Optimizing Seam Carving on Multi-GPU Systems for Real-time Image Resizing

Ikjoon Kim[†], Jidong Zhai[†,*], Yan Li[†] and Wenguang Chen[†,¶]
[†]Department of Computer Science and Technology, Tsinghua University, BeiJing
[¶]Technology Innovation Center at Yinzhou, Yangtze Delta Region Institute of Tsinghua University, ZheJiang
Email: jjkim622@gmail.com, zhaijidong@tsinghua.edu.cn, liyan665@gmail.com, cwg@tsinghua.edu.cn

***Abstract*—Image resizing is increasingly important for picture sharing and exchanging between various personal electronic equipments. Seam Carving is a state-of-the-art approach for effective image resizing because of its content-aware characteristic. However, complex computation and memory access patterns make it time-consuming and prevent its wide usage in real-time image processing. To address these problems, we propose a novel algorithm, called Non-Cumulative Seam Carving (NCSC), which removes main computation bottleneck. Furthermore, we also propose an adaptive multi-seam algorithm for better parallelism on GPU platforms. Finally, we implement our algorithm on a multi-GPU platform. Results show that our approach achieves a maximum 140× speedup on a two-GPU system over the sequential version. It only takes 0.11 second to resize a 1024×640 image by half in width compared to 15.5 seconds with the traditional seam carving.**

I. Introduction

Image resizing is increasingly important for exchanging and sharing pictures with different resolutions between various electronic equipments, such as smart phones, pads and smart TVs.

A large number of on-line image resizing softwares have been provided on the website, and they mainly fall into two categories, scaling and cropping [1]. Scaling is a popular image-resizing method through resizing a whole image without considering its contents, resulting in the main contents of the image are distorted (See Figure 1(b)). Cropping is another famous approach to resize an image, but the resized image may not include the important information of the image (See Figure 1(c)).

Recently, an important category of methods, called content-aware image resizing algorithms are proposed for effective image scaling [2], [3]. The purpose of these methods is to resize images without distortion on various media (cell phones, pads) through preserving main objects of the image and discarding unimportant parts (See Figure 1(d)). Because of few of image distortions, the content-aware image resizing method has been attracting more and more people's attention. Among these methods, seam carving [4] is a state-of-the-art algorithm for effective image resizing, which focuses on the energy value of each pixel and removes a number of seams that have the lowest energy value until satisfying the desired size of an image.

*Jidong Zhai is the corresponding author of this paper.

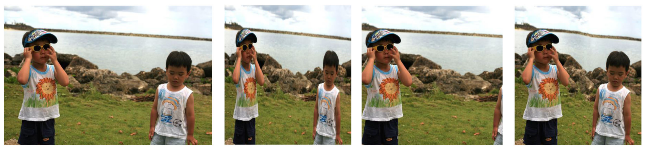


Fig. 1. Comparison of various image resizing methods. (a) Original image. (b) Scaling. (c) Cropping (d) Seam carving.

However, the main limitation of seam carving is its complex computation model. In order to find an optimal seam, a dynamic programming process is needed during the image resizing. Typically, these methods need to take a long time to resize a moderate size of image. For example, resizing a 1024×640 image by half in horizontal direction takes more than 15 seconds on a personal computer with 2GHz Intel Xeon processor. As a result, it is not proper for the real-time application on current user devices such as a personal computer and a hand-held device.

On the other hand, GPU devices are becoming more and more popular on current electronic equipments. A large number of high-end smart TVs and smart phones all equip high-performance GPUs. Moreover, since ARM began to support OpenCL programming interface [6] on its GPUs, NVIDIA also announced a plan to enter the embedded system market. In this paper, we focus on utilizing GPU devices to accelerate the content-aware algorithms to achieve real-time image resizing. We use seam carving as an example in this paper, but our optimizing techniques are general for this type of image resizing algorithms.

Effective parallelization of seam carving on GPU devices is a challenging problem due to its complex computation model. There are two main challenges: (1) **Computation Dependence**: Dynamic programming is a key step to compute an optimal seam during image resizing, which accounts for a large proportion of the program execution time. It is very hard to parallelize the dynamic programming on GPU devices due to the computation dependency. (2) **Intensive and Irregular Memory Access**: In order to compute various intermediate results, such as *energy maps* and *seam maps*, a variety of irregular memory access patterns are used, which can hurt the program performance significantly with the complex memory hierarchy of GPUs.

To address these problems, we propose a series of optimization methods in this paper to improve the seam carving performance on the GPU platform. In summary, we make the following contributions:

1) We propose a novel Non-Cumulative Seam Carving (NCSC) algorithm for effective image resizing, which removes the dynamic programming part in the traditional algorithm and guarantees the similar image quality. Our algorithm can be effectively scaled to large-scale GPU systems due to the reduction of computation dependence. Moreover, we also extend our algorithm for a hybrid CPU-GPU version for better utilization of CPU resources.
2) We also propose an adaptive multi-seam algorithm, which can compute multiple seams at each iteration and automatically decides the number of seams to be deleted. We implement these two algorithms, the NCSC and the adaptive multi-seam, on multi-GPU systems with the GPU *peer to peer access* feature.
3) We propose a series of GPU architecture-dependent implementation methods for improving the parallel seam carving algorithm. And we also share our optimization experience on the memory allocation, access patterns, and transmission techniques of the GPU memory management.

We implement our optimized seam carving algorithm on multi-GPU heterogeneous systems. Results show that our method can reduce the image resizing overhead significantly. For example, the single-GPU version carving a single seam at a time can achieve $57\times$ speedup over the CPU version, and the two-GPU version with the removal of 2 seams at one time achieves $140\times$ speedup. It only takes about 0.11 second to resize a 1024×640 image by half in width.

The rest of this paper is organized as follows. We first present related work in Section II and introduce seam carving algorithm in Section III, and describes NCSC, adaptive multi-seam and multi-GPU algorithm in the following section. In Section V, we present various optimizations for CUDA architecture. We show evaluation results in Section VI, and give the conclusions in Section VII.

II. Related Work

Image resizing has been studied widely in the literature on various sized display devices. The most popular method is to adapt scale down, but it causes image distortion and does not recognize important objects in the image. Cropping [1], [7], one of the other methods, contain insufficient information in the cropped image compared with the original image.

Avidan and Shamir [4] proposed a new method of resizing an image, called seam carving. Seam carving focuses on the contents of an image and remains the main objects that have large energy value in the resized image with little distortion. However, seam carving algorithm is not proper to the real-time applications because of its large computation and the limitation of parallelism using dynamic programming method.Hence previous work [8], [9] mainly focus on improving the quality of resized image rather than the performances.

The modern GPU architecture has more than hundred of cores and it can be used for general purpose computation. R. Duarte et al. [10] implemented seam carving with CUDA mainly focusing on energy map. Unlike above work, we analyze the memory access pattern and computation workload of the whole seam carving algorithm, and also consider CUDA characteristics, such as the overhead of kernel function call and the cost of using *pinned memory*. In addition, we propose a new approach for seam map, NCSC (Non-Cumulative Seam Carving) removing dynamic programming, and apply a new method to configure CUDA threads for reducing time complexity. *Stream* is a significant feature of CUDA for heterogeneous CPU-GPU system. We also use this feature to concurrently execute CPU and GPU. We hide most data transfer time between GPU and CPU to achieve better performance. In addition, we use multi-seam method to delete several seams at the same time, and achieve significant speedup with little image distortion. Finally, we implement multi-GPU version using GPU peer-to-peer access, which is a key feature of CUDA for optimization on multi-GPU platform.

III. Seam Carving Algorithm

Seam carving [4] resizes an image by removing or duplicating a seam, which is an optimal connected path of pixels having the lowest energy in an image from top to bottom for horizontal adjustment or left to right for vertical adjustment. It consists of three stages. Firstly, we construct an energy map by calculating the energy value for each pixel of an image according to the Eq. 1. Then we compute seams that connect low energy pixels crossing an image and find the optimal seam with the lowest energy summation. Finally, we delete the optimal seam and resize the image. The last two stages are executed repetitively until the desired size is reached. Figure 2 shows these stages for horizontal adjustment of an image.

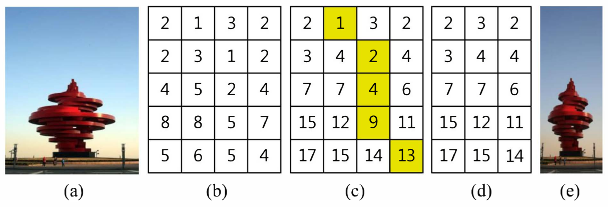


Fig. 2. Seam carving stages. (a) The original image. (b) The energy map of the image. (c) The optimal seam within the seam map. (d) Resize the map. (e) Resized image.

A. Energy map

One method to calculate the energy value of each pixel is to use the magnitude of the gradient. Let I be an $n \times m$ image and the energy function is generally defined as:

$$e_1(I) = |\frac{\partial}{\partial x}I| + |\frac{\partial}{\partial y}I| \tag{1}$$

The energy value of each pixel depends on the other eight surrounding pixels. We quantify the amount of changes from each pixel to the neighbor in x and y directions.

The serial implementation of energy map utilizes nested *for-loops* to compute those two directions of the derivatives for each pixel. We add the results in x and y directions, and write them to the energy map table. The computation of each pixel is independent, and it means that we can achieve vast parallelism.

Thus we can explore massive thread-level parallelism in GPU to accelerate the computation. However, the performance of this function on GPU is limited by its computation workload, and we analyze how the amount of computation workload affects the performance in the later section.

B. Seam map

After constructing energy map, we obtain the optimal seam, vertical or horizontal, which is determined by dynamic programming. We concentrate on finding of the optimal vertical seam for horizontal image resizing.

$$S_{i,j} = \begin{cases} E_{0,j} & \text{if } i = 0 \\ E_{i,j} + \mathtt{min}(S_{i-1,j-1}, S_{i-1,j}, S_{i-1,j+1}) & \text{otherwise} \end{cases} \quad (2)$$

In the Eq. 2, E is the energy map, S is the seam map, and i and j indicate row and column each. The first row of seam map is the same as energy map, and starts to use dynamic programming from the second row. Then we select the pixel has the minimum value from the last row, and use backward track from this pixel to the first row to make the path of the optimal vertical seam. Figure 3 describes the steps of seam map.

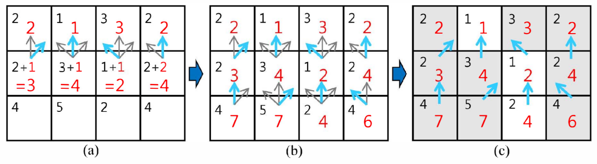


Fig. 3. Seam map steps. The top-left value in black represents the energy value of that pixel. The value in red represents the cumulative sum of energies including that pixel.

Each pixel looks up the adjacent three pixels in the upper row, and selects the pixel that has the lowest cumulative sum of energy values among them. That value is added to its own energy value. If we see the second pixel in the second row (shown in Figure 3(a)), its energy value is 3 which is marked in black and there are three possible choices (2,1 and 3) that are marked in red in the above row. Then, we select the minimum value (1) and add it to the orignal value to update the value. Hence the current energy value is 4.

The computation continues until the last row of the image is reached, and we finally obtain the cumulative energy values of all seams. Then we select the lowest value in the last row, and get the optimal vertical path following the blue arrow which starts with the last row (the white path in Figure 3(c)).

Unlike the energy map, the seam map has low parallelism due to its dynamic programming characteristic. We have to access the energy map row by row, thus it has *O(N)* time complexity if the height of an image is *N*. In this paper, we propose NCSC for seam map to improve the performance.

C. Resize map

Resize map is relatively simple than other stages. We already know the optimal seam path, then delete a single pixel in each row. We move the pixels that are located at the right of the deleted pixel from right to left. Since resizing operation is independent for each row and it is memory-intensive, GPU can offer good performance benefits.

In this section, we have presented the original seam carving algorithm and its serial implementation. We will show how to optimize seam carving using CUDA progamming model in the following sections.

IV. Optimizations for Seam Carving Algorithm

A big challenge of seam carving is how to reduce the time consumption of dynamic programming in a seam map. Since dynamic programming is very hard to parallelize, we propose a new approach, NCSC, which consists of three parts: *non-cumulative selection, reduced time complexity and CPU-GPU hybrid* method. Then, we also present adaptive multi-seam algorithm and multi-GPU method.

A. NCSC (Non-Cumulative Seam Carving)

1) Non-cumulative selection: Original seam carving uses cumulative energy sum to select the minimum energy value when each pixel decides the direction. Although this idea guarantees to find the optimal seam which has the lowest energy level among seams, its dynamic programming characteristic causes significant time loss. Therefore, we only focus on the energy value of a pixel not the cumulative energy sum to select the direction of a pixel.

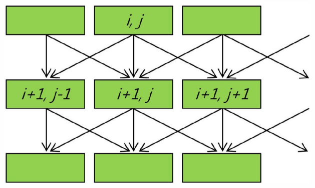
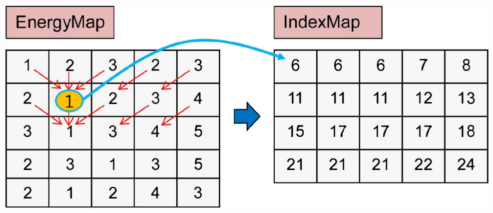


Fig. 4. Non-cumulative energy selection

Figure 4 shows all possible choices of each pixel. A pixel looks down the energy value of three adjacent pixels in the right bottom row, and chooses the one that has the minimum energy value. The selection of all pixels can be performed at the same time. This method owns vast parallelism, and we obtain the result in very short time. After the selection of each pixel, we write the value of the selected index to the index map that is used to construct a seam path later. For example, in Figure 5, the first three pixels in the first row select the second pixel of the second row as the minimum energy value, whose index is 6, thus the index value of those pixels is 6 in the index map.



1	2	3	2	3
2	1	2	3	4
3	1	3	4	5
2	3	1	3	5
2	1	2	4	3

6	6	6	7	8
11	11	11	12	13
15	17	17	17	18
21	21	21	22	24

Fig. 5. Construction of index map for tracing the path of seams

2) Reduced time complexity: In spite of not using dynamic programming, we still need to calculate the total energy value of each seam to find an optimal seam. Every single seam has the number of pixels as much as the height of the image, and we need to add all energy values of pixels. We access the energy map row-by-row based on the index map, and it means that we have to visit the global memory of GPU as much as the height of the image. If both the height and width of the image are N, time complexity is $O(N^2)$.

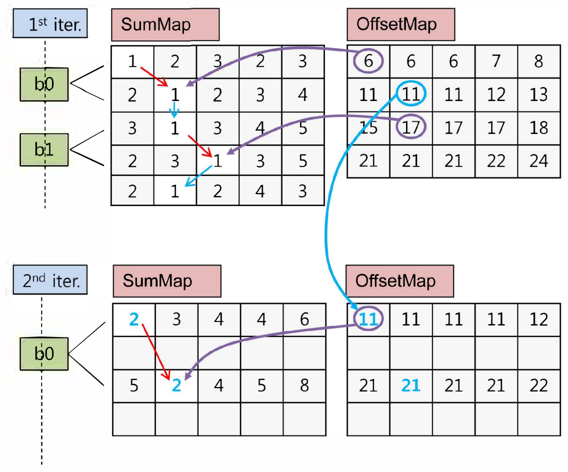


Fig. 6. Computation of total energy of each seam

Therefore, we propose a new algorithm using an offset map to reduce the time complexity. Figure 6 shows how to calculate the total energy value of each seam using the offset map. Initialized values of the offset map and sum map are the same as the index map and the energy map each. At the first iteration, the number of CUDA blocks is equal to the half of the height of an image and each block manages two consecutive rows. Then threads of each block add the energy value of own pixel to the energy value of another pixel in the next row connected as a seam based on the offset map. Next, threads write the sum of the energy value of two pixels to the sum map, and update own offset value with the offset value of another pixel in the offset map for the next iteration. The number of CUDA blocks in the second step is reduced to the half of the first step and each block manages two-line apart rows. According to this method, the number of block of CUDA is the half of the second step in the third iteration, and each block manages four-line apart rows. This procedure continues till only one row is remained, and finally the first row of the sum map has the cumulative energy value in each seam. The number of iterations to be executed is *O(NlogN)*, if N is the height of an image. Therefore, this approach reduces time complexity $O(N^2)$ to *O(NlogN)*.

3) CPU-GPU hybrid: After obtaining the cumulative energy value of each seam, we select the optimal seam that has the minimum value between seams, and trace the path of the optimal seam from the first entry in the first row to the bottom of the image based on the index map. It means to access global memory that is corresponding the index map row by row, and it requires very high cost. The access latency between CPU and GPU's global memory is very high, and we have to copy data from GPU to CPU to make CPU trace a path. Hence we need to pipeline the data transfer between host and device and computation in both sides. We exploit CUDA concurrency characteristic to obtain an efficient workload division between CPU and GPU.

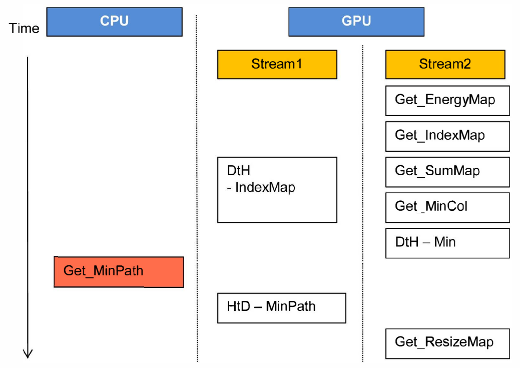


Fig. 7. CPU-GPU hybrid and concurrency method for seam carving

In Figure 7, after the index map is made, *stream1* starts to copy the index map to CPU, and *stream2* computes the sum map and finds the index value of the optimal seam that has the minimum energy sum at the same time. By doing this, the transfer time of index map is almost hidden. Then, the index value is transferred to CPU, and CPU starts to trace the path of the optimal seam based on the index map. CPU also needs to access DDR memory the times as much as the height of the image, but it takes much less time than GPU. Finally, the optimal seam path is copied into GPU to use it for resizing energy map.

B. Adaptive Multi-seam Algorithm

The original seam carving deletes one optimal seam at a time, and resizes the energy map to obtain newly reconstructed seams. Although this method is able to find the optimal seam per iteration, it takes too much time. Hence, we propose to sort the cumulated energy value of each seam to delete several seams in an ascending order at a time. We define the number of multi-seam first, then find the optimal seam using reduction method that is very faster than sorting algorithm, and reserve the indices of that optimal seam to another data set. After that, change the energy value of that optimal seam into maximum integer not to be selected as the optimal seam in the next iteration. Applying reduction method one more to find the second optimal seam, and this operation continues to find the number of seams same as the number of multi-seam we defined before. We tested several values of multi-seam, from 2 to 10, and 2 has good quality of resized image but achieves smaller speedup benefits respectively, otherwise, 10 causes too much distortion of most images in spite of dramatically improving the performance. 2 to 5 is proper for the most of images.

Multi-seam has good quality with fast speed, but we cannot easily decide which value of multi-seam is the best for an image. The large number of multi-seam is available in the beginning, but as the image is resized and the distance among objects in the image is closer, we need to carefully adjust the value of multi-seam to prevent too much distortion of objects in the image. Therefore, we propose an adaptive multi-seam method to define the number of seams to be deleted at a time automatically.

We find that the seams around the optimal seam are likely to be the optimal seam in the next iteration. Thus, at first, we set the maximum value of multi-seam and find first and the other optimal seams with above multi-seam method within the maximum value of the multi-seam we define, then determine the *distance* value and calculate the distance between the first optimal seam and other seams. If the distance of those seams is smaller than the *distance* we defined, we delete them at a time.

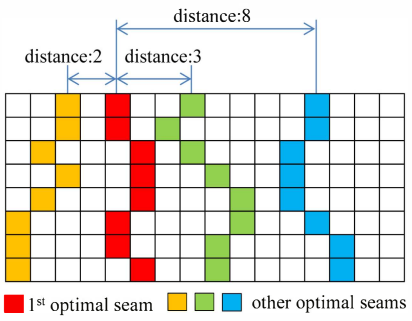


Fig. 8. Example of adaptive multi-seam method. If the red seam is the optimal seam and we set the *distance* value to 5, the orange and green seams are deleted with the red seam at a time, but the blue seam is remained because its distance value is greater than 5.

If we set the *distance* value as 5, all seams which the distance value from the optimal seam is less than 5 are deleted simultaneously within the maximum value of multi-seam (See Figure 8).

C. Multi-GPU Algorithm

In this section, we describe a multi-GPU implementation of seam carving on two GPUs. Since a host thread is only able to create a context of one GPU, we created two host threads using OpenMP to evaluate our experiments on two GPUs. Each GPU manages about half of the original data, and co-work with another CPU thread. We describe it below in detail.

1) Data partitioning: To implement seam carving on multi-GPU, the first challenge is how to separate the data of an image. At first, we vertically divide an image into two regions, namely upper half and lower half. Because each pixel requires data from surrounding pixels to calculate own energy value, the last row of upper half and the first row of lower half need one more additional row. In addition, to obtain the index map, each pixel of the last row in the upper half needs the energy value of the next row to select the minimum energy value, and it means that to compute the energy value of the next row requires one more next row. Therefore the upper half area needs two more rows than the half height of an image, on the other hand, one more row is needed for the lower half area. We set the size of two regions as the same in convenient, thus each GPU has the data as much as the half height of an image plus two rows (See Figure 9).

2) Reducing communication overhead between GPUs: After data partitioning, each GPU computes the energy map and index map at almost the same time, then transfers own index map to host using *stream1* to let host make an optimal seam path later. At this time, *stream2* of each GPU starts to calculate the sum map concurrently. Although each GPU is

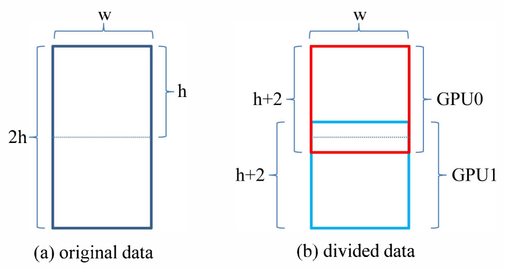


Fig. 9. Data partitioning for multi-GPU implementation

executed at a time to get own sum map, it manages half of each seam, so we should combine each result of the sum map from two GPUs to obtain the cumulative energy value of each seam. Since NVIDIA supports peer-to-peer access between GPUs, a GPU directly accesses the result of the sum map of another GPU and adds it to own sum map, then find the index of the optimal seam which has the lowest cumulative energy value. The GPU sends the index value to host, and the master thread in host starts to trace the optimal seam path. After that, the master thread distributes the data of optimal seam path to each GPU, then individual GPU executes the resizing work for the energy map. Figure 10 shows the block diagram for all of these processes.

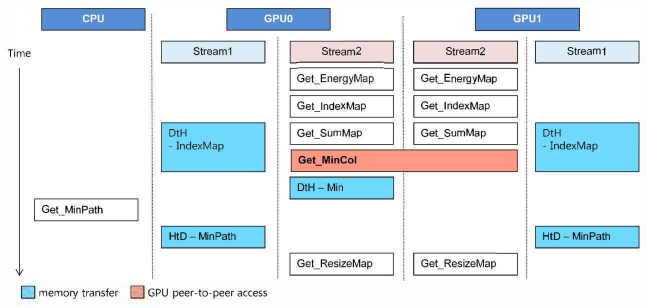


Fig. 10. CPU and multi-GPU implementation block diagram

V. Implementations for GPU Architecture

In this section, we describe several CUDA characteristics like memory management, branch overhead for block-thread model, and kernel launch overhead, then explain how these features affect the performance of seam carving.

A. Reducing memory transfer and allocation time

In order to use GPU, we have to allocate a space in host side first and transfer the data from CPU to GPU. Allocating memory in host is most often carried out using *malloc* function of C standard library. Another approach is to use *cudaMallocHost* provided by CUDA which offers *pinned memory* for high speed transfer between host and device memory [11]. We use this feature to allocate memory in the host to achieve higher throughput for moving the data of an image. However, allocating memory using *cudaMallocHost* is unfortunately very expensive than using *malloc* in spite of the advantage of reduced transfer time. R. Duarte et al. [10] included this feature to optimize the memory transfer time, but we evaluated the total time of both memory allocation and transfer for *malloc* and *cudaMallocHost* respectively, then reach a conclusion that the *pinned memory* provided by *cudaMallocHost* takes much more time than *page memory* offered by *malloc*. Moreover, seam carving algorithm does not need to frequently copy the image data between CPU and GPU, thus we decide to use *page memory* not *pinned memory*.

B. Coalescing global memory access

In CUDA programming, coalesced memory access to a global memory is one of the important rules to achieve high performance improvement. *cudaMallocPitch* guarantees that corresponding pointers in any row of an image will meet the alignment requirements for coalescing access to global memory. We use this feature with *cudaMemcpy2D* for each thread to be able to access to the global memory of alignment boundary.

C. Reducing computation workload

In general, thread divergence in CUDA will reduce parallel computation ability of block-thread model, thus there are various techniques to solve this problem. However, we still need to consider how much thread divergence affects system performance because avoiding it might cause more workload to computation. R. Duarte et al. [10] proposed a memory padding technique not to allow branch divergence in calculating an energy map, but this technique causes more computation workload. In seam carving, branch divergent problem only happens to the outer line of an image, and the speed of the energy map is affected by the amount of computation work. Therefore, we suggest allowing branch divergence for obtaining the benefit of the reduced computation work. Adding branch condition statement in calculating the energy value of the pixels in the outer line reduces the amount of computation work for those pixels up to 42%.

D. Reducing kernel launch overhead

The original seam map is not easy to parallelize because of its characteristic of dynamic programming. The basic version of our approach is much similar as R. Duarte et al. [10]. We divide each row of the image into horizontal tiles, and carefully select the width of the tile to utilize the maximum occupancy of the GPU resource. In this version, we must repeatedly call kernel function once per row to synchronize among threads on the different blocks because CUDA does not guarantee the completion time of each block inside a kernel. However, these recursive kernel calls cause vast kernel launch overhead and lead to performance degradation despite of the tile effect of each row. Therefore, we suggest using only one block to remove kernel launch overhead, and *for-loop* outside kernel is moved to inside kernel. We only call kernel function one time and use *__syncthreads* function to synchronize the work among threads for each row inside the kernel.

VI. Evaluation

In this section, we evaluate the performance and quality of output of GPU implementation. First of all, we verify NCSC by comparing the resized image of ours and original seam carving. Then, the results of performance improvement for various optimizations are presented. This section will give you answers to the following questions.

- What about the performance and quality of NCSC compared to original seam carving?
- How effective is using multi-GPU?
- how much performance improvement can we obtain by exploiting CUDA characteristics efficiently?

All of the experiments on single-GPU are evaluated on the 2GHz Intel Xeon CPU and NVIDIA Tesla K20c heterogeneous computer system. Tesla K20c has 13 SMs, 196 cores per SM, and 4.8GB global memory with CUDA computation capability of 3.5. Another system with 2.7GHz Intel Xeon CPU and NVIDIA Tesla K20m which has almost same features as K20c is utilized for multi-GPU implementation.

A. Verification of NCSC

We compare the resized image of NCSC with results from original seam carving method.

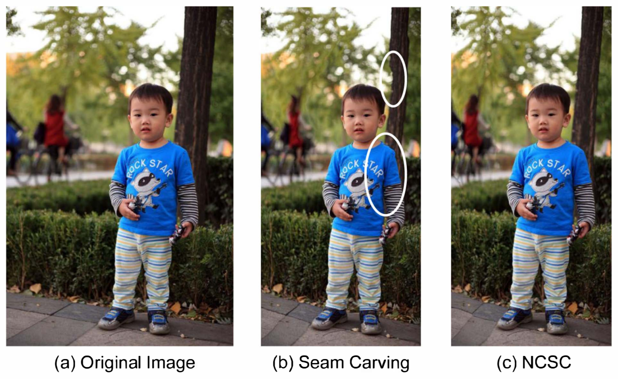


Fig. 11. Comparison of image resized by different methods. (a) Original image (683 × 1024). (b) Original seam carving (383 × 1024). (c) NCSC (383 × 1024).

In Figure 11(b), the left side of the tree is distorted, and the body of the child becomes narrower than the original image. Compared to original seam carving, NCSC (Figure 11(c)) can maintain the shape of the tree and the proportion of the child. Original seam caving considers cumulative energy level, and it might delete the important pixel having high energy value if the cumulative energy level of the seam which passes the pixel is low. In contrast, although NCSC can not guarantee to delete the seam that has the lowest energy level as original seam carving does, it can avoid to select the large energy value of the pixel because it just focus on the direction of each pixel. Through experiments, NCSC shows that it maintains the quality of the resized image on conserving contents of an image like structures and lines.

B. Results of Algorithm Optimizations

We evaluated NCSC and multi-seam method for the seam map (Figure 12). NCSC shows the performance improvement of 21.3× over the single-thread CPU implementation and removing the multiple seams at a time with NCSC presents dramatically high performance improvement, 39.4× and 171× speedup for using 2 and 10 *multi-seam* respectively.

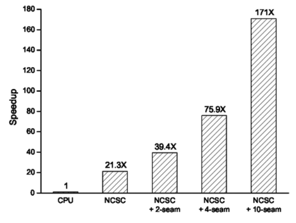


Fig. 12. Performance comparison of NCSC and multi-seam

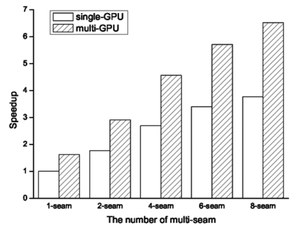


Fig. 13. Performance comparison of single and multiple GPU

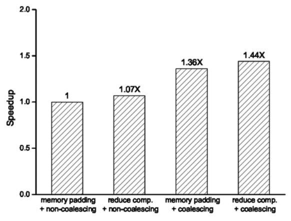


Fig. 15. Memory padding and reduced computation technique depend on memory access pattern

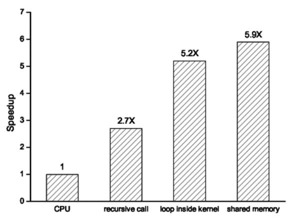
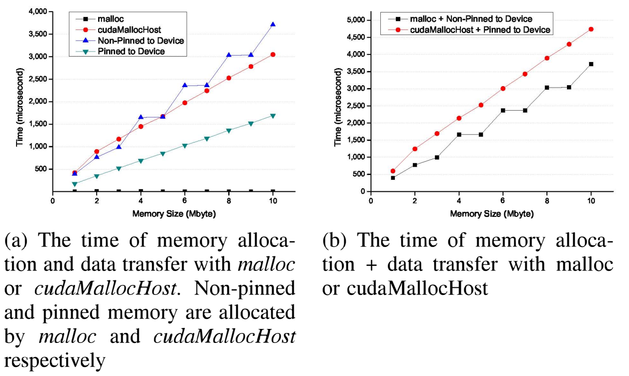


Fig. 16. Performance of seam map based on an original seam carving

Figure 13 shows the result of multi-GPU version. We use two NVIDIA Tesla K20m cards to implement and optimize multi-GPU experimentation. We have considered all of time for energy map, seam map, and resize map, even include the overhead of OpenMP primitives, barrier, for the serial execution part to find an optimal seam path. Figure 13 shows the speedup of the multi-GPU implementation over the single-GPU implementation with deleting a seam at a time. We achieve about 1.6× performance improvement than single-GPU version through the multi-GPU implementation.

C. Results of GPU Architecture Implementations

We measured the time for memory allocation of *malloc* and *cudaMallocHost* and the time for data transfer from CPU to GPU for each. Figure 14 presents the comparison results. Non-pinned memory refers to the memory allocated by *malloc* function and *cudaMallocHost* allocates a block of pinned memory. Although the data transfer time of pinned memory is 2.2× faster than non-pinned memory, the memory allocation time of the former is significantly slow, thus the total time of memory allocation and transfer shows that non-pinned memory is more efficient than pinned memory.



(a) The time of memory allocation and data transfer with *malloc* or *cudaMallocHost*. Non-pinned and pinned memory are allocated by *malloc* and *cudaMallocHost* respectively

(b) The time of memory allocation + data transfer with malloc or cudaMallocHost

Fig. 14. Comparison of the time for memory allocation and data transfer with non-pinned or pinned memory

Figure 15 shows the comparison results of memory padding and reduced computation technique with/without coalesced memory access. Reduced computation technique increases the performance improvement by 7% over the memory padding method, and we achieve 1.44× speedup using reduced computation technique with coalesced memory access over the memory padding technique with non-coalesced memory access.

Table I illustrates the performance speedup of two versions for the energy map on CUDA over single thread implementation. *Version1* consists of pinned memory, non-coalesced memory access, and memory padding technique, on the other hand, *Version2* adopts the use of non-pinned memory, coalesced memory access, and reduced computation method. *Version2* yields 231× performance improvement which is much higher than 146× of *Version1* over the single thread implementation. This results tell us that how much performance improvement we can achieve depends on how to use CUDA characteristics efficiently, as described in Section V.

Figure 16 represents the performance improvement of several CUDA versions based on original seam carving. First version using *recursive kernel call* method has 2.7× improvement over single-thread CPU implementation. To reduce kernel launch overhead in this version, we use *loop inside kernel* method and achieve 5.2× performance benefit. We can also see this version, despite using single block, is about 2× faster than *recursive kernel call* method using multiple blocks. In addition, applying *shared memory* to *loop inside kernel* version increases the speedup a little bit, 5.9×.

D. Execution Time

In the previous sections, we represent performance improvement for various optimization methods. However, to apply the seam carving to a real world application, total execution time should be evaluated including the time for copying the data from host to device and vice versa, and computing the energy map, seam map, and resize map.

Figure 17 illustrates the total time of 5 different versions with various image sizes. We measure the time to reduce the image size by 50%. CPU version takes huge time, and original seam caving method on GPU respectively needs less time than CPU, but it still takes several seconds (more than 4 seconds for 8 megabytes image). For NCSC, the total time is within a second, and multi-GPU implementation with deleting 2 seams at a time only takes about 45 milliseconds to reduce the size of a 1 MB image by a half and 270 milliseconds for an 8 MB image.

All experiments of this paper were performed on a desktop-scale computer not a mobile device, but with lighting trends in embedded systems industry, sooner or later, high performance GPU will be embedded into various personal devices. When the time comes, the method proposed in this paper can be used as an actual user application.

VII. Conclusion

Seam carving is a very powerful method for resizing an image. In general, content-aware image resizing is very efficient to sustain main content of the whole image with

TABLE I
COMPARISON OF TWO CUDA VERSIONS FOR ENERGY MAP

	host memory analysis		device memory analysis		workload analysis		speedup
	non-pinned memory	pinned memory	non-coalesced memory access	coalesced memory access	memory padding	reduce computation	
Version 1		✓	✓		✓		146.7
Version 2	✓			✓		✓	231.8

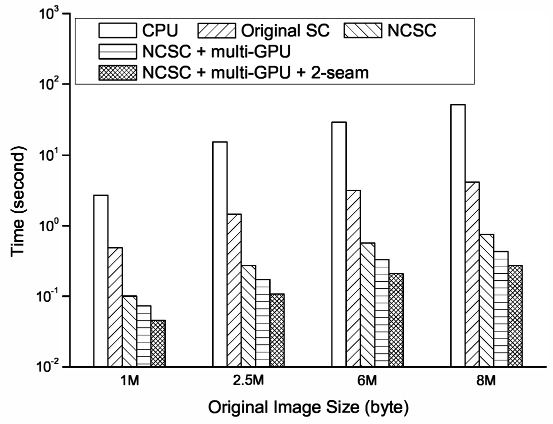


Fig. 17. Total time for all seam carving process, to copy from and to host, and to compute the energy map, seam map, and resize map to reduce the image width by 50%.

little distortion. However, seam carving not only has large computation amount but also needs dynamic programming, thus it is not proper to be applied to current personal devices.

In this paper, we analyzed the characteristics of seam carving and implemented it using CUDA-enabled GPU. We basically accomplished various optimization through exploring memory management method and CUDA characteristic analysis. Our new approach, NCSC (Non-Cumulative Seam Carving), achieved high performance enhancement by removing dynamic programming, reducing algorithm complexity and utilizing CPU-GPU hybrid parallel strategy while maintaining the quality of the resized images. In addition, *multi-seam* method brought significant performance improvement with little distortion of an image. Finally, we have achieved great speedup by implementing multi-GPU version with GPU peer-to-peer access feature.

ACKNOWLEDGMENT

We would like to thank anonymous reviewers for their insightful comments. We also thank PACMAN group members for their valuable feedback and suggestions. This work is supported by the National High-Tech Research and Development Plan (863 project) 2012AA010901, National 973 Basic Research Program 2014CB340402 and NSFC project 61103021.

REFERENCES

[1] A. Santella, M. Agrawala, D. DeCarlo, D. Salesin, and M. Cohen, “Gaze-based interaction for semi-automatic photo cropping,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’06. New York, NY, USA: ACM, 2006, pp. 771–780.

[2] Y. Pritch, E. Kav-Venaki, and S. Peleg, “Shift-map image editing,” in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 151–158.

[3] K. Thilagam and S. Karthikeyan, “An efficient method for content aware image resizing using psc,” *International Journal of Computer Technology and Applications*, vol. 2, no. 4, 2011.

[4] S. Avidan and A. Shamir, “Seam carving for content-aware image resizing,” *ACM Trans. Graph*, vol. 26, no. 3, p. 10, 2007.

[5] N. Hill and P. Eslambolchilar, “Seam carving for enhancing image usability on mobiles,” in *Proceedings of the 22Nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 2*, ser. BCS-HCI ’08. Swinton, UK, UK: British Computer Society, 2008, pp. 131–134.

[6] ARM, “Mali OpenCL SDK v1.1.0 Documentation,” http://malideveloper.arm.com/develop-for-mali/tutorials-developer-guides/sdk-tutorials/mali-opencl-sdk-tutorial/.

[7] B. Suh, H. Ling, B. B. Bederson, and D. W. Jacobs, “Automatic thumbnail cropping and its effectiveness,” in *UIST*. ACM, 2003, pp. 95–104.

[8] K. Thilagam and S. Karthikeyan, “Article: Optimized image resizing using piecewise seam carving,” *International Journal of Computer Applications*, vol. 42, no. 14, pp. 24–30, March 2012, published by Foundation of Computer Science, New York, USA.

[9] A. Mansfield, P. Gehler, L. Van Gool, and C. Rother, “Visibility maps for improving seam carving,” in *Proceedings of the 11th European Conference on Trends and Topics in Computer Vision - Volume Part II*, ser. ECCV’10. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 131–144.

[10] R. Duarte and R. Sendag, “Accelerating and characterizing seam carving using a heterogeneous cpu-gpu system,” *PDPTA*, 2012.

[11] NVIDIA, “Cuda C Programming Guide,” http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[12] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, “Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU,” in *Proc. 37th International Symposium on Computer Architecture (37th ISCA’10)*. Saint-Malo, France: ACM SIGARCH, Jun. 2010, pp. 451–460.

[13] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, “Accelerating CUDA graph algorithms at maximum warp,” in *Proceedings of the 16th ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming (16th PPOPP’11)*. San Antonio, TX, USA: ACM Press, Feb. 2011, pp. 267–276.

[14] M. Harris, “Optimizing parallel reduction in cuda (2007),” *CUDA SDK Whitepaper*, 2007.

[15] J. M. Cebrian, G. D. Guerrero, and J. M. Garcia, “Energy efficiency analysis of GPUs,” in *Proc. High-Performance, Power-Aware Computing – 3rd HPPAC’12, Proc. IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (26th IPDPS’12)*. Shanghai, China: IEEE Computer Society, May 2012, pp. 1014–1022.

[16] R. Achanta and S. Süsstrunk, “Saliency detection for content-aware image resizing,” in *ICIP*. IEEE, 2009, pp. 1005–1008.

[17] L.-Q. Chen, X. Xie, X. Fan, W.-Y. Ma, H. Zhang, and H.-Q. Zhou, “A visual attention model for adapting images on small displays,” *Multimedia Syst*, vol. 9, no. 4, pp. 353–364, 2003.

[18] G. Ciocca, C. Cusano, F. Gasparini, and R. Schettini, “Self-adaptive image cropping for small displays,” *IEEE Trans. Consumer Electronics*, vol. 53, no. 4, pp. 1622–1627, 2007.