Parallel Image Blurring With OpenMP

Prepared by: Sen Pan

Instructor: Dr. Russ Miller

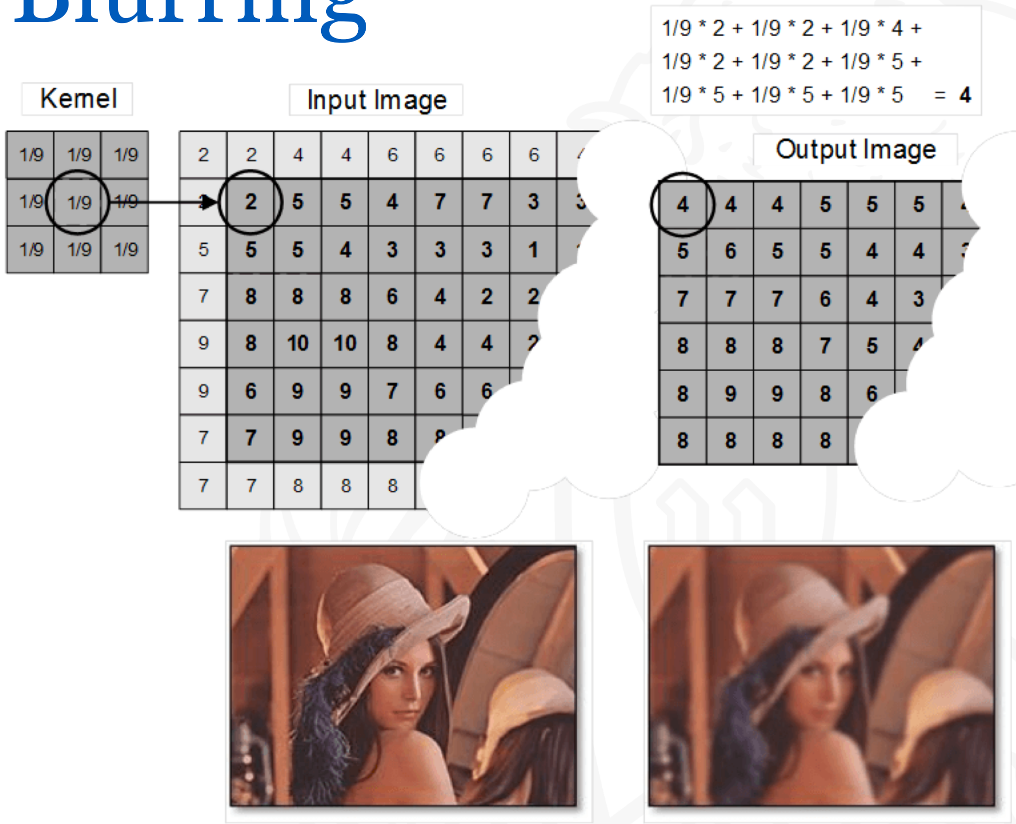
Date : Dec 10, 2020

University at Buffalo The State University of New York

Problem Definition: Image Blurring

Image blurring is a type of image filtering which is everywhere in our daily lives. filtered photos (blurred, sharppend etc.) are ubiquitous in our social media feeds, magazines, books.

The essence of image blurring (or any other type of filtering) is Matrix Multiplication. Which apply a kernel (matrix) on the image matrix to change its value and repeat the multiplication for each of the pixel in image matrix.



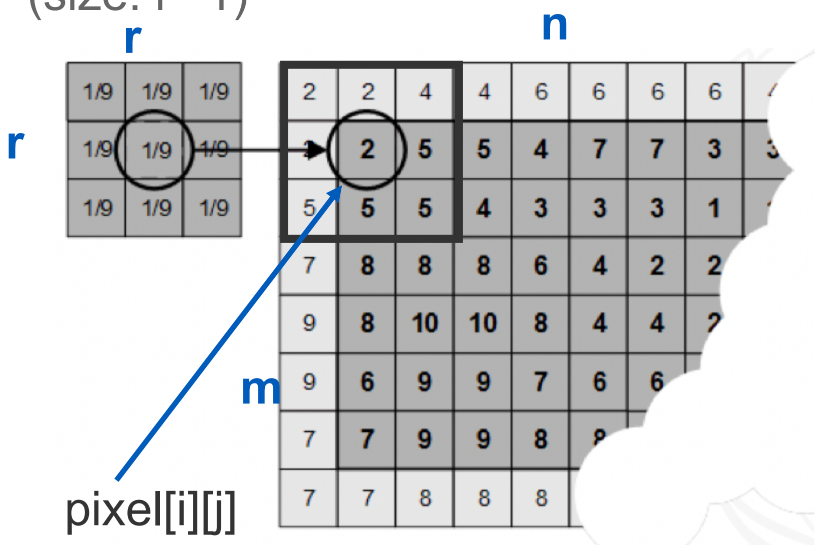
Img 1. image filtering . Source: https://www.imgtec.com/blog/heterogeneous-compute-case-study-image-convolution-filtering/

Image Blurring Sequential algorithm:

- Given a image with size m x n, and a filter of size r x r
- Do a r x r size matrix multiplication for each pixel in the image matrix.
- There are a total of m*n pixels in the image and the time complexity for a matrix multiplication is O(r^3) . Thus, the overall time complexity of the sequential algorithm is O(m * n * r^3).
- For simplicity, we assume m = n = r. So the time complexity of the sequential algorithm is O(n^5).

Filter matrix (size: r * r)

Image matrix (size : m * n)




```
for (i = 0; i < m; i++)
    for (j = 0; i < n; j++)
        matrix_multiplication(); O(r^3)
    end for
end for
```

Overall time complexity of this algorithm is O(n^5)

Image Blurring with parallel matrix multiplication

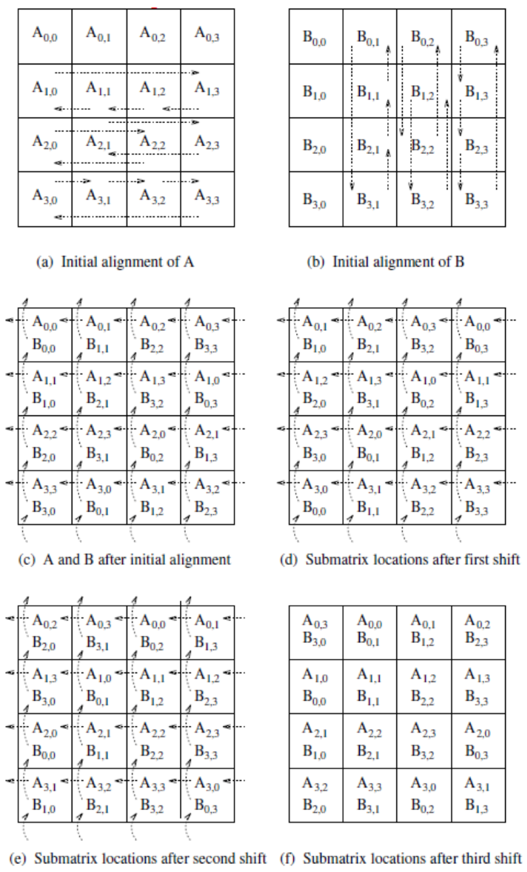
1.Partition these matrices in square blocks p, where p is the number of processes available. So there are sqrt(p) * sqrt(p) submatrices.

2.Each process (Pij) can maintain a submatrix of A matrix (Aij) and a submatrix of B matrix (Bij).

3.Each block is sent to each process, and the copied sub blocks are multiplied together and the results added to the partial results in the C sub-blocks.

4.The A sub-blocks are rolled one step to the left and the B sub-blocks are rolled one step upward.

5.Repeat steps 3 & 4 sqrt(p) times to get the final result.



Img 2. parallel matrix computing. Source: https://iq.opengenus.org/cannon-algorithm-distributed-matrix-multiplication/

Image Blurring with parallel matrix multiplication

Example: two 4 x 4 matrix multiplication with 4 processors.

A =

2	1	5	3
0	7	1	6
9	2	4	4
3	6	7	2

B =

6	1	2	3
4	5	6	5
1	9	8	-8
4	0	-8	5

First, partition each of the matrix into 4 submatrices:

A =

2	1	5	3
0	7	1	6
9	2	4	4
3	6	7	2

B =

6	1	2	3
4	5	6	5
1	9	8	-8
4	0	-8	5

P00 A00 B00	P01 A01 B01
P10 A10 B10	P11 A11 B11

Image Blurring with parallel matrix multiplication

Second, shift first round of row, column data for initial alignment. Then do the local matrix multiplication.

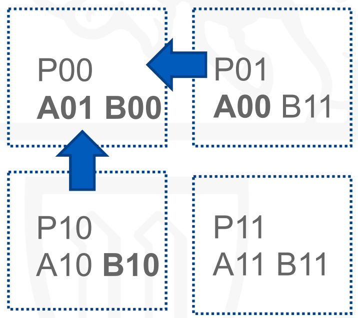
Data shifting between processors:

A =

2	1	5	3
0	7	1	6
9	2	4	4
3	6	7	2

B =

6	1	2	3
4	5	6	5
1	9	8	-8
4	0	-8	5



A0 =

5	3	2	1
1	6	0	7
9	2	4	4
3	6	7	2

B0 =

1	9	2	3
4	0	6	5
6	1	8	-8
4	5	-8	5

Partial result matrix C0 :

C0 =

17	45	10	11
25	9	42	35
62	19	0	-12
42	33	40	-46

Image Blurring with parallel matrix multiplication

Third, do second round of row, column data shifting, then do the local matrix multiplication:

Data shifting between processors:

$$A = \left[\begin{array}{cc|cc} 2 & 1 & 5 & 3 \\ 0 & 7 & 1 & 6 \\ \hline 9 & 2 & 4 & 4 \\ 3 & 6 & 7 & 2 \end{array}\right] \qquad B = \left[\begin{array}{cc|cc} 6 & 1 & 2 & 3 \\ 4 & 5 & 6 & 5 \\ \hline 1 & 9 & 8 & -8 \\ 4 & 0 & -8 & 5 \end{array}\right]$$

P00 A00 B00	P01 A01 **B11**
P10 **A11** B10	P11 **A10 B01**

$$A0 = \left[\begin{array}{cc|cc} 2 & 1 & 5 & 3 \\ 0 & 7 & 1 & 6 \\ \hline 4 & 4 & 9 & 2 \\ 7 & 2 & 3 & 6 \end{array}\right] \qquad B0 = \left[\begin{array}{cc|cc} 6 & 1 & 8 & -8 \\ 4 & 5 & -8 & 5 \\ \hline 1 & 9 & 2 & 3 \\ 4 & 0 & 6 & 5 \end{array}\right]$$

Partial result matrix C1 :

$$C1 = \left[\begin{array}{cc|cc} 16 & 7 & 16 & -25 \\ 28 & 35 & -40 & 22 \\ \hline 20 & 36 & 30 & 37 \\ 15 & 63 & 42 & 39 \end{array}\right]$$

Image Blurring with parallel matrix multiplication

Finally, update the partial result matrix C1 to C0 to get the final result.

C = C0 + C1 =

17	45	10	11
25	9	42	35
62	19	0	-12
42	33	40	-46

+

16	7	16	-25
28	35	-40	22
20	36	30	37
15	63	42	39

=

33	52	26	-14
53	44	2	57
82	55	30	25
57	96	82	-7

Image Blurring with parallel matrix multiplication

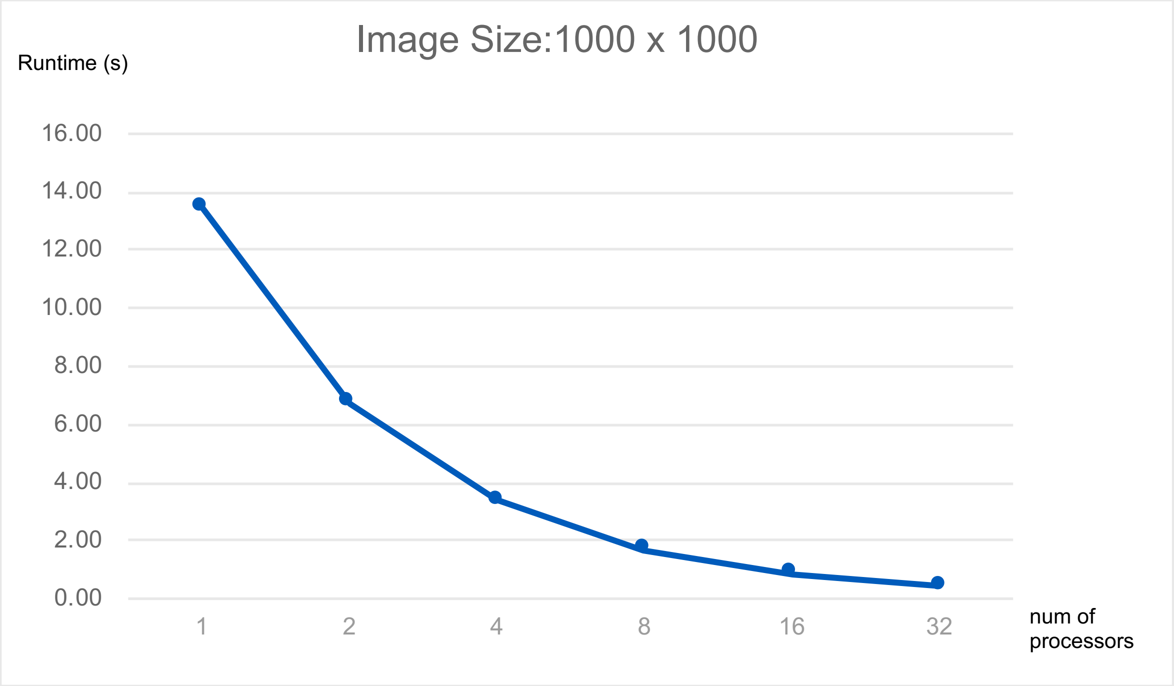
Run parallel image blurring algorithm with OpenMP:

- ❑ Convert input image data into matrix representation and define filter matrix.
- ❑ Write the program for image blurring with Cannon's algorithm.
- ❑ Parallelize the matrix multiplication part of the program using OpenMP.
- ❑ Test the program with different settings to compare the result.

Experiments:

Image size 1000 x 1000 test result:

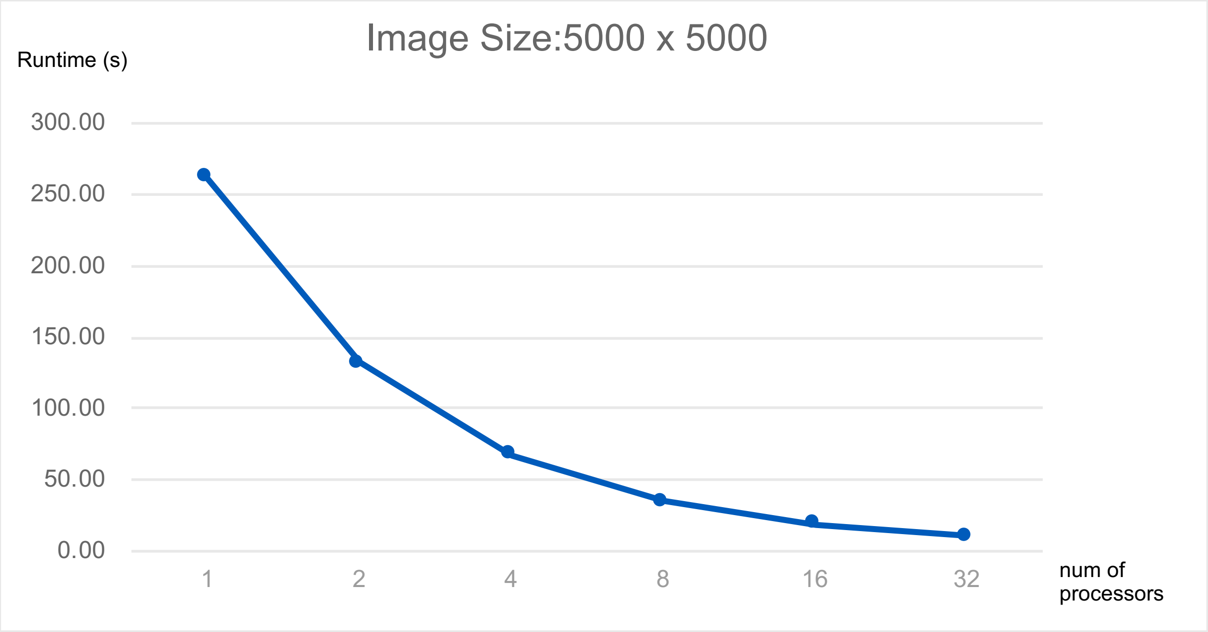
Num of processors	Run time (s)	Speed up
1	13.48	1.0
2	6.72	2.0
4	3.36	4.0
8	1.69	8.0
16	0.85	15.8
32	0.44	30.7



Experiments:

Image size 5000 x 5000 test result:

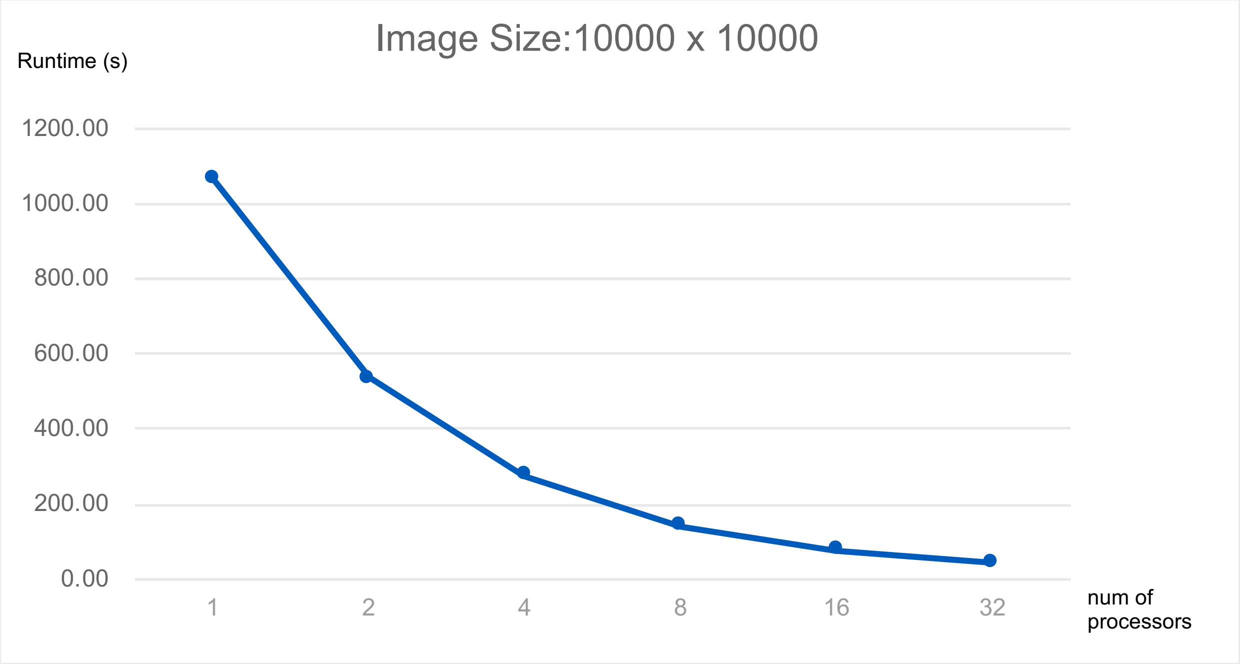
Num of processors	Run time	Speed up
1	261.37	1.0
2	131.65	2.0
4	67.51	3.9
8	35.14	7.4
16	18.61	14.0
32	10.91	24.0



Experiments:

Image size 10000 x 10000 test result:

Num of processors	Run time	Speed up
1	1067.80	1.00
2	535.62	1.99
4	275.84	3.87
8	143.11	7.46
16	76.76	13.91
32	44.45	24.02



Observations:

- Algorithm has very good scalability against input data size:

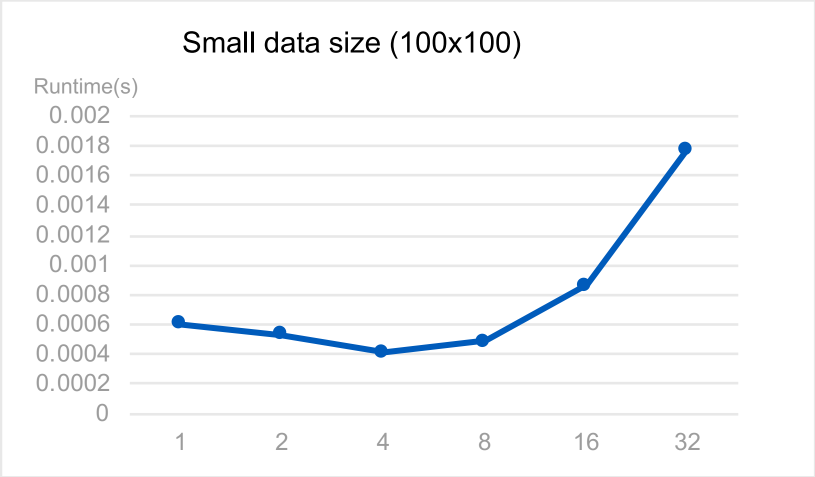
With the same number of processors being constant (32), data size change form 10^6 (1000x1000), $25 \text{ x } 10^6$ (5000 x 5000) to 100 x10^6 (10000 x 10000);

Input data size (relative)	Runtime (s)	Runtime(relative)
1	0.44	1
25	10.91	24.8
100	44.45	101.0

Observations:

- Algorithm doesn't work very well when the data size if very small. Probably due to multithreading message passing overhead over shadows the actual calculating time, which is very small.
- For example, with image size of 100x100:

Num of processors	Runtime(s)
1	0.000605
2	0.000531
4	0.000416
8	0.000488
16	0.000870
32	0.00178



References:

- Russ Miller, "Algorithms Sequential & Parallel: A Unified Approach"
- Larry Meadows, "A Hands-on Introduction to OpenMP ";
- Valentin Stoica, "Parallel Implementation of Image Filtering Algorithms in Multiprocessor Systems";
- Ortega, Patricia, "Parallel Algorithm for Dense Matrix Multiplication" https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Ortega-Fall-2012-CSE633.pdf ;
- https://www.youtube.com/watch?v=nE-xN4Bf8XI&list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG

Thanks!