2017 International Symposium on Nonlinear Theory and Its Applications,
NOLTA2017, Cancun, Mexico, December 4-7, 2017

Parallel Computing of Neural Network Algorithm for Fixed Channel Assignment Problem in Cellular Radio Networks with CUDA

Sho Ikeda, Yoichi Tomioka and Junji Kitamichi

†Graduate School of Computer Science and Engineering, The University of Aizu
Aizuwakamatsu, Fukushima, 965-8580, JAPAN
Email: m5211129@u-aizu.ac.jp, ytomioka@u-aizu.ac.jp and kitamiti@u-aizu.ac.jp

Abstract—In recent years, graphics processing units (GPUs) have been used for faster numerical calculation because they have many processing elements and can calculate via parallel computing. More recently, there is a growing interest in parallel computing using a general-purpose GPU (GPGPU) in the field of neural network algorithms. In this paper, we propose a CUDA C program that aims to accelerate an extended maximum neural network algorithm for a fixed channel assignment problem in cellular radio networks using GPGPU. CUDA C runs on NVIDIA GPU for faster processing speed. We evaluate the developed program with the existing benchmark problem in the fixed channel assignment. Results show that the processing speed of the developed program is 2.4 times to 15.1 times faster than in the case of using only CPU.

1. Introduction

There is a growing demand for mobile devices using cellular radio networks (CRNs). However, the limited number of frequency bands and channels are available. Therefore, much research has been conducted to solve the channel assignment problem (CAP). To resolve the CAP, certain number of demanded channels must be efficiently assigned to cells without mutual interference. The CAP is one of the NP-complete problems that cannot be solved in polynomial time, and many researchers have studied approximation algorithms to solve it. For example, Box [1] proposed to assign channels in descending order of demand difficulty, Sivarajan [9] viewed CAP as a graph-coloring problem, Funabiki [3] proposed a neural network algorithm for the CAP, and so on. Because the neural network algorithm is also parallel distributed processing, it is suitable for parallel computing using a multi-core processor. In this paper, we propose a parallel computing algorithm using GPU. Our aim is to accelerate the processing speed based on an expanded neural network algorithm proposed by Ikenaga and his colleagues [4] [5]. We develop a program for the fixed CAP using CUDA C programming that is a parallel computing platform and programming model for GPU by NVIDIA. Results show that the processing speed of the developed program is 2.4 times to 15.1 times faster than in the case of using only a CPU.

2. Background

2.1. Channel Assignment Problem

In this research, we treat a frequency allocation in cellular radio networks as a channel assignment problem (CAP). The frequencies we use for communication are called channels. Cellular radio networks consist of base transceiver stations (BTSs) and users. Users can communicate with other people because a BTS assigns an available channel to a mobile terminal. A cell is hexagonal area divided into a broad service area and has one BTS. The amount of interference is quantified in terms of the three conditions using Smith's formalization [10]. Three conditions are as follows:

1. Common channel constraint: The same channel can not be assigned to a certain pair of radio cells at the same time.
2. Adjacent channel constraint: Adjacent channels in the frequency domain can not be assigned to adjacent radio cells at the same time.
3. Common cells constraint: Any pair of frequencies assigned to a single radio cell must keep a certain distance from each other in the frequency domain.

The goal of solving the CAP is to assign the channel in a way that the sum of the mutual interference amounts becomes 0. The amount of interference is defined as the interference matrix E obtained from a symmetric compatibility matrix C. The number of channel requirements for each cell is defined as the demand matrix D. Assuming that there are N cells and M channels available, the matrix E is expressed three-dimensionally as $N \times N \times M$. e_{ijk} is an element of matrix E, and it is the amount of interference when channels assign to cell i and cell j with the distance between the channels being k as shown in formula (1). d_i is an element of matrix D and is the number of channels demanded for cell i. Figure 1 is an example of the result of the CAP. In this example, there are 4 cells and each cell has 11 channels. In addition, formula (2) represents matrix D and matrix C. Then, formula (3) is calculated from matrix C. In Figure 1, gray circles indicate channels assigned to a mobile terminal, and the total interference is 0.

$$e_{ijk} = \begin{cases} 0 & (if \quad c_{ij} \leq k) \\ c_{ij} - k & (if \quad c_{ij} > k) \end{cases} \tag{1}$$

$$D = \begin{pmatrix} 2 \\ 1 \\ 3 \\ 2 \end{pmatrix} \quad C = \begin{pmatrix} 5 & 4 & 0 & 0 \\ 4 & 5 & 0 & 1 \\ 0 & 0 & 5 & 2 \\ 0 & 1 & 2 & 5 \end{pmatrix} \tag{2}$$

$$e_{ij1} = \begin{pmatrix} 4 & 3 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix} \quad e_{ij2} = \begin{pmatrix} 3 & 2 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} \tag{3}$$

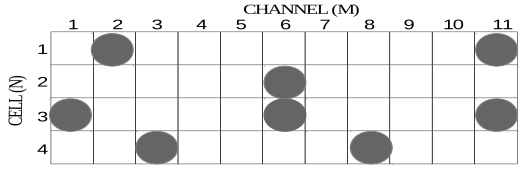

Figure 1: Example of Channel Assignment Result

2.2. Neural Network Algorithm

We adopt an extended maximum neural network algorithm (EMNNA) proposed by Ikenaga and his colleagues [4] [5] which is based on Hopfield Neural Network. A two-dimensional $N \times M$ matrix defines the neuron representation, and the ij neuron corresponds to element ij of this matrix. Here, i corresponds to a cell i and j corresponds to a channel j. The neuron input has an integer of U_{ij} and the neuron output has a binary value V_{ij}. $V_{ij} = 0$ means that there is no assigned channel, and $V_{ij} = 1$ means that there is an assigned channel. All neurons are grouped for each cell, and d_i neurons are selected for cell i. In formula (4), U_{ith} is the ith-th largest number when U_{ij} are sorted by descending order in cell i. If the number of neurons satisfies $U_{ij} \geq U_{ith}$ is more than d_i, the neuron that is 0 at the last time is given the priority [5].

$$V_{ij} = \begin{cases} 1 & (if \quad U_{ij} \geq U_{ith}) \\ 0 & (\quad otherwise \quad) \end{cases} \tag{4}$$

The energy E of entire network is defined by formula (5). V is the result of assigned channel when this function has the minimum value. This EMNNA is added some heuristic methods to improve the accuracy of solution. Therefore, the neuron input values are updated by formulas (6) and (7). Formula (6) is added *Shaking Term*, *Omega function* and *Hill-Climbing Term*. In addition, regular interval assignment and re-initialization method are used in this algorithm. Regular interval assignment assigns a channel for a regular interval to cell i that has the greatest d_i. The re-initialization method realizes large hill climbing by initializing input U_{ij} again.

$$E = \frac{A}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{\substack{q=1 \\ (i,j)\neq(p,q)}}^{M} e_{ip|j-q|} V_{ij} V_{pq} \tag{5}$$

$$\begin{aligned} \frac{dU_{ij}}{dt} = \frac{\partial E}{\partial V_{ij}} = & - A \sum_{p=1}^{N} \sum_{\substack{q=1 \\ (i,j)\neq(p,q)}}^{M} e_{ip|j-q|} V_{pq} \\ & (if(t \bmod T_\omega \geq \omega)) \\ & - A \sum_{p=1}^{N} \sum_{\substack{q=1 \\ (i,j)\neq(p,q)}}^{M} e_{ip|j-q|} V_{pq} V_{ij} \\ & (if(t \bmod T_\omega < \omega)) \\ & + Bh(\sum_{\substack{p=1, \\ p\neq i}}^{N} \sum_{q=1}^{M} e_{|j-q|} V_{pq}) \\ & + C(1 - V_{ij}) \end{aligned} \tag{6}$$

$$U_{ij\ new} = \frac{dU_{ij}}{dt} + U_{ij\ old} \tag{7}$$

If $\sum_{\substack{p=1, \\ p\neq i}}^{N} \sum_{q=1}^{M} e_{|j-q|} V_{pq}$ is 0, then function h is 1, otherwise, function h is 0.

2.3. GPGPU

A GPU is a microprocessor calculating 2D or 3D graphics, videos, and images. A modern GPU is a multi-thread multi-processor for faster calculation, and there is a growing interest in GPU computing. A GPU and CUDA have been developed by NVIDIA [6]. CUDA allows us to use the CUDA acceleration library, compiler directives, application programming interface, and an extended programming language such as C, C++, Fortran, or Python. CUDA C is based on ANSI C. Its program is composed of host code executed on a CPU and device code executed on a GPU.

A GPU by NVIDIA is designed with CUDA architecture. Second-generation Maxwell is one of the CUDA architecture, and we adopt it in this research. Maxwell's characteristic feature is the structure of its multi-processor. Its streaming multi-processor, known as SMM, is composed of four 32-core processing blocks. Therefore its performance per CUDA core is higher than 40% because of new data path organization and an improved instruction scheduler [8]. GTX960 has 1024 CUDA cores and 2 GB global memory. In addition, it is powered by 1127 MHz base clock and 112 GB/sec memory band width. In CUDA C programming with GTX960, maximum number of resident grids per device is 32, maximum x-dimension of a grid of thread blocks is $2^{31} - 1$ and maximum number of threads per blocks is 1024.

3. Proposed CUDA Program

We parallelize the EMNNA. We code a base program to do the following:

1. Execute on setting initialization
2. Initialize input value of U_{ij}
3. Obtain output value of V_{ij} by the MNNA

4. Calculate of total mutual interference using energy function
5. Update U_{ij} by motion equation
6. Repeat from *Step3* to *Step5* until the total mutual interference is 0 or the update number reaches a fixed time.
7. Assign channel where V_{ij} is 1.

In this program, *Step3*, *Step4*, and *Step5* need to be repeated many times for calculation. In addition, they are easy to parallelize beacause each neuron is calculated independently. Therefore, we introduce parallel techniques using CUDA for each step to accelerate the processing speed.

3.1. Introduction of CUDA C programming

We describe a parallel computing method for determining the output of each neuron corresponding to *Step3*, derivation of the total interference corresponding to *Step4* and updating each neuron input U_{ij} corresponding to *Step5* of the original program.

In *Step3*, we treat one thread as each cell. Therefore, we use one block and it has cell threads. Neurons of the same cell are treated as the same group. Each thread determines the output of neurons from the inuput of neurons based on neuron representation and allocates a channel. In a EMNNA, one neuron has a relationship with the value of other channels in the same cell. Therefore, it can not be processed independently. Figure 2 shows the deployment diagram of blocks and threads. The number of blocks is one and the number of threads per block is N. Therefore, we realize $1\times N$ parallelization. A thread calculates V at the i-th cell. For example, when blockID is 0 and threadID is $N-1$, the thread calculates V_{Nj}. In other words, the thread assigns demand channels to the N-th cell.

In *Step4*, we treat one block as one cell and one thread in the block as one channel. Therefore, a neuron output corresponds to the *threadID*-th channel assignment in the *blockID*-th cell. Each thread calculates the amount of interference between ouput neurons. Figure 3 shows a corresponding diagram of blocks and threads. The number of block is N and the number of threads per block is M. Therefore, we realize $N \times M$ parallelization. A thread calculates $\sum_{p=1}^{N}\sum_{q=1(i,j)\neq(p,q)}^{M} e_{ip|j-q|}V_{ij}V_{pq}$ at the i-th cell and j-th channel. For example, when blockID is $N-1$ and threadID is $M-1$, the thread calculates $\sum_{p=1}^{N}\sum_{q=1(N,M)\neq(p,q)}^{M} e_{Np|M-q|}V_{NM}V_{pq}$. After calculating that, the calculated results are summed up.

In *Step5*, we treat one block as one cell and one thread in the block as one channel. Therefore, a neuron corresponds to the *threadID*-th channel in the *blockID*-th cell. Each thread changes the state of the neuron according to the motion equation. Figure 4 shows a deployment diagram of blocks and threads. The number of blocks is N, and the number of threads per block is M. Therefore, we realize $N \times M$ parallelization. A thread calculates U at i-th cell and j-th channel. For example, when blockID is $N-1$ and threadID is $M-1$, the thread calculates $\frac{dU_{NM}}{dt}$ and updates U_{NM}.

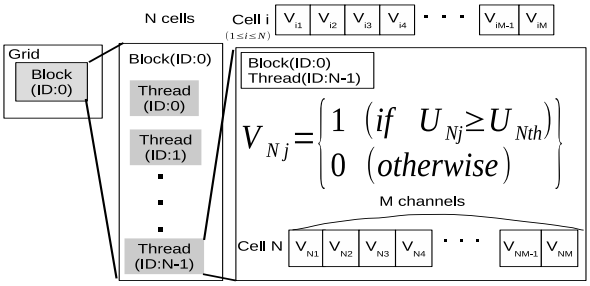

Figure 2: Deployment Diagram Determining Output of Neuron

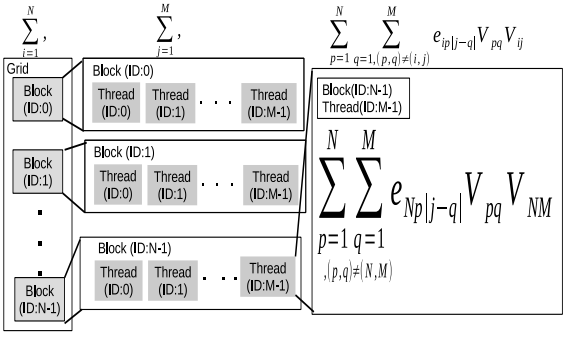

Figure 3: Derivation of the Total Interference

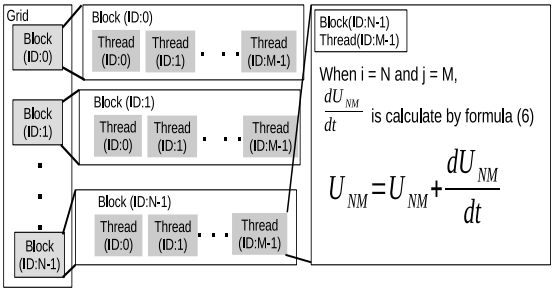

Figure 4: Updating Input of Neuron

4. Evaluation through Simulation

4.1. Benchmark Set

We provide simulation for the problems in reference [2] to compare the proposed program using CUDA and the program using C language. Table 1 shows the details of the problems. *Prob* is the number to identify the problem. N is the number of BTSs, and M is the number of channels per BTSs. C_{ii} is the amount of interference in the same cells. *a.c.c.* refers to the amount of interference in adjacent cells. *Total* indicates the total number of required channels, *Max* is the maximum number of required channels and *Min* is the minimum number of required channels.

4.2. Results of Execution

Table 2 shows the execution results, comparing using only a CPU with using a CPU and GPU. The program execution environments are GeForce GTX960 as the GPU, Intel Corei5-4570S (2.9GHz) as the CPU, CUDA Compiler Driver ver.7.5.17 as a compiler for the GPU, and GCC ver. 4.4.7 using option -O2 as compiler for the CPU. We simu-

late the developed program 10 times for each problem. In the CPU only, we execute a base program written in C language with OpenMP. In the CPU+GPU, we add CUDA to the base program and execute them using CUDA on a CPU and GPU. We use the systemcall *gettimeofday()* for time measurement. In Table 2, *Ave* is the average execution time of 10 simulations, and *Min* is the minimum execution time of them. In Prob 1 and Prob 2, execution times using a GPU are much longer than when using only a CPU. However, the processing speed when using a GPU is up to from 2.4 times to 15.1 times faster than in Prob 3,4,5,6,7,and 8. We profile the result of Prob 4 using *nvprof*. *nvprof* is a profiling application that can be used on the command-line from CUDA 5.0 [7]. In one simulation, the updating time is 81, the final total interference is 0 and the execution time is 1243.93 ms. Table 3 shows profiling results then. *Time* is the total execution times and *Ratio(%)* is the rate of them. *Ave* is the average kernel time per execution and *Min* is the minimum time of it. *memory copy to CPU* is the memory copy from GPU memory to CPU main memory. *memory copy to GPU* is the memory copy from CPU main memory to GPU memory. From this result, it is clear that "Derivation of Total Interference" and "Updating Input of Neuron" occupy a lot of time. Therefore, if we were to further reduce the execution time, these processes would be important.

Table 1: Benchmark Problems

			Matrix C		Vector D		
Prob	N	M	C_{ii}	a.c.c.	Total	Max	Min
1	4	11	5	-	6	3	1
2	25	73	2	-	167	11	4
3	21	381	5	1	481	77	8
4	21	533	7	1	481	77	8
5	21	533	7	2	481	77	8
6	21	221	5	1	470	45	5
7	21	309	7	1	470	45	5
8	21	309	7	2	470	45	5

Table 2: Execution Time

	CPU		CPU+GPU	
Prob	Ave[s]	Min[s]	Ave[s]	Min[s]
1	2.901	0.118	0.05450	0.05308
2	0.5049	0.3012	0.5258	0.3727
3	7.742	5.373	0.6042	0.5313
4	16.72	12.177	1.252	0.6517
5	16.39	13.815	1.086	0.9113
6	4.377	3.199	1.847	0.8970
7	10.25	4.704	2.904	2.902
8	12.93	3.029	2.899	2.898

Table 3: Profiling Results of Execution Time

Name	Ratio(%)	Time[ms]	Ave[ms]
Derivation of Total Interference	54.36	600.64	7.4153 [ms]
Updating Input of Neuron	40.89	451.75	5.5772 [ms]
Determining Output of Neuron	4.52	49.912	0.6162 [ms]
memory copy to CPU	0.13	1.3963	5.7450 [μs]
memory copy to GPU	0.11	1.2241	3.7660 [μs]

5. Conclusion

We propose a high-speed parallel program by CUDA. The proposed CUDA program uses an extended maximum neural network algorithm for the fixed channel assignment problem in cellular radio networks. This program is shown to accelerate the processing speed from 2.4 times to 15.1 times faster by CUDA programing using a NVIDIA GPU for the problem that needs a lot of time. The results show that it is more effective to introduce parallel computing by GPGPU to a neural network algorithm for a large-scale channel assignment problem.

In future work, we will apply a GPGPU not only to the fixed channel assignment problem but also to the dynamic channel assignment problem and show that program using GPGPU is also effective for solving the dynamic channel assignment Problem.

References

[1] F. Box, "A heuristic technique for assigning frequencies to mobile radio nets," IEEE Transactions on Vehicular Technology, vol. 27, no. 2, pp. 57–64, 1978.

[2] R.-H. Cheng, C.-W. Yu, and T.-K. Wu, "A novel approach to the fixed channel assignment problem," Journal of information science and engineering, vol. 21, no. 1, pp. 39–58, 2005.

[3] N. Funabiki and Y. Takefuji, "A neural network parallel algorithm for channel assignment problems in cellular radio networks," IEEE transactions on Vehicular technology, vol. 41, no. 4, pp. 430–437, 1992.

[4] K.Ikenaga, N.Funabiki, Y.Takenaka and J.Kitamichi, "An expanded maximum neural network algorithm for a channel assignment problem in cellular radio networks," IEICE Technical Report NLP, vol. 97, no. 592, pp. 85–92, Mar 1998.

[5] K.Ikengaga, N.Funabiki and Y.Takenaka, "An expanded maximum neural network algorithm for a channel assignment problem in cellular radio networks," IEICE Trans. Fundamentals A, vol. 82, no. 5, pp. 683–690, May 1999.

[6] NVIDIA Corporation, "Cuda c programming guide," (http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html).

[7] NVIDIA Corporation, "Nvidia cuda toolkit 8.0," (http://docs.nvidia.com/cuda/pdf/CUDA_Toolkit_Release_Notes.pdf).

[8] NVIDIA Corporation, "Whitepaper nvidia geforce gtx980," (http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF).

[9] K. N. Sivarajan, R. J. McEliece, and J. W. Ketchum, "Channel assignment in cellular radio," Vehicular Technology Conference, 1989, IEEE 39th, pp. 846–850, IEEE, 1989.

[10] K. Smith and M. Palaniswami, "Static and dynamic channel assignment using neural networks," IEEE Journal on selected areas in communications, vol. 15, no. 2, pp. 238–249, 1997.