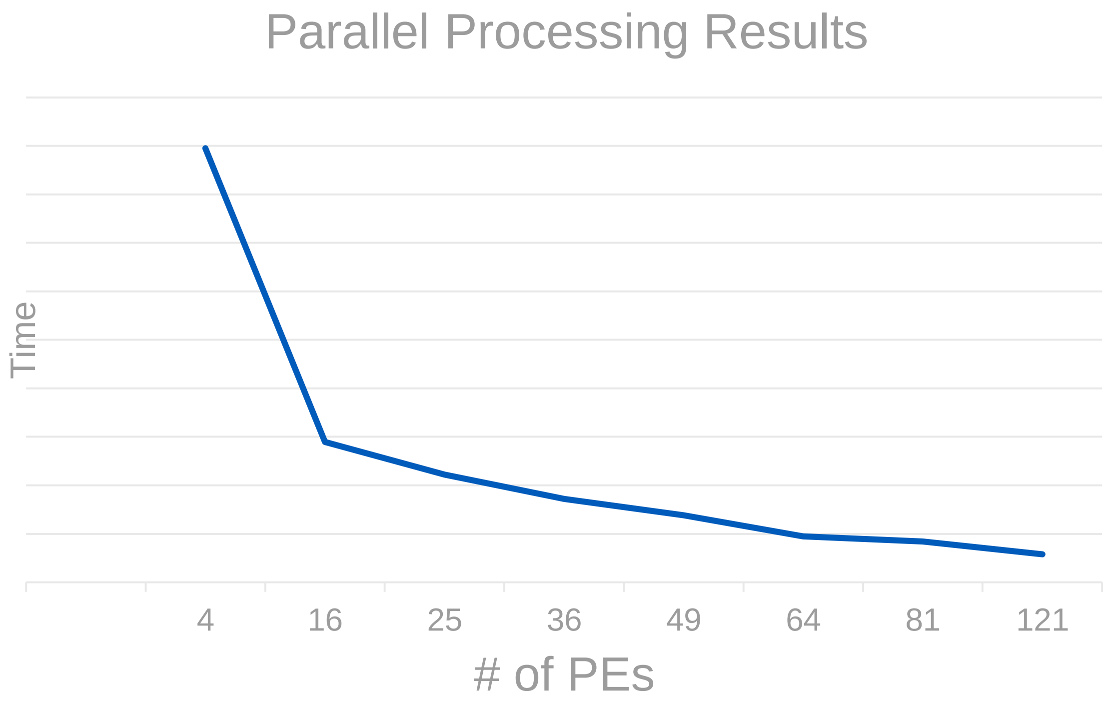
Tests run on a 5 x 5 size image



Parallel Implementation Results
Kernel Size = 4,096 x 4,096

# of PEs	Data Per Processor	Single Matrix Multiply	Overall Image Filter Time
4	4,194,303	447.47s	11,205s
16	1,048,575	144.69s	3,632s
25	671,088	110.91s	2,785s
36	466,033	85.84s	2,162s
49	342,392	69.08s	1,743s
64	262,143	47.39s	1,202s
81	207,126	42.05s	1,067s
121	138,654	28.9s	740s

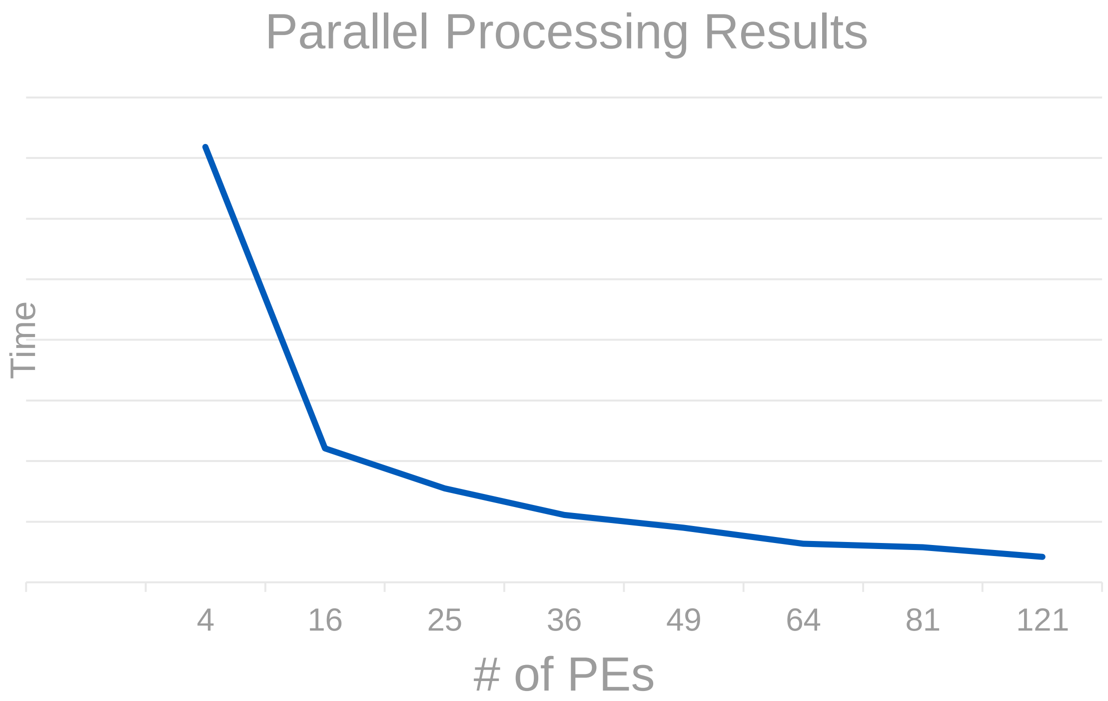
Tests run on a 5 x 5 size image



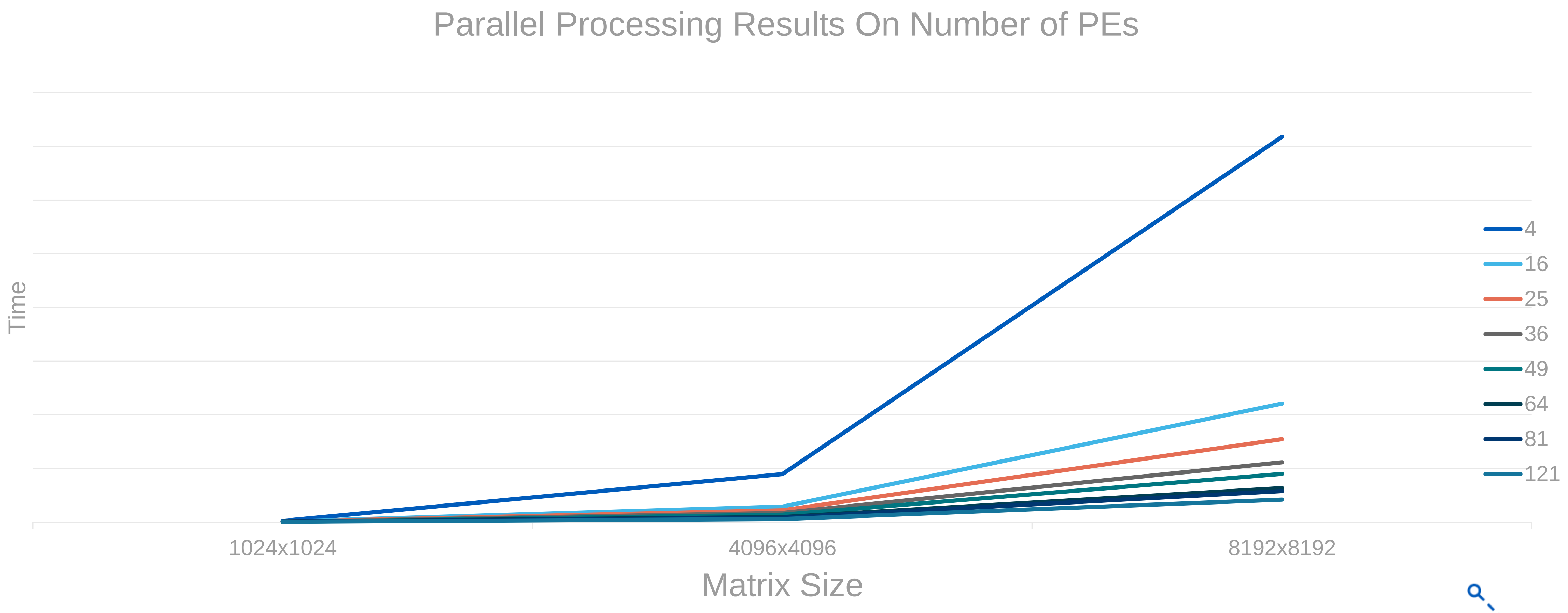
Parallel Implementation Results
Kernel Size = 8,192 x 8,192

# of PEs	Data Per Processor	Single Matrix Multiply	Overall Image Filter Time
4	16,777,216	3,590s	89,766s*
16	4,194,304	1,105s	27,642s
25	2,684,354	774s	19,366s
36	1,864,135	557s	13,941s
49	1,369,568	449s	11,241s
64	1,048,576	318s	7,967s
81	828,504	289s	7,242s
121	554,618	211s	5,275s

Tests run on a 5 x 5 size image
* Estimated due to long runtime



Overall Parallel Implementation Results



Challenges

- Organizing processors into a mesh and assigning matrices
- Message passing and getting correct neighbors
- Determining the correct Slurm parameters

Citations

- [1] https://www.researchgate.net/figure/512-512-grayscale-image-Cameraman_fig1_326140507
- [2] https://en.wikipedia.org/wiki/Cannon%27s_algorithm