Homayoun

Reference 45

Convolution on Splash 2

Nalini K. Ratha, Anil K. Jain & Diane T. Rover*
Department of Computer Science
* Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824
ratha@cps.msu.edu, jain@cps.msu.edu, rover@ee.msu.edu

Abstract

Convolution is a fundamental operation in many signal and image processing applications. Since the computation and communication pattern in a convolution operation is regular, a number of special architectures have been designed and implemented for this operator. The Von Neumann architectures cannot meet the real-time requirements of applications that use convolution as an intermediate step. We combine the advantages of systolic algorithms with the low cost of developing application specific designs using field programmable gate arrays (FPGAs) to build a scalable convolver for use in computer vision systems. The performance of the systolic algorithm of Kung et al. [1] is compared theoretically and experimentally with many other convolution algorithms reported in the literature. The implementation of a convolution operation on Splash 2, an attached processor based on Xilinx 4010 FPGAs, is reported with impressive performance gains.

1 Introduction

Convolution is an important operator in digital signal and image processing. Many machine vision systems use 2-dimensional convolution for image filtering, edge detection, and template matching. Convolution of two signals f and g is often expressed as $h = f * g$, where h is the result of applying convolution mask g on input signal f. For 2-dimensional discrete images, convolution can be defined as follows:

$$h(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x-i, y-j) g(i,j). \qquad (1)$$

For images of finite size (say $N \times N$), and masks of finite size (say $k \times k$), 2-dimensional convolution of f (image) and g (mask) can be expressed as

$$h(x,y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} f(x+i, y+j) g(i,j). \qquad (2)$$

In the second form, it is also known as template matching, or correlation. The image size and mask size need not contain square number of elements in general.

Typical use of convolution in image processing is for edge detection using, for example, Sobel, and Prewitt masks; computing texture orientation; object location using template matching; and image smoothing using Gaussian masks. Convolution operation described in Eq. (2) can include integer-valued and real-valued input images and masks. Examples of real-valued masks are image smoothing using Gaussian masks, and computing texture features using Gabor filters. Most commonly used masks such as Laplacian, Sobel, and Prewitt have not only integer mask values, but the mask values can be expressed as powers of two. In a typical application involving many image processing stages, each stage should be capable of processing real-valued images. Based on this observation, we can have a taxonomy of convolution operation as shown in Figure 1.

Many approaches to perform convolution using special architectures have been reported in the literature [1, 2, 3]. A very simple distributed approach for convolution is to split the input image into a set of smaller, possibly overlapping, sub-images; the number of subimages is the same as the number of processing elements (PEs). Each PE produces the result for the sub-image it receives. The spatial support for the convolution mask is provided through the overlap at the boundaries or data replication. However, this simplistic approach may not be suitable for all target architectures. For example, if the image is being acquired line by line, this approach may not be efficient as we have to wait till we acquire the whole image. In order to understand the advantages and disadvan-

0-8186-7086-X/95 $04.00 © 1995 IEEE

tages of various algorithms reported in the literature, we need to look at the communication and computation pattern in 2-dimensional convolution. The basic convolution operation is shown schematically in Figure 2. Suppose the value of the convolution operation is desired at the point (x,y). The center of the mask is placed at (x,y). A point-wise inner product of the image pixels and mask values is computed, followed by a reduction sum operation. This computes the output value at (x, y). The reduction operation can also be a prefix sum, although the intermediate results are not directly useful. This basic set of operations is repeated at all possible (x, y) locations.

The sequential version of convolution algorithm is very simple and is shown in Figure 3. There are four loops in the algorithm and its overall complexity is $O(N^2k^2)$. The simple data partitioning approach described above can reduce the total time by a factor equal to the number of available processors, ignoring the extra computations needed in the overlapping areas by each PE.

The previous work described in the literature can be summarized based on the target architecture on which convolution operation is implemented.

- Systolic: One of the most widely used convolution algorithm is the systolic algorithm by Kung et al. [1]. The algorithm is fairly straight forward and also scalable to higher dimensions using the 1-dimensional convolution algorithm as the building block. In his landmark paper "Why systolic architectures?" [4], Kung described many convolution algorithms on systolic structures. Based on a general inner product computation, Kulkarni and Yen [5] proposed a systolic algorithm for 1-dimensional and 2-dimensional convolution.

- Hypercube: Fang et al. [2] have described an $O(k^2/p^2 + klog(N/p) + logN * logp)$ algorithm, where $1 \leq p \leq k$, and using N^2k^2 PEs and an $O(N^2M^2/L^2)$ algorithm using L^2 PEs. Using N^2 PEs, Prasanna Kumar and Krishnan [6] proposed an algorithm with the best time complexity of $O(N^2/K^2 + logN)$. With a fixed number of PEs, their time complexity changes to $O(k^2logk + logN)$.

- Mesh: Many researchers [7, 3] have proposed schemes for convolution on mesh connected architectures. Lee et al. [7] use computation along a Hamiltonian path ending at the center of the convolution mask, called the convolution path. Ranka and Shani [3] do not broadcast the data values, thereby improving the performance of their algorithm by an order of magnitude.

- Pyramid: Pyramid architectures are useful in dealing with multi-resolution images. An $O(k^2 + logN)$ time complexity algorithm is described by Chang et al. [8].

- VLSI/ASIC: Most of the approaches recommend the suitability of their algorithm for a VLSI implementation. Chakrabarti and Jaja use a linear array of processors in their algorithm [9]. In [5] convolution is viewed as a generalized inner product and a VLSI implementation for 2-dimensional convolution is described. Ranganathan and Venugopal [10] have described a VLSI architecture for template matching using k^2 PEs and they achieve a time complexity of $O(N^2/2 + K^2)$.

In this paper, we describe a convolution algorithm suitable for implementation on field programmable gate arrays (FPGAs). FPGAs have been used in designing many custom computing structures. Recent advances in hardware technology, enabling more logic blocks in FPGAs, make them more suitable for many complex applications needing more logic. Recently, Barros and Akil [11] have used FPGAs for low-level image processing. Athanas et al. [12] describe many image processing functions implemented on Splash 2. We use Splash 2, an attached processor on Sun SPARCstations based on Xilinx 4010 FPGA PEs. Our 2-dimensional convolution algorithm implementation has been designed and synthesized for Splash 2 architecture.

The rest of the paper is organized as follows. Splash 2 architecture and software environment is described in Section 2. The convolution algorithm is described in section 3. The performance of the proposed algorithm is evaluated by comparing its execution time and its complexity against some other algorithms reported in the literature. The computer aided design (CAD) tools provide a performance estimate which is also compared with the real speed achieved in section 4. In Section 5 conclusions and plans for future work are given.

2 Splash 2 Architecture

The Splash 2 system consists of an array of Xilinx 4010 FPGAs, improving on the design of the Splash 1 based on Xilinx 3090s [13]. The Splash 2 system is connected to the Sun host through an interface board that extends the address and data buses. The

host can read/write to memories and memory-mapped control registers of Splash 2 via these buses. A detailed description of the system is given in [14]. Each Splash 2 processing board has 16 Xilinx 4010s as PEs ($X_1 - X_{16}$) in addition to a seventeenth Xilinx 4010 (X_0) which controls the data flow into the processor board. Each PE has 512 KB of memory. The Sun host can read/write this memory. The PEs are connected through a crossbar that is programmed by X_0. There is a 36-bit linear data path (SIMD Bus) running through all the PEs. The PEs can read data either from their respective memory or from any other PE. A broadcast path also exists by suitably programming X_0.

Splash 2 system supports several models of computation, including PEs executing the same instruction on multiple data (SIMD mode) and PEs executing multiple instructions on multiple data (MIMD mode). It can also execute the same or different instructions on single data by receiving data through the global broadcast bus. The most common mode of operation is systolic in which the SIMD Bus is used for data transfer. Also individual memory available with each PE makes it convenient to store temporary results and tables.

To program Splash 2, we need to program each of the PEs (X_1- X_{16}), the crossbar, and the host interface. The crossbar sets the communication paths between PEs. In case the crossbar is used, X_0 needs to be programmed. The host interface takes care of data transfers in and out of the Splash 2 board. A special library is available for these facilities for VHDL programming as described in [14].

3 Convolution Algorithm on Splash 2

Image processing algorithms, in general, convolution, in particular, demand high I/O bandwidth. Most of the algorithms proposed on special architectures assume that data are already available on the PEs. This, in a way, avoids the I/O bandwidth problem of the convolution operation. We do not make this assumption. Jonker [15] argues that linear arrays are better for image processing algorithms. A linear array of PEs operating in a systolic mode offers two advantages: (i) systolic arrays can balance I/O with computations, (ii) the nearest neighbor communication can eliminate the need for a global communication facility for some class of algorithm. One of the preferred modes of computations on Splash 2 is the systolic mode. In this mode, we assume nothing about availability of data in the individual PEs. The computations needed in a PE are also fairly simple. This helps us in balancing the I/O bandwidth requirement and computation requirement at a PE. Hence, we prefer a systolic algorithm for convolution on Splash 2.

First, we describe the 1-dimensional convolution algorithm. A systolic 1-dimensional convolution is simple. Let us assume that we have k PEs, where k is the size of the mask. Each PE receives from its left neighbor the pixel value and the partial result available so far. The PE multiplies the pixel value with the mask value it handles and adds the partial sum to it. This result and the pixel value are passed to its right neighbor. At the end of the systolic path, we get the convolution result after taking into account initial latency. A schematic diagram of 1-dimensional convolution on a set of PEs connected linearly is shown in Figure 4(a).

The above algorithm assumes that the PEs can do multiplication. In a FPGA-based PE, this is not true. A double precision floating point multiplier needs more logic than what is available in a PE. We can use the local PE memory to store the multiplication table indexed by the pixel value. This results in an additional delay of one cycle necessary to reference the multiplication result from the memory. This scheme is shown in Figure 4(b).

Our 2-dimensional convolution is an extension of the 1-dimensional convolution described above. The basic idea is based on the algorithm proposed by Kung et al. [1]. The $k \times k$ mask is extended to a $k \times N$ mask with 0's placed at locations where no entry was present. These kN entries are serialized to get a single 1-dimensional mask of kN entries. Now, we can apply the 1-dimensional convolution algorithms outlined above. Note that there are $(N - k)$ locations with 0's as their mask value. Hence, we can simply have $(N - k)$ stages of shift registers. Secondly, for improper positions of this new Nk-element mask we get values which are not really part of the output. These need to be ignored. Finally, the assumption on the input is that the pixels are being communicated in a raster scan order. Note that we can use either of the 1-dimensional convolution algorithms described above depending on the value of the convolution mask coefficients. The scheme is shown in Figure 4(c).

Implementation Issues

The general convolution algorithm needs to be tuned for the special hardware being used. For example, we have only 16 PEs. We need to map virtual PEs to physical PEs. The second issue is the number of shift registers which depends on the row size of the image. The third issue is implementation of multipliers

needed by the PEs. Following is a summary of our solution to these issues.

- Large number of mask entries: If mask size (k) is greater than the number of available PEs, the virtual PEs are mapped to available PEs. In carrying out the mapping, the timing model between PEs must be satisfied.
- Larger image width: Image has to be split into smaller sub-images of some predefined size. In order to handle this, we develop the 2-dimensional convolution with a fixed width (32 in our case). Larger images are split into sub-images of 32 $\times$ 32 pixels and depending on the mask size the required rows and columns at the border are copied.
- Multiplier implementation:
  - Masks with integer values: If mask values are of the type 2^p, then the multiplier in each virtual PE can be replaced with simple bit shifters.
  - Integer masks but not powers of 2: An efficient scheme for this is being developed.
  - Real valued masks: If mask values are normalized (ranges between -1 to 1), then a suitable scheme for implementation on FPGAs is needed.

The Splash PEs carry out two different activities, namely, (i) additions and multiplications, and (ii) shift operations. We have built two different elements for each of the activities, namely, (i) compute element, and (ii) shift element. The number of compute elements and shift registers is determined by the mask size and image width. The schematics of both these elements are shown in Figure 5.

4 Results

A brief summary of the analysis of several convolution algorithms is given in Table 1 based on the communication facility available on the respective architectures. In a systolic algorithm, the communication overhead is balanced by the computation phase. Moreover, no complex communication facility is needed. In this sense, systolic algorithm is better in terms of total work done and communication simplicity.

We compare our implementation on Splash 2 with implementations on different platforms in terms of total execution time. The timings for 3×3 convolution-based Sobel edge detector on a 512×512 image are shown in Table 2. The basic sequential convolution algorithm (Figure 3) running on different Sun host machines has been timed. In addition, timing on a recently developed i-860 based system from Alacron is reported. The timing results on CM-5 are taken from [16] and are for edge detection using a set of six 5×5 convolution masks.

For implementing the convolution algorithm on Splash 2, we have chosen three standard edge detectors used in low-level computer vision algorithms. The filters and their mappings on Splash 2 PEs are shown in Figure 6. These operators have two convolution stages and a final stage to compute the edge magnitude at each pixel by computing the absolute sum of the two convolution output images. Each PE accommodates the required shift registers for that stage. The outputs for the house image using 3×3 Sobel edge detector, 5×5 Prewitt edge detector and 7×7 Prewitt edge detector are shown in Figure 7. The timings for these edge detectors on Splash 2 are shown in Table 3.

Our approach for implementing convolution operation on Splash 2 is different in many ways from the approach taken by Peterson et al. [17]. The main differences are: (i) We are not limited by a fixed mask size of 8×8 as done in [17]. For smaller masks, Peterson et al. [17] have used the same 8×8 masks filled with zeros. This may be due to hard-coded model of computing on Splash 2; (ii) Peterson et al. use all the 16 PEs for implementing their algorithm. We use less number of PEs for smaller mask sizes ($k < 8$). Therefore, in our approach, we will have more PEs available for implementing many other stages of a vision system; and (iii) Our mapping on Splash 2 results in a higher performance of nearly 100 frames per second compared to 15 frames per second reported by Peterson et al. [17].

5 Conclusions

Convolution with integer mask values that are powers of two has been efficiently implemented on Splash 2. The algorithm scales well with mask size. For the most general case of real-valued masks could be handled using look-up tables.

We are currently designing a fixed point multiplier on the chip to achieve an efficient implementation with real-valued masks. Because of the limit on logic that can be accommodated in a PE in Splash 2, mapping of virtual PEs to physical PEs will become difficult after a certain point. Moreover, if we have a large number of lookups, then the performance degrades linearly. Manseur [18] has proposed a scheme to decompose

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.