CONVOLUTION

DIEGO SANZ VILLAFRUELA

Table of Contents

DIEGO SANZ VILLAFRUELA

Report your technique for loop level parallelization

The parallelization technique for the convolution consists on calculating the number of elements that form the structure (array, matrix or the tridimensional array) and split out the elements into NT (number of threads) equal parts. If there are remaining elements, each of the remaining elements will be assigned to a different thread.

Each element has 3 parameters:

- From
- To
- Extra element

2 threads and 8 elements

T1
T1
T1
T2
T2
T2
T1
T2

Thread 1:
From: 0
To: 2
Extra element: 6

Thread 2:
From: 3
To: 5
Extra element: 7

5 threads and 64 elements

T1	T1	T1	T1	T1	T1	T1	T1
T1	T1	T1	T1	T2	T2	T2	T2
T2	T2	T2	T2	T2	T2	T2	T2
T3	T3	T3	T3	T3	T3	T3	T3
T3	T3	T3	T3	T4	T4	T4	T4
T4	T4	T4	T4	T4	T4	T4	T4
T5	T5	T5	T5	T5	T5	T5	T5
T5	T5	T5	T5	T1	T2	T3	T4

Thread 1:
From: 0
To: 11
Extra element: 60

Thread 2:
From: 12
To: 23
Extra element: 61

Thread 3:
From: 24
To: 35
Extra element: 62

Thread 4:
From: 36
To: 47
Extra element: 63

Thread 5:
From: 48
To: 59
Extra element: 63

After each thread has the indexes that have to calculate, there is a calculation step for converting the indexes into the dimensional structure. As it is expected 2-D needs 2 indexes, and so on. I.e. of this calculation:

The index of the elements of the thread 1 of the previous examples are transformed into:

index	0	1	2	3	4	5	6	7	8	9	10	11	60

Index i = N / NT **Index j = N % NT**

N number of elements
NT number of threads

Index i	0	0	0	0	0	0	0	0	1	1	1	1	7
Index j	0	1	2	3	4	5	6	7	0	1	2	3	4

After that, the kernel is multiplied by the part of the input (image) of the same size whose central element is indexed by the indexes calculated above:

```
Convolution 1D
    For index : indexes
        result[index] = calculateSingleValue(index);
```

```
Convolution 2D
    For index : indexes
        i = getI(index)
        j = getJ(index)
        result[i,j] = calculateSingleValue(i , j);
```

```
Convolution 3D
    For index : indexes
        i = getI(index)
        j = getJ(index)
        z = getK(index)
        result[i,j,k] = calculateSingleValue(i , j, k);
```

CaculateSingleValue method returns the result of applying convolution on a single point. It is one step of the convolution.

Once the convolution has been calculated by the threads, the result is written on a file like an image.

DIEGO SANZ VILLAFRUELA

Report the system configuration on which you are running the code for E1, E2, and E3

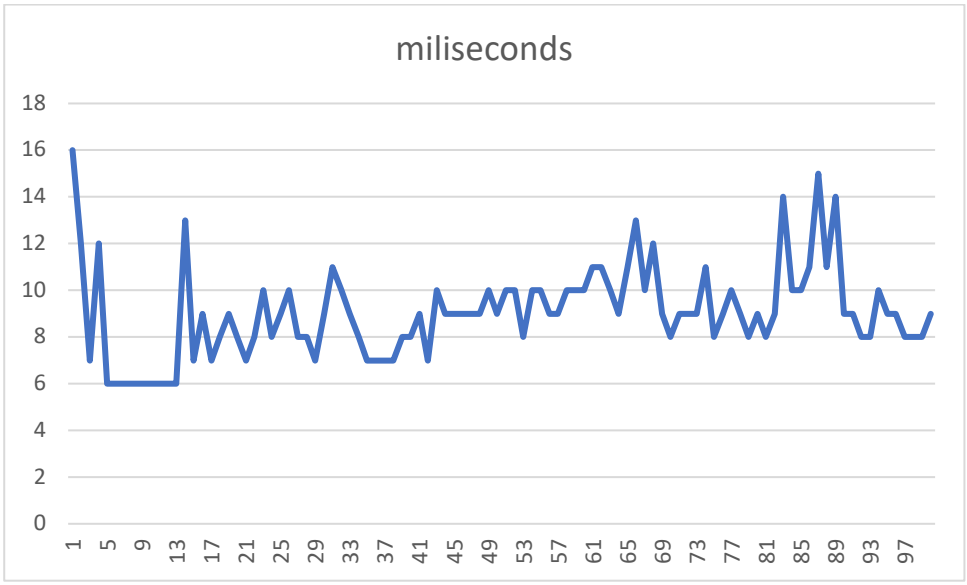
Model Name	MacBook Pro
Model Identifier:	MacBookPro12,1
Processor Name:	Intel Core i7
Processor Speed:	3,1 GHz
Number of Processors:	1
Total Number of Cores:	2
L2 Cache (per Core):	256 KB
L3 Cache:	4 MB
Memory:	16 GB

DIEGO SANZ VILLAFRUELA

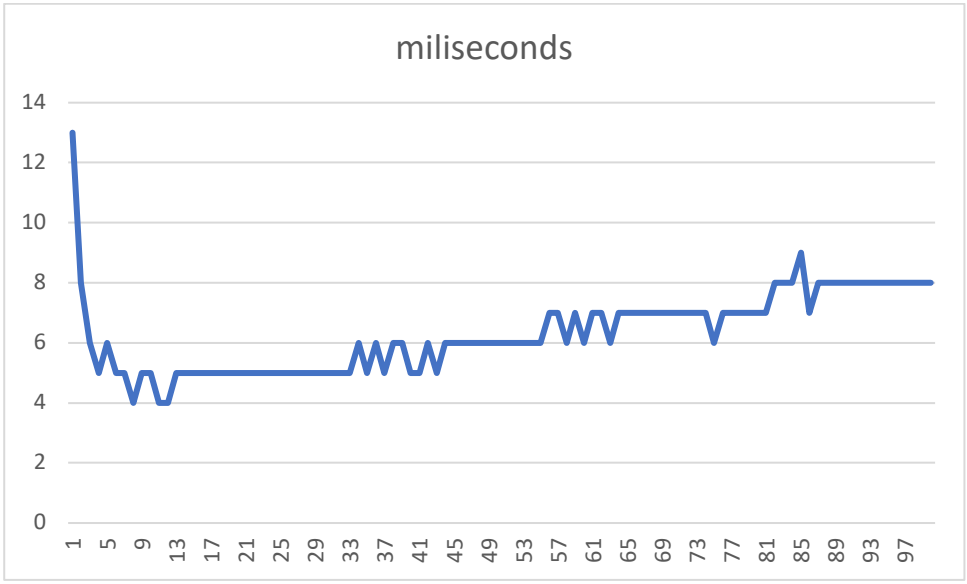
Report the execution time for each number of threads for E1, E2, and E3

The X-axis represents the **number of threads** and the Y-axis represents the **time in milliseconds** needed to calculate the convolution.

Input of **1000000 elements** and kernel size **5**:



Input: WomanImage.jpg. **512 * 512 elements** and kernel size **3 x 3**:



It could have been possible to use memorization techniques or dynamic programming to improve the time of the convolution.

DIEGO SANZ VILLAFRUELA

Report your explanation about differences between the execution time for E1.1, E1.2, E1.3

The main differences between the execution times between 1-D, 2-D, and 3-D convolutions are explained by the complexity of the algorithms that implement them. Using the big O notation, we can see clearly the difference in the execution time.

	Input	Kernel	Complexity = Input * kernel
1-D Convolution	N	A	O (N * A)
2-D Convolution	N*M	A*B	O (N*M*A*B)
3-D Convolution	N*M*O	A*B*C	O (N*M*O*A*B*C)

For checking its implementation, please check the code.

Report your explanation about the relationship between number of threads and size of the kernel

The number of threads and the size of the kernel have a significant effect in the times of the convolution. Both of them affects it. However, it is important to point out that the improvement the number of threads have a less important influence in the time if the size of the kernel is big.

The reason is that there is a parallelism in the input, that divides the input into equal parts, but there is not any papalism when we are multiplying a part of the input by the kernel.

Another parallelism algorithm could have been implemented to make the algorithm efficient when the kernel is big. Memorization techniques or dynamic programming could have been an option.

Speedup in each case

“The speedup is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors”.

$$S = \frac{T_S}{T_P}$$

	Sequential Best time (milliseconds)	Parallel Best time (milliseconds)	Speedup
1-D Convolution	16	6	2.66
2-D Convolution	15	4	3.25
3-D Convolution	No calculated	No calculated	No calculated

DIEGO SANZ VILLAFRUELA

How to run the code

The project is built using maven.

- To compile the project: **mvn clean package**
- To execute it**: java -jar target/theJarFile.jar**