International Journal of Foundations of Computer Science

FM Screening by the Local Exhaustive Search, with Hardware Acceleration

YASUAKI ITO
Department of Artificial Complex Systems Engineering
Hiroshima University
Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN

and

KOJI NAKANO
Department of Artificial Complex Systems Engineering
Hiroshima University
Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

The main contribution of this paper is to show a new approach for FM screening which we call Local Exhaustive Search (LES) method, and to present ways to accelerate the computation using an FPGA. FM screening, as opposed to conventional AM screening, keeps unit dot size when converting an original gray-scale image into the binary image for printing. FM screening pays great attention to generate moiré-free binary images reproducing continuous-tone and fine details of original photographic images. Our basic approach for FM screening is to generate 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 to the binary image. LES performs an exhaustive search for each of the small square subimages in the binary image and replaces the subimage by the best binary pattern. The exhaustive search is repeated until no more improvement is possible. The experimental results show that LES produces high quality sharp binary images. We also implemented LES on an FPGA to accelerate the computation and achieved a speedup factor of up to 51 over the software implementations.

Keywords: Image Processing, Screening for Printing, Local Search, FPGA-based computing

1. Introduction

Screening is an important task to convert a continuous-tone image into a binary image with pure black and white pixels [2, 8]. This task is necessary when printing a monochrome or color image by a printer with limited number of ink colors. AM (Amplitude Modulated) screening, a commonly used screening method, arranges

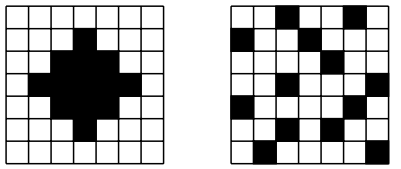

Figure 1: AM screening and FM screening

black dots in a regular grid and reproduces the intensity of an original continuous-tone image by the number of black pixels in a dot. A black dot involves fewer black pixels to reproduce highlight color, and has more black pixels to create a shadow image. FM (Frequency Modulated) screening, on the other hand, keeps dots of a unit size when converting an original continuous-tone image into the binary image for printing. The intensity level of an original continuous-tone image is reproduced by the density of black unit dots (or pixels). FM screening pays great attention to generate moiré-free binary images reproducing continuous-tone and fine details of original photographic images. We refer the reader to Figure 1 for illustrations of a dot of AM screening and dots of FM screening. The most well-known FM screening algorithm is Error Diffusion [5] that propagates rounding errors to unprocessed neighboring pixels according to some fixed ratios. Error Diffusion preserves the average intensity level between the original input image and the binary output image. It is also quite fast and often produces good results. However, Error Diffusion may generate worm artifacts, which is a sequence of dots like a worm, especially in the areas of uniform intensity. Several techniques have been developed to prevent artifacts in output binary images [14]. Further, Error Diffusion based techniques is that the pixel values are propagated to neighbors and the resulting images are defocused.

In applications requiring high fidelity of the printed material (such as printing fine art books, pictorial books, and replicas of paintings), an FM screening method that produces artifact-free higher quality binary images reproducing original work is expected even if the computation takes a lot of time. The first contribution of this paper is to present a new approach for FM screening that we call Local Exhaustive Search (LES). Our idea for FM screening, LES, is to use the local search technique investigated in the area of combinatorial optimization, which usually takes a lot of computing time. More specifically, our new FM screening method LES produces 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. We define the total error of the binary image to be the sum of the difference of the intensity levels over all pixels between the original image and the projected image. LES performs the local exhaustive search for a small square window of size, say, 2×2, 3×3, and 4×4,

in the binary image, and finds the best binary image pattern in the window, whose total error is the minimum over all possible binary patterns. After that, a binary subimage in the window is replaced by the best binary image pattern obtained. The local exhaustive search is repeated until no more improvement on the total error is possible.

A similar idea has been presented under the name of Direct Binary Search (DBS) [1]. In DBS, a pixel value is flipped if the resulting image has smaller total error. Also, neighboring pixel values are swapped if the total error of the resulting image decreases. Hence, LES for a window of size 2×2 or larger finds the best image for more candidates than DBS, and thus, the total error of the resulting binary image is smaller. In this paper, we also show the experimental results for DBS that performs swap operations for four immediate neighbors, above, below, right, and left (DBS4) and for eight immediate neighbors (DBS8) including diagonally adjacent pixels, as counterparts of LES.

The second contribution of this paper is to implement LES in an FPGA to accelerate the computation. An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware design can be embedded quickly. We have used Nallatech Xtreme DSP kit [13], which is a PCI board with Xilinx VirtexII family FPGA XC2V3000-4 [7], and embedded a circuit to perform the local exhaustive search for a window of size 3×3 and 4×4. To reduce the amount of used FPGA resource and the delay, we use the *instance-specific* approach [3, 4, 11], which embed a hardware depending on a part of the input instance. The instance-specific approach is applied as follows. One can think that the inputs of LES are an original image and a Gaussian filter. Since a Gaussian filter can be fixed during the computation by LES, we can embed a circuit to perform the local exhaustive search for a specific Gaussian filter. Further, we have developed an improved FPGA-implementation which minimizes the overhead caused by the high latency of the PCI bus. Consequently, we have succeeded in accelerating LES by a speedup factor of up to 51 over the software implementations.

This paper is organized as follows. Section 2 formalizes the problem of finding the best binary image of an original gray-scale image as a combinatorial optimization problem. In Section 3, we present the local exhaustive search (LES) method, which is an approximation algorithm for solving the combinatorial optimization problem. Section 4 shows an implementation of LES on an FPGA. In Section 5, we present the experimental results for screening images by known methods and LES. Section 6 offers concluding remarks.

2. FM Screening based on the Human Visual System

This section defines the problem of finding the best binary image of an original gray-scale image as a combinatorial optimization problem.

Suppose that an original gray-scale image $A = (a_{i,j})$ of size $n \times n$ is given[a] , 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 $[0, 1]$. The goal of screening is to find a binary image

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

$B = (b_{i,j})$ of the same size that reproduces original image A, where each $b_{i,j}$ is either 0(black) or 1(white). We measure the goodness of 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 $v_{k,l}$ $(-w \leq k, l \leq w)$ is determined by a 2-dimensional Gaussian distribution such that their sum is 1. In other words,

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

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

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

Clearly, from $\sum_{-w \leq k,l \leq w} v_{k,l} = 1$ 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 an 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 with the minimum total error 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^*.

3. The Local Exhaustive Search for FM Screening

The main purpose of this section is to present our ideas 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. Our approach, named *Local Exhaustive Search* (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 $k \times k$ in B whose top-left corner is at position (i,j). Our first idea is to compute the total error for all 2^{k^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

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

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 $(2w+k) \times (2w+k)$, which we call the *affected region*. We refer the reader to Figure 2 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 $(2w+k) \times (2w+k)$, because the change of the binary pattern does not affect errors at pixels outside the affected region.

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 $O(k^2)$ time by evaluating formulas (2) and (3). Hence all the errors in the affected region can be computed in $O(k^2(2w+k)^2)$ time. After that, their sum can be computed in $O((2w+k)^2)$ time. Thus, the total error in the affected region can be computed in $O(k^2(2w+k)^2)$ time. Since we need check all the possible 2^{k^2} binary patterns, the best binary pattern can be obtained in $O(2^{k^2}k^2(2w+k)^2)$ 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 m-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^m binary numbers with m bits. Using the gray code with k^2 bits, we can evaluate the errors for all binary patterns in $O(2^{k^2}w^2)$ 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

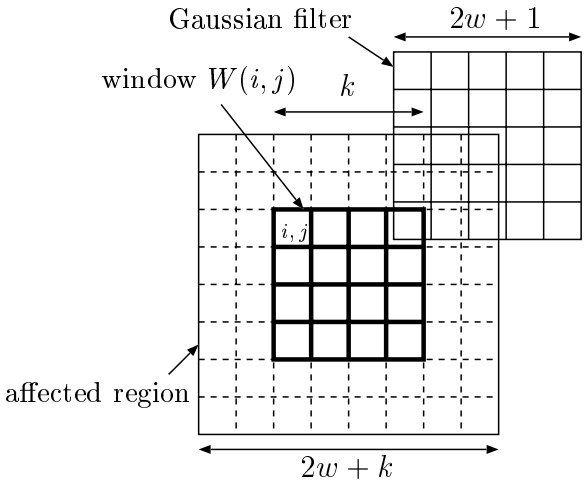

Figure 2: Illustrating a window of size $k \times k$, a Gaussian filter of size $(2w+1) \times (2w+1)$, and the affected region of size $(2w+k) \times (2w+k)$

computed in $O(w^2)$ time in an obvious way. Thus, the best binary pattern can be computed in $O(w^2) \times 2^{k^2} = O(2^{k^2} w^2)$ time using the exhaustive search.

We are now in position to show our new 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 $k \times k$ 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 3, 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:

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.

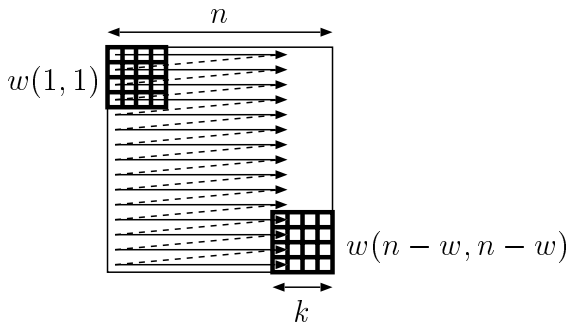

Figure 3: Sliding window in raster scan order

$t \leftarrow 1$;
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);

4. Hardware Acceleration for the Local Exhaustive Search using an FPGA

We have developed the hardware accelerator using the PCI-connected FPGA that performs the exhaustive search in order to find the best binary pattern in a window. This section is devoted to show the architecture of our hardware FPGA-based accelerator.

Before showing the architecture, we first show how our hardware accelerator is used by the host PC. Recall that, as shown in Figure 2, $W(i,j)$ contains pixels at position $(i+i', j+j')$ for $0 \leq i', j' \leq k-1$ and the affected region involves pixels $(i+i', j+j')$ for $-w \leq i', j' \leq w+k-1$. To perform the local exhaustive search for a window $W(i,j)$, the host PC initializes all binary pixels in window $W(i,j)$ to 0, i.e. sets $b_{i+i',j+j'} \leftarrow 0$ for all $0 \leq i', j' \leq k-1$, and computes pixel values $r_{i+i',j+j'}$ $(-w \leq i', j' \leq w+k-1)$ in the affected region of the projected image. After that, it computes all the errors $e_{i+i',j+j'}$ $(-w \leq i', j' \leq w+k-1)$ in the affected region by formula (3). The host PC sends these errors to the PCI-connected FPGA. The FPGA computes the best binary pattern for $W(i,j)$, and returns it to the host PC. It should be clear that, the best binary pattern can be computed using $e_{i+i',j+j'}$

$(-w \leq i', j' \leq w+k-1)$. We call this operation for a fixed window $W(i,j)$ *a round*. We can summarize the exhaustive search for a window $W(i,j)$ as follows:

Exhaustive Search$(W(i,j))$
for all $0 \leq i', j' \leq k-1$ do
  $b_{i+i',j+j'} \leftarrow 0$;
for all $-w \leq i', j' \leq w+k-1$ do
  $r_{i+i',j+j'} \leftarrow \sum_{-w \leq k,l \leq w} v_{k,l} b_{i+i'+k,j+j'+l}$;
for all $-w \leq i', j' \leq w+k-1$ do
  $e_{i+i',j+j'} \leftarrow a_{i+i',j+j'} - r_{i+i',j+j'}$;
host PC sends $e_{i+i',j+j'}$ $(-w \leq i', j' \leq w+k-1)$
  to the FPGA;
FPGA computes the best binary pattern in $W(i,j)$;
  and sends it to host PC;
host PC stores the best binary pattern in $b_{i+i',j+j'}$
  $(0 \leq i', j' \leq k-1)$;

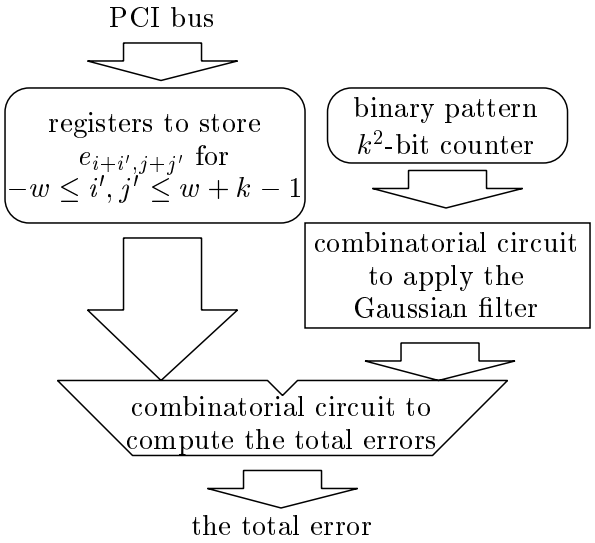

Figure 4: Illustrating a part of the hardware accelerator.

Next, we are going to show the architecture of our hardware accelerator. Figure 4 illustrates a part of the FPGA-based hardware accelerator, which outputs the total error for every binary pattern. Errors $e_{i+i',j+j'}$ sent from the host PC are stored in the registers. A k^2-bit counter is used to list all the binary patterns. Let $b'_{i+i',j+j'}$ $(0 \leq i', j' \leq k-1)$ denote the current pixel value of the binary pattern stored in the k^2-bit counter. A combinatorial circuit is used to apply the Gaussian filter to the current binary pattern stored in the counter. In other words, it computes, for

$0 \leq i', j' \leq k-1$,

$$e'_{i+i',j+j'} = e_{i+i',j+j'} - \sum_{-w \leq k,l \leq w} v_{k,l} b'_{i+i'+k,j+j'+l},$$

where $b'_{i+i'+k,j+j'+l} = 0$ unless $0 \leq i'+k, j+l' \leq k-1$. Recall that we can fix a Gaussian filter $V = (v_{k,l})$. Thus, the value of each $v_{k,l}$ is a constant value. After computing $e'_{i+i',j+j'}$, the total error in the affected region

$$\sum_{-w \leq i',j' \leq w+k-1} |e'_{i+i',j+j'}|,$$

is computed. This value can be computed by summing the integers, which can be done efficiently on the FPGA [9, 10, 12]. The architecture illustrated in Figure 4 outputs the errors of all binary patterns in 2^{k^2} clock cycles. Using all the total errors, the best binary pattern can be stored into a register using a comparator in an obvious way. After that, the best binary pattern is sent to the host PC, which stores it in the window of the binary image.

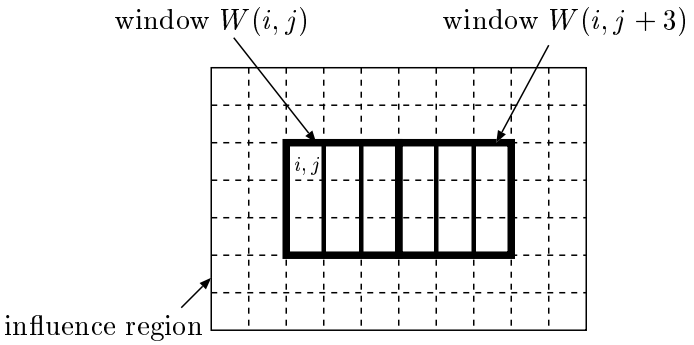

Figure 5: Illustrating a exhaustive search for four windows.

In principle, the hardware on the FPGA computes the best binary pattern in 2^{k^2} clock cycles. Since the circuit is quite simple (it mainly applies the Gaussian filter and computes the total sum), we can improve the clock frequency using the pipelining technique [12]. Actually, we have partitioned the combinatorial circuits into seven stages. The pipelined architecture works properly in frequency of 80MHz on Nallatech Xtreme DSP kit [13] with Xilinx VirtexII family FPGA XC2V3000-4 [7]. We have used Xilinx ISE Logic Design Tools [6] for logic synthesis.

For $k = 4$, the hardware accelerator works $2^{4^2} = 65536$ clock cycles to find the best binary pattern. In this case, the overhead caused by the communication through the PCI bus is negligible. However, if $k = 3$, then it only runs in $2^{3^2} = 512$ clock cycles for the exhaustive search. Hence, the time necessary to transfer the current errors and the resulting binary image though the PCI bus will be larger than that for the exhaustive search due to the high latency of the PCI bus.

Further, to reduce the communication through the PCI bus for $k = 3$, we have developed an improved FPGA implementation, that performs the exhaustive search for the four windows $W(i, j)$, $W(i, j+1)$, $W(i, j+2)$, and $W(i, j+3)$ in a round. More specifically, the host PC sends the current errors in the affected region illustrated in Figure 5. The FPGA performs the exhaustive search for four windows $W(i, j)$, $W(i, j+1)$, $W(i, j+2)$, and $W(i, j+3)$ in $4 \cdot 2^{3^2} = 2048$ clock cycles. After that, the best binary image of four windows, that is, a binary pattern of 3×6 is transferred to the host PC. Roughly speaking, the number of rounds decreases by a quarter, because each round performs exhaustive searches for four windows.

Images		RD	ED	LES1	DBS4
"squares" 64×256	Average Error	23.5	7.06	8.48	5.93
	Time (Software)	0.717ms	0.748ms	99.0ms	0.359s
	Time (FPGA)	-	-	-	-
	Speed up	-	-	-	-
"Lena" 512×512	Average Error	27.7	6.56	8.65	5.98
	Time (Software)	12.4ms	16.2ms	1.80s	6.96s
	Time (FPGA)	-	-	-	-
	Speed up	-	-	-	-

Images		DBS8	LES4	LES9	LES4X9	LES16
"squares" 64×256	Average Error	5.86	5.44	4.90	4.90	4.42
	Time (Software)	0.636s	1.05s	29.0s	29.0s	3360s
	Time (FPGA)	-	-	3.25s	1.54s	68.3s
	Speed up	-	-	8.92	18.8	49.2
"Lena" 512×512	Average Error	5.93	5.52	4.99	4.99	4.62
	Time (Software)	11.8s	16.8s	488s	488s	60500s
	Time (FPGA)	-	-	50.3s	23.5s	1186s
	Speed up	-	-	9.7	20.3	51.0

Table 1: The experimental results for "squares" and "Lena" using the Gaussian filter of size 7×7 with parameter $\sigma = 1.0$.

5. Experimental Results

This section presents experimental results. We have developed a software that performs LES for windows of sizes 1×1, 2×2, 3×3, and 4×4, which we call LES1, LES4, LES9, and LES16, respectively.

Table 1 show the experimental results for two 8-bit gray-scale images "squares" and "Lena". We have used a Pentium4-based PC (Xeon 2.8GHz) with Linux operating system (Kernel 2.4). The source program is compiled by gcc 3.2.2 with -O2 and -march=pentium4 options. We have used a library call rand() to generate random bits. Since the images are 8-bit gray-scale, the intensity of a pixel takes one of the rational numbers $0/255, 1/255, 2/255, \ldots, 255/255$. Gray-scale image "squares" of size 64×256 has four squares of size 64×64, which have uniform intensity levels, $239/255$, $223/255$, $191/255$, and $128/255$. Note that $15/16 \approx 239/255$, $7/8 \approx 223/255$, $3/4 \approx 191/255$, and $1/2 \approx 128/255$. Hence, the binary images for

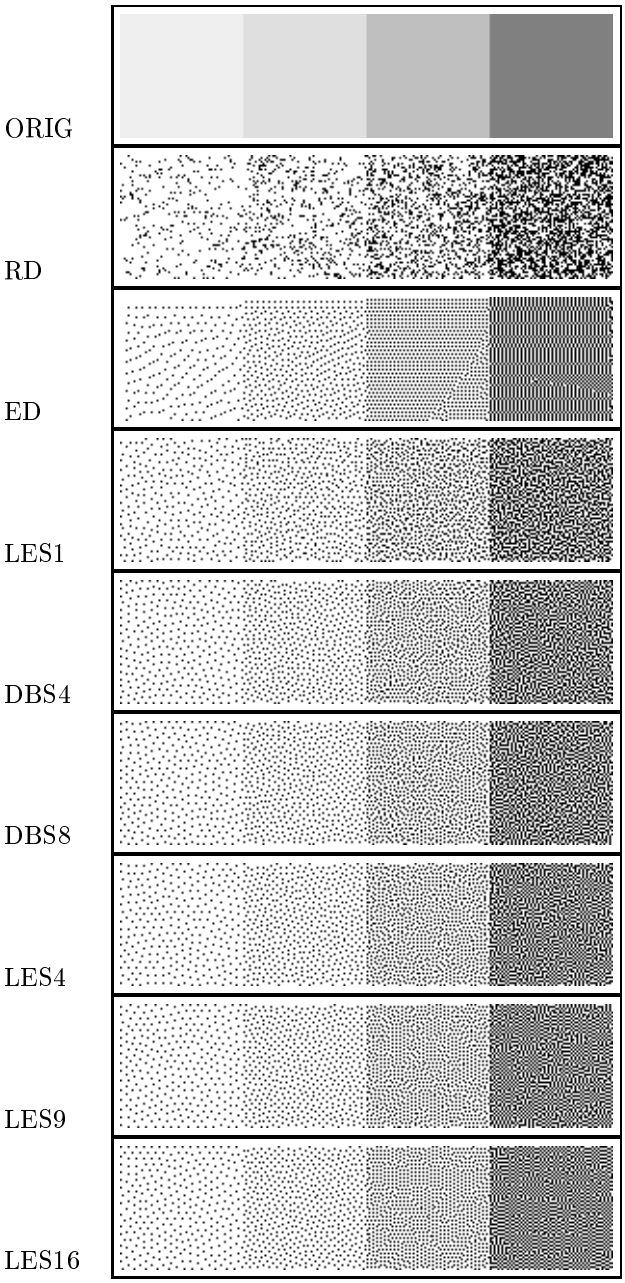

Figure 6: The original and the resulting binary images for "squares".

the four squares are expected to have one black pixel in every 16, 8, 4, and 2 pixels, respectively. Gray-scale image "Lena" is a well-known standard image of size 512×512. We have performed the experiment for these two images using the Random Dither(RD), the Floyd and Steinberg's Error Diffusion(ED), the Direct Binary Search for four neighbors (DBS4) and for eight neighbors (DBS8), LES for windows of size 1×1 (LES1), 2×2 (LES4), 3×3 (LES9), and 4×4 (LES16). It is quite interesting that the average error by ED is better than that by LES1, although its strategy is not based on the local search. We can list the six local search based methods in order of the quality of the binary image in terms of the average error as follows:

$$\text{LES16}, \text{LES9}, \text{LES4}, \text{DBS8}, \text{DBS4}, \text{LES1}.$$

Since LES1 just repeats the toggle operation, it cannot be better than DBS4 and DBS8. It should be noted that LES4 finds a better binary image than DBS4 and DBS8. The reason is as follows. Suppose that DBS8 is performed for a fixed pixel (i, j). The local exhaustive search for a window of size 2×2 whose top-left corner is (i, j) involves the swap operations between (i, j)-$(i-1, j)$, (i, j)-$(i, j-1)$, and (i, j)-$(i-1, j-1)$. Thus, the search space of DBS8 is completely included by the total search space for LES4 whose top-left corners are $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$, (i, j). It follows that LES4 can find better binary images than DBS4 and DBS8.

Table 1 also shows the computing time by the software and by an FPGA. LES9 performs the local exhaustive search for a single window of size 3×3 in a round, and LES4X9 performs it four windows in a round as illustrated in Figure 5. Thus, LES4X9 is approximately twice faster than LES9 due to the smaller communication overhead through the PCI-bus. The FPGA implementation for LES9 achieved a speedup factor of 18.8 and 20.3 for "squares" and "Lena", respectively, while that for LES16 is approximately 50 times faster than the traditional software solution. It follows that LES4X9 still have a large overhead for communication through the PCI-bus.

Figure 6 shows the original image (ORIG) of "squares", and the resulting images. The resulting image obtained by RD is awkward. ED generates an binary image that has worm artifacts, especially, in the first and the second squares. In the third and the fourth squares, it has regular patterns generating gaps which is not in the original image. For example, the fourth square has regular patterns separated horizontally, and the separated lines can be seen as gaps. Although the average error of ED is smaller than LES1, the image by produced the LES1 seems better. Among the images encountered by the local search based screening methods, the image by LES16 has the best quality, because the distribution of the black pixels is most uniform. For example, in the third square, it is expected that no two black pixels are adjacent. The third square of the image obtained by LES16 has fewest adjacent black pixels. The images generated by DBS4 and DBS8 has more adjacent black pixels which makes the images non-uniform and dusty, although the original image has uniform intensity.

Figures 7, 8 and 9 shows the resulting image for "Lena". Although screening is performed for image "Lena" of size 512×512, LES16 creates a smoother image,

especially in face and shoulder of the woman, than DBS8.

Figure 7: The resulting images for "Lena" using Error Diffusion

6. Concluding Remarks

We have presented the Local Exhaustive Search (LES) method for finding a high quality binary image that reproduce the original gray-scale image. Since this screening process requires huge amount of computation, it would be impractical if we would implement it naively. Although the processing time is still much larger than that of the currently used screening algorithms such as Error Diffusion and also much larger hardware resources are required, our algorithm would be useful in applications requiring high fidelity binary images.

Figure 8: The resulting images for "Lena" using DBS8

References

1. M. Analoui and J.P. Allebach. Model-based halftoning by direct binary search. In *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, volume 1666, pages 96–108, 1992.
2. T. Asano. Digital halftoning: Algorithm engineering challenges. *IEICE Transactions on Information and Systems*, February 2003.
3. J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, May 2003.
4. J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the cky parsing for large contex-free grammars. *to appear in International Journal on Foundations of Computer Science*, 2004.
5. R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. *SID*

Figure 9: The resulting images for "Lena" using LES16

75 Digest,Society for Information Display, pages 36–37, 1975.

6. Xilinx Inc. *ISE Logic Design Tools*, 2002.
7. Xilinx Inc. *Virtex-II Platform FPGAs: Complete Data Sheet*, 2003.
8. D. L. Lau and G. R. Arce. *Modern Digital Halftoning*. Marcel Dekker, 2001.
9. R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):838–850, 8 2000.
10. D. E. Muller and F. P. Preparata. Bounds to complexityies of network for sorting and for switching. *J. ACM*, 22:195–201, 1975.
11. K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.
12. K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes.

Theoretical Computer Science, pages 57–77, January 1995.

13. Nallatech. *Xtreme DSP Development Kit User Guide*, 2002.
14. V. Ostromoukhov. A simple and efficient error-diffusion algorithm. In *Proc.of the 28th SIGGRAPH*, pages 567 – 572, 2001.