2015 International Symposium on Nonlinear Theory and its Applications
NOLTA2015, Kowloon, Hong Kong, China, December 1-4, 2015

# GPU Implementation and its Evaluation of Particle Swarm Optimization

Satoshi Kondo†, Kenichiro Tanaka‡, and Yuichi Tanji†‡

Department of Electronics and Information Engineering,
Kagawa University,
Hayashi-cho 2217-20, Takamatsu, 761-0396 Japan
Email: †s15g466@stu.kagawa-u.ac.jp, ‡s14g468@stmail.eng.kagawa-u.ac.jp,
†‡tanji@eng.kagawa-u.ac.jp

**Abstract–** Recently, GPU is used for general purpose of computing. In this study, we implement particle swarm optimization (PSO) algorithms, which are applicable to various engineering applications, on GPU, and evaluate the performance using benchmark functions. Also we compare PSO with one with stochastic coupling.

## 1. Introduction

GPU is a graphic controller that is dedicated to display images to screen for PC, but it is used for general purpose of computations [1], [2]. Since the parallel feature with many high speed core processors enables us to process a large amount of data. As one of the most imperative computations that need heavy computation power, optimization algorithms are picked up.

In this paper, we focus how to accelerate PSO algorithm. PSO was proposed by Kennedy and Eberhart in 1995, and it is a metaheuristic optimization algorithm inspired by swarm intelligence of birds and fish [3]. PSO has generated much wider interests due to the simple concept and implementation easiness. The PSO algorithm consists of multiple individual agents called "particles" and searches the solution space of an objective function by the particles moving around. Each particle flies toward the position of the current global best and its own best position in history. To find a good solution in wide searching area over many dimensions, many particles would be necessary. Then, the finding process would require huge computational efforts

To reduce the computation cost, we implement PSO on GPU. Finding process of PSO is called dynamics that is expressed by difference equations. Since the dynamics of a particle is independent to another particle, it is computed in parallel to another particle. This means that all the dynamics can be calculated efficiently on GPU. However, the selection of global best position is essentially serial computation, which requires many access to global memory of GPU. To reduce the number of access, we introduce lbset PSO which has sparse connection with neighboring particles. Moreover, robustness of PSO with stochastic coupling is known [4], [5]. Hence, we also introduce the PSO with stochastic coupling.

In the illustrative examples using benchmark functions, we confirm that the GPU implementation is maximally 20 times faster than the CPU.

## 2. GPU and CUDA

CUDA, which is an integrated development environment for GPU, is provide by NVIDIA [2]. Using a large number of computing cores within the processor, it becomes recently possible to use for general-purpose computations with dramatically faster speed. A program of CUDA consists of host and device codes operated on CPU and GPU, respectively.

Shared and global memories can be used in CUDA. Shared memory is shared by many threads and global memory can be accessed by not only threads but host as shown in Fig. 1. Although shared memory is fast, the capacity is small and up to 16KB per block is available. On the other hand, capacity of global memory is large, but the accessible speed is slow. All threads can read from and write into global memory. Therefore, if a specific addressed content in global memory is accessed several times, the users should use shared memory as a cache in order to accelerate the operations.

In CUDA, up to 65535 × 65535 × 512 threads can be run with streaming processor of GPU. To process such a large number of threads effectively, the concept of grid and blocks is introduced, and threads are managed hierarchically.

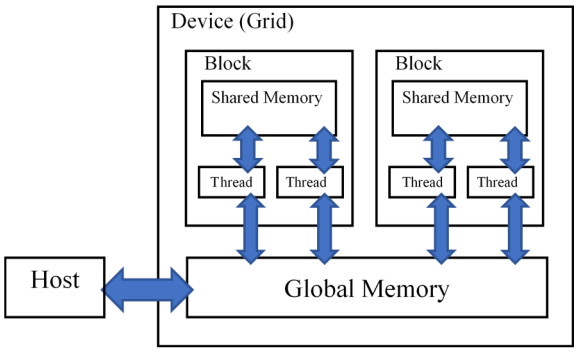


Figure 1: Memory model of CUDA

## 3. GPU Implementation

### 3.1. PSO

The dynamics of PSO is described by

$$x_{ij}(k+1) = x_{ij} + v_{ij}(k+1), \quad (1)$$

$$v_{ij}(k+1) = \omega v_{ij} + \varphi_1 U_{(0,1)}\big(P_{ij}(k) - x_{ij}(k)\big) + \varphi_2 U_{(0,1)}\big(P_{gj}(k) - x_{ij}(k)\big), \quad (2)$$

In (1) and (2), $v_{ij}$ is the velocity, $x_{ij}$ is the position, $\varphi_1, \varphi_2$ are acceleration coefficients, and $\omega$ is an inertia weight. $U_{[0,1]}$ is a random number that is uniformly distributed in the range [0, 1] for each dimension. $P_{ij}\ (k)$ is a minimum point (personal best position) that each particle is ever found. $P_{gj}\ (k)$ is a minimum point (global best position) that is discovered by the swarm so far.

PSO does not require differentiability, which implies that it is a direct search algorithm capable of solving nonlinear optimization problems by only information of the objective function. However, for multimodal function the swarm is caught to a sub-optimal solution. Hence, probability binding to the swarm is introduced in order to overcome this issue [4], [5].

### 3.2. Implementation

With the structural features of GPU, we can process a large amount of data simultaneously. It is possible to calculate (1) and (2) in parallel. Then, we assign each particle to a thread, and the dynamics expressed by (1) and (2) is calculated in parallel to another particle. However, since the minimum point $P_{gj}\ (k)$ for the swarm has the star structure as shown in Fig. 2, the selection becomes a sequential processing basically and is done after all the threads are synchronized. Then, efficiency of the computations can be lowered. Hence, we introduce a ring-shaped coupling as shown in Fig. 3(a), which is called lbest PSO. In lbest-PSO, the minimum point (locally global best) from the three adjacent particles including itself is extracted. Therefore, the selection of minimum point is easier than that in standard PSO, and we do not have to halt all the threads to select the global best. Hence, we use lbset PSO and implement it on GPU.

To avoid being trapped to a local minimum, the coupling in (2) is stochastically changed at every iteration. Although this idea is realized in IPSO [4], [5], it is too complex to implement it on GPU. Therefore, in order to take advantage of the feature of IPSO, we consider a random connection shown in Fig. 3(b). Although a particle in lbest PSO is connected to the right and left neighboring particles, it is coupled to two particles randomly. Using such connections, we expect that a better solution can be obtained. Since the PSO has random coupling, we call it stochastically connected PSO (SCPSO).

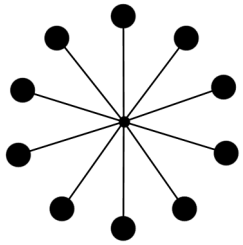

Figure 2: Relationship between each particle and the global best position.

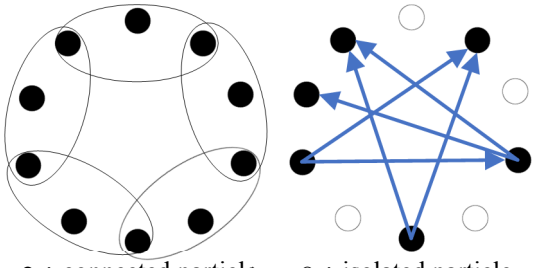


Figure 3: PSOs used for GPU implementation. (a) lbest PSO. (b) SCPSO.

## 4. Results

We implemented lbest-PSO and SCPSO on GPU and evaluated by using the objective functions given in Table 1. The environments used for evaluation are shown in Table 2. The computation times using CPU and GPU are respectively shown in Table 3, where lbest PSO was used. The comparison of computation times for SPSO is given in Table 4. In the results of Tables 3 and 4, 10,000 iterations were done for both PSOs with the Sphere function. The computation times were measured on Env. 1 in Table 2. For all the dimensions of dependent variables of objective functions, the results of CPU with 10 particles were faster than those of GPU. The computation times by GPU were compatible to the CPU for 16 × 16 particles, they were slower than those of CPU except for one example.

For the case of 256 × 256 particles, the GPU implementation for both lbest-PSO and SCPSO was about 20 times faster than the CPU. However, efficiency was reduced when dimension is 20. This is because access to global memory increases with higher dimensions.

Figures 4 and 5 show the convergence curves of lbest PSO and SCPSO for the Sphere function. The convergence of GPU is slightly different from that of GPU. The difference relies on each single-precision floating point operation. For lbest PSO in Fig. 4, all the trials are converged. Although convergence has been achieved for the trial of 5 dimensions for SCPSO as shown in Fig. 5, minimum solutions are not obtained for the trials for 10, 15, and 20 dimensions. PSOs with stochastically coupling are generally more robust than standard PSO in searching ability. However, lbest-PSO with stochastic coupling presented as Fig. 3(b) could not find a good solution. This means that we need to contrive ways to couple.

Figures 6 and 7 show the convergence curves of lbest PSO and SPSO, respectively, for the Rosenbrock function. The convergence curve of GPU is largely different from that of CPU. This situation also appeared for the 3rd De Jong function. Using another environment (Env 2. in Table 2), we wrote the convergence curve for the Ronsenbrock function. This result is shown in Fig. 8, where a large difference in convergence between CPU and GPU was not observed. From this fact, the differences in convergence in Figs. 6 and 7 would depend on the environment used for evaluation.

## 5. Conclusions

We have presented implementation of particle swarm optimization algorithms on GPU and evaluated it using benchmark functions. As a result, we could achieve 20 times speed-up in comparison to the CPU. Also we evaluated PSO with stochastic coupling, but could not confirm its superiority. In the near future, we will improve the performance of PSO with stochastic coupling.

## References


[1] NVIDA: http://www.nvidia.com/page/home.html

[2] T. Aoki and A, Nukada, *First CUDA Programming*, Kougakusha, 2009 (in Japanese).

[3] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization –an overview," Swarm Intell, vol. 1, pp. 33-57, 2007.

[4] H. Matsusita, Y. Nishio and T. Saito "Particle swarm optimization with novel concept of complex network," in Proc. of NOLTA'10, pp. 197-200, Sep. 2010.

[5] H. Matsushita, Y. Nishio and T. Saito, "Behavior of independent-minded particle swarm optimization," in Proc. NCP'11, pp.103-106, Mar. 2011.


Table 1: Test objective functions used for evaluation.

| | |
|---|---|
| Sphere | $f(x) = \sum_{d=1}^{D} (x_d^2 - 1) \quad x \in [-5.12, 5.12]$ |
| 3rd De Jong | $f(x) = \sum_{d=1}^{D} \lvert x_d - 1 \rvert \quad x \in [-2.048, 2.048]$ |
| Rosenbock | $f(x) = \sum_{d=1}^{D} 100\left(x_d^2 - x_{d+1}\right)^2 + (1 - x_d)^2 \quad x \in [-5.12, 5.12]$ |

Table 2: Environments used to evaluate the performance of GPU implementation.

| | Env. 1 | Env. 2 |
|---|---|---|
| CPU | Intel® Core™ i7 | Intel® Core™ i5 |
| Memory | 5954 MB | 4096 MB |
| GPU | Tesla C2050 | GeForce 9800GT |
| Global Memory | 2.62 GB | 512 MB |
| Shared Memory | 48 KB | 16 KB |
| Clock Rate | 1.15 GHz | 1.5 GHz |

Table 3 Comparison of computation times for lbest PSO.

| Dimensions | Particles | CPU [sec.] | GPU [sec.] | CPU/ GPU |
|---|---|---|---|---|
| 5 | 10 | 0.07 | 2.26 | 0.03 |
| | 16×16 | 1.36 | 2.56 | 0.53 |
| | 256×256 | 357.2 | 17.85 | 20.01 |
| 10 | 10 | 0.08 | 2.29 | 0.03 |
| | 16×16 | 2.79 | 2.83 | 0.99 |
| | 256×256 | 705.96 | 30.69 | 23.00 |
| 15 | 10 | 0.15 | 2.29 | 0.07 |
| | 16×16 | 3.60 | 3.19 | 1.13 |
| | 256×256 | 948.36 | 48.48 | 19.56 |
| 20 | 10 | 0.150 | 2.30 | 0.07 |
| | 16×16 | 3.16 | 3.64 | 0.87 |
| | 256×256 | 1127.67 | 77.32 | 14.58 |

Table 4: Comparison of computation times for SCPSO.

| Dimensions | Particles | CPU [sec] | GPU [sec] | CPU /GPU |
|---|---|---|---|---|
| 5 | 10 | 0.080 | 2.280 | 0.04 |
| | 16×16 | 1.190 | 2.590 | 0.46 |
| | 256×256 | 358.550 | 19.430 | 18.45 |
| 10 | 10 | 0.110 | 2.280 | 0.05 |
| | 16×16 | 2.010 | 2.820 | 0.71 |
| | 256×256 | 690.140 | 32.890 | 20.98 |
| 15 | 10 | 0.150 | 2.280 | 0.07 |
| | 16×16 | 3.470 | 3.160 | 1.10 |
| | 256×256 | 856.560 | 51.700 | 16.57 |
| 20 | 10 | 0.180 | 2.300 | 0.08 |
| | 16×16 | 4.700 | 3.620 | 1.30 |
| | 256×256 | 789.510 | 77.849 | 10.14 |

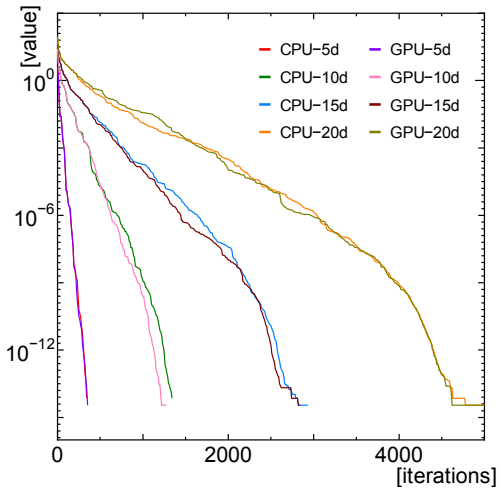


Figure 4: Convergence curves for lbest PSO (Sphere).

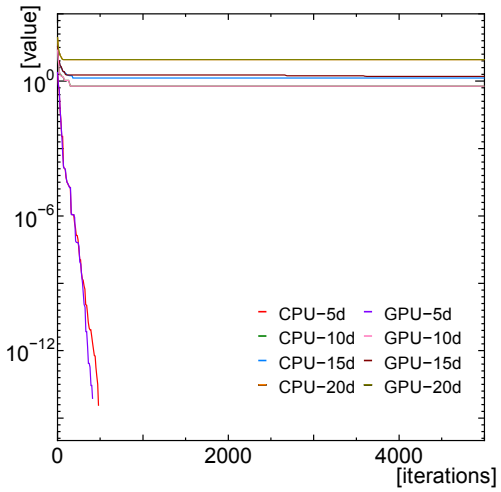


Figure 5: Convergence curves for SCPSO (Sphere).

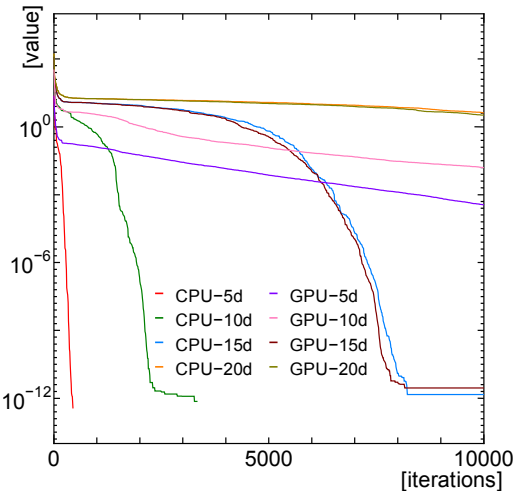


Figure 6: Convergence curves for lbest PSO (Rosenbrock).

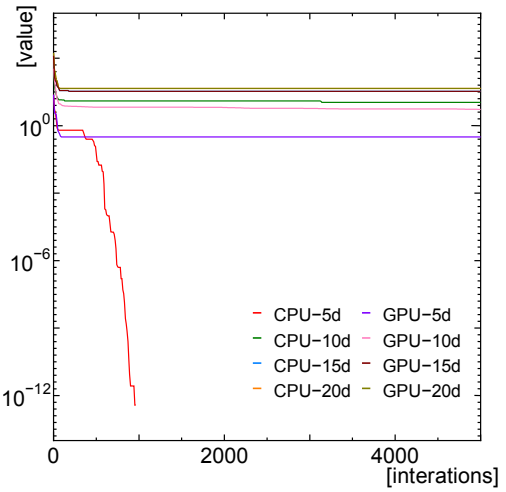


Figure 7: Convergence curves for SCPSO (Rosenbrock).

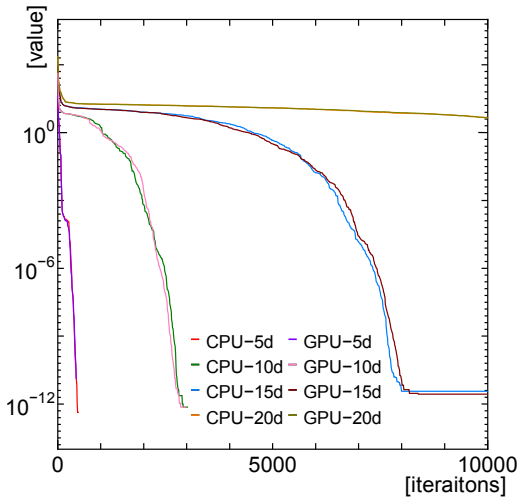


Figure 8: Convergence curves for SCPSO (Rosenbrock).