«Batch merge and merge path sort»

Calcul haute performance: programmation et algorithmique avancées

MAIN 5:
Hala Ben Ali
Paolo Conti

I – MERGE PATH AND SORT

Reference:

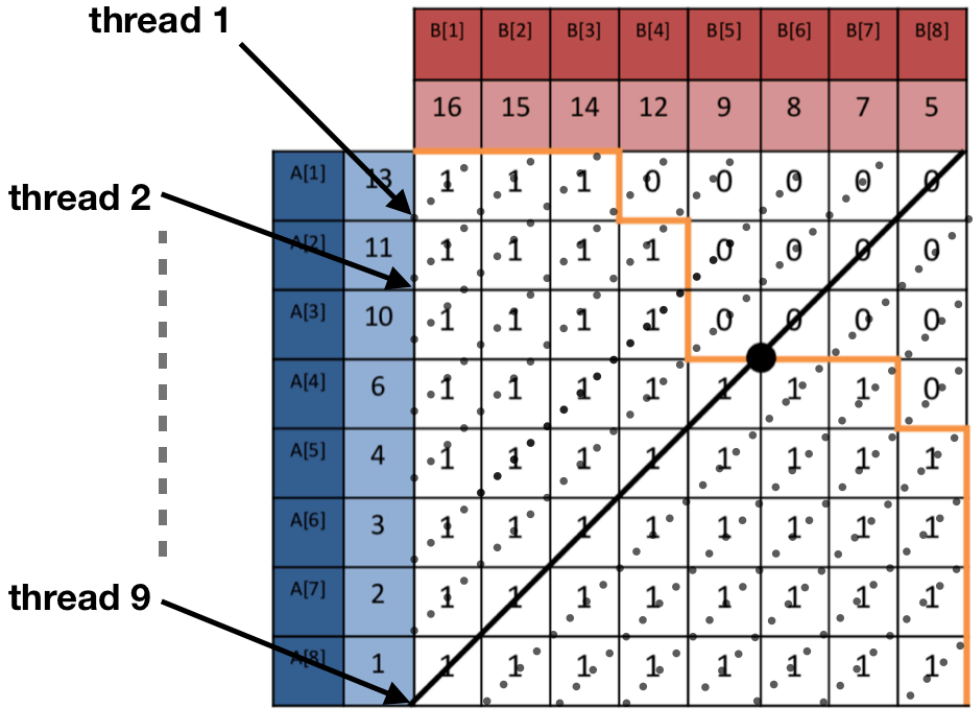
O. Green, R. McColl and D. A. Bader «***GPU Merge Path - A GPU Merging Algorithm***».

26th ACM International Conference on Supercomputing (ICS), San Servolo Islan, Venice, Italy, June 25-29, 2012

summary:

- Sorting an array on GPU
- Batch merging various small arrays
- Proposing an application

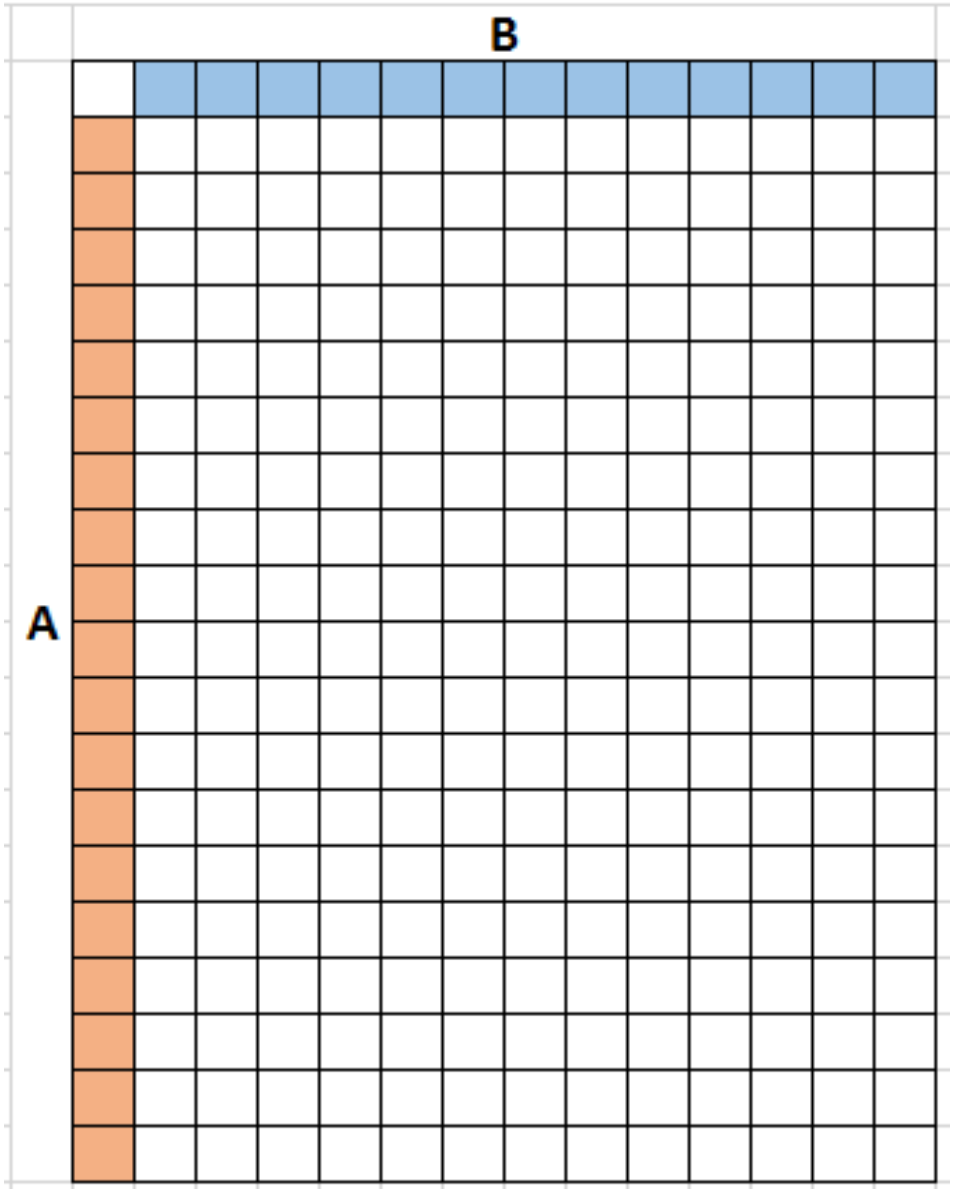
Only one block:



STEPS:

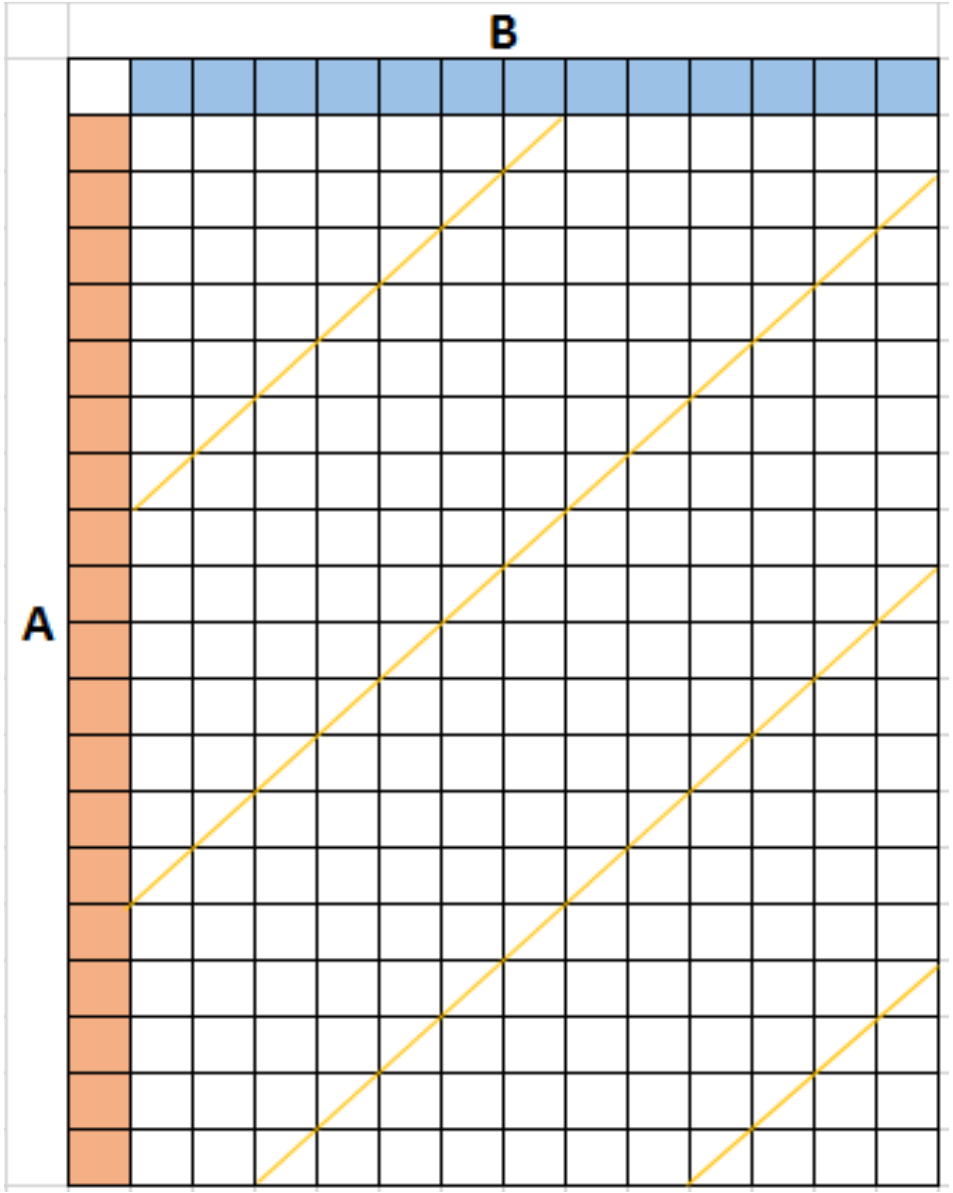
- PARTITIONING
- FINDING THE MERGE PATH
- MERGING

PARTITIONING



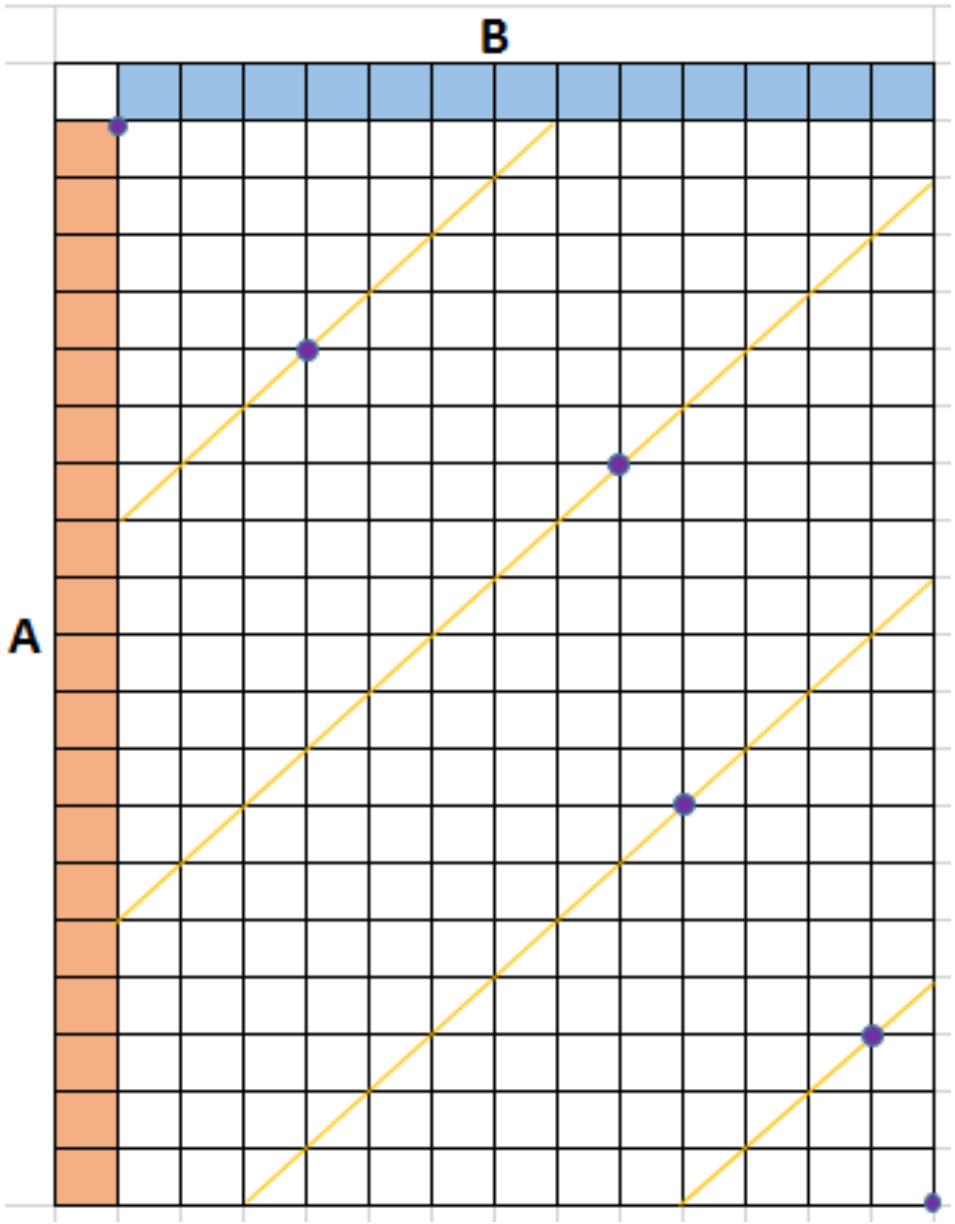
PARTITIONING

Divide the work among the blocks



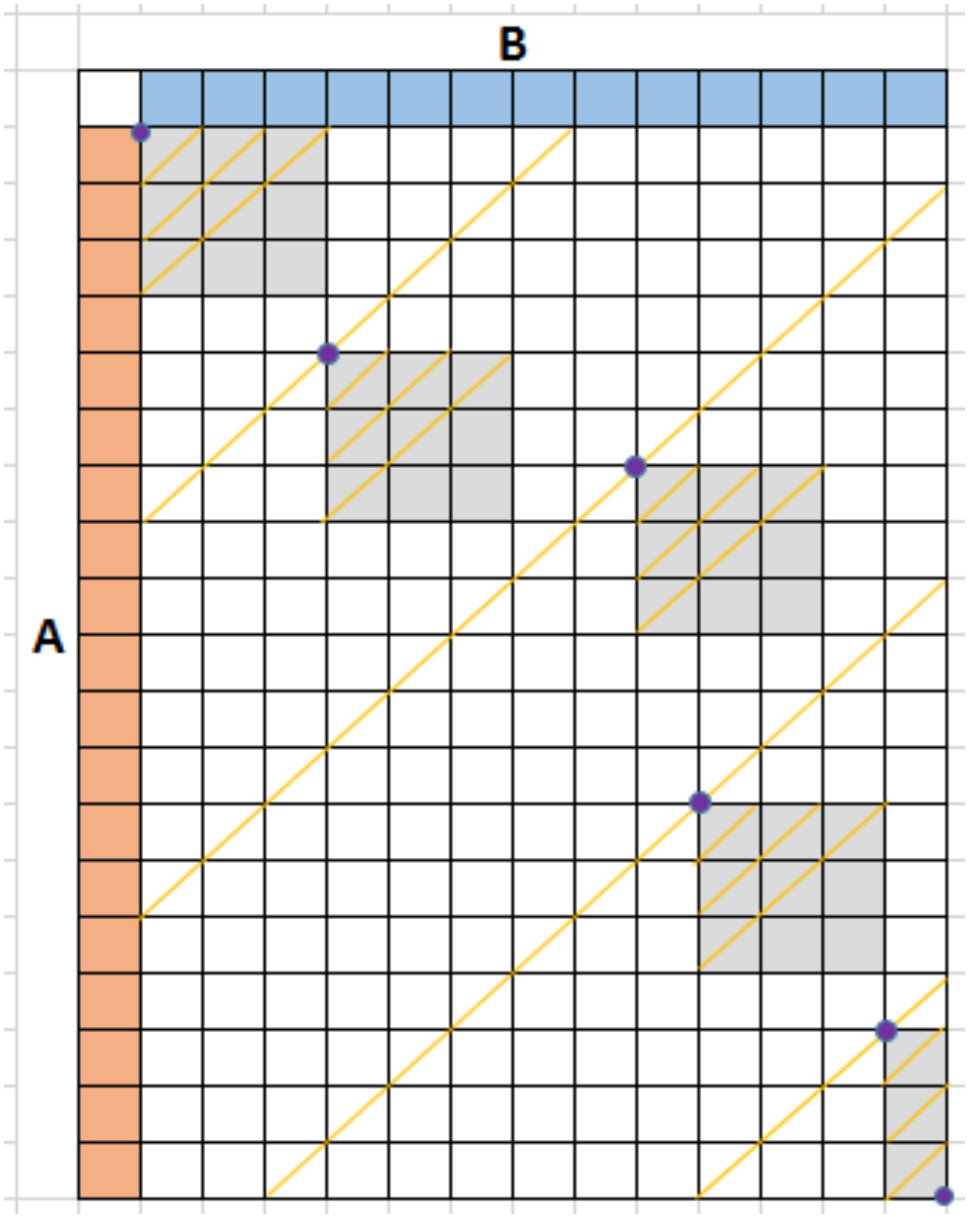
PARTITIONING

Binary cross diagonal search for Merge Path on the partition points on
global memory



FIND MERGE PATH

In parallel and relying on ***shared memory***



Example :
3 blocks
3 threads

- **X_gpu**

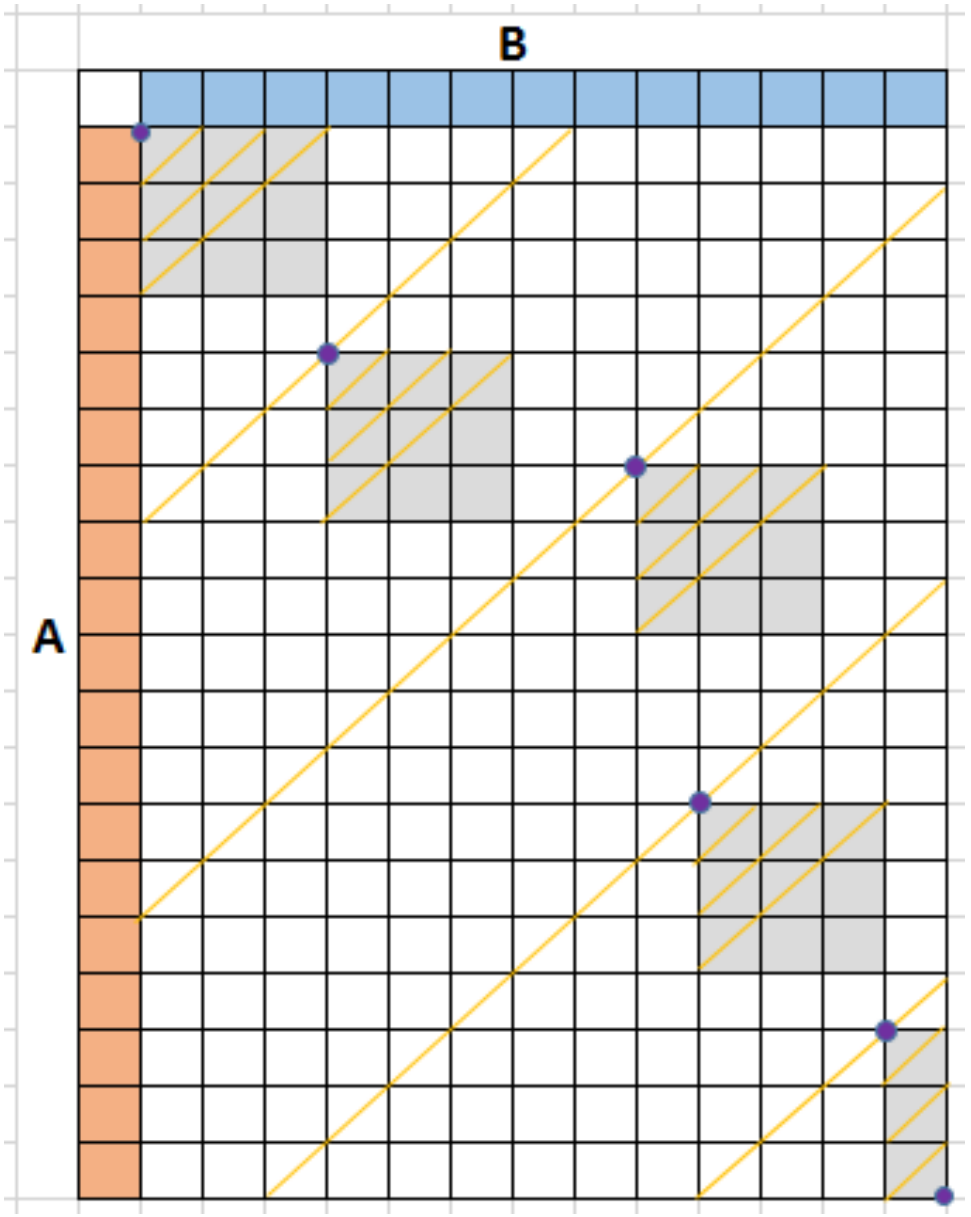
Qx block0			Qx block 1			Qx block 2		

- **Y_gpu**

Qy block0			Qy block 1			Qy block 2		

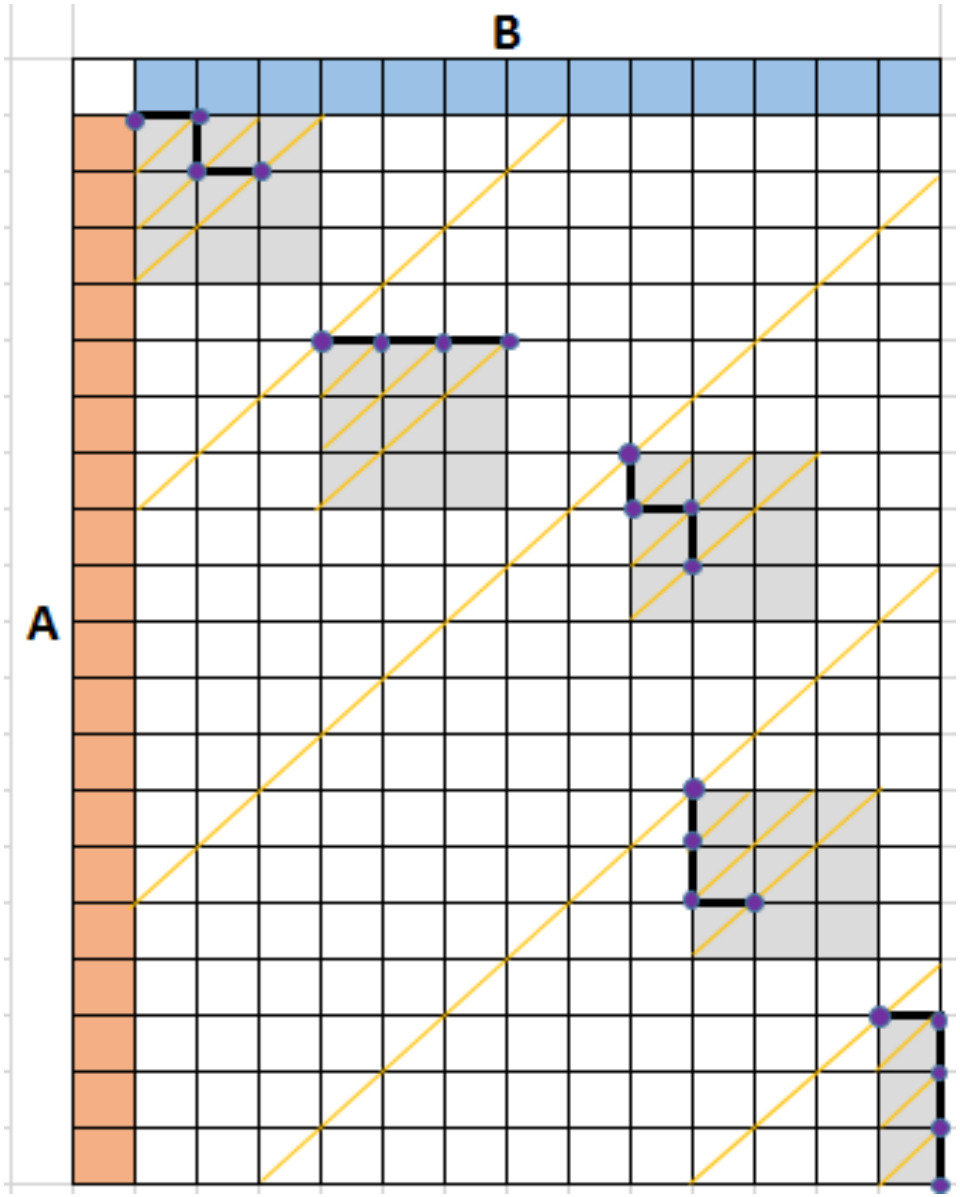
FIND MERGE PATH

In parallel and relying on ***shared memory***



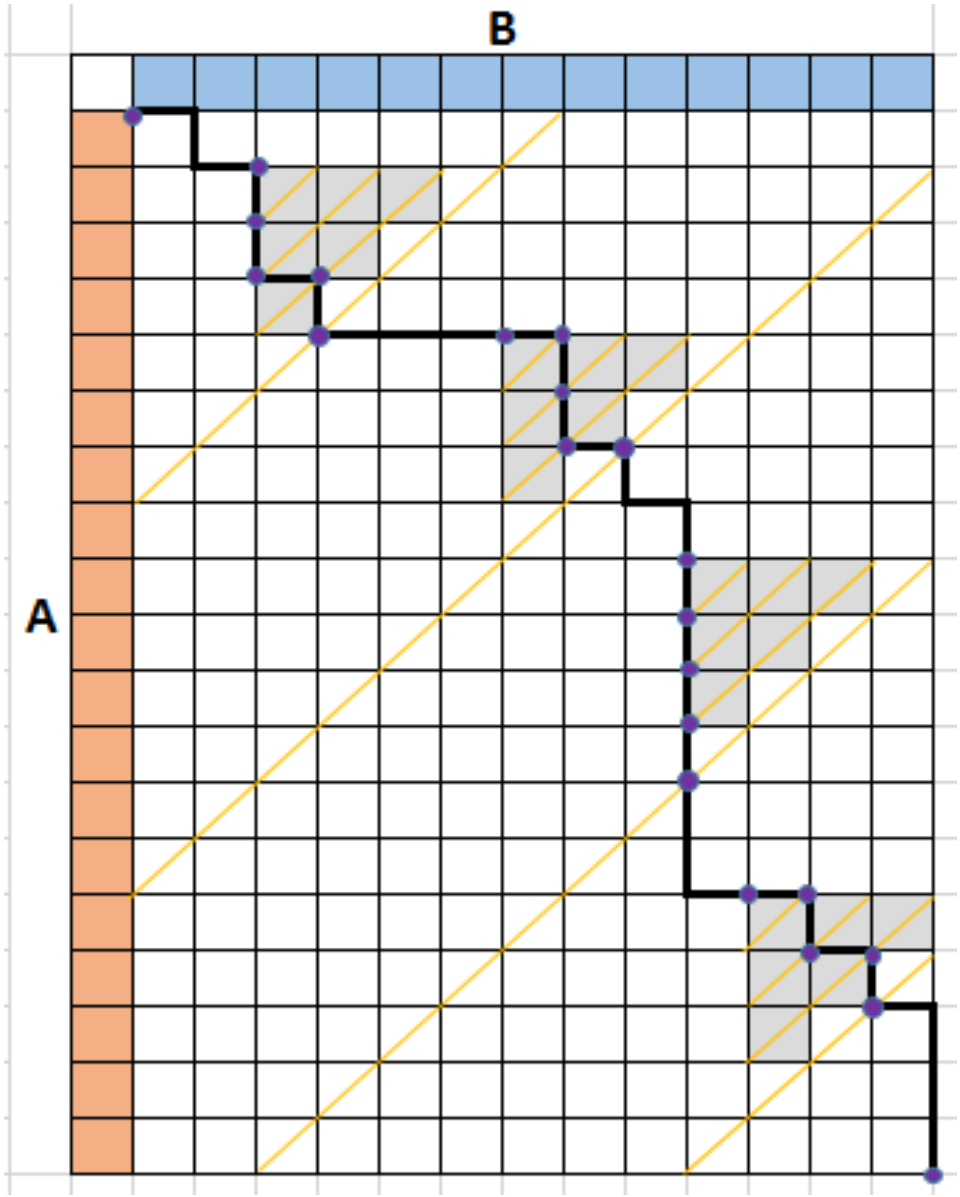
FIND MERGE PATH

Write the path on the ***global memory***



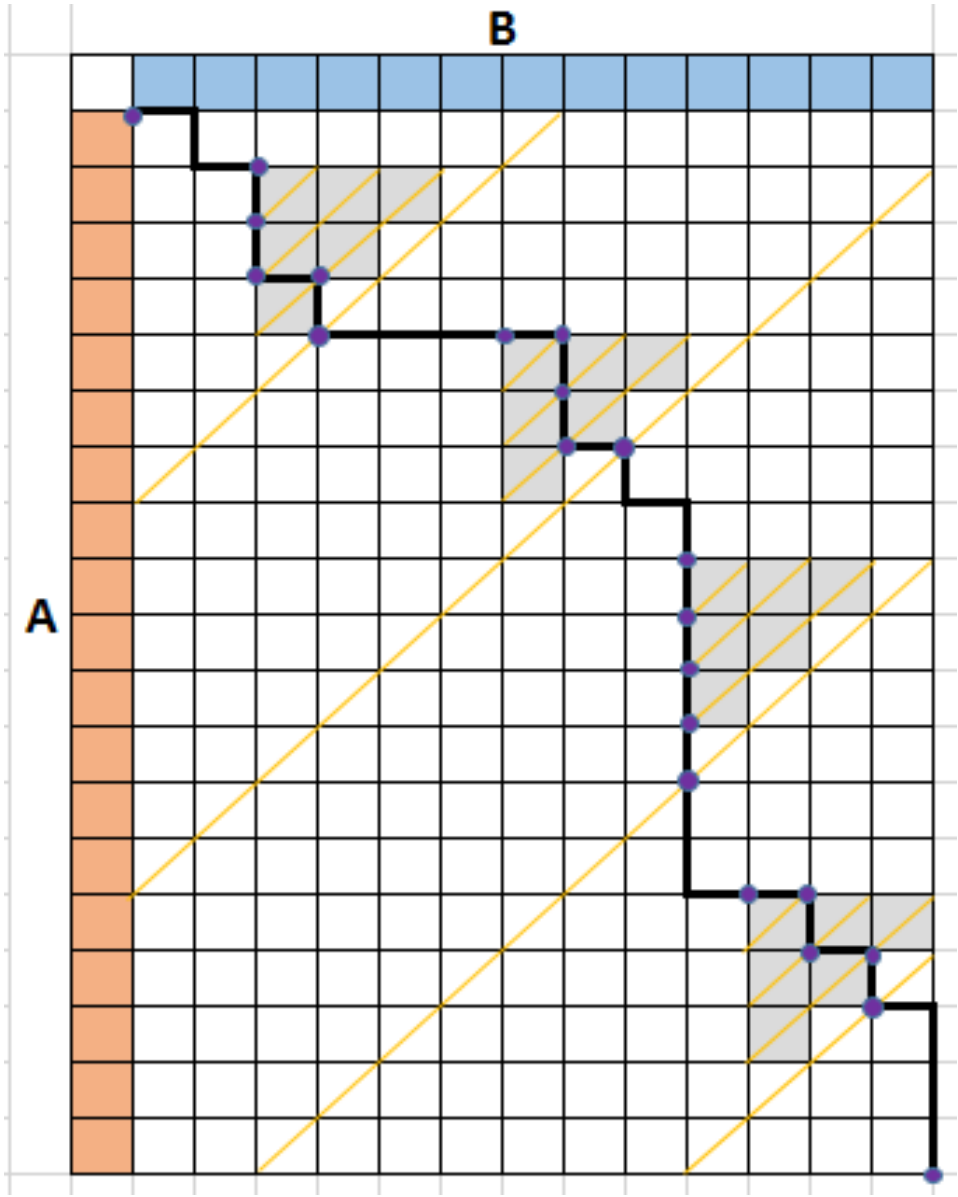
FIND MERGE PATH

Cycles in parallel

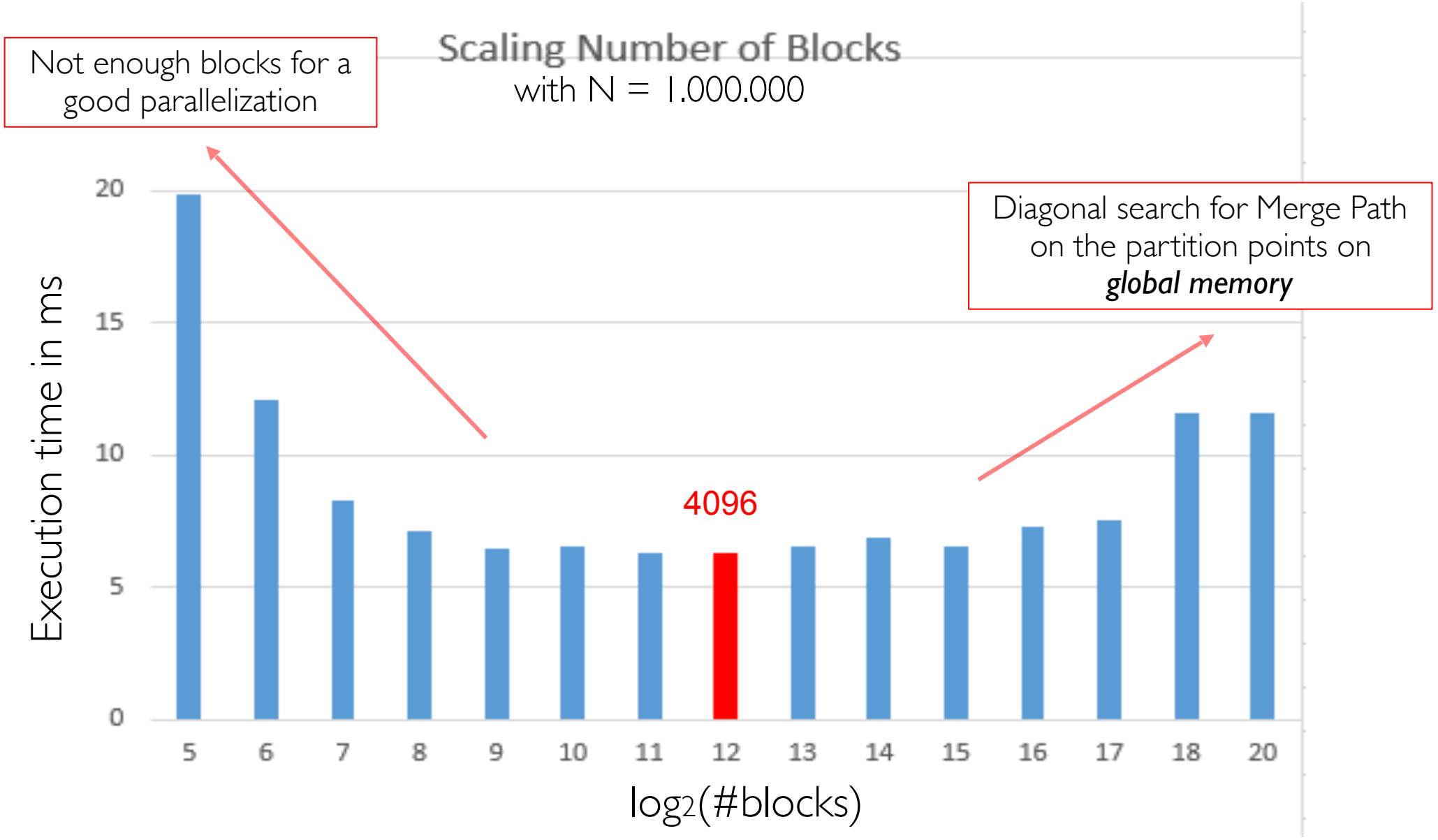


MERGE

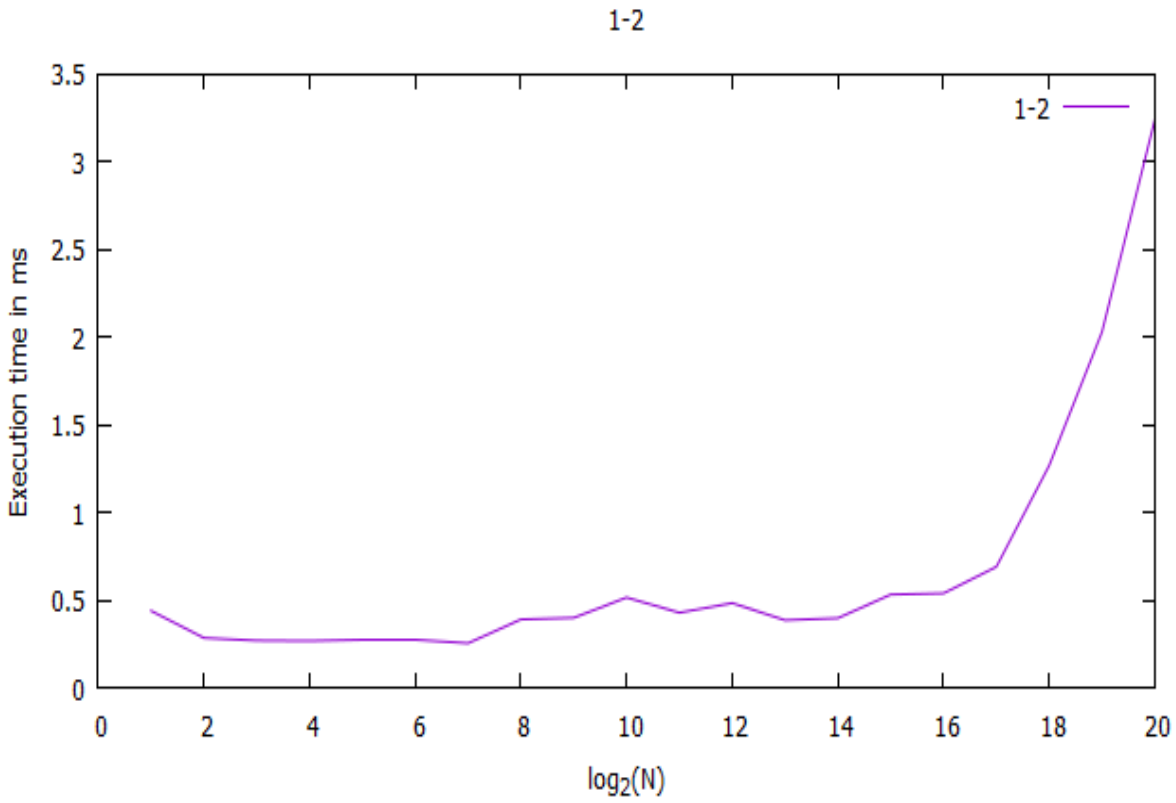
Placing the Merge Path on the
shared memory



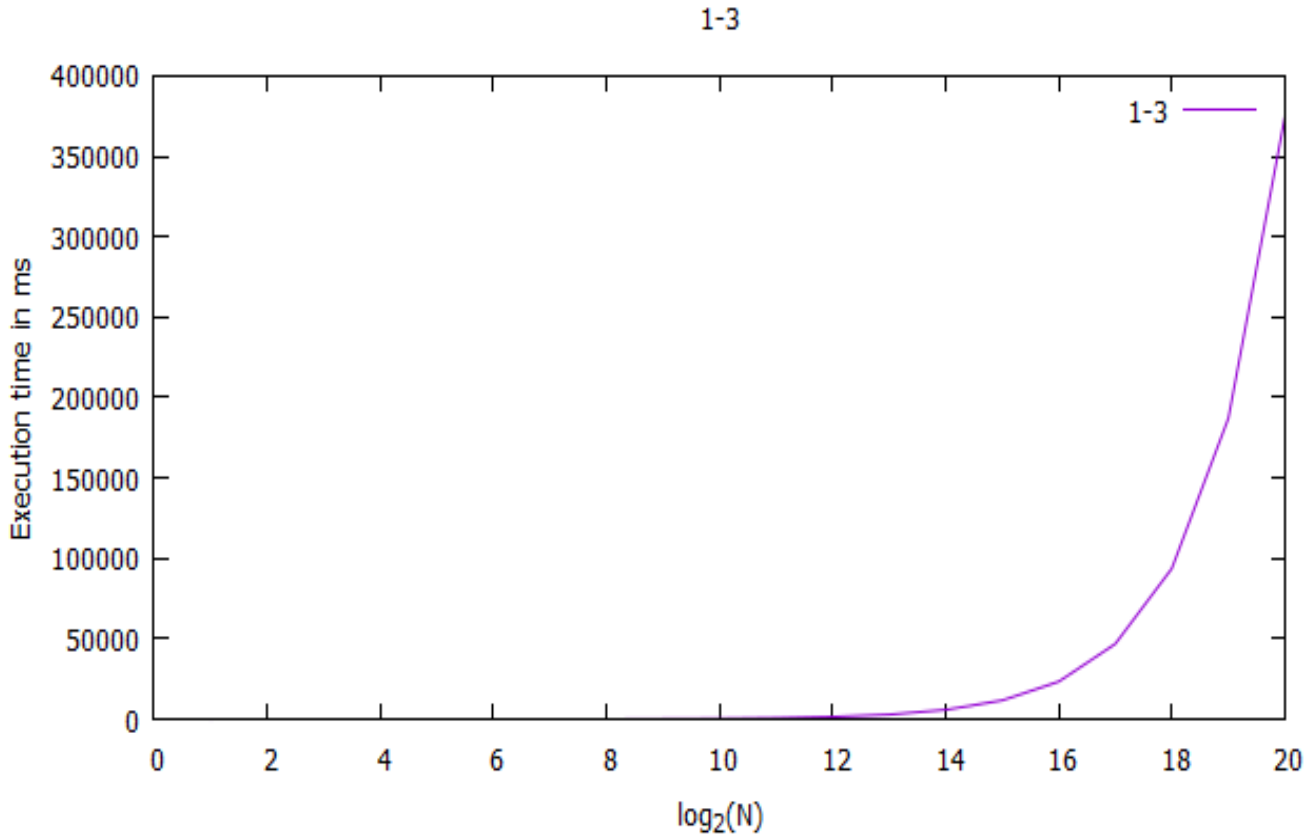
MERGE


Not enough blocks for a good parallelization
Scaling Number of Blocks
with N = 1.000.000
Diagonal search for Merge Path on the partition points on *global memory*
Execution time in ms
20
15
10
5
0
4096
5
6
7
8
9
10
11
12
13
14
15
16
17
18
20
log2(#blocks)

EXECUTION TIMES



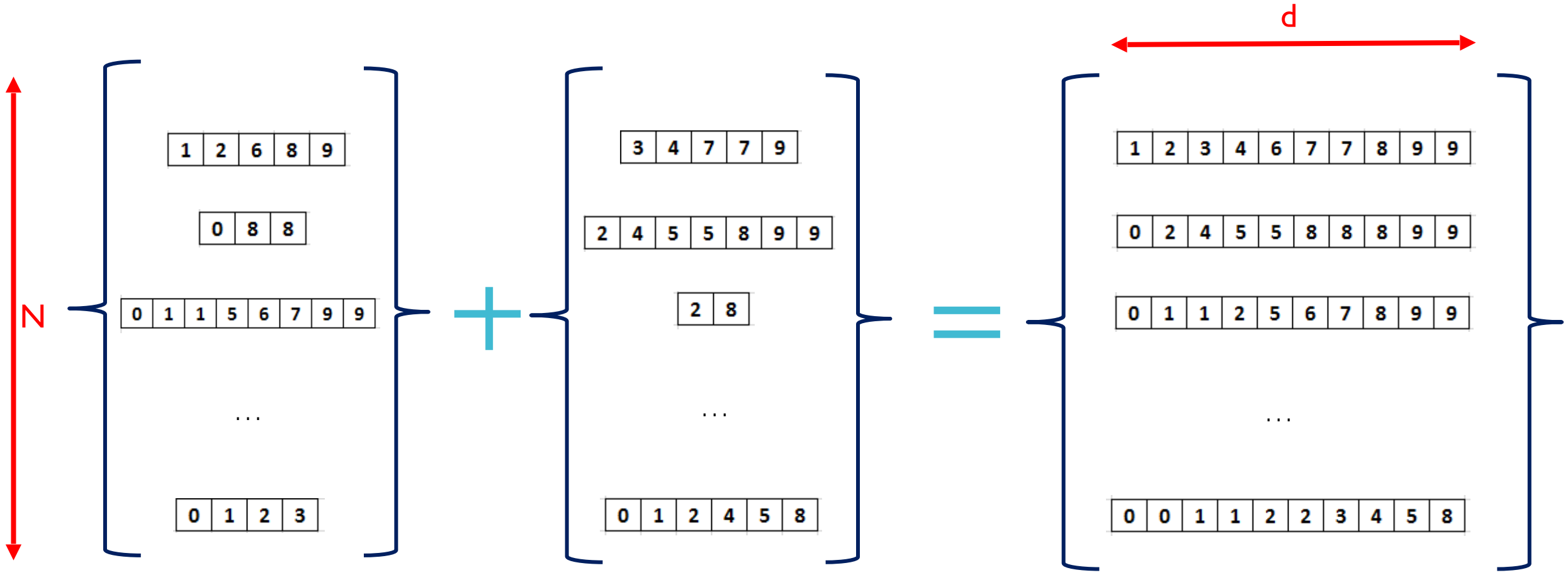
Our programme



Normal Merge Sort

2- BATCH MERGE

BATCH MERGE


d
N
1 2 6 8 9
0 8 8
0 1 1 5 6 7 9 9
...
0 1 2 3
3 4 7 7 9
2 4 5 5 8 9 9
2 8
...
0 1 2 4 5 8
1 2 3 4 6 7 7 8 9 9
0 2 4 5 5 8 8 8 9 9
0 1 1 2 5 6 7 8 9 9
...
0 0 1 1 2 2 3 4 5 8

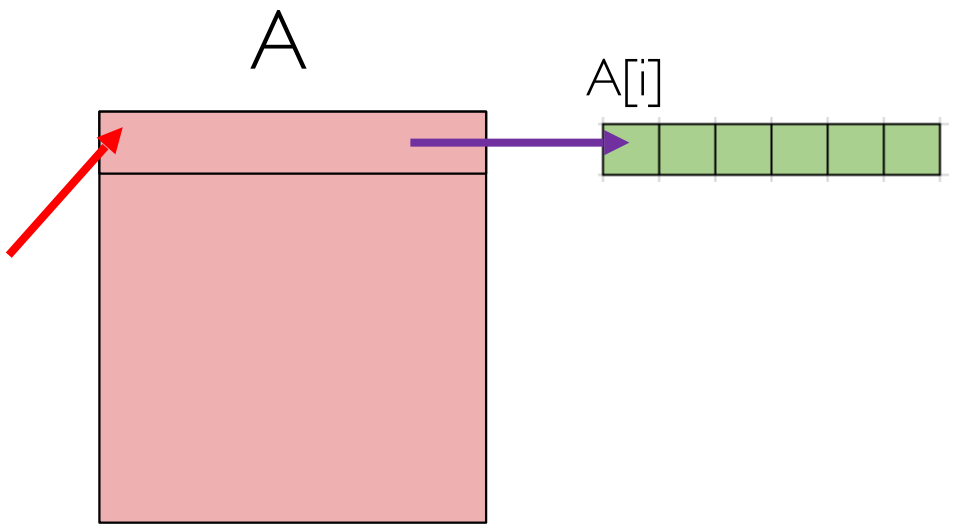
INDEXES

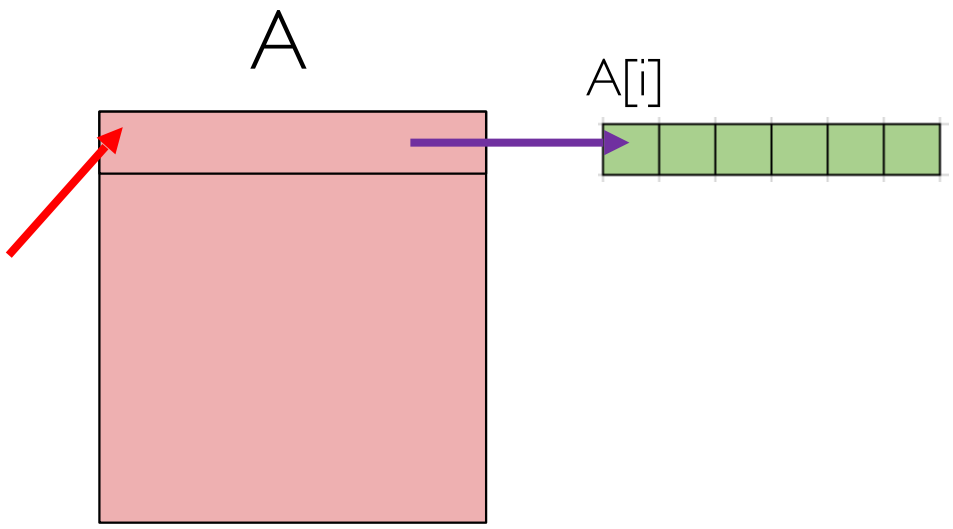
d: number of elements in a pair of array

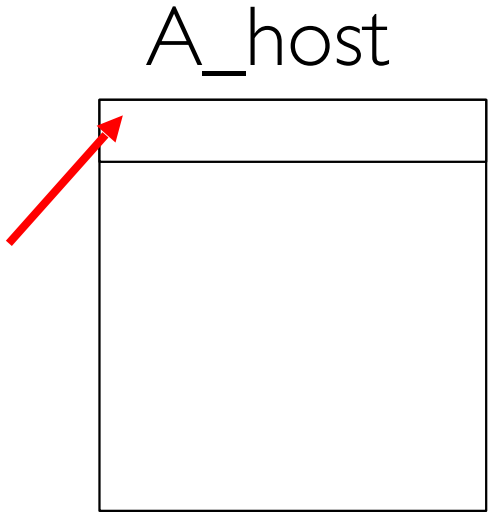
m: number of pairs of array mergesorted in a single block

```
int tidx = threadIdx.x % d;

int Qt = (threadIdx.x – tidx) / d;

int gbx = Qt + blockIdx.x * (blockDim.x / d);
```

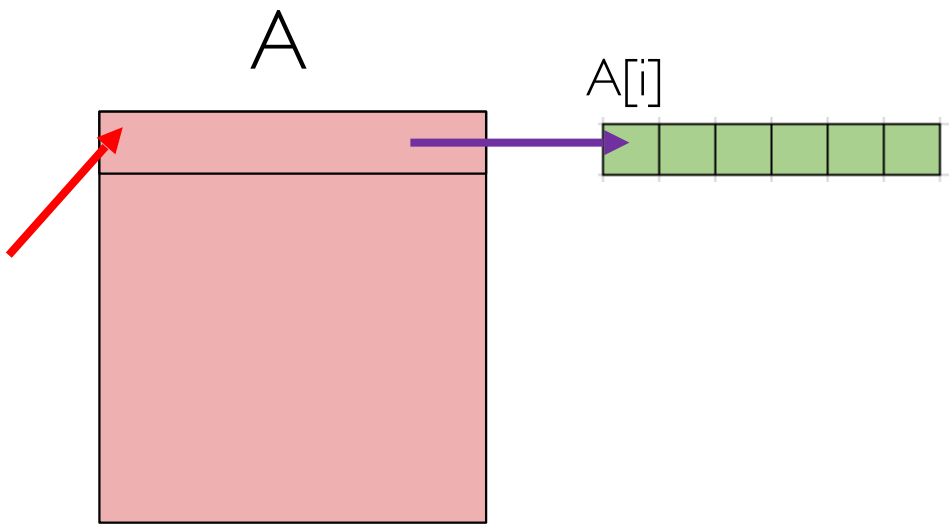
- **Enumeration** from 0 to d-1 of the **elements** of **each of the m pairs** of arrays which are going to be sorted in a single block
- **Local enumeration** from 0 to m-1 of the pairs of arrays in a single block
- **Global enumeration** from 0 to N-1 of the pairs of arrays in all the blocks

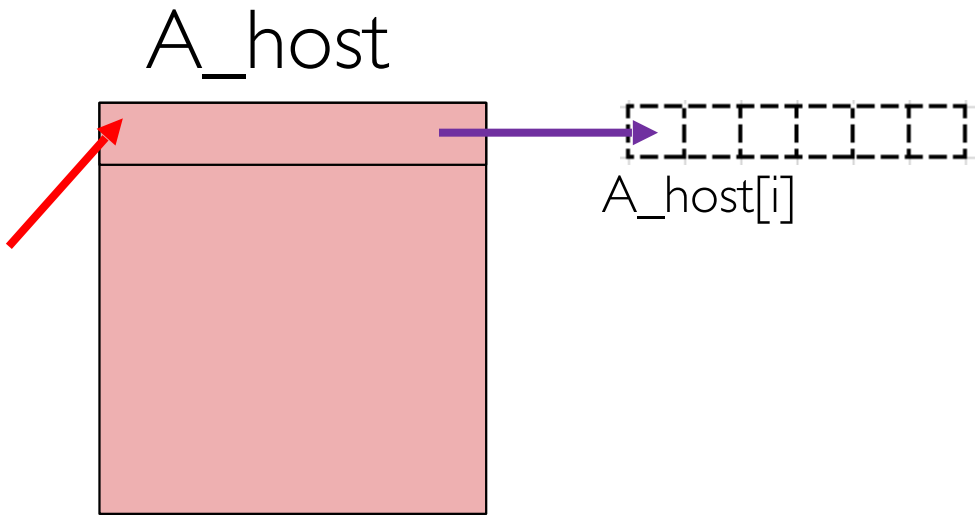

A
A[i]



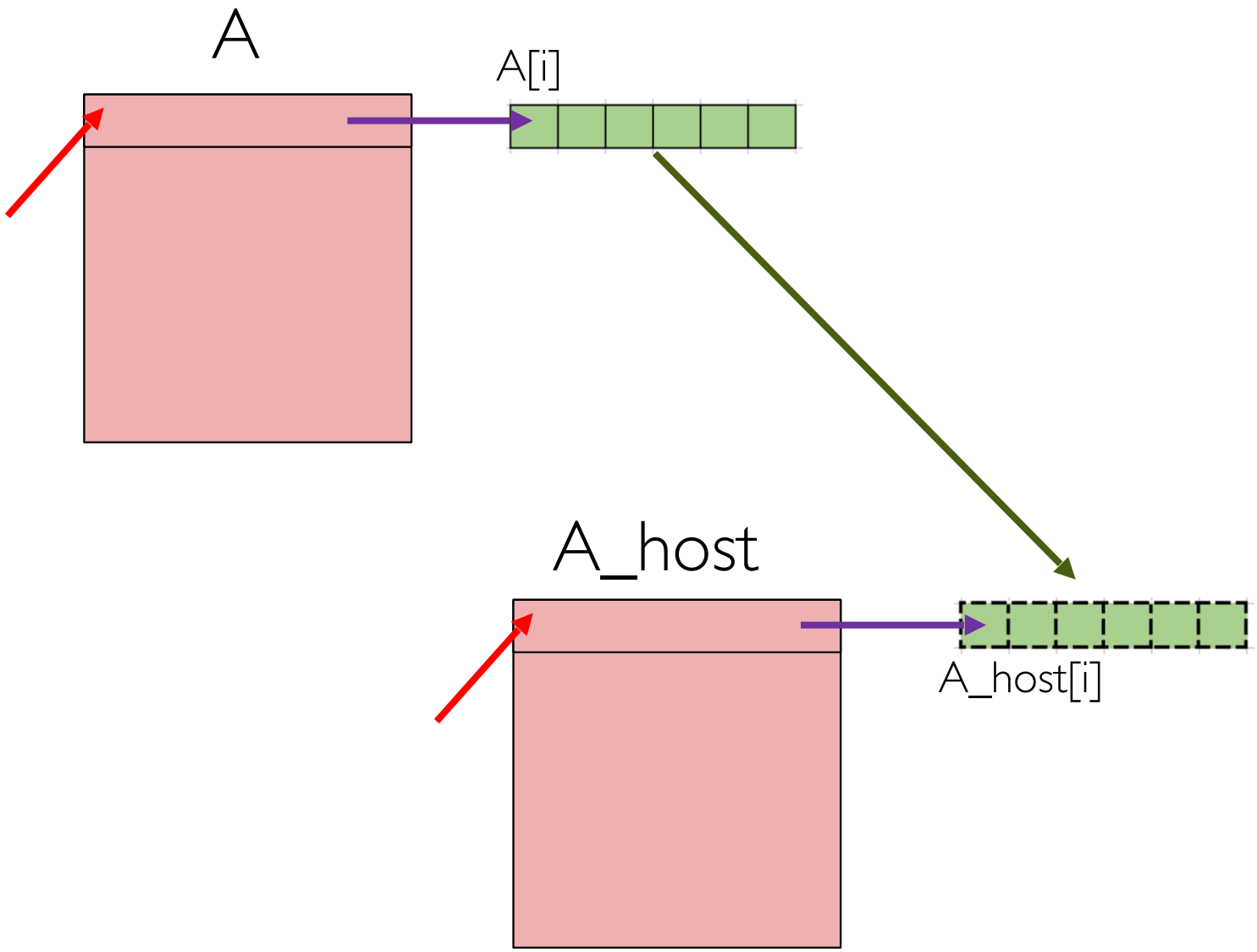


MALLOC

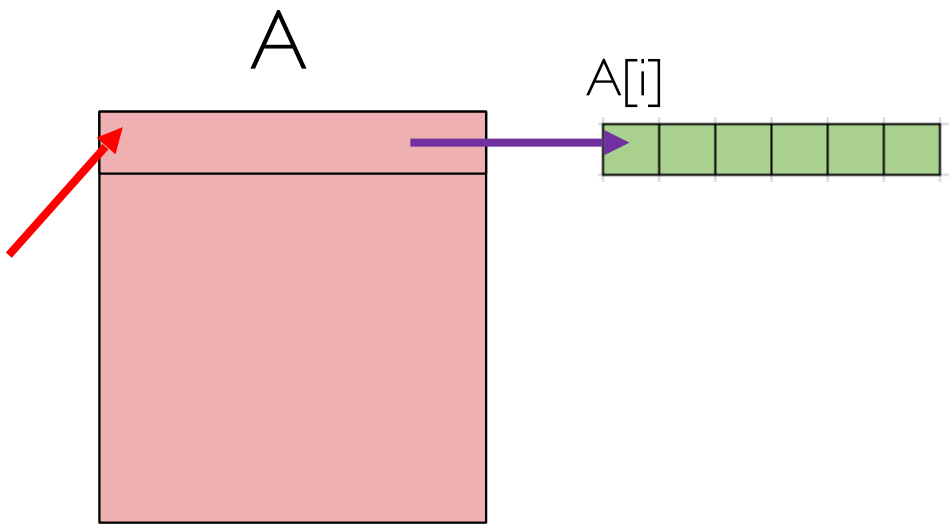


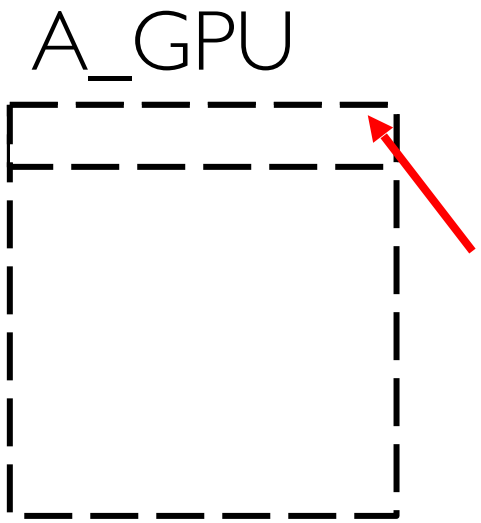


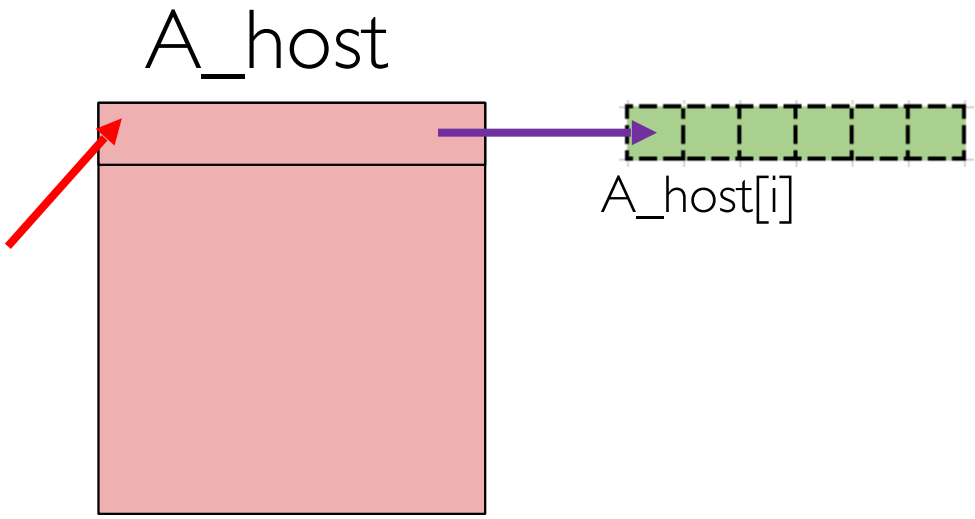
CUDA MALLOC



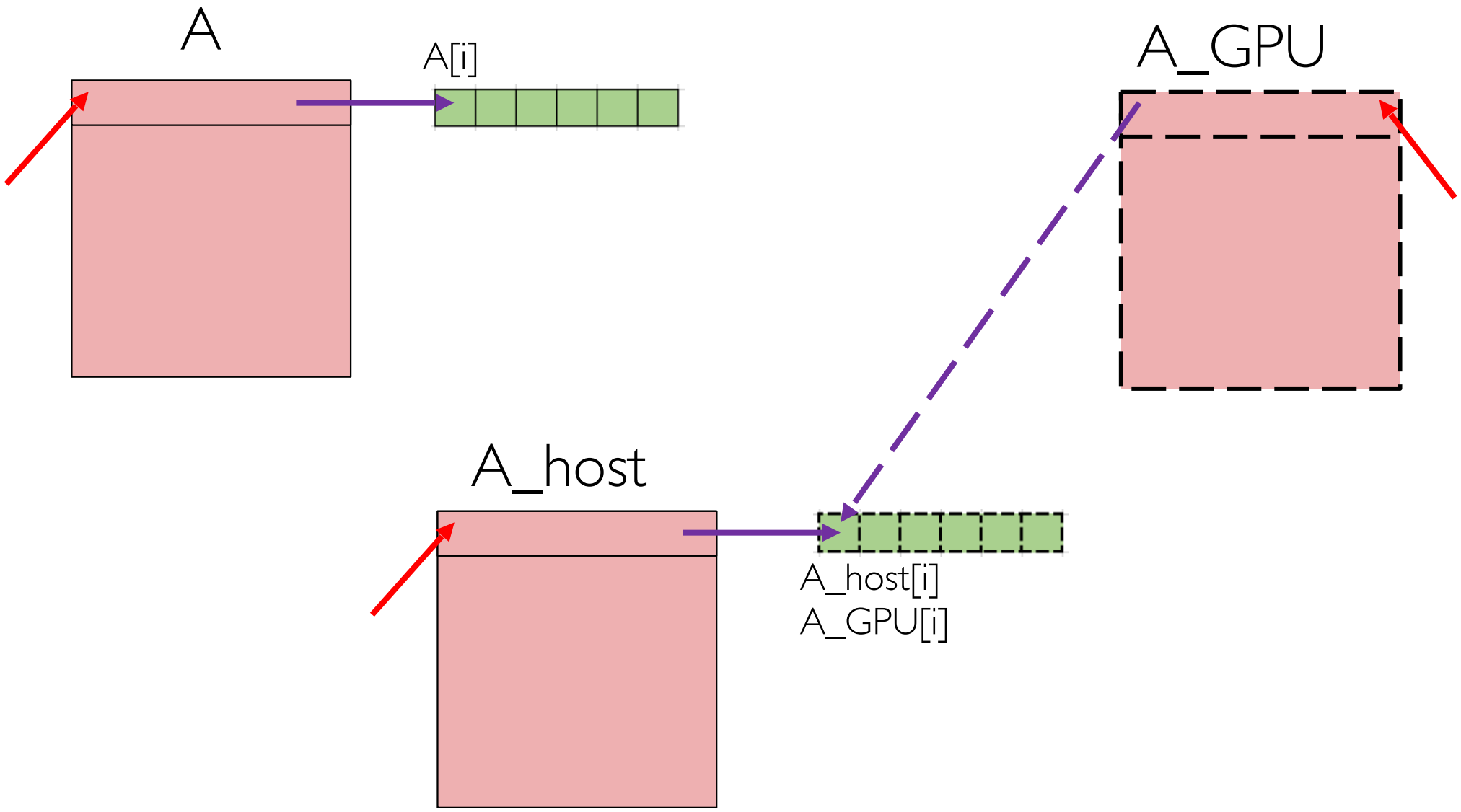
CUDA MEMCPY





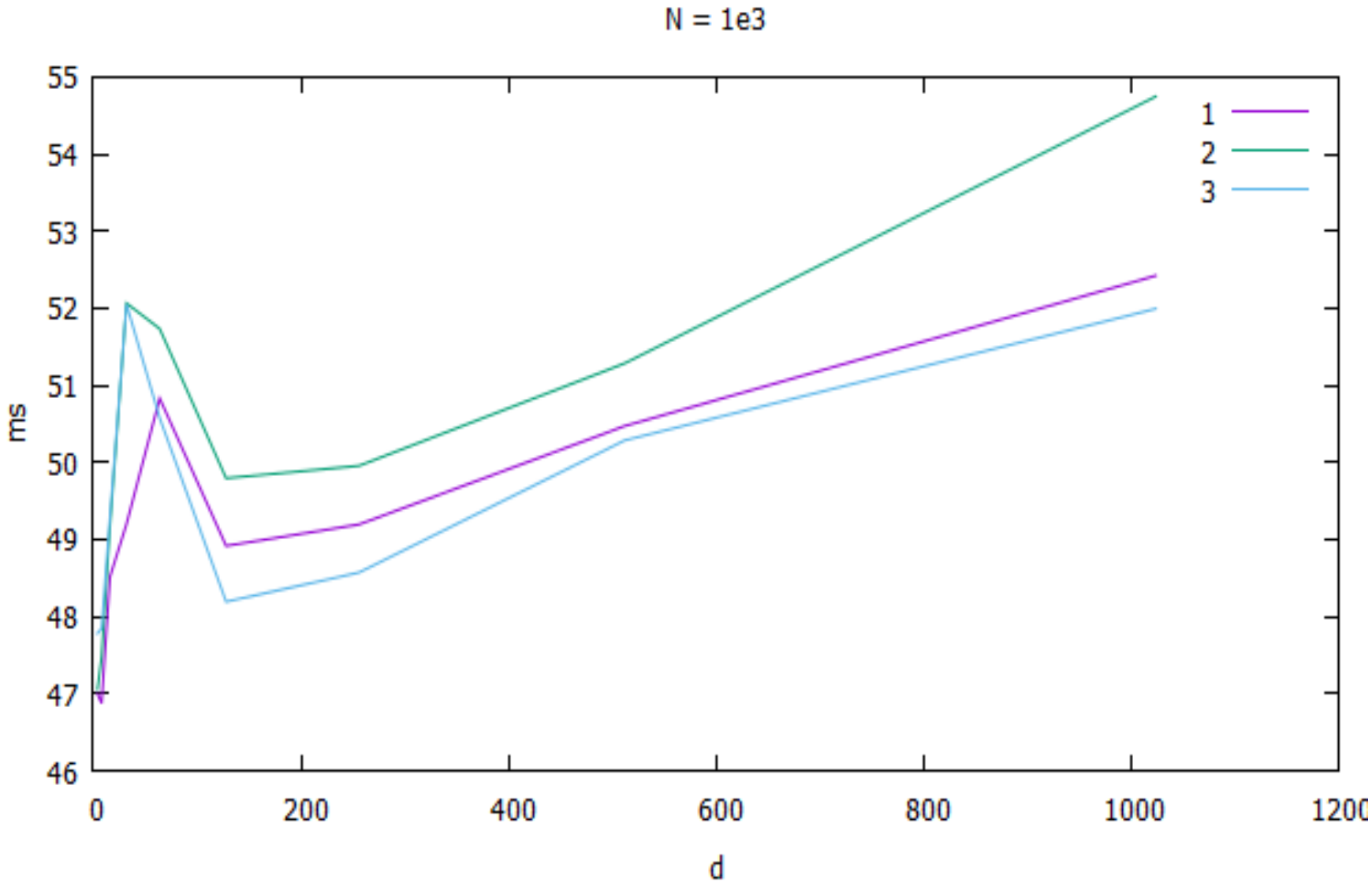


CUDA MALLOC

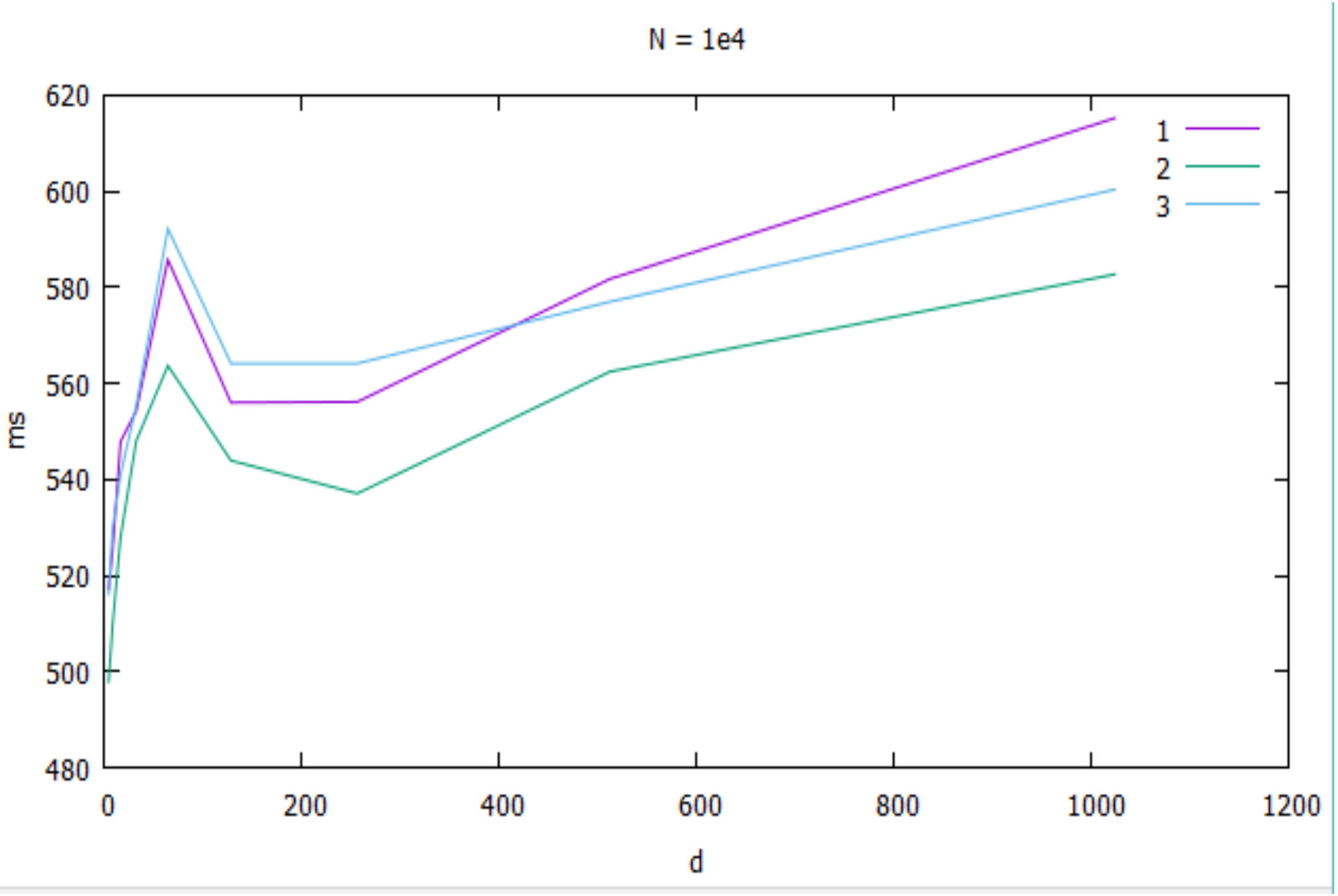


CUDA MEMCPY

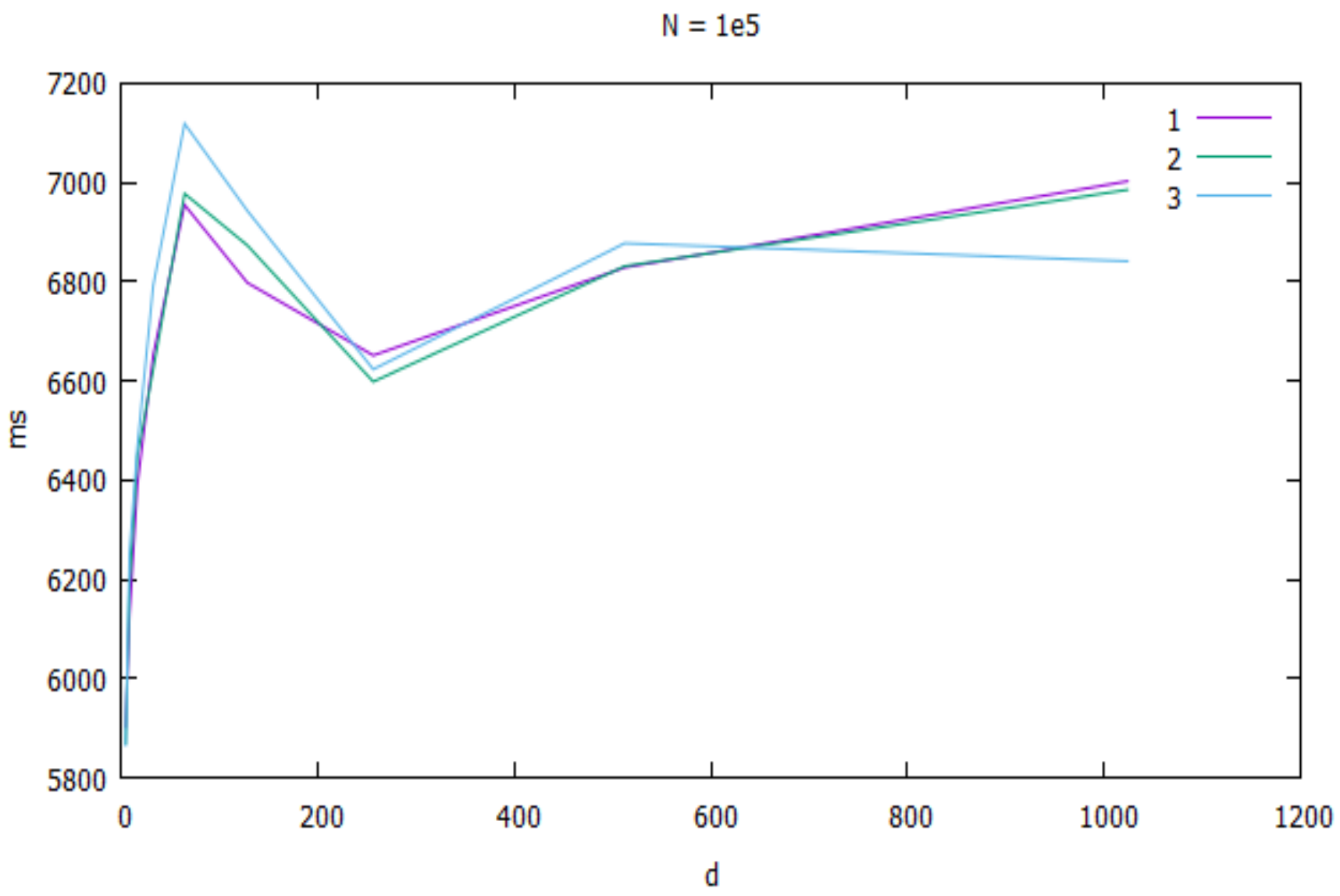
BATCH MERGE
N = 1000



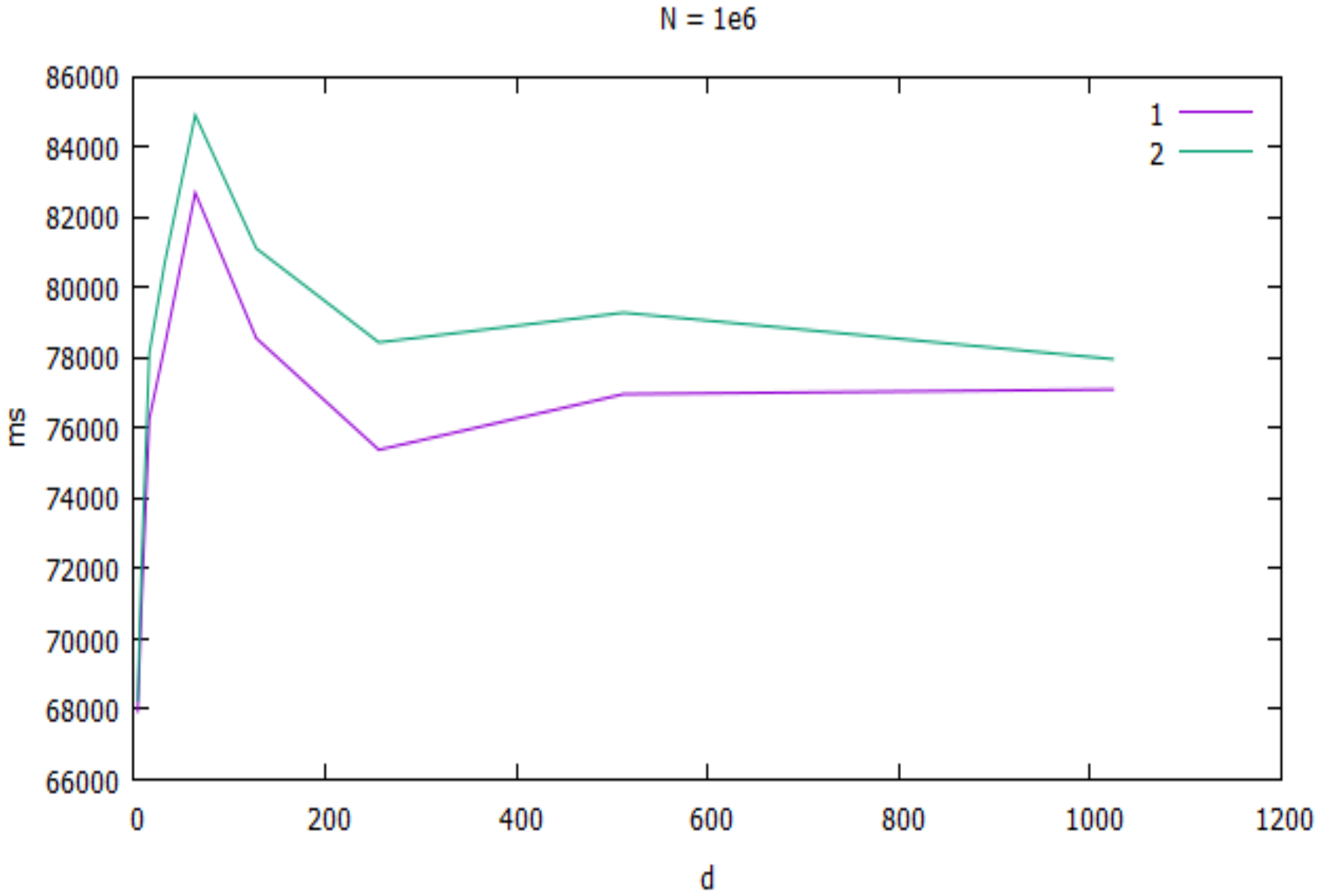
BATCH MERGE
N = 10.000



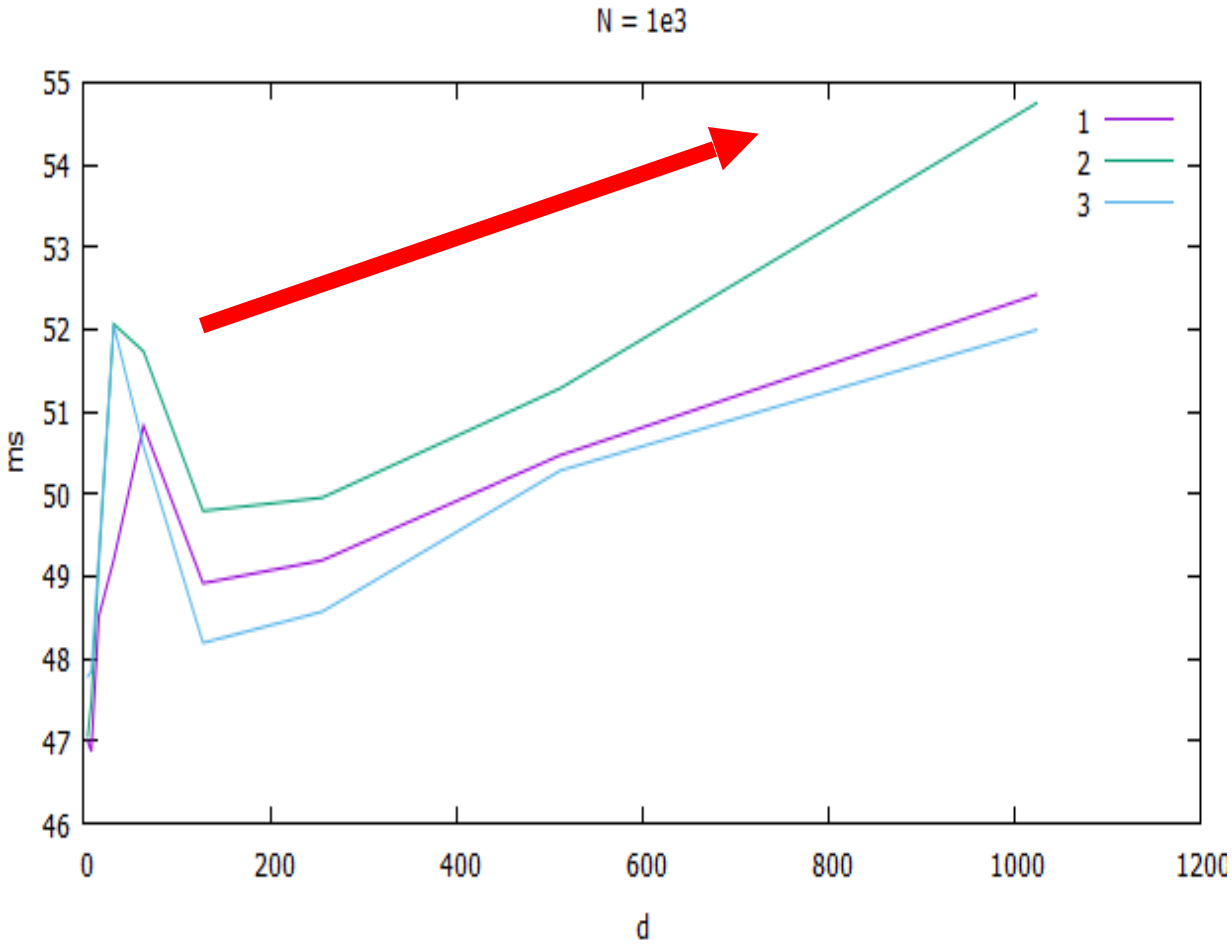
BATCH MERGE
N = 100.000

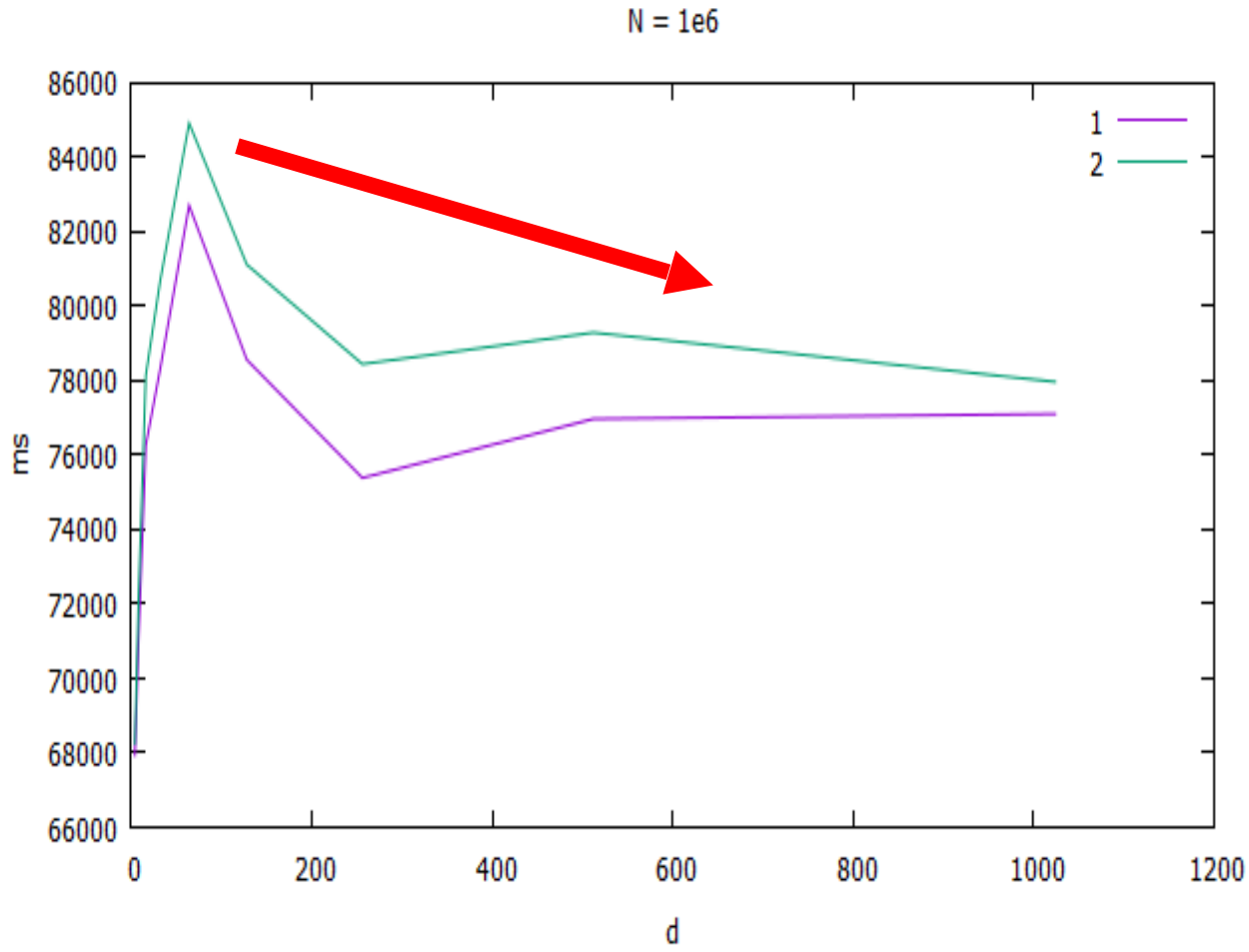


BATCH MERGE
N = 1.000.000



N = 1000 VS N = 1.000.000





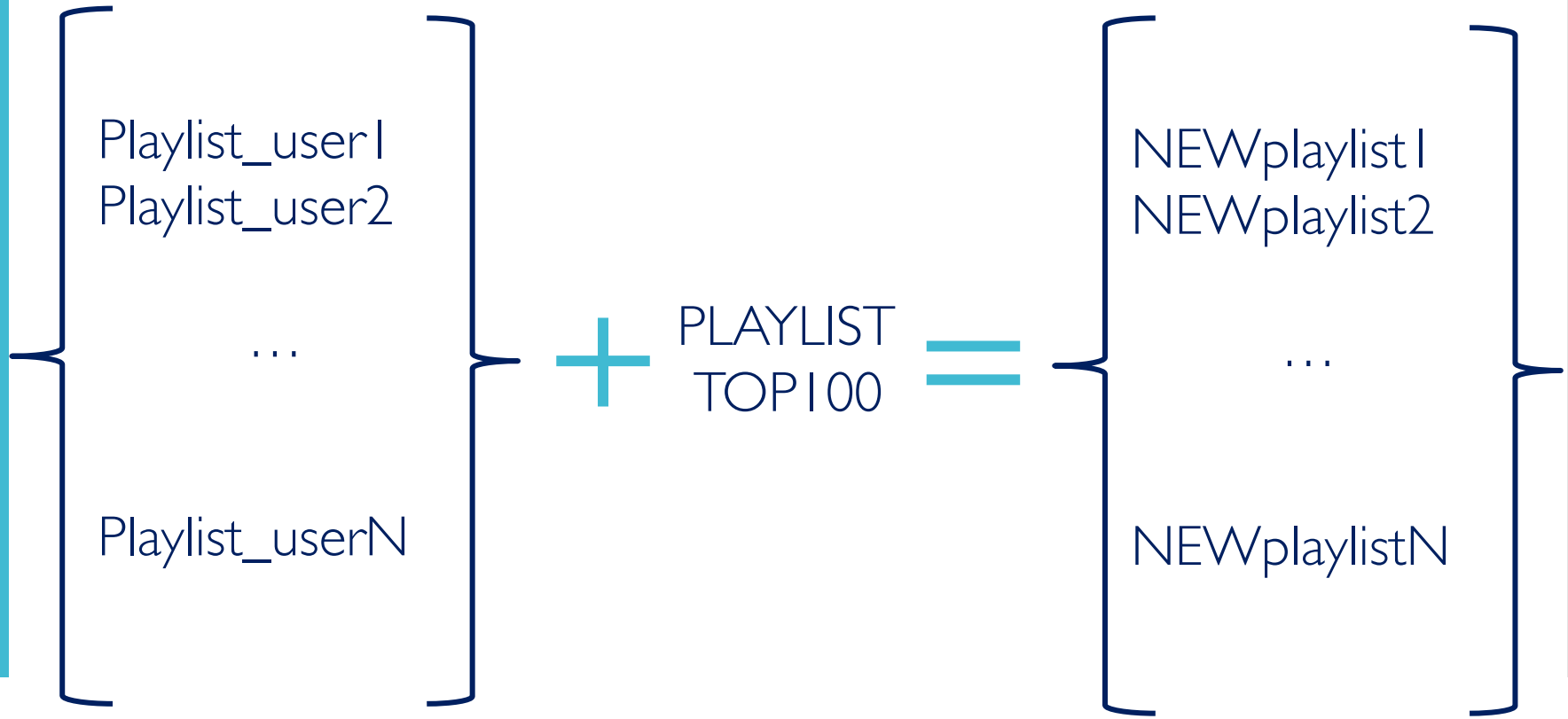
3- BATCH MERGE APPLICATION

*Merging «**Spotify**» Playlists*

Merging «*Spotify*» playlists

GOAL

- GOAL: to obtain **new playlists** which merge users **favourite songs** with **new songs**



Merging «*Spotify*» playlists

DATA STRUCT

- **SONG**

struct song{

- **Title**
- **Artist name**
- **Music genre**
- **Number of plays**

}

- **PLAYLIST**

Array of songs

Merging «*Spotify*» playlists

FEATURES and FUNCTIONS

- **UNIQUENESS**: No repetitions of the same song in the playlist.
  Based on:

  - Title

  - Artist name

- **SORT-CRITERIUM**: Sorting according to a **coefficient** based on:

  - Music Genre

  Frequency of songs of the same musical genre in a playlist

  - Numer of plays

  Percentage with respect to the user total number of plays

- High number of plays
- Favourite genre

COEFFICIENT 

- Low number of plays
- Less preferred genre

COEFFICIENT