15418 Parallel Computer Architecture

Carnegie Mellon University

Fall 2014: Final Project

Parallel Seam Carving

Aditya Bist (abist)
Vinay Palakkode (vpalakko)

Acknowledgement

We would like to thank **Cary Yang** and **Prof. Kayvon Fatahlian** for their valuable inputs and suggestions. It is also worth mentioning that we had a very brief email conversation with **Jonathan Ragan-Kelley** and **James Hegarty** while we were exploring the possibility of using Halide to accelerate Seam Carving.

Summary

With the aid of certain approximations, we designed a new algorithm for content aware retargeting which ran **~25x** faster than the conventional seam carving on a **x86** Haswell platform. We adopted this approach because owing to the inherent sequential nature seam carving mapped directly on to a GPU (NVIDIA GTX 780) yielded only an approximate speed up of **7x** when compared to a single threaded CPU baseline implementation. And was largely due to parallelization of the energy computation stage of the algorithm (which is quite trivial)

Background

Image targeting is the process of changing the resolution of an image/ video frame without most likely without preserving the aspect ration. Content agnostic image retargeting solutions like crop or scaling performs poorly in maintaining visual information proportional to the change in the aspect ration. Seam Carving is a content-aware image resizing algorithm developed by Shai Avidan, of Mitsubishi Electric Research Laboratories (MERL), and Ariel Shamir, of the Interdisciplinary Center and MERL. The gist of the algorithm is that we remove pixels of less importance which means the pixel removed need not be along a straight line in orthogonal directions. Such a low energy 8 connected path is called a Seam and evidently this is not a transformation which maps straight lines to straight lines (Affine).

A complete description of the algorithm is available in our project website. Or please look into the references cited in the end of this doc as well as in our web page
http://www.contrib.andrew.cmu.edu/~abist/seamcarving.html

Challenges

- The default seam carving algorithm uses dynamic programming for the seam map computation, which is not GPU friendly and there is very low computation to communication ratio (arithmetic intensity).
- In order to speed up this algorithm, we definitely have to make approximations which would result in degradation of the retargeted image quality (aka. result in visual artifacts).
- Since there is no ground truth for us to compare the modified algorithms outputs to, defining an evaluation metrics for us is as challenging as optimizing the baseline algorithm itself (if not more)
- Creating a test dataset (input images) with different energy distributions/patterns so as to see the extent of artifacts introduced by the approximations

Approach

We spent some time doing research on different energy functions to see how the choice of energy function affects the quality of the seam carved output. The best results are obtained with Histogram of Oriented gradients when it comes to quality of the output, but the this nearly an impossible algorithm to achieve realtime performance on images as big as **1280 x 720**. But for all practical purposes a simple image gradient magnitude works well and we decided to stick on to this as the preferred energy function.

As the lion share of the time is spent on computing the energy, we decided to attach this part first and it is heavily data parallel and CUDA seamed to be a really good choice for this.

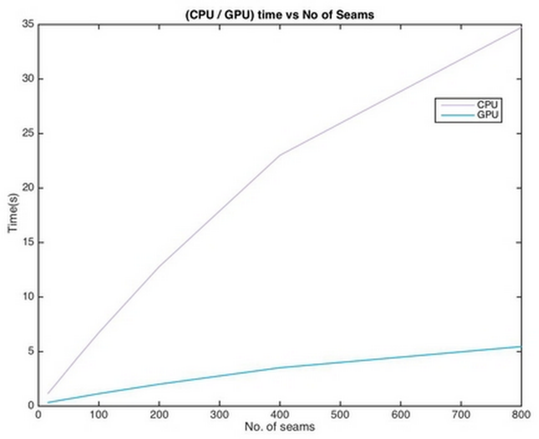
Seam Carving CUDA

We implemented Seam Carving on NVIDIA GTX 780 with the aid of 3 kernels. One for the energy map computation which basically spawns one CUDA thread per output element with a fixed block size of 32 x 32 (coming from the fact that the max # threads per block in GTX 750 is 1024). Boundary checks were done by the threads on the image boundary. So definitely the input pixels have to be copied to the Device memory and results to be copied back to the host memory.

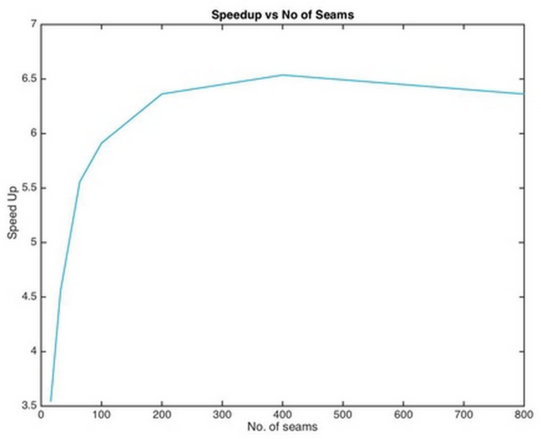
Minimum Cost table is updated at the granularity of one image row. This means we have a kernel launch per row of the image. And each row element needs the info from the previous row.

We used shared memory to slightly improve the results here, but unlike energy map this stage is not trivially data parallel.

Time taken on the CPU and the GPU vs Number of seams removed



Speed of GPU implementation wrt baseline vs Number of seams removed



The Minimum Reduction Kernel also does is bottle necked by the dependencies in the previous rows. So the bottom line is that when compared to a OpenCV aided single threaded implementation of **x86** Haswell, the CUDA version is approximately **7x** faster. This is pretty much in agreement with the literatures on CUDA implementation on Seam Carving (kindly see references). But this is a trivial and intuitive result, as the optimization comes from the data parallel nature of the energy computation. Hence we decided to come up with a new algorithm which targets at achieving much higher performance from a single threaded implementation. Dynamic programming is essentially sequential. The only way to parallelize such algorithms would be making approximations based on heuristics and have a balance / trade off between quality and performance.

418 Carving : A Hacky Image Retargeting Algorithm

Our proposed algorithm would not beat the quality of image generated by classical seam carving. However, we can improve the quality of our algorithm by choosing a more sophisticated energy function which is more verbose in terms of visual salience.

ALGORITHM

1. Compute the Energy map as in the Conventional Seam Carving, but just once per entire image unlike in the previous case where we computed this for every seam removal

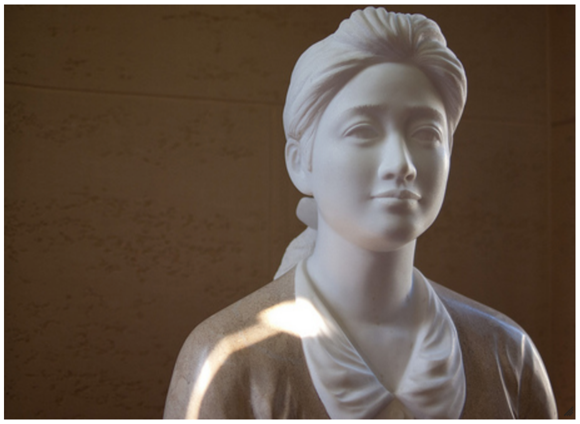
2. Sort the first row of energies and get the indices of the lowest "**n**" indices to choose the start points of the "n" seams to be removed.

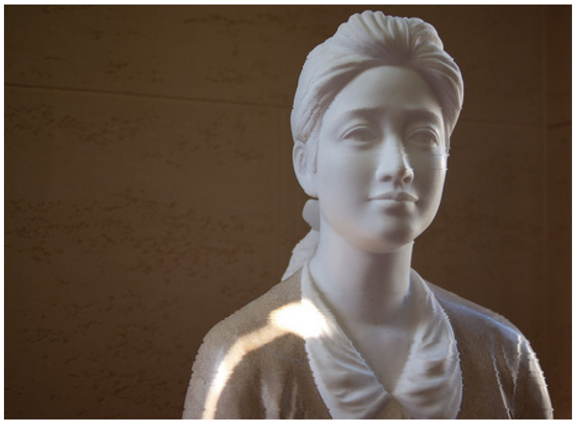
3. Use dynamic programming exactly as in Seam Carving but in order to prevent multiple seams converging on to a single path, the moment an pixel is chosen by a seam, mark that pixel as used (UMAX in the energy map), so that none of the seams will pick this point

4. Remove all the seams in one pass, which means we will resize the output matrix in one go. For an Image of width **W**, for **n** seam removal, we reduce the number of memory writes drastically. (Kindly see the analysis section to see the trend)

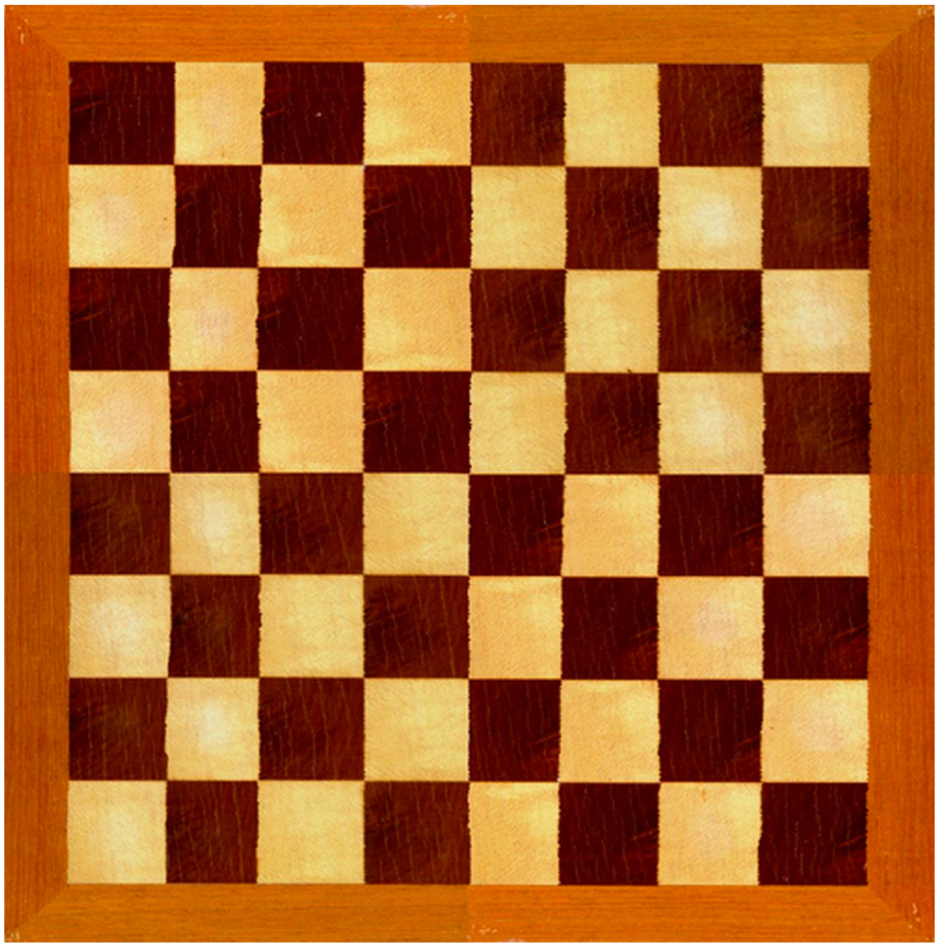
RESULTS AND PERFORMANCE ANALYSIS

A good example where this approximation works is where there is a significant amount of low energy paths possible in one region of the image


Seam Carved from W = 500 to 450


418 Carved from W = 500 to 450

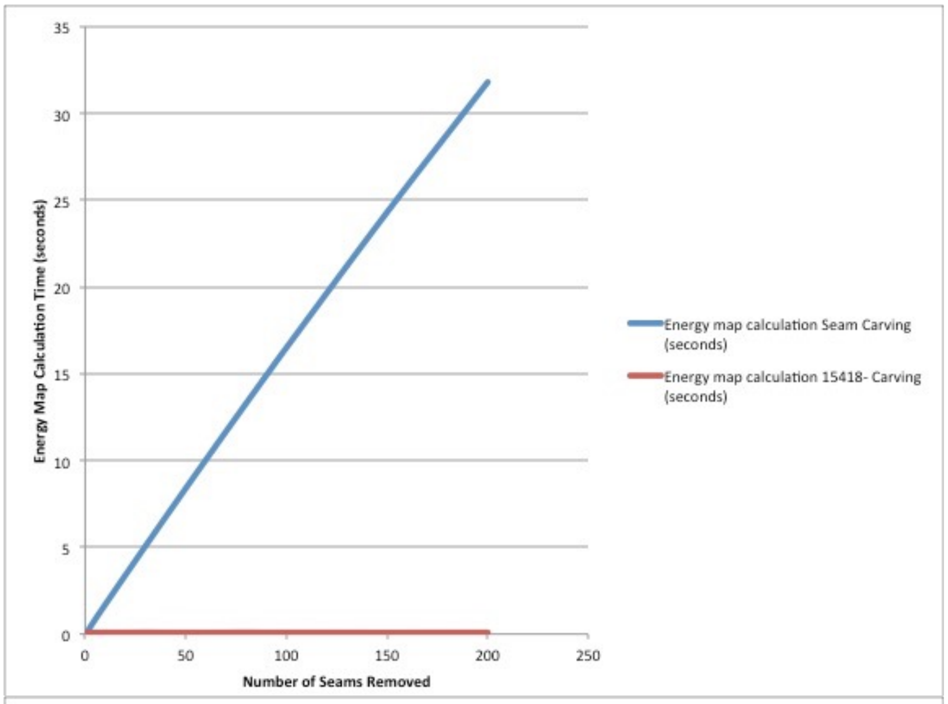
But if there is concentration of low energy paths periodically, then this can result in significant artifacts visually. Consider the follow 1200 x 1210 Chess board image when 200 seams where 418 carved looks like the following

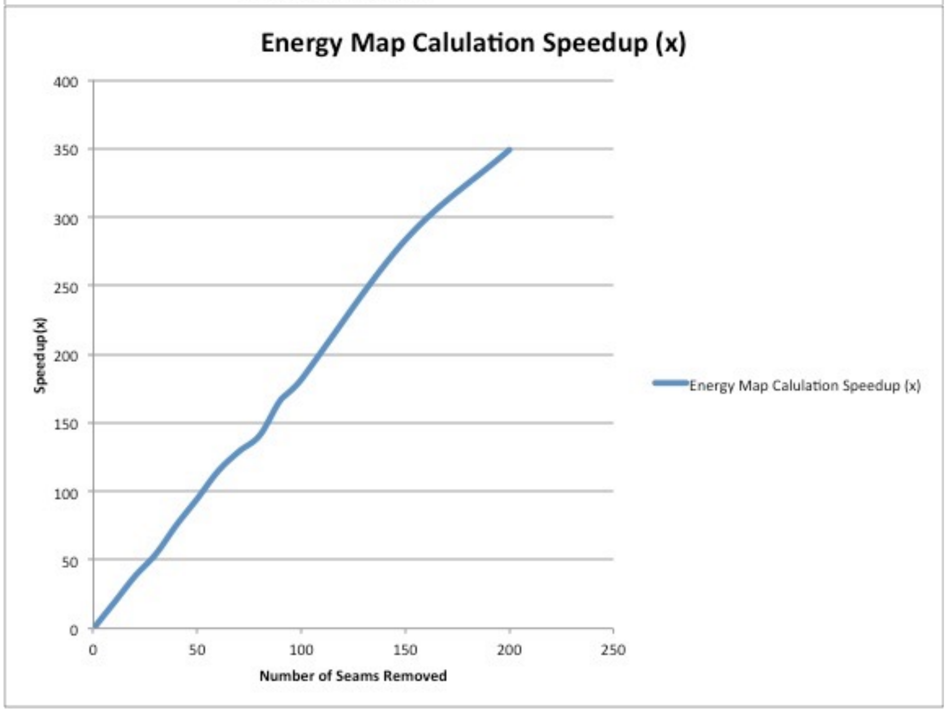


Performance

- All the figures reported are based on the readings obtained with image sized **1428** x **968**
- MacBook Pro "Core i7" 22 nm "Haswell/Crystalwell" 2.8 GHz Intel "Core i7" processor (4980HQ), a 6 MB shared level 3 cache, 16 GB of onboard 1600 MHz DDR3L SDRAM
- Baseline code is the same single threaded OpenCV aided Seam carving code used as the baseline to compare CUDA implementation.
- We used cylceTimer.h to profile all the code. All the speed up (X) reported are wrt. to the above baseline (unless and otherwise specified)

1. 418 Carving computes the energy only ones, so for a given size the cost of energy computation is independent of the number of seams to be removed. This is a tremendous optimization when it comes to removing a large number of seams. Interestingly the new algorithm is even more harder to parallelize (although a single threaded implemented outperforms any GPU implementation since we have reduced the cost of energy

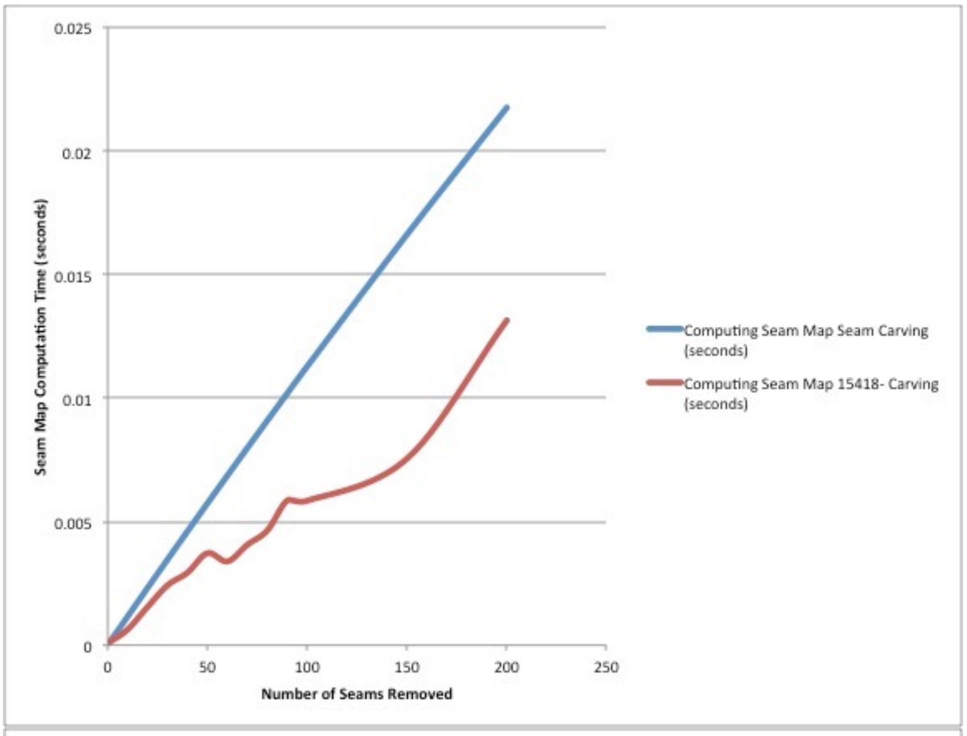


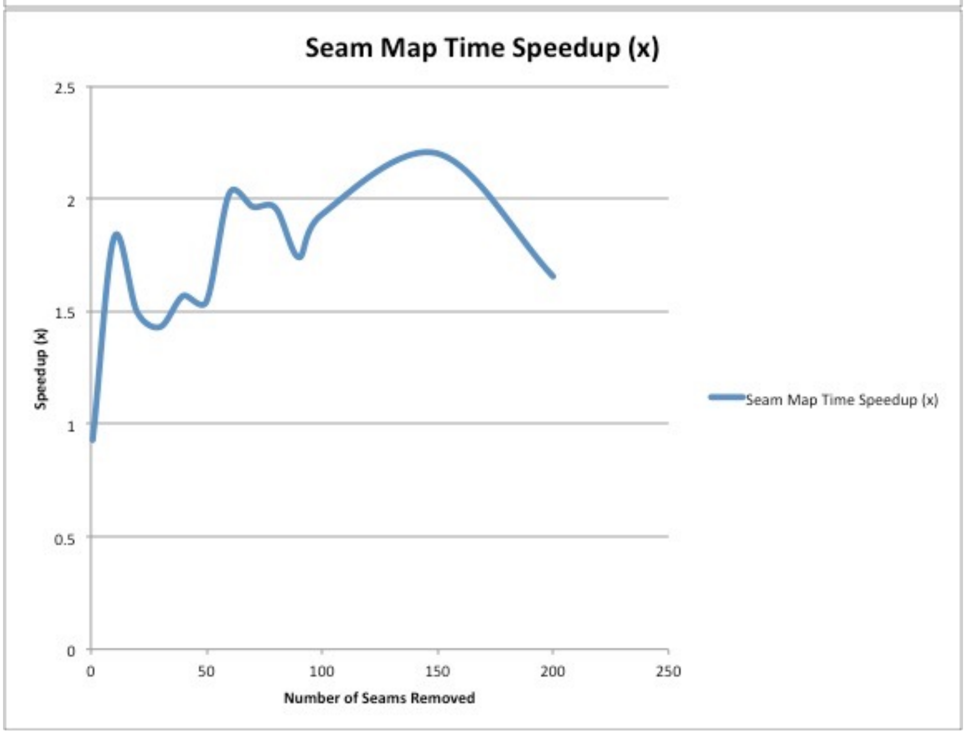


computation algorithm itself, so it is no more compute bound.

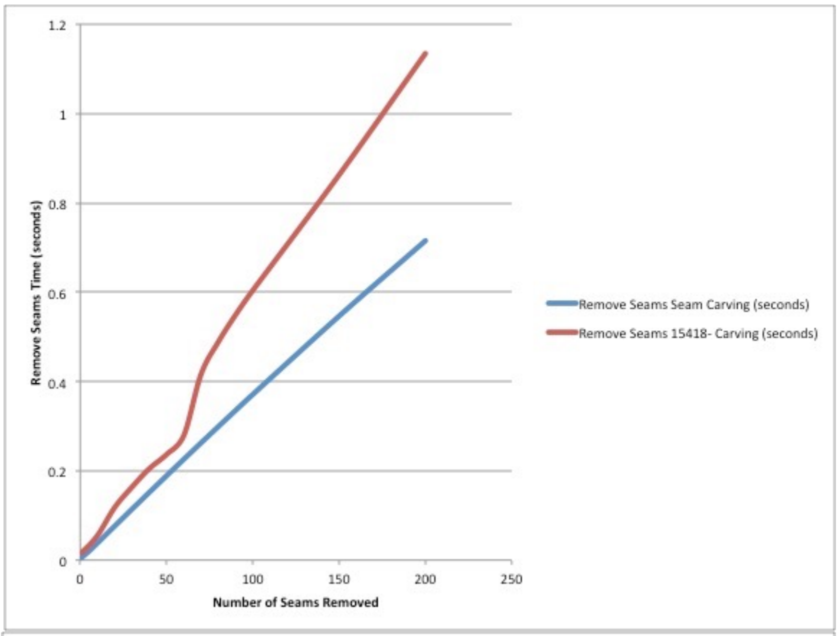
2. Seam Map Computation

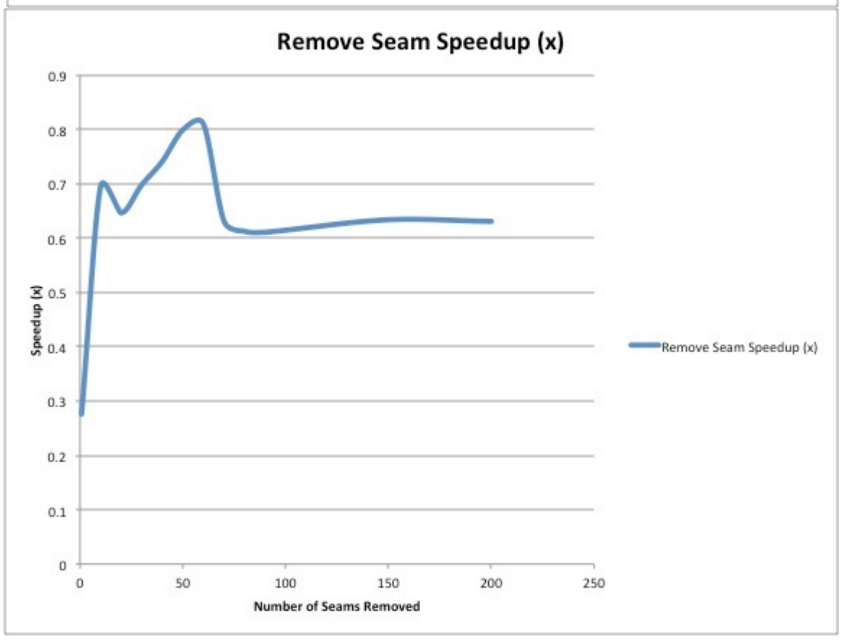
This stage is the highly sequential in nature and it still used the dynamic programming. But





we avoid doing this step on a per seam basis following by an intermediate matrix resizing.

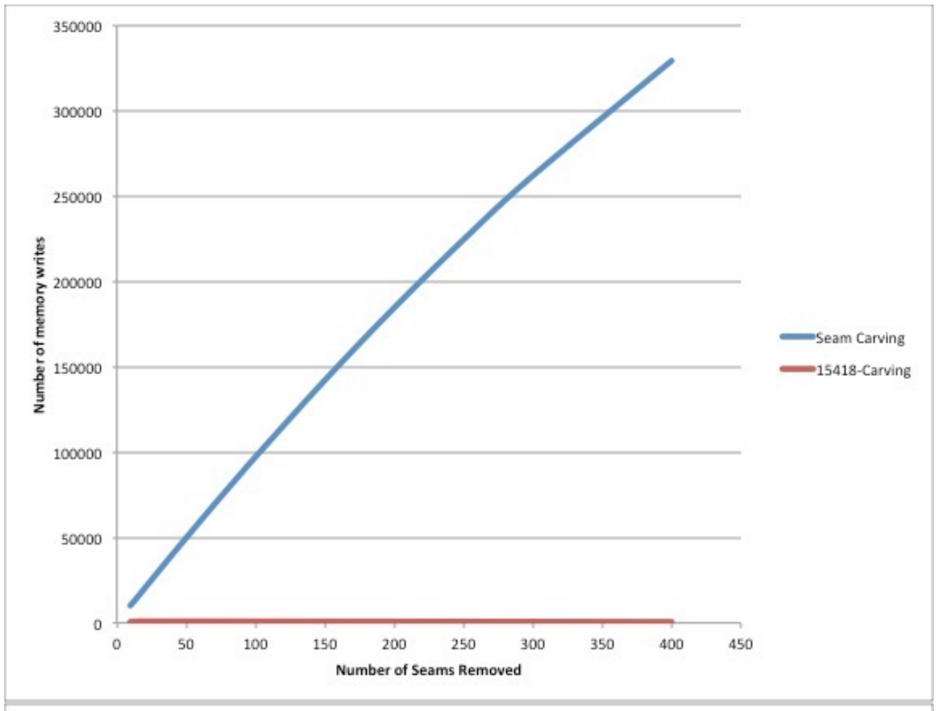


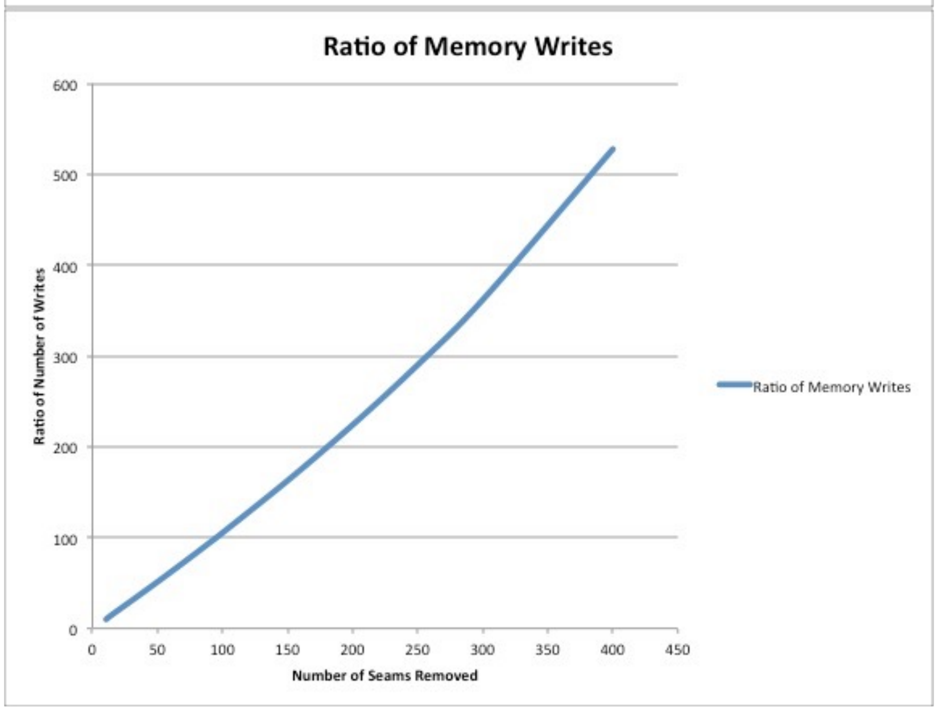


We mark all the seams in one pass and then proceed to remove all of them in one pass.
Seam removal is the stage where we use the results from the previous stage. This is one stage where 418 carving performs poorly with increase in the number of seams to be removed. This because for every row to be resized, in 418 carving in the worse case we would need of each pixel index with n Seam indices in the map to see if it is marked in the map or not.

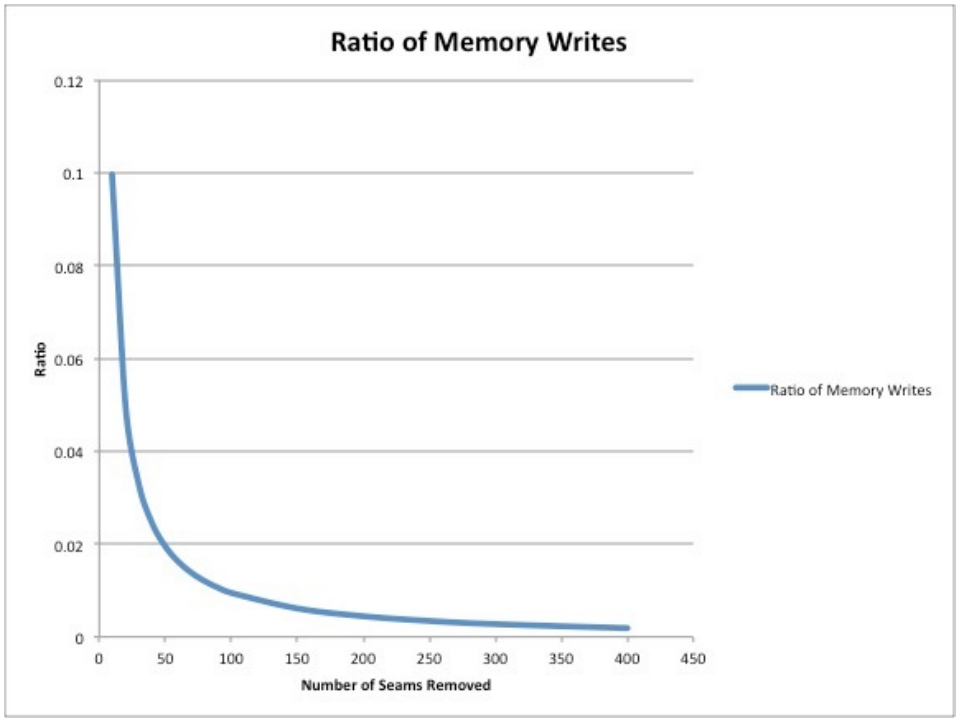
3. **Matrix resizing**:

This is the stage where we obtained huge performance boost by bringing down the number of





writes to the memory. Just like the energy computation stage we brought down the number in such a way that it is independent of the number of seams removed.



This is one of those algorithms where the entire data set would be in the L2 cache, which means, the only way to further optimize the algorithm is to reduce the number of writes (BusRdx) as much as possible.

References

- Avidan, S. AND Shamir, A. 2007. Seam Carving for Content-Aware Image Resizing.
- Duarte, R. AND Sendag, R. 2012. Accelerating and Characterizing Seam Carving Using a Heterogeneous CPU-GPU System.
- NVIDIA CUDA Programming Guide