Fast QAP Solver with ACO and Taboo Search on GPU using Move-Cost Adjusted Thread Assignment

GPUs for Genetic and Evolutionary Computation Competition at 2011 Genetic and Evolutionary Computation Conference

Shigeyoshi Tsutsui
Hannan University
5-4-33 Amamihigashi, Matsubara
Osaka 580-8502, Japan
tsutsui@hannan-u.ac.jp

Noriyuki Fujimoto
Osaka Prefecture University
1-1 Gakuenmachi, Nakaku,
Sakai, Osaka 599-8531, Japan
fujimoto@mi.s.osakafu-u.ac.jp

1. INTRODUCTION

There are several studies on solving the quadratic assignment problem (QAP) withGPUs using an evolutionary computation. In our previous studies [3], we applied GPU computation to solve quadratic assignment problems (QAPs) using a distributed parallel GA model on GPUs. However, in those studies no local searches were applied. In this QAP solver, we implemented a parallel ACO for QAPs on a GPU by combining tabu search (TS) with ACO in CUDA [4].

2. SINGLE INSTRUCTION, MULTIPLE THREADS (SIMT)

To obtain high performance with CUDA, here we need to know how each thread runs in parallel. The approach is called single instruction, multiple threads (SIMT). In SIMT each MP executes threads in groups of 32 parallel threads called *warps*.

A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp agree on their execution path. However, if threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path.

In our implementation to be described in Section 5, we designed the kernel function so that the threads that belong to the same warp will have as few branches as possible.

3. QUADRATIC ASSIGNMENT PROBLEM (QAP)

The QAP is the problem which assigns a set of facilities to a set of locations and can be stated as a problem to find a permutation ϕ which minimizes

$$cost(\phi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\phi(i)\phi(j)}, \tag{1}$$

where $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices and ϕ is a permutation of $\{1, 2, \ldots, n\}$. Matrix A is a flow matrix between facilities i and j, and B is the distance between locations i and j. The QAP is considered one of the hardest problem class in combinatorial optimization problems.

4. MOVE AND COMPUTATION OF MOVE COST IN QAP

Taboo Search (TS) seeks one with the best evaluation among all the neighboring solutions. If there are no improving moves, TS chooses one that least degrades the objective function. Thus, we need to calculate costs of all neiboring solutions efficiently. Let $N(\phi)$ be the set of neighbors of the current solution ϕ. Then a neighbor, $\phi' \in N(\phi)$, is obtained by exchanging a pair of elements (i, j) of ϕ. Then, we need to compute move costs $\Delta(\phi, i, j) = cost(\phi') - cost(\phi)$ for all the neighbouring solutions. The neighborhood size of $N(\phi)$ ($|N(\phi)|$) is $n(n-1)/2$ where n is the problem size.

When we exchange r and s elements of ϕ (i.e., $\phi(r)$, $\phi(s)$), the change of $cost(\phi)$, $\Delta(\phi, r, s)$, can be computed in computing cost $\mathcal{O}(n)$.

Let ϕ' be obtained from ϕ by exchanging r and s elements of ϕ, then fast computation of $\Delta(\phi', u, v)$ is obtained in computing cost $\mathcal{O}(1)$ if u and v satisfy the condition $\{u, v\} \cap \{r, s\} = \emptyset$.

To use this fast update, additional memorization of the $\Delta(\phi, i, j)$ values for all pairs (i, j) in a table are required.

5. IMPLEMENTATION OF ACO WITH TS ON A GPU

We coded the process of each step of ACO as a kernel function of CUDA. The overall configuration of ACO with TS for solving QAPs on a GPU with CUDA is shown in Figure 1.

Kernel functions are called from the CPU for each ACO iteration. In these iterations of the algorithm, only the best-so-far solution is transferred to CPU from GPU. It is used for checking whether termination conditions are satisfied.

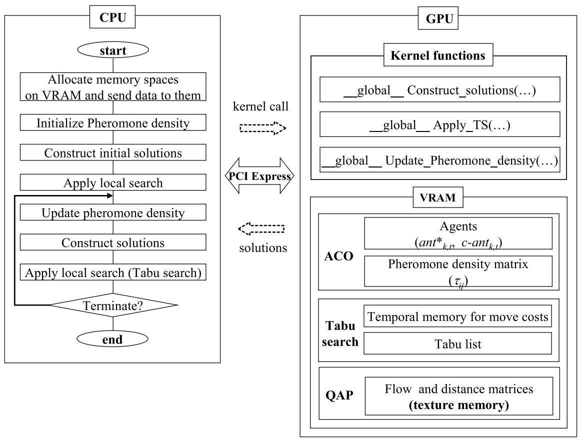
CPU
start
Allocate memory spaces on VRAM and send data to them
Initialize Pheromone density
Construct initial solutions
Apply local search
Update pheromone density
Construct solutions
Apply local search (Tabu search)
Terminate?
end
kernel call
PCI Express
solutions
GPU
Kernel functions
__global__ Construct_solutions(…)
__global__ Apply_TS(…)
__global__ Update_Pheromone_density(…)
VRAM
ACO
Agents ($ant^*_{k,t}$, c-$ant_{k,t}$)
Pheromone density matrix (τ_{ij})
Tabu search
Temporal memory for move costs
Tabu list
QAP
Flow and distance matrices (texture memory)

Figure 1: Configuration of ACO with TS on a GPU

Thus, in this implementation, overhead time used for data transfer between CPU and GPU can be ignored. In the end of a run, whole solutions are transferred from GPU to CPU.

6. MOVE-COST ADJUSTED THREAD ASSIGNMENT (MATA)

In general, for problem size n, the number of moves having move cost in $\mathcal{O}(1)$ is $(n-2)(n-3)/2$ and the number of moves having move cost in $\mathcal{O}(n)$ is $2n-3$. Table 1 shows these values for various problem sizes n. For larger size problems, ratios of $|N(\phi)|$ in $\mathcal{O}(n)$ to $|N(\phi)|$ have smaller values than those in smaller sized problems.

Table 1: Neighborhood sizes for various problem sizes

Problem size n	Neighborhood size $\lvert N(n)\rvert$	$\lvert N(n)\rvert$ in $O(1)$	$\lvert N(n)\rvert$ in $O(n)$	$\frac{\lvert N(n)\rvert \text{ in } O(n)}{\lvert N(n)\rvert}$
	$n(n-1)/2$	$(n-2)(n-3)/2$	$2n-3$	
40	780	703	77	0.099
80	3160	3003	157	0.050
120	7140	6903	237	0.033
160	12720	12403	317	0.025
200	19900	19503	397	0.020

In this research, we assign move cost computations of a solution ϕ which are in $\mathcal{O}(1)$ and in $\mathcal{O}(n)$ to threads which belong to different warps in a block. Figure 2 shows the thread structure in a block in computing move costs for TS in this study. Hereafter, we refer to this thread structure as Move-Cost Adjusted Thread Assignment, or $MATA$ for short.

Since the computation of a move cost which is $\mathcal{O}(1)$ is smaller than the computation which is $\mathcal{O}(n)$, we assign multiple number N_S of computations which are $\mathcal{O}(1)$ to a single thread in the block. Also, it is necessary to assign multiple calculations of the move costs to a thread, because the maximum number of threads in a block is limited (1024 for GTX 480).

7. CONCLUDING REMARKS

In this study, we used a PC which has one Intel Core i7 965 (3.2 GHz) processor and a single NVIDIA GeForce GTX480 GPU. The instances on which we tested our algorithm were taken from the QAPLIB benchmark library [1]. QAP instances in the QAPLIB can be classified into 4 classes; (i) randomly generated instances, (ii) grid-based distance matrix, (iii) real-life instances, and (iv) real-life like instances [2]. In this experiment, we used the following 10 instances which were classified as either (i) ; tai40a, tai50a, tai60a, tai80a, tai100a, or (iv) tai50b, tai60b, tai80b, tai100b, and tai150b (the numbers indicate the problem size n). Here, note that instances classified into class (i) are much harder to solve than those in class (iv).

GPU computation with MATA showed a promising speedup compared to GPU computation without MATA (non-MATA) and computation with CPU. Please refer to [4] for more detail.

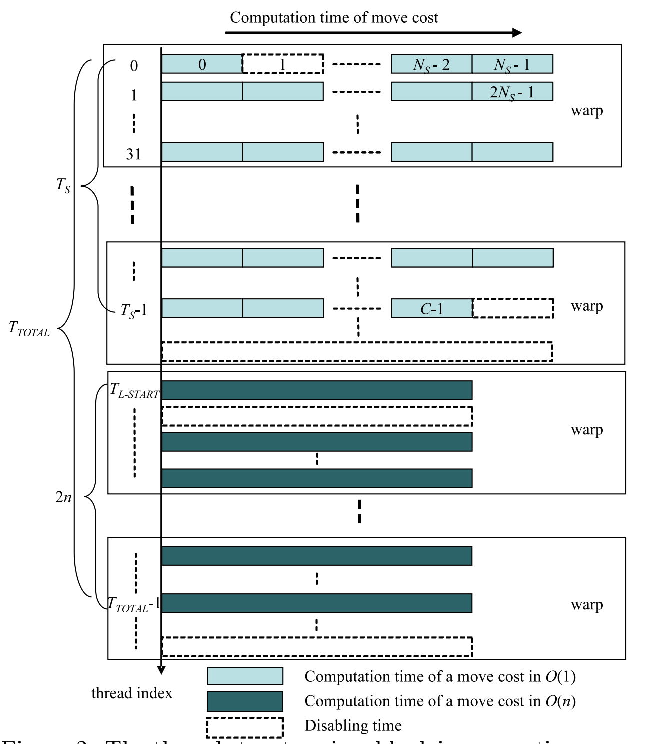

Figure 2: The thread structure in a block in computing move costs for TS (MATA)

8. REFERENCES

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

[2] É. Taillard. Comparison of iterative searches for the quadratic assignment problem. Location Science, 3(2):87–105, 1995.

[3] S. Tsutsui and N. Fujimoto. Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study. In Genetic and Evolutionary Computation Conference (Companion), pages 2523–2530. ACM, 2009.

[4] S. Tsutsui and N. Fujimoto. ACO with Tabu Search on a GPU for Solving QAPs using Move-Cost Adjusted Thread Assignment (presented at parallel evolutionary systems track of this gecco). In Genetic and Evolutionary Computation Conference. ACM, 2011.