Trinity College HARTFORD CONNECTICUT

Image Matching Using Hypergraphs on the GPU

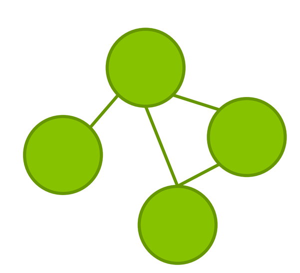
Lin Cheng[1], Reid Delaney[2], Minghui Liu[2] and Peter Yoon[2]
[1]Department of Engineering, Hartford, CT [2]Department of Computer Science, Hartford, CT

Abstract

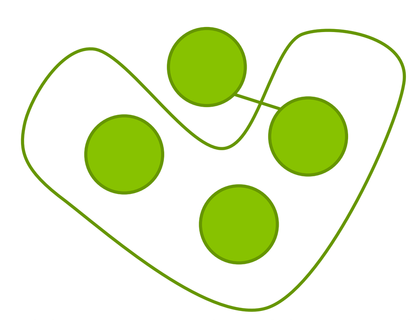
A *hypergraph* is a generalization of an ordinary graph in which edges can connect any number of vertices. The process of determining correspondence between the nodes in two hypergraphs, called *hypergraph matching*, helps solve problems such as image matching and object recognition. However, it is computationally demanding, making it impractical in real-world applications. Our main contribution is to accelerate the process by implementing it on the GPUs. Our preliminary result shows that a high accuracy and a high speed-up can be achieved when try to matching two datasets using hypergraphs when running on the GPU.

Hypergraphs

Hyperedges may connect any number of vertices rather than just two. This allows hypergraphs to model complex relations such as relationships of users in a social network website, or the reactions between compounds in a complex chemical reaction.

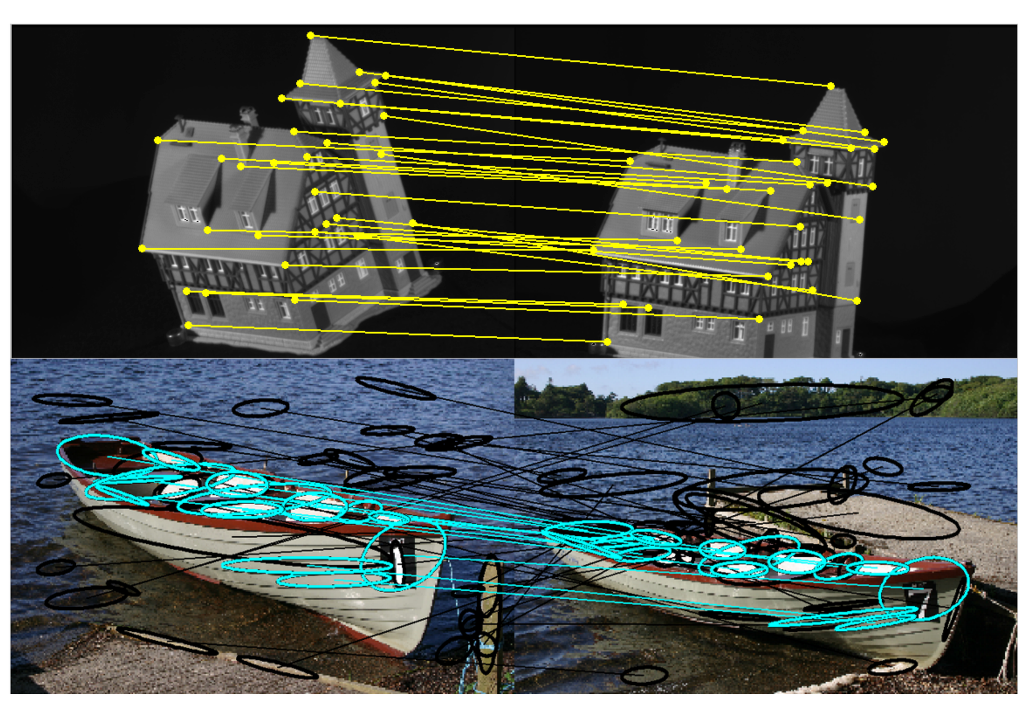


Ordinary graph



Hypergraph with an edge that connects 3 vertices

Image Matching



Example image from [1]

- Mark feature points of two images and represent them as nodes.
- Connections between nodes are represented by hyperedges.
- Conventional image matching algorithms use ordinary graphs, which can only represent pair-wise relations.
- Hypergraphs can be used to represent more sophisticated patterns and images.

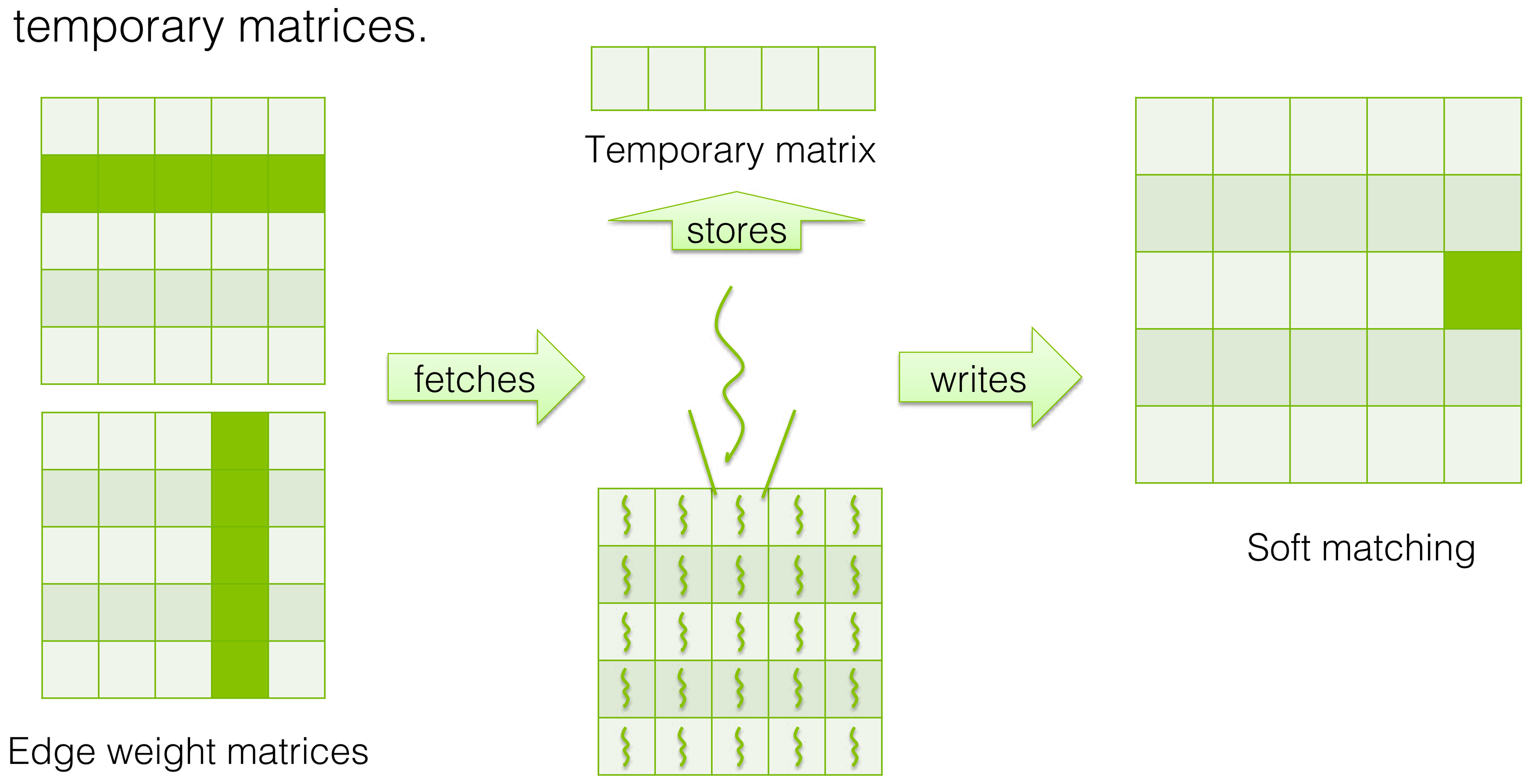
Algorithm

The algorithm based on a probabilistic approach of hypergraph matching [2]:

1. Take *weight matrices* of two hypergraphs and calculate the similarity scores of every pair of edges. Each score indicates how similar a pair of edges are.
2. Calculate similarity scores of every pair of nodes and the resulting matrix, called *soft matching*, contains the probabilities that any given pair of nodes match.
3. Convert the soft matching result matrix into a permutation matrix that contains only 1s and 0s, known as *hard matching*, gives a deterministic answer to the matching problem.

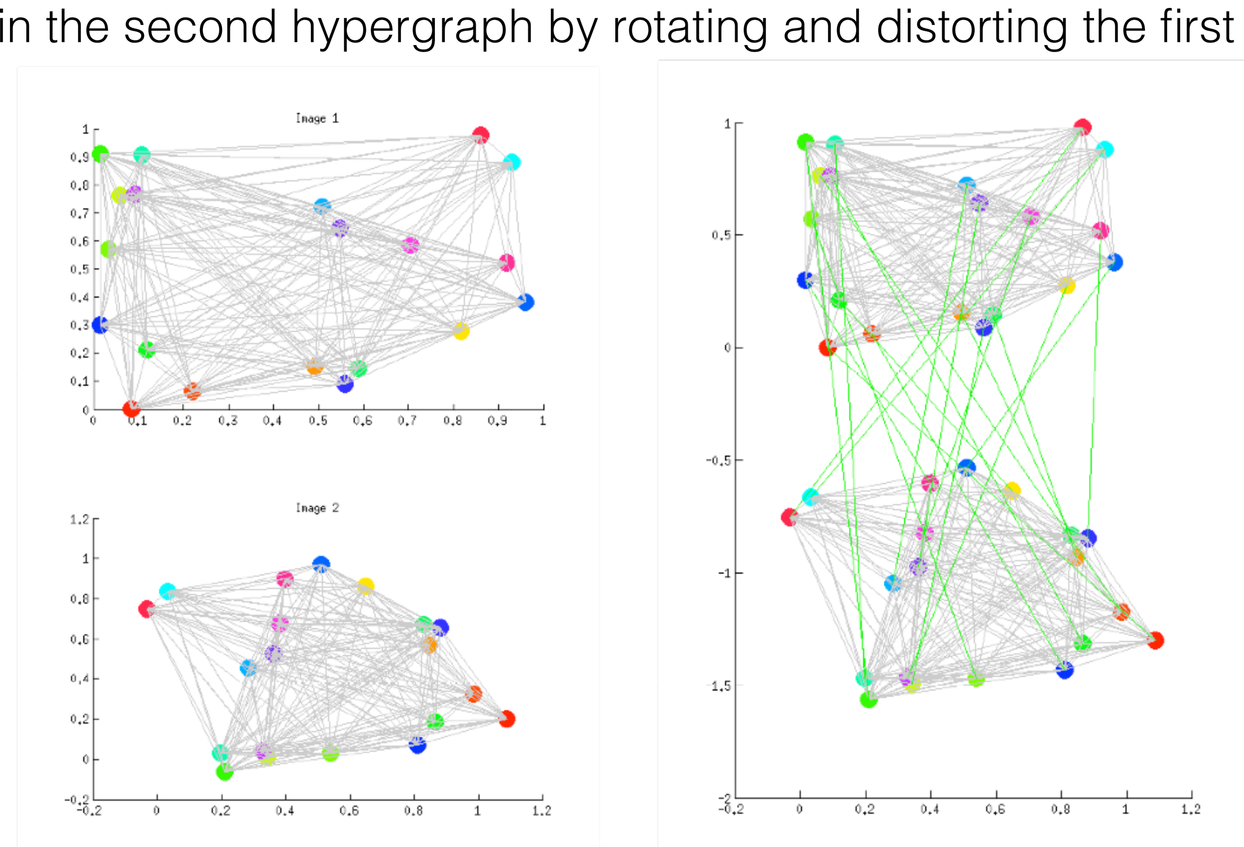
Implementation on GPU

- For hypergraphs with *N* nodes, launch a grid of N^2 threads on the GPU.
- Each thread fetches data from edge weight matrices, calculates edge similarity score, and stores the result in temporary matrices.
- Each thread then calculate vertex matching similarities using data in temporary matrices.



Testing Method

- Generate random points on a 2D-plane.
- Represent their distances as edge weights to construct a hypergraph.
- Obtain the second hypergraph by rotating and distorting the first one.



Example (test size = 10)

Preliminary Result

Our result shows that the parallel implementation on GPUs achieves a 8x to 10x speed up for moderate test sizes compared to the serial implementation on CPUs.

	8 Nodes	16 Nodes	32 Nodes	64 Nodes	128 Nodes	256 Nodes	512 Nodes	1024 Nodes
Parallel	0.07	0.08	0.19	0.55	2.92	26.29	310.79	4713.17
Serial	0.000471	0.000619	0.094972	1.49	23.85	378.6	6043.23	97319.61

Specification: Dual Intel® Xeon® E5-2620 CPUs with 64GB main memory, NVIDIA® Tesla® K20c GPU with 5GB memory.

- Y axis is drawn in log scale.
- Parallel implementation needs to allocate memory on and copy inputs to the GPU and therefore has a large overhead causing it to run slower for small test sizes.

Future Directions

- Implement the algorithm on multi-GPU platforms to further enhance performance.
- Sample and test using real images instead of randomly generated datasets.
- Utilize tensors to extend the algorithm to support higher degree hypergraphs.

Reference

[1] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. "Hypergraph matching via reweighted random walks." *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE, 2011.
[2] Ron Zass, and Amnon Shashua. "Probabilistic graph and hypergraph matching." *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on.* IEEE, 2008.

Acknowledgement

This research was supported by:
- CUDA Teaching Center Program, Nvidia Research
- Student Research Program, Trinity College