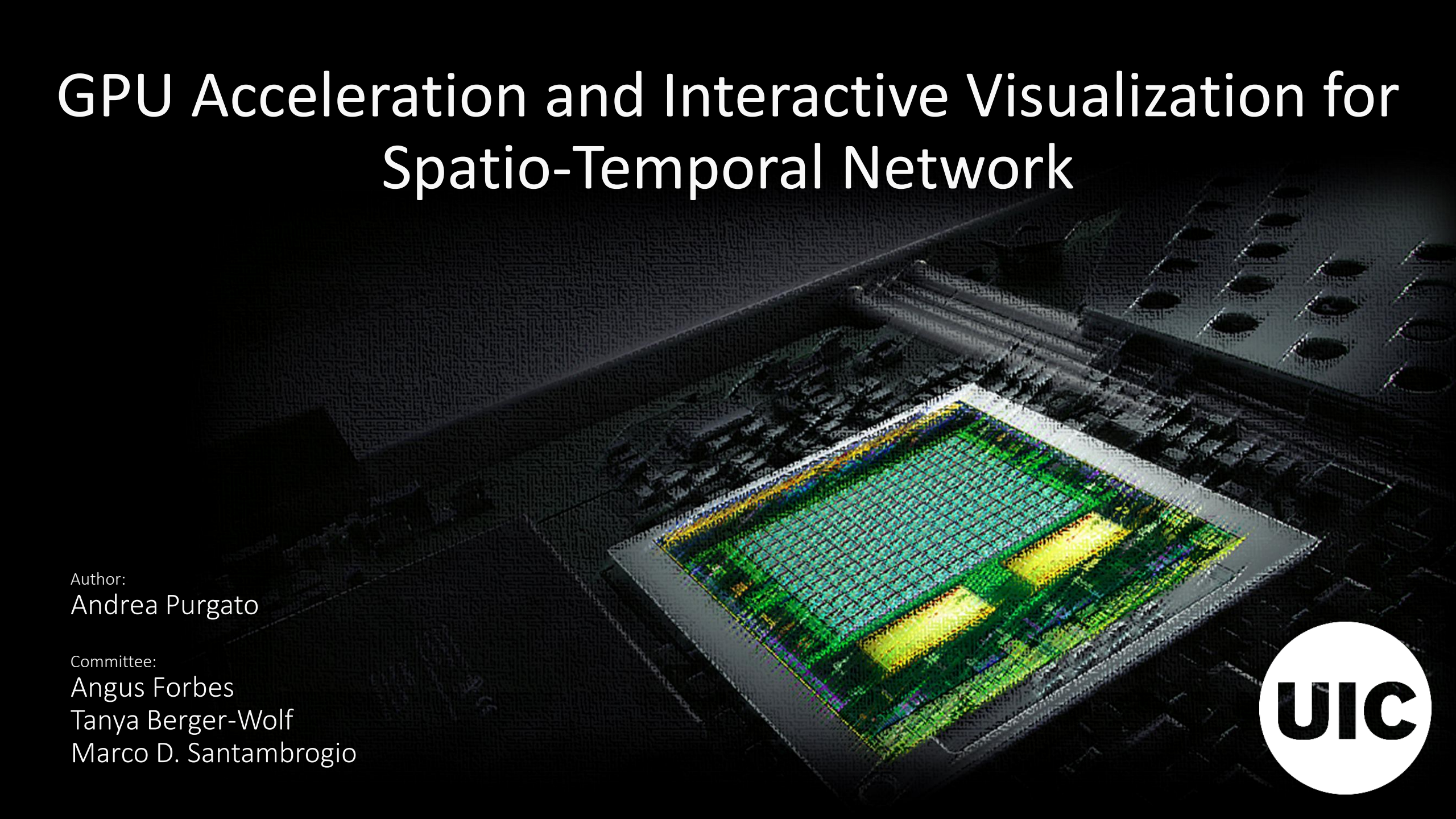
GPU Acceleration and Interactive Visualization for Spatio-Temporal Network



Author:
Andrea Purgato

Committee:
Angus Forbes
Tanya Berger-Wolf
Marco D. Santambrogio

UIC

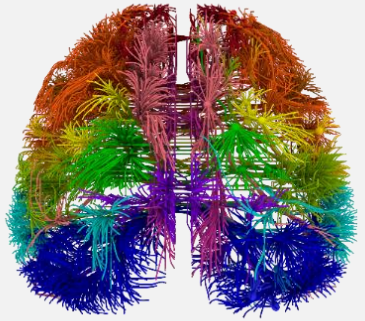
Objectives

Spatio-Temporal Networks computed on GPU.

- On CPUs requires time.
- Exploit GPU high computational power.
- Leaving one degree of freedom.

Spatio-Temporal Networks interactive visualization.

- Thanks to GPU computation results.
- Results exploration.
- Focused on Brain Networks.



Outline

1. **Spatio-Temporal Network**
2. **GPU Programming Model**
3. **Similarity Computation**
4. **Experimental Evaluation**
5. **Brain Network Visualization**
6. **Final Considerations**

Outline

1. **Spatio-Temporal Network**
2. **GPU Programming Model**
3. **Similarity Computation**
4. **Experimental Evaluation**
5. **Brain Network Visualization**
6. **Final Considerations**

Spatio-Temporal Network (1/5)

- Spatio-Temporal Network represents an evolution of usual graphs.
- Introduces the time component.
- Network structure changes during the time.
- Used to represent dynamic interactions among entities.
- Increased the graphs complexity.
- Applied to many different contexts. [1]

[1] Holme, P.: Modern temporal network theory: a colloquium.

Spatio-Temporal Network (2/5)

A simple **Graph** is defined as:

$$\boldsymbol{G} \coloneqq (\boldsymbol{V}, \boldsymbol{E})$$

- $\boldsymbol{V}$: set of vertexes.
- $\boldsymbol{E} \subseteq \boldsymbol{V} \times \boldsymbol{V}$: set of edges.

Spatio-Temporal Networks could be represented by a sequence of **Graphs**.

$$\boldsymbol{STN} \coloneqq (\boldsymbol{G_1}, \boldsymbol{G_2}, \dots, \boldsymbol{G_N})$$

- $\boldsymbol{N}$: number of time instant in the observation period.
- $\boldsymbol{G_i} \coloneqq (\boldsymbol{V_i}, \boldsymbol{E_i}) : \boldsymbol{V_i} \subseteq \boldsymbol{\Gamma}$, where $\boldsymbol{\Gamma}$ is the set of all the entities observed.

Spatio-Temporal Network (3/5)

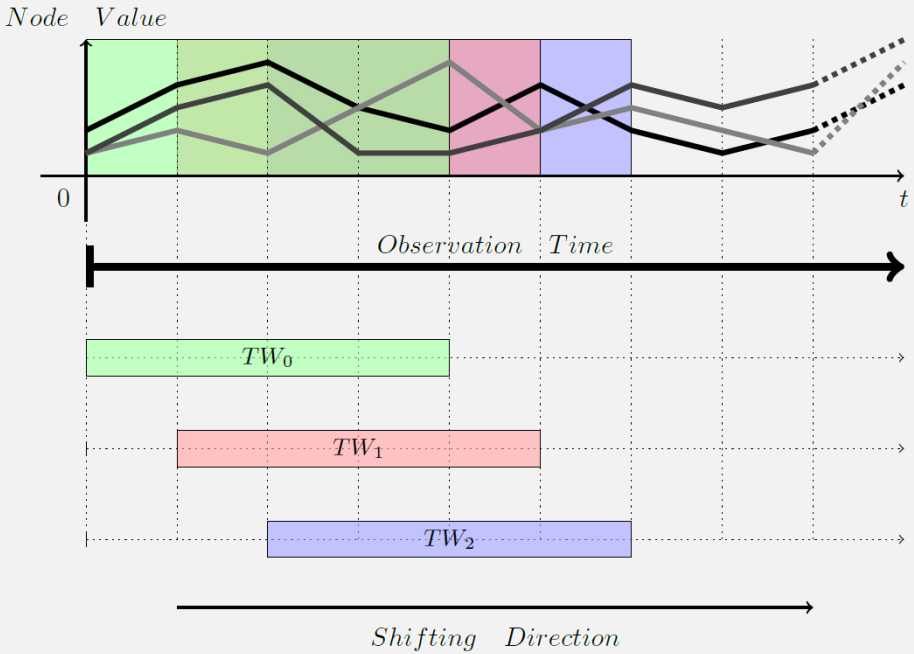
An edge $\boldsymbol{e_{t,i,j}}$ between node $\boldsymbol{i}$ and node $\boldsymbol{j}$ is defined for graph $\boldsymbol{G_t}$ if the similarity at time $\boldsymbol{t}$ computed between the nodes is greater than a fixed similarity threshold.

The definition of the edges depends on a **Similarity** measure, it is:

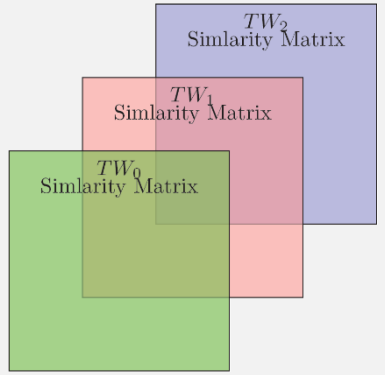
- Computed for each pairs of nodes.
- Computed for each observation instant.
- Computed using the values measured during the observation of the entities involved.
- Computed using a **subset** of values

Spatio-Temporal Network (4/5)

- The subset of values defines the concept of: **Time Window**.
- The **size** of the time window represent how far you look for interactions among nodes.

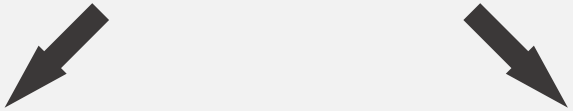


Similarity Computation



Spatio-Temporal Network (5/5)

The Spatio-Temporal Network definition has two degrees of freedom.



Similarity Threshold

Time Window Size

Outline

1. Spatio-Temporal Network
2. **GPU Programming Model**
3. Similarity Computation
4. Experimental Evaluation
5. Brain Network Visualization
6. Final Considerations

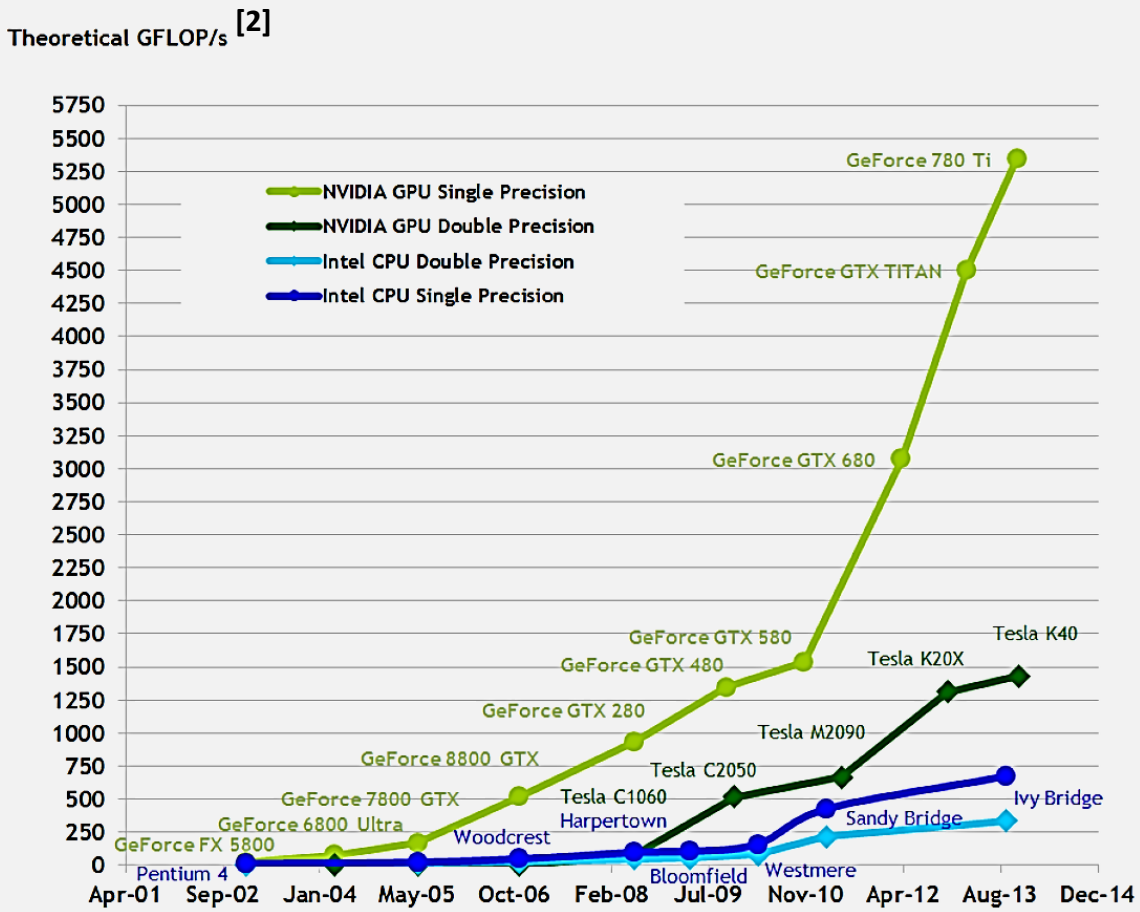
GPU Programming Model (1/4)

GPU computational power increased exponentially in the last years.

New trend in computational world:

- Use hardware devices to speed up computation.
- Use the new programming model imposed by these devices.

For this work we used **CUDA Library** [1].



[1] https://developer.nvidia.com/cuda-zone
[2] Sanders, J. and Kandrot, E.: CUDA by example: an introduction to general-purpose GPU programming.

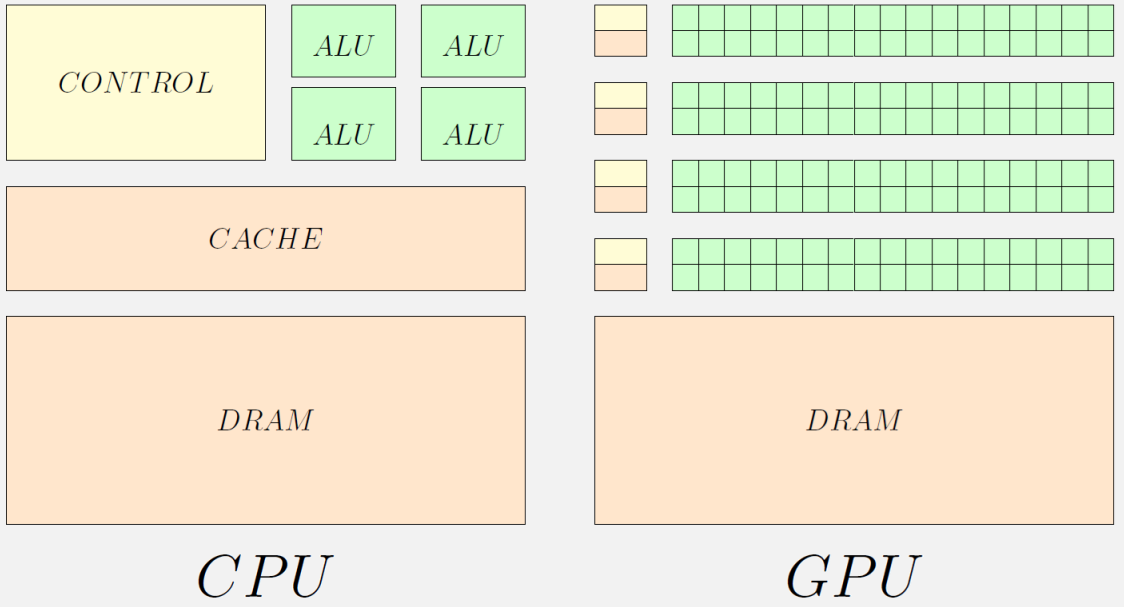
GPU Programming Model (2/4)

GPU architectures generally present:

- High number of execution units.
- Ability to execute a high number of threads simultaneously.
- Very low thread switching cost in the execution unit.

GPU programming model respect SIMT paradigm:

- Execution of the same function by different threads.
- Each thread work on a different piece of data.



GPU Programming Model (3/4)

Architectures that support CUDA programming presents two main components:

- Global Memory (GM)
- Streaming Multiprocessor (SM)

Streaming Multiprocessor:

- Collection of execution units.
- Independent each other.
- Execute threads in bunch, called **Warps**.

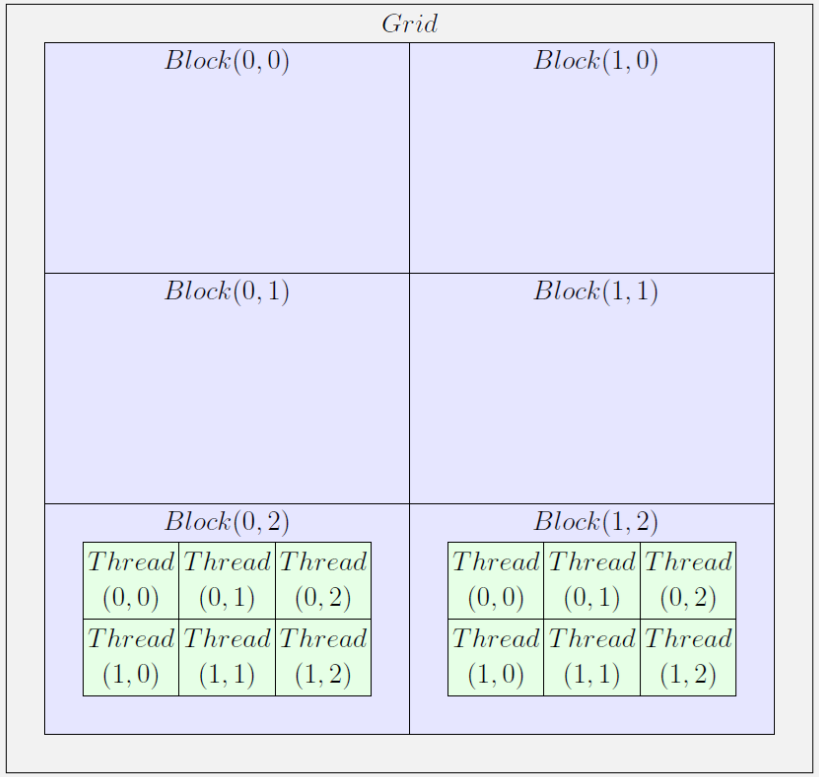
Function executed by a GPU thread called: **Kernel Function**.

- Executed after a GPU call performed by the CPU.
- Each call must have associated a **GPU Structure**.

GPU Programming Model (4/4)

A GPU Structure is composed by:

- One **Grid**: a matrix of **Blocks**
- **Blocks**: matrices of **Threads**.



GPU Structure characteristics:

- Each matrix can have at most 3 dimensions.
- The dimensions are decided at run time.
- The dimensions depends on the resources available.
- Threads can share memory only if in the same block.

Each thread in the grid executes the same Kernel Function

Outline

1. Spatio-Temporal Network
2. GPU Programming Model
3. **Similarity Computation**
4. Experimental Evaluation
5. Brain Network Visualization
6. Final Considerations

Similarity Computation

We adopted as similarity measure the **Pearson Correlation Coefficient:**

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \cdot \sigma_Y}$$

Input parameter:

- **Time window size**

Computation divided in six phases:

1. **Device Reading**
2. **Data Loading**
3. **Computation Balancing**
4. **Covariance Computation** (on GPU)
5. **Correlation Computation** (on GPU)
6. **Results Saving**

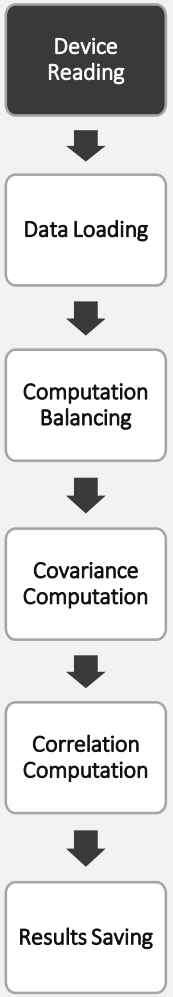
NVIDIA
CUDA
C/C++

1. Device Reading

Query the machine to get the devices characteristics:

- GUP memory size.
- Registers per block.
- Warp size.
- Threads per block.
- Block maximum dimensions.
- Grid maximum dimensions.

Used to balance the computation of the computation.



2. Data Loading

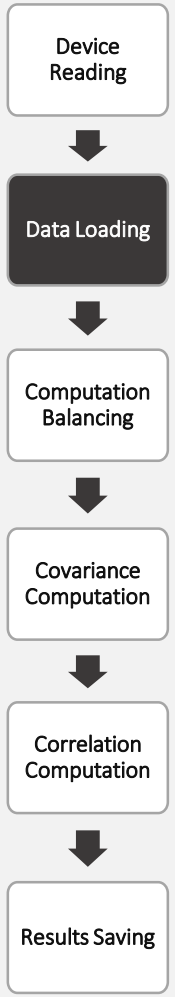
- The input data contains the values of each node in each time instant.
- Input data format with **N** time observation instants and **P** nodes:

$$
\begin{array}{llllll}
t_0, & v_{0,0}, & v_{0,1}, & \cdots & v_{0,P} \\
t_1, & v_{1,0}, & v_{1,1}, & \cdots & v_{1,P} \\
\vdots & & & & \\
t_N, & v_{N,0}, & v_{N,1}, & \cdots & v_{N,P}
\end{array}
$$

t_i: observation instant i

$v_{i,j}$: value of node j in observation instant i

- The input data are loaded in the memory of <u>each</u> GPU device.



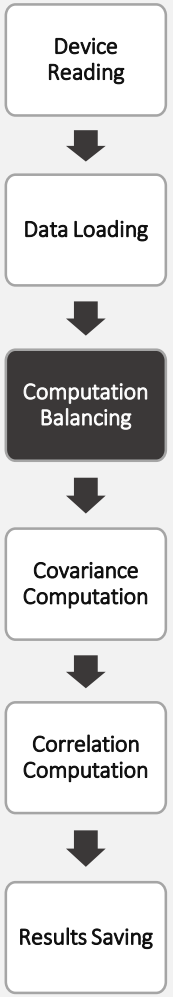
3. Computation Balancing (1/4)

The computation of Covariance and Correlation must be split due to the high input node numbers. In this phase are estimated the parameters that allow the computation balancing.

- Computation split over the time.
- Computation split over the devices.

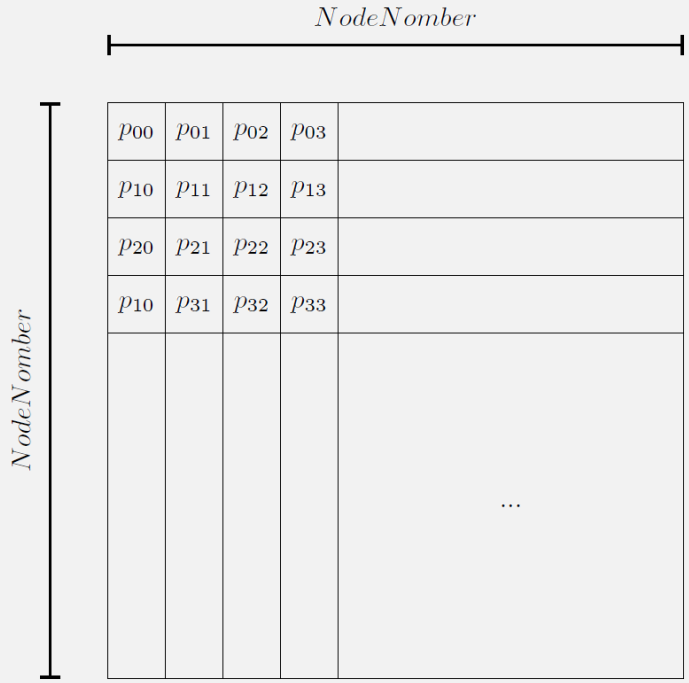
In this phase is computed:

- GPU structure for GPU functions.
- Number of steps for time split.



3. Computation Balancing (2/4)

The computation of Covariance and Correlation is pairwise.

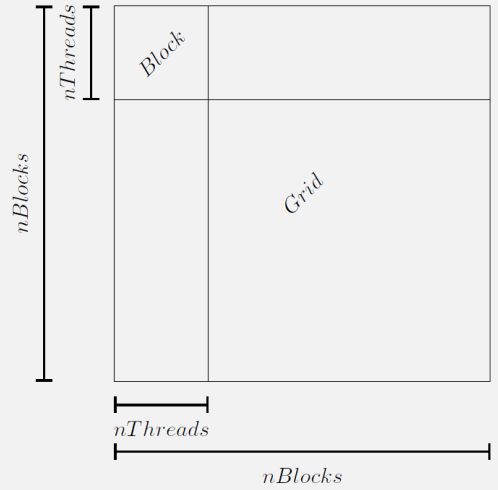


- Matrix that represent all the node pairs.
- $\boldsymbol{p_{i,j}}$: pair between node $\boldsymbol{i}$ and node $\boldsymbol{j}$.
- Each pair is associated to a thread in the GPU structure.
- Each thread computes the Covariance and the Correlation for the node pair associated.

3. Computation Balancing (3/4)

GPU structure definition:

- Number of threads per each dimension equal to node number in input data.
- Covariance and Correlation must be computed for each observation instant.

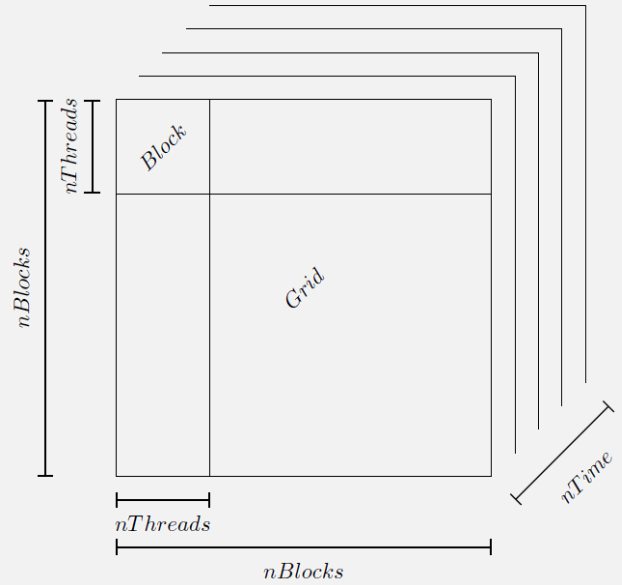


- The GPU structure is squared.
- X and Y block dimension are equal to $\boldsymbol{nThreads}$.
- $\boldsymbol{nBlocks * nThreads = NodeNumber}$

3. Computation Balancing (3/4)

GPU structure definition:

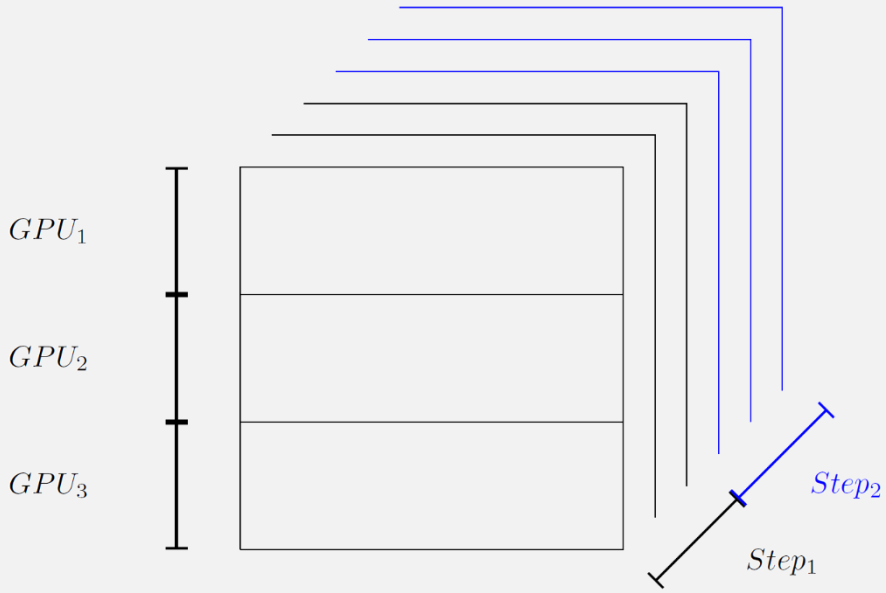
- Number of threads per each dimension equal to node number in input data.
- Covariance and Correlation must be computed for each observation instant.



- GPU Structure extended on *z-axis*.
- Each level of *z* represent one time instant.
- ***nTime***: number of observation instants.

3. Computation Balancing (4/4)

The resources limitation imposes a split of the GPU structure defined. It avoids to run out of resources.

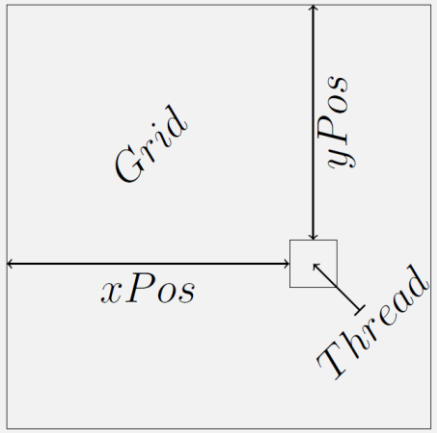


- Each step executed by a different GPU call.
- Each vertical division is executed on a different GPU.

4. Covariance Computation

$$Cov(X,Y) = E[XY] - E[X]E[Y]$$

- Computed on GPU.
- Each thread of the GPU structure computes the covariance between two nodes.
- The nodes numbers are given by the thread position in the structure.



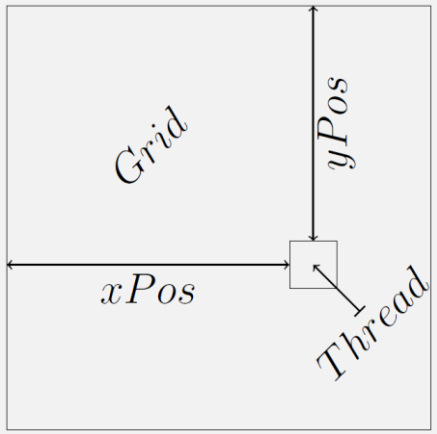
- $xPos$: first node number.
- $yPos$: second node number.

In case of covariance equal to zero a little noise introduced.

5. Correlation Computation

$$\rho_{X,Y} = \frac{Cov(X,Y)}{Cov(X,X) \cdot Cov(Y,Y)}$$

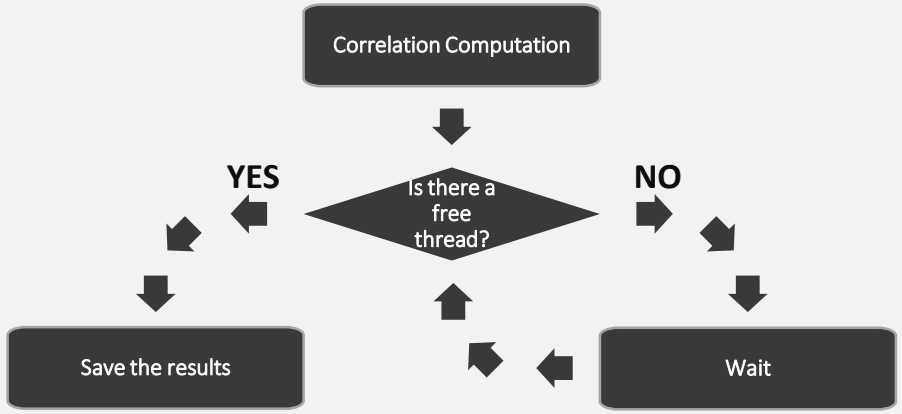
- Computed on GPU.
- Each thread of the GPU structure computes the correlation between two nodes.
- The nodes numbers are given by the thread position in the structure.

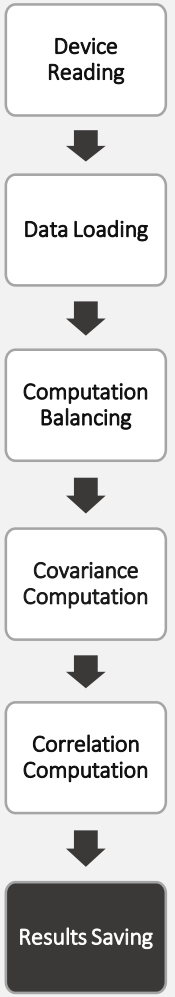


- $xPos$: first node number.
- $yPos$: second node number.

6. Results Saving

- Generated one file per each observation instant.
- Saved the whole similarity matrix.
- Saving process executed by a fixed number of independent threads.





Outline

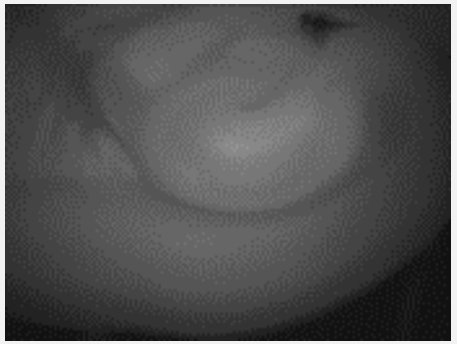
1. Spatio-Temporal Network
2. GPU Programming Model
3. Similarity Computation
4. **Experimental Evaluation**
5. Brain Network Visualization
6. Final Considerations

Experimental Evaluation (1/5)

We applied Similarity computation to brain network definition.

Input dataset:

- Taken from a video that records a brain slice activation.
- Composed by a series of images, one per each frame of the video.
- Image size: 172 x 130 pixels for a total of 22360
- Image number: 1000
- Each pixel of the images is a node in the input dataset.

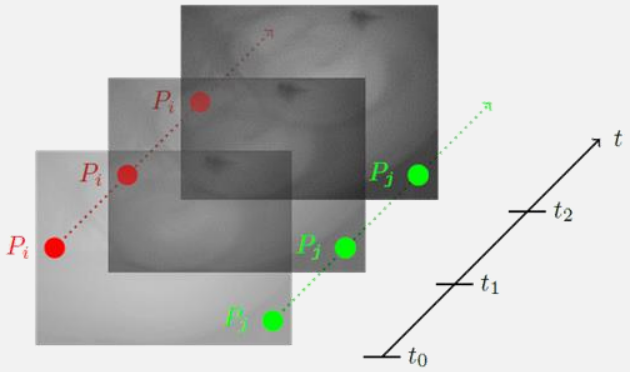


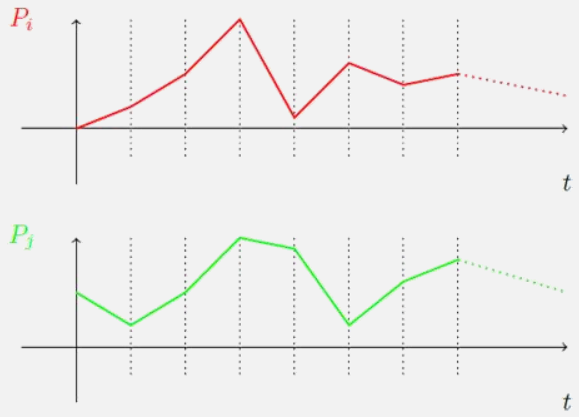
Sample image

Experimental Evaluation (2/5)

Each pixel is represented by a discrete function:

- For each observation instant is taken the pixel hue value.
- The discrete functions are transformed in the input file.





$$
\begin{array}{llllllll}
t_0, & v_{0,P_0}, & \ldots & v_{0,P_i}, & \ldots & v_{0,P_j}, & \ldots \\
t_1, & v_{1,P_0}, & \ldots & v_{1,P_i}, & \ldots & v_{1,P_j}, & \ldots \\
\ldots & & & & & & \\
\ldots & & & & & & \\
t_T, & v_{T,P_0}, & \ldots & v_{T,P_i}, & \ldots & v_{T,P_j}, & \ldots
\end{array}
$$

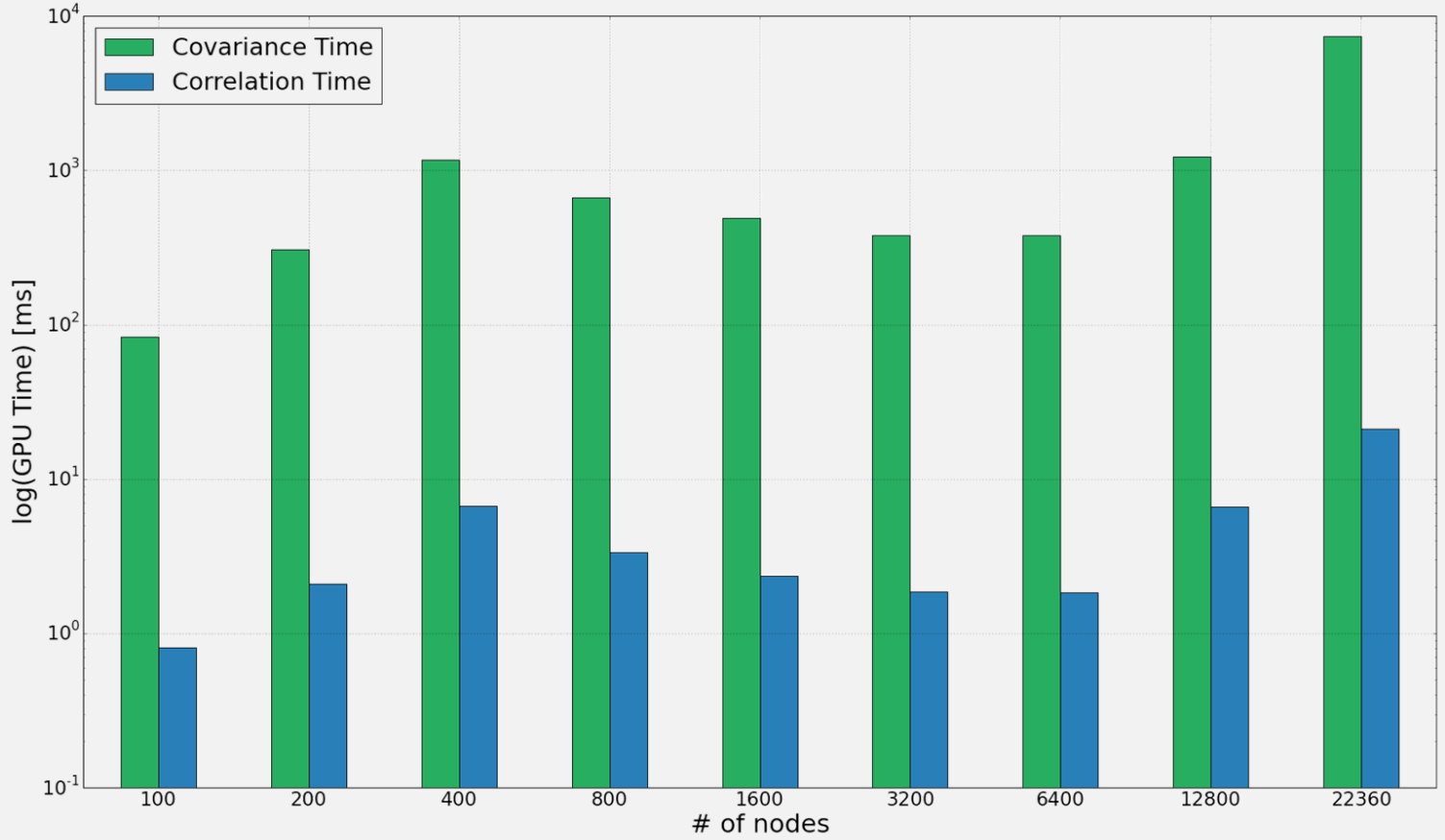
Experimental Evaluation (3/5)

Program execution time:

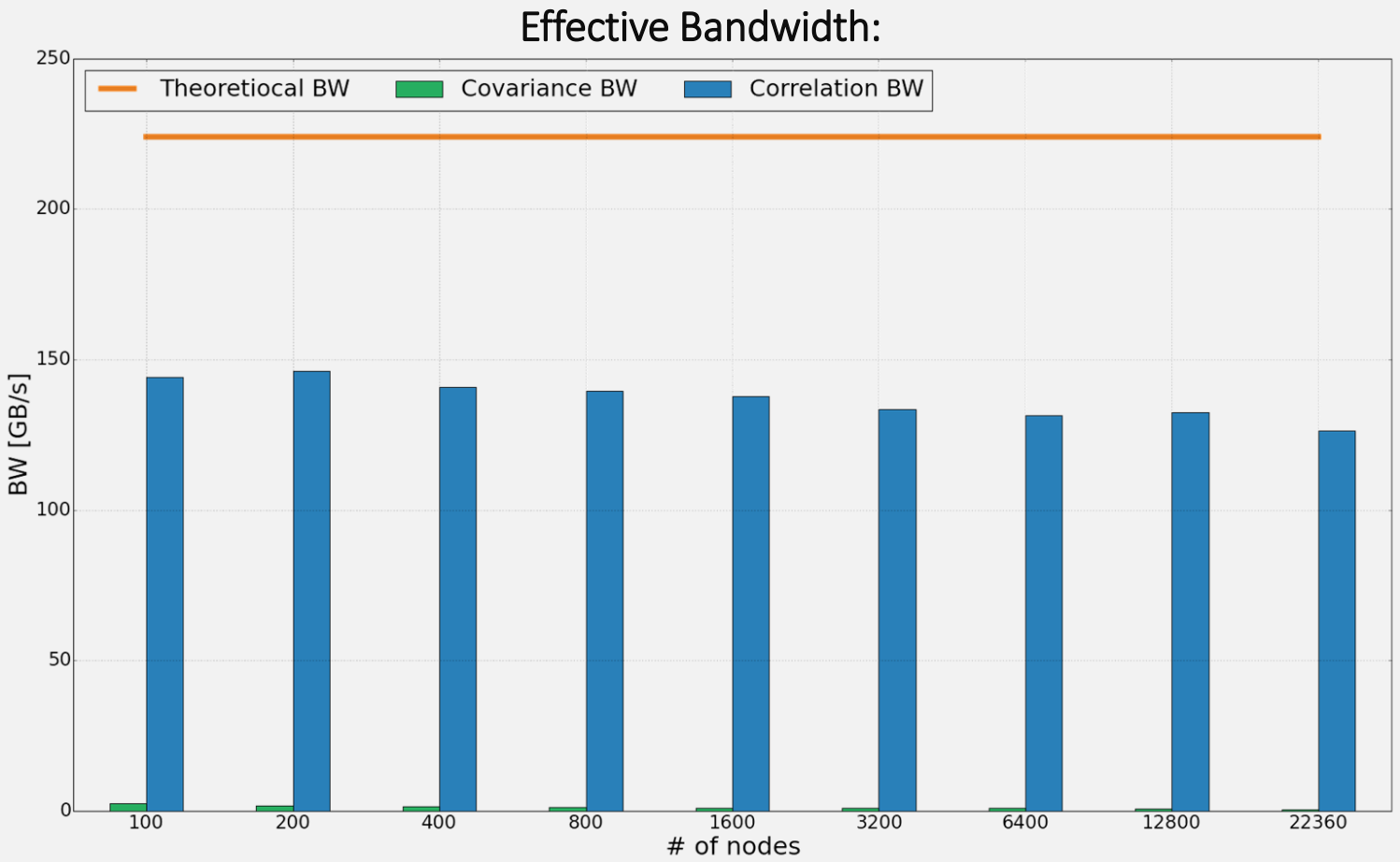
Nodes	Algorithm Time [s]	GPU Time [s]	Waiting Time [s]	GPU Percentage [%]	Waiting Percentage [%]
100	28,125	0,084	27,125	0,299	96,444
200	92,000	0,309	90,500	0,336	98,370
400	295,625	1,177	290,875	0,398	98,393
800	1149,125	4,709	1133,500	0,410	98,640
1600	4674,500	19,160	4618,625	0,410	98,805
3200	18493,000	76,296	18279,000	0,413	98,843
6400	72485,750	304,284	71616,750	0,420	98,801
12800	307466,000	1106,469	303913,000	0,360	98,844
22360	833987,000	5928,430	821084,000	0,711	98,453

Experimental Evaluation (4/5)

Covariance VS Correlation:



Experimental Evaluation (5/5)



$$BW_{Effective} = \frac{B_r + B_w}{10^6 * time}$$ [1]

- B_r: bytes # read from memory
- B_w: bytes # write on memory
- $time$: execution time of the GPU function.

[1] https://devblogs.nvidia.com/parallelforall/how-implement-performance-metrics-cuda-cc/

Outline

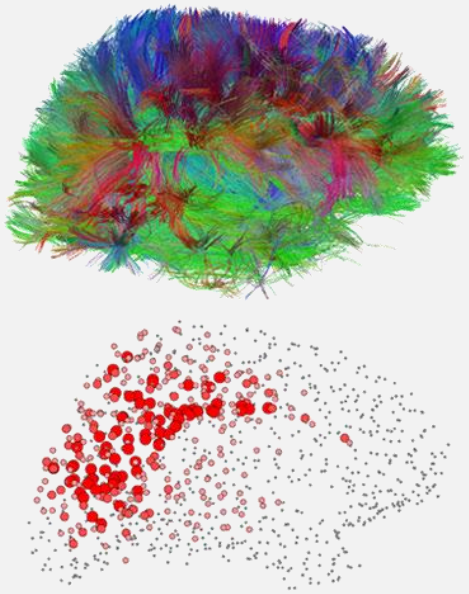
1. Spatio-Temporal Network
2. GPU Programming Model
3. Similarity Computation
4. Experimental Evaluation
5. **Brain Network Visualization**
6. Final Considerations

Brain Network Visualization (1/6)

Focused on Brain Networks, computed with the GPU program.

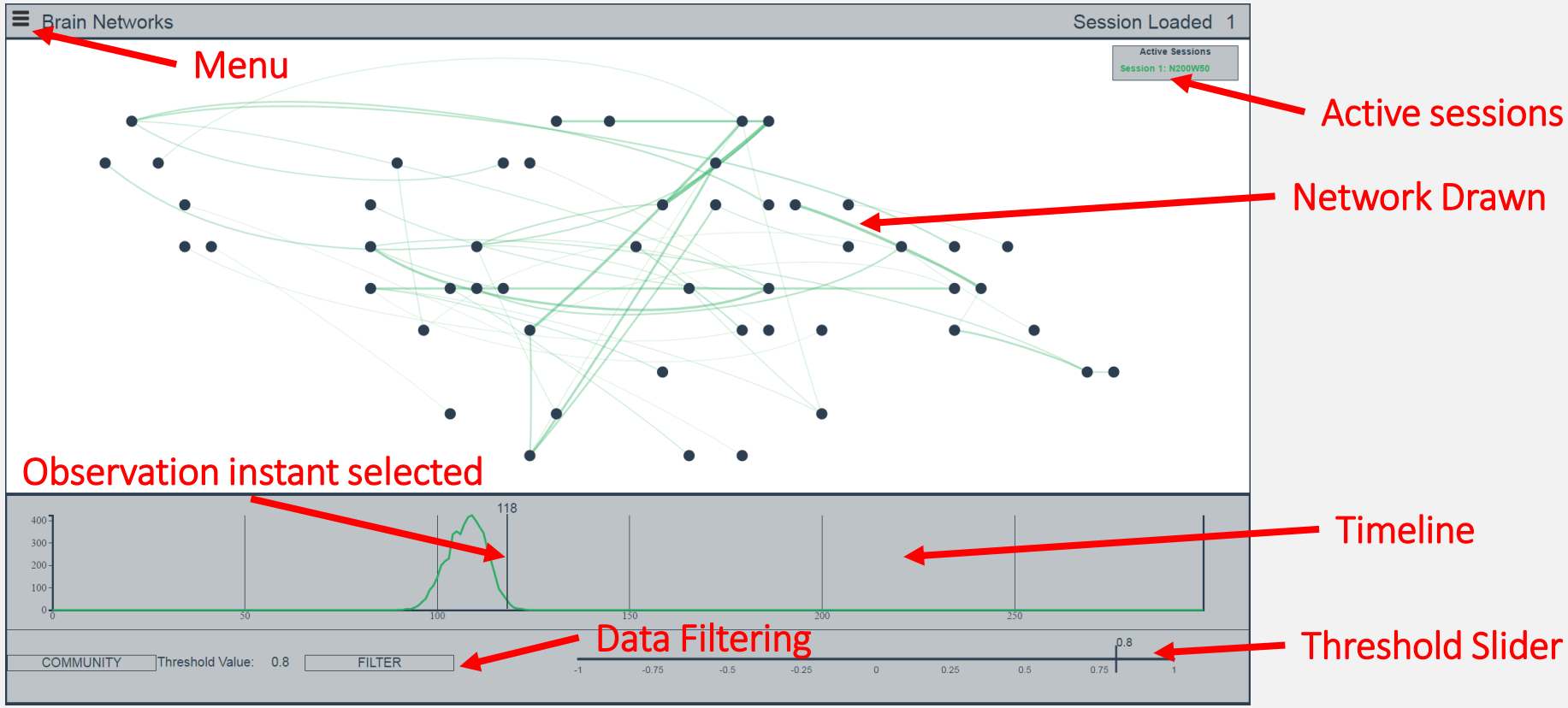
Visualization tasks:

- Dynamic Network Exploration
- Dynamic Threshold Update
- Dynamic Networks Comparison
- Local Analysis
- Future Complex Analysis

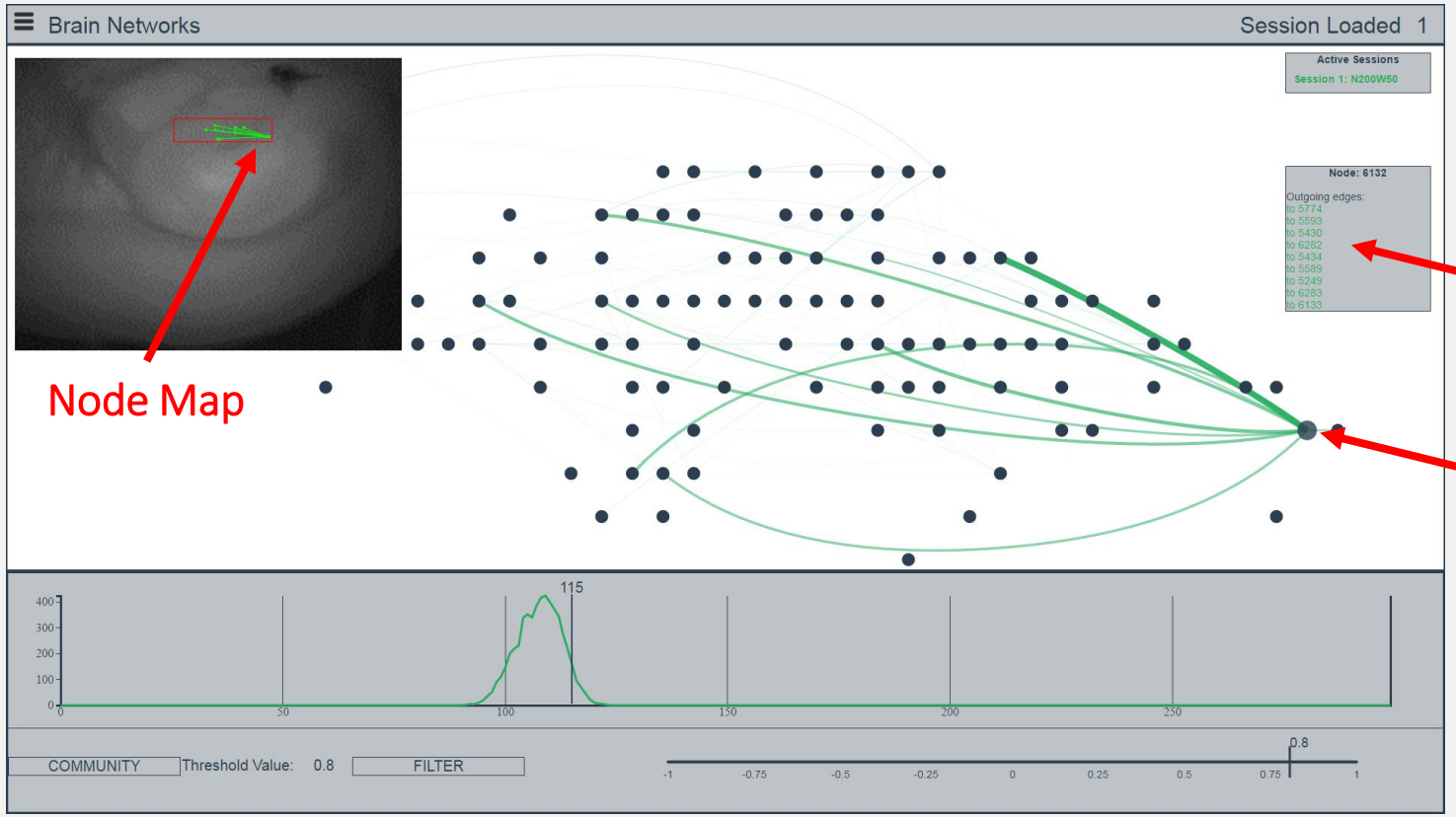


Top: Van Wedeen, MGH/Harvard. Bottom: Indiana Univ./Univ. of Lausanne/EPFL

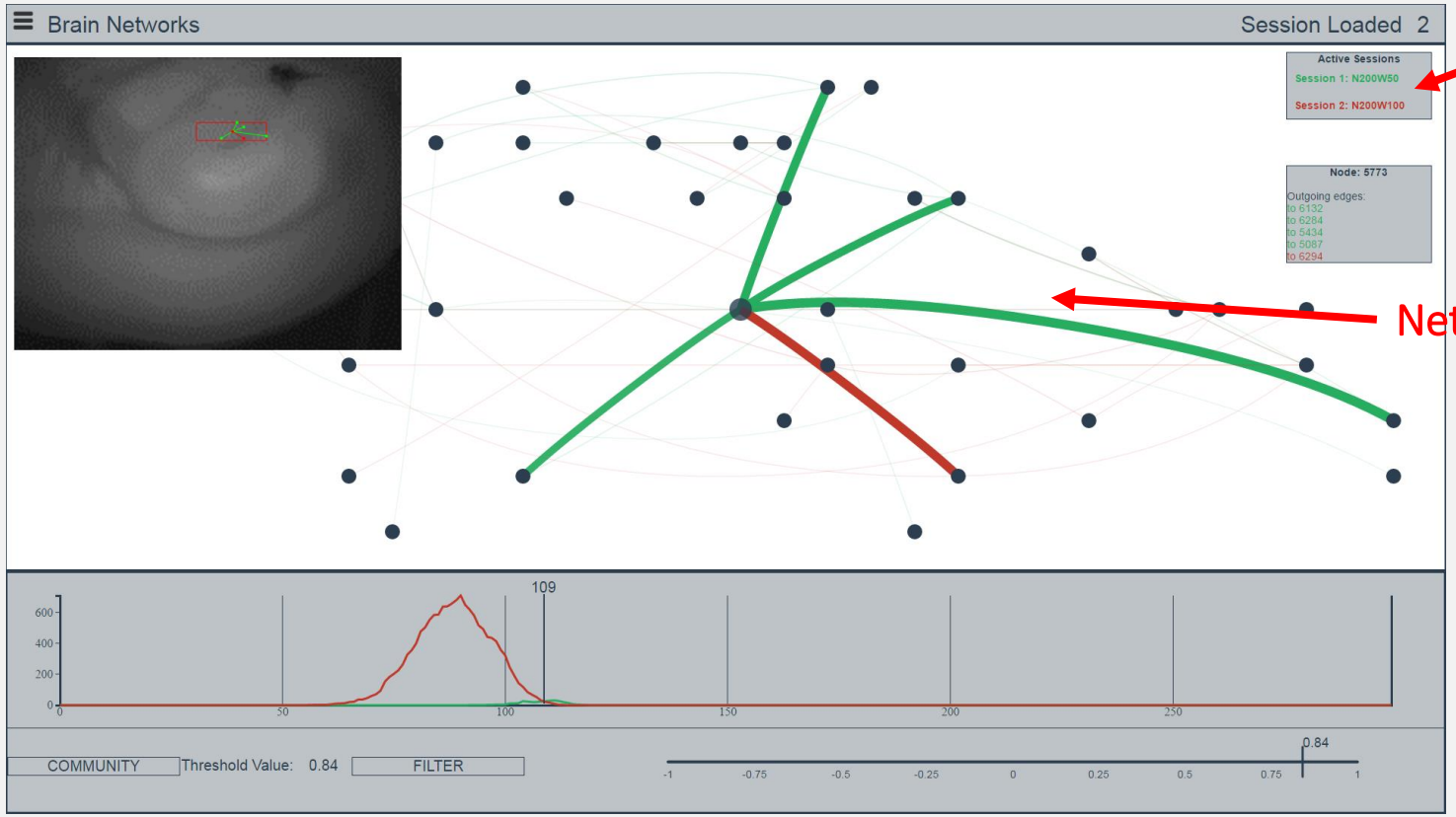
Brain Network Visualization (2/6)



Brain Network Visualization (3/6)



Brain Network Visualization (4/6)

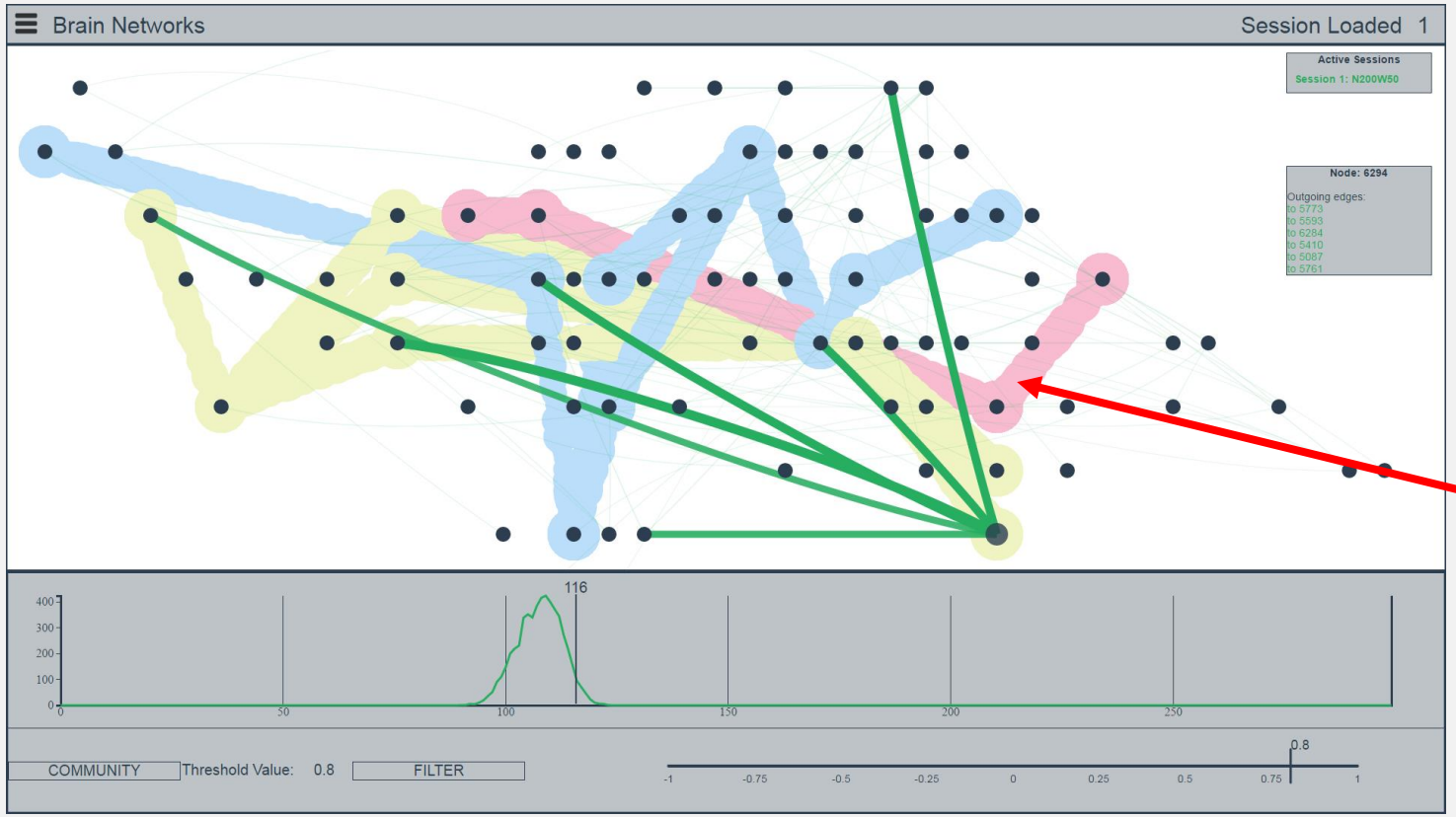


Two active sessions

Network Comparison

Brain Network Visualization (5/6)

Brain Network Visualization (6/6)



Community

Outline

1. Spatio-Temporal Network
2. GPU Programming Model
3. Similarity Computation
4. Experimental Evaluation
5. Brain Network Visualization
6. **Final Considerations**

Final Considerations

GPU Computation:

- Similarity Computation Time achieved very low.
- Avoid the saving of the result.
- Use the GPU computation as First Step of a more complex pipeline.
- Include other similarity measures.

Visualization Tool:

- Flexibility and Simplicity.
- Adaptability to any context of Dynamic Network application.
- Implements different analysis results (e.g. Communities Analysis).

QUESTIONS?

Thanks for your attention!