A QAP Solver with CUDA GPU Computing Architecture

A Two Page Description of the Application
Submitted for GECCO 2009 Competition :
GPUs for Genetic and Evolutionary Computation

Noriyuki Fujimoto
Graduate School of Science, Osaka Prefecture University
1-1 Gakuen-Cho, Naka-ku, Sakai-Shi,
Osaka, 599-8531, Japan
fujimoto@mi.s.osakafu-u.ac.jp

Shigeyoshi Tsutsui
Department of Management and Information Science, Hannan University
5-4-33 Amamihigashi, Matsubara,
Osaka 580-8502, Japan
tsutsui@hannan-u.ac.jp

1. INTRODUCTION

This application solves the quadratic assignment problem (QAP) [1]. In QAP, we are given l locations and l facilities and the task is to assign the facilities to the locations to minimize the cost. We chose QAP for the following reasons: First, problem sizes of QAPs in real life problems are relatively small compared with other problems in permutation domains such as the traveling salesman problem (TSP) and the scheduling problem. This enables us to use the shared memory of a GPU effectively. Second, QAP is one of the most difficult problems among problems in permutation domains. Thus, QAP is a good test bed to evaluate an optimization algorithm.

As the parallel method, a multiple-population, coarse-grained GA model was adopted. Each subpopulation is evolved by a multiprocessor in a CUDA GPU. At predetermined intervals of generations all individuals in subpopulations are shuffled via the VRAM of the GPU. Applying local search in solving QAP is very common in evolutionary algorithms [2, 3, 4]. Popular local searches used in solving QAP are the taboo search (TS) and 2-OPT heuristic. However, to apply these local searches efficiently, we need a large memory for each parallel thread. For example, the size needed for taboo search [5] is $32l^2 + l$ bytes. To take this size space for each thread in the shared memory with current GPU is almost impossible. Thus, we will not apply any local search for GPU. We applied local searches (2 OPT Best Move and 2 OPT First Move) for CPU.

The instances on which this algorithm was tested were taken from the QAPLIB benchmark library at [6]. Our results were promising, showing the performance of GPU programs without local searches is comparative to or even better than the performance of CPU programs with local searches on the Intel ® Core™ i7 965 processor, which is one of the fastest processors available currently.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

2. QUADRATIC ASSIGNMENT PROBLEM (QAP)

In QAP, we are given l locations and l facilities. For each pair of locations i and j, we are given their distance d_{ij}. For each pair of facilities i and j, we are given the flow f_{ij} between these facilities (regardless of their locations). The cost of each assignment, ϕ, is given by the sum over-all pairs of locations of the product of the distance between these locations and the flow between the facilities assigned to these locations. Assignments of facilities to locations can be encoded as permutations. The location for facility i is given by the value at position i of the permutation. The cost for such an assignment is given by

$$cost(\phi) = \sum_{i=1}^{l} \sum_{j=1}^{l} f_{ij} d_{\phi(i)\phi(j)}. \quad (1)$$

The task is to minimize $cost(\phi)$ over all possible assignments (permutations). Intuitively, $f_{ij}d_{\phi(i)\phi(j)}$ represents the cost contribution of simultaneously assigning facility i to location $\phi(i)$ and facility j to location j.

3. THE BASE GA MODEL FOR QAP

We describe the base GA model for QAP which is common both for GPU computation and CPU computation. In this study we use GA using commonly used crossover operators. Figure 1 shows the base GA model for QAP in this research. Let N be the population size. We use two pools P and W of size N. P is the population pool to keep individuals of the current population, and W is a working pool to keep newly generated offspring individuals until they are selected to update P for the nest generation. Then, the algorithm is as follows:

Step 1 Set generation counter $t \leftarrow 0$ and initialize P.

Step 2 Evaluate each individual in P.

Step 3 For each individual I_i in P, select its partner I_i ($i \neq j$) randomly. Then apply a crossover to the pair (I_i , I_j) and generate one child I'_i in position i in W.

Step 4 For each I'_i, apply a mutation with probability p_m.

Step 5 Evaluate each individual in W.

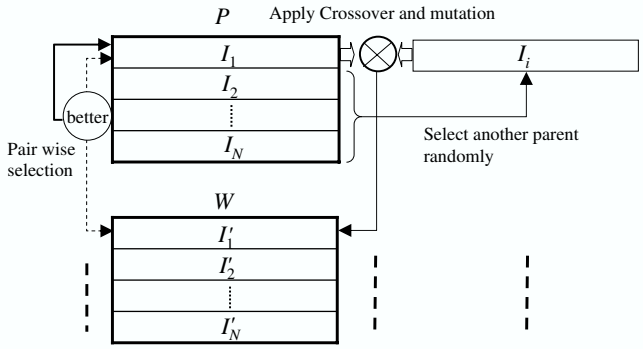

Figure 1: The base GA model for QAP

Step 6 For each i, compare the costs of I_i and I'_i. If I'_i is the winner, then replace I_i with I'_i.

Step 7 Increment generation counter $t \leftarrow t+1$.

Step 8 If the termination criteria are met, terminate the algorithm. Otherwise, go to Step 3.

For crossover operators, we perform From preliminary tests using two well known operators, i.e, the order crossover (OX) [7] and the partially mapped crossover (PMX) [8] by implementing the GA model both on CPU and GPU, we used PMX operator. For mutation operator, we used a swap mutation where values of two randomly chosen positions in a string are exchanged. Strings to which the mutation are applied are probabilistically determined with mutation rate of p_m.

4. PARALLEL GA MODEL FOR GPU COMPUTATION

The GPU NVIDIA GeForce GTX285 which we use in this study has 30 multiprocessors (MPs) and each MP has 8 stream processors (SPs) sharing 16KB high speed memory among them. To use this machines features efficiently, we divide individuals into subpopulations of size 128 each. For each subpopulation, we use the base GA model described in Section 3. So, we allocate the population pools P and W described in Section 3 to the shared memory of each MP. We define the total number of subpopulation as $30 \times k (k = 1, 2, ...)$. So, the total individuals number with this model is $128 \times 30 \times k$. The parameter k finally determines the total thread number in CUDA programming and the total thread number is the same with the total individuals number.

The procedure of the parallel GA model for GPU computation is as follows:

(1) In the host machine, all individuals are shuffled. Then, they are sent to the VRAM of the GPU.

(2) For a subpopulation not yet processed, each MP selects one of such subpopulations, and copies the corresponding individuals from VRAM to its shared memory, and performs the generational process up to $G_{interval}$ generations, and finally copies the evolved individuals from its shared memory to VRAM.

(3) The above process is repeated until all subpopulations are processed.

(4) Then, all individuals are copied back to the memory of its host machine and merged.

(5) These processes are repeated until termination criteria are satisfied.

As seen in these descriptions, this parallel GA model is a variant of distributed GAs.

5. IMPLEMENTATION DETAILS FOR GPU COMPUTATION

To load as many individuals as possible in shared memory, we represent an individual as an array of type unsigned char, rather than type int. This restricts the problem size we can solve to at most 255. However, this does not immediately interfere with solving QAP because QAP is fairly difficult even if the problem size is relatively small.

Let l be the problem size of a given QAP instance. Then, the size of each individual is l bytes. So, the size of population pools P and working pool W is $2l \times N$ where N is the number of individuals allocated to each MP at the same time. We have only 16KB shared memory per MP. To maximize both l and N, we chose N as 128 and the shape of a thread block as $128 \times 1 \times 1$ under the assumption that l is at most 56. In our implementation, we represent each individual as an array of L elements of type unsigned char and L is fixed to 56 regardless of l. Consequently, for W and P, our implementation consumes $2L \times N = 2 \times 56 \times 128 = 14336$ bytes in shared memory.

We stored distance matrix d_{ij} and flow matrix f_{ij} in the constant memory space so that they can be accessed via cache. To save the memory space size for these matrices, unsigned short was used for the elements of these matrices. We implemented a simple random number g enerator for each thread with independent seed number.

6. REFERENCES

[1] T. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. Econometrica, 25, 1957.

[2] T. Stützle and H. Hoos. Max-min ant system. Future Generation Computer Systems, 16(9), 2000.

[3] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. IEEE Transactions on Knowledge and Data Engineering, 11(5), 1999.

[4] S. Tsutsui. Parallel ant colony optimization for the quadratic assignment problems with symmetric multi processing. In Proceedings of the Sixth International Conference on Ant Colony Optimization and Swarm Intelligence. Springer, 2008.

[5] É. Taillard. taboo search code. 2004. http://mistic.heig-vd.ch/taillard/codes.dir/tabou_qap.cpp.

[6] QAPLIB - a quadratic assignment problem library, 2009. http://www.seas.upenn.edu/qaplib.

[7] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the travel salesman problem. In Proceedings of the 2nd International Conference on Genetic Algorithms. L. Erlbaum Associates Inc., 1987.

[8] D. Goldberg. Genetic algorithms in search, optimization and machine learning. Addison-Wesley publishing company, 1989.