A GPU Implementation of Clipping-Free Halftoning using the Direct Binary Search

Hiroaki Koge, Yasuaki Ito, and Koji Nakano

Department of Information Engineering, Hiroshima University,
Kagamiyama 1-4-1, Higashi Hiroshima 739-8527, Japan
{kouge,yasuaki,nakano}@cs.hiroshima-u.ac.jp

Abstract. Halftoning is an important process to convert a gray scale image into a binary image with black and white pixels. The clipping-free DBS (Direct Binary Search)-based halftoning is one of the halftoning methods that can generate high quality binary images. However, considering the computing time, it is not realistic for most applications such as printing purpose. The main contribution of this paper is to show a new GPU implementation for the clipping-free DBS-based halftoning. We have considered programming issues of the GPU architecture to implement the method on the GPU. The experimental result shows that our GPU implementation on NVIDIA GeForce GTX 780 Ti for a 4096×3072 gray scale image runs in 7.240 seconds, while the CPU implementation runs in 346.6 seconds. Thus, our GPU implementation attains a speed-up factor of 47.82.

Keywords: Image processing, Halftoning, Direct binary search, Clipping-free, GPGPU

1 Introduction

Recent Graphics Processing Units (GPUs), which have a lot of processing units, can be used for general purpose parallel computation. Since GPUs have very high memory bandwidth, the performance of GPUs greatly depends on memory access. CUDA (Compute Unified Device Architecture) [19] is the architecture for general purpose parallel computation on GPUs. Using CUDA, we can develop parallel algorithms to be implemented in GPUs. Therefore, many studies have been devoted to implement parallel algorithms using CUDA [5, 6, 8, 16, 22, 28, 29].

A gray scale image is a two dimensional matrix of pixels taking a real number in the range $[0, 1]$. Usually a gray scale image has 8-bit depth, that is, each pixel takes one of the real numbers $\frac{0}{255}, \frac{1}{255}, \ldots, \frac{255}{255}$, which correspond to pixel intensities. *A binary image* is also a two dimensional matrix of pixels taking a binary value 0 (black) or 1(white). *Halftoning* is an important process to convert a gray scale image into a binary image [2, 13, 17]. This process is necessary when a monochrome or color image is printed by a printer with limited number of ink colors.

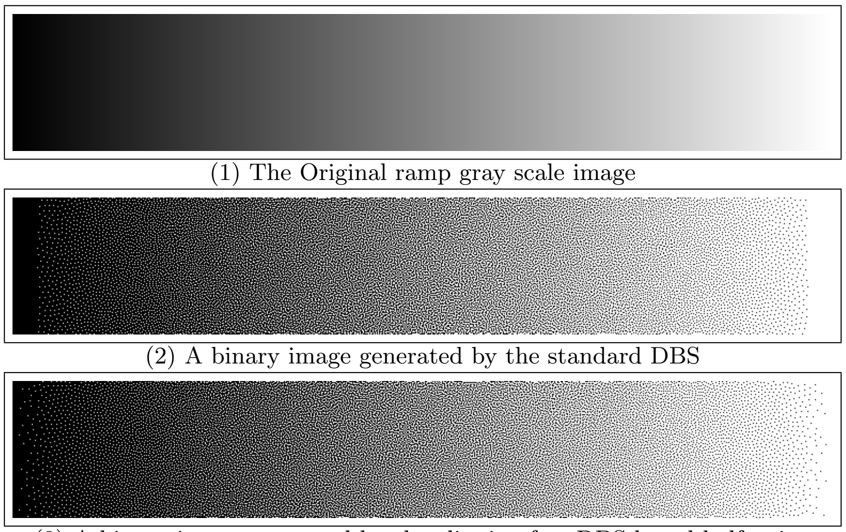
Many halftoning techniques including Error Diffusion [7], Dot Diffusion [12], Ordered Dither using the Bayer threshold array [3] and the Void-and-Cluster threshold array [26], Direct Binary Search (DBS) [1, 14], Local Exhaustive Search (LES) [10, 11], have been presented.

The Ordered Dither [3] uses a threshold array to generate a binary image from an original gray scale image. Each pixel of the original gray scale image is compared with an element of the threshold array. From the result of the comparison, the pixel value of the corresponding pixel of the binary image is determined. Binary images generated by the Ordered Dither method using the Bayer threshold array [3] have artifacts with periodic dots arranged in a two dimensional grid [27].

It is known that, in many cases, the DBS [1, 14] generates better quality images. The key idea of the DBS is to find a binary image whose projected image onto human eyes is very close to the original image. The projected image is computed by applying a Gaussian filter, which approximates the characteristic of the human visual system. Let the total error of the binary image be the sum of the differences of the intensity levels over all pixels between the original image and the projected image. In the DBS, a pixel value is toggled if the resulting image has smaller total error. Also, neighboring pixel values are swapped if the total error of the resulting image decreases. The DBS generates a sharp binary image, especially, for middle tone areas. However, the generated binary image by the DBS has no tone in highlights and shadows. Fig. 1 and 2 show the binary images generated by the DBS. The resulting image has *clippings*, that is, highlights and shadows have no minority pixels and lose the tone of the original image. For example, several columns from the leftmost of Fig. 1 have no white pixel, although the original image has tone. Also, there is no black pixel in several columns from the rightmost and a pseudo border line appears.

For most printing devices, black pixels gain by dot-gain [11]. In other words, the average intensity level of the actual printed image is smaller than that of a binary image used for printing. If this is the case, the intensity levels of an original gray scale image are calibrated such that the actual printed image reproduces the intensity of the original gray scale image correctly. For example, suppose that intensity level 254/255 of an original gray scale image is adjusted to 1023/1024. After that, the adjusted gray scale image is converted to the binary image. Clearly, the binary image thus obtained have fewer black pixels and the average intensity is 1023/1024. However, black pixels gain by dot-gain, and the intensity level of the actual printed image will increase to 254/255. This means that, halftoning methods are required to generate the binary image with average intensity level 1023/1024. If this is not possible, actual printed images cannot reproduce intensity level 254/255, and should have tone jumps.

To avoid the clipping generated by the DBS, in [30], a DBS-based halftoning method, called *the clipping-free DBS-based halftoning* was proposed. The method is based on the DBS and can generate clipping-free binary images. The key idea of the method is to apply the Ordered Dither method using a threshold array generated by the DBS to highlights and shadows of an original gray scale image.



(1) The Original ramp gray scale image

(2) A binary image generated by the standard DBS

(3) A binary image generated by the clipping-free DBS-based halftoning

Fig. 1. The resulting halftone images for the ramp image

In the method, minority pixels are preserved, that is, black pixels in the highlights and white pixels in the shadow areas, and apply DBS to the whole image. The resulting binary images have no clipping and reproduce the original tones very well. Further, the DBS-based halftoning preserves the linearity of intensity levels. In general, visually pleasing halftone textures are perceived as smooth, contain a large variety of patterns, and exhibit accurate tone rendition [15]. In other words, the resulting binary images also have high texture quality.

The resulting halftoned images generated by the DBS halftoning and the clipping-free DBS-based halftoning have high texture quality. However, compared with other well-known halftone methods, such as the Error diffusion, much more computing time is necessary. To accelerate the computation, therefore, several GPU implementations of the DBS has been proposed [4, 24, 25]. In [25], a GPU implementation of the DBS were proposed. Also, the implementation was extended to halftoning for color images [4] and multi-toning [24]. However, as far as we know, there is no implementation of the clipping-free DBS-based halftoning. The main contribution of this paper is to show a new GPU implementation for the clipping-free DBS-based halftoning. We have considered programming issues of the GPU architecture to implement the method. The experimental result shows that our GPU implementation on NVIDIA GeForce GTX 780 Ti for a 4096×3072 gray scale image runs in 7.240 seconds, while the CPU implementation runs in 346.6 seconds. Thus, our GPU implementation attains a speed-up factor of 47.82. Considering the computing time and the resulting clipping-free binary images, our GPU implementation is more realistic for most applications such as printing purpose.

2 Review of the Clipping-free DBS-based halftoning

The main purpose of this section is to introduce the Ordered Dither [3, 26] and the Direct Binary Search [1, 14], and review the clipping-free DBS-based halftoning method with them as key ingredients [30].

2.1 The Ordered Dither and the Direct Binary Search

Suppose that an original gray scale image $A = (a_{i,j})$ of size $n \times n$ is given[1], where $a_{i,j}$ denotes the intensity level at position (i, j) $(0 \le i, j \le n-1)$ taking a real number in the range $[0, 1]$. The goal of halftoning is to find a binary image $B = (b_{i,j})$ of the same size that reproduces the original image A, where each $b_{i,j}$ is either 0(black) or 1(white). The Ordered Dither uses a threshold array $T = (t_{i,j})$ of size $m \times m$, with each element taking a real number $\frac{0}{255}$, $\frac{1}{255}$, $\ldots$, or $\frac{254}{255}$. More specifically, the pixel value of each pixel $b_{i,j}$ is determined by the following formula:

$$b_{i,j} = \begin{cases} 0 \text{ if } a_{i,j} \le t_{i \bmod m, j \bmod m} \\ 1 \text{ if } a_{i,j} > t_{i \bmod m, j \bmod m} \end{cases}$$

Note that since $b_{i,j}$ is always 0 if $t_{i \bmod m, j \bmod m} = \frac{255}{255}$, the threshold value never takes $\frac{255}{255}$. The Bayer halftoning uses the Bayer threshold array which is defined recursively [3].

The idea of the DBS is to measure the goodness of the output binary image B using the Gaussian filter that approximates the characteristic of the human visual system. Let $V = (v_{k,l})$ denote a Gaussian filter, i.e. a 2-dimensional symmetric matrix of size $(2w+1) \times (2w+1)$, where each non-negative real number $v_{k,l}$ $(-w \le k, l \le w)$ is determined by a 2-dimensional Gaussian distribution such that their sum is 1. In other words,

$$v_{k,l} = c \cdot e^{-\frac{k^2+l^2}{2\sigma^2}}, \qquad (1)$$

where σ is a parameter of the Gaussian distribution and c is a fixed real number to satisfy $\sum_{-w \le k,l \le w} v_{k,l} = 1$. Let $R = (r_{i,j})$ be the projected gray scale image of a binary image $B = (b_{i,j})$ obtained by applying the Gaussian filter as follows:

$$r_{i,j} = \sum_{-w \le k,l \le w} v_{k,l} b_{i+k,j+l} \quad (0 \le i, j \le n-1). \qquad (2)$$

Clearly, from $\sum_{-w \le k,l \le w} v_{k,l} = 1$ and $v_{k,l}$ is non-negative, each $r_{i,j}$ takes a real number in the range $[0, 1]$ and thus, the projected image R is a gray scale image. We can say that a binary image B is a good approximation of original image A if the difference between A and R is small enough. Hence, we define the error of B as follows. The error $e_{i,j}$ at each pixel location (i, j) is defined by

$$e_{i,j} = a_{i,j} - r_{i,j} \qquad (3)$$

[1] For simplicity, we assume that images are square.

and the total error is defined by

$$Error(A, B) = \sum_{0 \le i,j \le n-1} |e_{i,j}|^2. \qquad (4)$$

Since the Gaussian filter approximates the characteristics of the human visual system, we can think that image B reproduces original gray scale image A if $Error(A, B)$ is small enough. The best binary image that reproduces A is a binary image B which is given by the following formula:

$$B^* = \arg\min_B Error(A, B). \qquad (5)$$

It is very hard to find the optimal binary image B^* for a given gray scale image A. The idea of the DBS is to find a near optimal binary image B such that $Error(A, B)$ is sufficiently small. For this purpose, the DBS repeats improvement of binary image B. The value of a particular pixel $b_{i,j}$ is modified by the following two operations:

Toggling This operation is to toggle the value of $b_{i,j}$, that is, $b_{i,j} \leftrightarrow 1 - b_{i,j}$. The value of $b_{i,j}$ is toggled if $Error(A, B)$ decreases.

Swapping Let $b_{i',j'}$ be a neighbor pixel of $b_{i,j}$, that is, both $|i - i'| \le 1$ and $|j-j'| \le 1$ hold. This operation is to exchange the values of $b_{i,j}$ and $b_{i',j'}$, that is $b_{i,j} \leftrightarrow b_{i',j'}$. Swapping operation is performed if $Error(A, B)$ decreases.

Clearly, toggling and swapping operations do not increase the error and improve the binary image B. In the DBS, these operations are executed in the raster order. Further, this raster order improvement is repeated until no more improvement by toggling or swapping operations is possible.

Although the DBS generates high-quality binary images, it does not work well in highlights and shadows. It has *clippings*, that is, the highlight and the shadow areas have no dots and lose the tone of the original image. Fig. 1 shows the resulting binary images by the DBS. The left shadow area has no white dots and the tone is lost. Similarly, in the right highlight area, black dots disappear. Fig. 2 shows the resulting binary image generated by the DBS. Black dots in the woman's face are lost and pseudo borders appear. Also, white dots in her hair are removed. In [30], the detailed explanation about the reason of clippings by the DBS is shown.

2.2 The Clipping-free DBS-based halftoning

In the following, we review the clipping-free DBS-based halftoning proposed in [30]. The key idea of this DBS-based halftoning method is to use the Ordered Dither for the pixels with intensity smaller than D or larger than $1-D$ and then use the DBS.

First a threshold array $T = (t_{i,j})$ of size $m \times m$ used for shadows is determined. Since this T is used for the pixel values no more than D, it is not necessary to determine threshold value larger than D. Thus, for each i $(0 \leq i \leq D)$, $\frac{m^2}{255}$ elements in T takes value $\frac{i}{255}$. For highlight pixel with intensity larger than $1-D$, we can use a threshold array $T' = (t'_{i,j})$ such that $t'_{i,j} = 1 - t_{i,j}$.

The goal of determining T is to distribute the threshold values in T uniformly. The uniformity is defined as follows. Let $u(i,j)$ denote the Euclidean distance to a closest threshold value no more than $t_{i,j}$. In other words,

$$u(i,j) = \min\{\sqrt{(i-i')^2 + (j-j')^2} \mid t_{i',j'} \leq t_{i,j}\}.$$

The uniformity $u(T)$ of T is the sum of $u_{i,j}$, that is,

$$u(T) = \sum_{0 \leq i,j \leq m-1} u(i,j).$$

Clearly, if threshold value distributed more uniformly, the uniformity $u(T)$ is larger.

The threshold value is assigned $\frac{0}{255}$ to $\frac{m^2}{255}$ elements in T. For this purpose, we select $\frac{m^2}{255}$ elements in T at random and assign $\frac{0}{255}$ to them. After that, we move each $\frac{0}{255}$ to a neighbor element if the uniformity $u(T)$ increases. The reader should have no difficulty to confirm that, this operation is very similar to swapping operation of the DBS. This swapping operation is repeated until no more improvement on $u(T)$ is possible. Next, we assign threshold value $\frac{1}{255}$ to $\frac{m^2}{255}$ elements in T. Similarly, we select $\frac{m^2}{255}$ elements that were not assigned $\frac{0}{255}$ and assign $\frac{1}{255}$ to them. After that, the swapping operation is performed for these elements with threshold value $\frac{1}{255}$. The same procedure is repeated until threshold values from $\frac{0}{255}$ to D are determined. If T is small, the generated binary image may have periodic artifact with frequency $m \times m$ pixels. Thus, T should be as large as possible. In the experiment [30] a threshold array of size 512×512 is large enough.

We are now in position to explain the clipping-free DBS-based halftoning. Suppose that a gray scale image $A = (a_{i,j})$ to be halftoned is given. We first apply the threshold array T to pixels $a_{i,j}$ of A such that $a_{i,j} < D$ or $a_{i,j} > 1 - D$, and obtain a binary image $B = (b_{i,j})$. Next, we assign label *determined/undetermined* to every pixel as follows:

$b_{i,j}$ is *determined*, if $(a_{i,j} < D$ and $b_{i,j} = 1)$ or $(a_{i,j} > 1-D$ and $b_{i,j} = 0)$, and
$b_{i,j}$ is *undetermined*, otherwise.

In other words, if $b_{i,j}$ is minority pixel in shadows or in highlights, then it is a determined pixel. After that, the DBS is executed for all undetermined pixels, that is, toggling and swapping operations repeated in the raster scan order until no more improvement of the error is possible.

According to the above, the outline of the clipping-free DBS-based halftoning that computes a halftoned binary image of an original gray scale image A is as follows:

[The clipping-free DBS-based halftoning]

Step 1: Initialization

We first assign label determined/undetermined to every pixel by applying a threshold array T. We obtain a binary image $B = (b_{i,j})$ such that

if label is determined, $b_{i,j}$ is halftoned by a threshold array T, and

if label is undetermined, $b_{i,j}$ is halftoned by *the random dither method*.

In the random dither method, a binary pixel takes value 1 with probability p if the pixel value of the corresponding pixel of an original image is p ($\in [0, 1]$). Thus, $b_{i,j} = 1$ with probability $a_{i,j}$ for every i and j except determined pixels. In Step 2, determined pixels are fixed and the DBS is performed for undetermined pixels.

Step 2: DBS

We pick an undetermined pixel $b_{i,j}$ in B one by one from the top-left corner to the bottom-right corner in the raster scan order. We select one of the operations toggling and swapping, which minimizes the total error, and update pixels by such operation. This update procedure by the raster scan order is repeated until one round of raster scan search from the top-left corner to the bottom-right corner does not update pixels and the error is not improved.

Fig. 1 and 2 show the resulting binary images. To obtain these images, we have generated a threshold array of size 512×512. We have also used the Gaussian filter with parameter $\sigma = 1.2$ and $w = 4$, and the threshold value $D = \frac{9}{255}$ is used. We can see clearly the original tone is preserved in shadows and highlights. The resulting images look smooth and contain more variety of patterns. They take on good visual texture.

3 Implementation of the clipping-free DBS-based halftoning

Before explaining our GPU implementation of the clipping-free DBS-based halftoning, in this section, we show how the clipping-free DBS-based halftoning is implemented as a sequential implementation.

In Step 1, we first assign label determined/undetermined to every pixel using a threshold array T and make a label map $L = (l_{i,j})$ of size $n \times n$ such that if $l_{i,j} = 1$ if label determined is assigned to pixel (i, j), and $l_{i,j} = 0$, otherwise. Referring L, we obtain a binary image B for determined pixels and undetermined pixels by thresholding with T and the random dither method, respectively.

In Step 2, we first compute the projected gray scale image $R = (r_{i,j})$ of the binary image B by computing formula (2). We compute the error matrix E by computing formula (3) and the total error from formula (4). In Step 2, we need to perform local search by toggling and swapping that minimize the total error. It is sufficient to compute the total error of the affected region that includes the neighboring 8 pixels. The affected region is a region of the image B such that the Gaussian filter for $b_{i,j}$ and its neighboring 8 pixels affects the pixel values of the blurred image. More specifically, the affected region is a set $\mathcal{A}_{i,j}$ of positions in the image such that

$$\mathcal{A}_{i,j} = \{(i', j') | i - w - 1 \leq i' \leq i + w + 1, j - w - 1 \leq j' \leq j + w + 1\}.$$

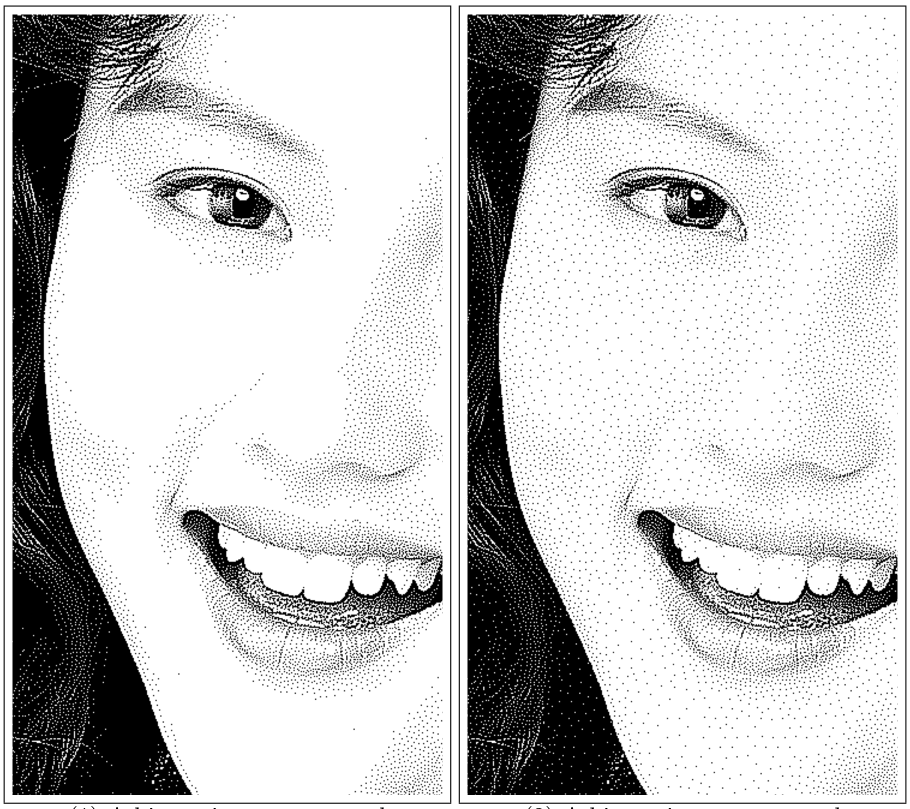

(1) A binary image generated by the standard DBS
(2) A binary image generated by the clipping-free DBS

Fig. 2. The resulting binary images (partial enlargement) for a woman image [9]

Since the size of the Gaussian filter is $(2w+1) \times (2w+1)$, that of the affected region is $(2w+3) \times (2w+3)$. Therefore, in the local search by toggling and swapping, we compute the total error at pixel location (i,j) by evaluating the following formula:

$$\sum_{(i',j') \in \mathcal{A}_{i,j}} |e_{i',j'}|^2. \tag{6}$$

We evaluate this formula for each operation of toggling and swapping, and replace pixels with the minimum total error.

To perform the local search by toggling and swapping, we need to compute the convolution in formula (2) for each operation of toggling and swapping. Since pixel values of B are 0 or 1, $r_{i,j}$ can be computed by adding/subtracting the values of the Gaussian filter $v_{k,l}$ to/from $r_{i,j}$ in $\mathcal{A}_{i,j}$. For example, when a pixel $b_{i,j}$ is changed from 0 to 1 by toggling, the values of the Gaussian filter $v_{k,l}$ are only added to $r_{i,j}$ in $\mathcal{A}_{i,j}$. We note that once the total error E is computed, the update of E by toggling and swapping can be also computed by adding/subtracting the values of the Gaussian filter. It can be performed without the update of the projected image R. Therefore, in our implementation, we directly update the total error E.

To obtain further acceleration, we use an update map. In Step 2, a round of the raster scan order search is repeated. It is possible that an area of a binary image is fixed in an earlier round, and no pixels in the area are not updated until Step 2 terminates. Hence, it makes sense to perform the local search by toggling and swapping for which pixels might be updated, we use an update map $M = (m_{i,j})$ of size $n \times n$. Before a round of the raster scan order search, all values in M is initialized by 0. We set $m_{i,j} = 1$ if the operation updates pixel $b_{i,j}$, that is, the value of $b_{i,j}$ is changed from 0 to 1 or from 1 to 0. Clearly, at the end of the round, $m_{i,j} = 1$ if $b_{i,j}$ has been updated in this period. Further, the affected region in which pixels might be updated in the next round consists of (i,j) such that $m_{i,j}$ or its neighbor takes value 1.

4 GPU Implementation

The main purpose of this section is to show our GPU implementation of the clipping-free DBS-based halftoning.

We briefly explain CUDA architecture that we will use. NVIDIA provides a parallel computing architecture called *CUDA* on NVIDIA GPUs. CUDA uses two types of memories in the NVIDIA GPUs: *the global memory* and *the shared memory* [20]. The global memory is implemented as an off-chip DRAM of the GPU, and has large capacity, say, 1.5-6 Gbytes, but its access latency is very long. The shared memory is an extremely fast on-chip memory with lower capacity, say, 16-48 Kbytes. Fig. 3 illustrates the CUDA hardware architecture.

CUDA parallel programming model has a hierarchy of thread groups called *grid*, *block* and *thread*. A single grid is organized by multiple blocks, each of

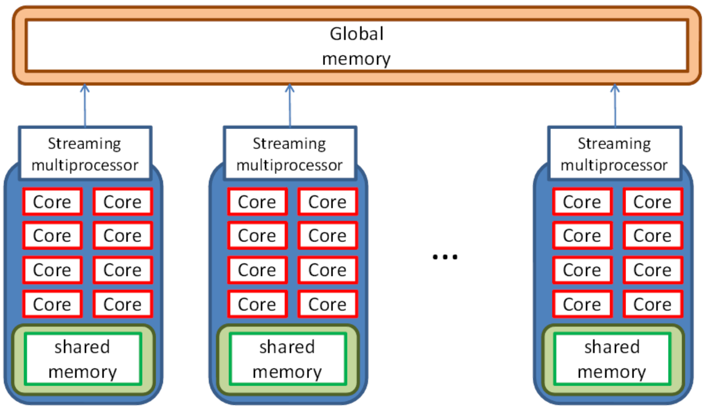


Fig. 3. CUDA hardware architecture

which has equal number of threads. The blocks are allocated to streaming multiprocessors such that all threads in a block are executed by the same streaming multiprocessor in parallel. All threads can access to the global memory. However, threads in a block can access to the shared memory of the streaming multiprocessor to which the block is allocated. Since blocks are arranged to multiple streaming multiprocessors, threads in different blocks cannot share data in the shared memories.

We are now in a position to explain how we implement the clipping-free DBS-based halftoning. We assume that an original gray scale image A of size $n \times n$ to be halftoned is stored in the global memory in advance, and the implementation writes the resulting binary image B' in the global memory. Further, we assume that the threshold array T is also stored in the global memory. In the following, to perform the computation in parallel, basically we divide an input image into subimages whose size is $k \times k$ and perform halftoning for each subimage in parallel. In our implementation, the size of the subimage is $\frac{n}{k} \times \frac{n}{k}$.

To implement Step 1, $\frac{n^2}{k^2}$ CUDA blocks are invoked one for subimages of size $\frac{n}{k} \times \frac{n}{k}$. Each CUDA block is responsible for generating an initial binary image $B = (b_{i,j})$ and computing the error matrix $E = (e_{i,j})$ of the corresponding subimage using the shared memory. For this purpose, each block copies pixel values in A of their affected region $\mathcal{A}_{i,j}$ in the subimage to the shared memory. After that, threads in each block concurrently assign label determined/undetermined to every pixel and generate an initial binary image for determined/undetermined pixels by thresholding with T and the random dither method, respectively. Finally, the error matrix $E = (e_{i,j})$ of the corresponding block is computed from the blurred image of B and pixel values in A of the affected region $\mathcal{A}_{i,j}$. The error matrix E of the resulting block is copied to the global memory.

In Step 2, a kernel is invoked for each round in the DBS. In each kernel, the DBS to evaluate formula (5) is performed in parallel using multiple CUDA

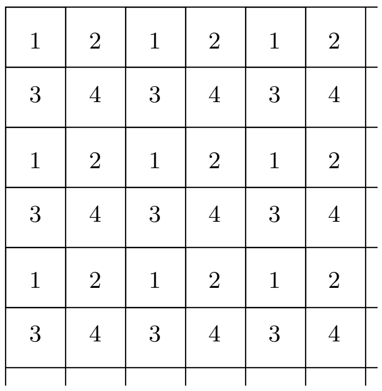


Fig. 4. Groups of blocks

blocks. Each CUDA block is responsible for executing the DBS of the corresponding subimage. The local search in the DBS is performed for pixels that are assigned to label undetermined and might be updated by referring label map L and update map M.

However, the operations toggling and swapping for adjacent blocks cannot be executed in parallel, because the application of the Gaussian filter to adjacent blocks affects each other. Thus, we partition blocks into four groups such that Group 1: even columns and even rows, Group 2: odd columns and even rows, Group 3: even columns and odd rows, and Group 4: odd columns and odd rows. The reader should refer to Fig. 4 illustrating the groups. We use $\frac{4n^2}{k^2}$ CUDA blocks, and perform the local search in all blocks of each group. Note that, if $k \geq 2w$ then the Gaussian filter of two blocks in a group never affect each other, where the subimage is $k \times k$ and the size of the Gaussian filter is $(2w+1) \times (2w+1)$. In other words, the affected regions of a particular group do not overlap each other. Actually, in our experiment, we choose $k = 32$ and $w = 4$. Step 2 performs the local search for Group 1, Group 2, Group 3, and Group 4, in turn. A CUDA block is invoked for each block of a group. The CUDA block copies the error matrix corresponding to the affected region in the global memory to the shared memory.

After that, each CUDA block performs the local search by toggling and swapping for the corresponding subimage in raster scan order to obtain the best combination of pixels in B. Concretely, multiple threads in a block perform the local search pixel by pixel for the corresponding subimage in the raster scan order. In the local search, formula (6) is evaluated for each operation of toggling and swapping. In our implementation, we utilize a summing technique by binary reduction proposed in [18] to evaluate the formula.

Finally, the updated binary image and the error matrix are copied to the global memory. Some readers may think that since the local search is concurrently performed using the partition shown in Fig. 4, the total error by computing formula (4) increases compared with that by the sequential one. However, in our experiment, the total errors are almost the same and the quality of the resulting binary images cannot be distinguished.

5 Experimental results

The main purpose of this section is to show the experimental results. We have used three gray scale images, Lena [23], Woman [9], and Flower basket [9], of size 512×512, 2048×2560, and 4096×3072, respectively. We use the Gaussian filter with parameter $\sigma = 1.2$ and $w = 4$. In the clipping-free DBS-based halftoning, we have generated a threshold array of size 512×512 and the threshold value $D = \frac{9}{255}$ is used.

In order to evaluate the computing time for generating the halftoned images, we have used NVIDIA GeForce GTX 780 Ti, which has 2880 processing cores in 15 SMX units [21]. We have also used Intel PC using Xeon X7460 running in 2.66GHz to evaluate the implementation by sequential algorithms. Table 1 shows the computing time for generating the binary images. The computing time is average of 10 times execution and the computing time of the GPU includes data transfer time between the main memory and the device memory in the GPU. Using the GPU, the computing time can be reduced by a factor of 35.88-47.82. Even if the large image is halftoned, the computing time is 7.240s by the GPU acceleration. This computing time is acceptable for most applications such as printing purpose.

Table 1. Computing time (in seconds) of the clipping-free DBS-based halftoning

Image	Lena	Woman	Flower basket
(size)	(512×512)	(2048×2560)	(4096×3072)
Intel CPU	8.351	113.3	346.6
NVIDIA GPU	0.2138	3.158	7.248
Speed-up	39.06	35.88	47.82

Additionally, according to the result of an existing GPU implementation of the original DBS, they reported that the execution time for two gray scale images of size 194×270 and 1536×1920 was 9.36s and 127s, respectively [25]. Since utilized GPUs and images differ and their implementation does not support clipping-free halftoning, it is difficult to compare their results with our results directly. Considering the computing time, however, it is clear that our GPU implementation is better than that of [25].

6 Conclusions

In this paper, we have proposed an implementation of the clipping-free DBS-based halftoning. In our implementation, we have considered programming issues of the GPU architecture. We have implemented it on NVIDIA GeForce GTX 780 Ti. The experimental result shows that our GPU implementation on NVIDIA GeForce GTX 780 Ti for a 4096×3072 gray scale image runs in 7.240 seconds, while the CPU implementation runs in 346.6 seconds. Thus, our GPU implementation attains a speed-up factor of 47.82. According to the results, we think that the computing time of our GPU implementation is realistic for most applications such as printing purpose.

References

1. Analoui, M., Allebach, J.: Model-based halftoning by direct binary search. In: Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology. vol. 1666, pp. 96–108 (1992)
2. Asano, T., Nakano, K.: Halftoning through optimization of restored images – a new approach with hardware acceleration. Tech. rep., The Institute of Electronics Information and Communication Engineers, COMP2002-75 (March 2003)
3. Bayer, B.: An optimum method for two-level rendition of continuous-tone pictures. In: IEEE International Conference on Communications. pp. 11–15 (1973)
4. Chandu, K., Stanich, M., Trager, B., Wu, C.W.: A GPU implementation of color digital halftoning using the Direct Binary Search algorithm. In: Proc. of IEEE International Symposium on Circuits and Systems. pp. 185–188 (2012)
5. Diaz, J., Muñoz-Caro, C., Niño, A.: A survey of parallel programming models and tools in the multi and many-core era. IEEE Transactions on Parallel and Distributed Systems 23(8), 1369–1386 (August 2012)
6. Farivar, R., Rebolledo, D., Chan, E., Campbell, R.H.: A parallel implementation of k-means clustering on GPUs. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. pp. 340–345 (July 2008)
7. Floyd, R., Steinberg, L.: An adaptive algorithm for spatial gray scale. In: Proc. of International Symposium Digest of Technical Papers, Society for Information Displays. pp. 36–37 (1975)
8. Harish, P., Narayanan, P.J.: Accelerating large graph algorithms on the GPU using CUDA. In: Proceedings of the 14th International Conference on High Performance Computing. pp. 197–208 (2007)
9. ISO/IEC International Standard 12640: Graphic technology – prepress digital data exchange – CMYK standard colour image data (CMYK/SCID) (1997)
10. Ito, Y., Nakano, K.: FM screening by the local exhaustive search with hardware acceleration. International Journal on Foundations of Computer Science 16(1), 89–104 (2005)
11. Ito, Y., Nakano, K.: A new FM screening method to generate cluster-dot binary images using the local exhaustive search with FPGA acceleration. International Journal on Foundations of Computer Science 19(6), 1373–1386 (2008)
12. Knuth, D.: Digital halftones by dot diffusion. ACM Transactions on Graphics 6(4), 245–273 (1987)

13. Lau, D.L., Arce, G.R.: Modern Digital Halftoning. Marcel Dekker (2001)
14. Lieberman, D.J., Allebach, J.P.: Efficient model based halftoning using direct binary search. In: Proc. of International Conference on Image Processing. vol. 1, pp. 775–778 (1997)
15. Lieberman, D.J., Allebach, J.P.: A dual interpretation for direct binary search and its implications for tone reproduction and texture quality. IEEE Transactions on Image Processing 9(11), 1950–1963 (2000)
16. Man, D., Uda, K., Ueyama, H., Ito, Y., Nakano, K.: Implementations of parallel computation of Euclidean distance map in multicore processors and GPUs. In: Proceedings of International Conference on Networking and Computing. pp. 120–127 (2010)
17. Nakano, K.: Various screening methods. Convertech 36(1), 72–77 (2008)
18. Nakano, K.: Optimal parallel algorithms for computing the sum, the prefix-sums, and the summed area table on the memory machine models. IEICE Transactions on Information and Systems E96-D(12), 2626–2634 (December 2013)
19. NVIDIA Corporation: CUDA ZONE. http://www.nvidia.com/page/home.html
20. NVIDIA Corporation: CUDA C Programming Guide Version 5.5 (2013)
21. NVIDIA Corporation: NVIDIA next generation CUDA compute architecture: Kepler GK110 whitepaper (2013)
22. Ogawa, K., Ito, Y., Nakano, K.: Efficient Canny edge detection using a GPU. In: International Workshop on Advances in Networking and Computing. pp. 279–280 (Nov 2010)
23. Po, L.M.: Lenna 97: A complete story of Lenna. http://www.ee.cityu.edu.hk/~lmpo/lenna/Lenna97.html (2001)
24. Trager, B., Chandu, K., Wu, C.W., Stanich, M.: A GPU based implementation of Direct Multi-bit Search (DMS) screen algorithm. In: IS&T/SPIE Electronic Imaging. vol. 8655, pp. 86550Z–1–86550Z–10 (2013)
25. Trager, B., Wu, C.W., Stanich, M., Chandu, K.: GPU-enabled parallel processing for image halftoning applications. In: Proc. of IEEE International Symposium on Circuits and Systems. pp. 1528–1531 (2011)
26. Uichney, R.: The void-and-cluster method for dither array generation. In: IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology. pp. 332–343. International Society for Optics and Photonics (1993)
27. Ulichney, R.: Halftone characterization in the frequency domain. In: Proc. of IS&T's 4th Annual Conference. pp. 464–467 (1994)
28. Wang, S., Cheng, S., Wu, Q.: A parallel decoding algorithm of LDPC codes using CUDA. In: Proceedings of Asilomar Conference on Signals, Systems, and Computers. pp. 171–175 (October 2008)
29. Wei, Z., JaJa, J.: Optimization of linked list prefix computations on multithreaded GPUs using CUDA. In: Proceedings of International Parallel and Distributed Processing Symposium (2010)
30. Zhuge, X., Nakano, K.: Clipping-free halftoning and multitoning using the direct binary search. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E92-A(4), 1192–1201 (2009)