CONVOLUTION ALGORITHMS ON THE PIPELINED IMAGE-PROCESSING ENGINE

by

Charles V. Stewart
and
Charles R. Dyer

Computer Sciences Technical Report #643

May 1986

Convolution Algorithms on the Pipelined Image-Processing Engine

Charles V. Stewart
Charles R. Dyer

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract

In this paper we present two algorithms for computing an N by N convolution on the Pipelined Image-Processing Engine (PIPE). PIPE is a special-purpose machine for low-level vision consisting of eight processing stages connected in a pipelined fashion. Each stage can compute a number of different basic image functions. Each of the algorithms decomposes the solution into a sequence of 3 by 3 neighborhood operations, shifts and additions. The first algorithm divides the results of the 3 by 3 partial convolutions into groups of concentric rings of images and successively collapses the images on the outer ring onto the next ring, merging the results until a single result image is computed. The second algorithm divides the partial convolution images into eight sectors and computes each sector independently. For a 27 by 27 convolution of a 256 by 256 image the first algorithm requires 0.633 seconds, while the second requires 0.683 seconds. These algorithms for PIPE are also shown to compare favorably with algorithms for arbitrary sized kernels on fast general purpose hardware, the MPP and Warp.

The support of the National Bureau of Standards under Order No. NANB510565 and the National Science Foundation under Grant No. DCR-8520870 is gratefully acknowledged.

1. Introduction

Because of its common usage and high computational cost, the convolution operation is one of the most important operations for low-level vision and image processing. For example, it is used for smoothing operations and local feature detection. N^2 multiplications and $N^2 - 1$ additions are required to compute the value of the convolution at each point of an image, where the kernel is size N by N. Thus, the total computation requires $O(M^2 N^2)$ operations on an M by M image.

The Pipelined Image-Processing Engine (PIPE) is a special purpose machine for low-level vision designed at the National Bureau of Standards [1]. PIPE can accept a 256 by 256 image (8 bits per pixel) as input every 1/60 second. The images are processed by a series of processing stages on PIPE that are connected in a pipelined fashion. Each stage can perform a number of arithmetic, neighborhood, and logical operations on one or more images within this 1/60 second. These operations are performed at every point in an image. PIPE outputs its processed images either to a host or to another processor, called ISMAP, which is designed to extract more global and symbolic features of the images. Thus, PIPE's design intent is to facilitate rapid processing of low-level features of images.

PIPE provides two 3 by 3 neighborhood operators in each stage for performing convolutions. To form a general N by N convolution, we will compose a sequence of 3 by 3 convolutions and use a variety of other operations to merge these 3 by 3 results so that the final image will contain the result of the N by N convolution at each point.

In the remainder of this paper we develop and analyze two algorithms for N by N convolutions on PIPE. Section 2 describes the PIPE architecture. Section 3 presents general requirements for convolutions on PIPE. Section 4 describes and analyzes Algorithm 1 in detail. Section 5 describes Algorithm 2. Section 6 examines some alternative algorithms for convolutions on PIPE. Section 7 compares convolutions on PIPE with convolutions on other processors. Section 8 presents some concluding remarks about PIPE and convolution operators on PIPE.

2. PIPE Architecture

Images are the basic units operated on by PIPE. All of the operations work on images, and all of the datapaths communicate images. As indicated above, PIPE consists of a sequence of stages. In addition PIPE has an input stage, an output stage, and each stage has a direct connection to the memory of a host machine. The connections between the successive stages of PIPE are shown in Figure 1.

During each time unit, each stage, i, is able to output three images independently, one to stage $i+1$ via the forward path, one to stage $i-1$ via the backward path and one to itself via the recursive path. In addition, there are two "wildcard" paths, VBUS A and VBUS B, which are used to output an image to one or more stages. These wildcard paths are not shown in Figure 1. Only two images may occupy these wildcard paths at any time. The input stage receives images from input devices and outputs them to stage 1 via its forward path. Input images may also go directly to other stages, if necessary, via one of the wildcard buses. The output stage, not shown in Figure 1, receives images from the last stage's forward path or from other stages via the wildcard bus. This stage outputs images independent of the processing of the other stages. At present there are eight stages on PIPE as well as the input and output stages.

The standard pipelined operation results in a sequence of images each being processed on a stage, passed to the next stage for further processing, and finally exiting PIPE through the output stage. Thus, there is a continuous flow of images through PIPE. However, the architecture allows a more flexible set of operations. The image convolution algorithms considered here demonstrates one example. For the most part, they treat the stages as synchronous parallel processors and complete the processing on one input image before proceeding to the next. For other possible uses and for more details on the hardware configuration see [1].

Figure 2 shows an individual stage of PIPE. A stage can perform three types of operations. All operations on a stage take place within 1/60 second. The simplest is a point-wise arithmetic operation. It is implemented via a look-up table (LUT). The second type of operation involves a

point-wise function of two images. A simple arithmetic logic unit (ALU) computes these results. Finally, as discussed above, there are two neighborhood operators (NOP's). All of these operators work with either arithmetic or boolean pixel values, but here we consider only arithmetic values.

The processing in a stage can be divided into two parts: those operations that are performed before BUFF X and BUFF Y, and those that follow them. BUFF X and BUFF Y are buffers that store an image. The pre-buffer operations combine up to three input images into one using the LUT's and ALU's shown. Each buffer receives its input from either ALU C, one of the wildcard buses, or the DMA bus. If no input to a buffer is specified, it maintains its previous image. This is important in maintaining results from prior operations for future use.

Following the stage buffers there are two parts. First there are the NOP's. They perform 3 by 3 convolutions at each point in the image. The individual weights of the convolutions are 8 bits. The internal temporary results of the convolutions are 12 bits, so that the 8 bit accuracy of the overall image is maintained. *One* of the two buffers is used as input to the PRE-NOP LUT and then piped into *both* neighborhood operators. The NOP's compute separate functions. The second part following the stage buffers allows images to be combined using an LUT and an ALU. The LUT (TVF LUT) receives one or two images as input. If there are two input images then a function of 12 bits is computed with 8 bit output. Otherwise, the normal 8 bit input and 8 bit output is used. The ALU (ALU C) requires two images as input. The images input to the TVF LUT or the ALU C may come from either of the stage buffers or from either of the NOP's. If a NOP is empty then the output from it is the result of the PRE-NOP LUT.

Finally, we consider the outputs from a stage. The first method of output allows the contents of either of the stage buffers to be output to the host via the DMA bus (see the middle of Figure 2). The second method sends output to any subset of the forward, recursive or backward paths, or either of the wildcard buses. The output to each is determined independently and may come from either of the buffers, either of the NOP's, the TVF LUT, or the ALU C. The overall affect of these features yields a considerably more flexible design than the conventional one-way pipeline.

This discussion of PIPE has excluded several features that are not germaine to the convolution algorithms described below. They include a provision for controlling image boundary effects and a "region-of-interest" operator. See [1,5] for more details on the PIPE hardware.

3. General N by N Convolution on PIPE

We now consider the problem of computing the convolution of an image with an N by N kernel. Formally, the convolution is defined as follows. Letting $m = \frac{N-1}{2}$, R be the result array, A be the kernel array, and I be the image array, then the result of the convolution at each point i, j is:

$$R(i,j) = \sum_{k=-m}^{m} \sum_{l=-m}^{m} A(m+1+k, m+1+l) * I(i+k, j+l)$$

For the sake of simplicity, assume that N is an odd multiple of 3 in the remainder of the discussion. Let $n = \frac{N}{3}$. The algorithm developed can be extended to any value of N, but the discussion is simpler with this restriction.

The overall strategy will be to partition the N by N kernel into 3 by 3 blocks, apply these 3 by 3 convolutions ("partial convolutions") independently to the entire input image using the NOP's, and then combine the partial result images to obtain the final result image. The major difficulty comes in combining the partial results. After application of the 3 by 3 operators there will be n^2 $(= \frac{N^2}{9})$ images to combine. These images are shifted with respect to one another, so that combining can not take place directly. To see this, suppose that A is the N by N kernel. Let I(1) be the image produced by the 3 by 3 partial convolution of the upper left corner of A, i.e. $A(i,j)$, $1 \le i, j \le 3$. Let I(c) be the image produced by 3 by 3 partial convolution at the center of A, i.e. $A(i,j)$, $\frac{1}{2}(N-1) \le i, j \le \frac{1}{2}(N+3)$. To combine I(1) and I(c) properly we must shift each point of image I(1) a distance of $\frac{1}{2}(N-3)$ so that I(1) is in register with I(C) and then add the two images. Finally, notice that the NOP's can not be used to combine images since each NOP has a single image as input.

Thus, the shift operation is central to the general convolution algorithm. In PIPE, it can only be accomplished using the 3 by 3 neighborhood operator. For example, the following set of 3 by 3 weights produces a result image that is shifted down and to the right with respect to the source image:

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Similarly, moving the 1 clockwise produces shifts: down, down left, left, up left, up, up right, and right respectively. Thus, an image can shift less than or equal to 1 row and less than or equal to 1 column using one NOP. To shift an image i rows and j columns requires *max* (i,j) operations. Hence, the algorithm must be organized for efficient image shifting.

The images resulting from the 3 by 3 partial convolutions on the boundary of the N by N kernel must be shifted farthest. There are $4(n-4)$ of these partial result images. If they are shifted to the center independently before adding them, they will each require $\frac{1}{2}(N-3)$, or $\frac{3}{2}(n-1)$ shifts, for a total of approximately $6n^2$ shifts. Each of these shifts requires an individual stage, for a total of $\frac{3}{4}n^2$ time units for all of them. However, if we can organize the shifting so that two partial result images reach the same intermediate position, then we may combine them immediately using an ALU. After this we only need to shift the summation image instead of the two separate images.

4. Algorithm 1

Using the ideas of the previous section, we now present an algorithm for the general N by N convolution. Since shifting is the critical operation, the algorithm works on the outermost partial result images first. These are all of the images for 3 by 3 partial convolutions at the border of the kernel (the *outermost ring* of 3 by 3 partial convolutions of the kernel). The algorithm shifts these towards the center of the kernel. While the images are shifted they are combined with each other in pairs or in triples after shifting two units. The general organization and shift directions are summarized in Figure 3. Note that the specific shifts and combinations are not shown. Also, recall that entire images are being shifted so that the points in Figure 3 correspond to entire images instead of individual values. After the images are shifted three positions they are combined with the 3 by 3 partial convolutions on the next outermost ring. This process continues, producing a ring of partial summation images which shrinks as it propagates toward the center of the kernel where the final result is obtained.

Now we consider the details of mapping this approach onto PIPE. As mentioned above, the algorithm does not make use of PIPE in a pipelined manner; instead it views the stages as parallel processors with a predetermined set of paths between them. The operational details are best clarified by two examples.

4.1. Example 1

Consider a four stage PIPE and the computation of an N by N convolution. In particular, consider combining the results of the six parts of the convolution operator in the upper left corner of the kernel. The six parts are the images resulting from the 3 by 3 partial convolutions centered at points (2,2), (5,2), (2,5), (2,8), (5,5) and (5,8) relative to the center of the kernel. Denote them as I(2,2), I(5,2), etc. Notice that the first four of these are on the outermost ring and the other two are on the next outermost ring. Figure 4 shows the details of the PIPE stage operations which combine these into two result images, centered at points (5,5) and (5,8).

At time 1 the input image is transmitted over the wildcard bus into the X buffer of each stage, and then immediately into the NOP where the appropriate 3 by 3 partial convolution is computed: I(2,2) in stage 1, I(2,5) in stage 2, I(5,2) in stage 3, and I(2,8) in stage 4. Notice that these are the four images on the outermost ring.

The results of each computation are recursively sent back into the stage and stored in the BUFF Y. At time 2 each of these images is shifted, sent out the recursive path, and stored in BUFF Y. I(2,2), in stage 1, is shifted directly toward (5,5). I(2,5) and I(5,2), in stage 2 and stage 3, are shifted so that they can meet I(2,2). I(2,8) is shifted at times 2, 3 and 4 so that it combines with I(5,8). At time 3, I(2,2) is again shifted down and right, while I(5,2) and I(2,5) are shifted right and down respectively. Figure 4 shows the shift paths for each partial convolution image.

At this point, these three images are in register with each other so they may be combined directly at (4,4). This is accomplished by sending I(2,2) out the forward path from stage 1, I(2,5) out the recursive path from stage 2, and I(5,2) out the backward path from stage 3, combining the results using ALU A and ALU B (with add operations) in stage 3 at time 4. Call the resulting image J(4,4). Notice that this occurs prior to the buffer. After storing J(4,4) in the buffer, use the NOP to shift it down and to the right again.

The combining of three images into J(4,4) has freed stage 1 and stage 3 at time unit 5. Their X buffers still contain the original image. Use stage 1 to compute I(5,5) and send the result out the forward path. Use stage 3 to compute I(5,8) and send the result out the recursive path.

At time 5 the final results are computed. In stage 2 I(5,5) and J(4,4) are combined using ALU B. The same happens in ALU A of stage 3 for I(2,8) and I(5,8). Call the results J(5,5) and J(5,8) respectively. Note that they are computed before the stage buffers, so that the NOP's following the stage buffers can be used for other operations at time 5. This fact will be used heavily later on.

4.2. Example 2

We now show another example of combining partial convolution images when these images are not near the corners of the kernel. Figure 5 shows the combination of I(8,2), I(11,2), I(14,2), I(17,2) (assume N > 21) on a 4 stage PIPE. These images on the outermost ring must be shifted toward the next ring, combined with each other in pairs, and then combined with some images in the next ring.

The first three time units are similar to Example 1. I(8,2) and I(11,2) are combined at the beginning of time 4, as are I(14,2) and I(17,2). This frees two of the stages at time 4 to compute I(11,5) and I(17,5). These two images are then combined with the results from the outer ring at the beginning of time 5. I(8,5) and I(14,5) are also computed at time 5 using NOP's. They require two of the four stages. For the moment we consider the other two stages to be idle although later we will make use of them to start part of the next set of operations.

4.3. Putting It Together

Using the techniques described for the previous examples, we now give the complete algorithm for computing the N by N convolution using eight stages on PIPE. There are n^2 3 by 3 partial convolution images. $4n - 4$ of these are on the outermost ring, i.e. $4n - 4$ of these images are shifted the maximum distance from the center point of the kernel. Take four sets similar to those of Example 1. These sets correspond to the four corners of the kernel. Two of these sets may be done concurrently, one on the first 4 stages, the other on the last 4 stages. Call these Type 1 operations.

Divide the remainder of the outermost ring into sets of 8 images and operate on them as shown in Example 2. Call these Type 2 operations. After these are done, the outermost concentric ring will be collapsed into the next outermost ring. Repeat the process for the next outermost ring. Note, however, that for this ring the first time units of both Example 1 and Example 2 are unnecessary: the starting points will be partial result images (e.g. J(5,5)), instead of the initial image.

Repeat this until there is only one ring around the center point. There are now nine images to combine centered as follows

a	b	c
d	e	f
g	h	i

Note that the eight images on the outside will each require three shifts to align with the center, e. Shift a, b and d using stages 1-3 so that they are combined using stage 2 at the beginning of time 3. Shift the result to the center position, e, at the end of time 3. Do the same for f, h and i in stages 5-7 computing the sum and shifting the result using stage 6. Meanwhile in time 1-3 shift g and c to the center in stages 4 and 8, respectively. At time 3 perform 3 by 3 partial convolution e in stage 3. Finally, at time 4 combine three of the results, from stages 2-4, in stage 3 and the other two results, from stage 6 and 8, in stage 7. Use the wildcard bus to get the result from stage 7 to stage 3 so that they may be combined in ALU C at the end of time 5. This image is the result of the N by N kernel applied to the entire image.

4.4. Analysis

Now we analyze this algorithm to find a closed form equation for the number of time units required for the N by N convolution of an image. Recall that when the algorithm is working on the outermost ring, one additional time unit is required in each set of operations for the initial 3 by 3 convolutions. Let c_n be the number of time units for the initial 3 by 3 partial convolutions, where $3n = N$. Let b_n be the number of shifts, adds, and 3 by 3 convolutions to complete the entire convolution after this first set of 3 by 3 convolutions. Thus the total number of time units is $a_n = b_n + c_n$.

There are $4n - 4$ 3 by 3 convolution images on the outermost ring. Eight of them are processed in one time unit. Since n is odd, $(4n - 4)\ mod\ 8 = 0$, and $c_n = \frac{1}{2}(n - 1)$ so,

$$a_n = b_n + \tfrac{1}{2}(n - 1)$$

For the corners of the outermost ring of the kernel, 16 3 by 3 convolution images are involved in Type 1 operations. The operations may be shifted in two groups of 8 images in a little more than 3 time units for each pair. This effectively becomes 3 time units since all the work at time 4 occurs before the stage buffer and, therefore, the NOP may be used for the next set of operations. So Type 1 operations require 6 time units. The remaining 4n-20 images in this ring are part of Type 2 operations. They are performed in groups of 8 images, requiring 4 time units for each group. The result is $2n - 10$ time units for Type 2 operations on the outermost ring. Thus, we have

$$b_n = 2n - 4 + b_{n-2}, \qquad b_3 = 5$$

Substituting d_k for b_n, where $2k + 3 = n$ yields

$$d_k = 4k + 2 + d_{k-1}, \qquad d_0 = 5$$

Solving this recurrence relation and substituting back into the equation for b_n yields

$$b_n = \tfrac{1}{2}n^2 - n + \frac{7}{2}$$

and

$$a_n = \tfrac{1}{2}n^2 - \tfrac{1}{2}n + 3$$

Finally, since $3n = N$, the total number of time units required is

$$A_N = \frac{N^2}{18} - \frac{N}{6} + 3$$

By a slight modification involving Type 2 operations we can save time for larger values of N (greater then 15). Recall that there were two unused stages in the final time unit of Example 2. When there are eight stages, four of them will be unused in the final time unit in Type 2 computations. By starting the next computation in these unused stages we can do two Type 2 operations in 7 time units instead of 8. Also, if a Type 1 operation follows a Type 2 operation we can start part of the Type 1 operation one time unit earlier and can complete that part of it earlier. Note that both of these modifications require a slight reorganization of the processing. To see how much is saved by this we will count the empty stages in Algorithm 1 and divide by the number of stages, eight.

For the outermost ring, there are 4n-20 images combined in Type 2 operations. With an eight stage PIPE, there are ½(n − 5) Type 2 operations on the outermost ring. Four stages are unused at the end of each Type 2 operation, for a total of $2n$ − 10 unused stages. Letting e_n be the number of stages saved with a N by N kernel we have:

$$e_n = 2n - 10 + e_{n-2}, \qquad e_5 = 0$$

Solving this relation yields:

$$e_n = \tfrac{1}{2}(n^2 - 8n + 15)$$

and, the total number of time units saved is:

$$\left\lfloor \frac{(n^2 - 8n + 15)}{16} \right\rfloor$$

Thus, the final time unit equation for Algorithm 1 is:

$$\left\lceil \frac{(7n^2 + 33)}{16} \right\rceil$$

or, in terms of N,

$$\left\lceil \frac{(7N^2 + 297)}{144} \right\rceil$$

An analysis of the space requirements of Algorithm 1 is also necessary since PIPE has a limited number of image buffers. In the algorithm, all 3 by 3 partial convolution and shifting operations are completed for the outermost ring of images before proceeding to the next outermost ring. With a 27 by 27 convolution this means there will be 24 partial result images. The original image also needs to be stored. Currently, PIPE contains two buffers per stage, so there is not enough buffer space to store all these images simultaneously. Reordering the computation does not, by itself, solve the problem. To solve the problem requires either more buffers or that the DMA bus be used to temporarily store some of the partial results on the host instead of on PIPE. Future versions of PIPE will expand the number of buffers from 2 to 32.

At present, using the DMA bus solution may result in space or timing problems if either there is not enough memory space on the host, or if the DMA bus is too slow to keep up with the

algorithm. When $N \leq 9$ no use of the DMA bus is required since there will only be a maximum of eight partial results at any time. When $9 < N \leq 15$ the DMA bus is also not necessary, but for a different reason. Only two Type 1 and no Type 2 operations are necessary. After the first Type 1 operation there will be four partial result images. There is enough buffer space for these images, but the initial image must be available again. It may be stored in the input stage in a buffer and passed back as necessary via a wildcard bus.

4.5. Extensions

Several extensions of PIPE could significantly affect the speed of this algorithm. The first is expanding it to 16 stages. (This extension is, in fact, possible with current hardware.) This does not result in halving the number of operations required, but it comes close. The final merging step can be reduced from 5 to 4 time units, and in both Type 1 and Type 2 operations twice as many partial result images may be handled. The resulting number of PIPE time units required is approximately $\frac{7N^2}{288}$ for large values of N. Note that the use of 16 stages reduces the buffer problem as well.

Another extension is to a 5 by 5 hardware neighborhood operator instead of 3 by 3. This extension is already under consideration [1]. It will add accuracy as well as speed since the internal results of the NOP is 12 bit instead of 8. Also, shifting can occur over a distance of up to two rows and two columns in one NOP. Notice that by assuming 5n = N, Algorithm 1 does not change. The partial result convolutions are 5 by 5 and the shifts occur over distance 5 in 3 time units. Above, the convolutions were for 3 by 3 and the shifts occurred over distance 3 in 3 time units. Thus, the resulting value of a_n is the same as above, but the total time, A_N, is approximately $\frac{7N^2}{400}$. Thus the decrease in time due to a 5 by 5 NOP is approximately $\frac{9}{25}$. This extension also reduces the buffer problem since fewer partial result images are computed.

5. Algorithm 2

In this section we describe a second algorithm for computing an N by N convolution on PIPE. It is slightly slower than Algorithm 1, but it has no problems with buffer space. The algorithm works as shown in Figure 6. The n^2 3 by 3 partial results are broken into eight separate groups such that each image in a group is three shift units from at least one other image in the group. Also, the images in each group are linearly ordered such that there is a path connecting each image in a group. This path starts at a 3 by 3 partial convolution on the border of the kernel and ends at a 3 by 3 partial convolution that is three shift units from the center point of the kernel. The processing proceeds independently on each group with each stage dedicated to processing a group. Thus, whereas Algorithm 1 oriented the computation with respect to concentric rings of partial convolution images, Algorithm 2 organizes the computation on sectors of partial convolutions.

The processing requires that the initial image be stored in BUFF Y of each stage at all times. The first 3 by 3 partial convolution in each group is computed at time 1. At times 2-4 this partial result is shifted to be in register with the next partial convolution. During each time the partial results are stored in BUFF X. At time 5 the next 3 by 3 convolution is computed and added to the previous partial result using ALU C. This continues in each stage until the stage's BUFF X contains the summation of the partial results for the entire sector, but offset a distance of three from the center of the kernel. The eight results are then combined in exactly the same way as in Algorithm 1.

5.1. Analysis

The analysis of this algorithm is straight forward. Each stage computes $\frac{n^2-1}{8}$ 3 by 3 convolutions associated with its assigned sector of the kernel. Letting $k = \frac{n^2-1}{8}$, there are k time units required for the 3 by 3 convolutions and $3(k-1)$ shift time units for a total of $4k-3$ time units. The final part of the computation requires 5 time units just as in Algorithm 1. Thus the total number of time units required is:

$$\frac{1}{2}(n^2 + 1) = \frac{N^2 + 9}{18}$$

5.2. Comparison Between Algorithms 1 and 2

From the analysis of Algorithms 1 and 2, we see that Algorithm 1 is faster than Algorithm 2 by

$$\left\lfloor \frac{n^2 - 25}{16} \right\rfloor = \left\lfloor \frac{N^2 - 225}{144} \right\rfloor$$

time units. Table 1 gives some actual comparison times for various value of N.

N	Alg 1		Alg 2	
	units	time	units	time
15	13	0.22	13	0.22
27	38	0.63	41	0.68
45	101	1.68	113	1.88
81	321	5.35	365	6.08

Table 1. Comparison Between Algorithms 1 and 2 (time in sec).

Although Table 1 shows that the Algorithm 1 is faster, the problem of the extra buffer space required by Algorithm 1 on larger sizes of the kernel may counter balance this.

6. Other Algorithms

Many other algorithms for performing N by N convolutions are possible, but all others developed to date require more time to execute. For example, one obvious algorithm is to break the N by N kernel into eight equal parts and process them independently. Each part works by multiplying the image by one value of the kernel (using an LUT) and adding the result (call it the "kernel element image") to an accumulator image. This accumulator image must be shifted from one kernel point to the next so that it is in register with the multiplied image. This is similar to Algorithm 2 except that it works on individual points in the kernel instead of 3 by 3 blocks.

The implementation of this method makes use of an interesting technique which allows the multiply, add and shift operations to be performed in just one time unit. First notice that the shifting of the accumulator image must be performed by the NOP. In order to get the result back into the stage buffer in the next time unit it must proceed out the recursive path. If the kernel element image were computed after the stage buffer (using the TVF LUT), then combining the kernel element image with the accumulator image could only take place in either ALU A or ALU B. Unfortunately, this implies that both the kernel element image and the accumulator image must be output via the recursive path from the same stage. A similar problem occurs if the kernel element image is computed before the buffer, since again both the original image and the accumulator image need to be output via the same recursive path.

One solution to this problem is to have two adjacent stages work together. At the end of each time unit, each stage outputs the original image to the other stage. Each stage also sends its own shifted accumulator image out its own recursive path. The kernel element image is computed in either the forward or backward LUT and combined with the accumulator image in an ALU before the stage buffer. In the NOP the accumulator image is shifted so that it is in register with the next kernel element image. When each stage completely processes its sector the accumulator images are combined to yield the final result image.

The accumulation and shifting of images using this algorithm requires $\frac{N^2}{8}$ time units for an 8 stage PIPE. The final accumulation requires an additional 3 time units. For example, a 27 by 27 convolution takes 85 time units (1.42 seconds). Algorithm 1 requires 38 time units for this convolution (0.63 seconds).

Another possible algorithm is to divide the N by N kernel into 9 by 9 blocks. First combine the 9 3 by 3 partial convolutions to form a single summed result. Then recursively combine 9 summation images over a larger area, repeating this until there is one result. This algorithm is slower than both Algorithm 1 and 2. For example, this new algorithm takes 62 time units (1.03 seconds) for a 27 by 27 convolution whereas Algorithm 1 takes 38 time units (0.63 seconds).

7. Comparison With Other Machines

Table 2 compares the execution times for convolutions on various types of processors. A 256 by 256 image is assumed in all cases. The values for PIPE are based on Algorithm 1. The variations in PIPE configurations show the utility of several possible enhancements.

Machine	3 by 3	9 by 9	15 by 15	27 by 27	45 by 45
General Purpose (100 MFlops)	0.012	0.106	0.295	0.956	2.654
General Purpose (2 MFlops)	0.589	5.31	14.7	47.8	132.7
MPP, 128 by 128 Processor Array	0.001	0.011	0.037	0.153	0.85
WARP, 10 processors	0.013	0.118	0.302	0.958	2.663
WARP, 20 processors	0.013	0.066	0.158	0.486	1.339
PIPE, 3 by 3 NOP, 8 stages	0.017	0.1	0.22	0.63	1.68
PIPE, 3 by 3 NOP, 16 stages	0.017	0.1	0.15	0.35	0.85
PIPE, 5 by 5 NOP, 8 stages	0.017	0.067	0.1	0.25	0.63

Table 2. 256 by 256 Image - N by N Kernel (times in sec)

The execution times for the conventional, general purpose machines assume that for an N by N convolution, N^2 multiplications and $N^2 - 1$ additions are required for each point in the image. The speed in MFlops were chosen to approximate the CRAY and VAX hardware, respectively.

The MPP (Massively Parallel Processor) is a two dimensional array SIMD processor with 128 by 128 processing elements [4]. Each processing element is relatively simple (bit-level) with a limited memory, and is connected to its four nearest neighbors. There is also a staging memory for moving images in and out of the array.

To derive execution times for the MPP we use the following algorithm applied simultaneously at each processing element. Each processing element holds a single pixel. Starting from the upper-left corner of the kernel, multiply the kernel value by the image value and shift the result to the right row neighbor. Multiply the next value of the kernel by the current pixel value, add the previous result and shift again. Repeat this until the end of the row of the kernel is reached, then continue the process down the column, etc. When the shifting reaches the upper left corner again, the outermost square of the kernel is complete. Repeat the process on the next outermost square, adding values to the previous result. Repeat this spiraling process until the center point of the kernel is reached. The

result will be the value of the convolution applied at that point.

The algorithm used requires N^2 multiplications (18 μs for a 12-bit mult), $N^2 - 1$ adds (3.7 μs for a 12-bit add), and $N^2 - 1$ data shifts (1.7 μs for a 12-bit shift). 12-bit values were used both to reflect the internal accuracy of the PIPE NOP and as a compromise between the 8-bit precision currently in PIPE and future enhancements. Notice that the 256 by 256 image cannot simply be partitioned into four 128 by 128 disjoint pieces. The processing elements that are $\leq N-1$ elements from the border cannot compute a final result. Thus, letting $p = 128 - 2(N-1)$ only a p by p area of the image may be computed at one time. Thus, in general, the total time for this algorithm on the MPP is $\left\lceil \frac{M}{p} \right\rceil^2 t_N$, where t_N is the time for one N by N convolution of a p by p image. For specific values of N some savings may be made by combining the computations for residual pieces of the image. Also, this time does not reflect the time required to shift the input image into the array and to shift the result back to the host.

The Warp processor is a 32-bit systolic array architecture currently being developed at Carnegie-Mellon University [2,3]. The current prototype contains ten stages, each containing 4K words of memory. The basic cycle time is 200 nsec, with ALU operations requiring 5 cycles. Kung has shown [2] that when N^2 is less than or equal to the number of Warp stages then each stage may be used to compute a single value of the kernel. Notice that Warp requires programming techniques emphasizing a sequence of operations on a single pixel whereas PIPE operations are defined with respect to an entire image.

Assuming Warp has k stages and $N^2 \leq k$ then Warp takes time

$$T = T_f + T_p$$

where T_f is the time to fill the pipeline and T_p is the time to process all the pixels in an M by M image once the pipe is full. Here, since there is one stage per kernel point ,

$$T_p = M^2 t_c$$

and

$$T_f = 5N^2 t_c$$

where t_c is the basic cycle time (recall that additions and multiplications each take 5 cycles). The value for T_p ignores some minor buffering delays. Hence,

$$T = (M^2 + 5N^2) t_c$$

When large kernels are used and $N^2 > k$, each stage must compute more than a single kernel point. In Warp this can be accomplished using the "wraparound" mode which is similar to PIPE's recursive path. Hence, $P = \left\lceil \frac{N^2}{k} \right\rceil$ points must be computed in each stage. This implies that the time $T \approx P M^2 t_c$. The values in Table 2 reflect this value of T when $N^2 > k$.

Another of the extensions to PIPE under consideration is to expand the image size to 512 by 512. Table 3 shows the values for some processors for a 512 by 512 image. Here we assume that the size increase does not affect the processing time of a stage in PIPE. Note in particular the speed of PIPE relative to the MPP. The difference on large kernel sizes reflects the difficultly in the MPP when the image is much larger than the processor array and when the kernel is large. If the MPP were to be expanded to 256 by 256 the values shown in the table would be the same as those in Table 2.

Machine	3 by 3	15 by 15	45 by 45
General Purpose (100 MFlops)	0.047	1.180	10.6
MPP, 128 by 128 Processor Array	0.005	0.189	5.7
WARP (10 processors)	0.052	1.206	10.64
WARP (20 processors)	0.052	0.625	5.35
PIPE, 3 by 3 NOP, 8 stages	0.017	0.33	1.68
PIPE, 5 by 5 NOP, 8 stages	0.017	0.1	0.63

Table 3. 512 by 512 Image - N by N Kernel (times in seconds)

The results presented in both Table 2 and Table 3 should not be used for absolute comparison purposes for several reasons. First, the convolution algorithms on any processor may be improved. The numbers shown represent the best algorithms we know. Second, the accuracy of the different processors is radically different. Since the MPP is a bit processor, requiring it to produce the 32-bit precision of the Warp processor would multiply the time required by a factor of nearly three. The

numbers given are only for general comparisons.

8. Concluding Remarks

In this paper we have presented several algorithms for computing an N by N convolution on PIPE and shown their execution times compared to other processors such as the MPP and Warp. Our results suggest several things about PIPE. For large convolution kernels it competes favorably with the other processors discussed. For very large kernels, Algorithm 1 requires a large number of buffers; if the PIPE configuration has too few buffers, then Algorithm 2 may be used without a substantial loss in execution time. On 15 by 15 convolutions PIPE can process more than four images per second. If the NOP size is increased to 5 by 5 then ten images per second may be convolved with a 15 by 15 kernel.

It is an open problem as to the optimal time for performing an N by N convolution on PIPE. Both Algorithm 1 and 2 require $O(N^2)$ time. It is easy to see that $O(N^2)$ is the best possible since there are $\frac{N^2}{9}$ 3 by 3 partial convolutions images, only a constant number of stages and at most two partial convolutions may be computed at one time in a single stage. If we do not compute the 3 by 3 partial convolutions, and compute only single values of the kernel at one time, then it is clear that $O(N^2)$ shifts of an accumulator image are required. Practically, the values of N used in image processing applications are relatively small (e.g. between 3 and 80), so the constants involved are as important as the order of magnitude. A tight lower bound on the number of time units required using PIPE has not been determined.

The algorithms presented also demonstrate both the usefulness and potential problems in several features of PIPE. Most readily apparent of these is the shift operation. In the present design, the only way to shift an image requires using the 3 by 3 neighborhood operator. This means that an image can only be shifted one row and one column per unit of time. This presents difficulties in combining results from the 3 by 3 blocks of the general convolution. The algorithms presented here spend most of the time shifting images so that they may be added. Shifting uses a single stage very poorly, with the shift operation dominating the central feature of the stage. Also, shifting has relevance to more than convolution operations. It has a central place in stereo algorithms [6] and

other low-level image computations. If a shift operation for an arbitrary distance could be added to PIPE stages then the execution speeds for these algorithms would be improved greatly.

Besides the difficulty in shifting, there is a second problem that has to do with the accuracy of this convolution. For an arbitrary N by N convolution, the 8 bit pixel value results in an unacceptable loss of accuracy for intermediate results. Clearly, in order to be useful for this operation, PIPE must have its pixel size increased.

In the general convolution algorithms presented, a stage almost never uses both of its NOP's in one stage-time. This stems from requiring the input to both NOP's to come from the same image buffer. Relaxing this restriction would allow shifts of two different images to be accomplished simultaneously on one stage. Similarly, while three images are input to a stage via the normal forward, recursive and backward paths, the present architecture of PIPE requires that they be combined into one image before entering the stage buffer. This greatly reduces the usefulness of the three separate inputs. In the convolution algorithm, the three separate inputs are only used to combine results when some images have been shifted so that they are in register. If the stages allowed the choice of not combining inputs and of using separate buffers as inputs to the neighborhood operators, then the use of the stages could be enhanced and the convolution algorithm improved.

References

1 E.W. Kent, M.O. Shneier, R. Lumia, PIPE, *Journal of Parallel and Distributed Computing* **2**, 1985, 50-78.

2 H.T. Kung, Systolic algorithms for the CMU Warp Processor, *Technical Report CMU-CS-84-158* , Department of Computer Science, Carnegie-Mellon University, 1984.

3 T. Gross, H.T. Kung, M. Lam, J. Webb, Warp as a machine for low-level vision, *IEEE International Conference on Robotics and Automation*, 790-800, 1985.

4 J.L. Potter, Image processing on the Massively Parallel Processor, *IEEE Computer*, **16**, 1983, 62-67.

5 Digital/Analog Design Associates, *PIPE Users Manual*, 1985.

6 W.E.L. Grimson, *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MIT Press, Cambridge, MA, 1981.

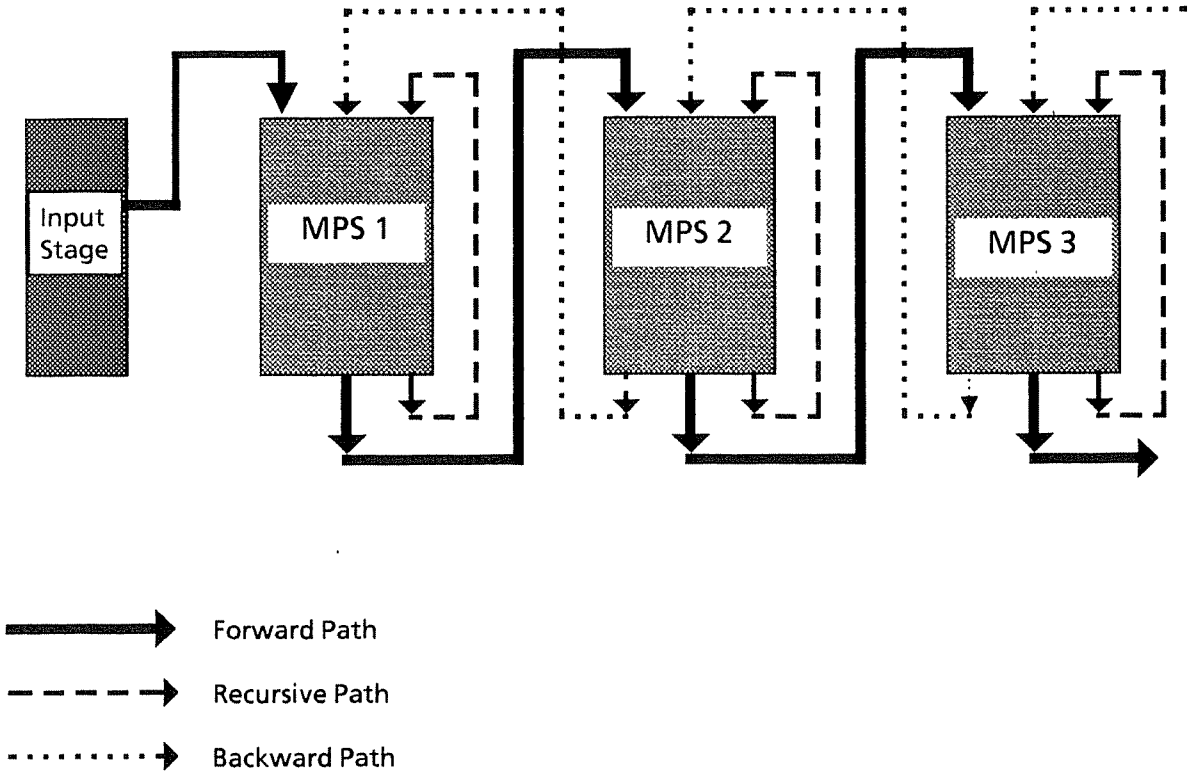


Figure 1. Connection Paths on PIPE.

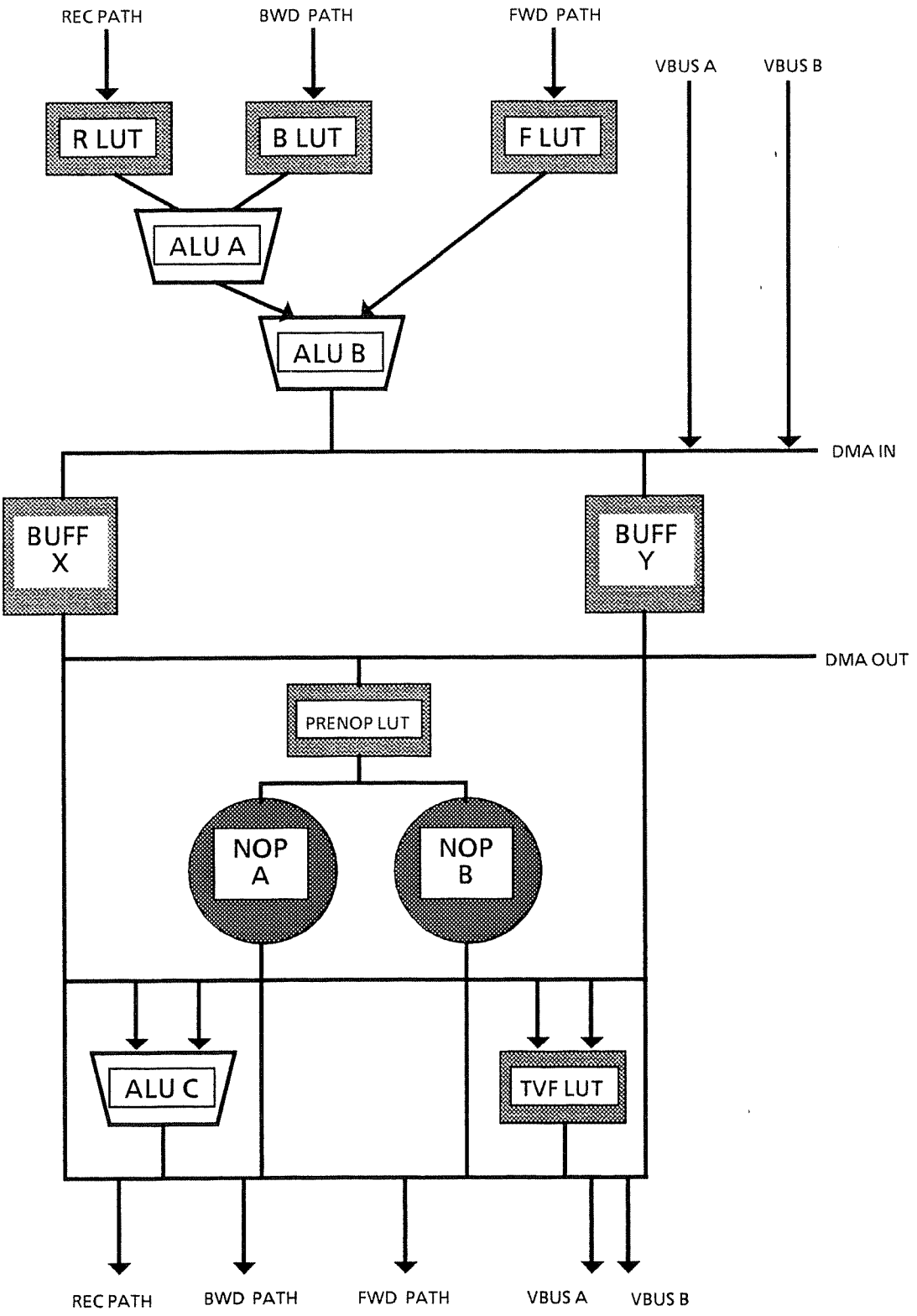
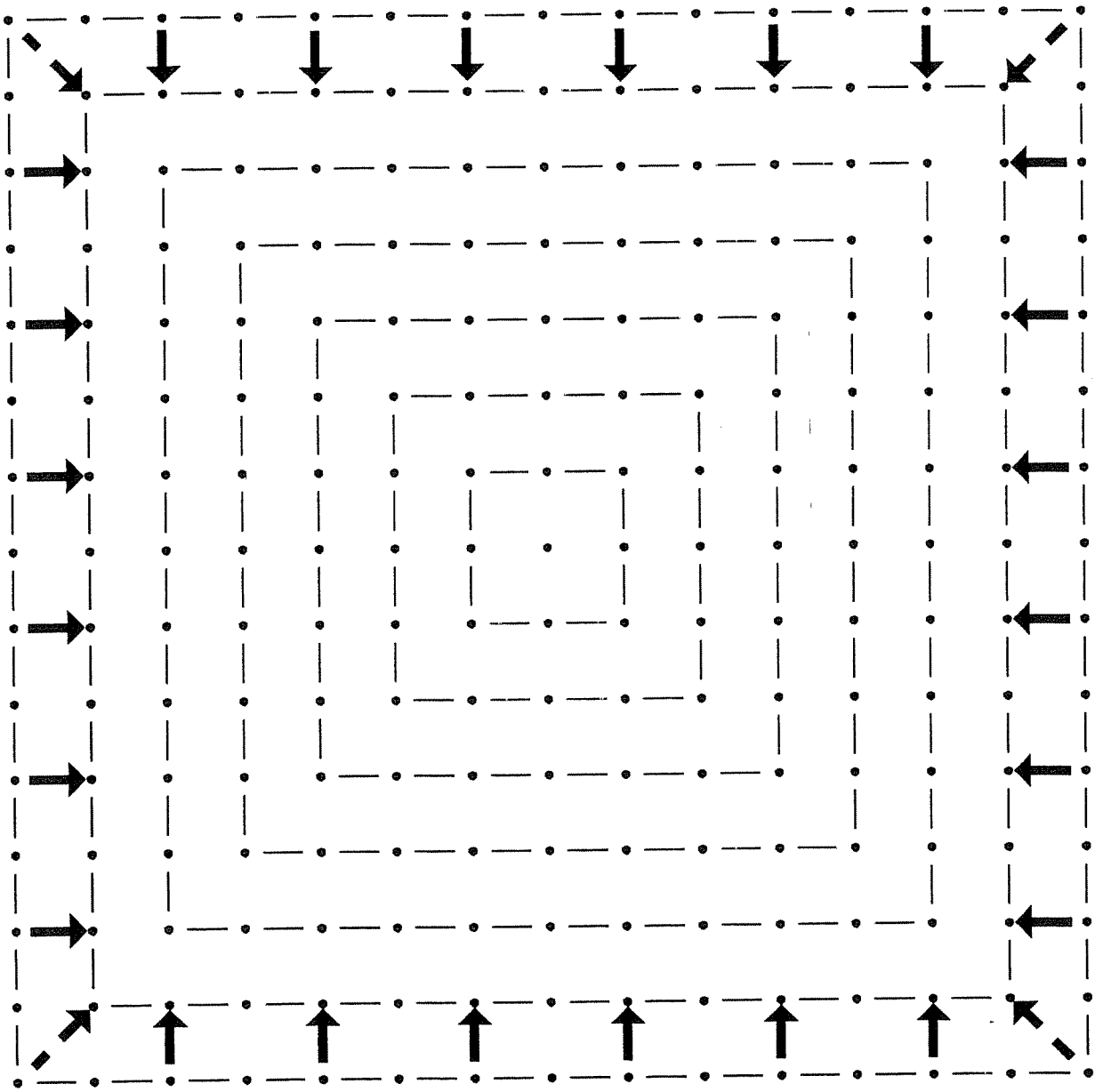


Figure 2, Diagram of Single MPS



KEY:

Partial Result Images In a Concentric Square

3*3 Partial Result Image

Shift Direction of the Concentric Square

Figure 3. Shifting of the Concentric Squares.

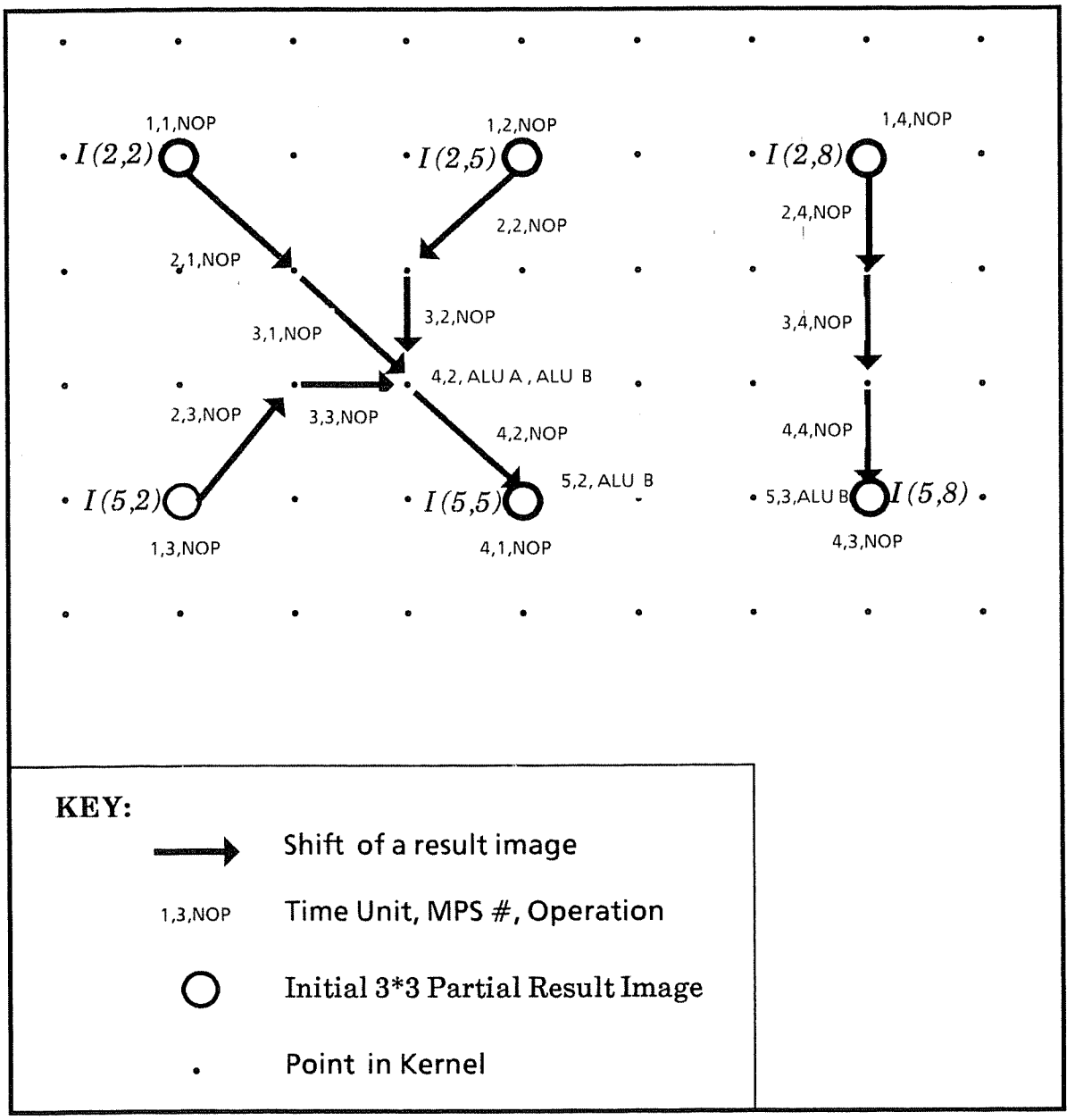


Figure 4. Shifting and Combining in Example 1.

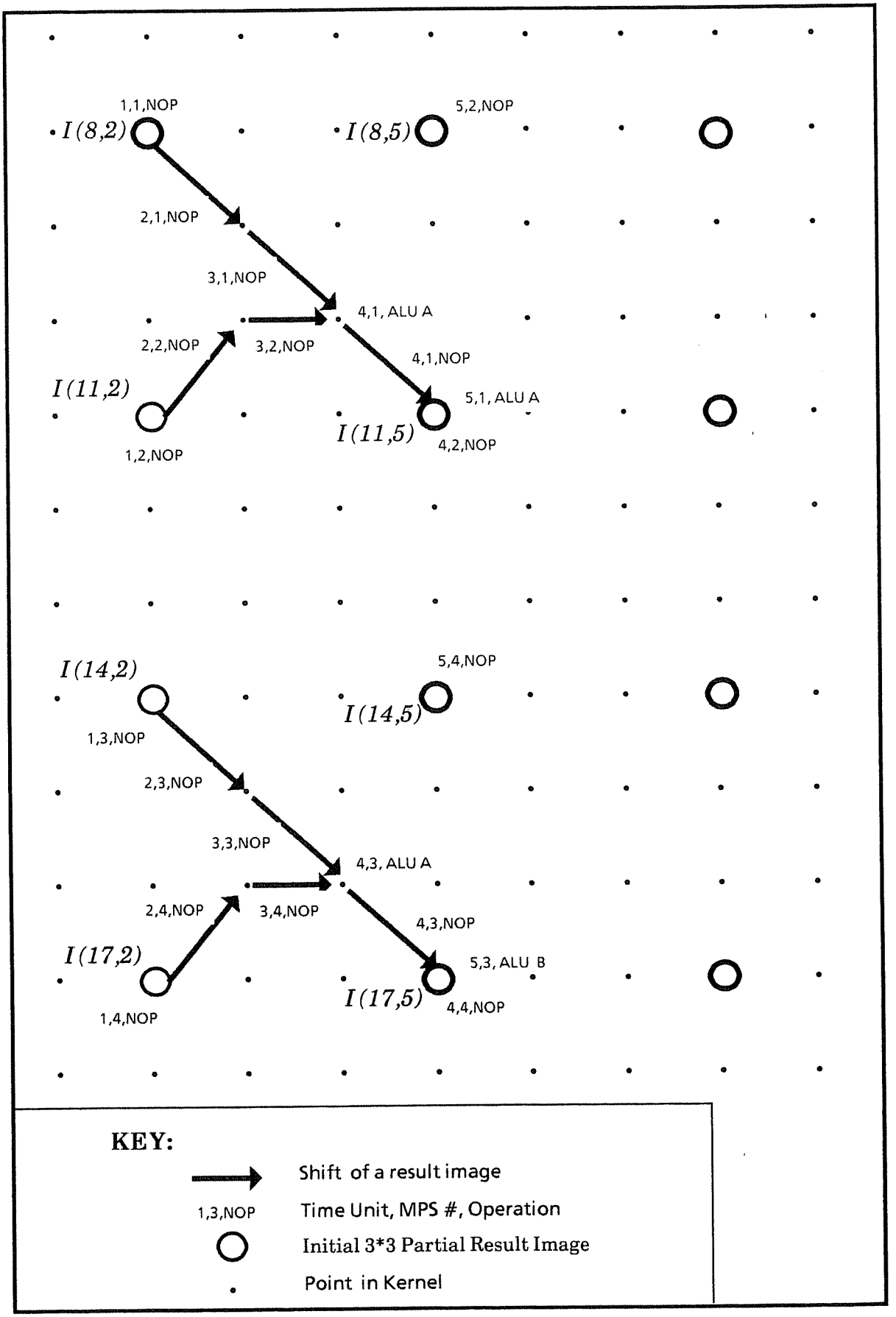


Figure 5. Shifting and Combining in Example 2.

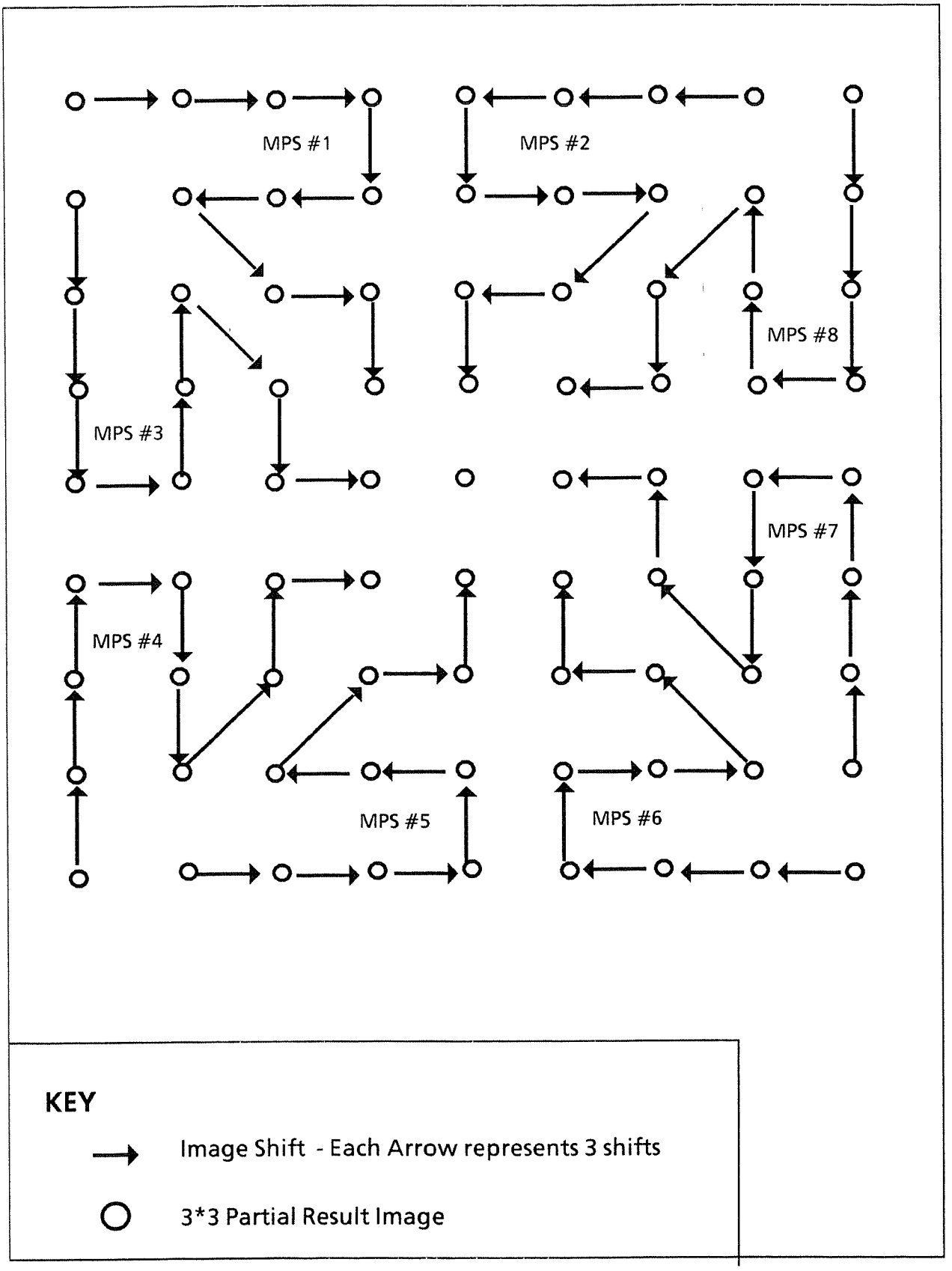


Figure 6. Organization of Shifting For Algorithm 2.