SP-CNN: A Scalable and Programmable CNN-based Accelerator

Dilan Manatunga, Hyesoon Kim, Saibal Mukhopadhyay
College of Computing
Georgia Institute of Technology
Atlanta, Ga, USA, 30332

Abstract: *Specialized accelerators have become prevalent in many mobile computing platforms for their ability to perform certain tasks, such as image processing, at a lower power cost than a generalized CPU or GPU. In this paper, we focus on using Cellular Neural Networks (CNN) as a specialized accelerator. CNN is a neural computing paradigm that is well suited for image processing applications. However, hardware implementations were originally developed only to handle relatively small image sizes. In this paper, we propose SP-CNN, an architecture as well as a multiplexing algorithm that provides scalability to the CNN applications. We demonstrate the proposed multiplexing algorithms over a set of six image processing benchmarks, as well as present a performance and power analysis of SP-CNN.*

Keywords: cnn; cellular neural networks; accelerators; image processing; neuromorphic computing

Introduction

With the rapid growth of mobile computing, the need to design energy-efficient and special purpose accelerators is high. For example, in Intel's Atom Lexington SOC and Bay Trail SOC **[1]**, more than 25% of the chip area is used for video or image-related processors. Excluding memory or storage, the majority of chip area is dedicated to special accelerators for image/graphics and security related engines. Furthermore, considering the growing need for processing sensor data like camera input, the demand for special purpose accelerators will continue to increase.

Recently, brain-inspired computing systems, especially neural network based systems, are getting high attention as special accelerators. Qualcomm's NPU systems or IBM's TrueNorth are two examples. These processors promise a high energy efficiency. Similar to these neural network processors, which are based on artificial neural networks, cellular neural network (CNN) processors **[2]** have been studied widely for image processing applications and are another option for neural network based accelerators.

One challenge of current CNN research is the scalability of CNN hardware and algorithms. So far, all CNN algorithms are developed under the assumptions that the number of cells is equal to the image size. Unfortunately, current CNN designs have relatively small array sizes, such as 80x60 **[3]**, and have not scaled to the multiple megapixel images present today. Furthermore, handling large images is not a trivial task. Using simple naïve multiplexing algorithms or simple stencil operations will quickly saturate memory bandwidth.

In this paper, we propose a mechanism to operate CNN algorithms on large images while still using relatively small arrays. Our contributions are as follows:

1. We propose a programmable CNN architecture, called SP-CNN, which is an accelerator to be connected with a host processor. SP-CNN has multiple CNN array processors, a scheduler, and a prefetcher.
2. We propose an algorithm to be used on our proposed architecture that can handle any input image size without causing heavy memory contention. We also demonstrate that the CNN paradigm can perform real time image processing on modern image sizes.

CNN Background

The CNN was introduced by Chua and Yang **[2]** and is a type of neural network that consists of a homogenous 2D array of cells, in which each cell communicates with only a fixed set of neighbor cells. The connections between cells are local, thus eliminating the need for long distance interconnects. Figure 1 shows the local connections for an example 3x3 network.

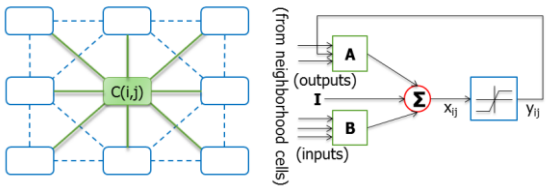

Figure 1. Left: CNN cell connections and neighbor cells of the cell at C(i, j). Right: Operation of CNN cell.

Each cell in the array operates based on a state and output equation. These equations can be specified by the application, but most applications tend to use the standard CNN state and output equations (Figure 2). A CNN application is specified by the CNN gene. The gene specifies the *A* and *B* scaling factors that are applied to the neighbor cell's output and inputs, state equation threshold, *I*, and the initial cell state, and boundary conditions.

$$x_{ij}(t+1) = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l) * y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l) * u_{kl}(t) + I \quad (1)$$

$$y_{ij}(t+1) = \frac{1}{2}\left[|x_{ij}(t)+1| - |x_{ij}(t)-1|\right] \quad (2)$$

Figure 2. Standard CNN state (1) and output (2) equations.

```
while change do
    change = parallel-for-i,j of Eq. (1)
    parallel-for-i,j of Eq. (2)
    t = vt += 1
end while
```

Algorithm 1: Traditional CNN Algorithm

```
while change do
    change = false
    for p in partitions do
        loadStateAndInputToCNN(p)
        for n = 0; n < interval; n++, t++ do
            change |= parallel-for-i,j on p with Eq. (1)
            parallel-for-i,j on p with Eq. (2)
        end for
        saveToNextStateFromCNN(p)
    end for
    swapStateAndNextStatePointers(p)
    iter += 1, vt += interval
end while
```

Algorithm 2: SP-CNN Algorithm (Multiplexing CNN)

Figure 3. Pseudo-code of CNN and SP-CNN algorithms

Programmable Multiplexing Algorithm

To enable the efficient operation of CNN applications on large images, we propose a multiplexing algorithm based on how stencil computation is performed on GPUs. In GPU stencil computation, the input image is spatially partitioned, and each GPU compute unit performs the stencil operation on an image partition. If we applied this approach naively, this means we would run each image partition on a CNN array using the traditional CNN algorithm (Algorithm 1).

As one might expect, this *naïve* multiplexing approach leads to errors for many CNN applications. These errors result from the fact that cells on the partition boundary do not correctly operate since they never see updated values for the neighboring cells in different partitions. Figure 4 shows an example of the incorrect of *naïve* multiplexing for the Hole-Filling CNN application.

The easy solution to this issue would be to employ what we call *ideal* multiplexing. In ideal multiplexing, we only run the partition on the CNN array for 1 CNN time unit, which is equivalent to one state/output computation. We do this for every partition, and then repeat the process. As one would expect, this method will always produce the correct output. Figure 5 illustrates the general process for *ideal* multiplexing.

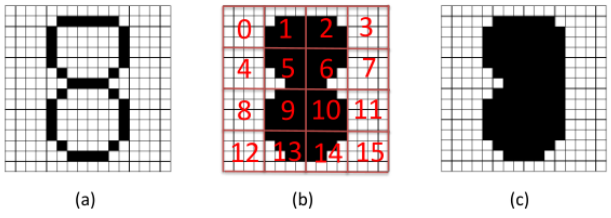

Figure 4. Example of Hole-Filling with Naïve Multiplexing (a) input (b) correct output (c) naïve multiplexing outcome

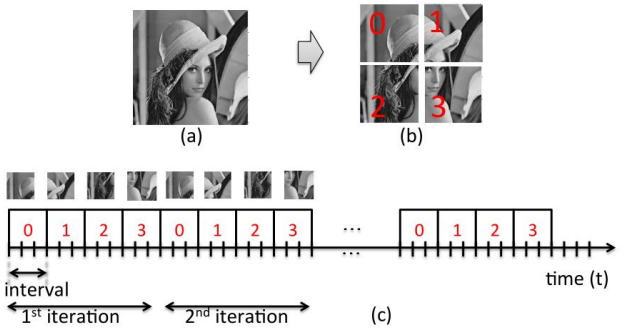

Figure 5. Multiplexing example (a) original image (b) partitioned images (c) illustration of multiplexing operation.

However, ideal multiplexing comes with a cost of high memory contention. Specifically, processing the partition on the CNN array for 1 CNN time unit takes relatively little time. On the other hand, transferring the image partition, even one as small as 128x128, takes a comparatively much larger time. This leads to heavy memory contention from the continuous transfer of image partitions to and from the CNN compute units.

SP-CNN Multiplexing Algorithm

The key insight to the SP-CNN multiplexing algorithm is that many CNN applications are robust to small errors within inputs and/or computation. Specifically, we realized that while it is important for the boundary cells to see updated values from neighboring cells in different partitions, for many CNN applications, the correct output could still be achieved even if the partition boundary cells did not have the correct neighbor values for a specified interval of CNN time units.

Algorithm 2 illustrates the pseudo-code for the SP-CNN multiplexing algorithm. In essence, we take the *ideal* multiplexing approach, but instead of running one state/output computation, we run multiple iterations of the computations. We term this period as the *interval*. Once every partitioned has been processed for one interval, the period which we call an *iteration*, we then repeat the process, except now boundary cells will see updated values for their neighboring cells in neighboring partitions. These iterations are repeated until we converge to a steady-state. Again, Figure 5 illustrates this process. Ideal multiplexing then becomes a special case of SP-CNN multiplexing, where the interval time equals 1 state/output computation.

Early-Finish: After some preliminary analysis, we saw that certain partitions would converge to a steady-state before their interval was over. This does not necessarily mean the entire image had converged, but it does indicate that for the rest of the interval, no further useful computation would occur. In order to avoid this, we introduce the concept of *Early-Finish*. With *Early-Finish*, a partition stops processing on the CNN array when it has either hit the interval or when it has converged to a steady state.

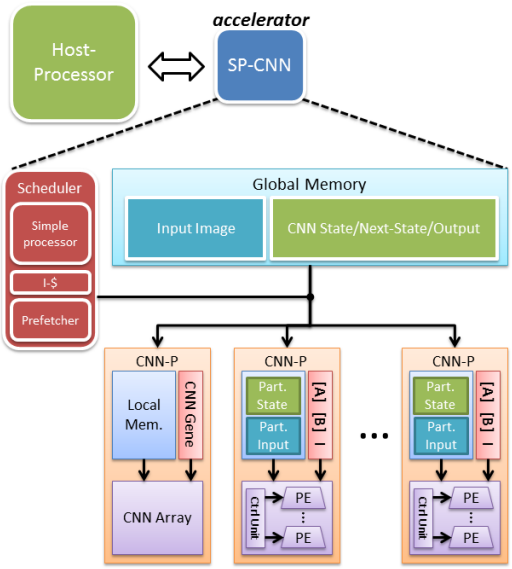

Figure 6. Example SP-CNN Architecture

SP-CNN Architecture

The SP-CNN architecture was influenced by NVIDIA'S G80 architecture **[3]**. Figure 6 shows an overview of the proposed architecture. SP-CNN serves as an accelerator to a host processor. The SP-CNN accelerator has a global memory, a scheduler, and possibly several CNN-Ps. A CNN-P is a CNN compute unit which has local memory, the CNN gene, and a CNN compute array, where the array is composed of a set of processing elements (PE).

Multiple CNN-Ps: As stated previously, the SP-CNN accelerator may have one or more CNN-Ps. The benefit of multiple CNN-Ps is that multiple partitions can be executed concurrently. Furthermore, multiple CNN-Ps provide a better ability to hide memory latency. However, too many CNN-Ps can increase memory bandwidth contention, leading to less than linear scaling in performance. Furthermore, a design tradeoff must be made between having many small CNN arrays versus a few large arrays.

Evaluation Methods

Table 1 describes our evaluated benchmarks, which are commonly used in CNN-based image processing applications. We use 10 test input images, with each image stored at a resolution of 1024x1024 or 2048x2048. We chose these dimensions since the number of pixels for each dimension roughly corresponds to the number of pixels in 720p and 2048x1536 respectively.

We developed a functional simulator of our SP-CNN architecture. The default SP-CNN parameters are 1 CNN-P unit with a CNN array size of 128x128 and an interval time of 128 CNN time units. The functional simulator returns the convergence time of an application in CNN time units, where a single unit corresponds to one state/output computation. Furthermore, transfers of partitions to and from memory are assumed to occur instantaneously.

Table 1. Evaluated Benchmarks

Benchmarks	Abbreviation
Corner Detection	Corner-Detect
Edge Detection	Edge-Detect
Connected Component	ConnComp
Hole Filling	Hole-Fill
Rotation Detector	Rot-Detect
Shadow Creator	Shadow

To observe the effects of memory transfers, we also developed a timing simulator which used DRAMSim2 as a memory simulator **[5]**, with CNN-P timing parameters based on the VAE architecture **[4]**. We used a 2 GB global DDR3 memory specified by the DRAMSim2 configuration file *DDR3_micron_16M_8B_x8_sg15.ini*.

Virtual Time vs. Total Time: Directly comparing the total time between the Ideal CNN (where the CNN array size equals the image size) and SP-CNN is not very informative since the SP-CNN's time will be at least scaled by the image to CNN array size ratio. Instead, we introduce the concept of virtual time. Virtual time represents how many CNN time units a partition has actually processed on the CNN compute unit. This means for SP-CNN, if $|P|$ is the number of partitions, then after one iteration, even though *interval* * $|P|$ total time units has passed, only *interval* virtual time units have passed, since each partition has only processed for interval units. For algorithms 1 and 2, *t* and *vt* represent the total time passed and virtual time passed.

Results

Figure 8 compares the average virtual convergence time between the Ideal-CNN and our proposed SP-CNN mechanism. As seen in the figure, SP-CNN is competitive with the ideal CNN for all benchmarks.

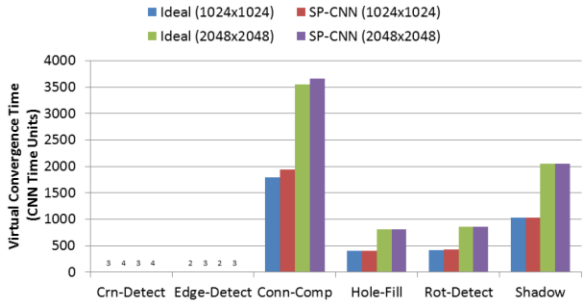

Figure 8. Avg. Virtual Conv. Time of Ideal CNN vs. SP-CNN

Interval Period Effects: As stated previously, one of the important contributions of this work is showing that certain CNN applications are robust enough to use interval values larger than 1 during multiplexing execution. Choosing an effective interval time is important in limiting the convergence time of an application. Too small of an interval time may lower the virtual convergence time, but it

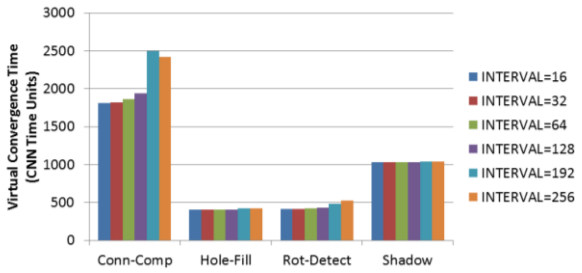

Figure 9. Effect of Interval Time on Virtual Conv. Time

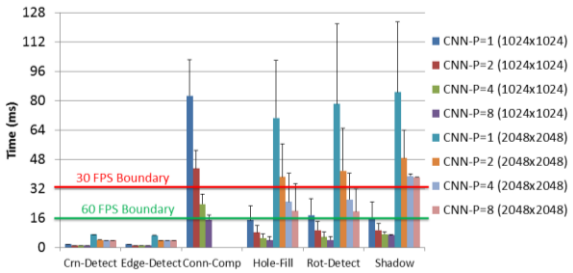

Figure 10. Average Execution Time of SP-CNN (Bar represents max execution time seen)

also increases the overhead of data transfers. On the other hand, too large of an interval time can increase the virtual and total convergence times.

Figure 9 shows the average virtual convergence time results as interval time increases, while Figure 10 show the average convergence time results when varying interval time. We exclude Corner-Detection and Edge-Detection from our results, since these two applications were local-type CNN applications and converge much sooner than the interval time limit. As expected, when increasing the interval time, the virtual convergence time tends to increase. This is best seen with Connected Component.

Timing Results: Figure 10 shows the timing results with SP-CNN. With the 1024x1024 images, SP-CNN can perform below the 60 FPS boundary for most applications. In the case of 2048x2048 images, SP-CNN meets the 30 FPS boundary for most applications when the number of CNN-Ps is equal to 4 or 8.

Multiple CNN-Ps: Figure 11 illustrates the effects of scaling the number of CNN-Ps. As seen, none of the applications show linear scaling when increasing the number of CNN-Ps. The reason SP-CNN does not scale is because memory contention becomes a dominating factor. This can be seen in Figure 12, which illustrates the percentage of time a partition execution spends communicating to memory. As seen for the benchmarks that show sub-linear scaling, memory communication accounts for over 50% of partition execution, and rapidly grows as the number of units increase.

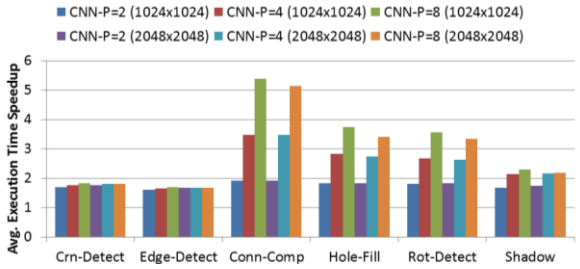

Figure 11. Average Execution Time of SP-CNN (Bar represents max execution time seen)

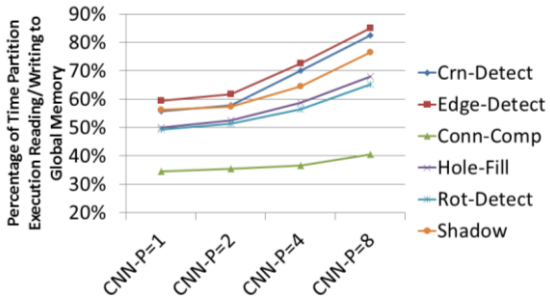

Figure 12. Avg. Percentage of Time Partition Execution Spends in Memory Transfers

Conclusion

In this paper, we proposed a scalable and programmable CNN, SP-CNN, to overcome the input scalability problem with traditional CNN paradigm. SP-CNN can handle any resolution image with a small fixed dimension CNN array. Our results show that the proposed multiplexing algorithms can successfully scale robust CNN applications while using smaller hardware resources. We also proposed a microarchitecture and demonstrated that this architecture can provide real-time image processing.

References

1. WCCFTECH, "Intel aims for mobile space with the 32nm atom z2420 lexington soc and 22nm bay trail soc," http://wccftech.com/intel-aims-mobile-space-32nm-atom-z2420/, Intel.
2. L. Chua and L. Yang, "Cellular neural networks: applications," Circuits and Systems, IEEE Transactions on, vol. 35, no. 10, pp. 1273–1290, 1988.
3. NVIDIA, "Geforce 8800 graphics processors," http://www.nvidia.com/page/geforce_8800.html.
4. S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo, "24-GOPS 4.5- mm^2 digital cellular neural network for rapid visual attention in an object-recognition soc," Neural Networks, IEEE Transactions on, vol. 22, no. 1, pp. 64–73, 2011.
5. P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," Computer Architecture Letters, vol. 10, no. 1, pp. 16 –19, jan.-june 2011.