Scalable Functional Connectivity and Functional Clustering for fMRI

Orhan Fırat, Ph.D. Student
Department of Computer Engineering,
Middle East Technical University
orhan.firat@ceng.metu.edu.tr

Affiliation and Collaboration

ODTÜ METU

Department of Computer Engineering
Neuro-ceng

KOÇ UNIVERSITY

Department of Psychology

ODTÜ METU

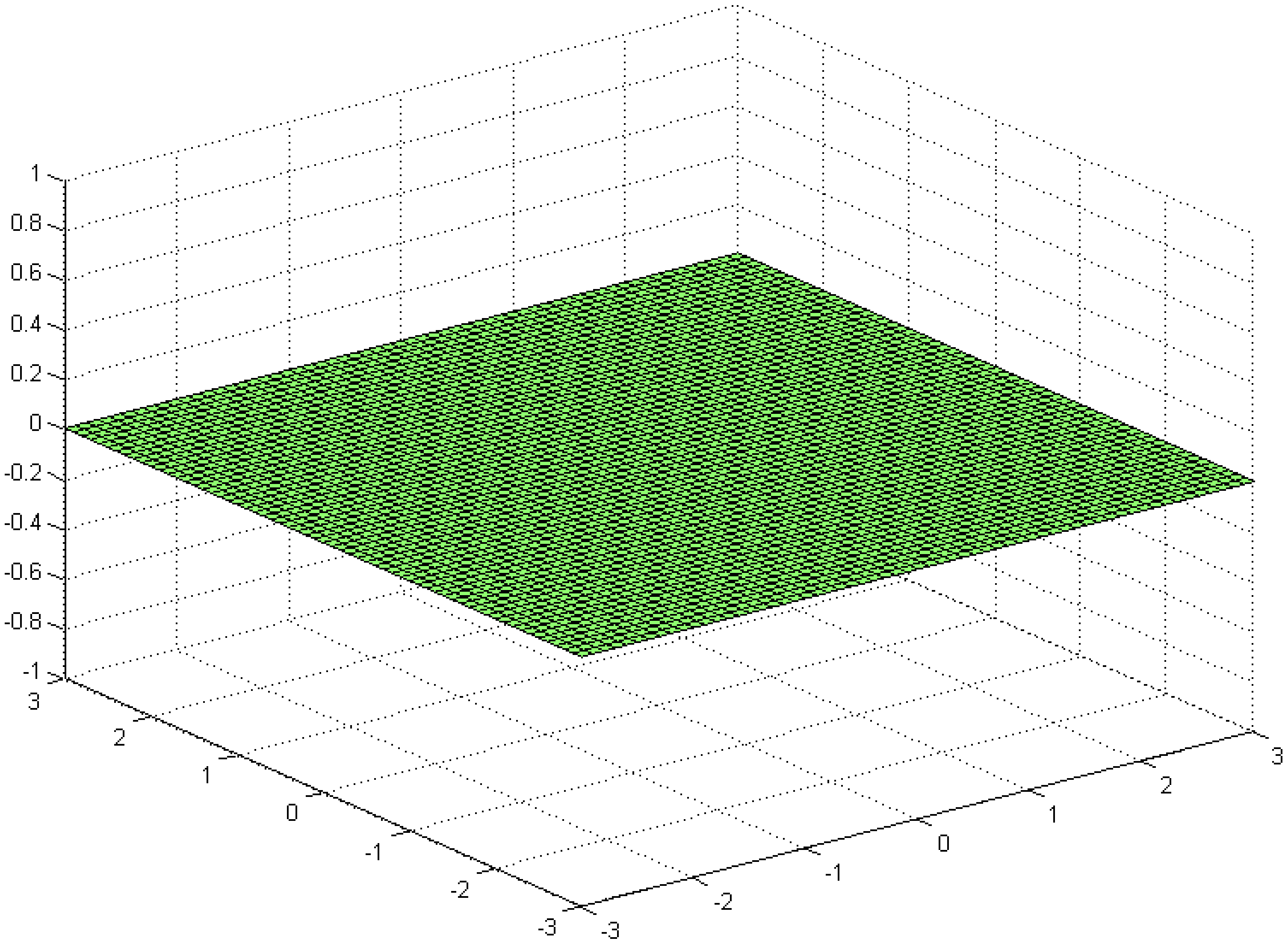
NVIDIA
CUDA™ RESEARCH CENTER

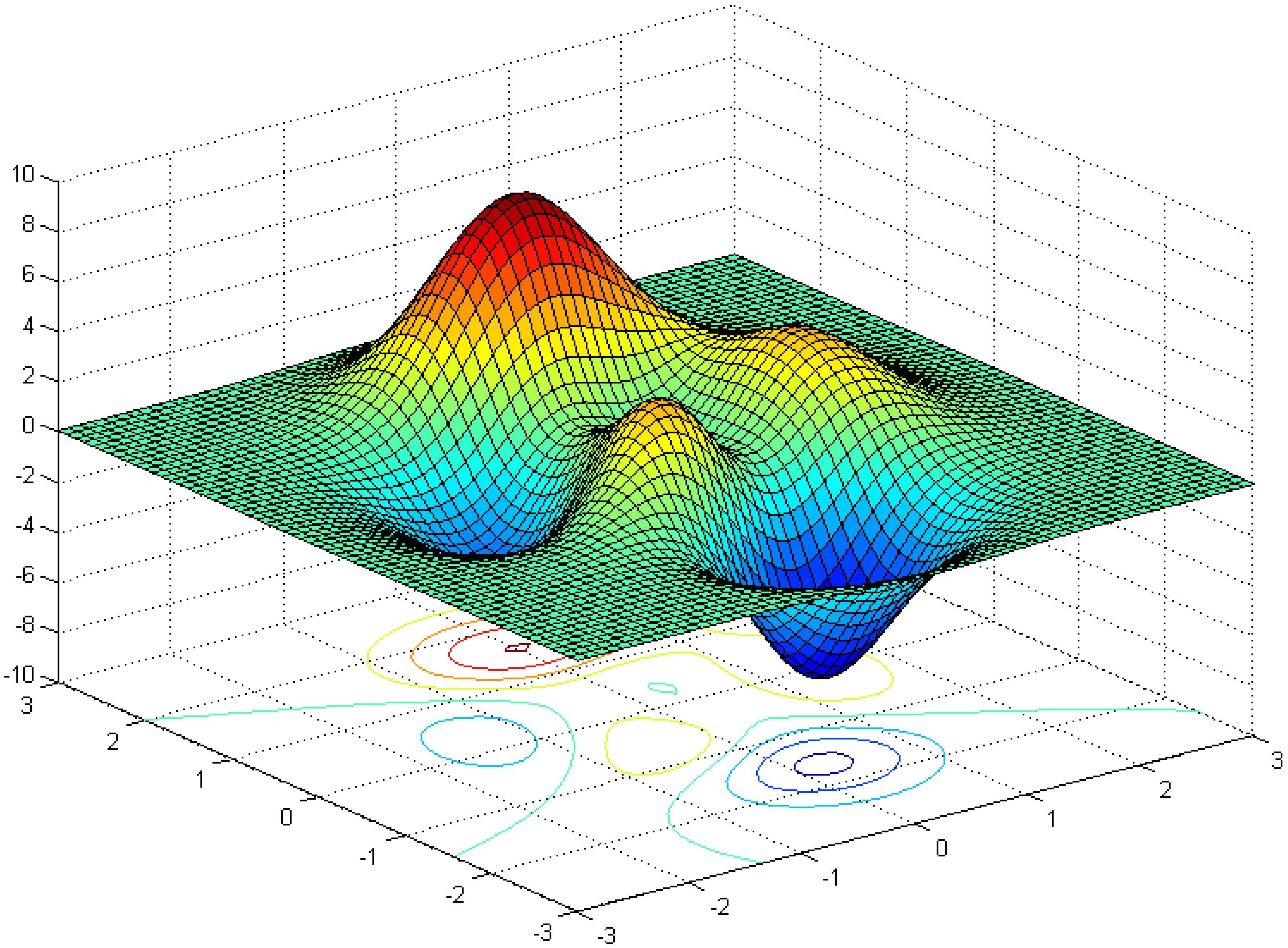
Graduate School of Informatics

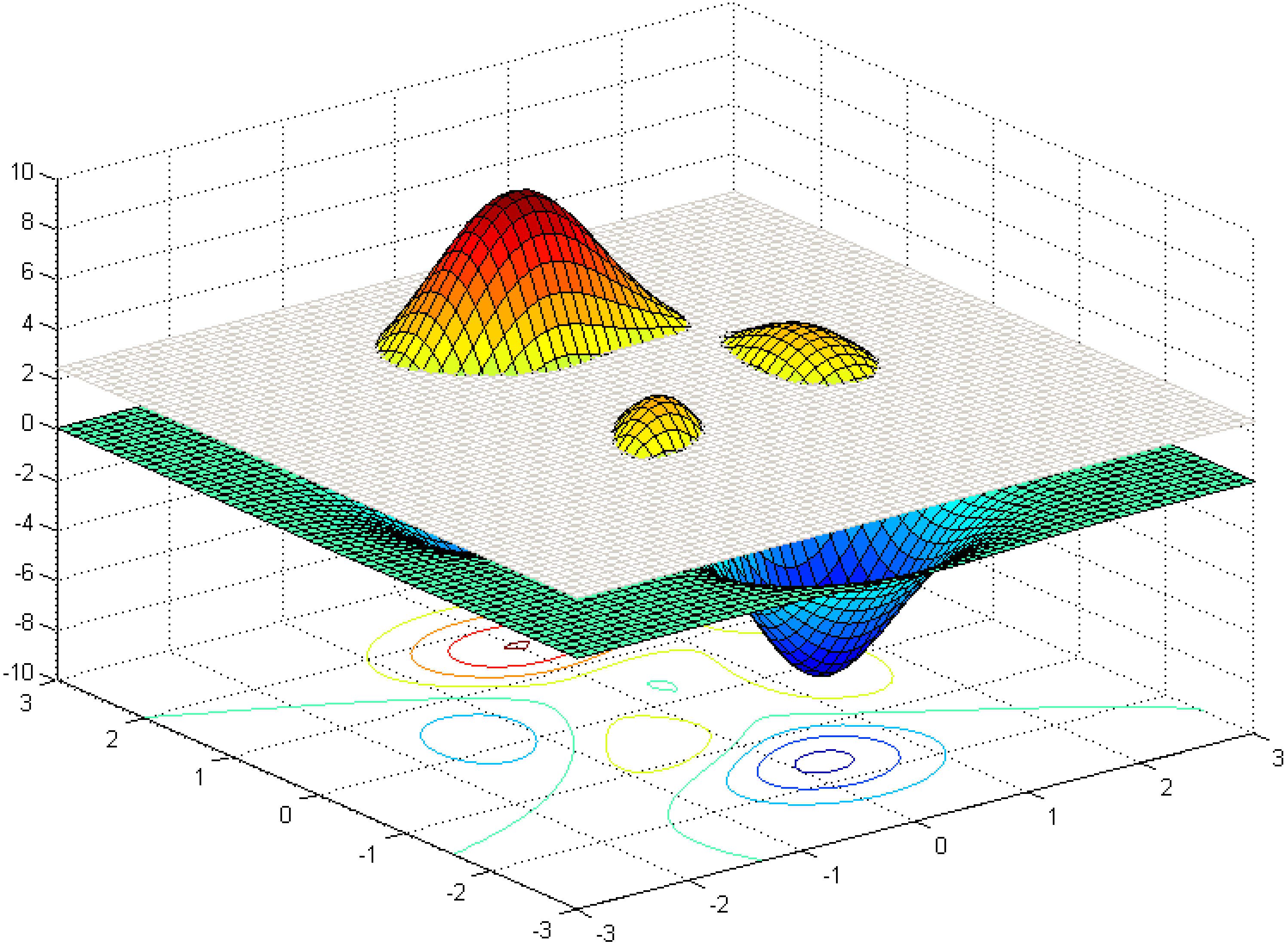
What do we do?

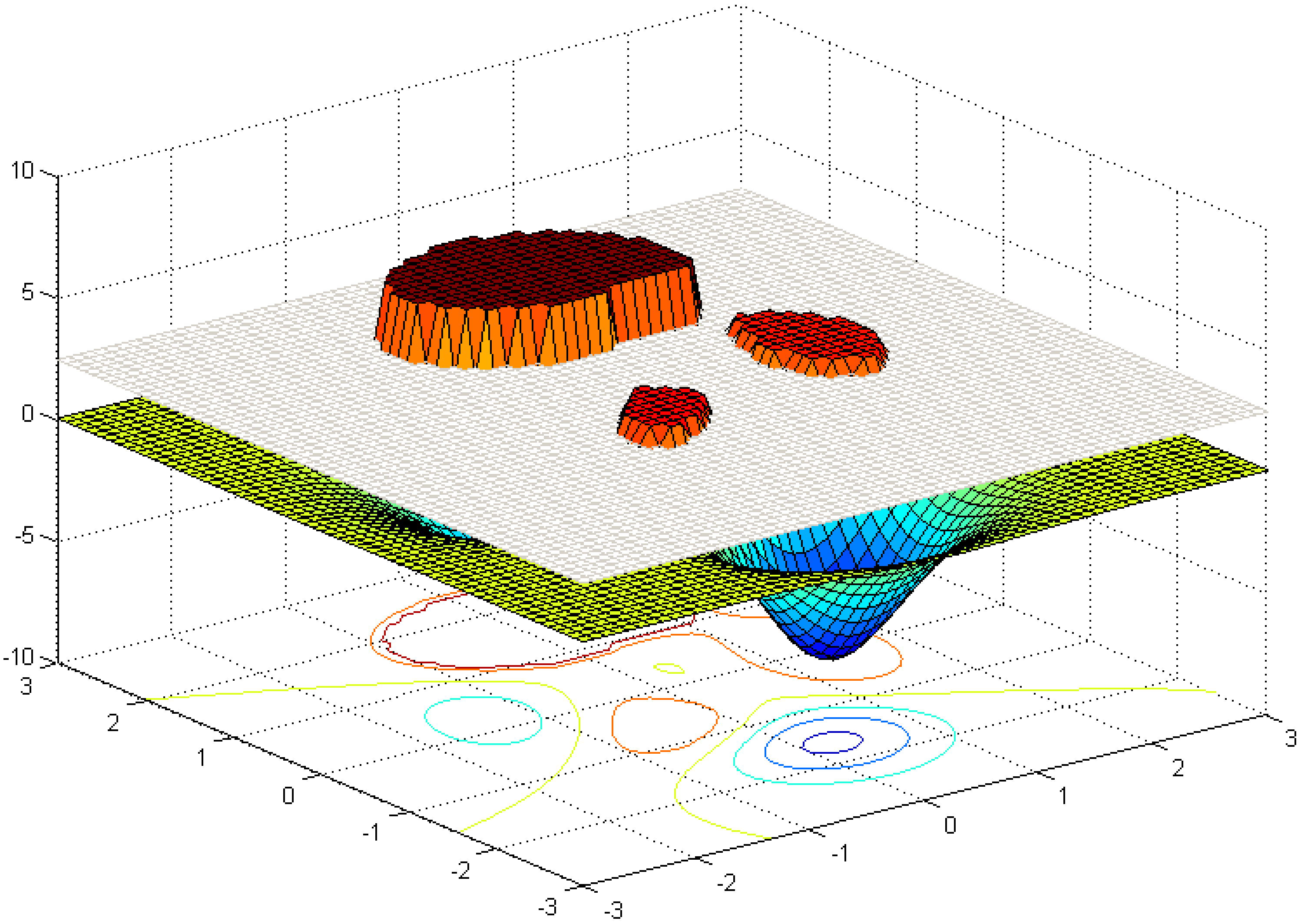
Computational Neuroscience

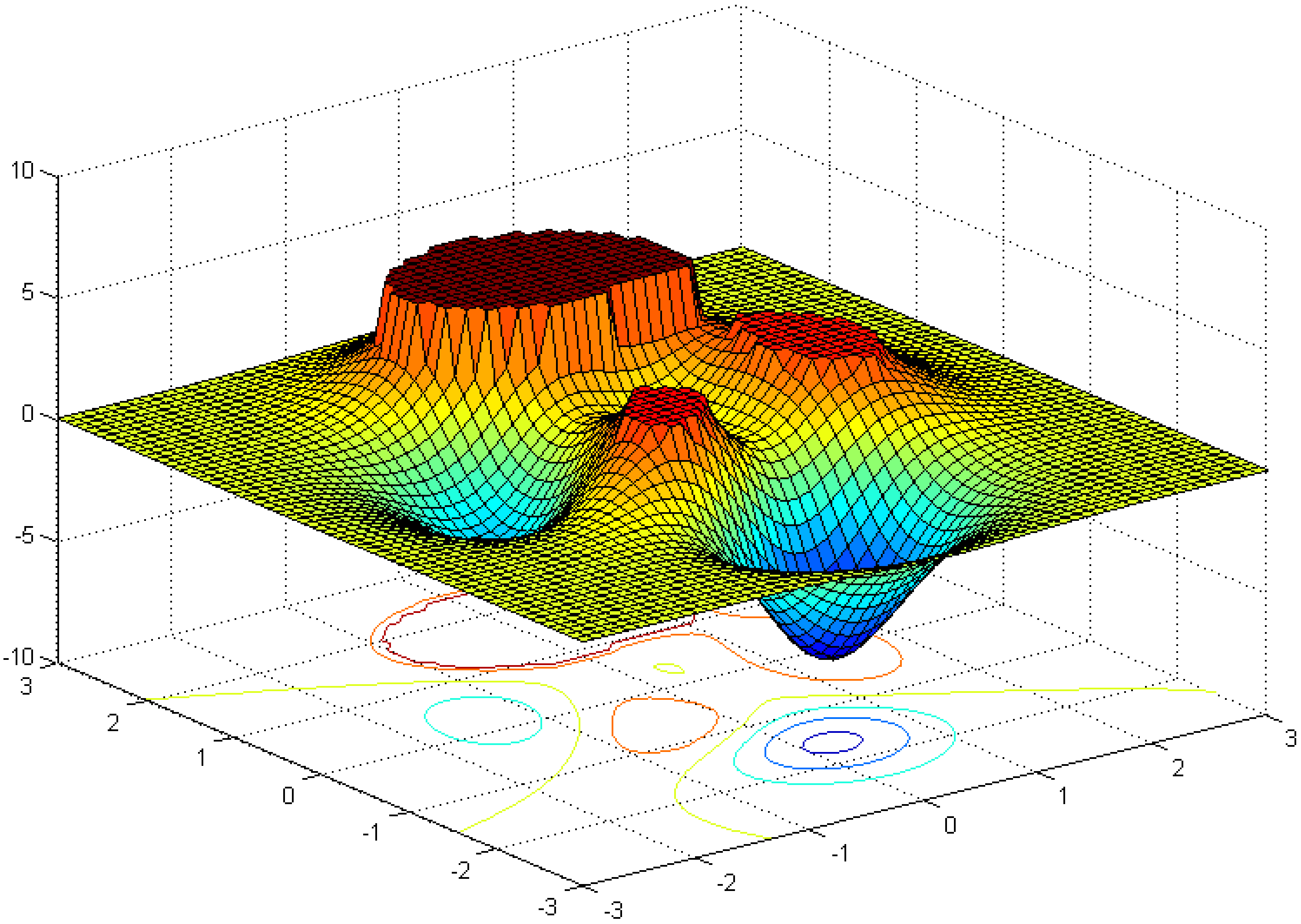
- Domain : fMRI
- Objective : Develop representative and informative models for cognitive processes
  - ➢ Pattern Analysis (MVPA), Machine Learning
  - ➢ Structure Learning
  - ➢ Structured Prediction
  - ➢ Mind Reading ☺


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

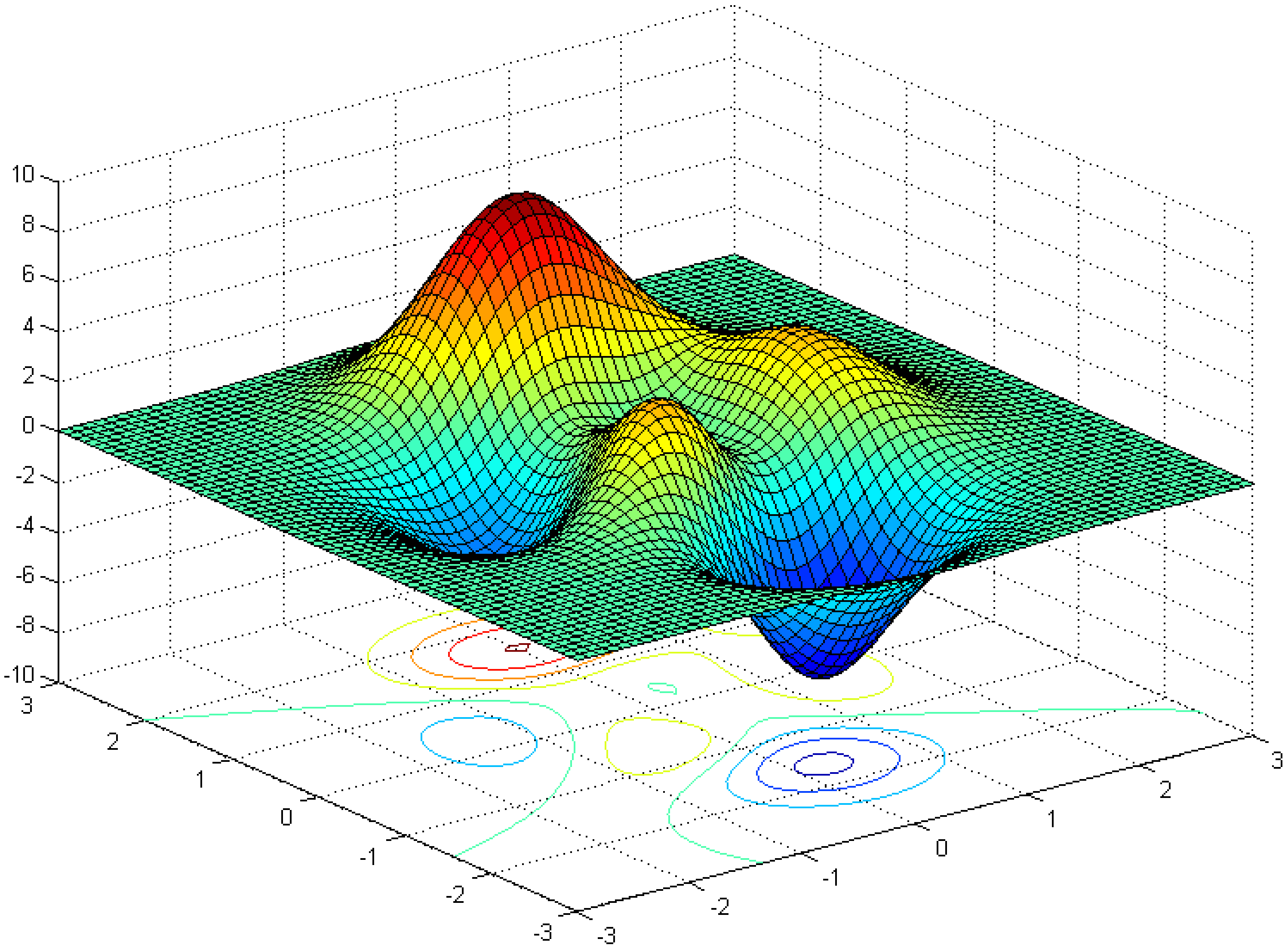

10
8
6
4
2
0
-2
-4
-6
-8
-10
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


10
8
6
4
2
0
-2
-4
-6
-8
-10
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


10
5
0
-5
-10
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3



10
5
0
-5
-10
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


10
8
6
4
2
0
-2
-4
-6
-8
-10
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

Pattern Analysis and Machine Learning

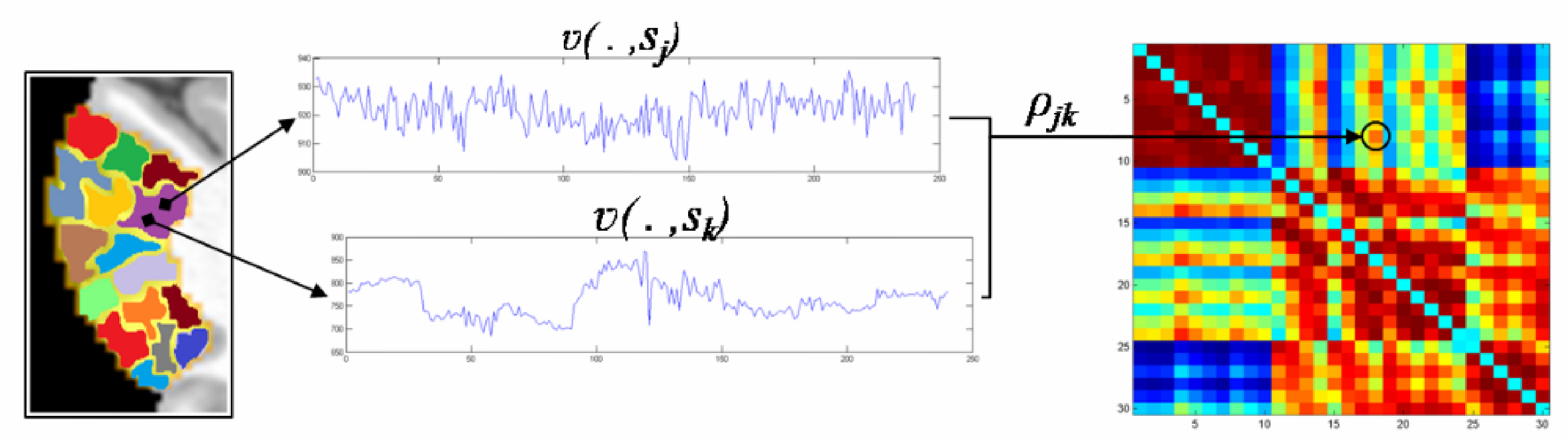
- More data → robust approaches for representation / discrimination
- Science
  - New learning paradigms
  - New models for cognitive process representation
- Engineering
  - Computational complexity; time-memory complexities.
  - Practical solutions for researchers

How do we do?
(Engineering Part)

Case Studies:

1. Applying large scale functional connectivity on GPU
2. Clustering / parcellation functional connectivity matrices on hybrid architectures

Functional Connectivity



Functional Connectivity (FC) is the statistical dependence between remote neural elements or regions across time.

1. Find correlation between two voxels' time series using a correlation metric.
2. Construct correlation matrix (connectivity matrix) by using each pairs' correlation response.

Scalability of Functional Connections

- Connectivity matrices are expensive in voxel level.

- Considering functional relations of a voxel with all other voxels;

- 8142 voxels makes ~33M functional relations
- 82926 voxels makes ~3438M funtional relations

Scalability of Functional Connections

- Suppose region of interest consists of *M* voxels
  - Let computation time of functional relations for a voxel pair (υ_i -υ_j) takes t second
  - Computation time for functional connectivity matrix

    T_{fc} takes

    $$T_{fc} = M^2 \text{ x } t \text{ seconds}$$
  - T_{fc} grows exponentially on *M.*

Ex: Suppose computation time of each functional connectivity metric takes ~1sec

For 8142 voxels,

T_{fc} = (8142*8412)/2-8142 = 33.137.940 sec ≅ 9204hours ≅ 383 days

with 256-way perfect parallelization (in a cluster) ≅ 1,5 days

Scalability Analysis

- Calculating even ~8000x8000 sized correlation matrix is expensive where our domain spans ~80.000 voxels and their relations.
- Apply GPU parallelization for speed-up.
- Reformulate correlation metric for GPU.

Pearson Correlation Coefficient

- Cross-correlation of any two individual time-series *x* and *y* is given by

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \ \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

- This can also be written as:

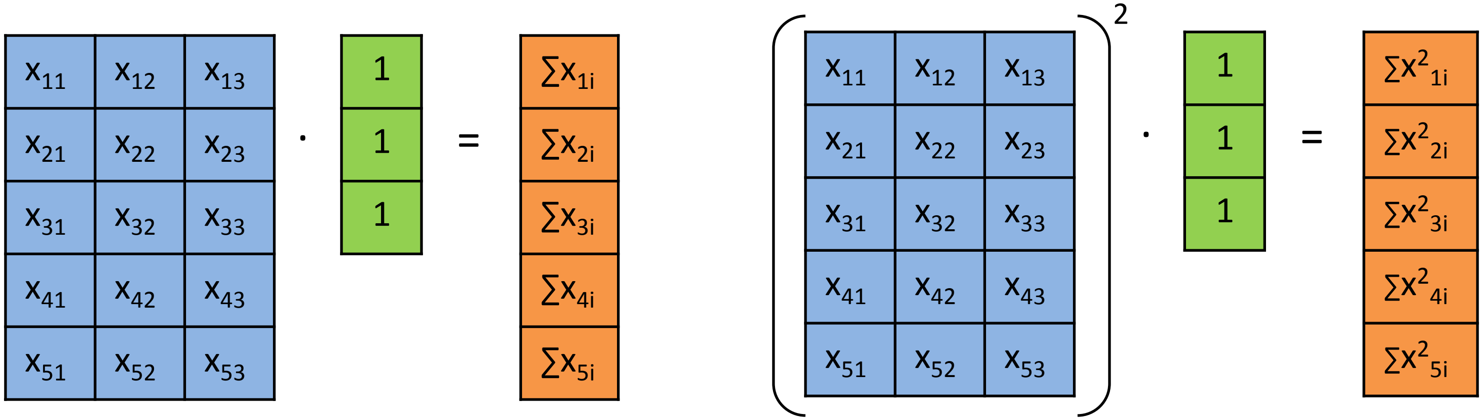
$$\rho_{xy} = \frac{\sum x_i y_i - n\overline{xy}}{\sqrt{\left(\sum x_i^2 - n\bar{x}^2\right)\left(\sum y_i^2 - n\bar{y}^2\right)}}$$

- Design appropriate GPU architecture for reformulated correlation

$$\rho_{xy} = \frac{\sum x_i y_i - n\overline{xy}}{\sqrt{\left(\sum x_i^2 - n\bar{x}^2\right)\left(\sum y_i^2 - n\bar{y}^2\right)}}$$

$$\rho_{xy} = \frac{\sum x_i y_i - n\left(\sum x_i / n\right)\left(\sum y_i / n\right)}{\sqrt{\left(\sum x_i^2 - n\left(\sum x_i / n\right)^2\right)\left(\sum y_i^2 - n\left(\sum y_i / n\right)^2\right)}}$$

- Highlighted summations can be calculated with a single matrix-vector multiplication



$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sum x_{1i} \\ \sum x_{2i} \\ \sum x_{3i} \\ \sum x_{4i} \\ \sum x_{5i} \end{bmatrix}$$

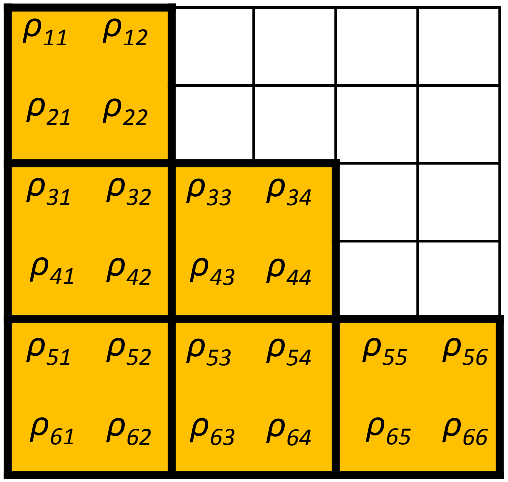
$$\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{bmatrix}\right)^2 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sum x^2_{1i} \\ \sum x^2_{2i} \\ \sum x^2_{3i} \\ \sum x^2_{4i} \\ \sum x^2_{5i} \end{bmatrix}$$

$$\rho_{xy} = \frac{\sum x_i y_i - n(\sum x_i/n)(\sum y_i/n)}{\sqrt{(\sum x_i^2 - n(\sum x_i/n)^2)(\sum y_i^2 - n(\sum y_i/n)^2)}}$$

- Only highligted summation needs to be calculated in a kernel as the last step.
- Other summation results are stored in texture memory.

How to Handle Massive Data

- Overlay CUDA-Grids on resulting symmetric correlation matrix (lower-diagonal).
- Determine chunk size and process each chunk sequentially.
- Transfer corresponding time-series data to texture memory.
- Process current chunk then load subsequent chunk (in column major order)

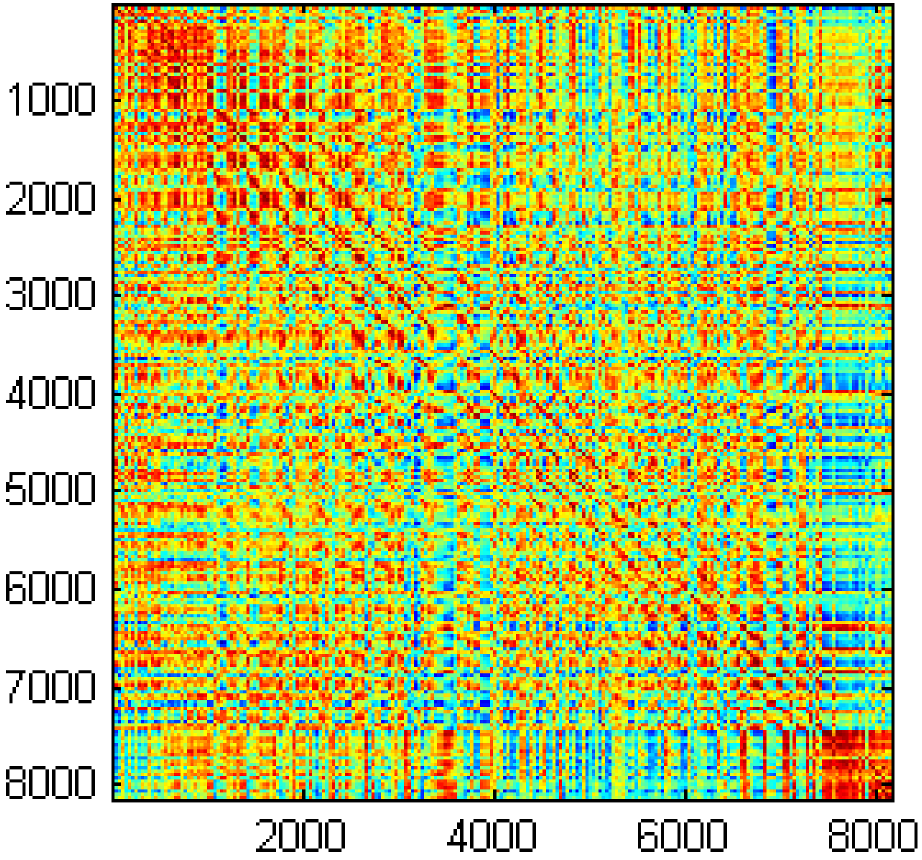


$$\begin{array}{cc|cc|cc}
\rho_{11} & \rho_{12} & & & & \\
\rho_{21} & \rho_{22} & & & & \\
\hline
\rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} & & \\
\rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} & & \\
\hline
\rho_{51} & \rho_{52} & \rho_{53} & \rho_{54} & \rho_{55} & \rho_{56} \\
\rho_{61} & \rho_{62} & \rho_{63} & \rho_{64} & \rho_{65} & \rho_{66}
\end{array}$$

Overall Algorithm

1. Load data chunk to device memory
2. Calculate $\sum x_i$ with *cublasSgemv*,
   - save resulting vector on device memory
3. Square data matrix by using thrust library transform routine and calculate $\sum x_i^2$ with *cublasSgemv.*
   - save resulting vector on device memory
4. Load source and destination time series into texture memory.
5. Calculate pairwise correlation on GPU
   - save resulting matrix chunk on disk
6. Return to step 4

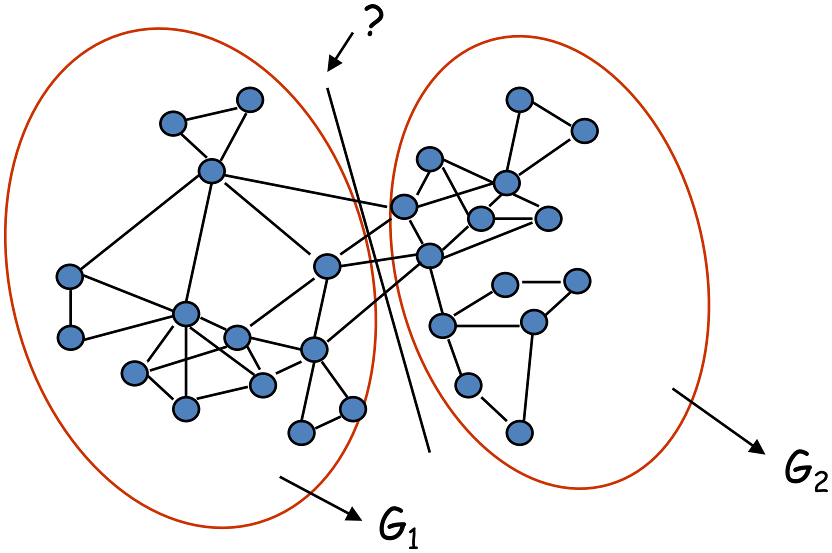
Functional Connectivity Matrix for 8000 voxels



Method	Conventional	Proposed
8142 voxels	~30 minutes	28 seconds
82926 voxels	~3 days	24 minutes

Case Study 2:
Functional Parcellation / Clustering

- Consider a weighted Graph G=(V, E, A)
  - it is possible to partition G



  - into smaller components G_1 and G_2 where $G = G_1 \cup G_2$ with specific properties defined over A

How to Partition the Graph?

- Problem:

  finding an optimal graph (normalized) cut is NP-hard

- Solution:

  approximation & heuristics

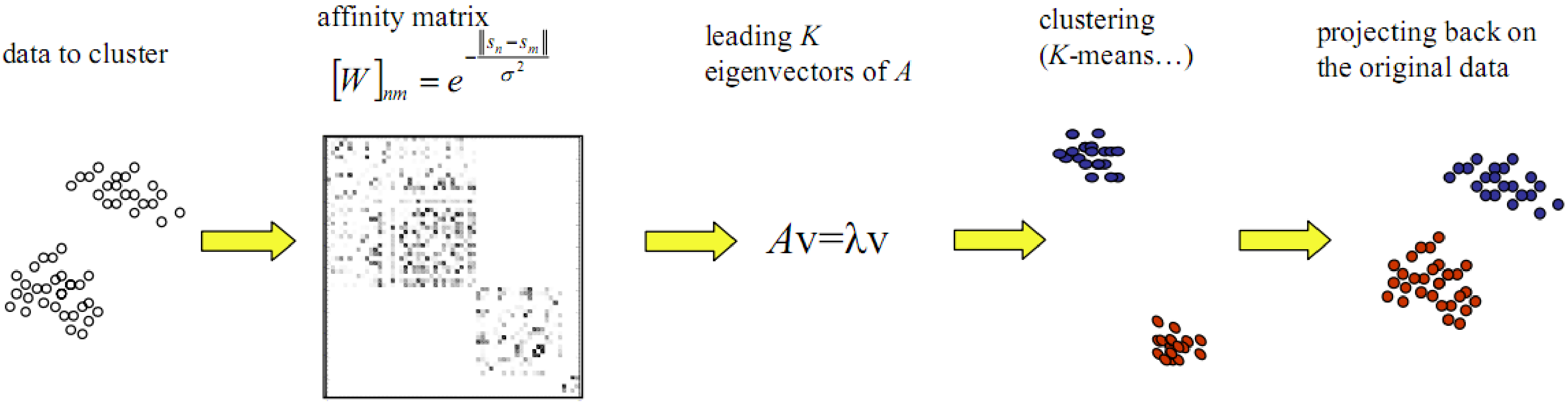
- Approximation: spectral graph partitioning
  - Partitioning the graph by spectral analysis

    (spectrum of a matrix is the set of its eigenvalues)

  - e.g. Spectral clustering

    Based on Laplacian Matrix, or **Graph Laplacian**

Spectral Clustering



data to cluster

affinity matrix

$$[W]_{nm} = e^{-\frac{\|s_n - s_m\|}{\sigma^2}}$$

leading K eigenvectors of A

$$A\mathrm{v} = \lambda \mathrm{v}$$

clustering (K-means…)

projecting back on the original data

- SC is sensitive to the scaling parameter of the RBF kernel
- Main difference between algorithms is the definition of A = func(*W*)

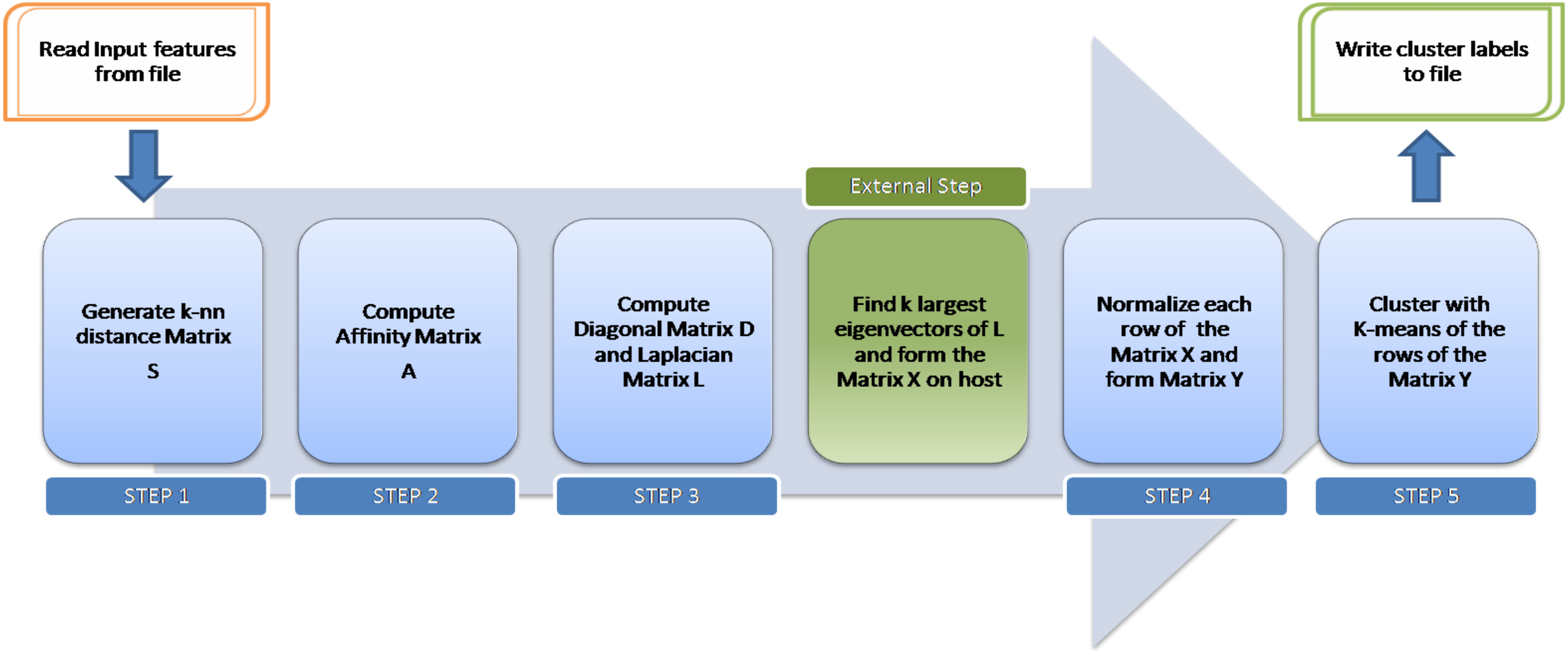
Major Operations and Parallelization of Spectral Clustering

- Sparse vector-vector multiplications

- Matrix-vector operations

- Matrix problems usually imply huge possibility to parallelization
  - Compute-intensive
  - Matrices are able to be divided by rows, columns or blocks

Algorithm Flow



Read Input features from file

Generate k-nn distance Matrix S

STEP 1

Compute Affinity Matrix A

STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L

STEP 3

External Step

Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y

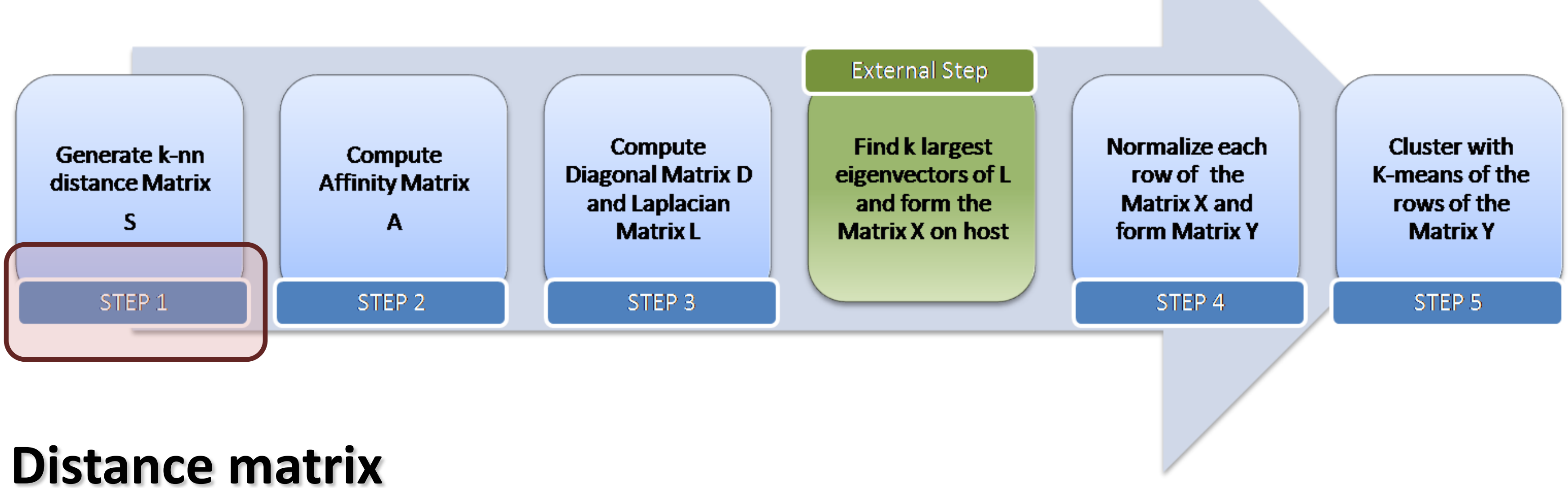
STEP 4

Cluster with K-means of the rows of the Matrix Y

STEP 5

Write cluster labels to file

- Step-wise analysis
  - Consists of comparisons with either a CPU implementation or GPU implementations



Generate k-nn distance Matrix S

STEP 1

Compute Affinity Matrix A

STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L

STEP 3

External Step

Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y

STEP 4

Cluster with K-means of the rows of the Matrix Y

STEP 5

Distance matrix

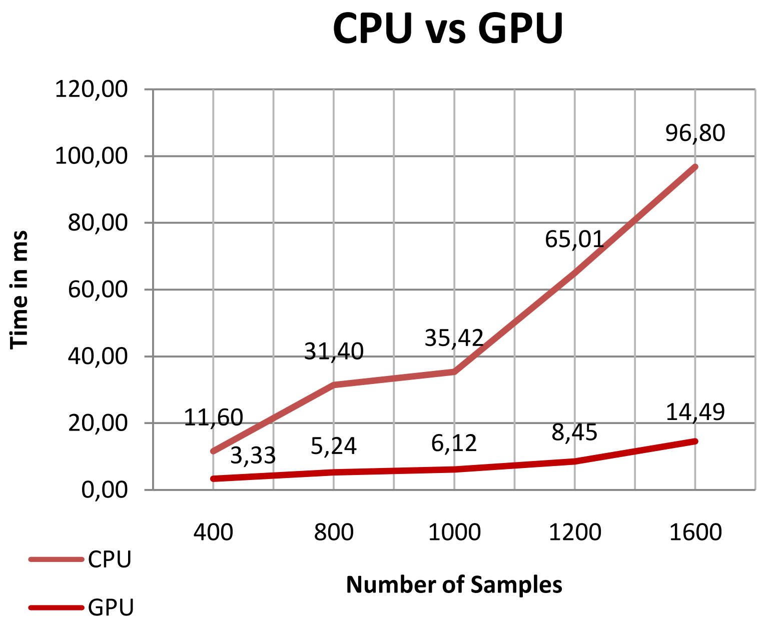
Calculate distance from each point to others

$$\mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

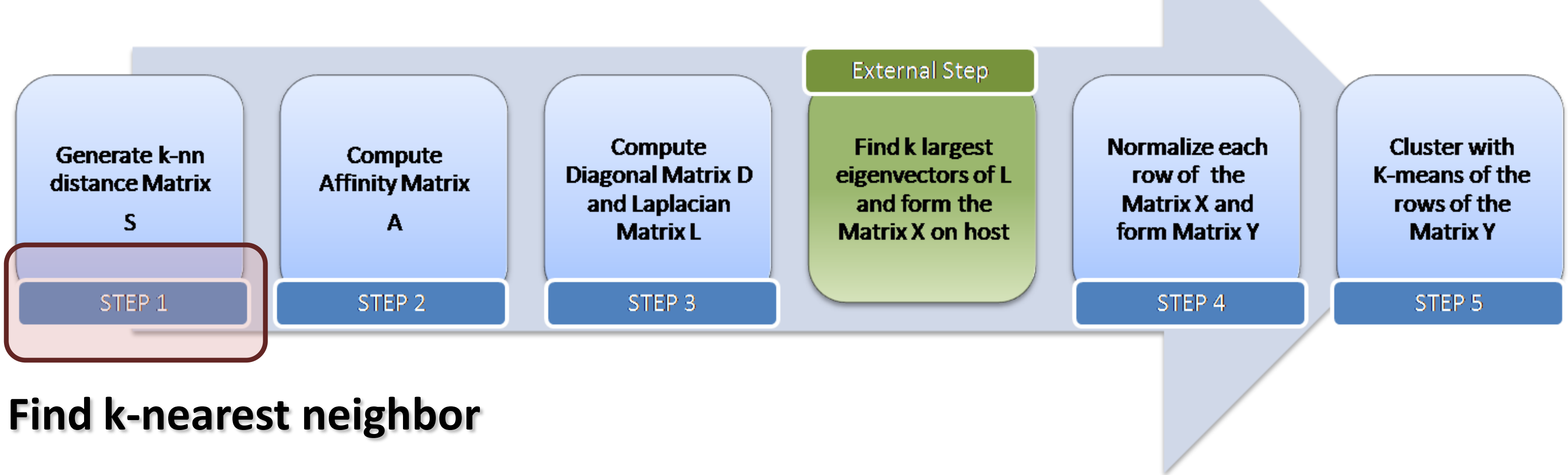
$$\|\mathbf{q} - \mathbf{p}\| = \sqrt{\|\mathbf{p}\|^2 + \|\mathbf{q}\|^2 - 2\mathbf{p}\cdot\mathbf{q}}.$$

Multiply feature matrix with its transpose using CUBLAS

$$\begin{bmatrix} pp & pq & pq \\ qp & pp & pq \\ qp & qp & pp \end{bmatrix}$$



Launch 1 additional kernel to calculate distance matrix



Generate k-nn distance Matrix S
STEP 1

Compute Affinity Matrix A
STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L
STEP 3

External Step
Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y
STEP 4

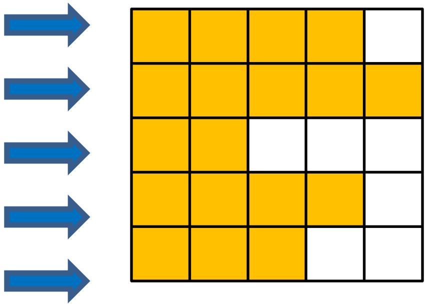
Cluster with K-means of the rows of the Matrix Y
STEP 5

Find k-nearest neighbor

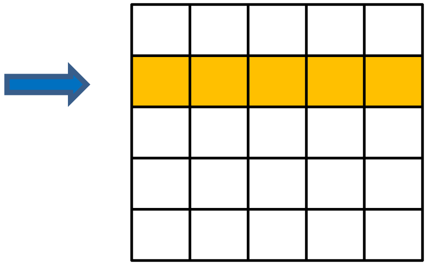
Extract closest points row-wise from its diagonal

Sort rows of the distance matrix

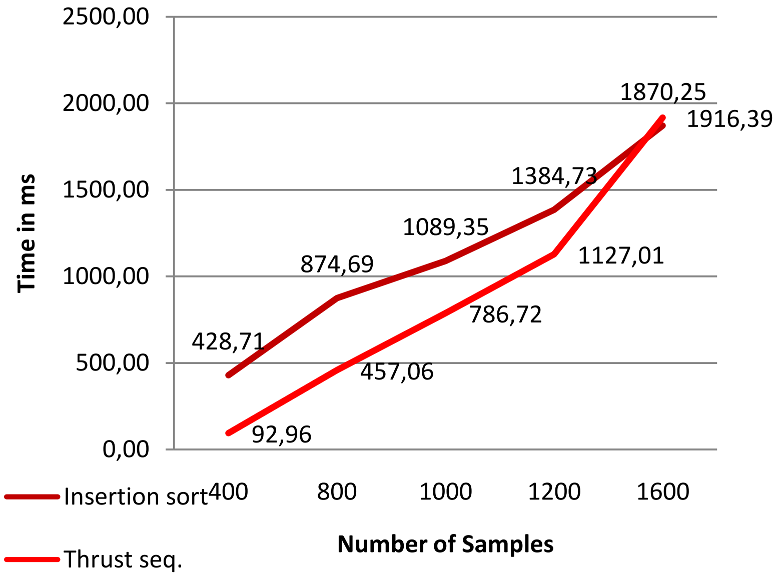
1. Insertion sort with kernel

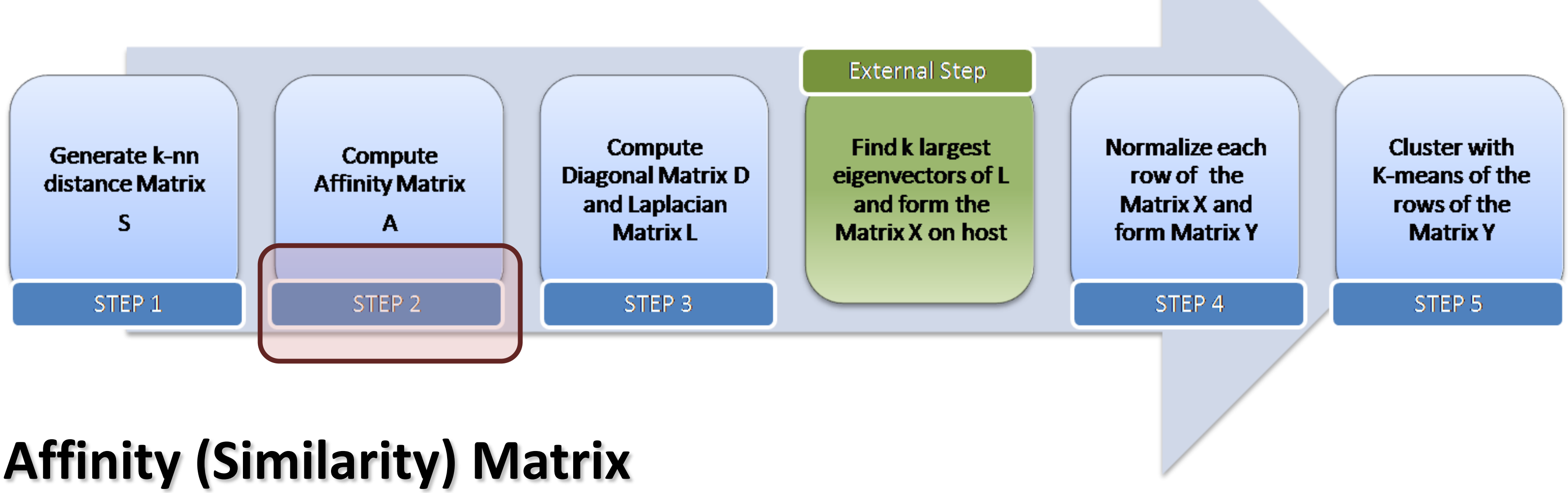


2. Sequential sort with thrust::sort



Insertion sort vs Thrust sort





Generate k-nn distance Matrix S
STEP 1

Compute Affinity Matrix A
STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L
STEP 3

External Step
Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y
STEP 4

Cluster with K-means of the rows of the Matrix Y
STEP 5

Affinity (Similarity) Matrix

Convert distance matrix to a sparse similarity matrix

Use radial basis kernel in order to calculate similarities between data points

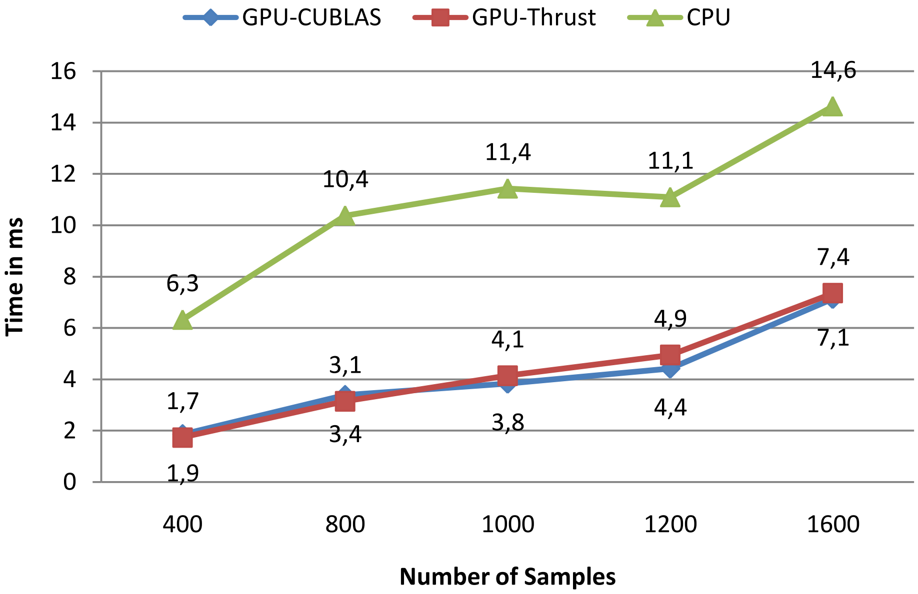
$$A_{ij} = e^{-\|s_i - s_j\|/2\sigma_i\sigma_j} \qquad i \neq j$$

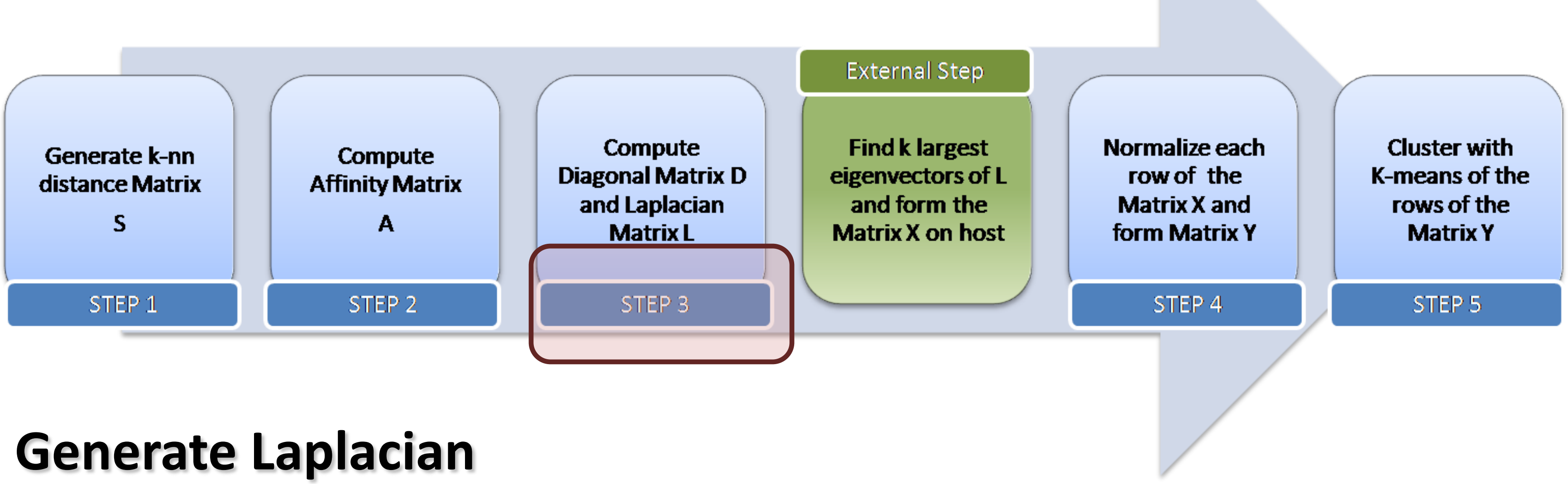
$$A_{ij} = 0 \qquad i = j$$

local scaling parameter σ_i calculated as the mean of row *i* in distance matrix *S*

Sum of each element in a row on GPU

1. sequential reduction using Thrust
2. Matrix-vector multiplication CUBLAS





Generate k-nn distance Matrix S

STEP 1

Compute Affinity Matrix A

STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L

STEP 3

External Step

Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y

STEP 4

Cluster with K-means of the rows of the Matrix Y

STEP 5

Generate Laplacian

Compute diagonal matrix D as the **row sum** of Affinity Matrix A

Using CUBLAS ***cublas_dgemv***

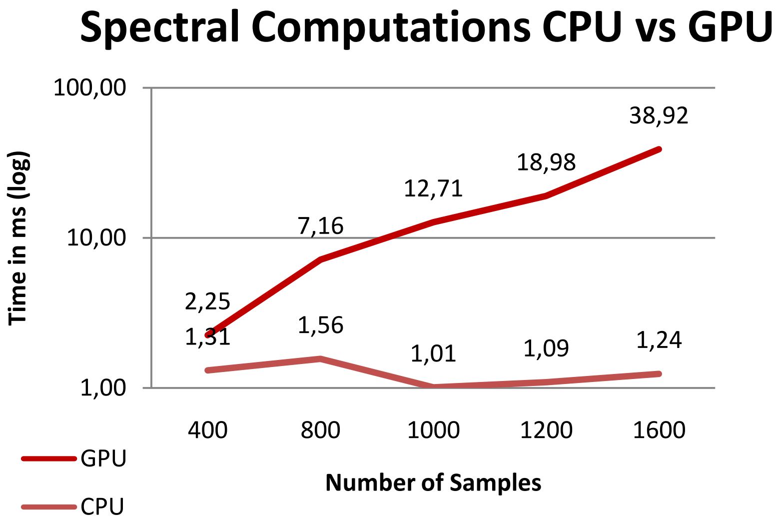
Multiply with Similarity matrix on left&right

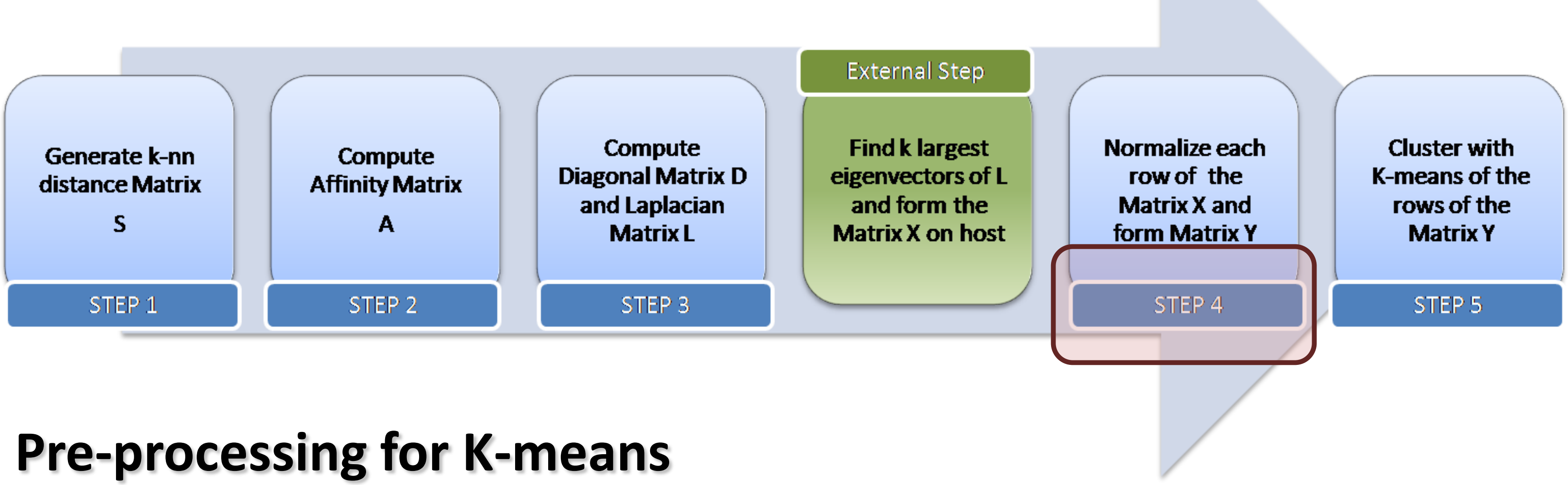
Using CUBLAS ***cublas_dgemm***

Launch 1 additional kernel to symmetrize

*CPU implementation outperforms GPU

Data transfer cost supresses the computation cost





Generate k-nn distance Matrix S

STEP 1

Compute Affinity Matrix A

STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L

STEP 3

External Step

Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y

STEP 4

Cluster with K-means of the rows of the Matrix Y

STEP 5

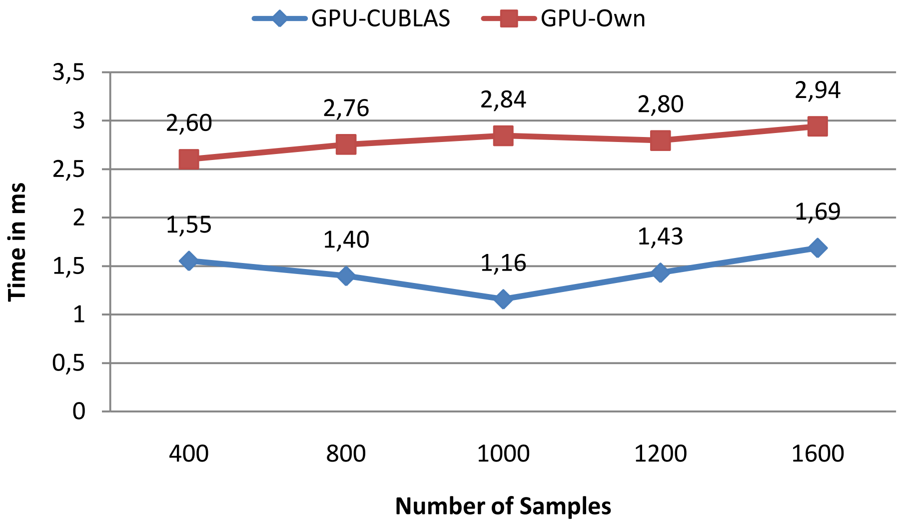
Pre-processing for K-means

Select k-largest Eigen values and their corresponding Eigen vectors form matrix X

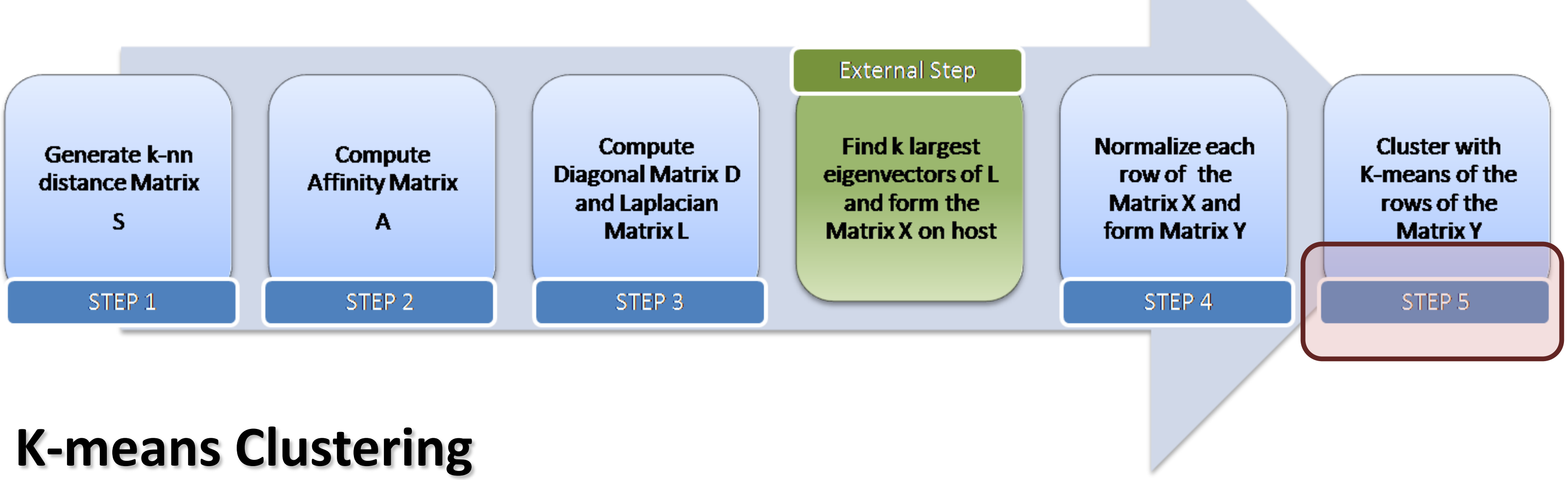
Using Thrust **thrust::sort_by_key**

Normalize rows of matrix X

1. Using CUBLAS ***cublas_Sgemm***
2. Own implementation with kernels



*This step is a minor step and can be considered as a preprocessing step of K-means step.



Generate k-nn distance Matrix S

STEP 1

Compute Affinity Matrix A

STEP 2

Compute Diagonal Matrix D and Laplacian Matrix L

STEP 3

External Step

Find k largest eigenvectors of L and form the Matrix X on host

Normalize each row of the Matrix X and form Matrix Y

STEP 4

Cluster with K-means of the rows of the Matrix Y

STEP 5

K-means Clustering

Cluster rows of matrix Y with k-means

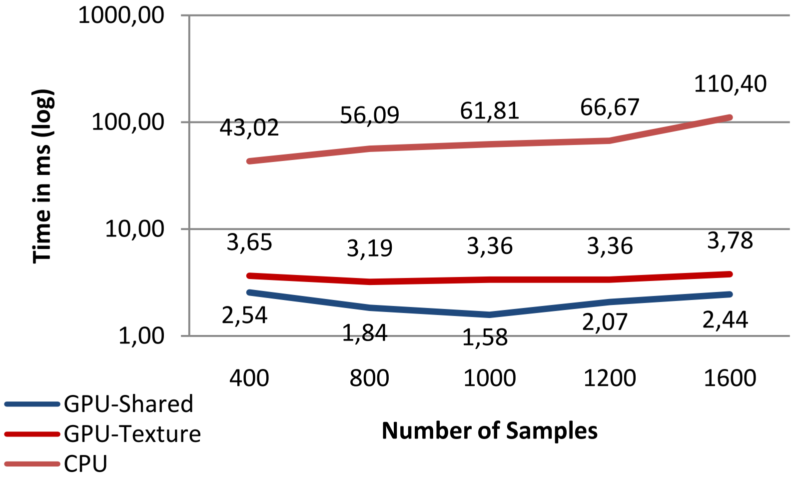
Two different utilization schemas

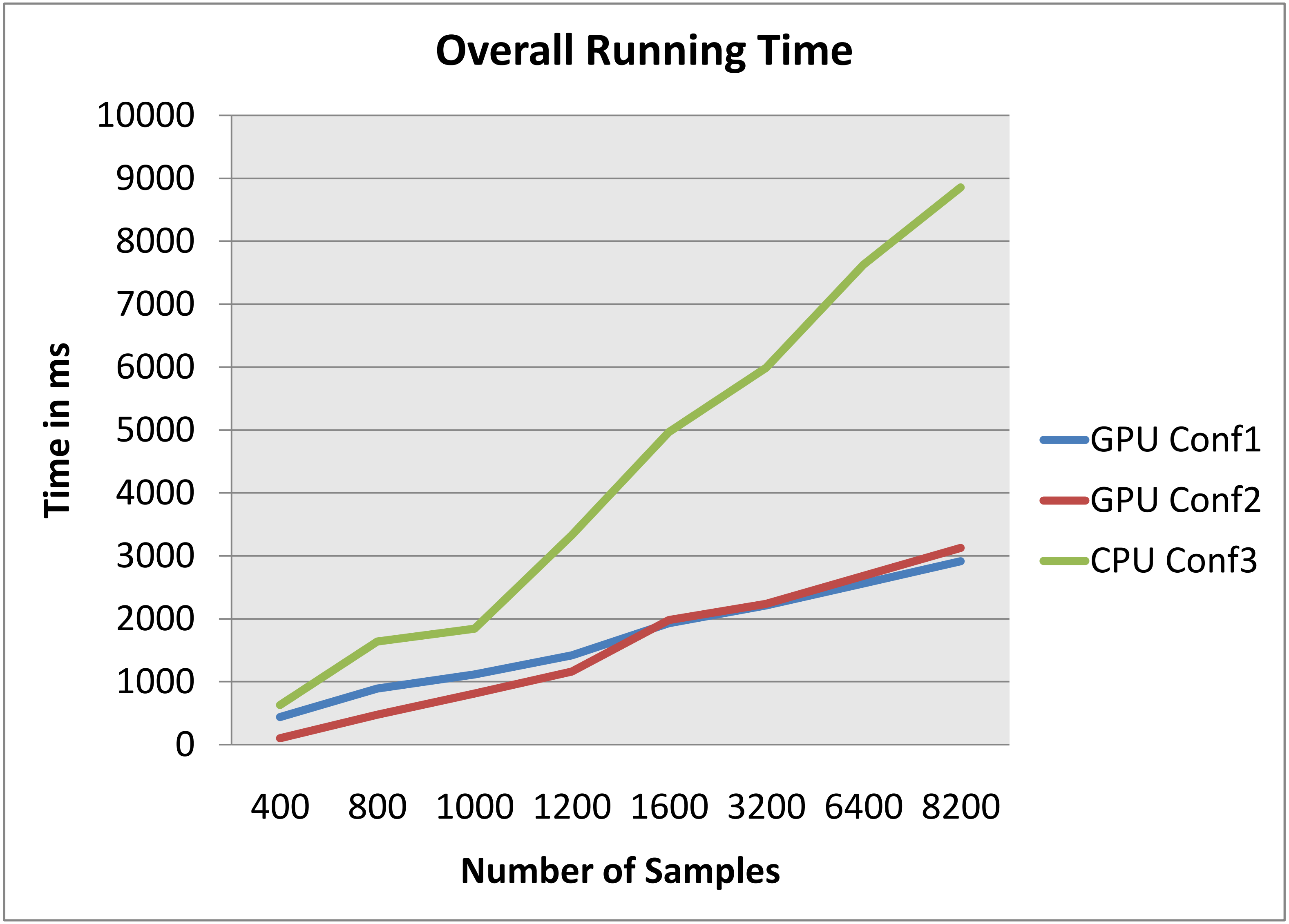
1. Shared memory reduction with texture
2. Dynamic block shared memory

Both have pros. and cons.

1. Suffers low shared memory usage when number of cluster is low
2. Initiates consequtive loops to fill shared memory when number of cluster is high

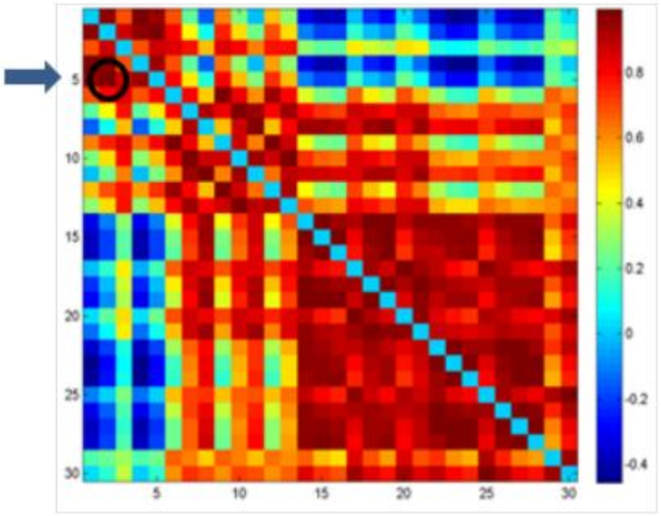
K-means on CPU vs GPU

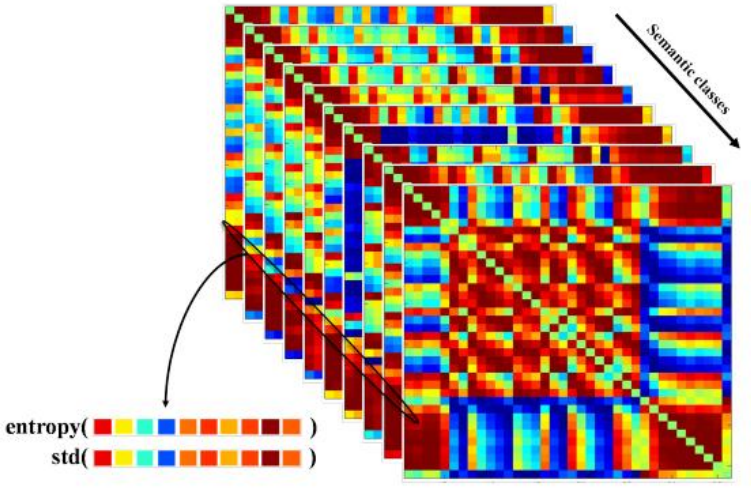


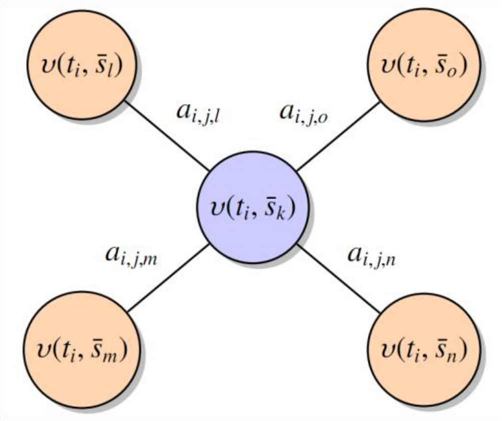

Overall Running Time
Time in ms
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
400
800
1000
1200
1600
3200
6400
8200
Number of Samples
GPU Conf1
GPU Conf2
CPU Conf3

What we “actually” do?

What we “actually” do?

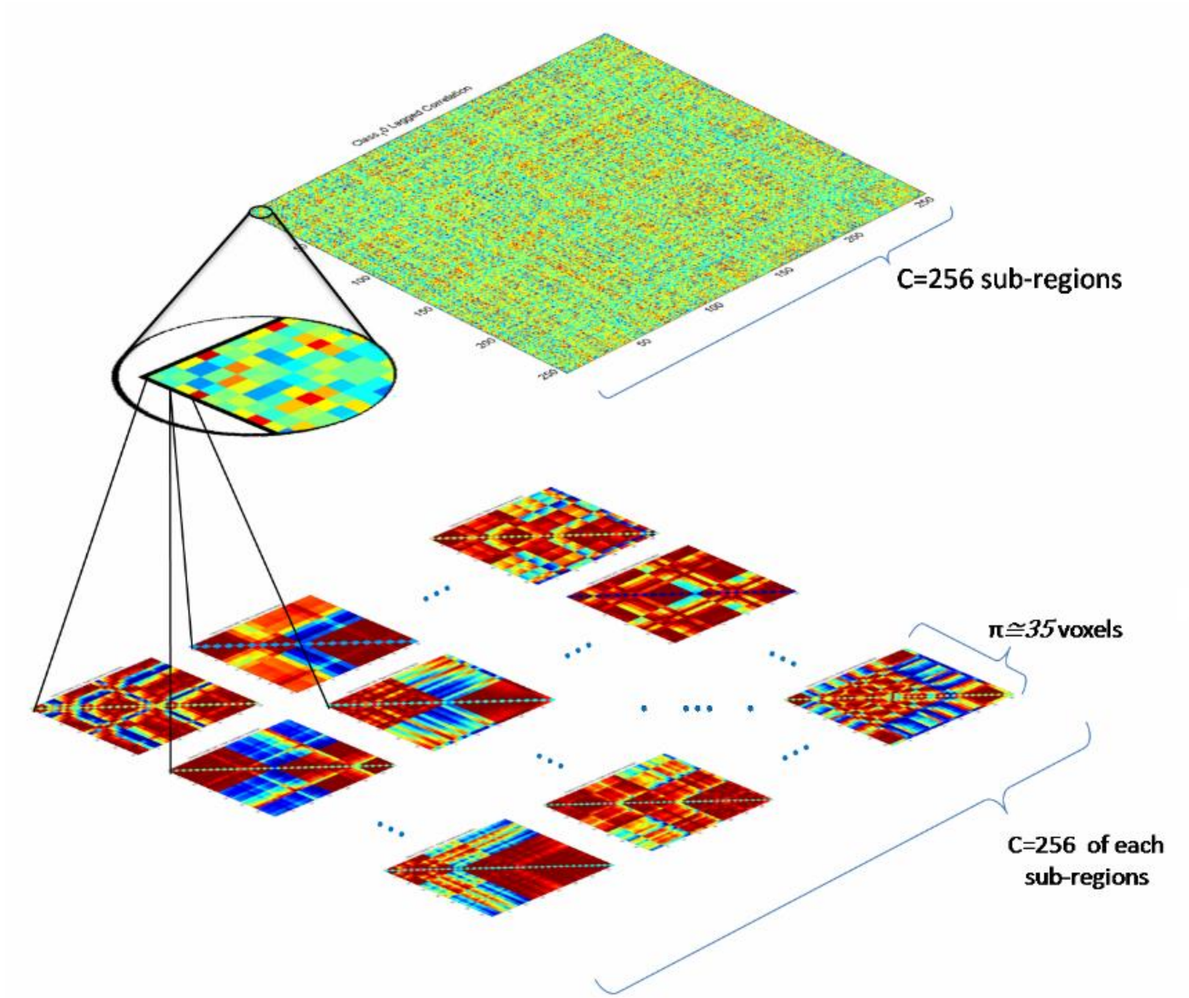






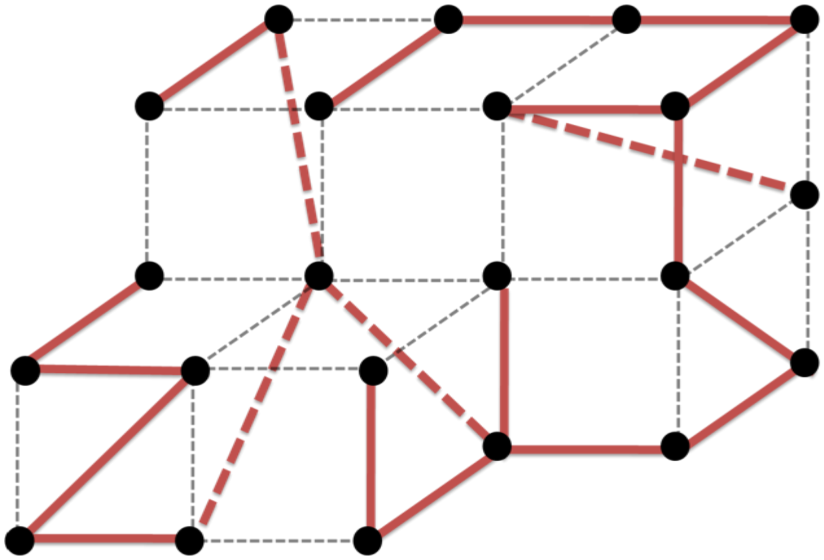
O. Firat, M. Ozay, I. Onal, I. Oztekin, F. T. Yarman Vural, "Functional Mesh Learning for Pattern Analysis of Cognitive Processes",12th IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC), 2013

What we “actually” do?



- O. Firat, A. Temizel ”Parallel Spectral Graph Partitioning on CUDA” GPU Technology Conference, San Jose, California, 2012
- O. Firat, M. Ozay, I. Onal, I. Oztekin, F. T. Yarman Vural, "Functional Mesh Learning for Pattern Analysis of Cognitive Processes",12th IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC), 2013

What we “actually” do?



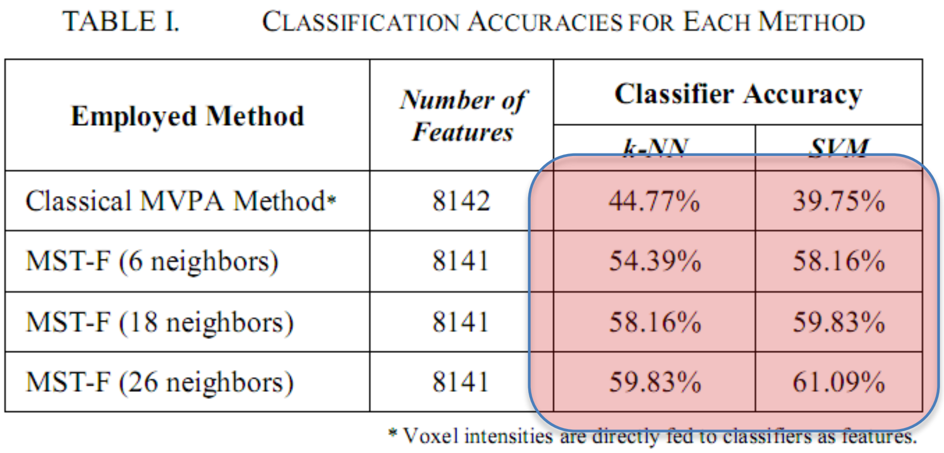


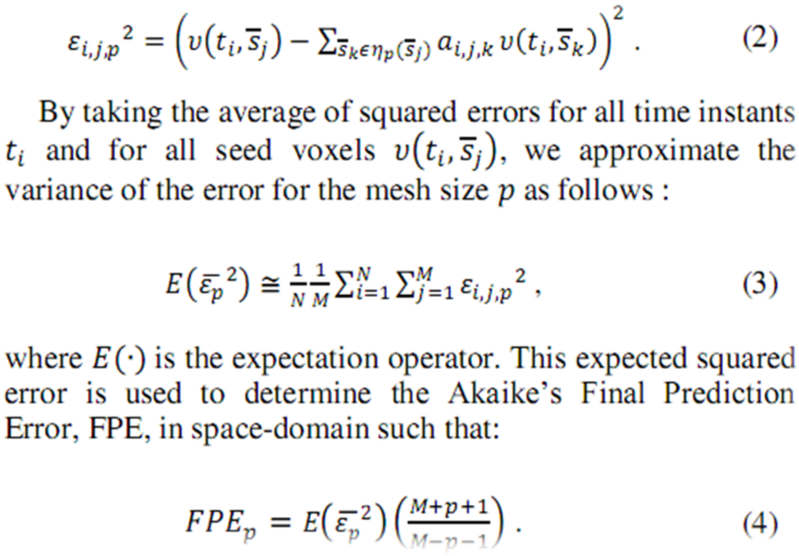
TABLE I. CLASSIFICATION ACCURACIES FOR EACH METHOD

Employed Method	***Number of Features***	**Classifier Accuracy**	
		k-NN	*SVM*
Classical MVPA Method*	8142	44.77%	39.75%
MST-F (6 neighbors)	8141	54.39%	58.16%
MST-F (18 neighbors)	8141	58.16%	59.83%
MST-F (26 neighbors)	8141	59.83%	61.09%

* Voxel intensities are directly fed to classifiers as features.

- O. Firat, M. Ozay, I. Onal, I. Oztekin, F. T. Yarman Vural, "Representation of Cognitive Processes Using the Minimum Spanning Tree of Local Meshes",35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS), 2013.

What we “actually” do?



$$\varepsilon_{i,j,p}{}^2 = \left(v(t_i,\bar{s}_j) - \Sigma_{\bar{s}_k \epsilon \eta_p(\bar{s}_j)}\, a_{i,j,k}\, v(t_i,\bar{s}_k)\right)^2 . \qquad (2)$$

By taking the average of squared errors for all time instants t_i and for all seed voxels $v(t_i,\bar{s}_j)$, we approximate the variance of the error for the mesh size p as follows :

$$E(\bar{\varepsilon}_p{}^2) \cong \frac{1}{N}\frac{1}{M}\Sigma_{i=1}^{N}\,\Sigma_{j=1}^{M}\,\varepsilon_{i,j,p}{}^2 , \qquad (3)$$

where $E(\cdot)$ is the expectation operator. This expected squared error is used to determine the Akaike’s Final Prediction Error, FPE, in space-domain such that:

$$FPE_p = E(\bar{\varepsilon}_p{}^2)\left(\frac{M+p+1}{M-p-1}\right) . \qquad (4)$$

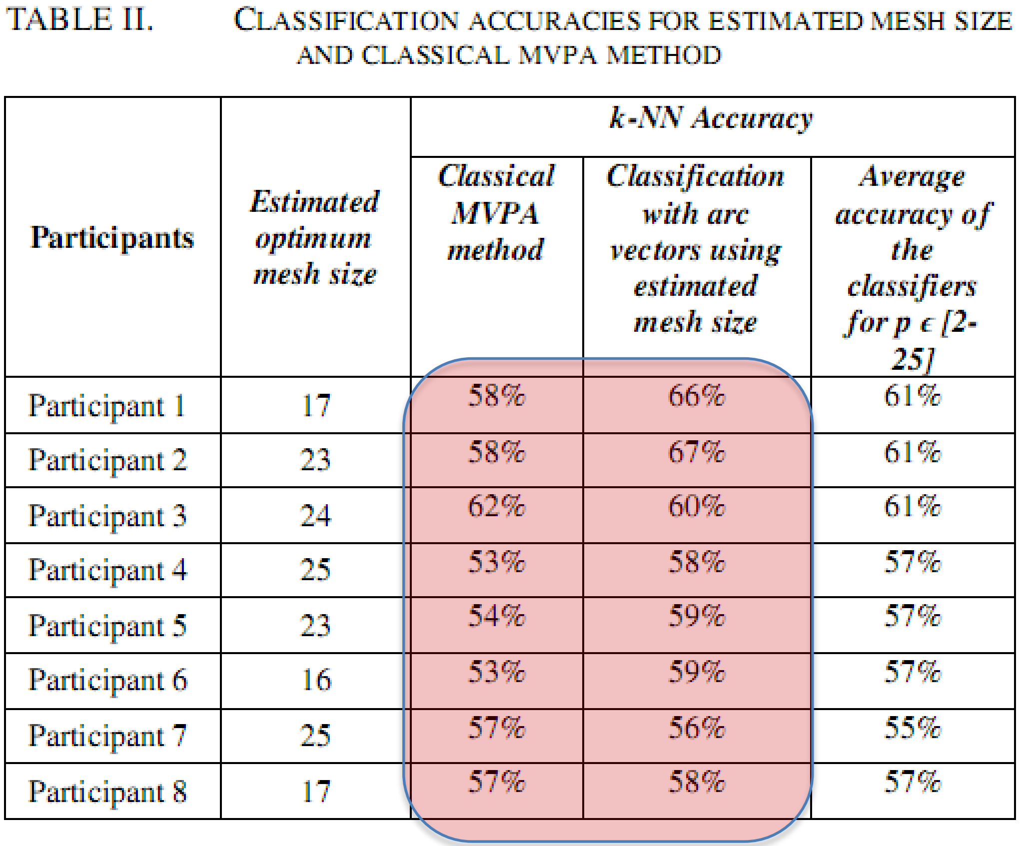


TABLE II. CLASSIFICATION ACCURACIES FOR ESTIMATED MESH SIZE AND CLASSICAL MVPA METHOD

Participants	*Estimated optimum mesh size*	*k-NN Accuracy*		
		Classical MVPA method	*Classification with arc vectors using estimated mesh size*	*Average accuracy of the classifiers for p ϵ [2-25]*
Participant 1	17	58%	66%	61%
Participant 2	23	58%	67%	61%
Participant 3	24	62%	60%	61%
Participant 4	25	53%	58%	57%
Participant 5	23	54%	59%	57%
Participant 6	16	53%	59%	57%
Participant 7	25	57%	56%	55%
Participant 8	17	57%	58%	57%

I. Onal, M. Ozay, O. Firat, I. Oztekin, F. T. Yarman Vural, "Analyzing the Information Distribution in the fMRI measurements by estimating the degree of locality",35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS), 2013.

What we “actually” do?

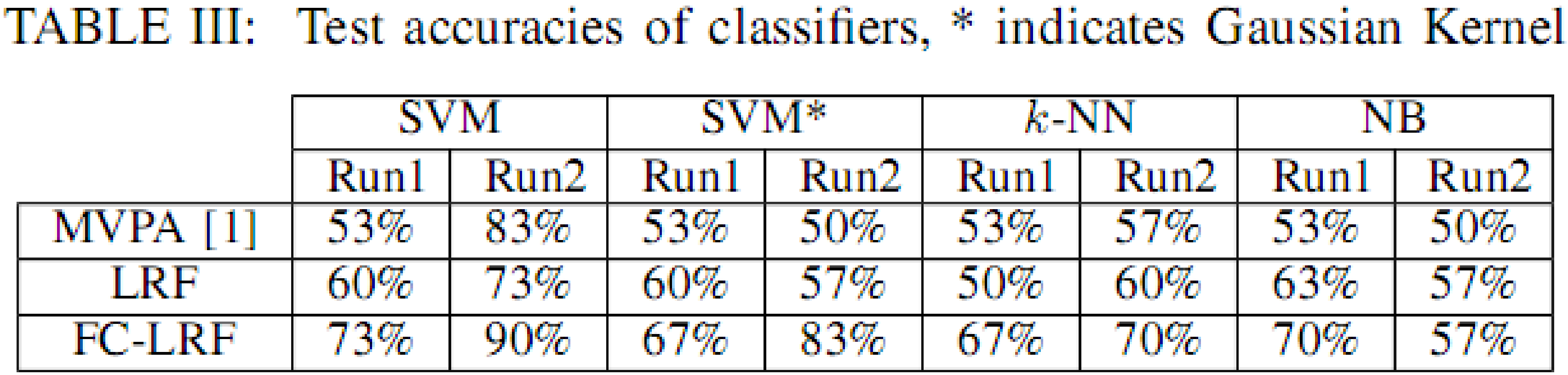


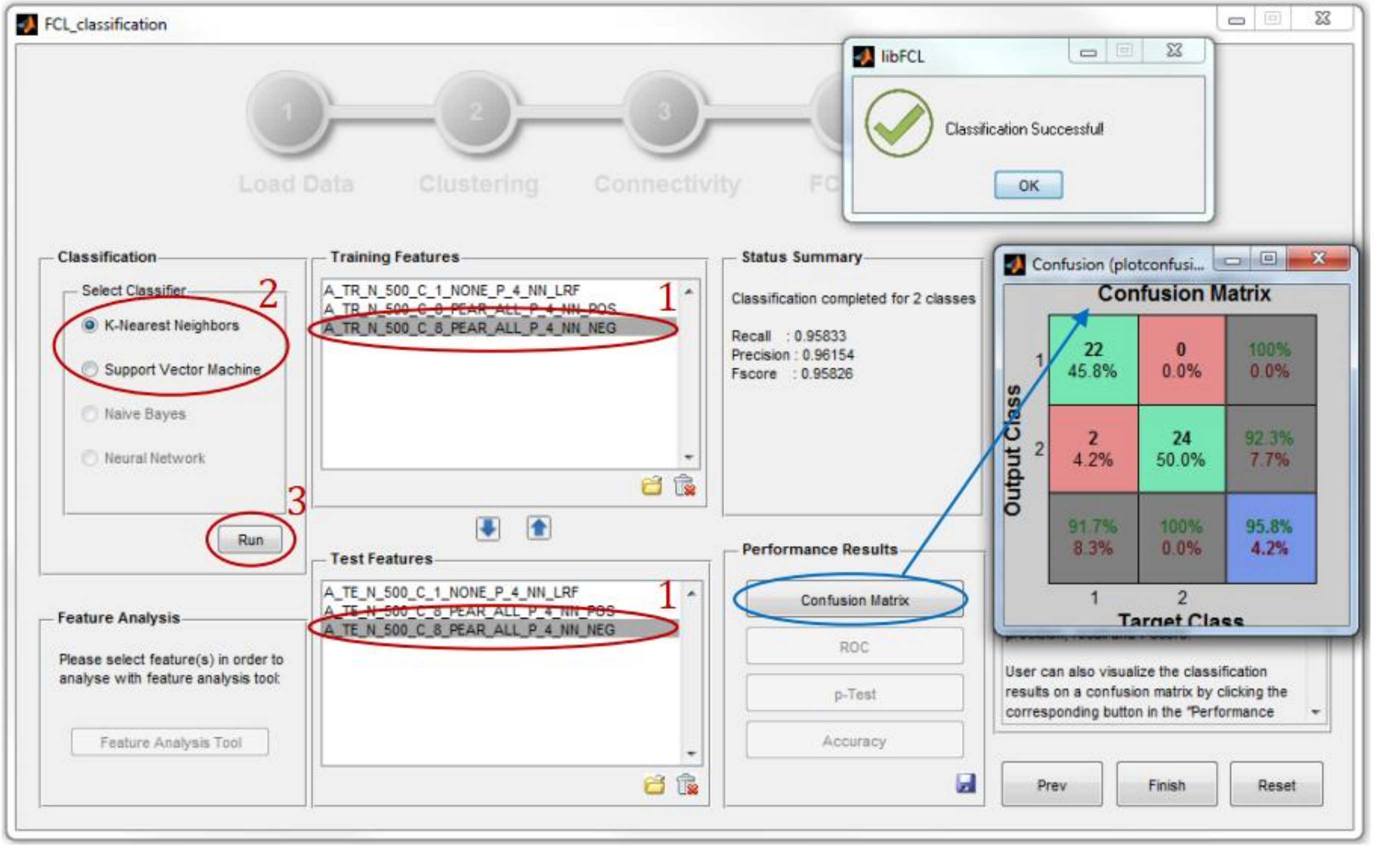
TABLE III: Test accuracies of classifiers, * indicates Gaussian Kernel

	SVM		SVM*		k-NN		NB	
	Run1	Run2	Run1	Run2	Run1	Run2	Run1	Run2
MVPA [1]	53%	83%	53%	50%	53%	57%	53%	50%
LRF	60%	73%	60%	57%	50%	60%	63%	57%
FC-LRF	73%	90%	67%	83%	67%	70%	70%	57%

O. Ekmekci, O. Firat, M. Ozay, I. Oztekin, F. T. Yarman Vural, U. Oztekin "Mesh Learning for Object Classification using fMRI Measurements",12th IEEE International Conference on Image Processing (ICIP), 2013.

Where we come in handy?

- *libFCL* GUI
- Scalable Functional Connectivity Tools



Classification Successfull

● K-Nearest Neighbors
○ Support Vector Machine
○ Naive Bayes
○ Neural Network

A_TR_N_500_C_1_NONE_P_4_NN_LRF
A_TR_N_500_C_8_PEAR_ALL_P_4_NN_POS
A_TR_N_500_C_8_PEAR_ALL_P_4_NN_NEG

A_TE_N_500_C_1_NONE_P_4_NN_LRF
A_TE_N_500_C_8_PEAR_ALL_P_4_NN_POS
A_TE_N_500_C_8_PEAR_ALL_P_4_NN_NEG

Classification completed for 2 classes

Recall : 0.95833
Precision : 0.96154
Fscore : 0.95826

Please select feature(s) in order to analyse with feature analysis tool:

Output Class \ Target Class	1	2	
1	22 45.8%	0 0.0%	100% 0.0%
2	2 4.2%	24 50.0%	92.3% 7.7%
	91.7% 8.3%	100% 0.0%	95.8% 4.2%

User can also visualize the classification results on a confusion matrix by clicking the corresponding button in the "Performance neuro.ceng.metu.edu.tr

orhan.firat@ceng.metu.edu.tr