Asynchronous K-Means Clustering of Multiple Data Sets

Marek Fiser, Illia Ziamtsov, Ariful Azad,
Bedrich Benes, Alex Pothen

High Performance Computer Graphics

CUDA RESEARCH CENTER

PURDUE UNIVERSITY

Motivation

Clustering bottleneck in Flow Cytometry research

3,000 data sets

25,000 points in **7D** per data set

19 separate clustering tasks per data set

Parallel CPU time: **295 minutes**

Other GPU implementations: **96 minutes** (3x)

K-means clustering

1. Initialize cluster centers (randomly)
2. Assign each data point to the nearest cluster center

$$c \leftarrow \underset{i \in \{1,2,\ldots,k\}}{\arg\min} ||\mathbf{x} - \boldsymbol{\mu}_i||^2$$

Easy to parallelize

3. Re-assign new cluster centers

$$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \cdot \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j, \quad \text{for } i = 1, 2, \ldots, k.$$

Harder to parallelize

4. If any cluster changed go to 2.

Problem definition

Multiple datasets (> 100)

Small data set size (2,000 – 200,000 points)

Low number of clusters (2 – 30)

Low number of dimensions (1 – 50)

All data sets are **processed in serial**

Synchronization overhead is high for small data sets

Synchronization has to be performed for every iteration of k-means algorithm

K-means clustering requires sync

1. Initialize cluster centers (randomly)
2. Assign each data point to the nearest cluster center

$$c \leftarrow \underset{i \in \{1,2,\ldots,k\}}{\arg\min} ||\mathbf{x} - \boldsymbol{\mu}_i||^2$$

3. Re-assign new cluster centers

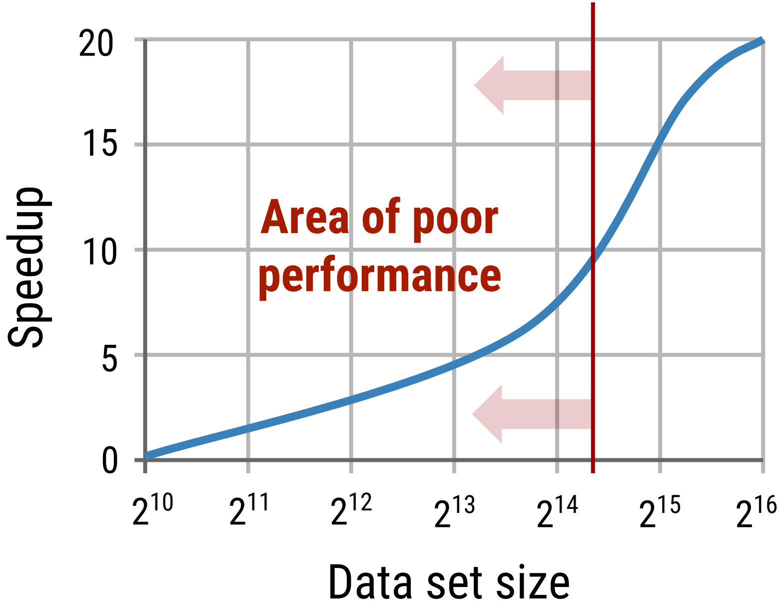
$$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \cdot \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j, \quad \text{for } i = 1, 2, \ldots, k.$$

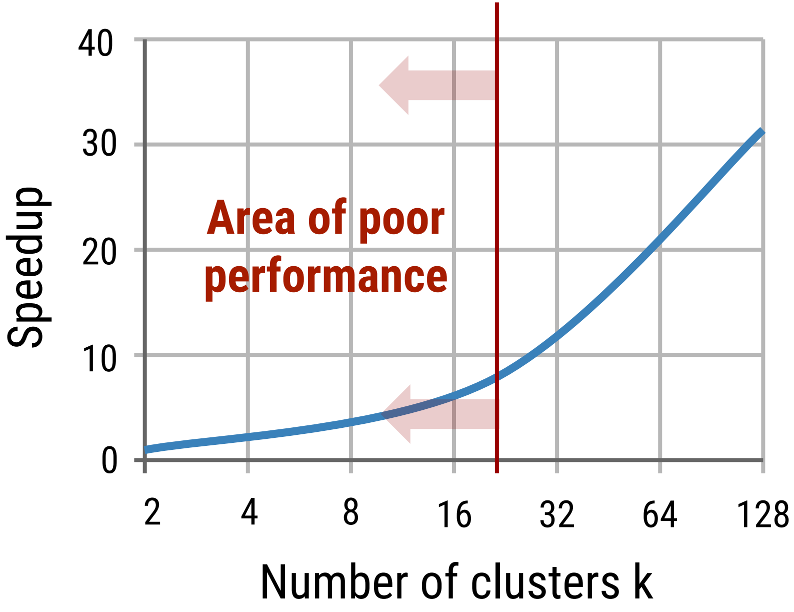
Synchronization

4. If any cluster changed go to 2.

The problem – graphs

Speedup of the GPUMiner (GPU) over the MineBench (CPU)





Our approach

Avoid kernel-wise CPU-GPU synchronization

Use only **one CUDA-block** for clustering

Single CUDA-block can be synchronized within GPU using __syncblocks()

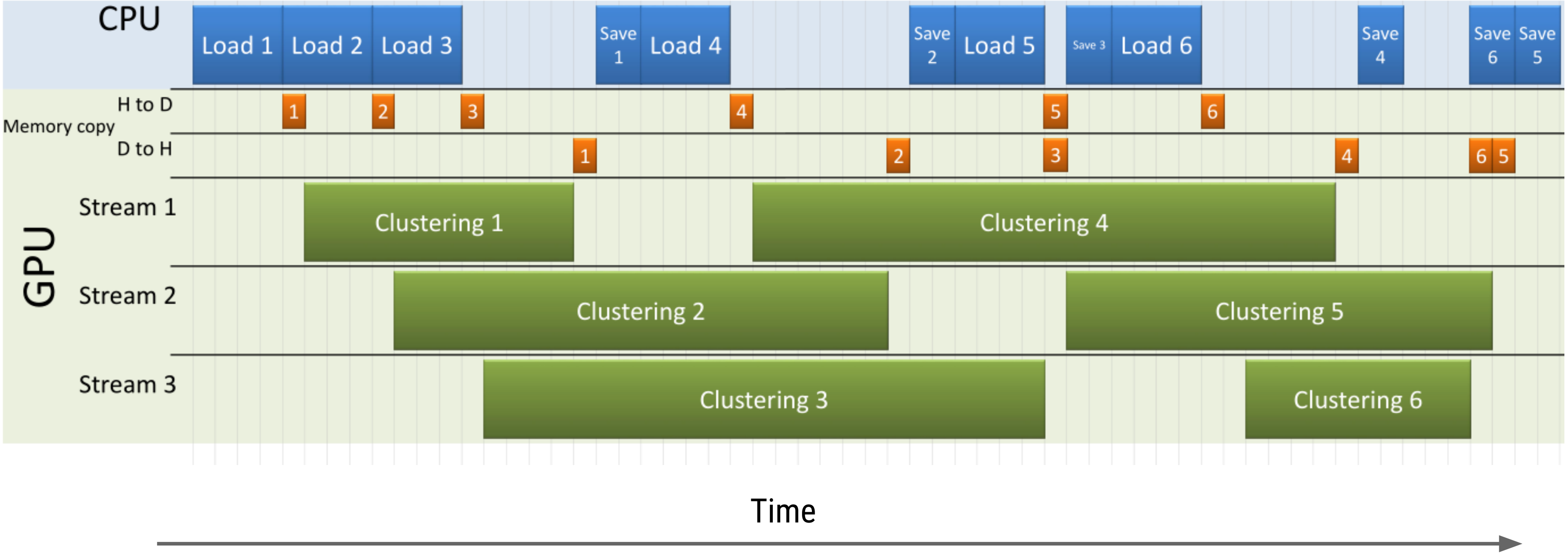
Use **CUDA-streams** to run as many blocks as possible

Thanks to CUDA-streams the clustering is fully asynchronous

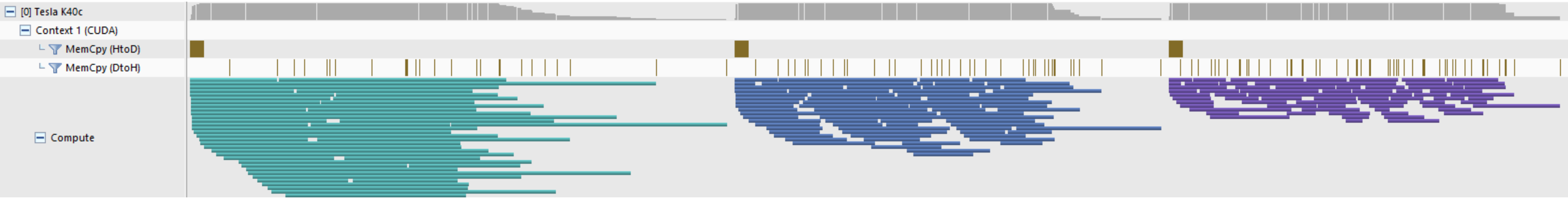
While the GPU is busy clustering the CPU is loading more data sets

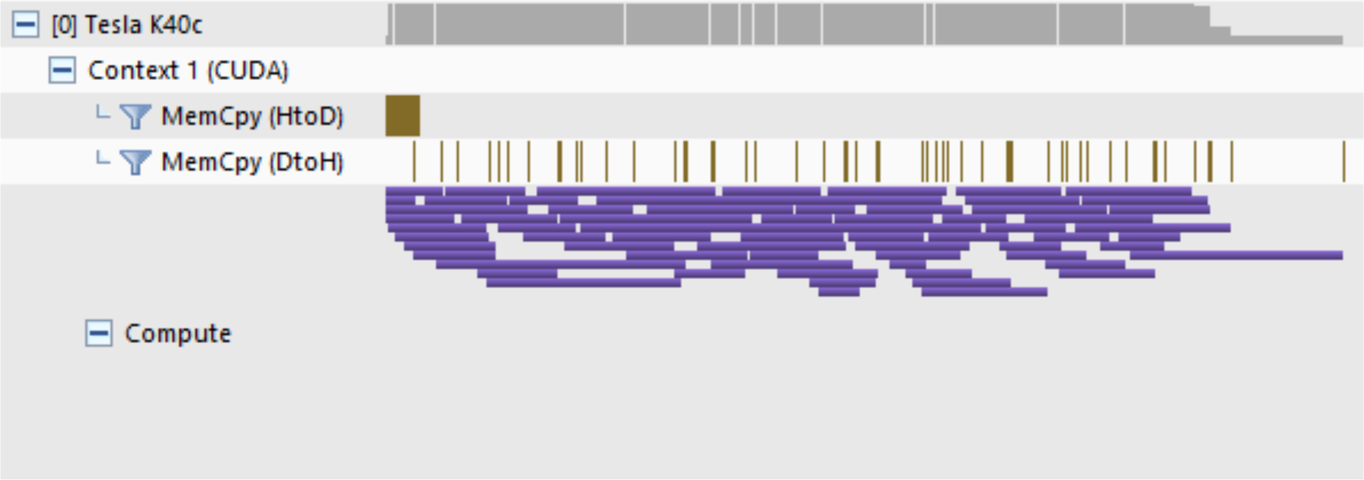
There is nearly no overhead with I/O operations of the CPU

Our approach – Timeline



Our approach – Real timeline





Implementation – Core

```
for each input data set i do {
    D = Load Data (i);  // Loads data from HDD or other source.
    s ← Get Available Cuda Stream ();  // Blocking operation
    Ensure Enough Pinned Memory (D, s);  // Every stream has associated pinned memory
    Copy Data To Pinned Memory (D, s);

    Schedule Mem Copy From Host To Device On Stream (s);  ┐
    Schedule Cuda Kernel Invocation On Stream (s);        ├ Asynchronous
    Schedule Mem Copy From Device To Host On Stream (s);  ┘ (non-blocking)
}
```

Implementation – Get Cuda Stream function

```
freeStream ← null ;
while ( freeStream == null ) {
    for each stream s_i do {
        if ( Is Stream Finished (s_i) ) {
            D ← Download Results From Pinned Memory (s_i);
            Save Results ( D );
            freeStream = s_i ;
}   }   }
return freeStream;
```

Non-paged (pinned) memory

Required to use with CUDA streams

Uses Direct memory access (DMA) for memory copies

Used for both input and output

It is allocated big enough, size = max(input size, output size)

Pooled per stream

Memory is re-used for consecutive datasets, or re-allocated if needed

Flow Cytometry Data

2,872 individual data sets

25,000 points per dataset, **7** dimensions

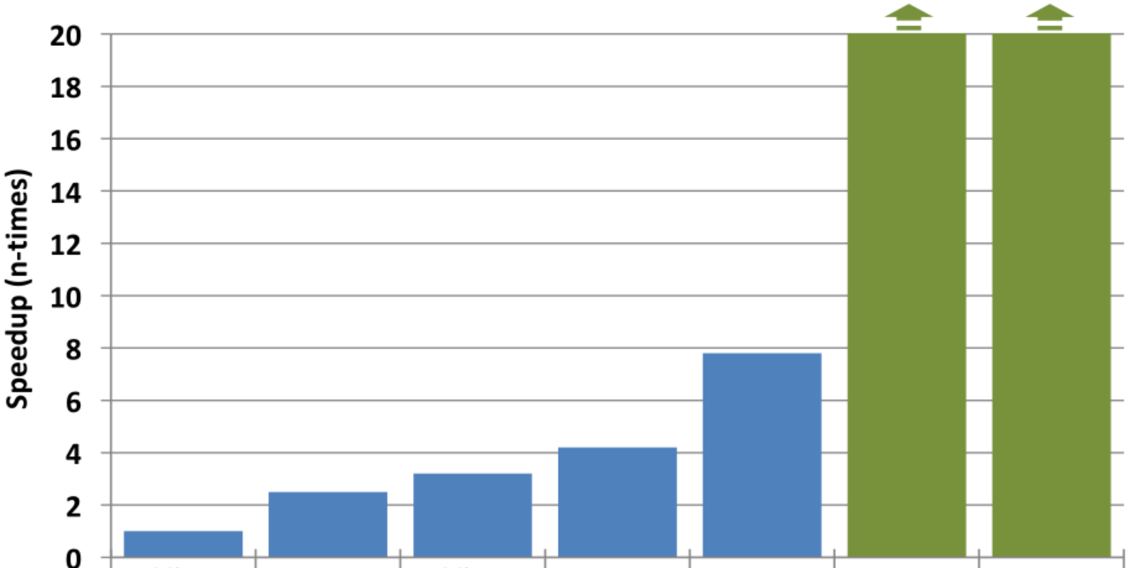
19 separate clusterings for k={2, …, 20}

Total: 2,872 · 19 = **54,568** individual clustering tasks

CPU: Intel Core i7 2600k @ 3.40GH

GPU: Tesla K40

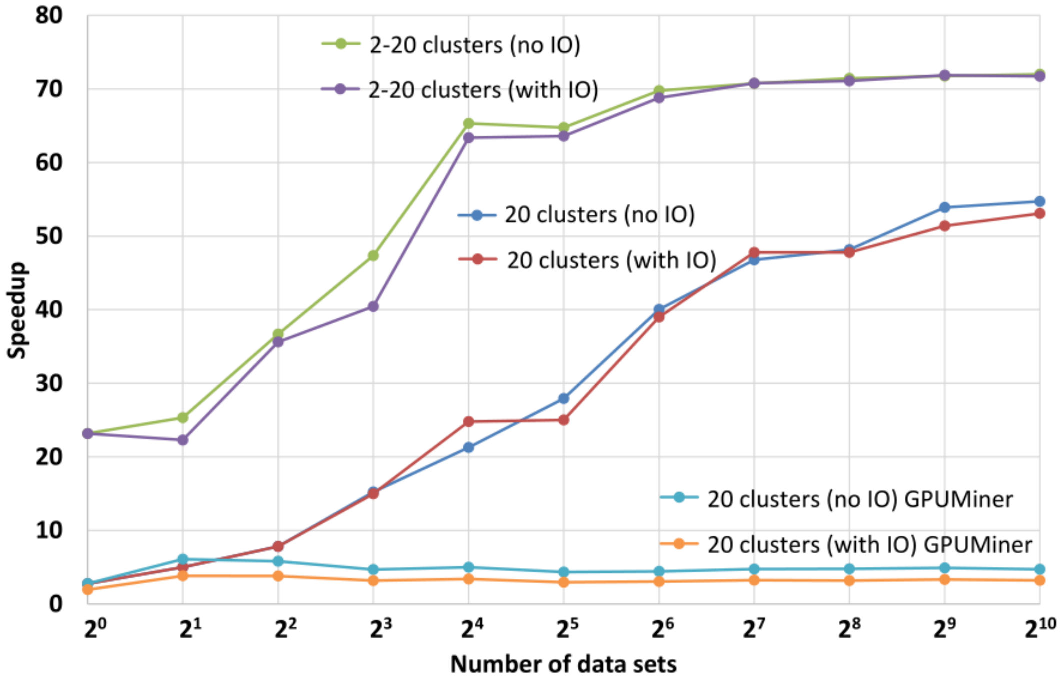
Results on the Flow Cytometry Data



	Mine-bench serial	STAMP parallel	Mine-bench parallel	Serban Giuroiu	GPUMiner	Our GPU	Our GPU kmeans++
Speedup	1	2.5	3.2	4.2	7.8	198.1	207.8
Time (m)	747	295	233	180	96	3.8	3.5

Mine bench – North Western, STAMP – Stanford, GPUMiner – Hong Kong University of Science and Technology

Speedup as a function of data sets count



d = 5
n = 20,000

Strengths

High performance on multiple data sets

Low memory requirements

- Can process unlimited amount of small data sets
- Data sets can have different sizes

Asynchronous – hides I/O overhead

The kernel uses only one CUDA block

- Simplifies programming and enables synchronization

Limitations

The kernel can use only one CUDA block

~30 data sets have to fit in the GPU memory at once

Number of points and their dimensions is the limitation

Has to process multiple data sets

Conclusion

High speedup due to synchronization overhead elimination

Our technique can be applied to other problems which:

- Independently process multiple input data sets
- Data sets are relatively small
- Algorithm may require synchronization

Asynchronous K-Means Clustering of Multiple Data Sets

Marek Fiser

mfiser@purdue.edu
http://www.marekfiser.com

This slides can be viewed on: http://goo.gl/arSaoF

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!