An Implementation of Coincidence Algorithm on Graphic Processing Units

Thitipan Tongsiri and Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University Bangkok, Thailand
Thitipan.T@student.chula.ac.th and prabhas.c@chula.ac.th

***Abstract*—Genetic Algorithms (GAs) are powerful search techniques. However when they are applied to complex problems, they consume large computation power. One of the choices to make them faster is to use a parallel implementation. This paper presents a parallel implementation of Combinatorial Optimisation with Coincidence Algorithm (COIN) on Graphic Processing Units. COIN is a modern GA. It has a wide range of applications. The result from the experiment shows a good speedup in comparison to a sequential implementation on modern processors.**

Keywords — Genetic Algorithm; Parallel Processing; Graphic Processing Unit

I. INTRODUCTION

Genetic Algorithm (GA) is a popular technique to solve complex problems. GA can find optimal solutions but it consumes large computation power. There are many works on parallel GAs [1]. They make GAs run faster by performing parallel execution on multiple processors. This work presents an implementation of a modern Genetic Algorithm called Coincidence Algorithm (COIN) [2]. This algorithm is based on a second generation of GAs, Estimation of Distribution Algorithm (EDA). COIN is shown to be competitive with well-known EDA algorithms such a MIMIC[3], TREE[4], EBNA[5], UMDA[6]. In COIN, besides the usual positive knowledge where the evolution depends on the recombination of the better solutions, the negative knowledge is also exploited. The worse solutions are used to enhance the search by avoiding the reproduction of undesired solutions. The speedup of COIN comes from parallel programming on Graphic Processing Units (GPU).

This paper is organized as follows. In the preliminary section, an overview of COIN algorithm is given. The GPU used in this work is introduced. Then, in the implementation section, the detail of the method to parallelize the COIN algorithm on GPU is explained. The next section describes the experimental setup. The results are discussed based on a comparison of the execution time between CPU and GPU and the quality of solutions from both methods. The last section presents the conclusion.

II. PRELIMINARIES

A. GPU and CUDA Architecture

Graphic Processing Units are powerful and inexpensive. This explains why many researchers and developers have interest in this computation device. GPU has tremendous computation and memory bandwidth. Therefore, many well-known algorithms can be speed up against the implementation on CPU. For example, Manavski [7] showed speed up 541% for AES cipher on GPU. Su Chang [8] showed the speed up 525% of the implementation of MD5 cipher. The implementation on GPUs shows a very high speed-up against the implementation on CPU. GPU is also low cost so it is attractive for researchers.

A Graphic Processing Unit (GPU) has two key advantages. First it has many core processors which dedicate to compute-intensive therefore it is highly parallel. The performance advantage of a GPU is illustrated in Fig. 1. Second, GPU has a very high memory bandwidth, for example 141 GBps on the NVIDIA GeForce GTX 280 [9]. This allows high data transfer rate and high throughput between its device memory. However there is some overhead such as the slow speed data transfer between host memory and device memory (8 GBps on the PCIe x16 Gen2) [9]. This must be minimized to gain the speed up as much as possible.

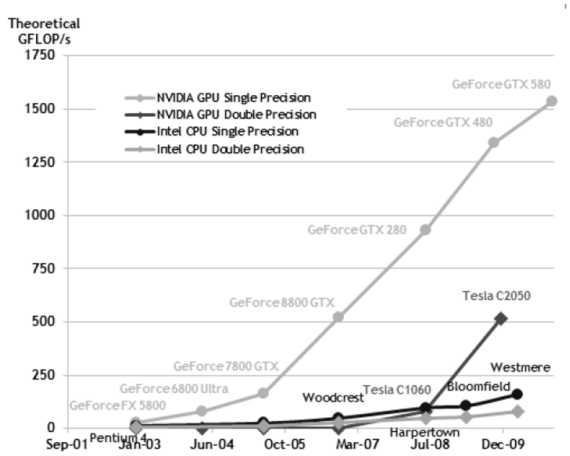

Figure 1. Floating-Point operation per second for the CPU and GPU

978-1-4673-1921-8/12/$31.00 ©2012 IEEE

NVIDIA introduced Compute Unified Device Architecture (CUDA) in 2006. It is a general purpose parallel computing architecture. CUDA introduces a new parallel programming model and an instruction set architecture. This facilitates the use of the parallel computing engine in NVIDIA GPUs to solve many complex computational problems.

CUDA architecture consists of a host which is a CPU side and one or more computing device (GPU) side. They work together to produce high throughput of data computation. The computing structure in devices is arranged in a hierarchy of blocks and threads as shown in Fig 2. The data can be simultaneously computed on GPU. The kernel calling from host side will trigger GPU to execute tasks on its processors which are arranged in blocks and threads.

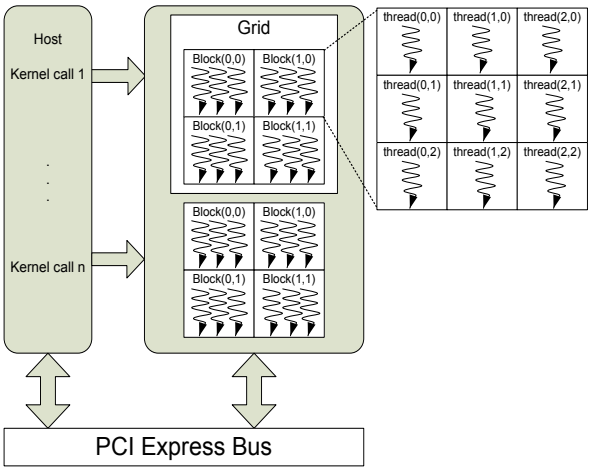

Figure 2. Hierarchy of computing structure in a GPU. Kernel function can be executed by calling from host. A number of blocks and threads are assigned to the kernel function.

B. *Coincidence Algorithm*

The Coincidence algorithm (COIN) was introduced by Wattanapornprom W. et al. in 2009 [2]. The main idea is to model combinatorial problems as Markov Chain. This representation can be realised by a matrix. This matrix is used as a distribution model of solutions. A population is drawn from this model. The evolution of solutions progresses by sampling from this matrix and adjusting the weights in the matrix according to the patterns learned from the population.

The learning is derived from selected subpopulation to apply reward and punishment to adjust the weights in the matrix. The distinct character of COIN is that besides learning from the better subpopulation, it also learns a negative knowledge from the worse subpopulation. This allows the algorithm to avoid sampling the undesired population.

COIN consists of six steps. In the first step, the matrix is initialized. In the second step, the population is sampling from the matrix. The population is evaluated by a fitness function (problem dependent) in the third step. In the fourth step the selected population is divided into two groups, good and bad according to fitness values. The fifth step, the matrix is updated dependent on the patterns found in good and bad population. The final step repeats the second step to the last step until the terminating condition is met.

Step 1. Initialize the generator.
Step 2. Generate the population using the generator.
Step 3. Evaluate the population.
Step 4. Select the candidates. There are two methods:
- **a.** Uniform selection: select the top and bottom c percent.
- **b.** Adaptive selection: select the above and below the average $\pm 2\sigma$

Step 5. For each joint probability $h(x_i|x_j)$, update the generator according to the reward and punishment :-

$$X_{i,j}(t+1) = X_{i,j}(t) + \frac{k}{(n-1)}\left(r_{i,j}(t+1) - p_{i,j}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{i,j}(t+1) - \sum_{j=1}^{n} r_{i,j}(t+1)\right)$$

where X_{ij} denotes the joint probability $h(x_i|x_j)$, k is the learning coefficient, r_{ij} denotes the number of coincidence $X_i X_j$ found in the good solutions, p_{ij} denotes the number of coincidence $X_i X_j$ found in the not-good solutions, n is the size of the problem.

Step 6. Repeat **Step 2.** Until the terminate condition is met.

Figure 3. The steps of COIN algorithm

The Markov Chain is modeled as a matrix of size n×n. Each column in a row contains the joint probability $h(X_i|X_j)$. X_i indicates the row of the matrix, while X_j indicates the column. X_i, X_j is a coincidence of the event. A coincidence . X_i, X_j means the event X_i is followed by the event X_j. The coordinate X_{ij} indicates the joint probability $h(X_i|X_j)$. For example, the solution of Traveling Saleman Problem (TSP) is the shortest paths which we can travel through all cities without visiting the same city twice. The solutions for TSP five cities are represented by $X_1X_2X_3X_4X_5$ where X_i = {A, B, C , D, E} we can go to every cities B, C, D or E from A. The path $X_{i,j}$ from A to B is represented by $X_{A,B}$ and joint probability $h(X_i|X_j)$ is written as $h(X_A|X_B)$.

Next, the detail of each step of the algorithm is explained.

Initializing the generator

The generator is initialized by filling $h(X_i|X_j)$ with $\frac{1}{(n-1)}$ except X_{ij} where $i = j$. This initialization represents a uniform distribution of each coincidence.

	A	B	C	D	E
A	0	0.25	0.25	0.25	0.25
B	0.25	0	0.25	0.25	0.25
C	0.25	0.25	0	0.25	0.25
D	0.25	0.25	0.25	0	0.25
E	0.25	0.25	0.25	0.25	0.25

Figure 4. the matrix 5x5 filled up with an initial joint probability

Sampling the population

To generate a solution, $h(X_i|X_j)$ is sampling as follows:

1. Begin with the first node, X_i chosen by its empirical probability $h(X_i)$.
2. Sampling the next node X_j, $i \neq j$ from the matrix.
3. Start from the node X_j repeat Step 2 until a solution with length n is attained. The solution is composed of $X_i, X_j, ..., X_n$ where all indexes are distinct.
4. The population is sampling by drawing each solution until the desired size is reached.

Evaluate the population

To Each solution in the population is evaluated for its fitness value. The fitness evaluation function is dependent on the problem. For example, in Traveling Salesman Problem, the fitness value is a summation of all paths in the tour.

Selection of the candidates

After all members in the population have their fitness values. They are ranked and divided into two groups: above and below average fitness. The above average group is a better group. The below average group is a worse group. The size of each group is determined so that its member's fitness is ranged ±2σ (of the fitness value).

Update the matrix

Both better and worse groups are used to update the weights in the matrix. For each coincidence X_c, X_r found in the better group, the matrix $h(X_c|X_r)$ is rewarded according to the equation (1) where k is the learning step size, $X_{c,r}$ is the total number of coincidence. Vice versa for the worse group, the matrix is punished according to the equation (2)

$$X_{c,r}(t+1) = X_{c,r}(t) + \frac{k}{(n-1)}\left(r_{c,r}(t+1)\right) - \frac{k}{(n-1)^2}\left(\sum_{j=2}^{n} r_{c,j}(t+1)\right) \quad (1)$$

$$X_{c,p}(t+1) = X_{c,p}(t) - \frac{k}{(n-1)}\left(p_{c,p}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=2}^{n} p_{c,j}(t+1)\right) \quad (2)$$

The reward and punishment can be combined into one equation. Given a coincidence X_{c1}, X_{c2} found in both better and worse group then the combined equation is:

$$X_{c1,c2}(t+1) = X_{c1,c2}(t) + \frac{k}{(n-1)}\left(r_{c1,c2}(t+1) - p_{c1,c2}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=2}^{n} p_{c1,j}(t+1) - \sum_{j=2}^{n} r_{c1,j}(t+1)\right) \quad (3)$$

The code for COIN on a generic CPU can be found at http://www.cp.eng.chula.ac.th/faculty/pjw/project/coin/index-coin.htm

III. IMPLEMENTATION ON GPU

COIN algorithm can be implemented by breaking down a task in each original step to many parallel tasks as follows.

i. The population generation

Each solution in the population is sampling independently so generating population task can be distributed into k threads which reside in n blocks where n is calculated from population size m divided to k threads. Fig 5 illustrates this step with the task distribution of m population to n blocks and k threads. The important point is to make sure that the number of threads assigned to each block is sufficient to exploit multiprocessor in GPU. Each task in k threads is the same as the original task, a solution is sampling from the matrix.

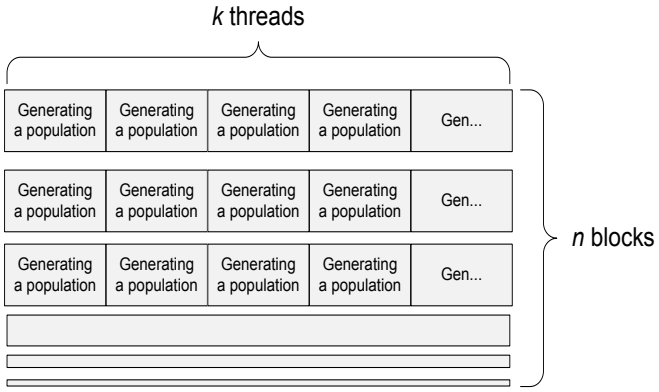

Figure 5. The population generation task can be assigned to n blocks and k threadswhere n is calculated from population size divided to number of thread.

ii. The fitness evaluation

The fitness of each solution (a tour in TSP) is calculated. Each solution is also independent therefore the whole population can be done in parallel. Again, n blocks and k threads is assigned to evaluate the fitness of a solution.

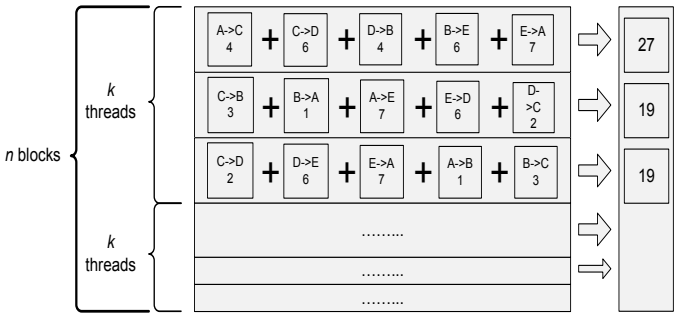

Figure 6. The evaluation of population can be assigned to n blocks and k threads.

iii. The population sorting

An important task which is not mentioned on the original paper on COIN is the sorting. Sorting the population is done before selection. The comparison function is the fitness value.

For population size 500 and 1000, it is considered as small for sorting and CPU can outperform GPU. The roundtrip time for memory transfer between CPU and GPU is fast, it is in microsecond.

iv. The matrix update

The last two key steps for COIN algorithm are selecting candidates from the population and updating the matrix. The uniform selection divides the population into two groups: better and worse. An implementation of the selection method is simply by marking the candidates. When updating the matrix, these candidates can be accessed.

The implementation of the matrix update is divided into two steps. The first step begins from counting coincidences found in each candidate. Both $r_{c1,c2}$ and $p_{c1,c2}$ are stored into a temporary matrix of size $2jk^2$ to the co-ordinate $c1$, $c2$ as illustrated in Fig. 8. The second step is the calculation of the equation (3) and updates the matrix.

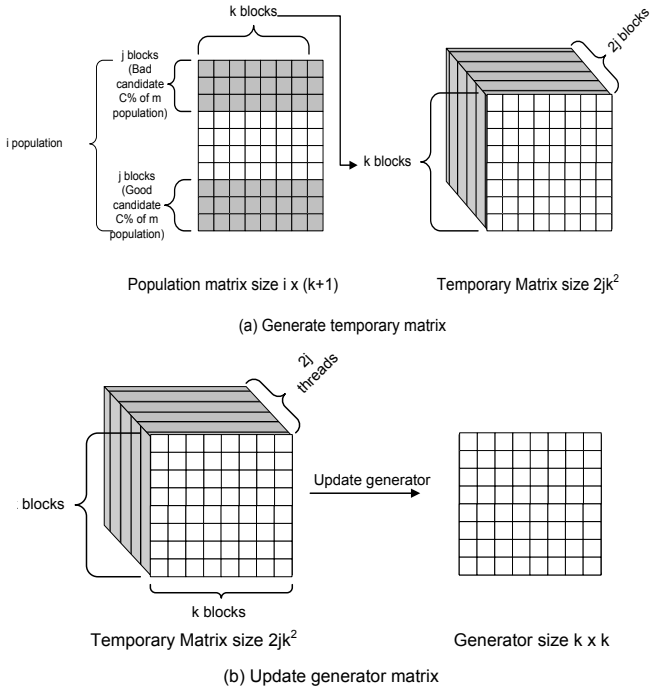

Figure 7. (a) Selecting the candidates and counting the coincidence. (b) Update the matrix.

IV. EXPERIMENTS

A. Experimental setup

To evaluate the parallel version of COIN on GPU, TSP problems are used. There are four problems with different size: Grostel24, Grostel48, Padberg/Rinaldi76, kroA100 [10]. Two population sizes are tested: 500 and 1000.

For the comparison, the sequential COIN runs on the CPU Intel core i3-2310M 2.1GHz, with memory 8 GB. The parallel COIN uses GPU GEFORCE 540M, clock 672 MHz, with memory DDR3 2GB. Each problem is iterated to 200 generations. To report the execution time, each experiment is repeated 10 times and the results are averaged.

B. Results

The execution time from the sequential and parallel version is compared. The speedup is calculated as $time_{seq}/time_{par}$. Table I shows the results of the speedup values of four problems, each with two population sizes. To ascertain the quality of the solutions, Table II records the actual values of the tours, best and average. The optimum solutions for these benchmarks are known.

TABLE I. SPEEDUP IN TSP PROBLEM

Problem	Population	Platform	Avg .Time(s)
grostel24	500	CPU	2.360
		GPU	0.333
		speedup	**7.083**
	1000	CPU	4.672
		GPU	0.534
		speedup	**8.746**
grostel48	500	CPU	6.017
		GPU	0.878
		speedup	**6.854**
	1000	CPU	12.815
		GPU	1.349
		speedup	**9.503**
pr76	500	CPU	14.121
		GPU	2.179
		speedup	**6.481**
	1000	CPU	29.901
		GPU	3.835
		speed up	**7.796**
kroA100	500	CPU	18.776
		GPU	3.536
		speedup	**5.311**
	1000	CPU	41.662
		GPU	6.338
		speedup	**6.574**

From Table I, it is clear that the larger size of population yields higher speedup by GPU. It is also true that the more computation load the higher speedup by GPU. This fact can be observed by comparing the speedup of two population size (500 and 1000) across all benchmark problems.

TABLE II. SOLUTION OF TSP PROBLEM

Problem	Population	Platform	Best	Avg.
Grostel24	500	CPU	1272	1318
		GPU	1272	1283
	1000	CPU	1272	1291
		GPU	1272	1275
Grostel48	500	CPU	5648	5975
		GPU	5414	5529
	1000	CPU	5606	5909
		GPU	5170	5379
Padberg/Rinaldi 76	500	CPU	135218	142694
		GPU	137041	143889
	1000	CPU	124292	134268
		GPU	131592	135548
kroA100	500	CPU	38698	39950
		GPU	36127	37309
	1000	CPU	35616	38417
		GPU	33065	34172

*Optimal Grostel24 1272
Grostel48 5046
Pr76 108159
kroA100 21282
**Total runs 200 generations

V. CONCLUSION

This paper presents an implementation of COIN algorithm on Graphic Processing Units. A parallel version of COIN has been designed to minimize data transfer between a host and devices. The experiment is carried out to compare the execution time of the sequential COIN runs on CPU and the parallel COIN runs on GPU. The results show that a good speedup can be achieved on a large population.

REFERENCES

[1] John, D.O., David, L., Naga, G., et al.: "A survey of general-purpose computation on graphics hardware," In: Eurographics, 2005, State of Art Reports, pp.21-51.(2005)

[2] Wattanapornporm, W., et al.: "Multi-objective Combinatorial Optimisation with Coicidence Algorithm," IEEE Congress on Evolutionary Computation, May 18-21,pp. 1675-1682. (2009)

[3] De Bonet J.S, Isbell, C.L., and Viola, P.: "MIMIC: Finding Optima by Estimating Probability Densities," Advance in Neural Information Processing Systems, volume 9. (1997)

[4] Chow, C. and Liu, C.: "Approximating Discrete Probability with Dependency Trees," IEEE Transactions on Information Theory, 14:462-467. (1967)

[5] Etxeberria, R. and Larrañaga, P.: "Global Optimization with Bayesian Networks," Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization, pages 322-339. (1999)

[6] Robles, V. and Larrañaga, P.: "Solving the Traveling Salesman Problemwith EDAs," Estimation of Distribution Algorithm: A New Tool for Evolutionary Computation. (2002)

[7] Manavski, S. A.: "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," IEEE International Conference on Signal Processing and Communication, pp. 65-68. (2007)

[8] SU Chang: "Fast operation of large-scale high-precision matrix based on GPU," Journal of Computer Applications, pp.1179. (2009)

[9] NVIDIA CUDA Programming Guide v4.0, 1:10-11. (2011)

[10] Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I. and Dizdarevic, S.: "Genetic algorithms for the travelling salesman problem: A review of representations and operators," Artificial Intelligence Review, 13:129-170. (1999)