An Image Retrieval System Using FPGAs *

Koji Nakano

School of Information Science
JAIST
Tatsunokuchi, Ishikawa 923-1292 Japan
Tel: +81-761-51-1220
Fax: +81-761-51-1149
e-mail: knakano@jaist.ac.jp

Etsuko Takamichi

School of Information Science
JAIST
Tatsunokuchi, Ishikawa 923-1292 Japan
Tel: +81-761-51-1699(ext.1367)
Fax: +81-761-51-1149
e-mail e-takami@jaist.ac.jp

Abstract— The main contribution of this paper is to present an image retrieval system using FPGAs. Given a template image T and a database of a number of images $I_1, I_2, \ldots$, our system lists all images that contain a subimage similar to T. More specifically, a hardware generator in our system creates the Verilog HDL source of a hardware that determines whether I_i has a similar subimage to T for any image I_i and a particular template T. The created Verilog HDL source is embed in an FPGA using the design tool provided by the FPGA vendor. Since the hardware embedded in the FPGA is designed for a particular template T, it is an instance-specific hardware that allows us to achieve extreme acceleration. We evaluate the performance of our image matching hardware using a PCI-connected Xilinx FPGA and a timing analyzer. Since the generated hardware attains up to 3000 speed-up factor over the software solution, our approach is promising.

I. Introduction

Suppose that an image database $\mathcal{I}$ containing a number of gray-scale images $\{I_1, I_2, \ldots\}$ and a template image T are given. We assume that T is small, say, 32×32 while each I_i is large, say, 1024×1024 or larger. We are interested in the task of listing all images in $\mathcal{I}$ that contains a similar subimage to T. This task has many applications in the areas such as object recognition, vehicle tracking, finding a particular pattern in VLSI masks, among others [1]. The main contribution of this paper is to present an FPGA-based instance-specific hardware solution for this task. More precisely, let $D(T, I_i)$ denote a function that returns a value indicating the difference between T and I_i such that the value of $D(T, I_i)$ is small if I_i has similar subimage to T. Our idea is to embedded a hardware that computes $D_T(I_i)$ $(= D(T, I_i))$ in a PCI-connected FPGA. We have developed a system illustrated in Figure 1 that computes $D_T(I_1), D_T(I_2), \ldots$ using the FPGA. Given a template image T, our hardware generator automatically creates a Verilog HDL source program which is designed for computing $D_T(I)$. The source program is complied using a design tool provided by an FPGA vendor. The created hardware is embedded in the PCI-connected FPGA. The host PC sends images $I_1, I_2, \ldots$ stored in an image database $\mathcal{I}$ to the PCI-connected FPGA. The FPGA computes the values $D_T(I_i)$ in turn, and returns each of them to the host PC. The host PC lists the images whose $D_T(I_i)$ is no larger than the threshold value. Although the time necessary to compile the Verilog HDL source and embedded into the FPGA is very long, say, several hours, the total computing time can be decreased if database $\mathcal{I}$ has a large number of images.

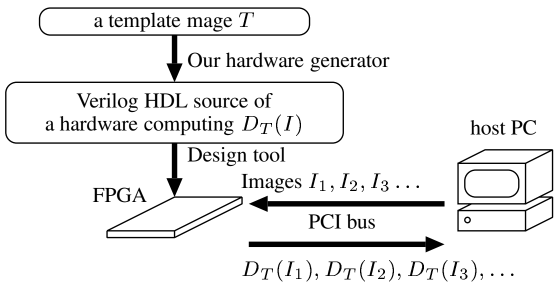

Fig. 1. Our FPGA-based image retrieval system

Let T be a template image with $m \times m$ pixels and I be an image with $n \times n$ pixels. We assume that T has e effective pixels ($e \leq m^2$) that are taken into account for image matching. As we are going to show later, the evaluation of $D(T, I)$ for L-level gray-scale images takes $O(n^2 e)$ time by a software solution. Our instance-specific FPGA solution runs in $\frac{n^2}{m}$ local clock cycles using $O(me)$ gates, $O(m^2)$ flip-flops, and an $(m-1)n$-bit block RAM for binary images, and in $\frac{n^2 \log L}{m}$ local clock cycles using $O(me)$ gates, $O(m^2 \log L)$ flip-flops, and $\log L$ block RAMs with mn bits each for L-level gray-scale images. Thus, from the theoretical point of view, our FPGA-based instance-specific solution is much faster than the conventional software solution.

We evaluate the performance of our hardware using a timing analyzer for the Xilinx VirtexII series FPGA XC2V8000. Further, we test our hardware using Spartan2(XC2S150) PCI card Strathnuey [3]. Since the generated hardware attains up to 3000 speed-up factor over the sequential algorithm, our approach is a promising solution. An image matching hardware using an FPGA has been proposed [2]. Their hardware is not

* Work supported in part by the Ministry of Education, Science, Sports, and Culture, Government of Japan, Grant-in-Aid for Exploratory Research (90113133).

instance-specific, does not support gray-scale images, and runs in $O(n^2)$ clock cycles. Thus, our hardware is a significant improvement on the FPGA-based image matching hardware. Also, an isntance-speicific solution for image matching has been shown[6]. However, it does not support gray-scale image and parallel matching.

II. The image difference function

An L-level gray-scale image I of size $m \times m$ is an $m \times m$ two dimensional array with each element (or pixel) taking an integer in the range $[0, L-1]$. The value of an (i,j) pixel $I_{i,j}$ $(1 \leq i,j \leq m)$ of I corresponds to its brightness. In other words, pixel (i,j) is black if $I_{i,j} = 0$ and white if $I_{i,j} = L-1$. We assume that pixel $(1,1)$ is the top of the leftmost column of I. An L-level gray-scale image I is a *binary image* if $L = 2$. An $m \times m$ template image T is an image with "don't care", that is, an $m \times m$ two dimensional array with each element taking either an integer in $[0, L-1]$ or a special value d. An (i,j) pixel $(1 \leq i,j \leq m)$ of T is "don't care" if $T_{i,j} = d$. Let e denote the number of *effective pixels*, which are non-"don't care" pixels in T. The value of e, which depends on the applications, can be much smaller than m^2.

Let D be the function that returns an integer for a template image T and an image I such that

$$D(T,I) = \sum_{T_{i,j} \neq d} |T_{i,j} - I_{i,j}|. \qquad (1)$$

Intuitively, $D(T,I)$ is the sum of the difference of the brightness over all effective pixels. Clearly, $D(T,I)$ takes a larger value if they are less similar. Note that, for a binary template T and a binary image I, their difference is

$$D(T,I) = \sum_{T_{i,j} \neq d} T_{i,j} \oplus I_{i,j}, \qquad (2)$$

where $\oplus$ denotes the exclusive OR operator.

Suppose that an image I is larger than a template image T. Let $n \times n$ $(n > m)$ be the size of image I. Further, let $I[x,y]$ $(1 \leq x,y \leq n-m+1)$ denote an $m \times m$ subimage of I that includes all pixels $I_{i',j'}$ $(x \leq i' \leq x+m-1$ and $y \leq j' \leq y+m-1)$. *The image difference function* $D(T,I)$ between a template T and an image I is

$$D(T,I) = \min_{1 \leq x,y \leq n-m+1} D(T, I[x,y]). \qquad (3)$$

Clearly, $D(T,I)$ is small if I has a similar subimage to T. Also, let D_T denote a function such that $D_T(I) = D(T,I)$.

By evaluating $D(T,I_1), D(T,I_2), \ldots$ in turn, we can retrieve all images in a database of images $I_1, I_2, \ldots$, which have a similar subimage to T. Our goal is to accelerate the computation of evaluating $D(T,I)$. For later reference, let us evaluate the computing time necessary to compute $D(T,I)$ by a software (or a sequential algorithm). For an $m \times m$ template image T with e effective pixels and a subimage $I[x,y]$, the value of $D(T,I[x,y])$ can be computed in $O(e)$ time. Hence, the evaluation of $D(T,I[x,y])$ for all $I[x,y]$ $(1 \leq x,y \leq n-m+1)$ takes $(n-m+1)^2 \times O(e) = O(n^2 e)$ time. Therefore, the task of computing the image difference $D(T,I)$ takes $O(n^2 e)$ time.

III. An image matching hardware for binary image retrieval

In this section, we are going to show our FPGA-based instance-specific hardware that computes $D_T(I)$ $(= D(T,I))$ for a fixed template T and various images I. We start with a binary template T and a binary image I. We then go on to extend our hardware to support gray-scale images later.

Figure 2 illustrates our hardware for $m = 4$ that evaluates $D(T,I)$ using formulas (2) and (3). For simplicity, we assume that, for a template T of size $m \times m$, m pixels of image I can be supplied via PCI-bus in every local clock cycle.

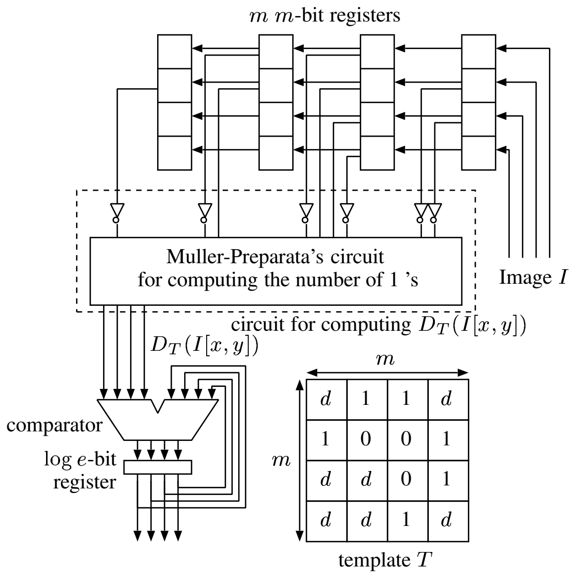

Fig. 2. A hardware implementation of our circuit computing $D_T(I)$.

Next, we are going to show how our hardware works. Let $I_m[i,j]$ denote the set of vertical adjacent m pixels $I_{i,j}$, $I_{i+1,j}$, $\ldots$, $I_{i+m-1,j}$. Note that the m pixels in any $I_m[i,j]$ can be transferred to the register in a local clock cycle. Every pixel in image I of size $n \times n$ is transferred to the m m-bit registers in $n \cdot (n-m+1)$ clock cycles as follows:

for $i \leftarrow 1$ **to** $n-m+1$ **do**
  for $j \leftarrow 1$ **to** n **do**
    Perform the following two operations in parallel:
    1. $I_m[i,j]$ is transfered to the rightmost register;
    2. the m registers are shifted to the left by one.

It should be clear that, subimage $I[1,1]$ is stored in the registers when $(i,j) = (1,m)$. Further, every subimage $I[x,y]$ $(1 \leq x,y \leq n-m+1)$ is stored in the registers when $i = x$ and $j = y+m-1$.

We use a combinatorial circuit that computes $D_T(I[x, y])$ for subimage $I[x, y]$ currently stored in the m m-bit registers. It consists of parallel inversions and the Muller-Preparata's circuit [4, 5] that computes the number 1's in the input bits. For every pixel of template image T, the corresponding register bit or its inversion is connected to the Muller-Preparata's circuit if it is 1 or 0, respectively. Since T has e effective pixels, the Muller-Preparata's circuit computes the sum of e bits, which is equal to the value of $D_T(I[x, y])$.

To compute the minimum of $D_T(I[x, y])$ over all x and y, a comparator and a $\log e$-bit register is used. The comparator computes the minimum of two $\log e$-bit integers. The register is storing the temporary minimum value of $D_T(I[x, y])$ so far. If the current value of $D_T(I[x, y])$ is smaller, then it is stored in the register. It should be clear that, after every pixel in image I is supplied to this circuit, the $\log e$-bit register stores $D_T(I)$.

Next, let us evaluate the performance and the hardware resouses used by our hardware. As we discussed, our hardware computes $D_T(I)$ in less than n^2 clock cycles. The Muller-Preparata's circuit [4] that counts the number of 1's in e bits has $O(e)$ gates. Furhter, the $\log e$-bit comparator has no more than $O(\log e)$ gates. The m m-bit registers uses m^2 flips-flops and the $\log e$-bit register uses $\log e$ $(< 2 \log m)$ flip-flops. Thus, our hardware uses $O(e)$ gates and $O(m^2)$ flip-flops.

IV. Parallel image matching for binary images

This section is devoted to show our parallel image matching architecture for further acceleration.

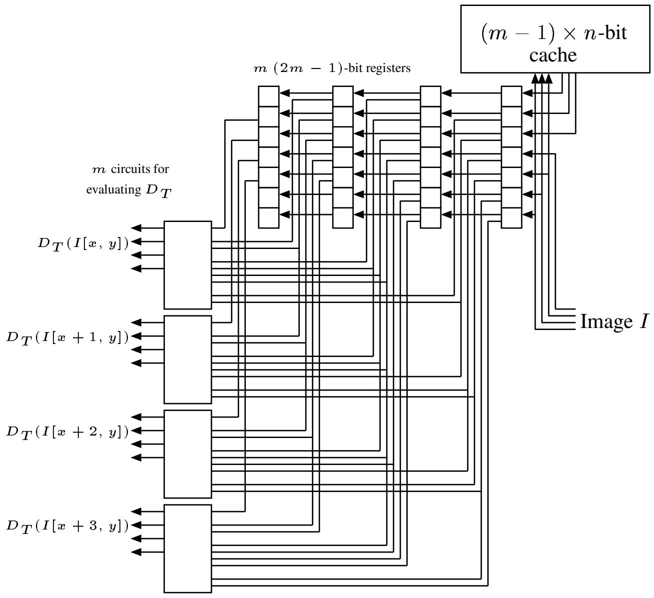

Fig. 3. Implementation of parallel m circuits computing $D_T(I)$.

Figure 3 illustrates a part of our parallel image matching hardware. In order to reduce the number of clock cycles, we use m circuits computing $D_T(I[x, y]), D_T(I[x+1, y]), \ldots, D_T(I[x+m-1, y])$ in parallel. Note that, $D_T(I[x, y]), D_T(I[x+1, y]), \ldots, D_T(I[x+m-1, y])$ combined correspond to a subimage with $(2m-1) \times m$ pixels. Thus, we use m registers with $(2m-1)$ bits each to store a subimage of $(2m-1) \times m$ pixels. Again, we assume that m pixels in an image I are supplied in every local clock cycle. Hence, vertical $(2m-1)$ pixels cannot be transferred to the rightmost $(2m-1)$-bit register in every local clock cycle. To supply the $(2m-1)$ pixels to the register in every local clock cycle, we use an $(m-1) \times n$-bit cache, that is, a cache with $(m-1)$-bit data and $\log n$-bit address.

In what follows, we will describe how our parallel image matching hardware illustrated in Figure 3 works. An image I is transferred to the registers as follows.

for $i \leftarrow 1$ **to** $\frac{n}{m}$ **do**
  for $j \leftarrow 1$ **to** n **do**
    Perform the following four operations in parallel:
    1. $I_m[(i-1) \cdot m + 1, j]$ is transfered to the rightmost register;
    2. $I_{m-1}[(i-1) \cdot m + 2, j]$ is transfered to the address j of the cache;
    3. the $m-1$ pixels stored in the address j of the cache is transferred to the rightmost register;
    4. the m registers are shifted by one to the left.

As illustrated in Figure 3, m circuits for evaluating D_T are connected to the m $(2m-1)$-bit registers. When the registers are storing $I[x, y], I[x+1, y], \ldots, I[x+m-1, y]$, the m circuits compute $D_T(I[x, y]), D_T(I[x+1, y]), \ldots, D_T(I[x+m-1, y])$ in parallel. Their minimum is computed by a tree of comparators in obvious way. Consequently, our parallel hardware computes $D_T(I)$ in $n \cdot \frac{n}{m} = \frac{n^2}{m}$ clock cycles. Further it has $O(m^2)$ flip-flops for registers, $O(me)$ gates, and an $(m-1)n$-bit block RAM. We have also constructed the parallel version of our gray-scale image matching hardware similarly to that for binary images. However, due to the page limitation, we omit the description of parallel gray-scale image matching hardware.

V. The Performance evaluation

The main purpose of this section is to evaluate the performance of our image matching hardware to compute $D_T(I)$ for templates T with 32×32 pixels and images I with 1024×1024 pixels.

Before we evaluate the performance of our hardware, we will show the computing time by the software approach as counterparts. Table I shows the computing time of $D(T, I)$ on a 2.4GHz Pentium4-based PC. For $L = 4$ and 16, the value of $D(T, I)$ is computed by evaluating formulas (1) and (3) in Section II combined. Since $D(T, I[x, y])$ is evaluated using formula (1), the computing time is proportional to e. For $L = 2$, we use the bitwise XOR operation of a word of 32-bit data to evaluate formula (2). Also, we accelerate the computation of the sum in formula (2) using the look-up table storing the number of 1's in a 16-bit data. More precisely, let N be a table of size $5 \times 2^{16} = 320\mathrm{Kbits}$ such that $N[x]$ is storing the number of 1's in a 16-bit word x. The number of 1's in a word of 32-bit data can be computed by looking up table N twice. Note that the computing time in Talbe I does not include the time necessary to build the table N.

TABLE I
THE COMPUTING TIME OF $D(T, I)$ BY SOFTWARE (msec)

$e =$	128	256	512	768
$L = 2$	1256	1345	1652	1797
$L = 4$	1495	2958	5812	8667
$L = 16$	1530	3042	5970	8918

We have tested our image matching hardware using Spartan2(XC2S150) PCI card Strathnuey [3]. This PCI card is connected to the host PC through the 33-MHz 32-bit PCI bus. Table II illustrates the performance of our image matching hardware which includes the clock frequency given by the timing analyzer, the actual time to evaluate $D_T(I)$, the speed-up over the software, the number of used slices over 1728 available slices, and the number of used slice flip-flops over 3456 available flip-flops. Unfortunately, due to the small capacity of XC2S150, we could test our non-parallel hardware for binary images with $e = 128, 256, 512, 768$ effective pixels and 4-level gray-scale images with $e = 128, 256$. For our parallel image matching hardware, we could test for binary images with only $e = 16$ pixels. Consequently, even if we use non-parallel image matching hardware embedded in the small FPGA, $D_T(I)$ can be computed more than 16 times faster than the software. The computing time for binary images is approximately 75msec, which is bounded by the bandwidth of the 33MHz 32-bit PCI bus. More presicely, since $1024 \times (1024 - 32 + 1) = 1.016\mathrm{M}$ words of 32-bit data are transfered in 75msec, the PCI bus sends images in 434Mbit/s, which is close to the actual maximum bandwidth of the 33MHz 32-bit PCI bus.

TABLE II
THE PERFORMANCE OF OUR IMAGE MATCHING HARDWARE IMPLEMENTED ON SPARTAN2

L	e	Freq. (MHz)	Time (msec)	Speed -up	Slices	Slice FFs
2	128	39.6	75.2	16.7	714	902
2	256	35.8	75.2	17.9	926	1018
2	512	32.7	75.0	22.0	1316	1107
2	768	30.1	75.1	23.9	1681	1122
4	128	33.0	76.5	19.5	1334	1747
4	256	34.3	76.2	38.7	1726	1978
2	16	24.1	4.9	-	1682	1741

We have also estimated the performance of our image matching hardware using the VirtexII series FPGA XC2V8000. Again, we assume that a template image T has 32×32 pixels and an image I has 1024×1024 pixels. We have estimated our hardware for randomly generated templates T of size 32×32 with effective pixels $e = 128, 256, 512$ and 768. Due to the stringent page limitation, we omit the performance of our non-parallel image matching hardware. Table III shows the performance of parallel image matching hardware. For $L = 2, 4$, and 16, we use $\frac{m}{\log L} = 32, 16$, and 8 circuits that compute $D_T(I)$ in parallel, respectively. For $L = 16$ and $e = 768$, the hardware uses $\frac{42251}{46592} \approx 90.7\%$ of available slices. For $L = 2, 4$ and 16, the hardware also uses 2, 4, and 8 block RAMs out of 168 block RAMs. The speedup factors over the software solution are in the range 349-3198. Thus, our approach for image matching and retrieval is promising.

TABLE III
THE PERFORMANCE OF OUR PARALLEL IMAGE MATCHING HARDWARE IMPLEMENTED ON VIRTEXII

L	e	Freq. (MHz)	Time (msec)	Speed -up	Slices	Slice FFs
2	128	28.0	1.17	1074	6188	2277
2	256	26.2	1.25	1076	11307	2899
2	512	24.6	1.33	1242	21765	4557
2	768	23.1	1.42	1265	32310	6243
4	128	29.3	2.24	667	6457	3197
4	256	27.3	2.40	1233	11403	3663
4	512	25.6	2.57	2261	21659	5276
4	768	24.2	2.71	3198	32056	7046
16	128	29.9	4.39	349	8860	5071
16	256	27.8	4.72	644	15021	5184
16	512	26.1	5.03	1187	27146	5264
16	768	24.6	5.31	1679	42251	5433

REFERENCES

[1] M. Gavrilov, P. Indyk, R. Motwani, and S. Venkatasubramanian. Geometric pattern matching: A performance study. In *Proc. of Symposium on Computational Geometry*, pages 79–85, 1999.

[2] T. Kean and A. Duncan. A 800Mpixel/sec reconfigurable image correlator on XC6216. In *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pages 382–391, 1997.

[3] Nallatech Ltd. *Strathnuey SPATAN-II PCI card users guide*, 2000.

[4] D. E. Muller and F. P. Preparata. Bounds to complexityies of network for sorting and for switching. *J. ACM*, 22:195–201, 1975.

[5] K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, 197:57–77, 1998.

[6] John Villasenor, Brian Schoner, Kang-Ngee Chia, Charles Zapata, Hea Joung Kim, Chris Jones, Shane Lansing, and Bill Mangione-Smith. Configurable computing solutions for automatic target recognition. In *4th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, 1996.