Parallel Particle Swarm Optimization on Many-Integrated-Cores Platforms

Yasuyuki Miyazato[1], Takeshi Tengan[2], Andrea Veronica Porco[3] and Morikazu Nakamura[4]
[1,3,4]Department of Information Engineering, University of the Ryukyus
1 Senbaru, Nishihara, Okinawa 903-0213, Japan
[2]Faculty of International Studies, Meio University,
1220-1 Biimata, Nago, Okinawa 905-8585, Japan
E-mail : [2]tengant@meio-u.ac.jp, [4]morikazu@ie.u-ryukyu.ac.jp

Abstract: This paper presents parallel particle swarm optimization (PSO) on Many-Integrated-Cores (MIC) architecture platforms and investigate how to utilize many cores of MIC platforms efficiently. We consider in our parallel PSO strategies for utilizing many cores from the viewpoints of depth and width of searching. Experimental results for a benchmark problem show the relation between searching performance and search strategies in the parallel PSO. We confirm that the balance between the width and depth in search strategies is essential for efficient searching.

Keywords—**Particle Swarm Optimization; PSO; Parallel Optimization; Many Integrated Cores**

1. Introduction

Particle Swarm Optimization (PSO), a promising algorithm for optimization problems, is inspired by social behavior of a group of living things such as fishes, birds, and insects. PSO is known as one of the population-based metaheuristics such as Genetic Algorithm (GA), Ant Colony Optimization (ACO). Many researchers reported that PSO is efficient as a heuristic algorithm and has broad application areas.

Many Integrated Cores (MIC) architectures are a new platform for high-performance computing[1]. For example, Intel Xeon Phi Coprocessor, a MIC platform, is getting much attention as an attractive parallel processing platform for dramatic performance gains of highly parallel processing application by its many cores and threads[2].

Parallelization of optimization algorithms and efficient utilization of massively parallel processing many-core hardware are essential for solving large-scale optimization problems. However, it is not straightforward to utilize many cores in effective searching, that is, larger parallelism may lead to being the worse quality of solutions.

This paper presents a parallel discrete binary PSO on Intel Xeon Phi coprocessor for large-scale optimization problems. Our parallel scheme utilizes many cores for efficient collaborative searching in PSO. In [6], we reported the first result of our research. In this paper, we improve the algorithm and perform a more detailed experimental evaluation of our method by solving well-known benchmark optimization problems, NK-landscape.

There are many research results for parallel PSOs; fine-grained PSO on GPU [7], parallel PSO on PC clusters [8], [9], and so on. However, they did not utilize many-cores platforms we consider in this paper. We need to design a parallel PSO been suitable for the MIC platforms since characteristics of platforms may give significant effects on its performance of the parallel program.

2. Preliminaries

This section explains briefly Intel Xeon Phi Coprocessor and NK Landscapes.

2.1 Intel Xeon Phi Coprocessor

Intel Xeon Phi is many-core coprocessor based on Intel Many Integrated Core architecture (Intel MIC). For example, Intel Xeon Phi Coprocessor 5110P has 60 cores, and a core is possible to process 4 threads simultaneously; therefore, the number of simultaneous possible threads is 240. Compared to GPU platforms such as NVIDIA Tesla, Intel Xeon Phi has the advantage of being able to utilize existing x86 software and CPU-like programmability. Parallel program models on GPU platforms are based on SIMT (Single Instruction Multiple Threads), while the Intel MIC architectures are available for MIMD (Multiple Instructions and Multiple Data). That is, more flexible parallel programs can be designed on the architectures.

2.2 NK Landscapes

NK landscapes (NK model) introduced by Stuart Kauffman [3] as a tunable mathematical model. The model is based on a chromosome, the parameter N and K represent the number of genes and the number of alleles, respectively.

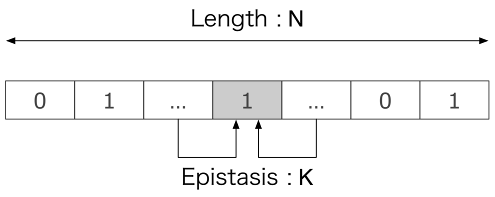

Figure 1. Chromosome in NK model

Consequently, the size and complexity of solution space to be generated can be adjusted by its parameters N and K.

The fitness value of a chromosome derived from the average of fitness component f_i determined by its own gene x_i and K alleles z_{ij}:

$$\begin{aligned} f(x) &= \frac{1}{N}\sum_{i=1}^{N} f_i(x_i, z_{i1}, \cdots, z_{iK}) \qquad (1) \\ & x_i \in \{0,1\}, \forall i, \\ & z_{ij} \in \{0,1\}, \forall i, j \end{aligned}$$

3. Discrete Binary PSO

A feasible solution of optimization problems corresponds to a particle in PSO. The position (coordinate) of a particle expresses directly a solution vector. PSO is a population-based, that is, groups of particles, called as *swarms*, search good quality of solutions by moving in the space [4].

The position and the velocity of a particle are updated by the equation, where it considers the best position the particle have ever visited, x^{pbest}, and the best position the entire PSO has ever visited, x^{gbest}.

$$\begin{aligned} v_i(k+1) &= w \cdot v_i(k) \\ &+ C_1 \cdot r_1 \cdot (x^{pbest}(k) - x_i(k)) \\ &+ C_2 \cdot r_2 \cdot (x^{gbest}(k) - x_i(k)) \quad (2) \\ x_i(k+1) &= x_i(k) + v_i(k+1), \quad (3) \end{aligned}$$

where w is inertia weight, C_1 and C_2 are acceleration coefficients, r_1 and r_2 are uniform random numbers on [0, 1].

To avoid premature convergence in searching, we use commonly the best position a particle in a sub-swarm visited instead of the best one in the entire swarm. Moreover, the neighborhood-based updating is also efficient, in which x^{gbest} means the best position a particle in the neighborhood visited so far. The neighborhood relation can be denoted as network topology, such as *ring*, *line*, *tree*, and *complete graph*. Figure 2 shows some examples of swarm topologies in PSO.

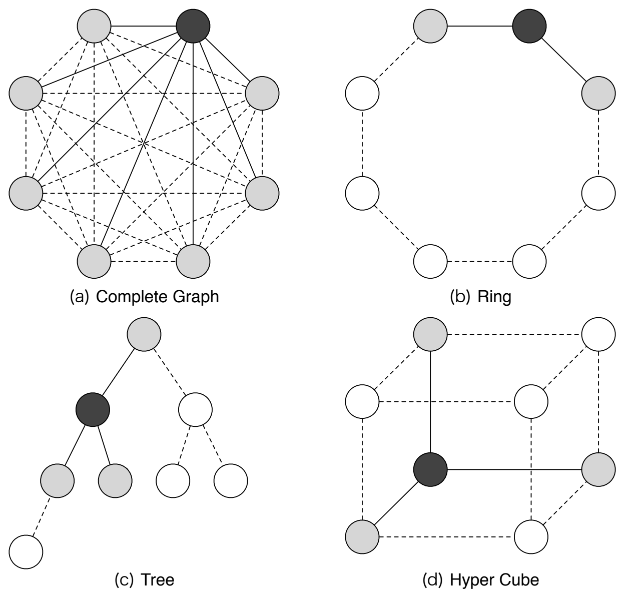

Figure 2. Example of typical neighborhood topologies used in PSO

Since particles in PSO traverse in the real-valued space, we need to convert the real-valued vectors into binary integers for evaluating the objective function. In our algorithm we use sigmoid functions $S(x_i)$:

$$x_i = \begin{cases} 1 & if(rand() < S(x_i)) \\ 0 & otherwise \end{cases} \quad (4)$$

$$S(X_i) = \frac{1}{(1+e^{-x_i})} \quad (5)$$

4. Parallel Discrete Binary PSO

In this paper, we parallelize the discrete binary PSO to take advantage of the MIC architecture. PSO possesses natural parallelism since it is a population-based algorithm. In our parallel discrete binary PSO, each thread in the shared memory programming model performs the role of a particle in a swarm. We assign a network topology which shows communication links between particles for the neighborhood-based updating in PSO.

This parallelizing is straightforward, and it seems to be suitable for the MIC architecture. More particles lead to wider width searching since particles and threads are the one-to-one correspondence. However, wider width searching does not always result in more efficient searching. Note that resources, processor cores and computation time, are not unlimited. We should always require efficient searching, that is, we need to get a better solution in a small number of evaluations in optimization.

In addition to *width*, *depth* should be an essential point for efficient searching on the MIC architecture. That is, we assign many cores to not only the width but also the depth of searching. Figure 3 explains the difference between width and depth, where two threads run two independent particles in the left side, while two threads at the right side perform different depth of a single particle.

Algorithm 1 represents our parallel PSO. The number of particles is the same as the number of threads. All the particle execute in parallel the **forall** statements and communicate with their neighbors to update the velocity.

Algorithm 1 Parallel PSO

```
1:  for all particle $i$ do
2:      Initialize $\mathbf{x}_i$ with a uniformly distributed random vector
3:      Initialize $\mathbf{x}_i^{pbest}$ and $\mathbf{x}_i^{gbest}$ by $\mathbf{x}_i$
4:      Initialize $\mathbf{v}_i$ with a uniformly distributed random vector
5:  end for
6:  for all particle $i$ do
7:      repeat
8:          Update the velocity $\mathbf{v}_i$ by equation (2)
9:          Update the position $\mathbf{x}_i$ by equation (3)
10:         if $f(\mathbf{x}_i) > f(\mathbf{x}_i^{pbest})$ then
11:             $\mathbf{x}_i^{pbest} \Leftarrow \mathbf{x}_i$
12:         end if
13:         Update $\mathbf{x}_i^{gbest}$ by communicating with the negihbors
14:     until The termination criterion is met
15: end for
```

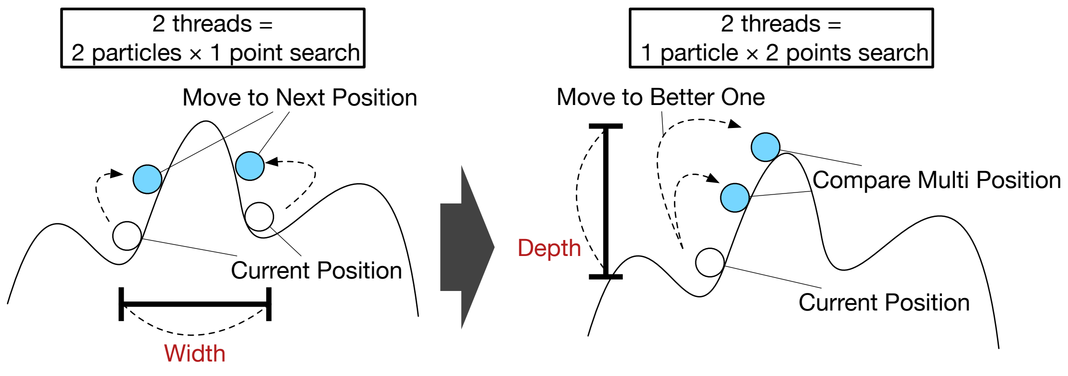

Figure 3. Parallel Discrete Binary PSO (Multi Point Search)

5. Experimental Evaluation

The parallel algorithm was coded using C language with the pthread library. We performed experiments for evaluation where we ran our parallel programs to solve NK landscape instances with fixed $K = 5$, varying N from 20 to 60, and the number of threads from 2 to 256. For each case of N and K, we averaged the obtained solutions from 100 times runs since we generated 5 different instances for each case and solved 20 times for each instance.

Firstly, we compared solution quality for four topologies, complete graph, ring, tree, hypercube, by varying problem size, and the number of iterations. Figures 4, 5, and 6 depict curves of the ratio of the solution quality compared to the quality obtained by a single particle's PSO for the number of iterations, 10,000, 2,500, and 625, respectively. Note that the total number of evaluations of the objective function is the same for all the cases, 80,000 times. It means that we ran our program for three cases of the parallel PSO; 10,000 iterations of 8 particles, 2,500 iterations of 32 particles, and 625 iterations of 128 particles, and for the case; 80,000 iterations of a single particle.

In the results shown in the figures, we observed that (1) the parallel PSOs performed better compared to the single particle searching, (2) complete graph communication led to premature convergence more than the other topologies, (3) the solution quality went down when N increased, this became more evident when the number of iterations were smaller, and complete graph was received less influence from this.

Figure 7 shows results for depth =1 and Fig. 8 for depth=2. The horizontal axis represents the number of threads and the vertical one the difference of the fitness value from the serial PSO. That is, the horizontal line at the fitness value 0 shows the solution quality of the serial PSO. Therefore, larger values than 0 mean better solution quality than the serial PSO.

We observe in the results that (1) the parallelization contributed to solution quality generally, (2) larger number of threads lead to worse quality of solutions than the serial PSO in case of larger problems, and (3) searching with depth 2 is better than one with depth 1 for larger problems.

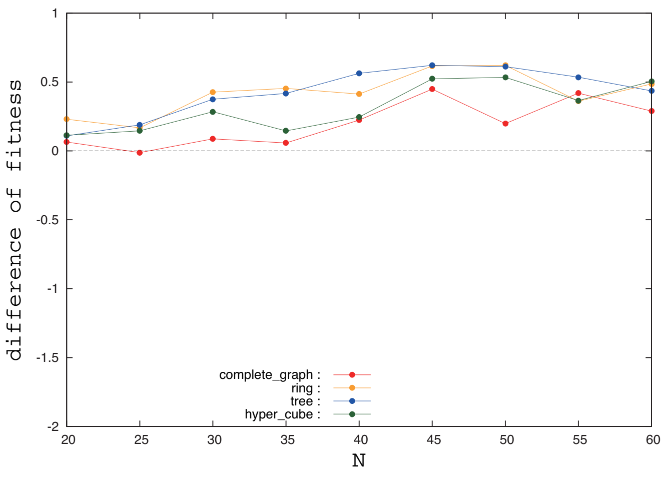

Figure 4. Solution Quality Comparison vs. N for Topologies (K = 5, Iterations = 10,000)

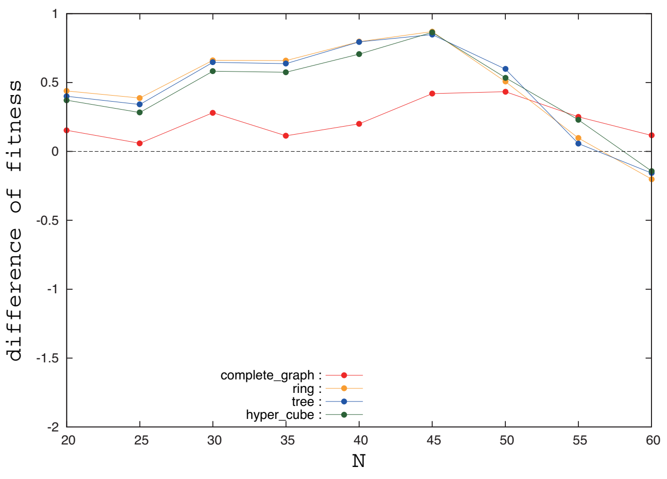

Figure 5. Solution Quality Comparison vs. N for Topologies (K = 5, Iterations = 2,500)

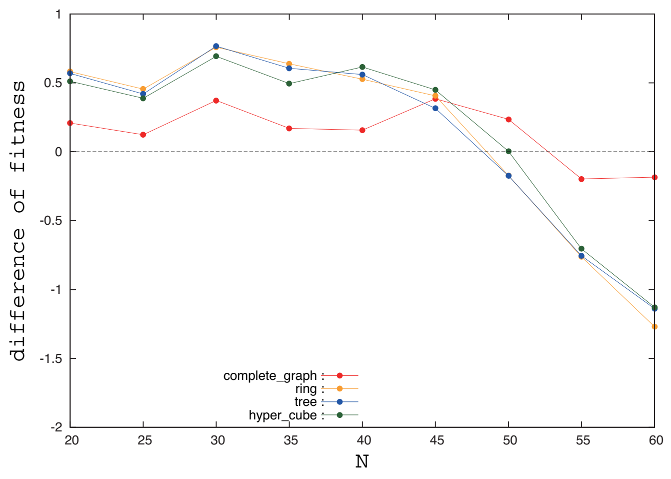

Figure 6. Solution Quality Comparison vs. N for Topologies (K = 5, Iterations = 625)

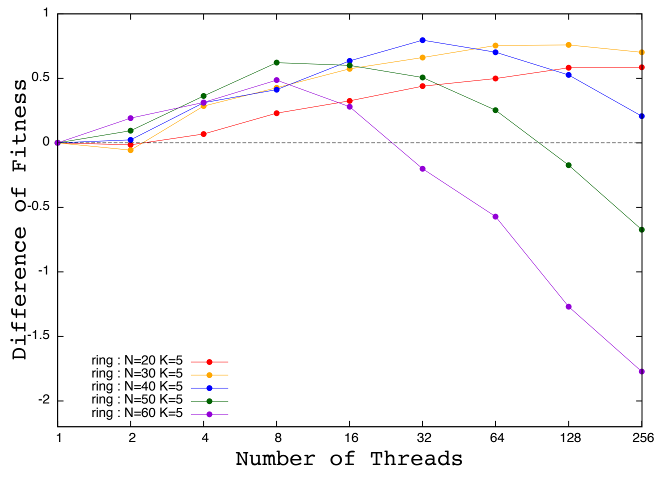

Figure 7. Parallel Discrete Binary PSO (Depth 1) with Ring

6. Concluding Remarks

In this paper, we presented parallel particle swarm optimization (PSO) on Many-Integrated-Cores (MIC) architecture platforms. To investigate how to utilize many cores of MIC platforms efficiently, we observed the relation between the searching performance and the width and depth of searching in the parallel PSO. We confirmed that the balance between the width and depth in search strategies is essential for efficient searching.

As future works, we will improve our algorithm so that it could search deeply more efficiently with multiple cores and will perform more detailed experimental evaluation.

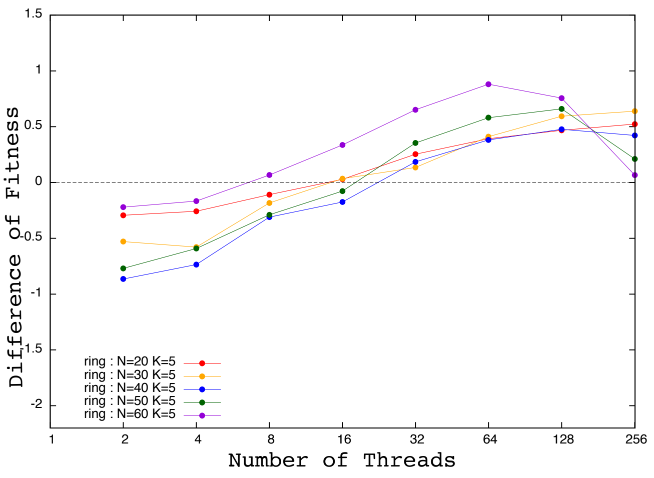

Figure 8. Parallel Discrete Binary PSO (Depth 2) with Ring

References

[1] Alexander Heinecke, Michael Klemm and Hans-Joachim Bungartz, "From GPGPU to Many-Core: Nvidia Fermi and Intel Many Integrated Core Architecture", Computing in Science & Engineering, vol.14, no.2, pp.78-83, 2012.

[2] Erik Saule, Ümit V. Çatalyürek, "An Early Evaluation of the Scalability of Graph Algorithms on the Intel MIC Architecture", Proc. of 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp.1629-1639, 2012.

[3] Stuart A. Kauffman, Edward D. Weinberger "The NK Model of Rugged Fitness Landscapes And Its Application to Maturation of the Immune Response", *Journal of Theoretical Biology*, Vol. 141, Issue 2, pp.211-245, 1989.

[4] James Kennedy, Russell Eberhart "Particle Swarm Optimization", *Encyclopedia of Machine Learning*, pp.760-766, 2010.

[5] James Kennedy, Russell Eberhart "A Discrete Binary Version of the Particle Swarm Algorithm", *IEEE International Conference On Systems, Man, And Cybernetics*, Vol. 5 pp.4104-4108, 1997.

[6] Yasuyuki Miyazato, Morikazu Nakamura, "Strategy Comparison in Particle Swarm Optimization on Many-Integrated-Cores Platforms" *Proceedings of the International Conference on Intelligent Informatics and Biomedical Sciences*, 2015.

[7] Jianming Li, Danling Wan, Zhongxian Chi, and Xiangpei Hu, "An Efficient Fine-Grained Parallel Particle Swarm Optimization Method based on GPU-Acceleration", International Journal of Innovative Computing, Information and Control, vol. 3, 6(B), pp.1707-1714, 2007.

[8] Jong-Yul Kim, Hee-Myung Jeong, Hwa-Seok Lee, and June-Ho Park, "PC Cluster based Parallel PSO Algorithm for Optimal Power Flow", Proc. of International Conference on Intelligent Systems Applications to Power Systems, pp. 1-6, 2007.

[9] Jong Yul Kima, Kyeong Jun Munb, Hyung Su Kimc, June Ho Park, "Optimal Power System Operation using Parallel Processing System and PSO algorithm", International Journal of Electrical Power & Energy Systems, vol.33, no.8, pp.1457-1461, 2011.