GPU Implementation of Asynchronous Discrete-Time Cellular Neural Networks

Kenichiro Tanaka and Yuichi Tanji†
Department of Electronics and Information Engineering,
Kagawa University
2217-20 Hayashi-cho, Takamatsu 761-0396, Japan
Email: †tanji@eng.kagawa-u.ac.jp

Abstract– GPU implementation of asynchronous type of discrete-time cellular neural networks for image processing is presented. To accelerate the operation, the parallel computation scheme for updating the cells of networks is provided. Furthermore, the techniques that utilize shared memory of GPU are presented in order to speed up further. In illustrative examples, we compare the performance of GPU implementation with CPU and confirm its efficiency.

1. Introduction

Discrete-time cellular neural networks (DT-CNNs) [1], [3]-[6] are a digital version of cellular neural networks (CNNs) [2], where the output of DT-CNNs is obtained by step or quantization function, and the dynamics is represented by a set of difference equations which are obtained via forward Euler method for solving the state equations of CNNs with a unit time step. CNNs are constructed by large scale of analog VLSI. Although DT-CNNs do not exceed CNNs functionally, DT-CNNs can be constructed via hardware description language [3], [6]. This means that DC-CNNs are more reliable than CNNs. The relation between morphology and DT-CNNs was suggested [4], thus, the universal machine with DT-CNN processor can be used for actual applications such as cancer detection.

In [8], it was suggested that there are two types of DT-CNNs depending on the update rules. The cells are synchronously updated for the synchronous DT-CNNs. On the other hand, for the asynchronous DT-CNNs, the cells are asynchronously updated. The difference of update is explained by the nonlinear relaxation methods which are numerical methods for solving a set of nonlinear equations [8]. The synchronous DT-CNNs are corresponding to one step Gauss Jacobi Newton (GJN) method for solution of equilibrium points for the state equations of CNNs, and the asynchronous DT-CNNs are to one step Gauss Seidel Newton (GSN) method. It is known that the GSN method is more robust than the GJN method. Therefore, applicability of asynchronous DT-CNNs is wider than synchronous DT-CNNs.

In this paper, the GPU implementation of asynchronous DT-CNNs is presented. GPU was a graphic processor to display images on monitor in real time. Recently, it is used for general purpose of computations with features of many core processors. The structures are similar to the DT-CNN processors [3], [6] which have many processing units for updating the cells. Hence, we implement the asynchronous DT-CNNs on GPU. First, the parallel updating scheme of cells is presented in order to accelerate the operation. Next, we consider using the shared memory which is an internal memory of GPU.

In the examples, it is demonstrated that the GPU implementation is much faster than CPU.

2. DT-CNN

2.1. Synchronous and Asynchronous DT-CNNs

DT-CNN is a sparse connected neural network composed of 2-D array of $M \times N$ cells. For a cell $C(i,j)$ ($i = 1,\ldots, M$, $j = 1,\ldots, N$) and the neighborhood $N_r(i,j)$, the dynamic of DT-CNN is written by

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

where $x_{ij}(n)$ is the internal state of the cell $C(i,j)$, $u_{ij} \in [-\xi, \xi]$ is the input, y_{ij} is the output, and T_{ij} is the threshold value. $A(i,j;k,l)$ and $B(i,j;k,l)$ are the connection weights between $C(i,j)$ and $C(k,l)$ which are related to the outputs and inputs, respectively, and these are called A and B templates.

The output function $f(x_{ij}(n))$ is defined as

$$f(x_{ij}(n)) = \begin{cases} 1 & (x_{ij}(n) \geq \xi) \\ g(x_{ij}(n)) & (-\xi < x_{ij}(n) < \xi), \\ -1 & (x_{ij}(n) \leq \xi) \end{cases} \quad (2)$$

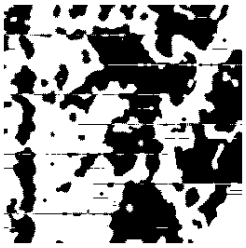
where $g(x_{ij})$ is m-level uniform quantization function.

DT-CNNs are classified into synchronous and asynchronous types. For the synchronous DT-CNNs, each cell is synchronously updated following (1) and (2). On the other hand, update of cells is asynchronously carried out for asynchronous DT-CNNs. Hence, results of

synchronous CNNs are not always identical to ones of asynchronous DT-CNNs. For example, we can obtain the conversion images shown in Fig. 1 using the halftoning templates [5]. The halftone image can be obtained by the asynchronous DT-CNN. However, the result obtained by the synchronous DT-CNN is not adequate, because the conversion image is not in good quality as halftone image. For DT-CNNs or CNNs, the networks are designed so that they reach the equilibrium points, and the output of each cell gives the conversion image. The reason why the result obtained by the synchronous DT-CNN does not provide a good halftone image is that the network does not converge to the equilibrium points. In the next subsection, we clarify the difference between synchronous and asynchronous DT-CNNs in convergence.

(a)

(b)



(c)

Figure 1: Halftone image obtained by DT-CNN. (a) Original Image. (b) Conversion Image by the asynchronous DT-CNN. (c) Conversion Image by the synchronous DT-CNN.

2.2. Nonlinear Relaxation Method

DT-CNNs are originated from CNNs. The dynamics of CNN is written by

$$\frac{dx_{ij}}{dt} = -x_{+ij} + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)y_{kl}(n) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)u_{kl} + T_{ij} \quad (3)$$

Hence, DT-CNNs are understood as being an approximation by the forward Euler method to (3). However, we cannot explain the difference between synchronous and asynchronous types. The reference [8] shows that DT-CNNs are best understood to be one step nonlinear relaxation methods for finding the equilibrium points of (3). Then, the one step GJN iteration is written by

$$x_{ij}^{m+1} = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)f(x_{kl}^{m}) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)u_{kl} + T_{ij}, \quad (4)$$

whereas the one step GSN iteration is written by

$$x_{ij}^{m+1} = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)f(P(x_{kl}^{m}))y_{kl} + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)u_{kl} + T_{ij}, \quad (5)$$

where

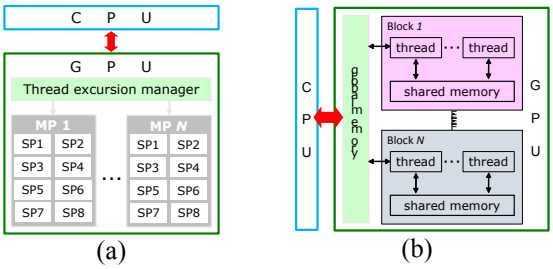
$$P(x_{kl}^{m}) = \begin{cases} x_{kl}^{m+1} & if\ x_{kl}^{m+1} \text{ is updated} \\ x_{kl}^{m} & \text{otherwise} \end{cases}. \quad (6)$$

In (4), the updated internal state x_{ij} depends on the past value, thus (4) is identical to (1), which implies that the GJN method is identical to the synchronous DT-CNNs. On the other hand, the internal state x_{ij} is asynchronously updated as (6). Hence, (5) is an asynchronous DT-CNN. We can adopt various update rules for the asynchronous DT-CNNs. In our GPU implementation, the cells are updated in parallel to accelerate the operations.

3. GPU Implementation

The structure of GPU is shown in Fig. 2(a). The multi processors MP in GPU include some streaming processors SP. These processors are managed by the concept of thread, block, and grid. Block consists of many threads which are executed in parallel. Grid is composed of all blocks. Programming for GPU is made via CUDA which is an integrated development environment provided by NVIDA. Programmers need not to be conscious of multiprocessors and only have to handle the threads.

Figure 3 shows the concept of update of cells. The cells painted with yellow are dummy sells which have only output with zero. The dummy cells are used to update the cells placed at edge of image. We assume that the size of A and B templates is 3×3. Then, the original image is divided into 3×3 subimage. In Fig. 3, the cells with the same color do not have dependency each other, there happens no collision in global memory when they are updated. Namely, the update as Fig. 3 gives an asynchronous DT-CNN. Then, the whole image is processed by nine blocks, which implies that the internal states ad outputs are read and written nine times from and/or to global memory.



(a) (b)

Figure 2: Structure of GPU and CUDA. (a) Relation between CPU and GPU. (b) Memory structure of GPU in CUDA.

To accelerate the operation, we should minimize reading from global memory. Since shared memory is faster than global memory, we make use of shared memory. As shown in Fig. 4, we take the every third cell and make a block. Then, the nx/3 cells corresponding to a block are simultaneously updated. In this case, the internal states and outputs in the first three lines are read from global memory into shared memory. We can take advantage of the contents of shared memory during three updates in horizontal direction, which means that the internal state and output are read and written three times from global image to process the whole image.

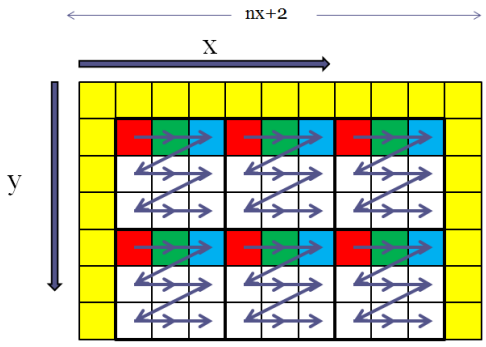


Figure 3: GPU implementation of the asynchronous DT-CNN, where 3×3 templates are assumed. The same colored cells are updated in parallel.

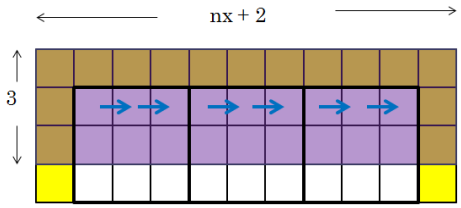


Figure 4: Use of shared memory for updating the cells placed in horizontal (x) direction.

Shared memory is also used to update the cells placed in horizontal and perpendicular directions. In this case, we need to separate the update into the two steps as shown Figs. 5(a) and 5(b). The updated first two lows do not have dependency as shown in Fig. 5(a). However, since the cells in the last low are updated in another block, we need to reread the last low. Hence, the $4 \times (nx+2)$ internal states and outputs shown in Fig. 5(a) are read from global memory into shared memory in the first step, and the $3 \times (nx+2)$ as shown in Fig. 5(b) are read in the second one. Therefore, the internal states and outputs are read and written twice from global memory, which accelerates the update of cells.

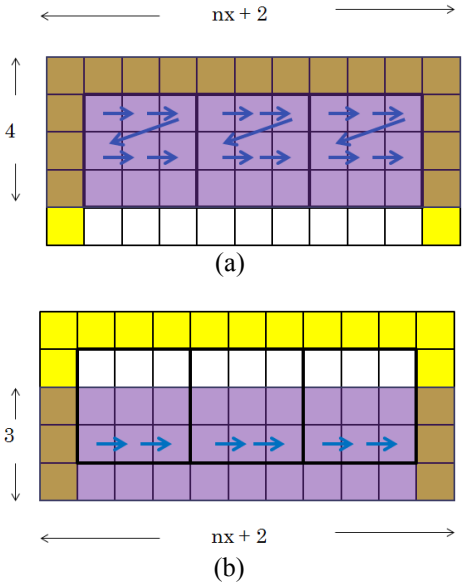


Figure 5: Shared memory is used for updating the cells placed in horizontal (x) and perpendicular (y) directions. (a) First step. (b) Second step.

4. Results

The image halftoneng templates [5] were used to evaluate the GPU implementation. We used GeForce GTX 560 Ti with 1.0GB global memory, 65.5KB shared memory, and 1.67GHz clock speed. For a comparison, the results were compared with one obtained by Intel® Core™ i5 with 4096 MB memory.

Figure 6 shows the conversion image obtained by the GPU, where an appropriate halftone image is obtained. The image is almost identical to Fig. 1(b) given by the GSN method [8]. From this fact, we see that the parallel computation scheme provided in Sect. 3 is valid.

Tables 1 and 2 shows the comparison of CPU times between GPU and CPU implementations for different size of images. In these tables, direction x implies that shared memory is used to update the cells placed in horizontal (x) direction as shown in Fig. 4, and directions x and y are corresponding to Fig. 5. The speed up ratio without shared memory is not high. Fortunately, the ratio becomes high taking advantage of shared memory for update of more cells. As a result, we can accelerate the operation until 20 times faster than CPU.

Figure 7: Halftone image obtained by the asynchronous DT-CNN implemented on GPU.

Table 1: Performance of GPU implementation for 256× 256 pixels image.

direction	block size	GPU [sec.]	speed up
	1	0.480	1.875
x	1	0.149	6.048
x	2	0.146	6.252
x and y	1	0.109	8.294
x and y	2	0.105	8.650

Table 2: Performance of GPU implementation for 512× 512 pixels image.

direction	block size	GPU [sec.]	speed up
	1	0.614	5.64
x	1	0.239	14.50
y	2	0.249	13.90
x, y	1	0.181	19.08
x, y	2	—	—

5. Conclusions

We have presented the GPU implementation of asynchronous DT-CNNs. To accelerate the operations, the parallel computation scheme and utilizing the shared memory are provided. Using the halftoning templates, we confirm that the proposed implementation certainly speeds up the operations.

References

[1]. H. Harrer and J. A. Nossek, “Discrete-time cellular neural networks,” Int. J. Circuit Theory and Applications, vol. 20, pp. 453-467, 1994.

[2]. T. Roska, Á. Zarándy, S. Zöld, P. Földesy, and P. Szolgay, “The computational infrastructure of analogic CNN computing-Part I: the CNN-UM chip prototyping system,” IEEE Trans. Circuits Syst.-I}, vol. 46, no. 1, 1999.

[3]. T. Ikenaga and T. Ogura, “A DTCNN universal machine based on highly parallel 2-d cellular automata CAM^2, IEEE Trans. Circuit Syst.- Part I, vol. 45, No. 5, pp. 538-546, 1998.

[4]. M. H. ter Brugge, J. A. G. Nijhuis, L. Spaanenburg, “Transformational DT-CNN design from morphological specifications,” IEEE Trans. Circuits Syst., vol. 45, no. 9, pp. 879-888, Sept. 1998.

[5]. M. Ikegami and M. Tanaka, “Image coding and decoding by discrete time cellular neural networks,” Trans. IEICE-A, vol. J77-A, pp. 954-964, 1994 (in Japanese).

[6]. D. Uchimoto , H. Numata , Y. Tanji , and M. Tanaka , ”Design of discrete-time cellular neural networks image processor,” Trans. IEICE-DII, vol. J84-D-II, no. 7, pp. 1464-1474, July 2001 (in Japanese).

[7]. J. M. Orgega and W. C. Rheinboldt, I*terative Solution of Nonlinear Equations in Several Variables*, New York: Academic, 1970.

[8]. Y. Tanji, “On a synchronous/asynchronous model of discrete time cellular neural network,” Trans. IEICE, vol. J85-A, no. 6, pp. 725-729, 2002.