First Results of Performance Comparisons on Many-core Processors in Solving QAP with ACO: Kepler GPU versus Xeon Phi

Mikiko Sato
Tokyo University of Agriculture and Technology, Tokyo, 184-8588 Japan
mikiko@namikilab.tuat.ac.jp

Shigeyoshi Tsutsui
Hannan University
Osaka 580-8502, Japan
tsutsui@hannan-u.ac.jp

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

Yuji Sato
Hosei University
Tokyo 184-8584 Japan
yuji@hosei.ac.jp

Mitaro Namiki
Tokyo University of Agriculture and Technology
Tokyo, 184-8588 Japan
namiki@cc.tuat.ac.jp

ABSTRACT
This paper compares the performance of parallel computation on two types of many-core processors, Tesla K20c GPU and Xeon Phi 5110P, in solving the quadratic assignment problem (QAP) with ant colony optimization (ACO). The results show that the performance on Xeon Phi 5110P is not so promising compared to the Tesla K20c GPU on these problems. Further efficient implementation methods must be investigated for Xeon Phi.

Categories and Subject Descriptors
I.2.8 [Articial Intelligence]: Problem Solving, Control Methods, and Search–Heuristic Methods; D.1.3 [Programming Techniques]: Concurrent Programming–Distributed programming, Parallel programming

General Terms: Algorithms

Keywords
Parallel EA, GPU, Xeon Phi, ACO, QAP, Tabu search

1. INTRODUCTION
Applications of evolutionary algorithms (EAs) often take a long time to obtain acceptable solutions, or require a large amount of computational resources to obtain high-quality solutions in a given time. Thus, since the earliest studies, researchers have attempted parallel EAs in order to obtain high-speed execution of EAs.

Many of the traditional parallel EAs run on multi-core machines, massively parallel cluster machines, or grid computing environments. However, recent advances in General-Purpose computing on Graphics Processing Units (GPGPU) in scientific computing has made possible using Graphics Processing Units (GPUs) for parallel EAs. GPUs are low-cost, parallel, many-core processors and thousands of threads run in parallel in SIMD manner on a GPU. With the low-cost GPU found on ordinary PCs, it is becoming possible to use parallel EAs to solve optimization problems of all sizes [1].

In 2012, Intel® Xeon® Phi™ coprocessors, another type of many-core processor, became available. Xeon® Phi™ has 60 cores based on x86 architecture and can execute a maximum of 240 (60×4) threads in parallel in MIMD (multiple instruction, multiple data) manner [2]. This enables us to re-use existing programs which are written for parallel execution without a major re-coding.

In this paper, we compare performance of these two types of representative many-core units, NVIDIA® Tesla® k20c GPU (based on the Kepler architecture) and Intel® Xeon® Phi™ 5110P in solving the quadratic assignment problem (QAP) with ant colony optimization (ACO), a meta heuristics classified into EA.

2. ACO TO SOLVE QAP
The quadratic assignment problem (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

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

where $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices and ϕ is a permutation of $\{1, 2, 3..., n\}$. 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 optimization problems. In this experiment, we use benchmark instances in QAPLIB [3].

ACO has been applied with great success to a large number of hard problems. Although the ACO is a powerful metaheuristic, in

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2602274

many applications of ACO in solving difficult problems, it is very common to combine it with a local search or metaheuristics. In this experiment, we combine ACO with the tabu search (TS) [5]. TS seeks the solution with the best evaluation among all the neighboring solutions in each TS iteration. If there are no improving moves, TS chooses one that least degrades the objective function. Thus, we need to calculate costs of all neighboring 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 $\Delta(\phi, i, j) = f(\phi')-f(\phi)$ for all the neighboring solutions. The neighborhood size of $N(\phi)$ ($|N(\phi)|$) is $n(n-1)/2$ where n is the problem size. Thus, the computation cost of $\Delta(\phi, i, j)$ is very costly.

3. IMPLEMENTATION OF ACO WITH TS ON TESLA® K20C AND XEON® PHI™ 5110P

Table 1 shows specifications of both Tesla® K20c and Xeon® Phi™ 5110P [4].

Table 1 Specifications of Tesla K20c and Xeon Phi 5110P

	Xeon Phi 5110P	Tesla K20c
Performance (Single precision)	2022 Gflops	3520 Gflops
Memory Bandwidth	320 GB/s (ECC off)	208 GB/s (ECC off)
Memory Size	8 GB	5 GB
Clock Speed	1.053 GHz	0.706 GHz
Number of Core	60	2496

Figure 1 shows the implementation of ACO described in Section 2 on Tesla® K20c. Here, all of the programs are executed on the GPU. Each calculation of $\Delta(\phi, i, j)$ in TS is assigned to the thread of the streaming multiprocessor of CUDA, and is executed in parallel (please refer to [5] for detail). Note here that all of the data of the algorithm are located in VRAM of GPU. They include ACO data (the population pools) the pheromone density matrix, TS data, and QAP data.

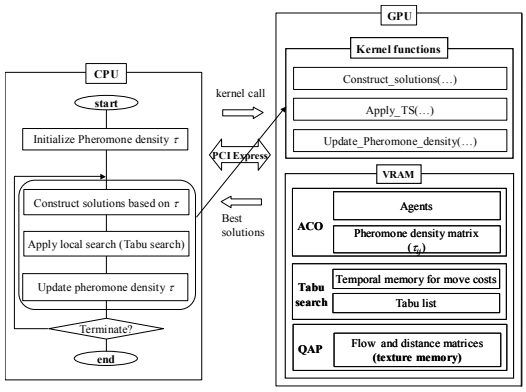

Figure 1 Implementation of ACO with TS on GPU

In the implementation using Intel Xeon® Phi™, almost all processing of ACO is offloaded to Xeon Phi, in a similar manner to the GPU method shown above. The program run on Xeon Phi and data transfer are specified by #pragma directive. First, the data which are needed for ACO processing, i.e., ACO data, the pheromone density matrix, TS data, and QAP data, are transferred to the memory on Xeon Phi. Then each individual of the population is assigned to a thread of Xeon Phi. This includes calculation of $\Delta(\phi, i, j)$. Note here that updating pheromone density is performed as a single thread. The host CPU functions only to receive solutions from the Xeon Phi cores for each generation and determines the termination.

4. RESULTS AND CONCLUDING REMARKS

We compared three types of runs, i.e., runs on Tesla® K20c with Intel Core i7 965 (3.2 GHz) CPU, runs on Xeon® Phi™ 5110P with Xeon X5690 (3.333GHz), and runs on Intel Core i7 965 (3.2 GHz) CPU with a single thread. For all three runs, population size was set to n (problem size). We run the algorithms until the optimal solutions are obtained. We measured the performance by the average run time T_{avg} over 25 runs. The results are summarized in Table 2.

The speedup ratios of Tesla GPU to CPU are in the range from x27.2 to 47.6, showing their average is x33.5. In contrast to these results on the GPU, the speedup ratios of Xeon Phi to CPU are in the range from x3.8 to 6.6, showing their average is 5.3. These speedup values of Xeon Phi are not so promising compared to Tesla K20c GPU on this application.

In this application, about 99% computation time is used by TS to find the best neighboring solution in each TS iteration [5]. On GPU, this computation was parallelized using the threads of CUDA. However, since the grain of the computation is very fine, the parallelization is difficult on Xeon Phi. Instead, using very wide (512-bit) SIMD units in Xeon Phi might be useful for an efficient parallelization. But this remains for future work.

Table 2 Summary of results

QAP instances	Run time in T_{avg} (sec)			Speedup in T_{avg}	
	Tesla K20c	Xeon Phi	CPU	CPU / Tesla K20c	CPU / Xeon Phi
tai50b	0.19	1.14	5.94	31.7	5.2
tai60b	0.39	2.75	12.77	32.8	4.6
tai80b	4.99	22.61	141.30	28.3	6.2
tai100b	6.22	57.35	296.28	47.6	5.2
tai150b	14.06	61.92	382.73	27.2	6.2
Average	-	-	-	**33.5**	**5.5**

5. REFERENCES

[1] S. Tsutsui and P. Collet (Ed). Massively parallel evolutionary computation on GPGPU, Natural Computing Series, Springer, 2013.

[2] Intel. http://blogs.intel.com/technology/2012/06/intel-xeon-phi-coprocessors-accelerate-discovery-and-innovation/

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

[4] T. Aoki. http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&GakubuCD=226&GakkaCD=226717&KougiCD=77065&Nendo=2013&Gakki=1&lang=JA&vid=05

[5] S. Tsutsui and N. Fujimoto. ACO with Tabu Search on a GPU for Solving QAPs using Move-Cost Adjusted Thread Assignment, Genetic and Evolutionary Computation Conference, pp. 1547-1554, ACM, 2011