GPU-accelerated Digital Halftoning by the Local Exhaustive Search

Hiroaki Kouge, 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*—The main contribution of this paper is to show a new GPU implementation for the digital halftoning by the local exhaustive search that can generate high quality binary images. We have considered programming issues of the GPU architecture to implement these two methods on the GPU. The experimental result shows that our GPU implementation for the local exhaustive search on NVIDIA GeForce GTX 980 for a 512×512 gray scale image runs in 732 seconds, while the CPU implementation runs in 37,364 seconds. Thus, our GPU implementation attains a speed-up factor of 50.98. Additionally, we also propose a GPU implementation for the digital halftoning by the partial exhaustive search of which the search space of the local exhaustive search is reduced. Similarly, we can accelerate the computation of the partial exhaustive search 30.73 times faster.**

***Keywords*-Image processing; Digital halftoning; GPGPU; Local exhaustive search; Partial exhaustive search**

I. Introduction

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 [1], [2], [3]. 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 [4], dot diffusion [5], ordered dither using the Bayer threshold array [6] and the void-and-cluster threshold array [7], Direct Binary Search (DBS) [8], [9], Local Exhaustive Search (LES) [10], [11], and Partial Exhaustive Search (PES) [12] have been presented.

It is known that, in many cases, the LES [10], [11] generates better quality images. The key idea of the LES 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. The LES performs an exhaustive search for each of the small square subimages in the binary image and replaces the subimages by the best binary pattern. The exhaustive search is repeated until no more improvement is possible. The LES generates a high quality sharp binary image.

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) [13] 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 [14], [15], [16], [17], [18], [19], [20].

The resulting halftoned images generated by the local exhaustive search have high texture quality. However, compared with other well-known halftone methods, such as the Error diffusion, much more computing time is necessary. The main contribution of this paper is to show a new GPU implementation for the local exhaustive search. 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 980 for a 512×512 gray scale image runs in 733 seconds, while the CPU implementation runs in 37,363 seconds. Thus, our GPU implementation attains a speed-up factor of 50.98.

The second contribution of this paper is to show a GPU implementation of the Partial Exhaustive Search (PES) of which the search space of the LES is reduced [12]. In order to reduce the search space, the PES uses an "n choose k" counter ($C(n, k)$ counter for short), which lists all n-bit numbers with $(n - k)$ 0's and k 1's. Using the $C(n, k)$ counter, the quality of the resulting halftoned image is kept and the computing time is reduced [12]. We have also implemented the partial exhaustive search on the GPU. The experimental result shows that our GPU implementation for a 512×512 gray scale image runs in 449 seconds, while the CPU implementation runs in 13,795 seconds. Thus, our GPU implementation attains a speed-up factor of 30.73.

There are FPGA implementations of the LES and the PES to accelerate the computation [10], [12]. Although specific circuits for them are used, our GPU implementations for the LES and the PES can perform the halftoning 1.47 and 1.22 times faster than FPGA implementations, respectively.

This paper is organized as follows. Section II reviews the

digital halftoning by the LES and the PES based on the human visual system. Section III shows how to implement the LES and the PES as a sequential implementation. In Section IV, we show how we have implemented the two methods in the GPU to accelerate the computation. Section V shows the computing time. Section VI offers conclusion.

II. Review of digital halftoning by the Local Exhaustive Search

The main purpose of this section is to review digital halftoning based on the human visual system and the Local Exhaustive Search (LES) [10]. Also, the Partial Exhaustive Search (PES) of which search space of the LES is reduced is reviewed [12].

A. Digital halftoning based on the Human Visual System

Suppose that an original gray-scale image $A = (a_{i,j})$ of size $N \times N$ is given, where $a_{i,j}$ denotes the intensity level at position (i, j) $(1 \leq i, j \leq N)$ taking a real number in the range $[0, 1]$. For simplicity, we assume that images are square. The goal of screening 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). We measure the goodness of the output binary image B using the Gaussian filter that approximates the characteristic of the human visual system. Let $V = (v_{g,h})$ denote a Gaussian filter, i.e. a 2-dimensional symmetric matrix of size $(2w + 1) \times (2w + 1)$, where each non-negative real number $v_{g,h}$ $(-w \leq g, h \leq w)$ is determined by a 2-dimensional Gaussian distribution such that their sum is 1. In other words,

$$v_{g,h} \quad = \quad c \cdot e^{-\frac{g^2+h^2}{2\sigma^2}} \tag{1}$$

where σ is a parameter of the Gaussian distribution and c is a fixed real number to satisfy $\sum_{-w \leq g,h \leq w} v_{g,h} = 1$. 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:

$$r_{i,j} \quad = \quad \sum_{-w \leq g,h \leq w} v_{g,h} b_{i+g,j+h} \quad (1 \leq i, j \leq n) \tag{2}$$

Clearly, from $\sum_{-w \leq g,h \leq w} v_{g,h} = 1$ and $v_{g,h}$ 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 are going to define the error of B as follows. Error $e_{i,j}$ at each pixel location (i, j) is defined by

$$e_{i,j} \quad = \quad a_{i,j} - r_{i,j}, \tag{3}$$

and the total error is defined by

$$\mathit{Error}(A, B) \quad = \quad \sum_{1 \leq i,j \leq n} |e_{i,j}|. \tag{4}$$

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 $\mathit{Error}(A, B)$ is small enough. The best binary image that reproduces A is a binary image B is given by the following formula:

$$B^* \quad = \quad \arg\min_B \mathit{Error}(A, B). \tag{5}$$

If the size of the Gaussian filter is 1×1, then B^* can be obtained by the simple thresholding method. In other words, the binary image $B^* = (b_{i,j})$ such that $b_{i,j} = 1$ if and only if $a_{i,j} \geq \frac{1}{2}$ is an optimal binary image satisfying (5). However, in general, it is very hard to find the optimal binary image B^* for a given gray-scale image A if the Gaussian filter is not small, say, 7×7. Although we do not have the proof, we believe that the problem of finding the optimal binary image B^* is NP-hard. A straightforward method to find the best image is to evaluate $\mathit{Error}(A, B)$ for all possible 2^{N^2} binary images B. Clearly, this takes more than $\Omega(2^{N^2})$ computing time. Since n is usually much larger than 100, this approach is not feasible. Thus, the challenge is to find, in practical computing time, a nearly optimal binary image B, whose total error is close to that of the optimal image B^*.

B. The Local Exhaustive Search

In the following, we review a digital halftoning method using the Local Exhaustive Search (LES) to find a good binary image B whose total error with respect to original gray-scale image A may not be minimum but is small enough. The LES updates a small square region of a temporal binary image by the best binary pattern, in which the total error is the minimum over all possible binary patterns.

Suppose that an original image A and a temporary binary image B are given. Further, let $W(i, j)$ be a window of size $m \times m$ in B whose top-left corner is at position (i, j). Our first idea is to compute the total error for all 2^{m^2} binary patterns in $W(i, j)$ and replace the current binary subimage in the window by the best binary pattern that minimizes the total error. In other words, we find a binary image B' such that

$$\begin{aligned} B' \quad = \quad & \arg\min\{\mathit{Error}(A, B) \mid \\ & B \text{ and } B' \text{ differ only in } W(i, j)\}. \end{aligned} \tag{6}$$

Clearly, $\mathit{Error}(A, B') \leq \mathit{Error}(A, B)$ always holds, and thus, we can say that B' is an improvement over B since it is a better reproduction of original gray-scale image A.

Next, let us see the details on how B' satisfying formula (6) above is computed. Since we use a Gaussian filter of size $(2w + 1) \times (2w + 1)$, the change of the binary pattern affects the errors in a square region of size $(2w + m) \times (2w + m)$, which we call the *affected region*. We refer the reader to Figure 1 for illustrating a window, a Gaussian filter, and the affected region. It should be clear that the best binary pattern can be selected by computing the total

errors of the affected region of size $(2w+m)\times(2w+m)$, because the change of the binary pattern does not affect errors at pixels outside the affected region.

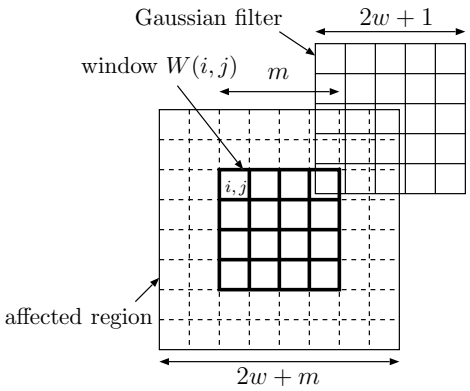

Figure 1. Illustrating a window of size $m \times m$, a Gaussian filter of size $(2w+1)\times(2w+1)$, and the affected region of size $(2w+m)\times(2w+m)$

Let us evaluate the computing time necessary to find the best binary pattern in the window. The error of a fixed pixel in an affected region can be computed in $O(m^2)$ time by evaluating formulas (2) and (3). Hence all the errors in the affected region can be computed in $O(m^2(2w+m)^2)$ time. After that, their sum can be computed in $O((2w+m)^2)$ time. Thus, the total error in the affected region can be computed in $O(m^2(2w+m)^2)$ time. Since we need to check all the possible 2^{m^2} binary patterns, the best binary pattern can be obtained in $O(2^{m^2}m^2(2w+m)^2)$ time. We can improve the computing time by flipping a pixel in the order of the gray code of binary numbers. Recall that the gray code represents a list of all d-bit binary numbers such that any two adjacent numbers differ only one position. Thus, by flipping an appropriate bit using the gray code, we can list all the 2^d binary numbers with d bits. Using the gray code with m^2 bits, we can evaluate the errors for all binary patterns in $O(2^{m^2}w^2)$ time as follows. Starting with the current pixel pattern in the window, we repeat flipping an appropriate pixel according to the gray code. In each flipping operation, we compute the total error in the affected region for the current binary pattern in the window. Since the flipping operation for a single bit affects the error of $(2w+1)\times(2w+1)$ pixels, the total error can be computed in $O(w^2)$ time in an obvious way. Thus, the best binary pattern can be computed in $O(w^2)\times 2^{m^2} = O(2^{m^2}w^2)$ time using the exhaustive search.

We are now in position to show a screening method LES. Let $B_0 = (b^0_{i,j})$ be an appropriate initial binary image. Although we can initialize the binary image B_0 using any screening method, we assume that B_0 is initialized 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, $b^0_{i,j} = 1$ with probability $a_{i,j}$ for every i and j. We repeat sliding a window of size $m \times m$ and improving the binary pattern in the window by replacing the pixel values in it by the best binary pattern. The window sliding can be done in any order. We perform window sliding in the raster scan order as illustrated in Figure 2, to obtain a better quality binary image B_1. The same procedure is repeated, that is, the window sliding operation is applied to B_{t-1} and obtain a better binary image B_t ($t \geq 1$) until B_{t-1} and B_t are identical and no more improvement is possible. When computing B_t for $t \geq 2$, we do not have to perform the exhaustive search for all the windows. If the projected image of the affected region for the current window did not change, then we can omit the exhaustive search. The details of our algorithm Local Exhaustive Search (LES) are spelled out as follows:

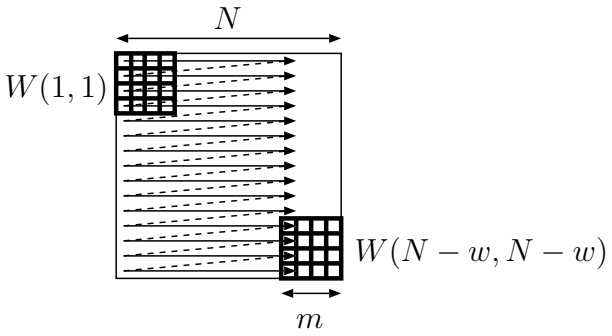

Figure 2. Sliding window in raster scan order

Local Exhaustive Search(A)
Set an appropriate initial binary image in B_0;
$B_1 \leftarrow B_0$;
for $i \leftarrow 1$ to $N-w+1$ do
  for $j \leftarrow 1$ to $N-w+1$ do
    Perform the exhaustive search in $W(i,j)$ for B_1
    and update B_1 by the best binary pattern.
$t \leftarrow 1$;
do {
  $t \leftarrow t+1$;
  $B_t \leftarrow B_{t-1}$;
  for $i \leftarrow 1$ to $N-w+1$ do
    for $j \leftarrow 1$ to $N-w+1$ do
      If the projected image in the affected regions of
      $W(i,j)$ for R_t and R_{t-1} are not identical then
      perform the exhaustive search in $W(i,j)$ for B_t
      and update B_t by the best binary pattern.
  } until (B_t and B_{t-1} are identical)
output (B_t);

C. The Partial Exhaustive Search

In this section, we review a digital halftoning method, named *the Partial Exhaustive Search* (PES), that reduces the computation of the right-hand side of formula (6)

Instead of the exhaustive search that finds an optimal binary pattern for all possible 2^{m^2} binary patterns, the following partial search is performed. We briefly explain the idea of the PES. Let $\boldsymbol{x}$ be a binary pattern in $W(i,j)$, and $B/\boldsymbol{x}$ be the binary image such that $W(i,j)$ of B is replaced by $\boldsymbol{x}$. Let f be a function such that

$$f(\boldsymbol{x}) = Error(A, B/\boldsymbol{x}). \tag{7}$$

In the LES, by computing $f(\boldsymbol{x})$ for all possible 2^{m^2} bit patterns $\boldsymbol{x}$, we can obtain an optimal pattern $\boldsymbol{x}$. Clearly, the total error of $B/\boldsymbol{x}$ is not larger than that of B. Let $C(n,k)$ denote a set of binary numbers that has $(n-k)$ 0's and k 1's. For example, $C(6,3)$ is

$$\begin{aligned} C(6,3) \quad = \quad & \{000111, 001011, 001101, 001110, 010011, \\ & 010101, 010110, 011001, 011010, 011100, \\ & 100011, 100101, 100110, 101001, 101010, \\ & 101100, 110001, 110010, 110100, 111000\}. \end{aligned}$$

Let $\boldsymbol{r}_k$ be the optimal binary pattern of $f(\boldsymbol{x})$ over all binary patterns representing numbers in $C(m^2,k)$, that is,

$$\boldsymbol{r}_k = \arg \min_{\boldsymbol{x} \in C(m^2,k)} f(\boldsymbol{x}).$$

It should be clear that

$$\boldsymbol{r} = \arg \ \min\{f(\boldsymbol{r}_k) | 0 \le k \le m^2\},$$

where $\boldsymbol{r}$ is the optimal solution of $f(\boldsymbol{x})$.

Let β be the total intensity in window $W(i,j)$ of a current binary image B, that is

$$\beta = \sum_{0 \le s,t \le m-1} b_{i+s,j+t}. \tag{8}$$

Since B is an intermediate solution, it is not "bad" binary image. Therefore, the number of 1's in the best binary pattern in $W(i,j)$ must be close to β. Thus, we start the search for $f(\boldsymbol{r}_k)$ with $k = \beta$. By increasing and decreasing q in an obvious way, we can find the bottom of the concave sequence $f(\boldsymbol{r}_0), f(\boldsymbol{r}_1), \ldots, f(\boldsymbol{r}_{m^2})$. If we can start with o $(1 \le o \le m^2 - 1)$ such that $f(\boldsymbol{r}_o)$ is the bottom (i.e. $f(\boldsymbol{r}_{o-1}) > f(\boldsymbol{r}_o) < f(\boldsymbol{r}_{o+1})$), then the linear search just computes $f(\boldsymbol{r}_{o-1})$, $f(\boldsymbol{r}_o)$, and $f(\boldsymbol{r}_{o+1})$. If we start with $o = 0$ then the linear search may just compute $f(\boldsymbol{r}_0)$ and $f(\boldsymbol{r}_1)$, if $f(\boldsymbol{r}_0)$ is the bottom. Using the PES, the searched space can be reduced. Compared with the LES, the searched space by the PES can be reduced to $\frac{1}{3}$ to $\frac{1}{6}$ [12]. These facts allow us to reduce the computing time. Some reader may think the quality of resulting binary images obtained by the PES becomes lower than that by the LES. However, the quality is almost the same and we will show the difference between them in Section V.

III. Implementation for the Local Exhaustive Search and the Partial Exhaustive search

Before explaining our GPU implementation, in this section, we show how the LES and the PES are implemented as a sequential implementation. Since the difference between the LES and the PES is only local search, we will explain the sequential implementation of the LES.

Our implementation consists of the two steps:

- **Step 1: Initialization**
  An initial binary image B is computed from an input gray scale image A by the random dither method.
- **Step 2: The Local search**
  The local/partial exhaustive search is performed for all the window and repeated until no more improvement is possible.

In the following, we will explain the details of each step.

In Step 1, we perform the random dither method to obtain the initial binary image of B. For each pixel, if a randomly generated value is larger than the value of the pixel, the pixel value of the corresponding pixel of B takes value 1. Otherwise, it takes value 0.

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 $E = (e_{i,j})$ by computing formula (3) and the total error from formula (4). In this step, we need to perform the LES that minimizes the total error. It is sufficient to compute the total error of the affected region that includes a window $W(i,j)$ of size $m \times m$ in B as illustrated in Figure 1. The affected region is a region of the image B such that the Gaussian filter for $W(i,j)$ 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

$$\begin{aligned} \mathcal{A}_{i,j} = \{(i',j') | & i - w \le i' \le i + w + m, \\ & j - w \le j' \le j + w + m\}. \end{aligned}$$

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

$$\sum_{(i',j') \in \mathcal{A}_{i,j}} |e_{i',j'}|. \tag{9}$$

We evaluate this formula for all the possible 2^{k^2} binary patterns, and replace pixels with the minimum total error. In other words, we execute the following operation:

$$b_{i,j} = \arg\min \sum_{(i',j') \in \mathcal{A}_{i,j}} |e_{i',j'}|. \tag{10}$$

To perform the LES, we need to compute the convolution in formula (2) for each binary pattern. Since pixel values of B are 0 or 1, $r_{i,j}$ can be computed by adding/subtracting the values of the Gaussian filter $v_{g,h}$ to/from $r_{i,j}$ in $\mathcal{A}_{i,j}$. For

example, when a pixel $b_{i,j}$ is changed from 0 to 1, the values of the Gaussian filter $v_{g,h}$ 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 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 updated until Step 2 terminates. Hence, it makes sense to perform the LES for which pixels might be updated, we use a replacement map $U = (u_{i,j})$ of size $n \times n$. Before a round of the raster scan order search, all values in U is initialized by 0. We set $u_{i,j} = 1$ 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, $u_{i,j} = 1$ if $b_{i,j}$ has been replaced in this period. Further, the affected region in which pixels might be updated in the next round consists of (i, j) such that $u_{i,j}$ or its neighbor takes value 1. Figure 3 illustrates an example of a replacement map and the affected region. In the next round, it is sufficient to perform the local search for the affected region.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0

Figure 3. The replacement map in an affected region

IV. GPU Implementation

The main purpose of this section is to show our GPU implementations of the local and partial exhaustive search.

A. CUDA Architecture

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* [21]. 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, 16-96 Kbytes. Figure 4 illustrates the CUDA hardware architecture.

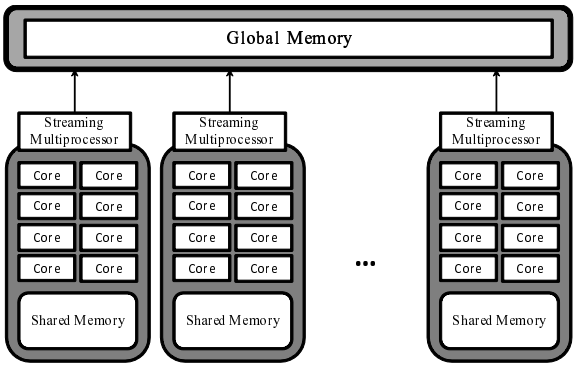

Figure 4. CUDA hardware architecture

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 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.

B. GPU implementation for the local exhaustive search

We are now in a position to explain how we implement the LES. 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 random values and d-bit gray code numbers are also stored in the global memory to perform the random dither method and the LES, respectively. In the following, to perform the computation in parallel, basically we divide an input image into subimages whose size is $q \times q$ and perform halftoning for each subimage in parallel. In our implementation, the size of the subimage is $\frac{N}{q} \times \frac{N}{q}$.

To implement Step 1, $\frac{N^2}{q^2}$ CUDA blocks are invoked one for each subimage of size $\frac{N}{q} \times \frac{N}{q}$. 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. After that, threads generate an initial binary image by the random dither method. In the GPU implementation, the computing cost of generating random numbers is very high, that is, the generating time by the GPU is much longer than that by the CPU. Therefore, we use random values generated by the CPU and store them into the global memory beforehand. Threads read the random value from the global memory and perform the random dither method. The result of B as the initial binary image is stored

into the global memory. 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 LES In each kernel, the LES to evaluate formula (5) is performed in parallel using multiple CUDA blocks. Each CUDA block is responsible for executing the LES of the corresponding subimage.

However, the LES 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 Figure 5 illustrating the groups. We use $\frac{4N^2}{q^2}$ CUDA blocks, and perform the LES in all blocks of each group. Note that, if $q \geq 2w$ then the Gaussian filter of two blocks in a group never affect each other, where the subimage is $q \times q$ 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. 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.

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

Figure 5. Groups of blocks

After that, each CUDA block performs the LES 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. As shown in Section III, to compute the convolution in formula (2), it can be computed by adding/subtracting the value of the Gaussian filter. Also, according to the gray code, we repeat flipping an appropriate pixel. In the implementation, threads use d-bit gray code numbers stored in the global memory beforehand. In each flipping operation, we compute the total error in the affected region for the current binary pattern in the window. In our implementation, we utilize a summing technique by binary reduction proposed in [22] 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 LES is concurrently performed using the partition shown in Figure 5, 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.

C. GPU implementation for the partial exhaustive search

In our implementation of the PES, basically the difference from the above implementation of the LES is the local search in Step 2 that finds the optimal binary pattern. Therefore, we will show how the pixels are updated in window $W(i,j)$ of a current binary image B. First, the total intensity β is obtained in the window by counting the number of white pixels. Then, we start the search for $f(\boldsymbol{r}_k)$ with $k = \beta$ and find the bottom $f(\boldsymbol{r}_o)$ such that $f(\boldsymbol{r}_{o-1}) > f(\boldsymbol{r}_o) < f(\boldsymbol{r}_{o+1})$ as shown in Section II-C. To perform the local search, it is necessary to list $C(m^2, k)$ numbers. Also, we use the same idea that gray code is used in the LES. We list $C(m^2, k)$ numbers for each k ($0 \leq k \leq m^2$) and each list is resorted like gray code. We omit the detailed explanation, but any two adjacent numbers in each resorted list differ only at most two positions. Therefore, for each list, we compute the difference positions from the next numbers one by one. We store the first number and the positions in the global memory in advance. Thus, in the PES, we also compute the total error by adding/subtracting the values of Gaussian filter and utilize a summing technique by binary reduction like the implementation of the LES.

V. Experimental results

The main purpose of this section is to show the experimental results. We will show the resulting images and the computing time. We have used three gray scale images, Lena [23] of size 512×512.

In order to evaluate the computing time for generating the halftoned images, we have used NVIDIA GeForce GTX 980, which has 2048 processing cores in 16 SM units [24]. We have also used Intel PC using Xeon X7460 running in 2.66GHz to evaluate the implementation by sequential algorithms. We use the Gaussian filter with parameter $\sigma = 1.0$ and $w = 3$. In the local search, the window size of $W(i,j)$ is 4×4, i.e., $m = 4$. Also, the size of subimage used in the GPU implementation is 9×9, i.e., $q = 9$. Figure 6 shows the resulting binary image using the LES. We can find that the resulting image is high texture quality. Since the resulting image using the PES is almost the same quality

and we cannot distinguish them, we omit the showing the halftoned image by the PES. Table I 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 of the LES can be reduced by a factor of 50.98.

Table I
COMPUTING TIME (IN SECONDS) OF THE LOCAL EXHAUSTIVE SEARCH AND THE PARTIAL EXHAUSTIVE SEARCH

	CPU	FPGA [10], [12]	GPU
Local exhaustive search	37,364	1,186	733
Partial exhaustive search	13,794	546	449

Table II shows average errors when "Lena" is halftoned by Error Diffusion, DBS, LES, and PES. The average error is an error per pixel that is computed from the total error in formula (3). According to the table, the average error of the LES and the PES is smaller than that of Error diffusion and DBS and the difference between the LES and the PES is much small. Also, the error of the parallel execution of the LES and the PES is a little larger than that of sequential ones. However, the difference of the error is small and the appearance is almost the same.

Table II
AVERAGE ERROR WHEN "LENA" IS HALFTONED BY ERROR DIFFUSION, DBS, LES, AND PES.

	CPU	GPU
Error Diffusion	7.06	—
DBS	5.86	—
Local exhaustive search	4.70	4.75
Partial exhaustive search	4.71	4.75

VI. CONCLUSIONS

In this paper, we have proposed an implementation of the digital halftone method by the local exhaustive search on the GPU. In our implementation, we have considered programming issues of the GPU architecture. We have implemented it on NVIDIA GeForce GTX 980. The experimental result shows that our GPU implementation for the local exhaustive search on NVIDIA GeForce GTX 980 for a 512×512 gray scale image runs in 733 seconds, while the CPU implementation runs in 37,364 seconds. Thus, our GPU implementation attains a speed-up factor of 50.98. Compared to the existing FPGA implementations using specific circuits, the computing time of our GPU implementation is shorter. We have also proposed a GPU implementation of the digital halftone method by the partial exhaustive search that reduces search space. Similarly, we can accelerate the computation of the partial exhaustive search 30.72 times faster.

REFERENCES

[1] T. Asano and K. Nakano, "Halftoning through optimization of restored images – a new approach with hardware acceleration," The Institute of Electronics Information and Communication Engineers, COMP2002-75, Tech. Rep., March 2003.

[2] D. L. Lau and G. R. Arce, *Modern Digital Halftoning*. Marcel Dekker, 2001.

[3] K. Nakano, "Various screening methods," *Convertech*, vol. 36, no. 1, pp. 72–77, 2008.

[4] R. Floyd and L. Steinberg, "An adaptive algorithm for spatial gray scale," in *Proc. of International Symposium Digest of Technical Papers, Society for Information Displays*, 1975, pp. 36–37.

[5] D. Knuth, "Digital halftones by dot diffusion," *ACM Transactions on Graphics*, vol. 6, no. 4, pp. 245–273, 1987.

[6] B. Bayer, "An optimum method for two-level rendition of continuous-tone pictures," in *IEEE International Conference on Communications*, 1973, pp. 11–15.

[7] R. Uichney, "The void-and-cluster method for dither array generation," in *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 1993, pp. 332–343.

[8] M. Analoui and J. Allebach, "Model-based halftoning by direct binary search," in *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, vol. 1666, 1992, pp. 96–108.

[9] D. J. Lieberman and J. P. Allebach, "Efficient model based halftoning using direct binary search," in *Proc. of International Conference on Image Processing*, vol. 1, 1997, pp. 775–778.

[10] Y. Ito and K. Nakano, "FM screening by the local exhaustive search with hardware acceleration," *International Journal on Foundations of Computer Science*, vol. 16, no. 1, pp. 89–104, 2005.

[11] ——, "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*, vol. 19, no. 6, pp. 1373–1386, 2008.

[12] K. Nakano and Y. Yamagishi, "Hardware n choose k counters with applications to the partial exhaustive search," *IEICE Trans. on Information & Systems*, pp. 1350–1359, July 2005.

[13] NVIDIA Corporation, "CUDA ZONE," http://www.nvidia.com/page/home.html.

[14] J. Diaz, C. Muñoz-Caro, and A. Niño, "A survey of parallel programming models and tools in the multi and many-core era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1369–1386, August 2012.

Figure 6. The resulting binary image for "Lena" using the LES

[15] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on GPUs," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, July 2008, pp. 340–345.

[16] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," in *Proceedings of the 14th International Conference on High Performance Computing*, 2007, pp. 197–208.

[17] D. Man, K. Uda, H. Ueyama, Y. Ito, and K. Nakano, "Implementations of parallel computation of Euclidean distance map in multicore processors and GPUs," in *Proceedings of International Conference on Networking and Computing*, 2010, pp. 120–127.

[18] K. Ogawa, Y. Ito, and K. Nakano, "Efficient Canny edge detection using a GPU," in *International Workshop on Advances in Networking and Computing*, Nov. 2010, pp. 279–280.

[19] S. Wang, S. Cheng, and Q. Wu, "A parallel decoding algorithm of LDPC codes using CUDA," in *Proceedings of Asilomar Conference on Signals, Systems, and Computers*, October 2008, pp. 171–175.

[20] Z. Wei and J. JaJa, "Optimization of linked list prefix computations on multithreaded GPUs using CUDA," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2010.

[21] *CUDA C Programming Guide Version 6.5*, NVIDIA Corporation, 2014.

[22] K. Nakano, "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*, vol. E96-D, no. 12, pp. 2626–2634, December 2013.

[23] L.-M. Po, "Lenna 97: A complete story of Lenna," http://www.ee.cityu.edu.hk/~lmpo/lenna/Lenna97.html, 2001.

[24] NVIDIA Corporation, "Whitepaper NVIDIA GeForce GTX 980 v1.1," 2014.