Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.4, No.2, 2013
www.iiste.org

Parallel Implementation of Systolic Array Design for Developing Medical Image Rotation

Mudhafar M. Al-Jarrah [1*] Saad A. Amin [2] Reyadh S. Naom [1]

1. Faculty of Information Technology, Middle East University,
P.O.Box 383, Amman 11831, Jordan

2. Faculty of Engineering and Computing, Coventry University,
Priory Street, Coventry CV15FB, United Kingdom

[*] Email of the corresponding author: mudhafar.aljarrah@gmail.com

Abstract

Many image-processing algorithms are particularly suited to parallel computing, as they process images that are difficult and time consuming to analyse. In particular, medical images of tissues tend to be very complex with great irregularity and variability in shapes. Furthermore, existing algorithms contain explicit parallelism, which can be efficiently exploited by processing arrays. A good example of an image processing operation is the geometric rotation of a rectangular bitmap. This paper presents a set of systolic array designs for implementing the geometric rotation algorithms of images on VLSI processing arrays. The examined algorithm performs a trigonometric transformation on each pixel in an image. The design is implemented as a distributed computing system of networked computers using Parallel Virtual Machine (PVM) model. Each node (computer) in the network takes part in the task in hand – such as image processing – using message passing. Comments and conclusions about the implementation of the design as a distributed computing system are discussed.

Keywords: parallel computing, distributed computing. PVM, image rotation, systolic array.

1. Introduction

The purpose of this paper is to identify a design suitable for implementing image processing algorithms on VLSI processing arrays. We consider techniques for geometric transformations for digital images. One area, which has received considerable attention in recent years, is the design of real-time systems for early processing of sensory data. This paper examines geometric transformations; a category of image processing which includes, amongst others, magnifying, rotating, and shearing of images or parts of images. All of these operations involve changing the position of pixels to create a new, modified image, and are used extensively in graphic design, digital animation and elsewhere. Medical images of tissues tend to be very complex with great irregularity and variability in shapes. The focus here is on the rotation transformation, which has been selected as a good example of a non-trivial geometric transformation [1,2].

Oncological images of tissues are highly irregular with inherent shape complexity and variability, thus their processing is computationally time consuming. Nevertheless, such analysis is imperative in the histopathology of cancer. This paper aims to investigate geometric transformations for the identification and classification of colonic tissue implemented on parallel computers [3,4].

One of the issues surrounding geometric transformations, is that it is highly processor intensive, in particular when presented with massive images. In addition, existing algorithms contain explicit parallelism, which can be efficiently exploited by processing arrays. Therefore, parallel computing has the potential to provide image processing algorithms with an excellent environment to reduce computational time, and improve hardware utilisation and throughput [5,6]. This is of particular importance in the case of large medical digital images, which may consist of millions of pixels. There are different architectures for carrying out these varied operations. A number of systolic designs for 2D convolution and digital filters have been implemented [7-9].

For this purpose, we have developed a model of systolic arrays. Different types of cells are used, which are implemented in a distributed system using the PVM model [10], to improve hardware utilization and throughput. Various modifications of this systolic design are analysed, to handle geometric transformations. This paper is written alongside an experimental distributed image processing software ("ImageDS"), which rotates images using a distributed network of Linux based machines.

2. The Inverse Rotation Transformation Algorithm

The basic algorithm operating on a single processor shall first be described, providing a basis for explaining its operation on multiple processors. It takes an input image and a decimal angle through which that image is to be rotated. Optimised methods of rotating are available, but since the focus here is on the relative speed-up, a simple algorithm shall suffice. The underlying mathsematics of the transformation is contained within the following trigonometric matrix multiplication. This allows each input image pixel at co-ordinates (*x*,*y*) to be rotated by angle degrees to arrive at new coordinates (*x'*, *y'*).

A very basic algorithm iterates over the pixels in the input image, instantiating (*x*, *y*) as the co-ordinate of each pixel in turn. The new output image co-ordinates (*x'*, *y'*) are calculated, as shown in figure 1, and given that these are non-integer values, they would be rounded to integer values before copying to (*x'*, *y'*) the pixel colour from (*x*, *y*).

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Figure 1. The trigonometric transformation used to rotate a point

As an illustration, a pixel is transformed through 45° around the origin (0, 0) in the 5 by 5 picture in figure 2.

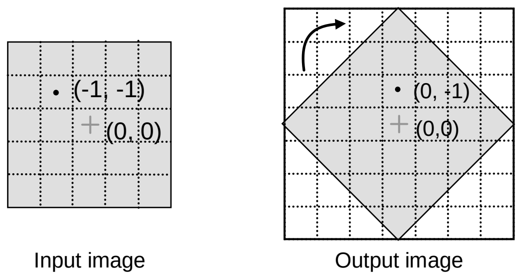

Figure 2. pixel at (-1, -1) transformed to (0, -1.4142) which has to be rounded to (0, -1)

However the rounding of these co-ordinates means that some output pixels are left uncoloured while others receive more than one pixel. This results in an inaccurate rotation of the original, which contains unfilled pixels, therefore a modified version of this algorithm is required. The modified algorithm creates an empty output image in which the rotated output image is to be drawn and having dimensions so that the output image fits into it exactly. It iterates over each integer co-ordinate in the output image and performs the inverse of the rotation transformation to find the corresponding input image co-ordinate. In short, the transform is arriving at (*x*, *y*) having been given (*x'*, *y'*) and so transforms using –*angle* instead of *angle*. This approach is necessary to avoid missing out pixels in the output image, which would otherwise occur.

Of course the results of the matrix operation (to arrive at input image co-ordinates) are still non-integer in nature – which is an issue dealt shortly. Figure 3 provides a convenient synopsis: at **a**, an output co-ordinate is input to the inverse rotation equation and transformed to an input image co-ordinate at **b**. The pixel at the input co-ordinate position is then copied to the output co-ordinate position (at **c**).

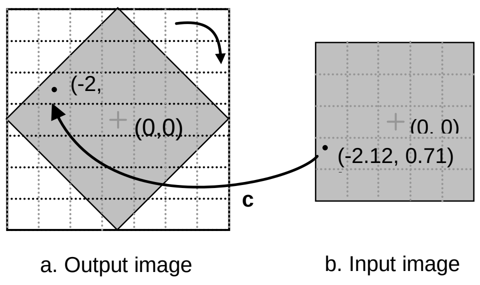

Figure 3. inverse trigonometric transformation

2.1 Pixel sampling technique

As previously stated, the result of the inverse transformation from output image to input image co-ordinates results in non-integer co-ordinate values. Given that pixels occur only on integer boundaries, a method of resolving decimal co-ordinates to a particular pixel is required. The simplest method is to round the co-ordinates to the nearest integer: using nearest neighbour method. However, this method results in stepped and pixelated output (aliasing). A method, which will reproduce the image more faithfully, is one which understands a non-integer co-ordinate as being partway between the closest integer values above and below it. Therefore the algorithm should return a pixel colour which is a weighted average of the colours of the two nearest pixels. Conceptually, this can be thought of as having an averaging grid of one pixel square centred over the co-ordinate in question, and for the weighted average to be of the pixels which fall under it.

The illustration in figure 4 shows the improvement in clarity achieved by making use of anti-aliasing.

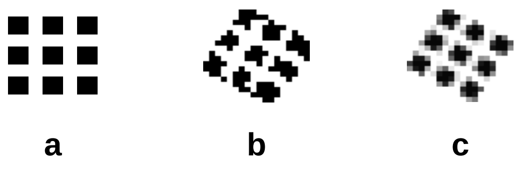

Figure 4. original image, nearest-neighbour rotation and anti-aliased rotation

Figure 5 illustrates how this is achieved with a pixel on integer co-ordinate boundaries (grid #1) and

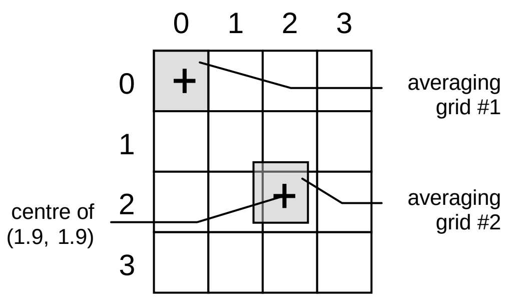

Figure 5. two averaging grids applied to an image

with one that is not (grid #2).

As the diagram shows, the grid falls on just one pixel with the first example, but on four for the second. The average for the latter would be substantially weighted on pixel (2, 2) but would include relatively small amounts of (1, 1), (2, 1) and (1, 2).

This technique is called *anti-aliasing*, because it removes the aliased pixels introduced by the nearest neighbour method. The result is a smoother and altogether better quality rotated image.

2.2 Image transparency

Transparency is a property of a pixel, and describes how transparent that pixel is– i.e. how much background shows through if the image is drawn over it. At its most basic, transparency information is binary– so pixels are either transparent or opaque. A more complex transparency mechanism records one of a number of intermediate stages of transparency (from opaque through translucent to entirely transparent) and is, in fact, the system used here. This is fairly simple to process, as it can be calculated using the previously described pixel-sampling technique in exactly the same way as colour information.

It is important that the algorithm routine can handle transparency, even if input images are guaranteed to be entirely opaque. As shown in figure 6, the rotating of a rectangular non-transparent image introduces blank areas between the rotated shape and its rectangular bounds.

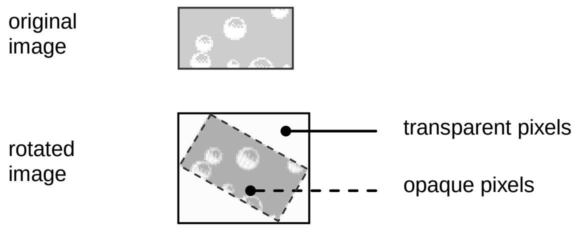

Figure 6. transparent pixels must be introduced even when rotating an opaque image

3. Parallel Implementation of The Algorithm

The phrase 'parallel computer' is often used to cover a multitude of computing machines, from single-purpose hardware devices which operate on a few data items simultaneously, to mainframe machines containing a large number of independent processors. Parallel computers like these represent the cutting edge of technology but are prohibitively expensive, and so users requiring fast computing machines have had to look elsewhere [11,12]. To this end, we have developed a model of systolic arrays design to handle geometric transformations.

3.1 Systolic Array Design for the Geometric Transformations

The design consists of a double pipeline systolic array, pipe one accommodates the x co-ordinate of the image pixel, and pipe two accommodates the y co-ordinate of the image pixel[13,14]. This design can be implemented in a straightforward manner as shown in figure 7. The first cell in the array, delays then makes a duplicate of the input data, and pumps them both into the array through both pipes.

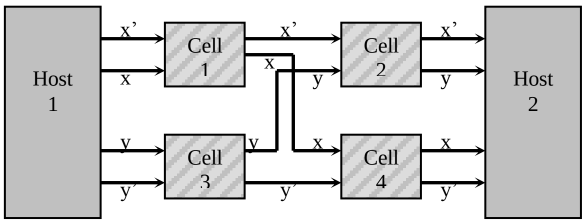

Figure 7. Double pipeline systolic array for the geometric transformations.

Each of the two pipes consists of 2 bifunctional cells and each cell contains a kernel element value (cosine or sine of the rotating angle) . In each cell (bifunctional cell) a signal element is held by a register and the cell contains two subtasks, multiplication of input by a weight w and addition of the result to the new co-ordinates of the image pixel (x',y') where the multiplier may be pipelined to an arbitrary degree, The cell design is shown in figure 8. Each cell produces its partial result one cycle earlier than the cell to its right[15]. The skew can be accomplished by replacing the register in each cell , which transforms the signal stream, with a multistage shift register. The number of stages of the shift register of the addition in each cell is 2.

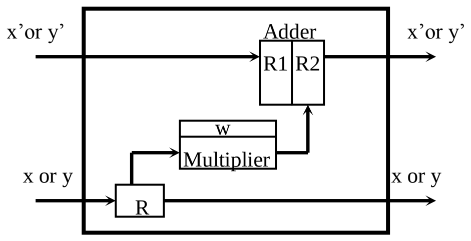

Figure 8. Cell design with add shift register (Lk).

As shown in figure 7, input data are pumped into the two pipes by the first host, in regular clock beats, through different channels. The data and the results are pumped through both pipes, to their corresponding cells, the input data are swithed between the two pipes, the x co-ordinate data is sent from the first pipe to the second pipe, while the y co-ordinate is sent from the second pipe to the first pipe, the partial results are pumped through both pipes, the final results for each co-ordinates of the image pixel and the original data are collected from both pipes by the second host. The algorithm is repeated for every input data for each cell concurrently[16].

Only cells on the array boundaries are permitted to communicate with the nearest host and each of the cells communicates with the left or right neighbour cells only. It is assumed that there is a global clock synchronizing the computation of all components in the system, having a time cycle long enough to accommodate the most complex function performed by a cell, plus the data transfer. In each step, all cells simultaneously perform their I/O and execute their operations.

The following is the parallel algorithm computed for each input data in all the cells:

<u>Cell 1</u>

read input value x from host 1
read x' from the host

calculate partial result of x'

send x to Cell 4

<u>Cell 3</u>

read input value x from host 2
read y' from the host

calculate partial result of y'

send y to Cell 2

```
send x' to Cell 2

Cell 2
read y from Cell 3
read x' from Cell 1
calculate partial result of x'
send y and x' to Host 2
```

```
send y' to Cell 4

Cell 4
read x from Cell 1
read y' from Cell 3
calculate partial result of y'
send x and y' to Host 2
```

3.2 Distributed Systems Implemntation

An increasingly popular alternative to employing specialised computing hardware is the use of networked computers (distributed system), where each computer in the network takes part of the task in hand – such as image processing –using message passing, thus speeding the time required to complete it. Often, users in research and in business already have available such networked machines, so this approach represents very little additional cost.

Distributed systems have a number of things in common with specialised parallel computers; the first of which is the need for processors to 'talk' to each other. In the simplest of parallel processing models, there is a master (or parent) task, which commands the other tasks (slaves or children); this is known as the master/slave model. Communication will occur between master and slave, usually to pass data to be processed or to return data that has been processed; also, communication will occur between slaves if they need to share information to complete their task [10,12,17].

A second issue common to both is that whatever communication mechanism is used to send messages between processors, doing so adds a time penalty (communication time) to the execution of the program. This also leads to a decrease in system efficiency, as the sending processor has to wait for an answer and the receiving processor has to break off to deal with the request.

3.3 Parallel Virtual Machine

The software written for this work makes use of the Parallel Virtual Machine C library that allows for co-operation and synchronisation to occur between tasks through the use of explicit message-passing. It provides calls to spawn and kill tasks within the distributed system (the *virtual machine*), and to facilitate the transmission and reception of messages between tasks.

Under PVM, a parent task running on a single machine is used to add subsequent child tasks to computers within the network to the virtual machine. New child tasks can be started on a specified computer, or as is more often the case, started on a computer chosen by PVM to maintain optimum load balance across the network. PVM then allows for any of these processes to send a message to any other processes.

3.4 Task communication strategy

It is necessary to determine a communication structure between tasks which will facilitate processes receiving data that they need, in a way that is reasonably efficient. A key issue here is the way in which the problem is shared between nodes (the granularity of the problem).

One way is to divide work at a very low level (usually statement level), and assign suboperations within this level to processors (task parallelisation). Applied to the image transformation problem, this would involve breaking the transformation for each pixel into a number of simple maths operations, each of which being computed by its own processor [18].

An alternative approach that is clearly more suited to distributed computing is data parallelisation, in which processors each perform the same operation (with individual threads of control) on substantial sequences of data. Each child is therefore sent a block of pixels, which are transformed and returned to the parent, as shown in figure 8. Inter-process communication is unnecessary here, since each section can be done independently of others.

3.4.1 Sharing image data between tasks

The images should be divided between tasks. This is sensibly done by cutting the graphic up into something of a jigsaw puzzle, and for each piece to be operated on before re-assembly. This has the advantage that pixels can be supplied as rows of sequential pixels, both before and after processing, without the need to specify the co-ordinates of each one [19].

Since the bulk of transmitted information will be image data, it needs to be ensured that these are transmitted efficiently from process to process. A contributing factor to this efficiency is the shape of the subimages into which images are separated. The shape needs to be chosen in order to minimise the amount of background data which is included in the output. This maximises the amount of useful data– the actual original image shape–, which is returned in each subimage.

It has been found that the square is the most space-efficient shape, therefore the strategy used should reflect this.

It is not possible to always decompose images entirely into square pieces, since often their shape will mean some pieces have to be rectangular as shown in figure 9.

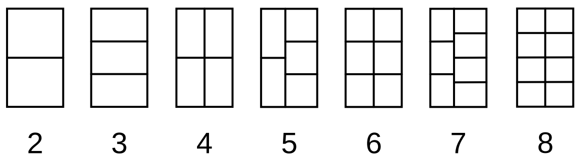

Figure 9: image cut into approximately square pieces for wherever possible

4. Performance of the Design on a PVM Network

In order to evaluate the effectiveness of the distributed system, test software was written and run on a network of Linux machines. Each machine was running an identical Pentium IV PC processor, and was linked together via central network servers through standard 100Mbit Ethernet. The parent executable was run on one node, and child executables on all the other nodes, each of the latter simply looping until instructed otherwise by the parent. Figure10 presents an image divided into four parts, each child is therefore sent a block of pixels, which are transformed and returned to the parent to be recombined in a single image, which represents a rotation of the original image.

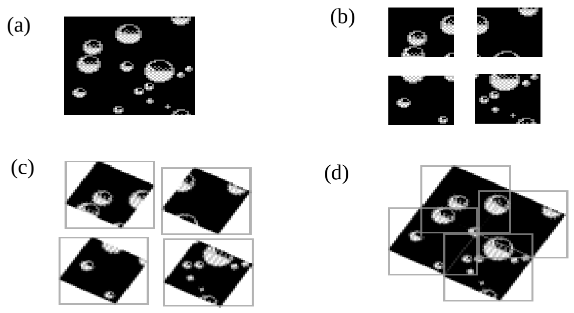

Figure10: The stages of rotating an image piece by piece

5. Conclusion

This study included the implementation of a systolic array design for geometric transformation algorithms on a distributed computing system using Parallel Virtual Machine (PVM) model. Medical images of tissues are split into a number of parts and each part is sent to a separate computing node. Several types of cells are used in this work, which have improved the hardware utilisation and throughput. The number of cells in the systolic arrays are not related to the size of the problem, which reduces the design area to a minimum compared to other schemes.

Each node of the system performs a rotation on its partial image before returning it to the master node to be recombined in a single image. The results obtained were in agreement with the aims of this work. These designs seem capable of solving general problems involving medical image applications by implementing the design on a distributed system.

6. References

[1] Umbaugh, S. E. (2010) Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools, Second Edition, CRC Press.

[2] Umbaugh, S.E. (1998). Computer Vision and Image Processing: A practical approach using CVIPtools, International Edition, Prentice Hall.

[3] Gonzalez, R.C. Woods, R.E. (2008), "Digital Image Processing", Third Edition, Prentice-Hall.

[4] Sharp, John A. (1987), "An Introduction to Distributed and Parallel Processing", Blackwell Scientific Publications.

[5] Kung, H.T., Ruane, L.M. Yen, D.W. (1983), Two-Level Pipelined Systolic Array for Multidimensional Convolution, Image and Vision Computing, Vol.1, No.1.

[6] Burian, A., Rusu, A. Kuosmanen, P. (1998), "Efficient realization of the M-D nonrecursive filters: from sequential implementation to mapping on systolic array processors". IEEE International Conference on Electrics, Circuits and Systems.

[7] Megson, G.M. (1992), "An Introduction to Systolic Algorithm Design", Clarendon Press, Oxford, UK.

[8] Hwang, K. Briggs, F. A. (1998), "Computer Architecture and Parallel Processing", McGraw-Hill Int. Editions.

[9] Chandrasekhar, C. Narayana Reddy, S. (2012), "Design of 3D-DWT Architecture using Systolic Array Based High Speed 1D –DWT", Conference on Computer Science & Computational Mathematics (CCSCM 2012), pp.88-96.

[10] Beguelin, A. Dongarra, J. Geist, A. (1994), "PVM: Parallel Virtual Machine", MIT Press.

[11] Hockney, R. W. Jesshope, C. R. (1988), "Parallel Computers 2, Architecture, Programming, and Algorithms", Adam Hilger.

[12] Houzet, D. (1996), "Real Time Image Processing with a MIMD Computer, Real Time Imaging 2", Academic Press.

[13] Lee, C., Wang, Y. F., Yang, T, (1997). "Global Optimisation for Mapping Parallel Image Processing Tasks on Distributed Memory Machines", Journal of Parallel and Distributed Computing 45, pp. 29-45.

[14] LeRiguer, E., Woods, R., Ridge, D., McCanny, J. A. (1998), "Programmable Image Processing Chip", IEEE International Symposium on Circuits and Systems.

[15] Evans, D. J. Amin, S. A. (1995), "Systolic Algorithms for Digital Image Filtering" , Parallel Computing, 21.

[16] Amin, S.A. Naguib, R.N.G. (2000), "Parallel Implementation for Low-Level Medical Image Processing Algorithms", Chicago 2000 World Congress on Medical Physics and Biomedical Enginerring, IEEE, USA.

[17] Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, Sun-deram, V. (1993), "PVM3 User's Guide and Reference Manual". Oak Ridge National Laboratory, Oak Ridge, Tennesse.

[18] Amin, S.A., Naguib, R.N.G. (2003), "A Parallel Implementation of a Genetic Algorithm for Colonic Tissue Image Classifications", 4th Annual IEEE Conf on Information Technology Applications in Biomedicine, UK.

[19] Fukuda, M, Bic, L. F., Dillencourt, M. B., Cahill, J. M. (1999), "Messages versus Messengers in Distributed Programming", Journal of Parallel and Distributed Computing 57, pp. 188-211.