Parallel Algorithms For Image Convolution †

Shuling Yu, Mark Clement, Quinn Snell and Bryan Morse
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
(801) 378-7608
FAX (801) 378-7775
{shulingy,clement,snell,morse,}@cs.byu.edu

Abstract

In this paper, we compare the Redundant Boundary Computation (RBC) algorithm for convolution with traditional parallel methods. This algorithm dramatically reduces the communication cost for the the computation in certain environments. We theoretically and experimentally study the conventional parallel algorithm and the RBC algorithm. First, we discuss the performance of these parallel algorithms, focusing on the execution time, efficiency, speedup and scalability. Then we compare their performance to each other. Finally, we present experimental results and analysis. Our objective is to develop methods to select the correct algorithm depending on machine and problem parameters.

Keywords: analysis of algorithms, parallel algorithms, image convolution, image processing, parallel performance.

1 Introduction

Increasing demands for computing power have led to rapid improvements in parallel processing hardware [5, 14]. Several techniques have been used to increase the speed of parallel computations at the algorithmic and architectural level. Choosing the correct parallel algorithm can have a significant effect on the performance of certain parallel problems. Various applications have forced parallel programmers to develop new algorithms in order to meet performance requirements [2, 12].

The number of computations involved in image processing and computer vision increases as large, high resolution images become more common. The widespread availability of high speed networks makes it possible for clusters of workstations to perform well on image processing applications. Because of the communication and I/O overhead in parallel processing, parallel algorithms don't always obtain better performance. The performance is strongly dependent on the size and complexity of the problem, the features of the network, the performance of the machines and the algorithm applied.

Many algorithms in image processing have been found to have efficient parallel implementations. Convolution is a basic operation in image processing and computer vision. It has a multitude of applications including image filtering and enhancement, image restoration, feature detection and template matching. The result of image convolution computed at a pixel point is dependent only on the value of neighboring points. Because of this, the convolution operation can also be efficiently mapped to parallel computers.

The parallel implementation of the conventional convolution algorithm [9, 11] divides the image into small subimages (blocks or stripes), then distributes the subimages to each processor. Each processor obtains an up-to-date copy of its boundary region from neighboring processors and performs the convolution computation on its part of the image data during each iteration. Finally, processors send their results to the master machine. The crucial part of the conventional algorithm is the communication and synchronization of the boundary region. It not only introduces communication overhead, but also may increase the idle time between iterations if the computation and communication speed of each processor is not constant.

The RBC algorithm avoids boundary region communication and synchronization and its implementation is easier than that of the conventional algorithm.

† This research was supported in part by Sprint Corporation.

It divides the image into small subimages which overlap at the boundaries. Each processor can perform its computation independently. This algorithm reduces the communication cost and idle time at each iteration. With multiple iterations, boundary values are computed redundantly by multiple processors.

In order to achieve the highest performance, the correct algorithm must be selected for a given convolution problem. In some cases, the RBC algorithm will achieve better results, but in other cases the conventional algorithm performs better. This research characterizes the two algorithms and lays the groundwork for a library which would automatically select the best algorithm and number of processors for a given problem.

2 Convolution Model

The convolution topic arises in a variety of ways with images [1, 10, 15]. Digital filtering, image smoothing, image sharpening and some other image operations all may involve convolution. In this section, we outline the basics of the convolution problem and then analyze its computational requirements. Finally, we give an example using a convolution kernel in an image processing application.

2.1 Image Convolution

Spatial convolution in two dimensions is a neighborhood operation and is defined by

$$f[x,y] * g[x,y] = \sum_{j,k} f[j,k] g[x-j, y-k] \qquad (1)$$

It can be described as a weighted sum of neighboring samples.

Convolution is quite an expensive operation in terms of the amount of computation involved. Considering a square convolution kernel, the number of addition and multiplication operations increases with the the square of the size of the kernel. We assume the image is of size $N \times N$ and the kernel is $m \times m$. Each pixel of the input image requires m^2 additions, m^2 multiplications and a single division to compute a normalized convolution. The computation time is:

$$t_c(2m^2+1)N^2$$

where t_c is the average time for a computation operation.

If we perform the convolution with several kernels (assuming the number of kernels is K and all the kernels are of size $m \times m$) the computation time is

$$t_c(2m^2+1)N^2K \qquad (2)$$

Convolution can be computed directly on images in the spatial domain. Alternatively, it can be computed after the image has been transformed into the frequency domain. For larger kernels, there is considerable benefit to be gained by performing the parallel convolution in the Fourier domain [4]. For this reason, we limit the discussion in this paper to convolution with small kernels. The next section presents an example of an image processing application.

2.2 Convolution for Edge Detection

One of the goals of computer vision is to identify objects. Edge detection may be used to locate the boundaries of objects. An edge detector looks for regions of the image where the intensity is rapidly changing or in other words, points where the first derivative values are large. The place where the first derivative of the step is a maximum is exactly the place where the second derivative of the step has a zero crossing. Generating the Laplacian of an image corresponds to taking the derivative in two-dimensions. The Laplacian of a function f(x,y) is defined by:

$$\nabla^2 f = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

For a discrete image:

$$\begin{aligned}
\frac{\partial f}{\partial x} &\approx f(x+1,y) - f(x,y) \\
\frac{\partial f}{\partial y} &\approx f(x,y+1) - f(x,y) \\
\frac{\partial^2 f}{\partial x^2} &\approx (f(x+1,y) - f(x,y)) - (f(x,y) - \\
&\quad f(x-1,y)) \\
\frac{\partial^2 f}{\partial y^2} &\approx (f(x,y+1) - f(x,y)) - (f(x,y) - \\
&\quad f(x,y-1)) \\
\nabla^2 f &\approx f(x+1,y) + f(x,y+1) - 4f(x,y) + \\
&\quad f(x-1,y) + f(x,y-1)
\end{aligned}$$

We can carry out Laplacian zero crossing edge detection using convolution with the following kernel:

0	1	0
1	-4	1
0	1	0

3 Parallel Performance

Predicting parallel algorithm performance for a given problem is more complicated than predicting sequential algorithm performance. A parallel algorithm that solves a problem well on a particular architecture may perform poorly on another architecture. There are many factors that influence the performance. The relative importance of these factors will vary from algorithm to algorithm and from problem to problem. The performance model for a parallel architecture plays an important role in evaluating the scalability of different algorithms and choosing an appropriate algorithm.

There are many techniques that have been developed to characterize the performance of parallel algorithms [4, 8, 3]. We based our model on the homogeneous multicomputer parallel architecture model with irregular communication patterns [7]. We use the performance metrics developed by Foster [4], because they abstract away unimportant details, explain available observations and predict future circumstances. For each algorithm, we discuss execution time, efficiency and speedup. The execution time T_p is defined as the time that elapses from the start of the first processor executing on the problem to the end of the last processor completing execution. It includes computation time, communication time and idle time. We assume uniform message routing distribution [13] and neglect the idle time. T_p may be defined as

$$T_p = T^i_{comp} + T^i_{comm} \tag{3}$$

where T^i_{comp}, T^i_{comm} are the computation time and the communication time on a processor, respectively.

If an algorithm executes on p processors with time T_p and T_1 is its sequential execution time, the efficiency is defined by the ratio

$$E = \frac{T_1}{PT_p} \tag{4}$$

The speedup is defined by the ratio

$$S = \frac{T_1}{T_p} = PE \tag{5}$$

4 Conventional Parallel Convolution Algorithm

The conventional parallel algorithm for convolution decomposes the input image into subimages and distributes them among P tasks. Thus, for simplicity, we partition the input image into stripes in the horizontal direction. Each task is responsible for a subimage of size of $N \times N/P$ and performs the same computation on each pixel. Figure 1 shows the decomposition. The full input image is $N \times N$ and the kernel is $m \times m$. Because this algorithm does not replicate computations, from Equation 2 we get

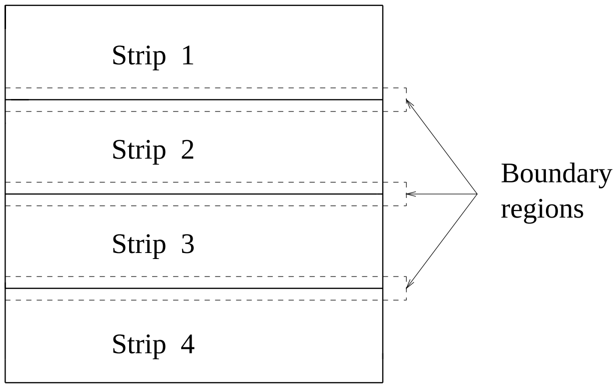

Figure 1: Conventional algorithm decomposition of the input image

$$T^i_{comp} = \frac{1}{P}T_1 = \frac{t_c(2m^2+1)N^2K}{P}$$

Each processor needs to exchange two messages with its neighboring processors. The size of each message is

$$N\frac{m-1}{2}$$

Since many processors may need to send data over the same wire at the same time, the impact of competition for bandwidth may be severe. The per-processor communication cost is

$$T^i_{comm} = 2K(t_s + t_b SN\frac{m-1}{2}) \tag{6}$$

The variable t_s represents the message startup time which is the time required to initiate the communication. t_b represents the transfer time per byte which is determined by the bandwidth of two processors between source and destination. S represents the number of messages needing to be sent concurrently over the same wire to indicate the effect of competition for bandwidth. S is determined by the network topology, communication pattern and algorithm applied.

We combine Equation 6 with Equation 3 and obtain the execution time

$$T_p = \frac{T_{comp} + T_{comm}}{p} \tag{7}$$

$$= \quad t_c \frac{(2m^2+1)N^2K}{P} + 2t_sK + t_bKSN(m-1) \tag{8}$$

From Equation 4 and Equation 7, we obtain the efficiency

$$E = \frac{1}{1 + \frac{2t_s + t_bSN(m-1)}{t_c(2m^2+1)N^2}P} \tag{9}$$

From Equation 5 and Equation 9, we get the speedup

$$S = \frac{P}{1 + \frac{2t_s + t_bSN(m-1)}{t_c(2m^2+1)N^2}P} \tag{10}$$

These equations show that the conventional algorithm will perform poorly for large numbers of processors with slower networks.

5 Redundant Boundary Computation Algorithm

The communication overhead has a significant effect on the speedup and efficiency achieved. As seen previously, in the conventional parallel convolution algorithm, the communication cost increases with processor count. If a communication network cannot provide adequate performance, processors will frequently be forced to wait for the up-to-date boundary region messages to arrive. This causes a bottleneck. The Redundant Boundary Computation (RBC) algorithm decreases the communication cost by trading communications for redundant computations. We decompose the input image into stripes in the horizontal direction, but stripe boundaries overlap with neighboring stripes as shown in Figure 2. After image partitioning, the overlapped stripes are distributed to P processors. All the data required to carry out the convolution of one stripe are in each processor, so there is no need for communications.

The boundary data is negligible when compared to the distributed data for a small kernel with a limited number of iterations. Since the data distribution time is present in both algorithms, this overhead is neglected in the analysis. Redundant computations are performed at each processor. If a m by m kernel is used and the convolution is performed three times, the configuration shown in Figure 3 results. At each processor, the first iteration is performed over rows from A to A'; the second iteration is performed from B to B'; the third iteration is performed from C to C'. The total number of computations for the redundant algorithm includes one pass over the data from A to B and from B' to A', two iterations from B to C and from C' to B'. Points between C and C' are computed three times. Each of the groups of rows of size $(m-1)/2$ is referred to as a mini-strip. The result of the redundant algorithm is equivalent to performing the convolution three times with a data size of $2N\frac{m-1}{2}$. The additional redundant computations are really just overhead and will slow down the peak performance of the algorithm. For some configurations, however, trading these computations for expensive communications will be beneficial.

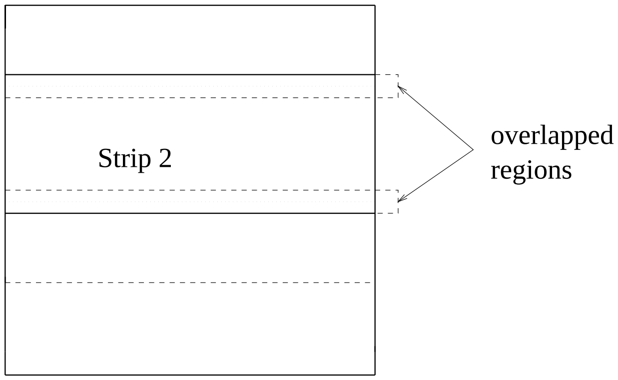

Figure 2: Redundant computation algorithm decomposition of input image

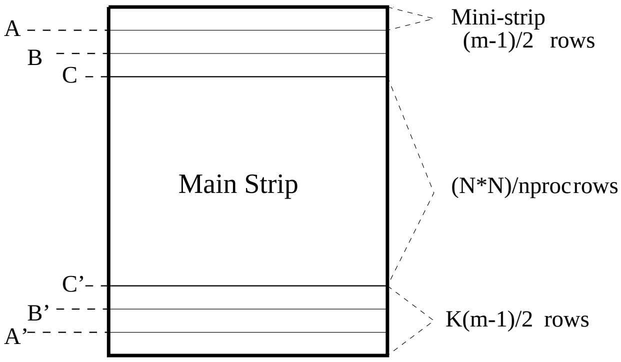

Figure 3: One stripe of redundant computation algorithm decomposition

With K as the number of iterations to be performed, and with i as the iteration number $(1...K)$ the redundant computation (overhead) is equivalent to carrying out K convolutions on data of size $(K-i)N\frac{m-1}{2}$.

The total redundant computation is

$$T_{overhead} = \sum_{i=1}^{k} t_c(2m^2+1)(K-1)N\frac{m-1}{2}K \quad (11)$$

The sequential execution time is :

$$T_1 = t_c(2m^2+1)N^2K$$

From Equation 3 and Equation 11, we obtain the execution time Note that there is no communication time.

$$\begin{aligned} T_p &= T^i_{comp} = T_1 + T_{overhead} & (12)\\ &= t_c\frac{(2m^2+1)N^2K}{P} + & (13)\\ & \quad t_c(2m^2+1)(K-1)N\frac{m-1}{2}K \end{aligned}$$

From Equation 4 and Equation 12, we obtain the efficiency

$$E = \frac{1}{1+P\frac{(K-1)(m-1)}{2N}} \quad (14)$$

From Equation 5 and Equation 14, we obtain the speedup

$$S = \frac{P}{1+P\frac{(K-1)(m-1)}{2N}} \quad (15)$$

The RBC convolution algorithm is not effected by network speed, but will result in poorer performance with large kernels and a large number of iterations. Figure 4 shows the speedup for a kernel size of 9×9 with 8 iterations. As one can see, the speedup curves cross. The crossing point is important because it shows where it would be advantageous to change from one algorithm to another. The crossing point is determined by the problem size, network features and machine speed. In either the mesh/hypercube or bus-based network, the conventional algorithm should be selected when there are a smaller number of processors than those present at the crossover point. If more processors are available, the RBC algorithm should be selected.

Comparing the two algorithms, the RBC algorithm is attractive for several reasons:

- Because there is no need to consider the communication and synchronization among the tasks, the RBC algorithm is suitable for shared media networks where contention can occur.

- There is no idle time between iterations since synchronization is not necessary.

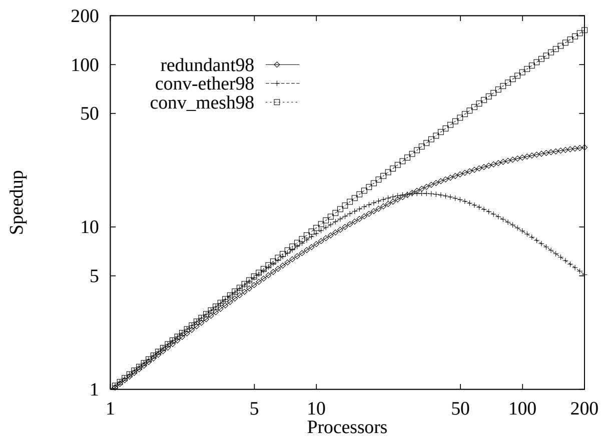

Figure 4: Speedup comparison for a large kernel and large number of iterations when input image size is 1024×1024,$t_c = 0.1\mu sec$,$t_s = 1500\mu sec$, $t_b = 1\mu sec$, the kernel size is 9×9 and 8 iterations.

- The speedup is not effected by the communication and machine features. We can get good speedup when the product of kernel size and number of iterations is not large.

The conventional algorithm is beneficial if the network topology is mesh or hypercube. Competition for bandwidth is small on a mesh or a hypercube because the communication structure of convolution can be embedded in these networks using nearest-neighbor connections.

6 Experimental Studies

The programs were coded with the PVM libraries[6] using the master-worker paradigm. The master sends input data to a worker and the worker sends back output data to the master after finishing its computations. The programs were implemented on a network of HP workstations. Each worker program ran on a separate workstation and one extra workstation ran the master program. We performed the test with an image size of (1024×1024), three kernel sizes $(3 \times 3, 5 \times 5 and 9 \times 9)$ and 2, 4 and 6 iterations. We divided the input image into horizontal stripes, where the number of stripes was the same as the number of tasks sent to workers and varied from 1 to 16.

We measured only the time of the convolution, not the time needed for initialization I/O since these times are the same for either algorithm. This allows us to concentrate on the areas of the algorithms that were

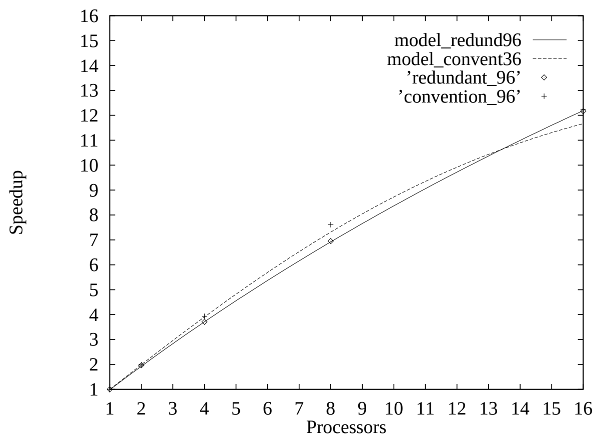

Figure 5: Speedup of the two algorithms with kernel size 9×9, and 6 iterations.

examined in our analytical models.

Figure 5 provides a sample of the predicted and experimental results. The RBC algorithm has better scalability in shared media networks when the kernel size is small or when the kernel size is large and there are a small number of iterations. But the speedup becomes worse when the product of the number of iterations and kernel size increases. For a specific problem, the maximum speedup value obtained for the RBC algorithm is $\frac{2N}{(K-1)(m-1)}$. The speedup is worse than the conventional algorithm if the number of processors is small and the product of number of iterations and kernel size is large. The RBC algorithm is beneficial if the cost of redundant computations is less than the communication costs for the conventional algorithm.

7 Conclusions

We have discussed two parallel algorithms for image convolution with small kernels. Selecting an algorithm suitable for all application parameters is difficult. The careful comparison of the conventional algorithm and the RBC algorithm shows that the speedup of the RBC algorithm is independent of network features and machine speed. The RBC algorithm is most promising for situations where competition for bandwidth is severe. Although the RBC algorithm has redundant computation overhead, it has no communication cost among tasks and the speedup is independent of machine and network features. The performance of the RBC algorithm is only dependent on the kernel size and the number of iterations for a given input image. The RBC algorithm obtains better parallel performance when the product of kernel size and number of iterations is small. For a mesh or a hypercube network, the redundant computation overhead introduced by the RBC algorithm may be larger than the communication overhead introduced by the conventional algorithm. In this situation, the conventional algorithm would be the best choice. The models developed in this work allow programming tools to select the appropriate algorithm and number of processors for a given problem and machine characteristics.

References

[1] Ronald N. Bracewell. *Two-dimensional Imaging.* Prentice-Hall Inc., 1995.

[2] J. M. Crichlow. *An Introduction to Distributed and Parallel Computing.* Prentice Hall, 1997.

[3] S. M. Figueira and F. Berman. Predicting Slowdown for networked Workstations. *IEEE*, August 1997.

[4] Ian Foster. *Designing and Building Parallel Programs.* Addison Wesley Publishing Company, 1994.

[5] G. C. Fox, R. D. Willians, and P. C. Messina. *Parallel Computing Works!* Morgan Kaufmann Publishers Inc., 1994.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine–A User's Guide and Tutorial for Networked Computing.* MIT Press, 1994.

[7] J. C. Gomez, V. Rego, and V. Sunderam. CLAM: Connection-less, Lightweight, and Multiway Communication Support for Distributed Computing. *Communication and Architectural Support for Network-Based Parallel Computing: First International Workshop; Proceedings/CANPC'97*, February 1997.

[8] A. Y. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures. *IEEE Parallel & Distributed Technology*, August 1993.

[9] A. S. Grimshaw, W. T. Strayer, and P. Narayan. Dynamic, Object-Oriented Parallel Processing . *IEEE Parallel & Distributed Technology*, May 1993.

[10] R.M. Haralick and L.G. Shapiro. *Computer and Robot Vision.* Addison Wesley Publishing Company, 1992.

[11] J. F. Karpovich, M. Judd, W. T. strager, and A. S. Grimshaw. A Parallel Object-oriented Framework for Stencil Algorithms. *Proceedings of Second International Symposium on High Performance Distributed Computing*, July 1993.

[12] M. J. Quinn. *Parallel Computing : Theory and Practice.* McGraw-Hill, 1994.

[13] D. A. Reed and R. M. Fujimoto. *Multicomputer Networks: Message-based Parallel Processing.* The MIT Press, 1987.

[14] Charles L Seitz. *Resources in Parallel and Concurrent Systems.* ACM Press, 1991.

[15] Jan Tenber. *Digital Image Processing.* Prentice-Hall Inc., 1991.