An Algorithm for a Simple Image Convolution on the Titanic Content Addressable Parallel Array Processor

Charles Weems

COINS Technical Report 83-07
June 1982

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

This research was funded by the U.S. Army Research Office Grant DAAG 29-72-G-0046 and by the Defense Advanced Research Projects Agency Contract N00014-82-K-0464.

An Algorithm for a Simple Image Convolution on the Titanic Content Addressable Parallel Array Processor

Charles Weems

June 1982

Abstract

An algorithm is presented for the Titanic Content addressable Parallel Array Processor [1] which will cause it to perform a simple image convolution very quickly. It is further shown that this algorithm can be generalized to perform more complex convolutions with only a moderate reduction in speed.

Background

Our previous work on Conway's Game of Life implemented on a CAM [2] demonstrated that such a device could be effectively used to speed up algorithms which dealt with rectangular grids of cells and small neighborhoods about each of those cells. Because Conway's Game of Life actually involves performing a very simple image convolution, it was soon realized that the technique developed for Life could be applied to more general convolutions. This method was further refined with the Titanic design -- a content addressable parallel array processor.

Basic Technique

One simple form of convolution involves each cell on a rectangular grid examining its immediate neighborhood and then updating its own contents based upon some function of that neighborhood. The update must, of course, be performed after all cells have finished examining their neighborhoods. On a parallel array processor this examination can be performed simultaneously by all of the cells on the grid, as can the update operation. Thus the algorithm for the convolution can be described as the actions of a single cell with the understanding that each action is performed simultaneously by all of the cells.

There are two different ways of approaching the problem of examining the neighborhood. The one that first comes to mind is that each cell "looks" at each cell in its neighborhood, gathering what information it needs to perform an update. In practice this involves moving data from each cell in the neighborhood into the "central" cell where some function is then applied to it and the result stored for the update phase of the convolution. The problem with this is that the data must often pass through other cells before it reaches the central cell. For example, when the neighborhood is 7x7 cells, data from the outer ring of cells must pass through at least two other cells before reaching the center cell. Because movement of data takes time, this "passing through" is rather inefficient. The solution is to have the data stored in the intermediate cells on its way to the center, thus taking advantage of the fact that those

cells will also need to know the values in order to compute the function of their neighborhoods. Although this will work, the algorithm becomes rather messy since we must now consider the actions of several cells at once and how these relate to each other. It also becomes a complex problem to determine an optimal set of data collection paths as the neighborhood's diameter varies.

It turns out that the other approach to examining the neighborhood greatly simplifies these problems. This approach takes the opposite view of the collection process. Instead of each cell collecting all of the data from its neighborhood, each cell distributes its own data to every cell in the neighborhood. Because every other cell is also doing this, the end result is that the central cell (and hence all cells) gets the data it needs from all of the cells in the neighborhood. The problem of establishing an optimal distribution path is trivial for a square array of odd diameter: It is simply a rectangular spiral out from the center cell. For even diameter square neighborhoods the problem is only slightly more difficult because the center cell is actually half of a cell width off center in two diections. In this case it is simply required that the appropriate choice of initial direction and of clockwise or counter clockwise spiral be made to select the optimal path. The only other point that requires mentioning is that, because this is a distribution process rather than a collection process, the funtion mask for the convolution must be mirrored across the central cell. For example, when

the cell's value is being stored in its north neighbor, the function applied to that value is the south neighbor function. The reason for this can be seen when it is realized that the central cell is actually the south neighbor of the cell to its north. The mirroring of the convolution function mask is actually quite easy to accomplish: we simply step through the mask in exactly the opposite direction that the distribution path takes.

Let's look at an example: A simple convolution for smoothing isolated cells of noise out of an image. We will use a 3x3 convolution mask in which the cell accumulates the sum of its neighbor's values, weighted inversely with distance away from the center. The sum will then be normalized. Define the mask to be an array $M^{i\ j}$:

M =

i \ j	0	1	2
0	1	2	1
1	2	4	2
2	1	2	1

Where $M_{1,1}$ is the central cell. Then the following algorithm will perform the convolution (north is up, west is to the left, etc.):

```
i := 1
j := 1
sum := value *M i j
move value north
i := i+1
sum := sum + value * Mij
move value east
j := j+1
```

```
sum := sum + value * M_ij
move value south
i := i-1
sum := sum + value * M_ij
move value south
i := i-1
sum := sum + value * M_ij
move value west
j :+ j-1
sum := sum + value * M_ij
move value west
j := j-1
sum := sum - value * M_ij
move value north
i := i+1
sum := sum + value * M_ij
value := sum * normalizing factor
```

It should be noted that the time required to perform a convolution using the parallel processor is independent of the size of the image and only dependent upon the area of the convolution mask. Since the Titanic does cell level arithmetic bit-serially, the size of the data values also affects the speed of the algorithm.

Convolution on Titanic

The following algorithm gives the list of instructions required to make Titanic perform the convolution given in the above example. In this case we have taken advantage of special characteristics in the mask values to help direct the shift and add process of the required multiply operations. The algorithm is written for 8 bit data values and runs in an estimated time of 98 microseconds.

```
(* Initialize *)

A := 1!
Empty_Edges

(* Send to North *)
```

```
Z := 0
For Bit := 2 to 9 do
   X := M(Bit)
   Shift_X_North
   M(Bit+10) := X
   Y := M(Bit -1)
   Y := X+Y
   M(Bit - 1) := Y
End For
X := 0
For Bit := 9 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to Northwest *)

Z := 0
For Bit := 12 to 19 do
   X := M(Bit)
   Shift_X_West
   M(Bit) := X
   Y := M(Bit - 12)
   Y := X+Y
   M(Bit - 12) := Y
End For
X := 0
For Bit := 8 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to West *)

Z := 0
For Bit := 12 to 19 do
   X := M(Bit)
   Shift_X_South
   M(Bit) := X
   Y := M(Bit - 11)
   Y := X+Y
   M(Bit - 11) := Y
End For
X := 0
For Bit := 9 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to Southwest *)

Z := 0
```

```
For Bit := 12 to 19 do
   X := M(Bit)
   Shift_X_South
   M(Bit) := X
   Y := M(Bit - 12)
   Y := X+Y
   M(Bit - 12) := Y
End For
X := 0
For Bit := 8 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to South *)

Z := 0
For Bit := 12 to 19 do
   X := M(Bit)
   Shift_X_East
   M(Bit) := X
   Y := M(Bit - 11)
   Y := X+Y
   M(Bit - 11) := Y
End For
X := 0
For Bit := 9 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to Southeast *)

Z := 0
For Bit := 12 to 19 do
   X := M(Bit)
   Shift_X_East
   M(Bit) := X
   Y := M(Bit - 12)
   Y := X+Y
   M(Bit - 12) := Y
End For
X := 0
For Bit := 8 to 11 do
   Y := M(Bit)
   Y := X+Y
   M(Bit) := Y
End For

(* Send to East *)

Z := 0
For Bit := 12 to 19 do
```

```
      X := M(Bit)
      Shift_X_North
      M(Bit) := X
      Y := M(Bit - 11)
      Y := X+Y
      M(Bit - 11) := Y
   End For
   X := 0
   For Bit := 9 to 11 do
      Y := M(Bit)
      Y := X+Y
      M(Bit) := Y
   End For

   (* Send to Northeast *)

   Z := 0
   For Bit := 12 to 19 do
      X := M(Bit)
      Shift_X_North
      M(Bit) := X
      Y := M(Bit - 12)
      Y := X+Y
      M(Bit - 12) := Y
   End For
   X := 0
   For Bit := 8 to 11 do
      Y := M(Bit)
      Y := X+Y
      M(Bit) := Y
   End For

   (* Scale Result *)

   For Bit := 2 to 11 do
      X := M(Bit)
      M(Bit - 2) := X
   End For
   M(10) := 0
   M(11) := 0


   980 CAM Operations
   98 uS per Convolution

   340 Conv / Frame Time
   10204 Conv / Sec
```

Convolutions with more general and/or larger masks will take longer. A very rough worst case estimate of the time required for such convolutions can be obtained from the

formula:

$$T = P(.8N+.2M+.1) + .3M(N^2P+N+1)$$

where T = time in microseconds
N = number of bits in a pixel
M = number of bits in a mask value
P = number of pixels in the mask area

This is a worst case time which assumes that all of the bits in all of the mask values are ones (since this gives the slowest multiply speed). Under normal circumstances, T will be about half of the value obtained from the formula. This also assumes a totally general square mask where the values can change. If constants are to be used for the mask values, a significant speed increase can be obtained by optimizing the multiples for those values. Thus, for example, a convolution on 16 bit values with 8 bit mask values could be applied over at most a 7x7 mask in one video frame time with a worst case situation. For normal situations, it should be possible to convolve a 10x10 area. Given constant mask values, and depending upon the amount of optimization possible, even a 25x25 mask could be done in one video frame time.

As a final note, this method is not restricted to square masks and in fact should be readily generalizeable to any mask shape. All that is required for this is an algorithm for efficiently shifting the center cell's value so that it covers the mask area. Thus it should be possible to easily adapt it to such mask shapes as annuli and disjoint areas.

Conclusion

A method has been shown which can be used to program the Titanic content addressable parallel array processor to perform image convolutions simply and efficiently. Such a program, for a simple convolution, was shown which operates in ninety-eight microseconds. The time of the algorithm is independent of the size of the image and depends only upon the size of the mask and, for bit serial processing, upon the number of bits in the pixel and mask values. A formula was given for a worst case time estimate and a factor for estimating normal case time from this was discussed. It was also noted that the method could be applied to masks of other than square shapes.

References

1. C. Weems, S. Levitan, and C. Foster, "Titanic: A VLSI-Based Content Addressable Parallel Array Processor, Proceedings of IEEE International Conference on Circuits and Computers, September 1982, pp.236-239.

2. C. Weems, "Life is a CAM-Array Old Chum", unpublished paper, January 1980.