UNIVERSITÀ
DEGLI STUDI
FIRENZE

Sequential and parallel algorithm implementations

Andrea Moscatelli

Course:
Parallel computing

Teacher:
Marco Bertini

Outline

- **SAD in JAVA**
  - Sequential/parallel implementations
  - Performance analysis
- **Bigram/Trigram detection with CUDA**
  - CPU/GPU implementations
  - Performance analysis

Project 1

SAD in JAVA

SAD Technique

SAD = **S**um of **A**bsolute **D**ifferences

It's a very simple *pattern recognition* technique based on algebraic differences between *pattern* and *dataset*.

pattern

2	1	4

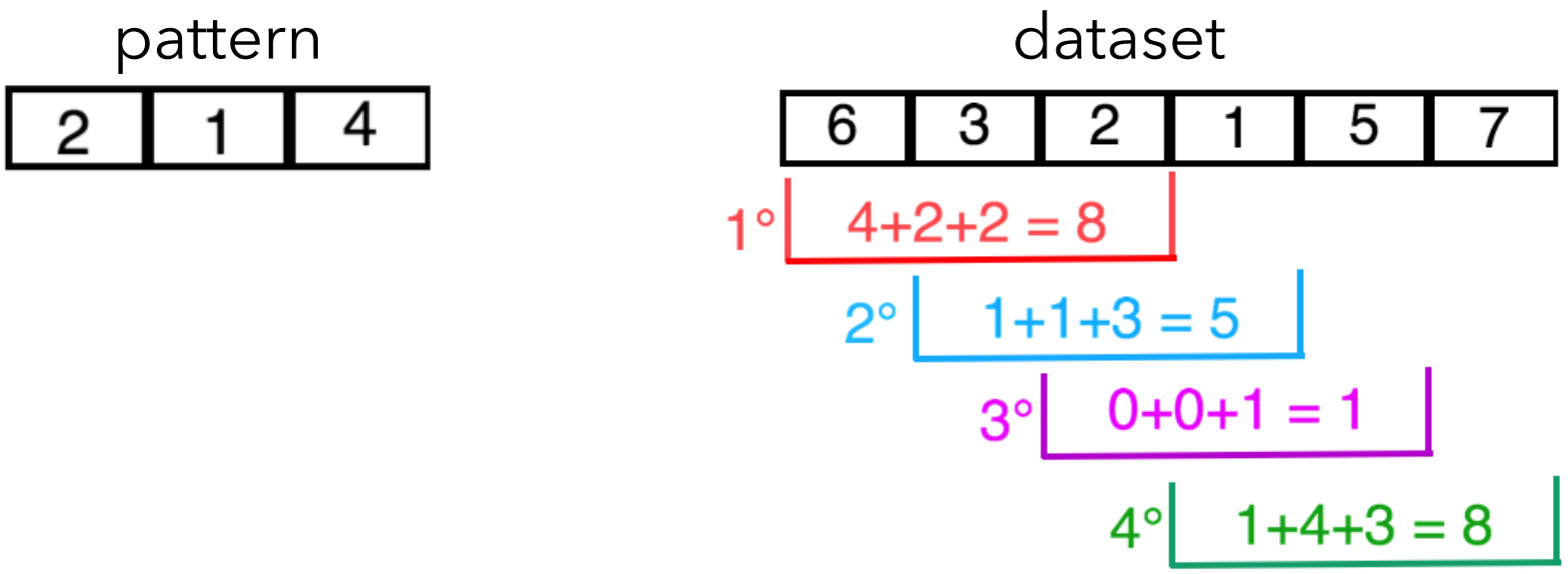
dataset

6	3	2	1	5	7

SAD Technique

SAD = **S**um of **A**bsolute **D**ifferences

It's a very simple *pattern recognition* technique based on algebraic differences between *pattern* and *dataset*.

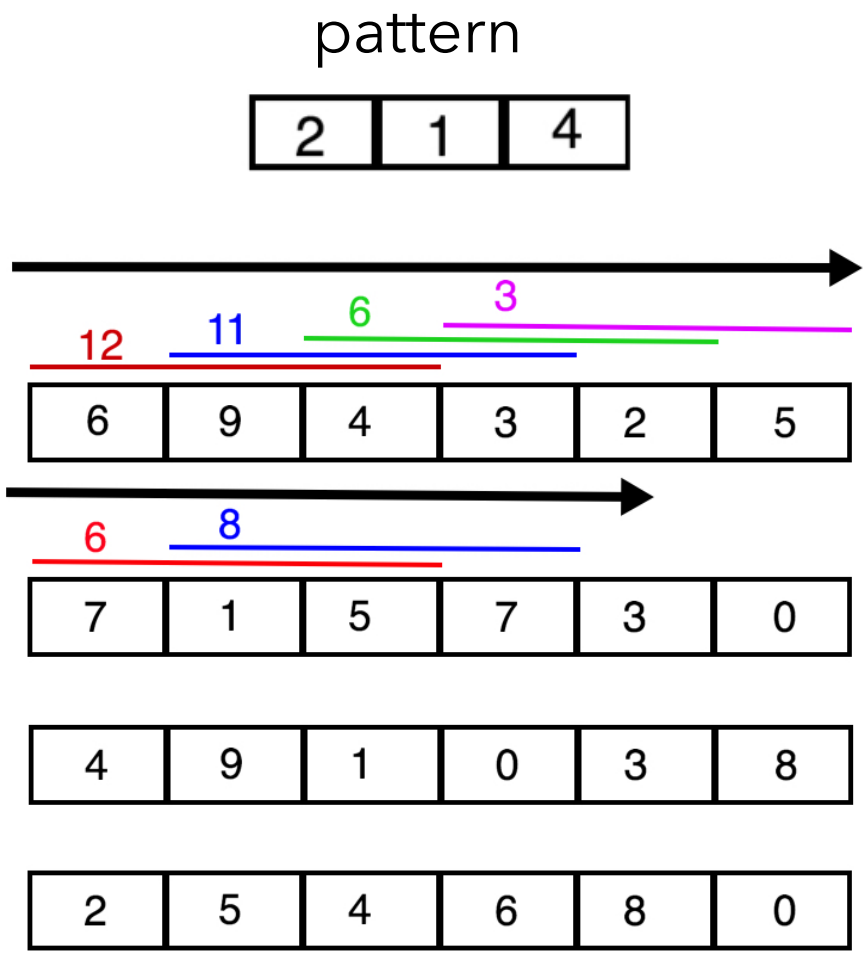


Java implementation

Seqential:

```
sequentialMethod(){
        return findBestSAD(0, datasetHeight);
}


findBestSAD (startIndex, endIndex) {
        bestSAD = MAX_VALUE;
        foreach i in [startIndex, endIndex]{
                foreach j in [0, datasetLenght]{
                        sad = calculateSAD(i, j);
                        if (sad < bestSAD)
                                bestSAD = sad;
                }
        }
        return bestSAD;
}
```

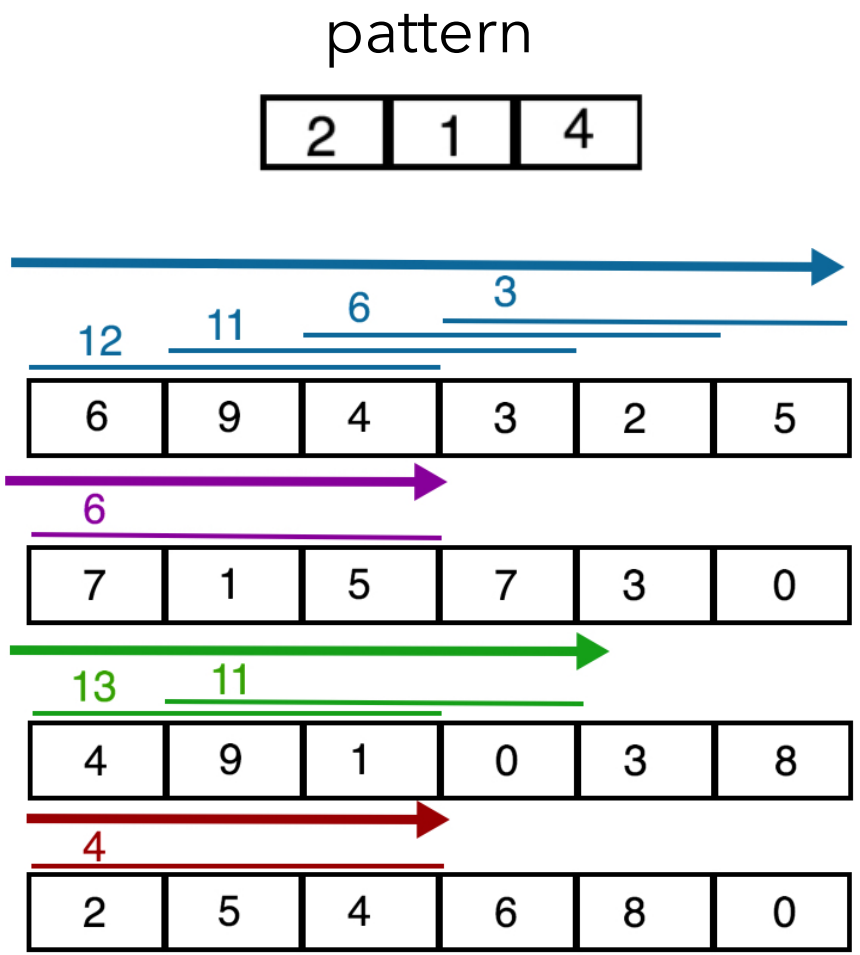


Java implementation

Parallel:

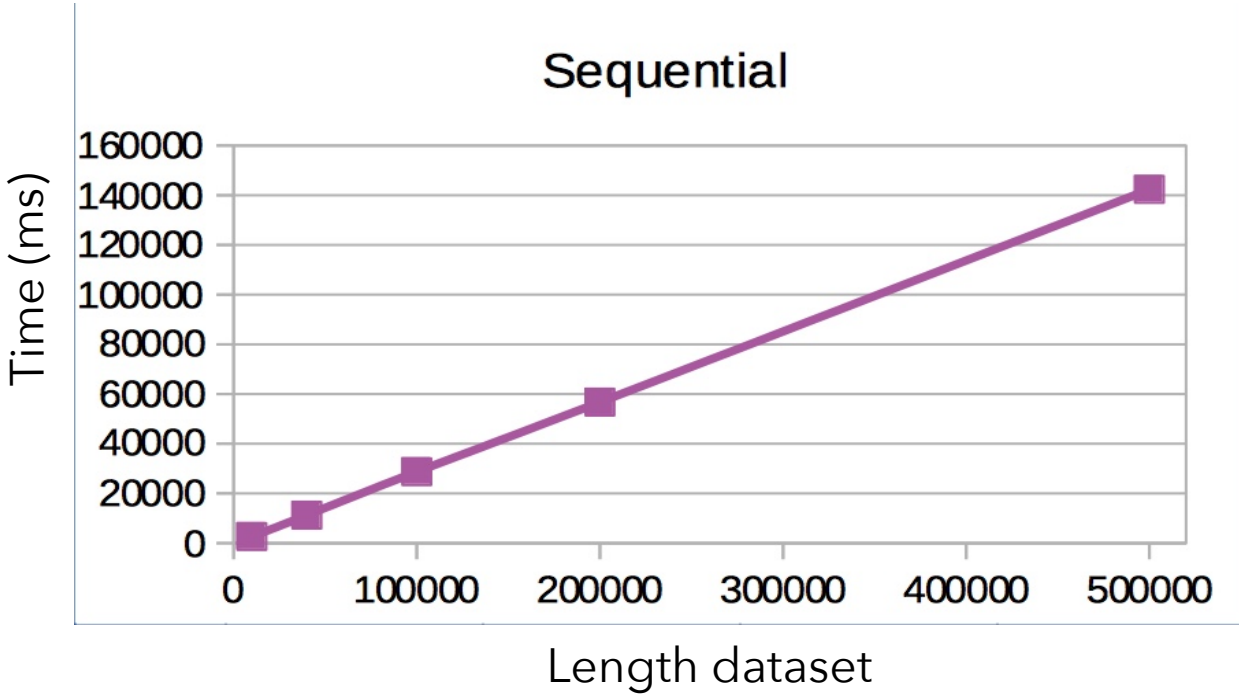
```
parallelMethod(){
 slice = datasetLenght / numberOfThreads;
 bestSAD = MAX_VALUE;
 executor = newExecutorFixedThreadPool(numberOfThreads);
 for i from 0 to n-1{
    executor.add(new Thread(
                    sad = findBestSAD(i * slice, (i+1) * slice);
                    bestSAD = min(sad, bestSAD);
                    ));
 }
 executor.runAll();
 return bestSAD;
}

findBestSAD (startIndex, endIndex) {
        bestSAD = MAX_VALUE;
        foreach i in [startIndex, endIndex]{
                foreach j in [0, datasetLenght]{
                        sad = calculateSAD(i, j);
                        if (sad < bestSAD)
                                bestSAD = sad;
                }
        }
        return bestSAD;
}
```



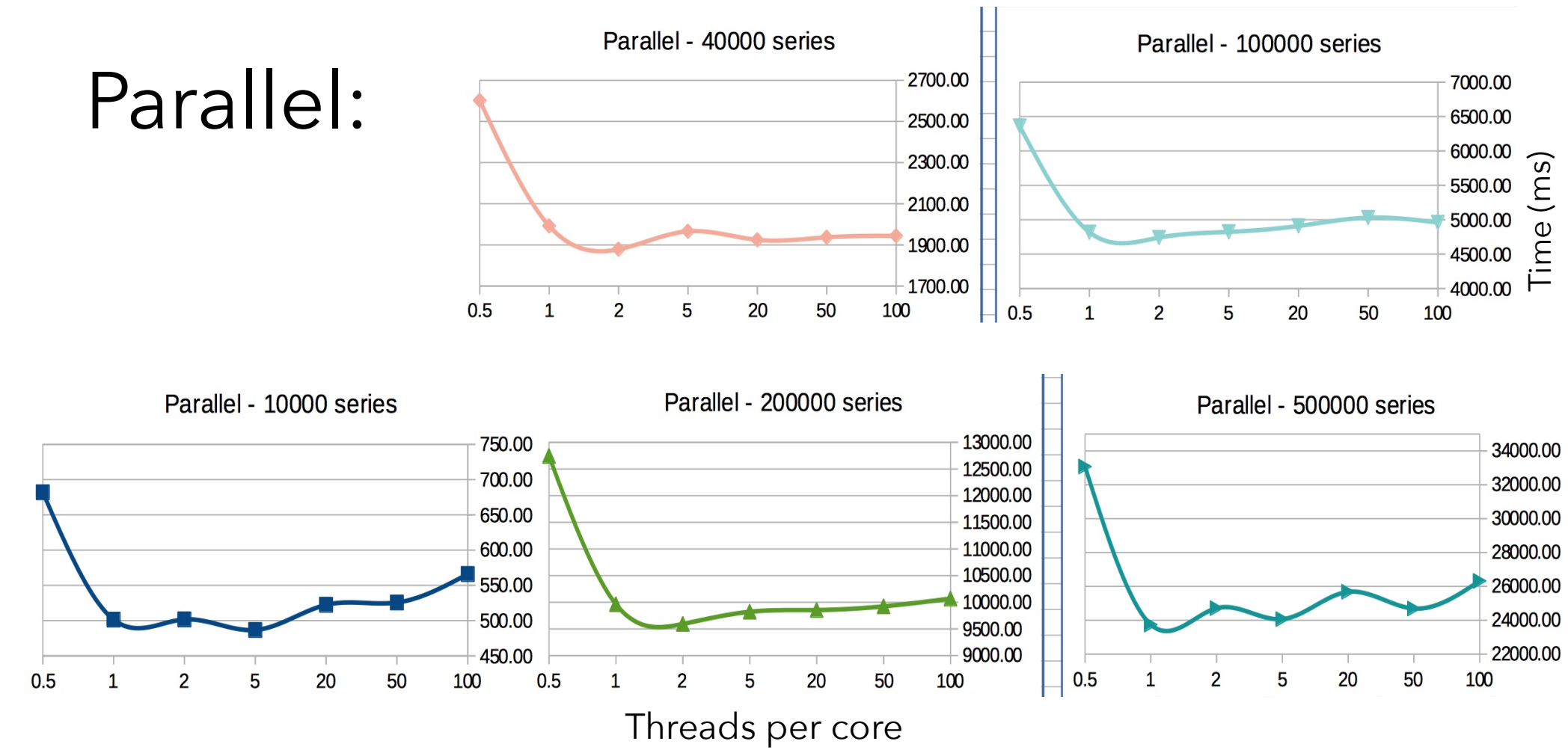
Performance

Seqential:



Performance

Parallel:



Conclusion

- Parallel version is **always better** then the sequential one
- **1-2 threads per core** it's usually enough to have good performances
- Over 5 threads per core is rarely useful

Project 2

Bigram and
Trigram in CUDA

Bigrams and Trigrams

What's a bigram (or trigram)?

It's a couple (or triplet) of adjacent letters or word in a text.

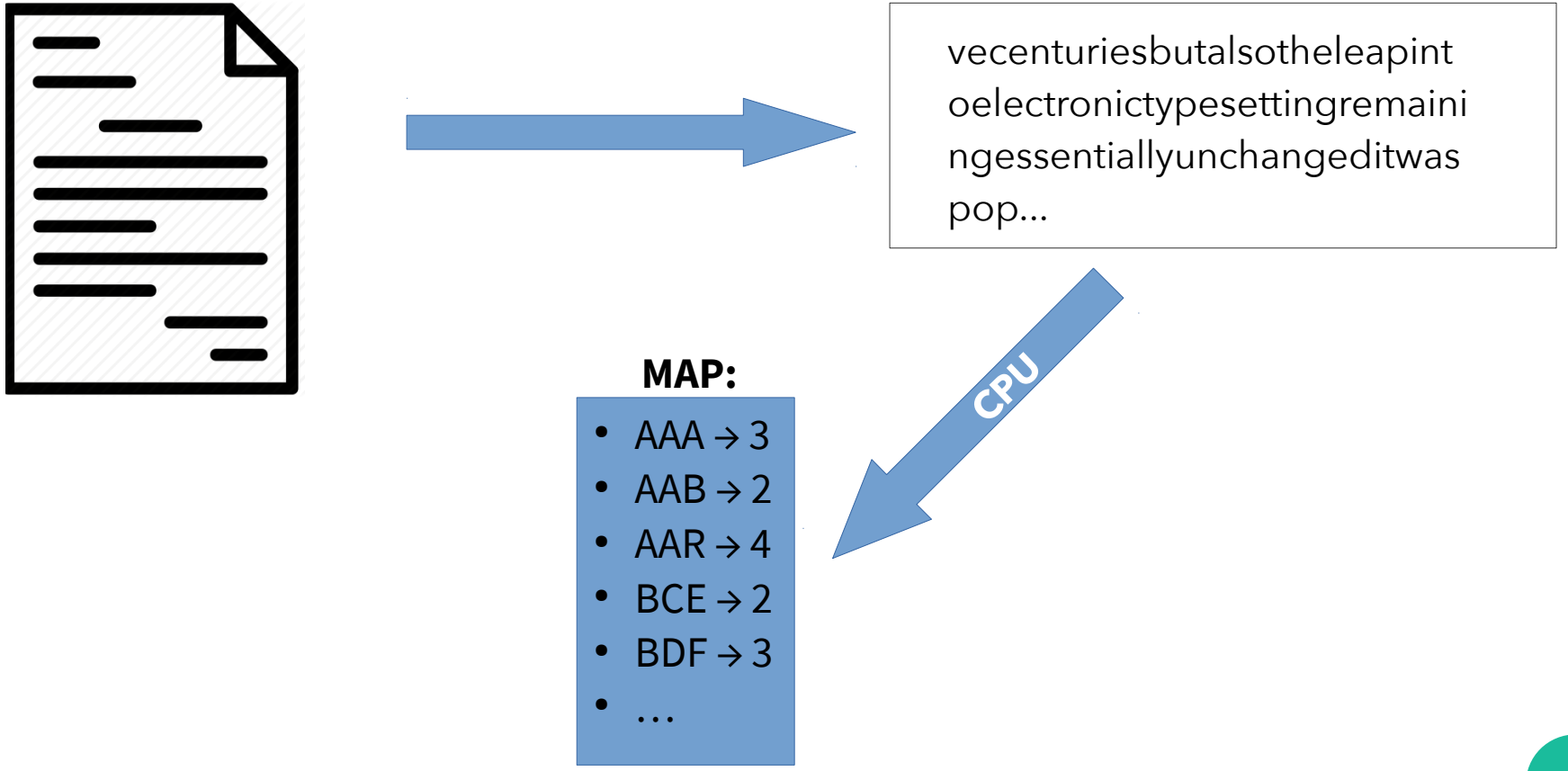
For instance:

"aabb cabb"

Bigram	Occurences	Trigrams	Occurences
aa	1	aab	1
ab	2	abb	2
bb	2	bbc	1
bc	1	bca	1
ca	1	cab	1

CPU Implementation

The version with the CPU scan the input text file, character by character in a sequential manner populating a map of results.



GPU Implementation: CUDA

The used language is CUDA (Compute Unified Device Architecture), a language created by NVIDIA aimed to use the NVIDIA graphic board for parallel programming.

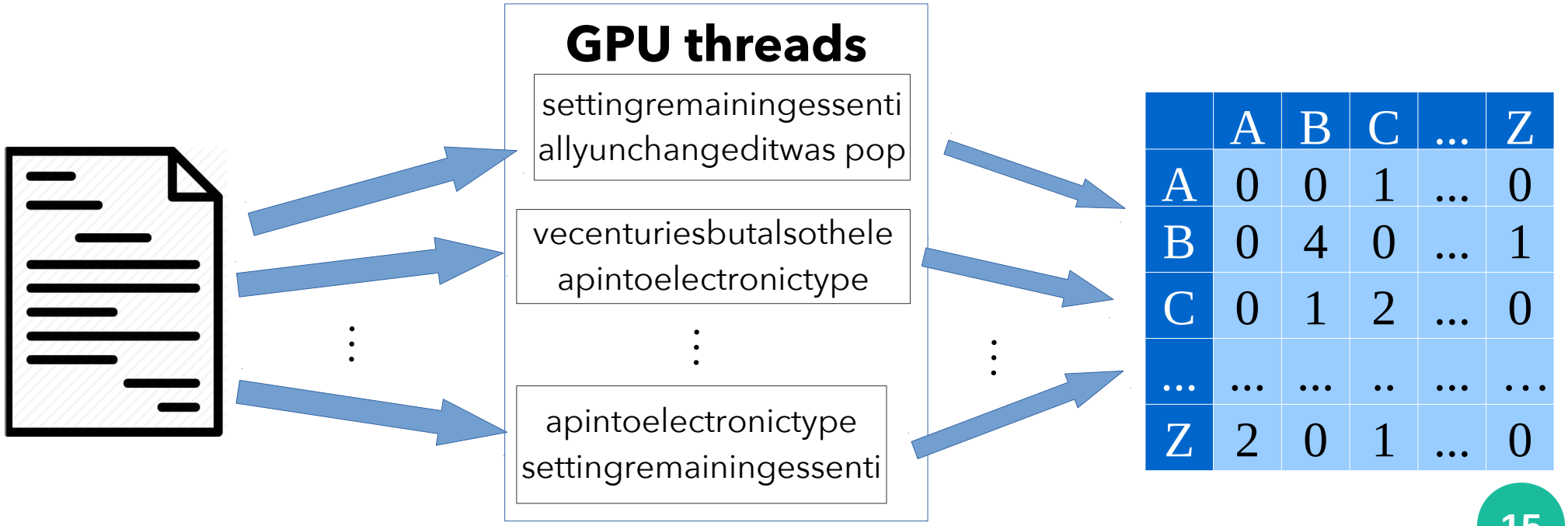
- The GPU is viewed as a compute **device** the is a co-processor of the CPU (host)
- The GPU has its how global memory
- The device can run many threads in parallel organized in **grid** and **blocks** *(each block can contain till 1024 threads and the grid can contain till 65535^3 blocks)*

GPU Implementation

The version with the GPU in CUDA scan the input text file, splitting it in N parts and giving them to N threads where:

$$N = \textit{GRID DIMENSION} \times \textit{BLOCK DIMENSION}.$$

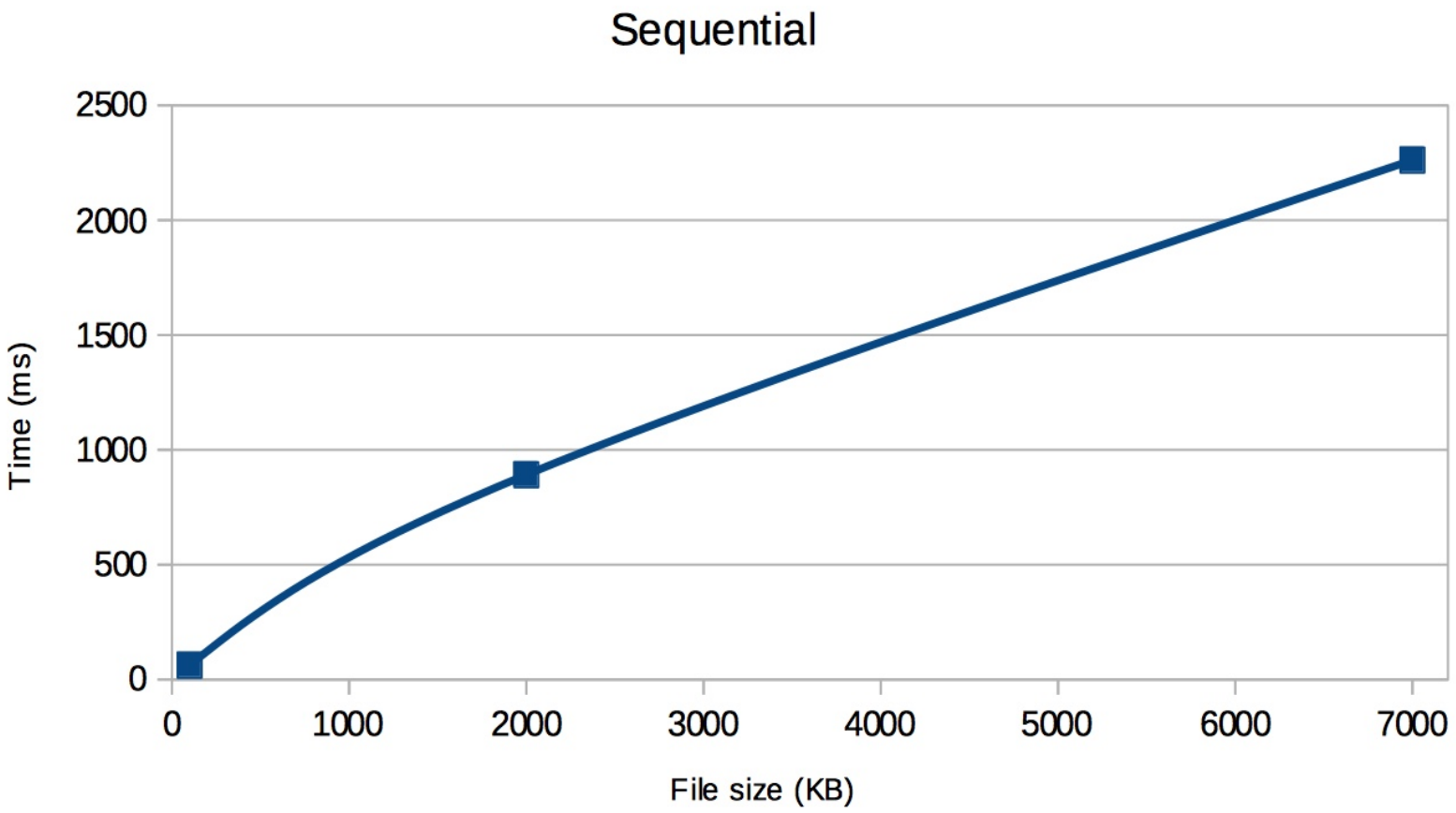
Each thread scan its portion of text and populate a global matrix (or cube, in case of trigram) of occurrences.



	A	B	C	...	Z
A	0	0	1	...	0
B	0	4	0	...	1
C	0	1	2	...	0
...	...	...	..	...	...
Z	2	0	1	...	0

Performance

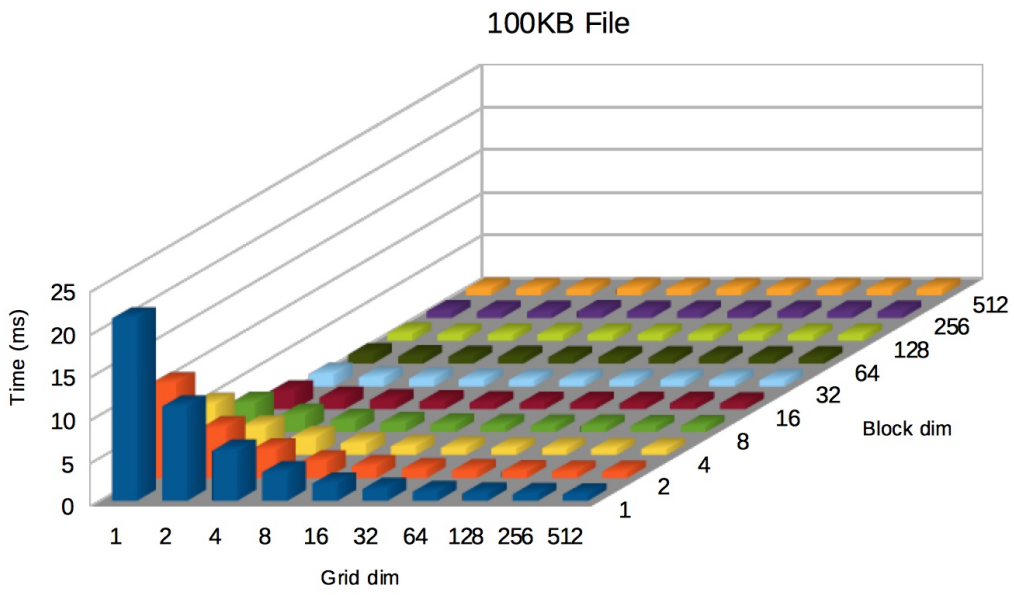
Results of the algorithm varying the size of the text file (**100KB**, **2MB**, **7MB**).

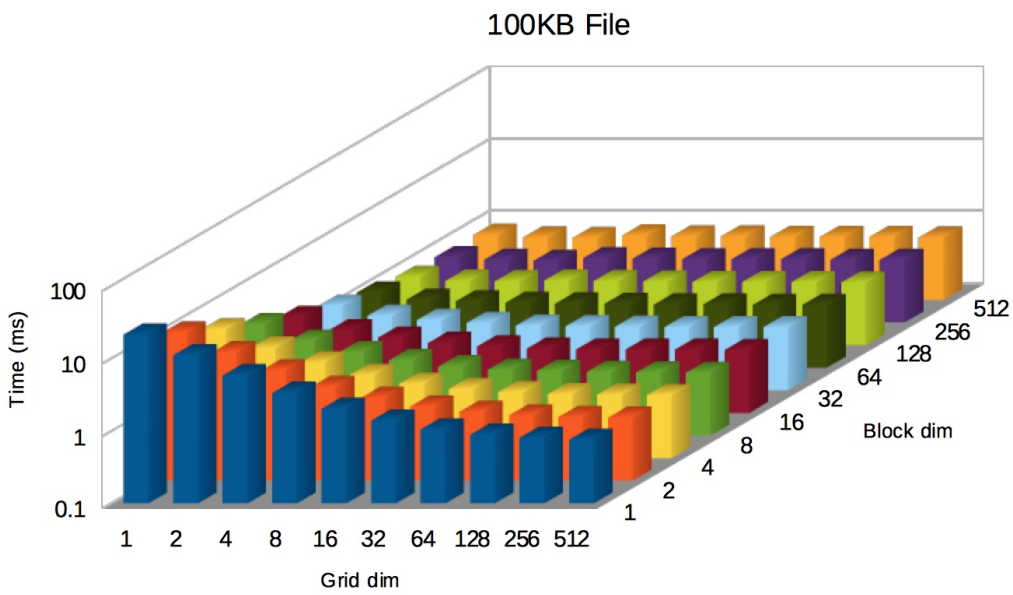


Performance

Results of the algorithm varying the size of the text, grid and blocks.

GPU:



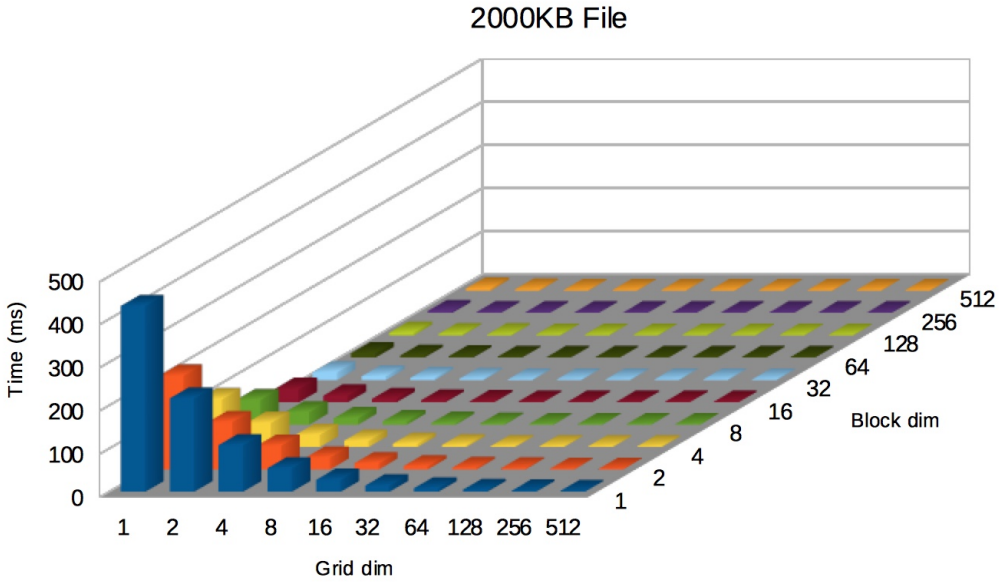


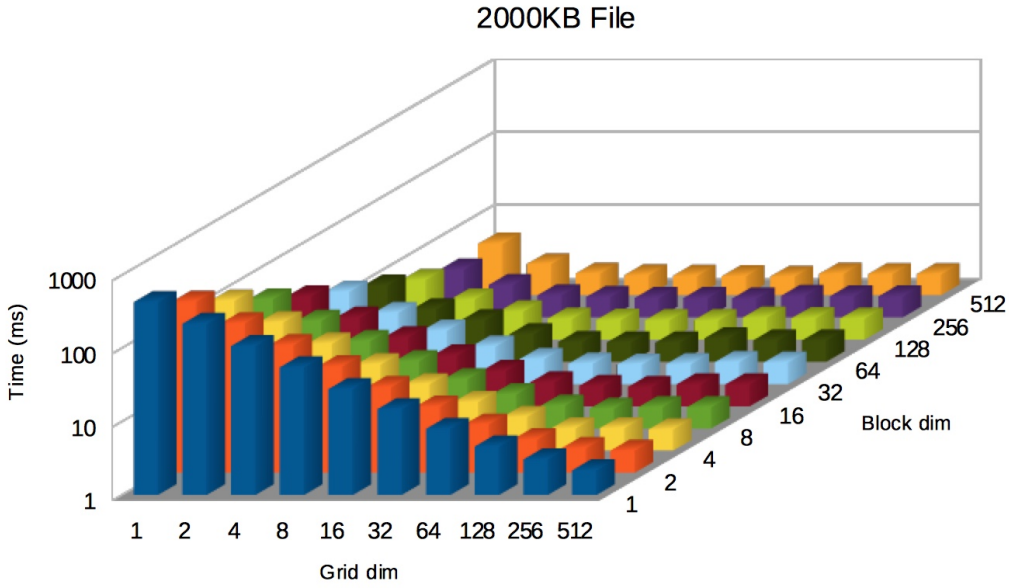
first access: ≈**750**ms

Performance

Results of the algorithm varying the size of the text, grid and blocks.

GPU:



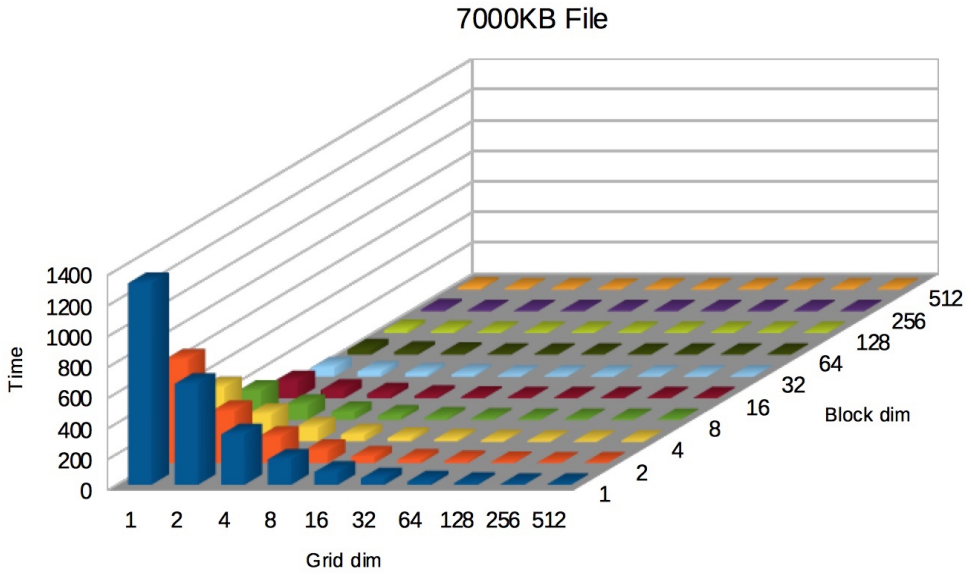


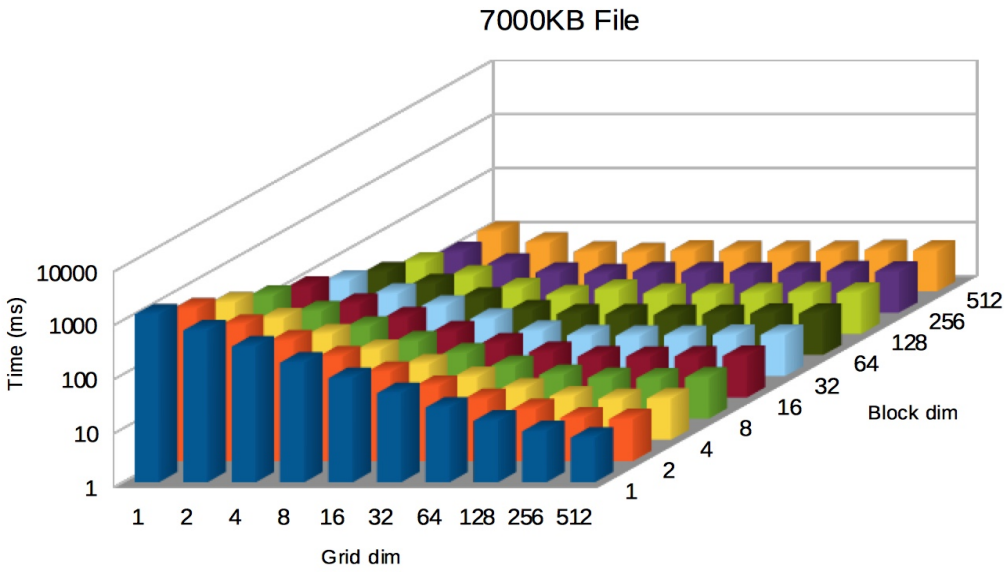
first access: ≈**750**ms

Performance

Results of the algorithm varying the size of the text, grid and blocks.

GPU:

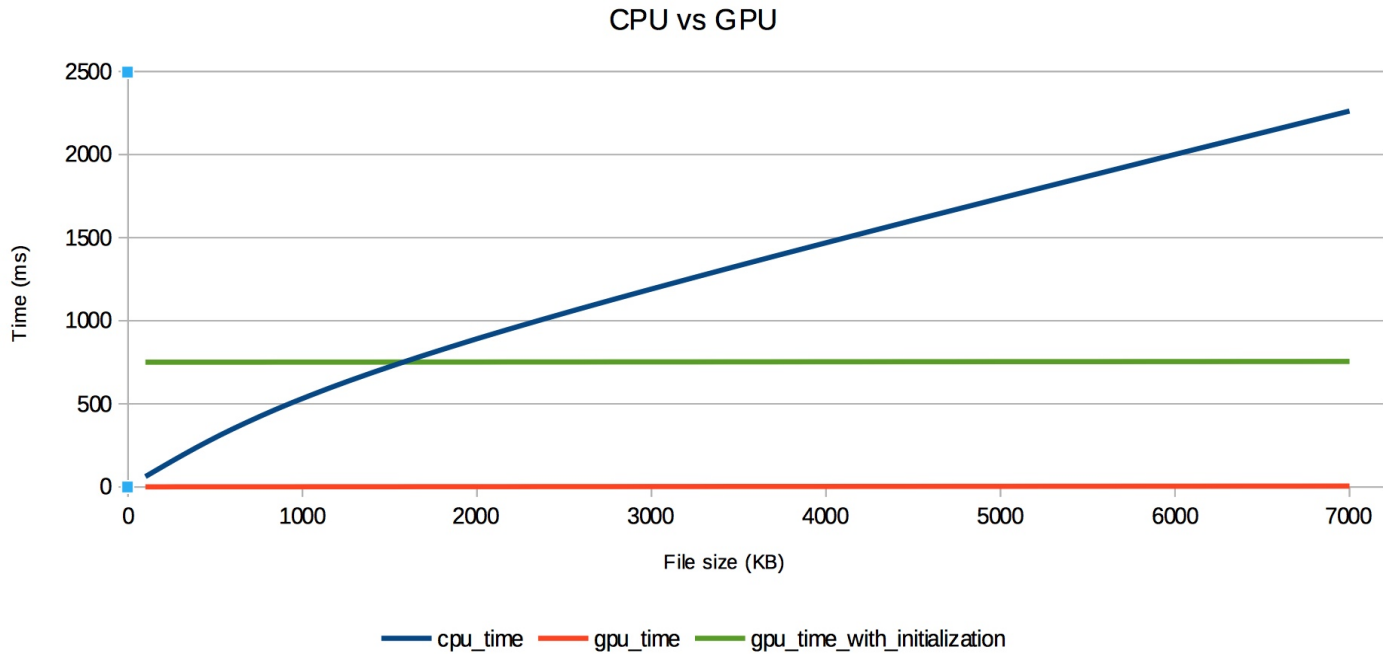




first access: ≈**750**ms

Conclusion

- Ignoring the first access time, GPU is always better then CPU.
- The best parallel result is around block size **512** and grid **4-8** but after a certain value the gain it's almost insignificant (for small file).
- Considering the first access time, it seems that the CPU is better till ~1600KB, after this value GPU is always better.



Thank you!