PVM implementation for low-level image processing systolic array designs *

S.A. Amin, D.J. Evans and L. Lefèvre

Computer Department, MidKent College - Horsted Centre
Chatham - Kent - United Kingdom
S.A.Amin@lut.ac.uk

Parallel Algorithms Research Center (PARC)
Lougborough University of Technology - United Kingdom
D.J.Evans@lut.ac.uk

Laboratoire de l'Informatique du Parallélisme - E.N.S. Lyon
69364 LYON Cedex 07 France
Laurent.Lefevre@lip.ens-lyon.fr

Abstract

This paper describes PVM implementation of systolic array designs of parallel algorithms for low-level digital image processing, in particular we consider the mean, weighted mean and laplacian filters. The PVM implementation of the design, is discussed with improvements of the original systolic design (macro and multi-pipeline). Comments and conclusions related to the implementation of the systolic array on PVM are provided in the performance section.

1 Introduction

The purpose of this paper is to identify a set of designs suitable for implementing low level image processing algorithms on distributed systems using Parallel Virtual Machine (PVM) model. We consider techniques for filtering digital images. Image smoothing has the purpose of removing noise and making the pixels' grey levels uniform. Most of the image processing algorithms need massive amounts of banded matrix operations. However, these algorithms contain explicit parallelism which can be efficiently exploited by processor arrays. All sections of the image have to be processed in exactly the same way, regardless of the position of the image section within the image, or the value of the pixel data. Low level functions involve matrix vector operations, which are repeated at very high speed. Typically, images must be processed in real time, at 25 images per second. Therefore with image sizes of 128x128, 256x256, and 512x512 or greater, there is a large amount of data to be processed in a highly repetitive process. Many low-level image processing operations, termed local operators, require access to the four or eight neighbouring intensity values of a pixel, when computing the new value for the pixel. Each member element in the image is replaced by some function of itself and the neighbouring elements within a window centred on that element. There are different architectures for carrying out these varied operations.

*This work has been supported by The Human Capital & Mobility EC Project NATHAN

A number of systolic designs for 2D convolution and digital filters have been implemented [KS92],[Meg92],[WV81]. In this paper, a class of local smoothing filter algorithms are covered, mean, weighted mean and laplacian filters. For this purpose, we have developed several models of systolic arrays. Different kinds of cells are used. Various modifications of the systolic design are analysed, to handle a set of digital image filters.

The report starts with a brief definition of the algorithms. Another section presents a systolic array design for these filters.This paper also presents the implementation of a variety of digital image filter algorithms on the distributed computing using Parallel Virtual Machine (PVM) model [GBD$^+$93]. One of the aims was to design and build a programming workbench for developing image processing operations for low-level vision. The motivation for the work is to develop a methodology for the implementation of an image processing library on the distributed computing

2 Operator Algorithms.

We have developped various kinds of low-level image processing methods (like convolution, gradient filter, Inverse Gradient filter...). As example of our work, we present our systolic design of Laplacian Operator algorithms. The Laplacian operator is computed by convolving a mask with the image. The Laplacian low pass filter is a typical smoothing filter, a Laplacian mask is passed over the entire image, and the convolution operation is performed on each pixel. Each pixel is replaced by the sum of the products of the mask weighting and the appropriate neighbouring pixel values. Different choices are available when using this mask in two dimensions. The Laplacian filter using 3 by 3 convolution mask may be used. This utilises a mask or weighting matrix defined on two standard masks as shown in figure 1.

	0	-1	0		-1	-1	-1
Laplacian 2 :	-1	4	-1	Laplacian 3 :	-1	8	-1
	0	-1	0		-1	-1	-1

Figure 1: Laplacian masks

The value in the weighting matrix allowed a simpler and faster version of the algorithm than was obtained using the general convolution case. The algorithm given here 'smooths' a grey level input image and has I as an input image and O as an output image; both I and O contain M by M pixels, with P = M2. Each point of I for generating y, is as follows (for an image n by n).

for $i, j = 1, 2, \ldots.., n$

$$y_{ij} = w_1 x_{i-1,j} + w_2 x_{i,j-1} + w_3 x_{i,j} + w_4 x_{i,j+1} + w_5 x_{i+1,j}$$

By applying the weighting mask shown in figure 1 to this equation, then,

$$y_{i,j} = \Sigma f(1,5)[x_{i-1,j} + x_{i,j-1} + x_{i,j} + x_{i,j+1} + x_{i+1,j}]$$

Also if we employ the square window shown in figure 1 the output pixel is obtained by using the relation,

$$y_{ij} = w_1 x_{i-1,j-1} + w_2 x_{i-1,j} + w_3 x_{i-1,j+1} + w_4 x_{i,j-1} + w_5 x_{i,j}$$

$$w_6 x_{i,j+1} + w_7 x_{i+1,j-1} + w_8 x_{i+1,j} + w_9 x_{i+1,j+1}$$

and,

$$y_{i,j} = \Sigma f(1,9)[x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + x_{i,j}$$
$$+x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1}]$$

This algorithm approach for the weighted mean filter is similar to the mean filter algorithm described in the previous section . The difference in this case, a weighted mean filter is often used in which the weight for a pixel is related to its distance from the centre point. For a 3x3 window, the filter weights are shown in figure 2. The neighbours that lie on the same side of the point (i,j) are weighted more heavily than the others.

		1/6			1/16	1/8	1/16
Weighted mean 1 :	1/6	1/3	1/6	Weighted mean 2 :	1/8	1/4	1/8
		1/6			1/16	1/8	1/16

Figure 2: Weighted mean masks

3 Systolic Array Design

The systolic array design to accommodate the previous filters, consists of $k2$ cells, where $k2$ is the size of the window. Each cell contains a multistage shift register with three

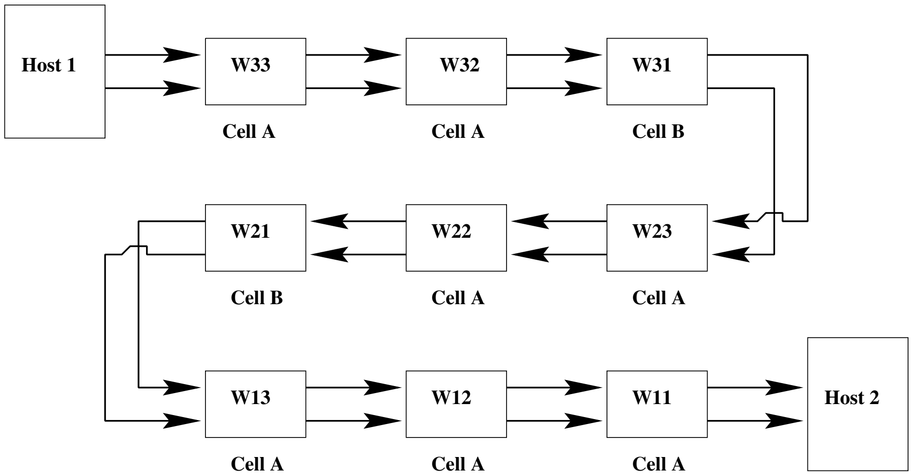

Figure 3: Systolic Array for Laplacian

different numbers of shift registers in three different multifunctional cells. All processes inside the cell run in parallel for input data, so that communication between processes is overlapped with computation. Each cell produces its partial result one cycle earlier than the cell to its right . The skew can be accomplished by replacing the register in each cell , which transforms the signal stream, with a multistage shift register. As shown in figure 3, the image data is pumped into the array by the first host, and then send them down to the multifunctional cells. Each multifunctional cell requires 2 input and 2 output channels for communications with the left and right neighbour cells. The data and results are pumped through the array and the accumulated partial products are collected by the final cell , then it sends the final results to the second host. The algorithm is repeated for every input data for each cell concurrently. Only cells on the array boundaries are permitted to communicate with the host and each of the cells communicates with the left and right neighbour cells only. Each cell forms a second level of pipelining, the first level being the global pipelining between array cells, while this additional level of pipelining of computations within a cell can increase the system throughput. In each step, all cells simultaneously perform their I/O and execute their operations.

4 PVM implementation for the systolic design

The systolic system described previously required further modification in order for the array to be implemented on distributed computing using PVM. The master-slave model in which a separate 'control' program termed the master is responsible for process spawning, initialisation, collection and display of results. The slave programs perform the actual computation (cell) involved; they either are allocated their workloads by the master or perform the allocations themselves. The first and the final slaves are connected to the master (host), the first slave receives all the input data from the master and pump it to the network, the final slave receives the results from the neighbouring slave and sends it down to the master. For simplicity each slave is responsible for a cell of the systolic.

5 Experimentations

For trying, our new applications, we, first, execute them on a network of stations; but due to poor efficiency of Ethernet communications, the results were not good. The next experimentations have been done on a parallel machine (Cray T3d) which implements PVM upon a tore topology [LBR94b, LBR94a] which proposes an architectue with fast links between processors (with 1μs of latency and a bandwidth of 300 Mbytes/sec). We have also improved the performances of systolic implementation by mixing macro-pipeline methods and multi-pipeline implementation.

5.1 Grouping messages : Macro-pipeline

The amount of computing is so small in each cell that the application spends most of the execution time to communicate data and to wait pixels for his neighbours. So we have decided to increase the application granularity by using macro-pipeline methods. Instead of sending a message for each pixel, each cell constructs and sends packets of pixels.

	16x16	32x32	64x64	128x128
1	0.066	0.62	12.1	
2	0.035	0.19	2.5	52.3
10	0.016	0.06	0.23	0.94
100	0.048	0.1	0.25	1.01

Table 1: Laplacian filter execution time depending on packet size for various picture size (in seconds)

In table 1, we can see that we have greatly improve the performances of our systolic Laplacian filter by using macro-pipeline methods. For example, on a picture of 128x128 pixels, we can apply our Laplacian filter in 0.94 second with packet of 10 pixels instead of spending 53 seconds with small packets of 2 pixels.

The figure 4 shows more precisely the execution time of the filter, depending on packet size, with a given image size. We can see that the optimal paket size for applying a Laplacian filter on a 32x32 picture is closed to 22 pixels.

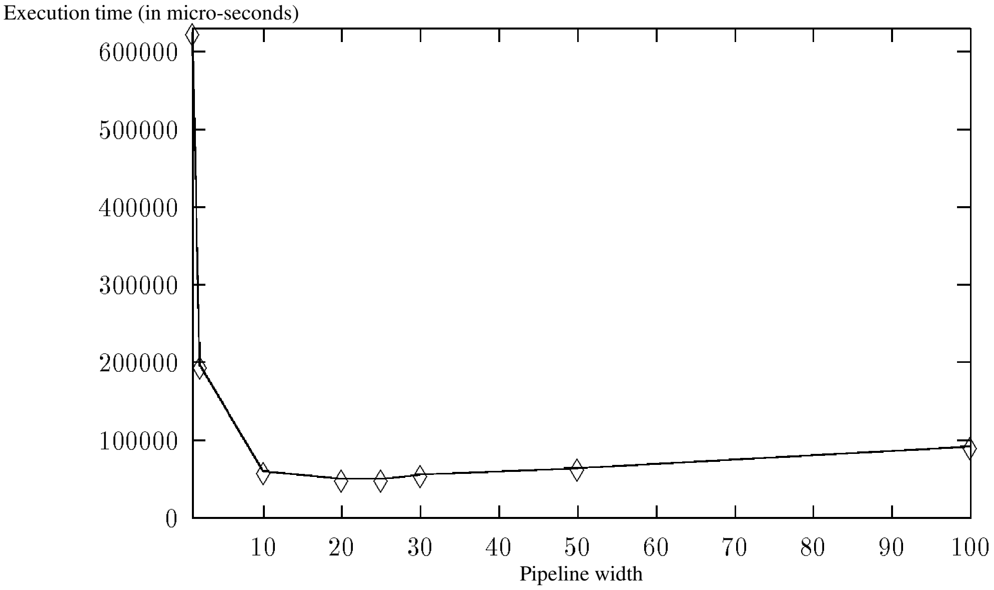

Figure 4: Laplacian filter on a small 32x32 picture, depending on packet size (pipeline width)

5.2 Grouping cells : Multi-pipeline

Our main purpose is to developp a complete image processing library on a distributed architecture. Instead of using the same pipeline to apply all the filters, we have implemented a multi-pipeline application. By doing different low-level image processing in parallel on different pipelines, we compute in parallel different pictures. The results presented in the final paper show that the execution time for computing one picture is very closed of the one for obtaining different pictures with multi-pipeline version. The small differences result of initialising data and multi-results collection on master slave. Moreover we also demonstrate that merging various filters on the same pipeline of cells can improve the performances of the filtering.

6 Conclusion and future works

The PVM implementation of pipelined filters have shown interesting results on Cray T3D. But to improve the performances, we have to developp macro-pipeline methods to reduce the communication time of the application. Moreover, the use of multi-pipeline low-level image processing improves greatly the speedup of the application by outputing different pictures at the same time. We are currently finishing the implementation of the whole image processing library with various kinds of operators (convolution, Gamma filter, Inverse Gradient filter...). To compare our systolic view of the problem filtering with other techniques, we are developping also different approachs of the problem by splitting the picture and sending the parts to the processors (data-parallelism approach). We also study a distributed-shared memory implementation to show that these filtering application can be implemented in an easy way on that kind of system.

References

[GBD+93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam.

PVM3 User's Guide and Reference Manual. Oak Ridge National Laboratory, Oak Ridge, Tennesse, May 1993.

[KS92] H.T. Kung and S.W Song. A systolic 2d convolution chip. *Multiprocessors and Image Processing Algorithms and Programs*, 1992.

[LBR94a] Vincent Le Barazer and Gunter Roth. Applications sur cray t3d. : Tome iv : Documentation shmem et pvm 3.3. Technical report, Cray Research France, 1994.

[LBR94b] Vincent Le Barazer and Gunter Roth. Applications sur cray t3d. tome 1 : Généralités mpp et t3d. Technical report, Cray Research France, 1994.

[Meg92] G.M. Megson. An introduction to systolic algorithm design. *Clarendon Press, Oxford, UK*, 1992.

[WV81] D.C.C. Wang and A.H. Vagnucci. Gradient inverse weighted smoothing scheme and the evaluation of its performance. *Computer Vision: Graphics and Image Processing*, 15:167–181, 1981.