To define and identify a region-of-interest (ROI) in a digital image, the shape descriptor of the ROI has to be described in terms of its boundary characteristics. To address the generic issues of contour tracking, the yConvex Hypergraph (yCHG) model was proposed by Kanna et al [1]. This yCHG model represents any connected region as a finite set of disjoint yConvex hyperedges (yCHE), which helps in performing the contour tracking precisely without retracting the same contour. We observe that the serial implementation of the yCHG is quite costly in terms of memory and computational requirements for high resolution images. These issues motivated us to exploit the high level data parallelism available on Graphic Processing Units (GPGPUs). In this work, we propose a parallel approach to implement yCHG model by exploiting massively parallel cores of NVIDIA Compute Unified Device Architecture (CUDA). We perform our experiments on the MODIS satellite image database by NASA, and based on our analysis we observe that the performance of the serial implementation is better on smaller images, but once the threshold is achieved in terms of image resolution, the parallel implementation outperforms its sequential counterpart by 2 to 10 times (2x-10x). We also conclude that an increase in the number of hyperedges in ROI of given size does not impact the performance of the overall algorithm.

In the yCHG model, union of all the hyperedges, referred to as yConvex Hypergraph (yCHG) gives the original region of interest. A yCHG is one in which no vertical line intersects the boundary curve more than twice. Intersection of any two hyperedges is empty and all hyperedges are simply-connected bounded by either Jordan or non-Jordan curve.

With the ever increasing size of images, image processing and representation using the yCHG algorithm becomes costly in terms of memory and computational time.

In the yCHG model, union of all the hyperedges, referred to as yConvex hyperedges (yCHE) gives the original region of interest. A yCHE is one in which no vertical line intersects the boundary curve more than twice. Intersection of any two hyperedges is empty and all hyperedges are simply-connected bounded by either Jordan or non-Jordan curve.

Our results with the sequential implementation of the yCHG show that:

1. The runtime of the yCHG algorithm increases linearly for images up to a resolution of 2000x2000 but a significant change in runtime is observed for images with a higher resolution.
2. The runtime remains constant for images with varying number of hyperedges.

In our contribution, we exploit the inherent data parallelism in the yCHG algorithm on the GPU, which is well suited for parallel processing. Based on the existing yCHG algorithm, we divide an input image to a number of individual column vectors depending on the image resolution, and process each vector on a separate thread on the GPU concurrently.

4. PROPOSED ALGORITHM

4.1 Finding Cut Vertices

5. RESULTS

In order to keep constant hyper edges, we take an image of a resolution of 21000x21000 and vary the resolution.

We used the following system specification for our analysis.

- Number of cores: 2
- Clock Speed: 2660 MHz
- Memory Clock: 1600 MHz
- Memory Interface: 64 bit

On basis of our experiments, we find out the following:

1. For an image with resolution greater than the threshold of 2000x2000, the GPU implementation outperforms the CPU by a factor of 2x-10x.
2. Varying the number of hyperedges in an image with constant resolution does not impact the performance of the overall algorithm. This is due to the fact that our proposed algorithm is directly dependent on the resolution of the input image irrespective of other factors.

### 6. MODIS DATASET

Some of the sample images from MODIS dataset are shown above. The MODIS satellite image database, which is a lower spatial resolution to the ASTER, covers the entire earth everyday frequently repeating its coverage to provide real-time images.

### 7. CONCLUSIONS

In this paper, we exploit data parallelism in an existing yCHG algorithm by enhancing the algorithm to remove the data dependencies. Our implementation results with NVIDIA CUDA show that with an increase in the image resolution the parallel implementation improves the performance of an already fast (relatively) algorithm by 2X-10X, opening up a host of potential new applications that require real time image processing.

### REFERENCES