Distributed Fingerprint Enhancement on a Multicore Cluster

N.P. Khanyile[1], **J.-R. Tapamo**[2], **and E. Dube**[2]
[1]Council for Scientific and Industrial Research; University of KwaZulu-Natal, South Africa
[2]University of KwaZulu-Natal, South Africa / Council for Scientific and Industrial Research, South Africa

Abstract— *Fingerprint enhancement is a crucial step in fingerprint recognition. The accuracy of the recognition algorithm directly depends on the accurate extraction of features which is achieved through a series of image enhancement steps. Unfortunately, the fingerprint enhancement process consists of a series of computationally expensive image processing techniques. This results in slow recognition algorithms. Researchers have examined ways of improving the performance of fingerprint enhancement algorithms through parallel processing. The majority of such techniques are architecture- or machine-specific and do not port well other platforms. We propose a cheaper and portable alternative through the utilization of mixed-mode distributed and parallel algorithms that make use of multicore clusters for processing strength. We tackle a few design concerns encountered when distributing image processing operations. One such concern is dealing with pixels along the borders of the partitioning axis. The other is distributing data that needs to be processed in blocks rather than pixel-wise.*

Keywords: Fingerprint enhancement, SPMD, MPI, boundary pixels, parallel I/O

1. Introduction

Fingerprint recognition is the most used biometric technique in the commercial industry and crime forensics. Fingerprints are a highly universal and unique trait. A fingerprint pattern is made up of ridges and valleys. The chaotic way in which these ridge and valley structures are formed makes them unique to each individual, including identical twins. Latent prints are easily and unintentionally left behind on surface contact which makes them well adapted for crime scene forensics. The fingerprint recognition process can be fully automated by Automatic Fingerprint Identification Systems (AFIS). The process includes sensing and acquisition; fingerprint enhancement; feature extraction; matching and final reject/accept decision. Of these steps, image enhancement is very crucial yet very computationally expensive. Sensing devices rarely produce perfect ready for use input images. Images are often corrupted by noise and by variations in fingerprint impression conditions. Image enhancement helps remove the effect of these corruptions and makes minutiae more visible to facilitate the subsequent feature extraction. The enhancement procedure takes in an input image and divides it into two categories - recoverable and unrecoverable region. The recoverable regions are well-defined or slightly corrupted but visible and can be recovered by using the neighbouring regions to predict their true structure. Unrecoverable regions are corrupted to an extent that the ridge structure is not visible and neighbouring regions do not provide sufficient information to predict their structure [1].

The rest of the paper is structured as follows: Section 2 gives a brief background to concepts discussed in this paper. Section 3 discusses the data layout and memory distribution. Section 4 gives the algorithms description, while Section 5 introduces ways to deal with boundary pixels. Section 6 presents the performance analysis and Section 7 is devoted to conclusions and future works.

2. Background

2.1 Fingerprint Enhancement

Fingerprint enhancement improves the quality of recoverable regions and removes the unrecoverable regions. The main stages in fingerprint enhancement include normalization, mask region generation, ridge orientation estimation, ridge frequency estimation and ridge filtering [1]-[6].

Normalization is a global operation that adjusts data to fit a certain acceptable region. Subjects rarely present a trait in exactly the same way. There often are variations in impression condition, including dryness of skin, uneven pressure on the scanner surface, etc. Normalization standardizes the intensity of each pixel to lie within a required range. This process reduces the effects of variations that occur during acquisition and reduces chances of false rejections. The mask region generation process segments a normalized image into recoverable and unrecoverable regions per processing block. The mask is then used to separate the Region Of Interest (ROI) from the rest of the image. Hong et al [1] use the amplitude, frequency and variance of each block centered at pixel (i, j) to characterize a sinusoidal-shape wave into the two regions.

Fingerprint patterns are regarded as oriented texture patterns [1], [4]. An orientation image is made up of directional vectors estimated from a normalized image which represent the orientation of local ridges [2]. Numerous techniques exist which can be used to estimate the local orientation of an image [1]-[5], [7]-[9]. The method of averaging square gradients of the gradient covariance matrix seems to be the most widely used approach [1], [2], [7], [8], [10],

[11]. Hong et al use this approach to compute gradients of the normalized image. These gradients are then used in the least mean square orientation estimation algorithm. An image is divided into $W \times W$ non-overlapping blocks then gradients are computed for each block using a Sobel or Marr-Hildreth operator. For each block, an orientation vector is derived by averaging all vectors orthogonal to the x and y gradients. These orientation vectors may not always be accurate due to corruptions and noise. To account for these corruptions, a low-pass filter is used to tune incorrect local ridge orientations on a continuous vector field equivalent to the orientation field [1].

Frequency estimation is a block-wise operation which determines the local frequency of ridges along a direction normal to the local ridge orientation [1]. The frequency estimation procedure requires a normalized image which is divided into $W \times W$ blocks. A frequency estimation algorithm used by Hong et al [1] computes oriented windows of size $L \times W$ for each block. These oriented windows are used to compute x-signatures, defined in [2] as the projection of gray-level values from the oriented window to the ridge orientation along an orthogonal direction. The x-signatures of windows without singularities and minutiae form sinusoidal-shape waves with the same frequency as the ridges in the oriented window. Thus the frequency of ridges can be directly estimated from consecutive x-signatures by calculating the distance between their wavelengths[1].

Gabor filters are bandpass filters that have frequency-selective and orientation-selective properties. Thus the success of the filtering stage of fingerprint enhancement relies on the accurate construction of the orientation field and ridge frequency from the previous stages for parameter tuning. As mentioned earlier, fingerprint patterns are essentially oriented texture patterns. This property together with the ability to estimate local ridge frequency makes Gabor filters ideal for fingerprint filtering as the orientation and frequency parameters of Gabor filters can be tuned to match the local ridge orientation and frequency [2].

Some of the techniques discussed above can be computationally expensive. Researchers are continually looking for ways to improve the processing of the enhancement algorithms [12], [3], [13]. This paper has distributed the process of fingerprint enhancement using a Single Program Multiple Data (SPMD) parallel programming model. Images are split up and assigned to different processing elements which then perform the enhancement techniques on their respective regions of an image.

2.2 Distributed and Parallel Processing

Most traditional software has been written for serial computation. These algorithms work by executing a serial stream of instructions. No two instructions may execute at the same time. As one might deduce, this style of programming produces slow software. When execution time is key in a software, a parallel implementation is preferred.

Parallel programming is a form of programming in which many computations are carried out simultaneously. A large problem is divided into smaller subproblems which are solved concurrently. Parallel computers are classified according to the level at which their hardware supports parallelism. Multi-core systems are single stand alone machines which comprise of multiple processing elements. Clusters, MPPs and grids use multiple computers to complete a task.

Distributed Programming divides a task into several subtasks which are executed concurrently by different processing nodes. A distributed system is made up of a collection of autonomous computers that communicate through a network. The main difference between parallel and distributed systems lies in the memory usage. Parallel systems are usually referred to as shared memory systems. All processors access shared memory for their I/O operations. The shared memory can be used to pass information between processors.

Distributed systems on the other hand use local memory. Each processor reads and writes to its own private memory. Information is passed between processors using a technique known as message passing. Message passing provides a way for computing nodes to share information. A few message passing paradigms exist, including PVM and MPI. In this paper, implementations were performed using MPI. This paper uses a mixed-mode distributed and parallel processing model. MPI is used for coarse-grain across processor parallelism with OpenMP threads for fine-grain intra-processor parallelism.

2.3 MPI

MPI is a message-passing paradigm used to primarily address the message-passing in parallel programming models. It allows for efficient communication between processors by avoiding memory-to-memory copying, allowing overlaps in computation and communication [14]. This research uses Open MPI 1.4.2 which is an open source MPI-2 implementation. MPI provides hardware abstraction, hence code written in MPI is portable and can be run on heterogeneous systems. Processors can only read and write to their local memory. Communication between processes, although crucial is an expensive operation, as such it must be kept at a minimum [14].

2.4 OpenMP

OpenMP is an open specification for multiprocessing. It is a shared memory programming model which is normally used for fine-grain parallelism. OpenMP does not provide message passing capability which makes it inappropriate for distributed processing on clusters. The parallelism on OpenMP is explicit, allowing the user control over the parallelization. OpenMP threads share the same address space, however, each thread has its own private memory, but sees the global memory.

3. Distributed Memory Model and Data Layout

Most fingerprint enhancement techniques operate in a block-wise manner as opposed to pixel-wise. This makes distributing the data a slightly more complex. There is a need to ensure that all partitions are in multiples of the processing block size. Let I be an $N \times M$ gray scale image and np be the number of processors. Let us show how the data is distributed when a processing block of size $W \times W$ is used.

The remainder theorem states that if $r, d \, \epsilon \, \mathbb{N}$ with $d > 0$, then $\forall a \in \mathbb{R}, \quad \exists q \, \epsilon \, \mathbb{N}$ such that:

$$a = qd + r \tag{1}$$

where $0 \leq r < d$. Let N' be the number of columns on each processor. We can express $(\frac{N}{np})/W$ in terms of equation (2):

$$\frac{N}{np} = qW + r \tag{2}$$

The difference between the integer r and window size W represents the number columns that each processor needs to add to $\frac{N}{np}$ in order to have local columns that are a multiple of W. Hence:

$$N' = \frac{N}{np} + (W - r) \tag{3}$$

This ensures that the partitions are in multiples of W so that no pixels are left unprocessed. The image is split into np sub-images of size $N' \times M$, each operated on by its respective processor. MPI offers parallel I/O, where each processor is responsible for its own I/O operations. Parallel I/O allows computations and I/O operations to overlap. This increases performance as it reduces chances of idle processors which are waiting on I/O operations in order to continue processing. Idle processors can cause massive overheads. Fig. 1 illustrates parallel I/O. The image is split up and assigned to different processors. Each processor works on its own region of the file and writes back the results to a corresponding region on the output image.Using parallel I/O leverages performance of a distributed algorithm. Algorithms with serial I/O where only the master node is responsible for I/O operations suffer from overhead caused by idle processors, because while the master is performing I/O, the rest of the processors are not doing anything. The enhancement algorithm presented here makes use of parallel I/O.

Distributed image processing often requires excessive inter-processor communication in order to access neighbour information to process boundary pixels for pixel-wise operations. Communication (message passing) is associated with large amounts of overhead, and should be kept at minimum.

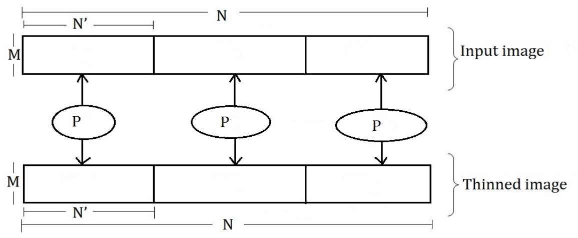

Fig. 1: Parallel input/output diagram showing how I/O is performed by the algorithm.

4. Image Enhancement Algorithm Description

The fingerprint enhancement algorithm takes in a raw gray-scale digital image as input, applies a series of steps then finally outputs a thinned binary image ready for feature extraction. This algorithm follows a sequence: Normalization; Mask region estimation; Orientation field estimation; Ridge frequency estimation; Ridge filtering; Binarization; Ridge thinning. All these operations are block-wise operations with an exception of normalization, binarization and thinning which are pixel-wise operations. In order to facilitate thinning which is a pixel-wise operation depending on neighbour data, two solutions overlap the data across processors, while the third solution depends on message passing. Since the block-wise operation require the sub-images to be in multiples of processing windows, we have to overlap the window containing the boundary pixels for all processors. This facilitates the two overlapping solutions discussed in sections 5.2 and 5.3.

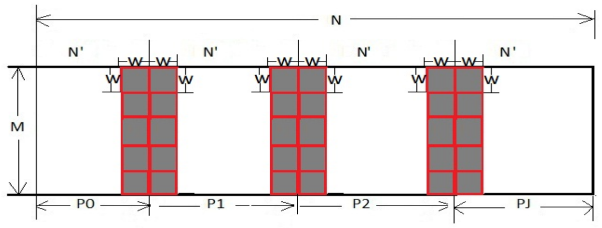

Fig. 2: Data overlapped along processor boundaries. Each overlap portion is of size $W \times M$ in order to ensure that the overall sub-image sizes are in multiples of processing block sizes

We thus modify the initial sub-image size of $N' \times M$ to add ghost cells of size $W \times M$ which gives new sub-image size of $(N' + W) \times M$. For notation simplicity we assign $N' = N' + W$. Fig. 2 shows the decomposition graphically.

4.1 Normalization

We employ the technique used by Hong et al [1] to perform normalization. For a gray-scale partial image I_p, we denote M_p as the estimated mean and VAR_p as the estimated variance. A normalized pixel is computed as follows:

$$N_p(i,j) = \begin{cases} M_0 + \sqrt{\frac{VAR_0(I_p(i,j)-M_p)^2}{VAR_p}} & \text{if } I_p(i,j) > M_p \\ M_0 - \sqrt{\frac{VAR_0(I_p(i,j)-M_p)^2}{VAR_p}} & \text{otherwise} \end{cases} \tag{4}$$

where M_0 and VAR_0 represent the desired mean and variance, respectively. M_0 and VAR_0 have values 0 and 1, respectively. The reason we give the mean and variance these values is because we use a Gaussian function that has a standardized normal distribution.

4.2 Mask Region Estimation

Masking is a type of segmentation that assigns a block to either recoverable or unrecoverable region. Segmentation is an image processing technique which separates strong correlated parts of the image into regions [15]. We use thresholding to segment the image into the two regions. Thresholding transforms an input gray-scale image to a binary image with intensity value 1 representing recoverable regions and intensity value 0 representing unrecoverable regions. We make use of only the standard deviations of the normalized image to estimate the mask. Standard deviations are computed for each block, and if they fall below a given threshold the block is assigned to unrecoverable region, otherwise it is assigned to recoverable. From the mask image, the ROI can be constructed by removing all the unrecoverable regions in the mask from the normalized image.

4.3 Ridge Orientation Estimation

We estimate the local orientation of pixels per block. Fig. 3 shows an example of a ridge orientation at pixel (i,j). The orientation estimation is a prerequisite step for fingerprint filtering, as Gabor filtering relies on accurate local orientation to work correctly.

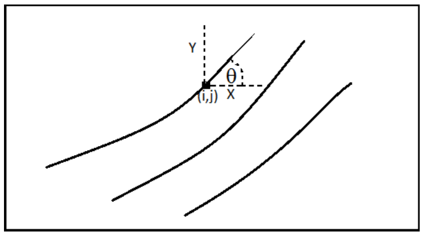

Fig. 3: Orientation of a ridge at pixel (i,j)

Each processor divides its sub-image into $W \times W$ blocks. Then each block computes the x and y directional gradients $G^p_x(i,j)$ and $G^p_y(i,j)$. To compute these gradients, we use first order derivative of a gaussian function. The gradient sub-image is given by the vector:

$$\bigtriangledown G^p(i,j) = [G^p_x(i,j), G^p_y(i,j)]^T \tag{5}$$

The gradient vectors are estimated using Cartesian coordinates. After which they are converted to polar coordinates in order to obtain double angles and squared lengths

$$\begin{bmatrix} G^p_\rho \\ G^p_\phi \end{bmatrix} = \begin{bmatrix} \sqrt{(G^p_x)^2 + (G^p_y)^2} \\ \tan^{-1} \frac{G^p_y}{G^p_x} \end{bmatrix} \tag{6}$$

with $-\frac{1}{2}\pi \leq G^p_\phi \leq \frac{1}{2}\pi$. When these are obtained, the gradient vectors are squared and expressed in terms of the double angles. Using trigonometric identities, the vectors are expressed as follows:

$$\begin{bmatrix} (G^p_\rho)^2 \cos 2G^p_\phi) \\ (G^p_\rho)^2 \sin 2G^p_\phi) \end{bmatrix} = \begin{bmatrix} (G^p_\rho)^2(\cos^2 G^p_\phi - \sin^2 G^p_\phi) \\ (G^p_\rho)^2(2\sin G^p_\phi \cos G^p_\phi) \end{bmatrix} = \begin{bmatrix} (G^p_x)^2 - (G^p_y)^2 \\ 2G^p_x G^p_y \end{bmatrix} \tag{7}$$

The square gradients are then averaged to obtain:

$$\begin{bmatrix} \sum_W [(G^p_x)^2 - (G^p_y)^2] \\ \sum_W 2G^p_x G^p_y \end{bmatrix} = \begin{bmatrix} G^p_{xx} - G^p_{yy} \\ 2G^p_{xy} \end{bmatrix} \tag{8}$$

This gives the variances and crosscovariance of G^p_x and G^p_y averaged over a block of size $W \times W$. To obtain the orientation field we divide the average square gradients by the absolute values of the squared gradients.

$$\left| \sum_W ((G^p_x)^2, (G^p_y)^2) \right| = \sqrt{(G^p_{xx} - G^2_{yy})^2 + (2G^p_{xy})^2} \tag{9}$$

The detailed derivations of some of these equations can be obtained from [8]. The local orientation of the block centered at pixel (i,j) can then be estimated by $(\Phi^p_x(i,j), \Phi^p_y(i,j))$ in the following manner:

$$\Phi^p_x(i,j) = \frac{G^p_{xx} - G^p_{yy}}{\sqrt{(2G^p_{xy})^2 + (G^p_{xx} - G^p_{yy})^2}} \tag{10}$$

$$\Phi^p_y(i,j) = \frac{2G^p_{xy}}{\sqrt{(2G^p_{xy})^2 + (G^p_{xx} - G^p_{yy})^2}} \tag{11}$$

The orientation field is smoothed using a low-pass Gaussian filter to reduce possible effects of noise. The field is filtered as follows:

$$\Phi'^p_x(i,j) = \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \Im(u,v)\Phi^p_x(i-uw)(j-vw) \tag{12}$$

$$\Phi'^p_y(i,j) = \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \Im(u,v)\Phi^p_y(i-uw)(j-vw) \tag{13}$$

where $\Im$ denotes a Gaussian low-pass filter of size $w_\Phi \times w_\Phi$. The final orientation sub-image image is given by:

$$O_p(i,j) = \frac{\pi + tan^{-1}(\frac{\Phi'^p_x(i,j)}{\Phi'^p_y(i,j)})}{2} \tag{14}$$

4.4 Frequency Estimation

We use the approach given by Hong et al to perform local ridge frequency estimation. The x-signature signals form discrete sinusoidal-shape waves which consists of the same frequencies as ridges in the oriented window. This can be used to directly estimate the local ridge frequencies by averaging the number of pixels between the wavelengths, denoted as $\tau(i,j)$. The ridge frequency F_p for a block centered at pixel (i,j) is thus computed as:

$$F_p(i,j) = \frac{1}{\tau(i,j)} \tag{15}$$

Corrupted blocks and ones that contain singularities and minutiae do not form well-defined sinusoidal-shape waves. For such blocks, an estimation for F_p is interpolated from neighbouring blocks [1].

4.5 Ridge Filtering

Gabor filters are used to remove noise and preserve the true ridge structure. The following is an even-symmetric Gabor filter given by a cosine wave modulated by Gaussian [2]:

$$H(x,y;\theta,f) = exp\left\{-\frac{1}{2}\left[\frac{x_\theta^2}{\sigma_x^2} \quad + \quad \frac{y_\theta^2}{\sigma_y^2}\right]\right\}, \tag{16}$$

$$x_\theta = x\cos\theta + y\sin\theta, \tag{17}$$

$$y_\theta = -x\sin\theta + y\cos\theta \tag{18}$$

where θ is the orientation of a Gabor filter, f is the frequency cosine wave, σ_x and σ_y are standard deviations of the Gaussian envelope along the x and y axes, respectively. In order for a Gabor filter to convolute a pixel (i,j) belonging to processor p, it requires the corresponding orientation pixel $O_p(i,j)$ and frequency pixel $F_p(i,j)$ of that pixel. The enhanced pixel, $E_p(i,j)$, is computed as follows:

$$E_p(i,j) = \sum_{u=-\frac{w_x}{2}}^{\frac{w_x}{2}} \sum_{v=-\frac{w_y}{2}}^{\frac{w_y}{2}} G(u,v;O_p(i,j),F_p(i,j))(N_p(i-u,j-v)) \tag{19}$$

where G denotes a Gabor filter and N_p denotes the normalized fingerprint sub-image of processor p, and w_x and w_y are the width and height of the Gabor mask, respectively. Hong et al fixes both σ_x and σ_y to 4.0. This becomes problematic when there are variations in the value of a ridge frequency. It can lead to non-uniform enhancement [2]. Raymond [2] used the values σ_x and σ_y such that they are dependent on the ridge frequency parameter.

$$\sigma_x = k_x F_p(i,j), \tag{20}$$

$$\sigma_y = k_y F_p(i,j), \tag{21}$$

where k_x and k_y are some constant variables. In order to accommodate Gabor waveforms of different sized bandwidths, [2] set the filter size to depend on standard deviations parameters

$$w_x = 6\sigma_x, \tag{22}$$

$$w_y = 6\sigma_y \tag{23}$$

where w_x and w_y are the width and height of the Gabor filter mask, and σ_x and σ_y are the standard deviations of the Gaussian envelope along the x and y axis, respectively.

4.6 Binarization

Binarization is a pixel-wise operation which converts a gray-scale fingerprint image into a binary image with 1-valued pixels representing ridges and 0-valued pixels representing the valleys. The gabor filters used to enhance the fingerprint images have DC-balanced waveforms resulting in filtered images with zero mean pixel values [2]. Binarization is achieved through the thresholding technique described in earlier sections. The mean value of 0 is used as the global threshold to transform the sub-image E_p into binarized sub-image B_p as follows:

$$B_p(i,j) = \begin{cases} 1 & \text{if } E_p(i,j) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where B_p is the binary sub-image of processor p and E_p is the filtered sub-image.

4.7 Thinning

Thinning reduces the binarized images to unit width skeletons. This reduces the amount of data the minutiae extractor has to process and helps to make critical features such as bifurcations, lakes and ridge endings more visible and easier to extract. Thinning is a pixel-wise operation which requires neighbouring pixel values to make a decision about the deletion of a particular pixel. The thinning algorithm used in this paper is based on work presented on [16].

I/O occurs only twice in the algorithm presented on this paper; when the raw image is first read, and at the end of thinning. Since thinning is a pixel-wise operation which relies on neighbour data, extra care has to be taken when performing thinning and I/O. Pixels along the partitioning axis do not have access to their neighbouring pixels and hence can not be correctly processed. There are two ways around this:

1) Inter-processor communication to exchange boundary pixels (known as halo exchange)
2) Overlapping processor data along the boundaries

5. Dealing With Boundary Pixels

Distributed image processing algorithms are required to provide a way in which the boundary pixels can access their neighbouring processors boundary data, as it becomes impossible to process a pixel without knowledge of its neighbouring pixels.

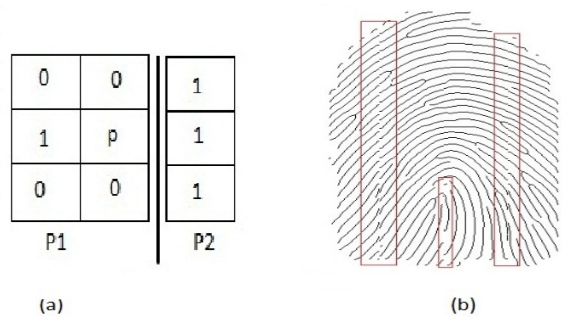

Fig. 4: (a) Boundary pixels between P1 and P2 (b) Example of an image processed disregarding neighbour data

Fig. 4 shows an example of a connected component being partitioned. Boundary pixel p according to processor P1 has a total value of one 1-valued neighbours, which by definition of the thinning algorithm on Section 4.7 entails that p is a ridge ending pixel. This means p will be treated as an endpoint and preserved even though when we look at P2 we see it is in fact not a ridge ending. Kwok [17] defined different contour configurations at the borders of the sections by using chain code representations for border pixels. The configurations help preserve connectivity of components along the borders. In order to access neighbouring processor boundary data, a communication channel may need to be established between the processors which is often expensive. As an alternative, processors may be allowed overlapping access to boundary data. The image is partitioned into subarrays which are then assigned to different processing nodes. In order for boundary pixels to access their neighbouring pixels, processors need to overlap data along boundaries or use message passing to share data. The overlapped areas of subarrays are usually referred to as ghost cells.

Whenever more than one processor access the same file region for a read and write operation, we often run into nasty racing conditions. Data race occurs when a processor writes to a file region that has not yet been read by all processors that are required to read before any write, or when processors interleave their write operations. This can lead to disastrous results. It is for this reason that MPI requires a developer to enforce operational atomicity. Consider a scenario where column-wise overlapping data is being written to a file. If the write operation is not atomic, it might be interleaved and since processors arrive at their write operations in any arbitrary order, it is impossible to know before hand which processor will write after which. This could result in a final solution that does not reflect the actual computation configuration.

The next three subsections present some solutions to dealing with boundary pixels.

5.1 Halo Exchange

In order to process boundary pixels, neighbouring processors establish communication along the boundaries. A processor is allocated extra memory along the boundaries known as ghost cells. These are used to store data from neighbouring processors, which is used only for computational purposes.

Performing a halo exchange consists of processing nodes sharing their data with their neighbouring processors through message passing. A halo exchange is quite an expensive operation which is unscalable. Increasing the number of processors increases the number of communication nodes. Communication often incurs a large overhead which needs to be minimized. Direct memory access is easier to use than message passing. With message passing, processors must agree to communicate. Each processor must first send its buffers and then wait for the corresponding buffers to arrive from neighbouring processors [18].

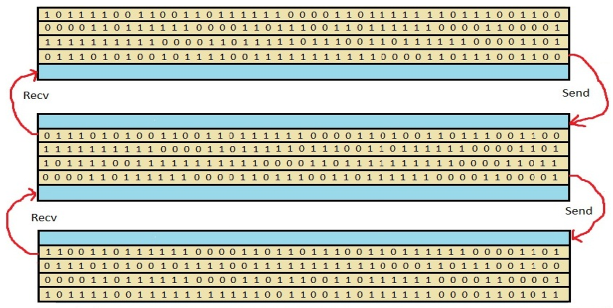

Fig. 5: Halo exchange between three processes

Fig. 5 shows a diagram of 3 processors engaging in a halo exchange. Each processor maintains a ghost cell which will accommodate its neighbours' boundary subarrays. The ghost cells are updated using MPI send/recv prior to any processing.

5.2 File locking

The most naive way to dealing with overlaps is enforcing explicit file locks in order to grant processors exclusive access to overlaps. This can be achieved though the use of mutual exclusion synchronization. When a processor is operating on its region of the file, no other processor can operate on that region. Partitioning data in a column-wise manner leads to noncontiguous data access patterns, making file locking very inefficient. Locking a file region while using this type of partition ultimately locks the entire file, resulting in completely serialized I/O operations which renders MPI parallel I/O useless. Idle processors cause quite a large overhead while waiting on the operating processor to complete its write operation. The solution presented in the next subsection discusses a way around this.

5.3 Process-Rank Ordering

This third approach to dealing with boundary pixels is based on a strategy termed by Liao et al [19] as process-rank ordering. The processors are granted priority levels to be used to give them exclusive access to overlapped data. The higher ranked processor wins when an overlap is encountered. Lower ranked processors then must modify their requests by subtracting the overlaps. Data resulting in overlaps will only be written by the processor with the higher rank. This effectively eliminates all overlaps and achieves atomicity. The overhead in rank ordering is the cost of re-generating access regions for all processors [19].

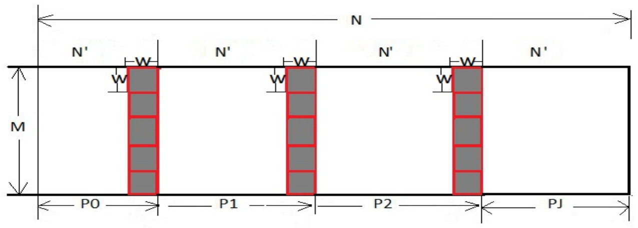

Fig. 6: Overlapped data access using process-rack ordering (adapted from [19])

Fig. 6 shows a graphical view of the file views resulting from process-rank ordering. A file view for processor Pi ($0 < i < j$), is an $M \times N'$ subarray and the file view for P0 and Pj are $M \times (N' - W)$ and $M \times (N' + W)$, respectively. Since each processor surrenders its write to the rightmost column, all overlaps are removed and MPI atomicity is maintained.

6. Experimental Results

6.1 System Configuration

The system used for experiments has the following specifications:

-Model: SuperMicro
-Filesystem : GPFS
-Network : Gigabit Infiniband
-CPU Cores : 80
-CPU Model : Intel Xeon
-CPU Speed : 2.4 GHz
-Peak Performance : 16 TFlops

6.2 Data Set and Performance Analysis

Fig. 7 shows the outputs of the enhancement algorithm.[Note: I/O is only performed twice, these output images are only for experimental purposes]. Here the image has been normalized, masked, segmented, oriented, filtered

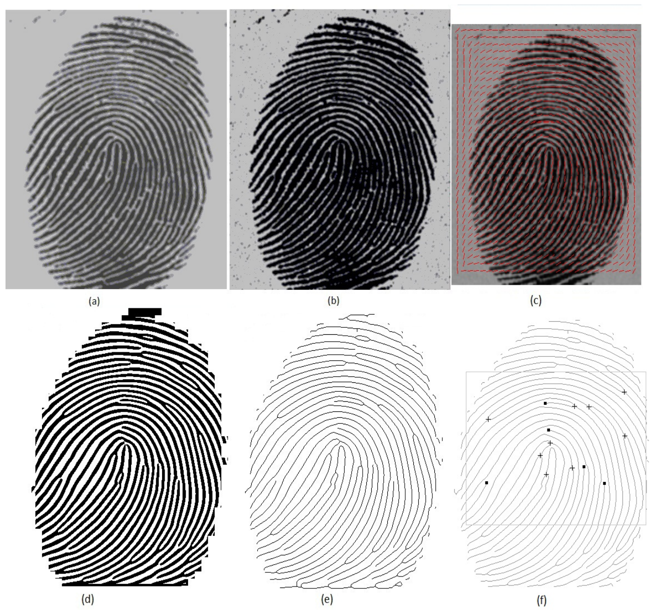

Fig. 7: (a) Original image (b)Normalized image (c) Orientation field (d) Filtered binarized image (e) Final output thinned image (f) Minutiae extracted from the thinned image

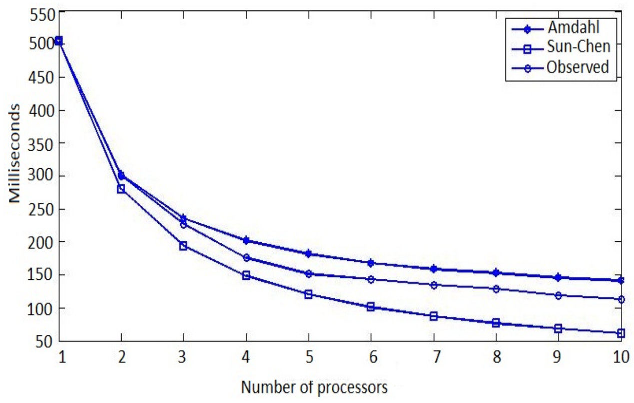

Fig. 8: Expected results vs. actual execution time

and thinned. From here the image is ready for minutiae extraction. The CASIA fingerprint database version 5.0 was used for testing.

Fig. 8 shows the averaged performance of the algorithm tested on 60 images. The expected performance according to Amdahl's law and Sun-Chen's [20] fixed-time model is plotted against the actual observed performance. While Amdahl's model under-estimates the performance gain with an average error of 19.33, Sun-Chen's model over-estimates it with an error of 35.73. The reason behind this is that Sun-Chen's model is for non-distributed multicore systems and thus does not take into account the cost of message passing. While Amdahl's law on the under hand works under the assumption that parallel processing is unscalable [21].

Fig. 9 shows the total execution time separated into actual computation plus communication time. Total communication is affected by network factors along with the frequency and size of messages. We chose to model the halo exchange communication model as it consists of the largest frequency of communication and hence represents the upper extreme of the experiment. It can be seen from the diagram that the communication cost increases with the increase in the number of computation nodes. This is one of the main limitations of distributed processing. While it can provide great scalability and performance gain, if communication is not optimized, the total performance is affected.

The distributed algorithm shows promising results, achieving up to as much as 4.5x speedup. The original Hong et al algorithm in [1] was executed on a *Pentuim 200MHz* PC and achieved a running time of 2.49s on the MSU fingerprint database. While the hardware implementation given in [22] achieved a running time of up to 0.742s.

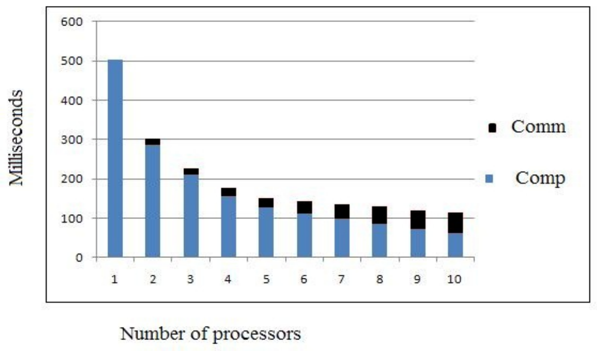

Fig. 9: Experimental results: Total execution time (computation + communication).

7. Conclusions and Future Works

This research shows a promising cheaper way to improve the scalability of image processing applications and successfully shows a great improvement of the fingerprint enhancement algorithm. It shows how boundary pixels on distributed image processing algorithms can be processed in order to ensure accurate results without compromising the performance. The algorithm performs better than the Amdahl's law predicted it would, but does not reach the expectations of Sun-Chen's model, mainly because of the rigorous communication that is associated with the algorithm. All processors aside from first and last are required to send to and receive two arrays from each of their neighbours. Communication causes large overheads due to network latency. The more computation nodes one adds, the more communication needed to complete the halo exchange process. For future work we plan to manipulate the network latency*bandwidth product in order the increase the transmission speed and hence improve the communication performance of the system.

References

[1] L. Hong, Y. Wan, and A. Jain, "Fingerprint image enhancement: Algorithm and performance evaluation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, no. 08, pp. 777–789, Aug 1998.

[2] R. Thai, "Fingerprint image enhancement and minutiae extraction," *Honours Thesis, University of Western Australia*, 2003.

[3] T. Nakamura, M. Hirooka, H. Fujiwara, and K. Sumi, "Fingerprint image enhancement using a parallel ridge filter," *IEEE Proc. Int'l Conf. Pattern Recognition*, vol. 01, no. 08, pp. 536–539, Aug 2004.

[4] N. Ratha and S. C. A. Jain, "Adaptive folw orientation based feature extraction in fingerprint images," *Pattern Recognition*, vol. 28, no. 11, pp. 1657–1672, 1995.

[5] S. Chikkerur, C. Wu, and V. Govindaraju, "A systematic approach for feature extraction in fingerprint images," *Biometric Authentication*, 2004. [Online]. Available: http://www.springerlink.com/index/7a63a3q9qf1p9ttt.pdf

[6] L. Hong, A. Jain, S. Pankanti, and R. Bolle, "Fingerprint enhancement," *IEEE Workshop on Applications of Computer Vision*, pp. 202–207, 1996.

[7] X. Jiang, "On orientation and anisostropy estimation for online fingerprint authentication," *IEEE Trans. Signal Processing*, vol. 53, no. 10, pp. 4038–4049, Oct 2005.

[8] A. Bazen and S. Gerez, "Systematic method for the computation of the directional fields and singular points of fingerprints," *IEEE Trans. Pattern Analysis And Mechine Intelligence*, vol. 24, no. 07, pp. 905–919, July 2002.

[9] M. Kass and A. Wikkin, "Analyzing oriented patterns," *Proc. Int'l. Joint Conf. Artificial Intell.*, 1985.

[10] A. Almansa and T. Linderberg, "Fingerprint enhancement by shape adaptation scale-space operator with automatic scale selection," *IEEE Trans. Image Process.*, vol. 09, no. 12, pp. 2027–2042, Dec 2000.

[11] A. Jain, D. Prabhakar, and L. Hong, "Filterbank-based fingerprint matching," *IEEE Trans. Image Process.*, vol. 09, no. 05, pp. 846–859, May 2000.

[12] D. Bader, J. JaJa, D. Harwood, and L. Davis, "Parallel algorithm for image enhancement and segmentation by region growing with an experimental study," *Proc. 10th Int'l Parallel Processing Symposium*, pp. 414–423, Apr 1996.

[13] N. Ikeda, M. Nakanish, K. Fujii, and T. Hatano, "Fingerprint image enhancement by pixel-parallel processing," *IEEE Proc. Int'l. Pattern Recognition. Machine Intell.*, vol. 03, pp. 752–755, Dec 2002.

[14] M. P. I. Forum, *MPI-2.2: A Message Passing Interface Standard*, Sep 2009. [Online]. Available: http://www.mpi-forum.org.docs/docs.html

[15] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision, Interantional Student Edition*, 3rd ed. Thompson Learning.

[16] Z. Guo and R. Hall, "Parallel thinning with two-subiteration algorithms," *Communications of the ACM*, vol. 32, pp. 359–373, Mar 1989.

[17] P. Kwok, "Thinning in a distributed environment," *Proc. 6th Euromicro Workshop. Parallel and Distributed processing*, pp. 257–263, Jan 1998.

[18] A. Wallcraft, P. Pacheco, and I. Foster. (Last accessed on 23 Aug 2011) Co-array fortran vs MPI. Internet draft. [Online]. Available: http://www.co-array.org/cafvsmpi.htm

[19] W.-K. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and N. Pundit, "Scalable design and implementations for MPI parallel overlapping I/O," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1264–1276, Nov 2006.

[20] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *J. of Parallel and Distributed Computing*, vol. 70, pp. 183–188, 2010.

[21] J. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, pp. 532–533, May 1988.

[22] M. Qin, "A fast and low cost simd architecture fingerprint image enhancement," *MSc Thesis, Technische Universiteit Delft*, 2005.