Accelerating Medical Image Registration Using a SIMD Array

I. K. Jeong¹, M. S. Kang¹, C. H. Kim² and J. M. Kim¹,*
¹School of Electrical Engineering, University of Ulsan, Ulsan, South Korea
²School of Electronics and Computer Engineering, Chonnam National University, Gwangju, South Korea

Abstract - Medical image registration plays an important role in medical imaging in the early detection of cancers. An essential component in most medical registration approaches is resampling algorithms. These algorithms, however, demand tremendous computational power associated with similarity computation. The increasing availability of parallel computers makes parallelizing these tasks an attractive option. This paper presents parallel approaches for the resampling algorithms using a representative parallel Single Instruction, Multiple Data (SIMD) processor array to meet the computational requirements. This paper also presents not only a general theory of resampling algorithms including rotation, scaling, and translation, but also parallel implementations of these algorithms on the SIMD processor array. Experimental results show that parallel approaches achieve a speedup of 2.6x over the FPGA implementations with the same clock frequency of 80 MHz.

Keywords: Medical image registration, parallel processing, SIMD processor arrays, resampling algorithms

1 Introduction

In medical imaging techniques, an important step for intensity-based image registration is resampling [1][2]. It is utilized when a discrete image is transformed into a new set of coordinate points and changes the number of sample (or pixels) per unit length of the directions of the image. During transforming parameters, resampling including rotation, scaling, and translation is estimated by geometrically mapping intensity coordinates in the reference (fixed) image to corresponding locations in the sensed (moving) image. However, these resampling algorithms require tremendous computational power due to the iterative nature of the algorithms.

Application-specific integrated circuits (ASICs) can meet the needed performance for such algorithms, but they provide limited, if any, programmability or flexibility needed for varied application requirements. General-purpose microprocessors (GPPs) offer the necessary flexibility. However, they will not be able to meet the much higher levels of performance required by emerging medical imaging applications on higher resolution images. This is because they lack the ability to exploit the full data parallelism available in these applications.

Among many computational models available for imaging applications, single instruction multiple data (SIMD) processor arrays are promising candidates for application-specific applications including medical imaging since they replicate a simple processing element (PE), data memory, and I/O to provide high processing performance with low node cost. Whereas instruction-level or thread-level processors use silicon area for large multiported register files, large caches, and deeply pipelined functional units, SIMD processor arrays contain many simple processing elements for the same silicon area. As a result, SIMD processor arrays often employ thousands of PEs while possibly distributing and co-locating PEs with the data I/O to minimize storage and data communication requirements.

This paper presents parallel approaches for the resampling algorithms to meet the computational requirements using a representative SIMD array architecture. This paper also evaluates the impact of the parallel approaches on processing performance and compares with the performance of FPGA (Field Programmable Gate Array) solutions. Experimental results show that our parallel approaches achieve a speedup of 2.6x over the FPGA implementation using modified compensated CORDIC [3] with the same clock frequency of 80 MHz.

The rest of the paper is organized as follows. Section 2 presents a SIMD processor array architecture used in this paper and parallel approaches for resampling algorithms. Section 3 describes parallel implementations of rotation, scaling, and translation algorithms. Section 4 evaluates the performance of the resampling algorithms, and Section 5 concludes this paper.

2 Parallel Approaches for Medical Image Registration using SIMD Processor Arrays

2.1 SIMD Processor Array Architecture

A block diagram of the SIMD model [4] used here is illustrated in Figure 1. This SIMD processor architecture is symmetric, having an array control unit (ACU) and an array consisting of processing elements (PEs). When data are distributed, the PEs executes a set of instructions in a lockstep fashion.
With 4x4 pixel sensor sub-arrays, each PE is associated with a specific portion (4x4 pixels) of an image frame, allowing streaming pixel data to be retrieved and processed locally. Each PE has a reduced instruction set computer (RISC) datapath with the following minimum characteristics:

- **ALU** – computes basic arithmetic and logic operations,
- **MACC** – multiplies 32-bit values and accumulates into a 64-bit accumulator,
- **Sleep** – activates or deactivates a PE based on local information,
- **Pixel unit** – samples pixel data from the local image sensor array,
- **ADC unit** – converts light intensities into digital values,
Three-ported general-purpose registers (16 32-bit words),
Small amount of local storage (64 32-bit words),
Nearest neighbor communications through a NEWS (north-east-west-south) network and serial I/O unit.

2.2 Methodology Infrastructure

Figure 2 shows a methodology infrastructure that is divided into three levels: application, architecture and technology. At the application level, an instruction-level simulator was used to profile execution statistics such as cycle count, dynamic instruction frequency and PE utilization by retargeting and optimizing the resampling algorithms based on the architecture and its execution properties. At the architectural level, the heterogeneous architectural modeling (HAM) [5] of functional units for the specified SIMD array was used to calculate the design parameters of the PE configuration. The design parameters were then passed to the technology level. At the technology level, the Generic System Simulator (GENESYS) [6] was used to calculate technology parameters such as latency, area, power and clock frequency. Finally, a design space analysis tool collected and combined the database information (e.g., cycle times, instruction latencies, instruction counts, areas and powers of the functional units) obtained from the application, architectural and technology levels in order to determine execution times, area efficiency and energy efficiency.

2.3 Parallel Approaches for Resampling Algorithms

We implement resampling algorithms in parallel using the SIMD model. When data is distributed, a small pixel region (4x4 pixels) of the entire image space is assigned to each PE as shown in Figure 3. Then, the PEs execute a set of operations in a lockstep fashion.

In medical image registration, resampling is a phase to transform the image I (x, y) to a rotated, scaled, and translated version of this image, I’ (x, y), defined as

\[ I'(x, y) = I(\sigma(x \cos \alpha + y \sin \alpha) - x_0, \sigma(-x \sin \alpha + y \cos \alpha) - y_0) \]  

(1)

where \( \alpha \) is the angle of rotation, \( \sigma \) is the factor of scaling, and \((x_0, y_0)\) is shift amount by \(x_0, y_0\) against x, y axes, respectively.

3 Parallel Implementations of Rotation, Scaling, and Translation

3.1 Rotation Algorithm

The rotation algorithm is an essential component of medical image processing. The basic image rotation operation is defined as

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  \cos \alpha & \sin \alpha \\
  -\sin \alpha & \cos \alpha
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

(2)

A pixel at position (x, y) in the original image is mapped to the position \((x', y')\) in the destination image by following the rotation of angular magnitude \(\alpha\) [7].

Many different implementations of this algorithm have been developed to meet the computation requirements. This study prefers to overcome computational burden by using a parallel implementation of a skew transformation where each pixel is shifted in parallel with each coordinate axis by means of the neighbor communication unit of PE. The skew transformation algorithm takes the rotation matrix in (2) and splits it on the multiplication of three matrices, defined as

\[
\begin{bmatrix}
  \cos \alpha & \sin \alpha \\
  -\sin \alpha & \cos \alpha
\end{bmatrix} =
\begin{bmatrix}
  1 & -\tan(\alpha/2) \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  \tan(\alpha/2) & 0 \\
  0 & 1
\end{bmatrix}
\]

(3)

These three matrices are applied independently to the pixels of the image to find out the new location of a given pixel in the picture. The first and the last matrices cause a skew west in the image. After multiplying the first or the last matrix by the coordinates of a given pixel, (4) is produced.

\[
\begin{bmatrix}
  1 & 0 \\
  -\tan(\alpha/2) & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix} =
\begin{bmatrix}
  x - y \times \tan(\alpha/2) \\
  y
\end{bmatrix}
\]

(4)

where the skew west is a displacement of the rows to the left by the multiplication of the row number by \(\tan(\alpha/2)\).

The second matrix in (3) causes a skew north. After multiplying the second matrix by the coordinates of a given pixel, (5) is produced.
where the skew north is a displacement of the columns to the top by the multiplication of the column number by \( \sin(\alpha) \).

According to the above skew transformation, the image expands after the west and north skews in (3). Since the SIMD processor array is fixed and the image is filled in completely, there is no space to hold the expanded image part. Thus, the image must be compressed first to be able to skew the image. Since the skew west expands the image to the left and the skew north expands to the top, the image is compressed to the right-bottom corner of it, as shown in Figure 4(a).

Fig. 4. Three steps for rotation: (a) compression, (b) skew west, (c) skew north.

Fig. 5. The original image (a) is rotated with different angles: (b) rotated 10°, (c) rotated 30°, (d) rotated 60°, and (e) rotated 90°

3.2 Scaling and Translation Algorithms

From (1), scaling and translation are defined as

\[
I'(x, y) = I(\sigma x - x_0, \sigma y - y_0)
\]

In practice, both scaling and translation are often used together for transformations. Using these algorithms, we can transfer pixels between PEs to achieve a target picture size from the original picture.

3.2.1 Translation

For a parallel implementation of translation, all the pixels are mapped to every PE as shown in Figure 6(a). Then the transformation equation, \( I'(x, y) = I(x - x_0, y - y_0) \) where \( x_0, y_0 \) are translated distances, is applied to the original image, producing a translated image, shown in Figure 6(b).
To perform translation in parallel, the image is equally separated by x axis with the x₀ distance. Then 16 pixels of each PE are transferred to the neighbors with x₀ distance loops. For the y direction, y₀ loops are applied. Since all PEs execute in parallel, the image is translated very fast just by x₀ loops for x axis and y₀ loops for y axis.

3.2.2 Scaling

Image scaling is a frequent operation in medical imaging to enlarge or shrink an image. This section presents a parallel implementation of scaling. An image could be scaled separately in column and row directions. Suppose an image is scaled in column direction. The image is divided into slices in vertical direction. Each column occupies a slice of pixels and then all columns are carried out in parallel relying on PEs which involve those pixels. From (6), we have \( I'(x, y) = I(x, y) \) for the column approach.

Figure 7 shows an example of the image scale down from a resolution 4 by 5 pixels to a resolution 4 by 4. It is equivalent with \( \sigma = \frac{5}{4} \). We assume that the area of a source pixel is 1.0, therefore, if columns are numbered as \( 0, 1, 2, \ldots, i, \ldots \) and \( \{ \text{source} \} = \{ 0, 1, 2, \ldots, i, \ldots \} \) for X-axis, \( x_{\text{source}} = i \). The first target, \( x_{\text{target}} = \frac{1}{2}(\sigma - 1) \). For example, with \( \sigma = \frac{5}{4} \), we have \( x_{\text{target}} = \frac{1}{2}(\frac{5}{4} - 1) = \frac{1}{8} \) and \( x_{\text{target}} = \frac{1}{2}(\sigma - 1) + \sigma i \) as shown in Figure 7.

Finally, one PE contains two types of x values as shown in (7)

\[
\{ x_{\text{source}} = i; x_{\text{target}} = \frac{1}{2}(\sigma - 1) + \sigma i \}
\]

The remaining task is how to determine pixel values of \( I'(x, y) \) of target pixels. Several interpolation algorithms perform this, including nearest neighbor, bilinear, and bicubic interpolations. In this paper, the nearest neighbor interpolation is applied to all pixels based on x-axis as well as related pixel values to calculate the X-axis of all the pixels. Figure 8 illustrates the operation of scale down with \( \sigma = \frac{5}{4} \). We assume that source pixels require 10 PEs and they are scaled down target pixels which require 8 PEs. According to X-axis, a loop is performed 2 times. For each loop, one PE keeps target pixels and source pixels received from a neighbor. Then the PE relies on \( \{ x_{\text{source}}; x_{\text{target}}; \text{the proposed interpolation} \} \) to determine value pixel (i) of the PE.

After only a few loops, the image is scaled. The speed of this algorithm is considerably improved by the operation of PEs in parallel. Figure 9 illustrates a brain image which is scaled up \( \frac{5}{4} \) for X-axis and then \( \frac{5}{4} \) for Y-axis using the nearest neighbor interpolation algorithm.

4 Experimental Results

To implement and determine the performance of the resampling algorithms, we use cycle accurate SIMD simulator [8]. We develop the algorithms in their respective
assembly languages for the SIMD processor array. In this study, an image size of $256 \times 256$ pixels is used. For a fixed $256 \times 256$ pixel system, the number of 4,096 PEs is used because each PE contains $4 \times 4$ pixels. Table 1 summarizes the parameters of the system configuration.

**Table 1. Modeled system parameters.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of PEs</td>
<td>4,096</td>
</tr>
<tr>
<td>Pixels/PE</td>
<td>16</td>
</tr>
<tr>
<td>Memory/PE [word]</td>
<td>256 [32-bit word]</td>
</tr>
<tr>
<td>VLSI Technology</td>
<td>100 nm</td>
</tr>
<tr>
<td>Clock Frequency</td>
<td>80 MHz</td>
</tr>
<tr>
<td>Interconnection Network</td>
<td>Mesh</td>
</tr>
<tr>
<td>intALU/intMUL/Barrel</td>
<td>1 / 1 / 1 / 1 / 1</td>
</tr>
<tr>
<td>Shifter/intMACC/Comm</td>
<td>1 / 1 / 1 / 1 / 1</td>
</tr>
</tbody>
</table>

We evaluate the performance of the algorithms in terms of execution time and sustained throughput, defined in Table 2.

**Table 2. Summary of evaluation metrics.**

<table>
<thead>
<tr>
<th>Execution time</th>
<th>Sustained throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{exec} = \frac{C}{f_{c\circ}}$</td>
<td>$\eta_{exec} = \frac{O_{exec} \cdot U \cdot NPE}{t_{exec}}$ [Gops/sec]</td>
</tr>
</tbody>
</table>

where $C$ is the cycle count, $f_{c\circ}$ is the clock frequency, $O_{exec}$ is the number of executed operations, $U$ is the system utilization, and $NPE$ is the number of processing elements.

Table 3 summarizes the execution parameters for each algorithm in the SIMD processor array. Scalar instructions control the processor array. Vector instructions, performed on the processor array, execute the algorithm in parallel. System utilization is calculated as the average number of active processing elements. This table lists the statistics for specific cases such as rotation with $\alpha = 30^\circ$, scale down and up with factor $5/4$ and translation with the distance of $30\%$ of the original. As expected, the rotation algorithm takes longer time than others due to its inherent complex skew operations. Overall, the execution times are in the order of milliseconds. Thus, these algorithms are executed at a real-time frame rate (30 frame/sec or 33 ms).

**Table 3. Algorithm performance on a 4,096 PE system running at 80 MHz.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Vector Instruction</th>
<th>Scalar Instruction</th>
<th>System Utilization [%]</th>
<th>Total Cycle [cycles]</th>
<th>$t_{exec}$ [ms]</th>
<th>Sustained Throughput [Gops/sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation ($\alpha = 30^\circ$)</td>
<td>160,193</td>
<td>71,759</td>
<td>31.4</td>
<td>232,022</td>
<td>2.9</td>
<td>71</td>
</tr>
<tr>
<td>Scale down ($\sigma = 5/4$)</td>
<td>20,259</td>
<td>9,050</td>
<td>60.3</td>
<td>29,326</td>
<td>0.37</td>
<td>135</td>
</tr>
<tr>
<td>Scale up ($\sigma = 5/4$)</td>
<td>18,967</td>
<td>8,126</td>
<td>63.6</td>
<td>27,124</td>
<td>0.34</td>
<td>145</td>
</tr>
<tr>
<td>Translation ($x_0, y_0=30%$)</td>
<td>6,094</td>
<td>2,766</td>
<td>100</td>
<td>8,862</td>
<td>0.11</td>
<td>227</td>
</tr>
</tbody>
</table>

**Table 4. The execution time comparison of parallel approaches and other approaches.**

<table>
<thead>
<tr>
<th>Angle</th>
<th>SIMD (clock freq. 80 MHz)</th>
<th>FPGA Using Modified Compensated CORDIC [3] (clock freq. 80 MHz)</th>
<th>Reconfigurable FPGAs [9] (clock freq. 20 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>static dynamic</td>
</tr>
<tr>
<td>10°</td>
<td>1.51 ms</td>
<td>-</td>
<td>24.5 ms</td>
</tr>
<tr>
<td>30°</td>
<td>2.9 ms</td>
<td>-</td>
<td>40 ms</td>
</tr>
<tr>
<td>45°</td>
<td>3.99 ms</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>60°</td>
<td>5.16 ms</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>150°</td>
<td>5.7 ms</td>
<td>14.7 ms</td>
<td></td>
</tr>
</tbody>
</table>
Figure 10 shows the distribution of vector instructions for the resampling algorithms. Each bar divides the instructions into the arithmetic-logic-unit (ALU), memory (MEM), communication (COMM), PE activity control unit (MASK), and image loading (PIXEL). The ALU and MEM instructions are computation cycles while COMM and MASK instructions are necessary for data distribution and synchronization of the SIMD processor array. Results indicate that the resampling algorithms are dominated by ALU and MEM operations.

Table 4 shows the performance comparison of our parallel approaches and other approaches including FPGA implementations. Parallel approaches outperform FPGA implementations for image rotation in terms of execution time. For all cases, a 256×256 pixel is used. Our approaches achieve a speedup of 2.6x over the FPGA implementation using modified compensated CORDIC [3] with the same clock frequency of 80 MHz. These results demonstrate that parallel approaches on the SIMD processor array are suitable candidates for performance-hungry medical imaging.

5 Conclusion

The increasing availability of parallel computers makes parallelizing performance-hungry medical imaging tasks an attractive option. This paper presented parallel implementations of resampling algorithms including rotation, scaling, and translation in medical image registration. Using a representative SIMD processor array, the execution times of these algorithms are in the order of milliseconds, executing at a real-time frame rate (30 frame/sec or 33 ms). In addition, our parallel approaches outperform other FPGA implementations, reducing significant computation time. These results demonstrate that parallel approaches on the SIMD processor array are suitable candidates for emerging medical imaging. In the future, we will explore other medical imaging applications on the SIMD array.

6 Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. NRF-2013R1A2A2A05004566)

7 References


