An FPGA-based Point Pattern Matching Processor with Application to Fingerprint Matching

Nalini K. Ratha
Department of Computer Science
Michigan State University
East Lansing, MI 48824
ratha@cps.msu.edu

Anil K. Jain
Department of Computer Science
Michigan State University
East Lansing, MI 48824
jain@cps.msu.edu

Diane T. Rover
Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824
rover@ee.msu.edu

Abstract

We describe the design and synthesis of a high-performance coprocessor for point pattern matching with application to fingerprint matching using Splash 2 - an attached processor for SUN SPARCstation hosts. Each of the field programmable gate array (FPGA)-based processing elements (PEs) is programmed using VHDL behavioral modeling. Using the simulation tools, the program logic is verified. The final control bit stream for the PEs is generated using the synthesis tools. The point feature matching coprocessor can run at a peak speed of 17.1 MHz per feature vector of a fingerprint. With 65 features per fingerprint, the matching speed has been projected at the rate of $2.6 * 10^5$ *fingerprints/sec. The synthesized coprocessor was tested on a 10,000 fingerprint database.*

Keywords: Reconfigurable hardware, fingerprint matching, FPGA, hardware-software codesign, Splash 2

1 Introduction

Point pattern matching, i.e., finding the correspondence between two sets of points in an m-dimensional space, is a fundamental problem in many computer vision tasks. For example, rigid object recognition using point features can be considered as an instance of point pattern matching. In motion analysis, point pattern matching is used to solve the correspondence problem. In remote sensing applications, point pattern matching is used in image registration.

For the general case of point pattern matching, where no *a priori* knowledge about the two sets of points is available, many algorithms have been described in the literature [1, 2, 3, 4, 5, 6]. Baird's $O(n^2)$ algorithm, where n is the numbers of points in each of the two point sets, becomes more complex when the number of points are not same in the two sets. Vinod et al. [2] propose a neural network for point pattern matching after formulating the problem as a 0-1 integer programming problem. A genetic algorithm has been suggested by Ansari et al. [4]. Most of these algorithms do not permit distortion of the points.

We restrict our focus only to the following scenario motivated from the problem of fingerprint matching. The two point sets can have different number of points. We do not handle scaling and rotation, but allow elastic distortion. Due to elasticity of the skin and non-ideal nature of the process involved in collecting fingerprint impressions, distortions of the feature vectors are inevitable. In fingerprint matching, we are interested in the set of "paired features" between the query fingerprint and database (reference) fingerprints. This process is repeated over all the records in the fingerprint database. Typically, there are millions of fingerprint records in the database. In order to provide a quick response time, special hardware accelerators are needed for matching.

Fingerprint-based identification is the most popular biometric technique used in automatic personal identification [7]. Law enforcement agencies use it routinely for criminal identification. Now, it is also being used in several other applications such as access control for high security installations, credit card usage verification, and employee identification [7]. The main reason for the popularity of fingerprints as a form of identification is that the fingerprint of a person is unique and remains invariant through age.

A fingerprint is characterized by ridges and valleys. The ridges and valleys alternate, flowing locally in a constant direction (see Figure 1). A closer analysis of the fingerprint reveals that the ridges (or the valleys) exhibit anomalies of various kinds, such as ridge bifurcations, ridge endings, short ridges, and ridge crossovers. Eighteen different types of fingerprint features have been enumerated in [8]. Collectively, these features are called *minutiae*. For automatic feature extraction and matching, the set of fingerprint features is restricted to two types of minutiae: ridge endings and ridge bifurcations. Ridge endings and bifurcations are

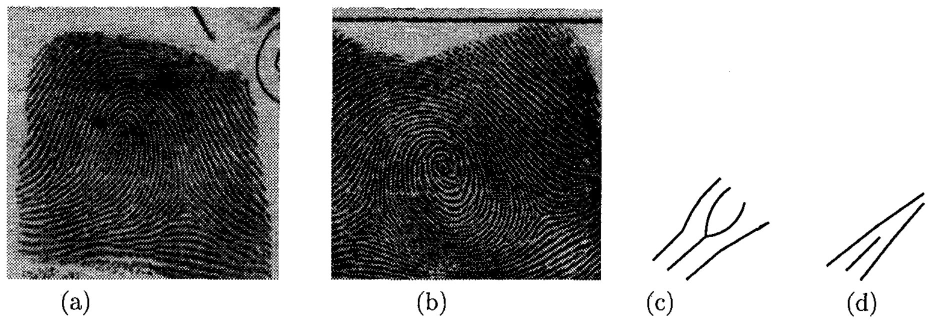


Figure 1: Gray level fingerprint images and two commonly used fingerprint features : (a) Loop; (b) Whorl; (c) Ridge bifurcation; (d) Ridge ending.

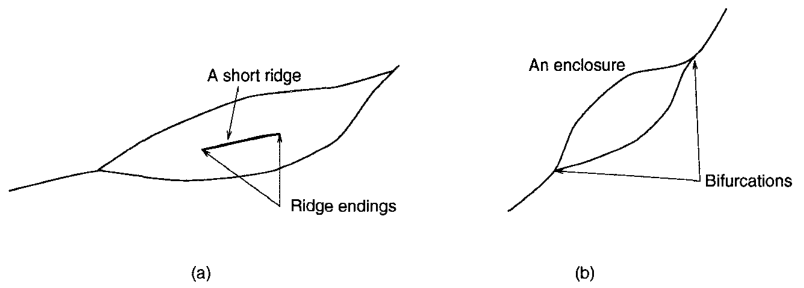


Figure 2: Complex features as a combination of simple features: (a) Short ridge; (b) Enclosure.

shown in Figures 1(c) and 1(d). We do not make any distinction between these two feature types since data acquisition conditions such as inking, finger pressure, and lighting can easily change one type of feature into another. More complex fingerprint features can be expressed as a combination of these two basic features. For example, an enclosure can be considered as a collection of two bifurcations, and a short ridge can be considered as a collection of a pair of ridge endings. These features are shown in Figure 2.

In order to provide a reasonable response time for each query, commercial systems use dedicated hardware accelerators or application-specific integrated circuits (ASICs). While application-specific architectures and ASICs have been designed to meet the computing requirements of complex image processing tasks, such designs have the following two major limitations: (i) once fabricated, they are difficult to modify; and (ii) the cost of building special-purpose application accelerators is very expensive for low-volume applications. Both of these limitations have been the driving force behind the design of custom computing machines (CCMs) using reconfigurable logic arrays known as field programmable gate arrays (FPGAs). An attached processor built with FPGAs can overcome the two limitations noted above. High performance is achieved by exploiting an important principle: most of the processing time of a compute-intensive job is spent within a small portion of its execution code [9], and if an architecture can provide efficient computation for the frequently executed code, then the overall performance can be improved substantially. Portions of the matching algorithm have been identified for implementation on Splash 2, an attached processor for Sun hosts, leaving the remainder to be implemented using software on the host.

In this paper, we describe fingerprint matching as a special case of point pattern matching. A sequential algorithm of $O(mn)$ computational complexity for two point sets with m and n points is presented. We focus on parallelizing this algorithm using Splash 2. The mapping process and the performance results are presented. The parallel algorithm has been implemented on the hardware and tested on a large database.

2 Splash 2 Architecture and Programming Models

The Splash 2 system consists of an array of Xilinx 4010 FPGAs, improving on the design of the Splash 1 based on Xilinx 3090s [10]. Figure 3(a) shows a system-level view of the Splash 2 architecture. Splash 2 is connected to the host through an interface board

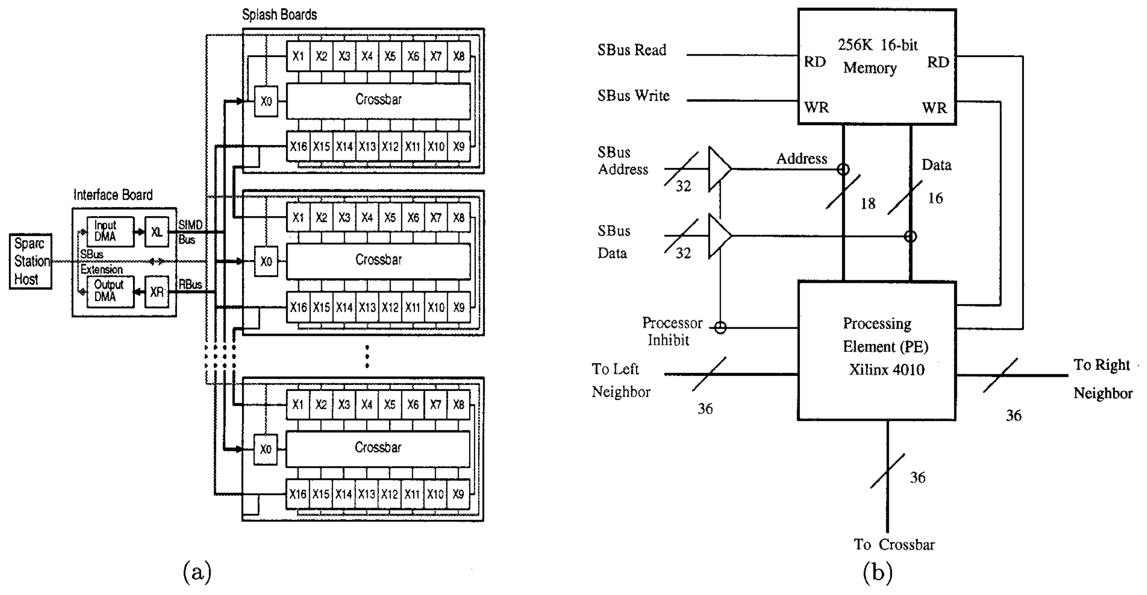


Figure 3: Splash 2: (a) Architecture; (b) One processing element.

that extends the address and data buses. The Sun host can read/write to memories and memory-mapped control registers of Splash 2 via these buses. A detailed description of the system is given in [11, 12]. We describe the major components of the Splash 2 system below. Each Splash 2 processing board has 16 Xilinx 4010s as PEs ($X_1 - X_{16}$) in addition to a seventeenth Xilinx 4010 (X_0) which controls the data flow into the processor board. Each PE has 512 KB of memory. The Sun host can read/write this memory. There is a 36-bit linear data path (SIMD Bus) running through all the PEs. The PEs can read/write data from their respective memory through a private address and data bus after setting appropriate control signals. The PEs are connected through a crossbar that is programmed by X_0. A broadcast path also exists by suitably programming X_0. The processor organization for a PE is shown in Figure 3(b).

The Splash 2 system supports several models of computation, including PEs executing the single instruction on multiple data (SIMD mode) and PEs executing multiple instructions on multiple data (MIMD mode). It can also execute the same or different instructions on single data by receiving data through the global broadcast bus. The most common mode of operation is systolic in which the SIMD Bus is used for data transfer. Individual memory available with each PE makes it convenient to store temporary results and tables.

The Xilinx 4010 consists of 400 (20 $\times$ 20) configurable logic blocks. The structure of a CLB is shown in Figure 4. Programming an FPGA-based computer is different from usual high-level programming. The design automation process consists of two steps: simulation and synthesis. The programming flow for Splash 2 is shown in Figure 5. In simulation, the logic designed using VHDL is verified. This involves comparing the results of the VHDL simulation with those obtained manually or by a sequential program. In synthesis, the main concern is to achieve the best placement of the logic in an FPGA in order to minimize the timing delay. At this point in the design process, the logic circuit may or may not fit on a single FPGA (i.e., be mappable to the configurable logic blocks (CLBs) and flip-flops which are available internal to an FPGA). If it does not fit, the designer needs to revise the logic in the VHDL code and the process is repeated. If it does fit, the timing for the entire digital logic is obtained. In case this timing is not acceptable, the design process is repeated.

To program Splash 2, we need to program each of the PEs (X_1- X_{16}), the crossbar, and the host interface. The crossbar sets the communication paths between PEs. In case the crossbar is used, X_0 needs to be programmed. The host interface takes care of data transfers in and out of the Splash 2 board. A special library is available for these facilities for VHDL programming as described in [11]. The synthesis process involves the following stages: (i) VHDL to XNF translation: to obtain a vendor specific netlist from

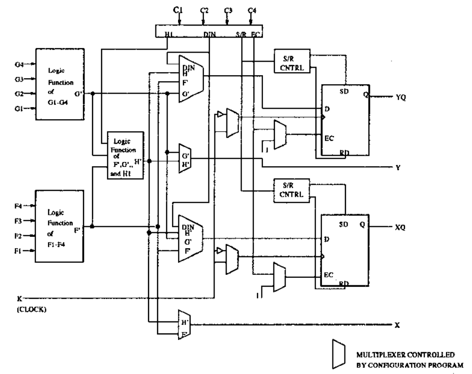
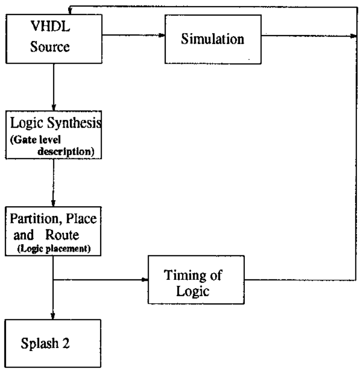


Figure 4: Structure of a Xilinx 4010 CLB.



VHDL Source
Simulation
Logic Synthesis (Gate level description)
Partition, Place and Route (Logic placement)
Timing of Logic
Splash 2

Figure 5: Programming Flow for Splash 2.

the VHDL source code; (ii) Partition, Placement and Routing: to fit the logic generated onto a physical PE; (iii) Delay and Timing Analysis: to analyze the timing and delay; (iv) XNF to bit stream translation; and (v) Bit stream to raw file generation. The raw file is loaded onto each PE, and a configuration file for the crossbar is used to describe the crossbar usage. The host uses these files to control the attached processor through a set of routines callable by a C program. Using the host interface, the memory addressable by each PE can be initialized.

3 Fingerprint Matching Algorithm

The feature extraction process takes the input fingerprint gray-level image and extracts the minutiae features described in Section 1, making no efforts to distinguish between the two categories (ridge endings and ridge bifurcations). In this section, an algorithm for matching rolled fingerprints against a database of rolled fingerprints is presented. A query fingerprint is matched with every fingerprint in the database, discarding candidates whose matching scores are below a user-specified threshold. Rolled fingerprints usually contain a large number of minutiae (between 50 and 100). Since the main focus of this paper is on parallelizing the matching algorithm, we assume that the features (minutiae points) have already been extracted from the fingerprint images. In particular, we assume that the core point of the fingerprint is known and that the fingerprints are oriented properly.

3.1 Minutia Matching

Matching a query and a database fingerprint is equivalent to matching their minutiae sets. Each query fingerprint minutia is examined to determine whether there is a corresponding database fingerprint minutia. Two minutiae are said to be *paired or matched* if their components (x, y, θ) are equal within some tolerance after registration, which is the process of aligning the two sets of minutiae along a common core point (see section 3.2 for precise definitions). Three situations arise as shown in Figure 7.

1. A database fingerprint minutia matches the query fingerprint minutia in all the components (paired minutiae);
2. A database fingerprint minutia matches the query fingerprint minutia in the x and y coordinates, but does not match in the direction (minutiae with unmatched angle);
3. No database fingerprint minutia matches the query fingerprint minutia (unmatched minutia).

Of the three cases described above, the minutiae are said to be paired only in the first case.

3.2 Matching Algorithm

The following notation is used in the sequential and parallel algorithms described below. Let the query fingerprint be represented as an n-dimensional feature vector $\mathbf{f^q} = (\mathbf{f^q_1}, \mathbf{f^q_2}, \ldots\ldots, \mathbf{f^q_n})$. Note that each of the n elements is a feature vector corresponding to one minutia, and the i^{th} feature vector contains three components, $\mathbf{f_i} = (f_i(x), f_i(y), f_i(\theta))$.

The components of a feature vector are shown geometrically in Figure 6. The query fingerprint core point is located at (C^q_x, C^q_y). Similarly, let the r^{th} reference (database) fingerprint be represented as an m_r-dimensional feature vector $\mathbf{f^r} = (\mathbf{f^r_1}, \mathbf{f^r_2}, \ldots\ldots, \mathbf{f^r_{m_r}})$, and the reference fingerprint core point is located at (C^r_x, C^r_y).

Let (x^t_q, y^t_q) and (x^b_q, y^b_q) define the bounding box for the query fingerprint, where x^t_q is the x-coordinate of the top left corner of the box and x^b_q is the x-coordinate of the bottom right corner of the box. Quantities y^t_q

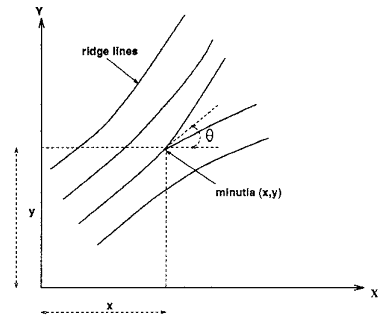


Figure 6: Components of a minutia feature.

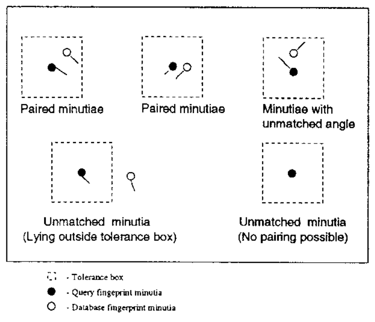

Paired minutiae
Paired minutiae
Minutiae with unmatched angle
Unmatched minutia (Lying outside tolerance box)
Unmatched minutia (No pairing possible)
- Tolerance box
● - Query fingeprint minutia
○ - Database fingerprint minutia

Figure 7: Possible scenarios in minutia matching.

and y_q^b are defined similarly. A bounding box is the smallest rectangle that encloses all the feature points. Note that the query fingerprint $\mathbf{f^q}$ may or may not belong to the fingerprint database $\mathbf{f^D}$. The fingerprints are assumed to be registered with a known orientation. Hence, there is no need of normalization for rotation. The matching algorithm is based on finding the number of paired minutiae between each database fingerprint and the query fingerprint. It uses the concept of minutiae matching described in Section 3.1. In order to reduce the amount of computation, the matching algorithm takes into account only those minutiae that fall within a common bounding box. The common bounding box is the intersection of the bounding box for query and reference (database) fingerprints. Once the count of matching minutiae is obtained, a matching score is computed. The matching score is used for deciding the degree of match. Finally, a set of top scoring reference fingerprints is obtained as a result of matching. In order to accommodate the shift in the minutia features, a tolerance box is created around each feature. The size of the box depends on the ridge widths and distance from the core point in the fingerprint.

The sequential matching algorithm is described in Figure 8. In the sequential algorithm, the tolerance box (shown in Figure 9 with respect to a query fingerprint minutia) is calculated for the reference (database) fingerprint minutia. In the parallel algorithm described in the next section, the tolerance box is calculated for the query fingerprint (as in Figure 9). A similar sequential matching algorithm is described in [13]. Depending on the desired accuracy, more than one finger could be used in matching. In that case, a composite score is computed for each set.

4 Parallel Matching Algorithm

We parallelize the matching algorithm exploiting the specific characteristics of Splash 2 architecture. While performing this mapping, we need to take into account the limitations of the available FPGA technology. Any preprocessing needed on the query minutiae set is a one-time operation, whereas reference fingerprint minutiae matching is a repetitive operation. Computing the matching score involves floating point division. The floating point operations and one-time operations are performed in software on the host whereas the repetitive operations are delegated to the FPGA-based PEs of Splash 2. The parallel version of the algorithm involves operations on the host, on X_0, and on each PE.

One of the main constructs of the parallel algorithm is a lookup table used in translating computations to lookups. The lookup table consists of all possible points within the tolerance box that a feature may be mapped to. The Splash 2 data paths for the parallel algorithm are shown in Figure 10.

4.1 Preprocessing on the Host

The host processes the query and database fingerprints as follows. The query fingerprint is read first and the following preprocessing is done:

1. The core point is assumed to be available. For each query feature $\mathbf{f_j^q}$, j=1, 2, ... n, generate a tolerance box. Enumerate a total of $(t_x \times t_y \times t_\theta)$ grid points in this box, where t_x is the tolerance in x, t_y is the tolerance in y and t_θ is tolerance in θ.

2. Allocate each feature to one PE in Splash 2. Repeat this cyclically, i.e., features 1-16 are allocated to PEs X_1 to X_{16}, features 17-32 are allocated to PEs X_1 to X_{16}, and so on.

3. Initialize the lookup tables by loading the grid points within each tolerance box in step (1) into the memory.

In this algorithm, the tolerance box is computed with respect to the query fingerprint features. The

```
Input: Query feature vector $\mathbf{f^q}$ and the rolled fingerprint database $\mathbf{f^D}=\{\mathbf{f^r}\}_{r=1}^{N}$.
        The $r^{th}$ database fingerprint is represented as an $m_r$-dimensional feature vector and the query feature vector is n-dimensional.
Output: A list of top ten records from the database with matching scores > T.
Begin
    For r=1 to N do
        1. Register the database fingerprint with respect to the core point $(C_x^q, C_y^q)$ of the query fingerprint:
            For i=1 to $m_r$ do
                $f_i^r(x) = f_i^r(x) - C_x^q$
                $f_i^r(y) = f_i^r(y) - C_y^q$
        2. Compute the common bounding box for the query and reference fingerprints:
            Let $(x_q^t, y_q^t)$ and $(x_q^b, y_q^b)$ define the bounding box for the query fingerprint.
            Let $(x_r^t, y_r^t)$ and $(x_r^b, y_r^b)$ define the bounding box for the $r^{th}$ reference fingerprint.
            The intersection of these two boxes is the common bounding box.
            Let the query print have $M_e^q$ and reference print have $N_e^r$ minutiae in this box.
        3. Compute the tolerance vector for $i^{th}$ feature vector $f_i^r$:
            If the distance from the reference core point to the current reference feature is less than K then
                $t_i^r(x) = ldcos(\phi)$,
                $t_i^r(y) = ldsin(\phi)$, and
                $t_i^r(\theta) = k_3$,
            else
                $t_i^r(x) = k_1$,
                $t_i^r(y) = k_2$, and
                $t_i^r(\theta) = k_3$,
                where l, $k_1$, $k_2$ and $k_3$ are prespecified constants determined
                empirically based on the average ridge width,
                $\phi$ is the angle of the line joining the core point and the $i^{th}$ feature with the x-axis,
                and d is the distance of the feature from the core point.
                Tolerance box is shown geometrically in Figure 9.
        4. Match minutiae:
                Two minutiae $\mathbf{f_i^r}$ and $\mathbf{f_j^q}$ are said to match if the following conditions are satisfied:
                $f_j^q(x) - t_i^r(x) \leq f_i^r(x) \leq f_j^q(x) + t_i^r(x)$,
                $f_j^q(y) - t_i^r(y) \leq f_i^r(y) \leq f_j^q(y) + t_i^r(y)$, and
                $f_j^q - t_i^r(\theta) \leq f_i^r(\theta) \leq f_j^q(\theta) + t_i^r(\theta)$,
                where $t_i^r = (t_i^r(x), t_i^r(y), t_i^r(\theta))$ is the tolerance vector.
            Set the number of paired features, $m_p^r = 0$;
            For all query features $\mathbf{f_j^q}$, j=1,2, ... $M_e^q$, do
                If $\mathbf{f_j^q}$ matches with any feature in $\mathbf{f_i^r}$, i=1,2, ..., $N_e^r$, then increment $m_p^r$.
                Mark the corresponding feature in $\mathbf{f^r}$ as paired.
        5. Compute the matching score (MS (q,r)):
                MS(q,r) = $\frac{m_p^r * m_p^r}{(M_e^q * N_e^r)}$.
    Sort the database fingerprints and obtain top 10 scoring database fingerprints.
End
```

Figure 8: Sequential fingerprint matching algorithm.

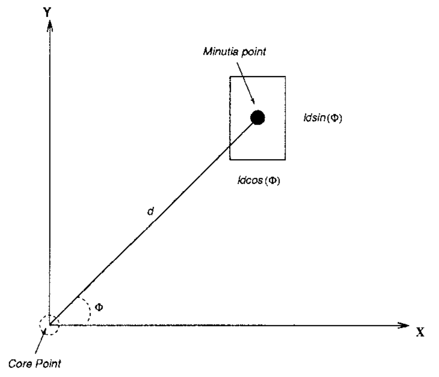


Figure 9: Tolerance box for X- and Y-components.

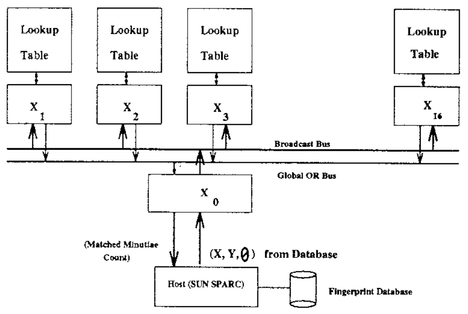


Figure 10: Data flow in parallel algorithm.

host then reads the database of fingerprints and sends their feature vectors for matching to the Splash 2 board.

For each database fingerprint, the host performs the following operations:

1. Read the feature vectors.
2. Register the features as described in step (1) of the sequential algorithm in Figure 8.
3. Send each of the feature vectors over the Broadcast Bus to all PEs if it is within the bounding box of the query fingerprint.

For each database fingerprint, the host then reads the number of paired features m_p^r that was computed by the Splash 2 system, r = 1, ... N, where N is the number of records in the database. Finally, the matching score is computed as in the sequential method.

4.2 Computations on Splash

The computations carried out on each PE of Splash 2 are described below. As mentioned earlier, X_0 plays a special role in controlling the crossbar in Splash 2.

1. Operations on X_0:
   Each database feature vector received from the host is broadcast to all PEs. If it is matched with a feature in a lookup table, the PE drives the Global OR Bus high. When this OR Bus is high, X_0 increments a counter. The host reads this counter value (m_p^r) after all the feature vectors for the current database fingerprint have been processed.
2. Operations on each PE:
   On receiving the broadcasted feature, a PE computes its address in the lookup table through a hashing function. If the data at the computed address is a '1', then the feature is paired, and the PE drives the Global OR Bus high.

5 Performance Analysis

The bit stream files for Splash 2 are generated from the VHDL code using design automation tools from Synopsys and Xilinx. Using the C interface for Splash 2, a host version of the fingerprint matching application is generated. The host version reads the fingerprint database from the disk and obtains the final list of candidates after matching.

The sequential algorithm, described in Section 3.2, executed on a Sun SPARCstation 10 performs at the rate of 70 matches per second on database and query fingerprints that have approximately 65 features. A match is the process of determining the matching score between a query and a reference fingerprint. The Splash 2 implementation should perform matching at the rate of 2.6×10^5 matches per second. This matching speed is obtained from the 'timing' utility. The host interface part can run at 17.1 MHz and each PE can run at 33.8 MHz. Hence, the entire fingerprint matching will run at the slower of the two speeds, 17.1 MHz. Assuming 65 minutiae, on an average, in a database fingerprint, the matching speed is estimated at 2.6×10^5 matches per second. We evaluated the matching speed using a database of 10,000 fingerprints, created from 100 real fingerprints by randomly dropping, adding and perturbing minutiae in a given set of minutiae. The measured speed on a Splash 2 system running at 1 MHz is of the order of 6,300 matches per second on this database. Our experimental Splash 2 system has not yet been run at higher clock rates. Assuming a linear scaling of performance with an increase in clock rate, we would achieve approximately 110,000 matches per second at 17.1 MHz clock speed. We feel that the disparity in the projected and achieved speeds (2.6×10^5 versus 1.1×10^5) is due to different tasks being timed. The time to load the data buffers onto Splash 2 has not been taken into account in the projected speed, whereas this time is included in the time measured by the host in an actual run. We are in the process of timing only the matching component of the code on the system.

The matching algorithm can scale well as the number of Splash 2 boards on the system is increased. Multiple query fingerprints can be loaded on different Splash 2 boards, each matching against the database records as they are transferred from the host. This would result in a higher throughput from the system.

The processing speed can be further improved by replacing some of the soft macros on the host interface part (X_0) by hard macros, where the latter are customized configurations that make efficient use of the FPGA logic. To sustain the matching rate, the data bandwidth should be at a rate of over 250,000 fingerprint records per second (with an average of 65 minutiae per record). This may be a bottleneck for the I/O subsystem.

6 Conclusions

We have addressed the parallel implementation of a point pattern matching algorithm applicable to fingerprint matching. The sequential fingerprint matching algorithm with complexity $O(mn)$ has been successfully parallelized with a complexity of $O(m)$, where m is the average number of minutiae in the database fingerprint and n is the average number of minutiae in a query fingerprint. The Splash 2 architecture is highly suitable for rolled fingerprint matching. The parallel point pattern matching algorithm has been designed to match the Splash 2 architecture, thereby resulting in a substantially improved performance. The algorithm applies a hardware-software design approach to maximize the performance of the overall system.

Acknowledgments

We would like to thank Duncan Buell, Jeff Arnold and Brian Schott of Supercomputing Research Center, Bowie, Maryland for their help and suggestions. We appreciate the assistance provided by the Synopsys and Xilinx university programs. This research was supported by a contract from the Institute for Defense Analyses, Alexandria, Virginia.

References

[1] H. S. Baird, *Model-Based Image Matching using Location.* Cambridge, Massachusetts: The MIT Press, 1985.

[2] V. V. Vinod and S. Ghose, "Point matching using asymmetrical neural networks," *Pattern Recognition*, vol. 8, pp. 1207–1214, August 1993.

[3] D. Skea, I. Barrodale, R. Kuwahara, and R. Poecker, "A control point matching algorithm," *Pattern Recognition*, vol. 26, pp. 269–276, Feb 1993.

[4] N. Ansari, M.-H. Chen, and E. S. H. Hou, "A genetic algorithm for point pattern matching," in *Dynamic, Genetic, and Chaotic Programming* (B. Soucek, Ed.), pp. 353–371, New York: John Wiley and Sons, 1992.

[5] S. Umeyama, "Parameterized point pattern matching and its application to recognition of object families," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 136–144, February 1993.

[6] J. Ton and A. K. Jain, "Registering Landsat images by point matching," *IEEE Trans. on Geoscience and Remote Sensing*, vol. 27, pp. 642–651, September 1989.

[7] B. Miller, "Vital signs of identity," *IEEE Spectrum*, vol. 31, pp. 22–30, February 1994.

[8] Federal Bureau of Investigation, U. S. Government Printing Office, Washington, D. C., *The Science of Fingerprints: Classification and Uses*, 1984.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* San Mateo, California: Morgan Kaufman, 1990.

[10] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," in *Proceedings 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316–322, 1992.

[11] J. M. Arnold and M. A. McGarry, "Splash 2 programmer's manual," Tech. Rep. SRC-TR-93-107, Supercomputing Research Center, Bowie, Maryland, 1994.

[12] D. A. Buell, "A Splash 2 tutorial," Tech. Rep. SRC-TR-92-087, Supercomputing Research Center, Bowie, Maryland, 1992.

[13] J. H. Wegstein, "An automated fingerprint identification system," Tech. Rep. 500-89, National Bureau of Standards, 1982.