**11th World Congress on Structural and Multidisciplinary Optimisation**
07th -12th, June 2015, Sydney Australia

# Parallel Particle Swarm Optimization on GPU with Application to Trajectory Optimization

**Qi Wu [1], Fenfen Xiong[2]**

[1] Beijing Institute of Technology, School of Aerospace Engineering, Beijing,wuqi_55@163.com
[2]Beijing Institute of Technology, School of Aerospace Engineering, Beijing, fenfenx@bit.edu.cn. Corresponding author

## Abstract

In simulation-based design optimization, one of the greatest challenges is the intensive computing burden. In order to reduce the computational time, a parallel implementation of the particle swarm optimization (PSO) algorithm on graphic processing unit (GPU) is presented in this paper. Instead of executed on the central processing unit (CPU) in a serial manner, the PSO algorithm is executed in parallel taking advantage of the general-purpose computing ability of GPU in the platform of compute unified device architecture (CUDA). The processes of the fitness evaluation, the updating of velocity and position of all the particles of PSO are parallelized and respectively introduced in detail. Comparative studies on optimization of three benchmark test functions are conducted by running the PSO algorithm on GPU (GPU-PSO) as well as CPU (CPU- PSO), respectively. The impact of design dimension, as well as the number of particles and optimization iteration in PSO on the computational time is investigated. From test results, it is observed that the computational time of GPU-PSO is much shorter compared to that of CPU- PSO, which demonstrates the remarkable speedup capability of GPU-PSO. Finally, GPU-PSO is applied to a practical gliding trajectory optimization problem to reduce the computing time, which further demonstrates the effectiveness of GPU-PSO.

**Keywords:** PSO; GPU; CUDA; Trajectory Optimization

## 1. Introduction

Particle swarm optimization (PSO) developed by Kennedy *et. al.* in 1995 is an intelligent random global optimization algorithm inspired by the social behaviour of bird flocking or fish schooling [1]. In PSO, each particle in the swarm adjusts its position in the search space based on the best position it has found so far as well as the position of the known best-fit particle of the entire swarm, and finally converges to the global best point in the whole search space. Due to its easy implementation and competitive performances, the PSO algorithm has been extensively applied to optimization of very complex functions in a wide range of applications [2]. However, since the optimizing process of PSO requires a large number of fitness evaluations in the whole search space, it takes a long time for PSO to find optimal solutions especially for problems with high dimension or that needs a large swarm population for search. This becomes more serious when the performance functions are highly computational expensive. Traditionally, the fitness evaluations in PSO are done in a sequential way on the central processing unit (CPU). Thus, the computing speed of PSO may be quite slow for practical applications.

At present, it is difficult to improve the computing speed of PSO from the viewpoint of algorithm. Meanwhile, it may reduce the computing accuracy. As a traditional graphics-centric workshop, the graphics processing unit (GPU) shows faster float-point operation and higher memory bandwidth in scientific computing fields compared to CPU [3]. Through integrating CPU and GPU and taking advantages of both, the heterogeneous computing technique has become a research focus for computational speedup in recent years. The compute unified device architecture (CUDA) developed by NVIDIA corporation is a famous platform for heterogeneous computing, which has greatly simplified programming on GPU and been applied to lots of general computing [4]. In order to reduce the computational time of PSO, a parallel implementation of the PSO algorithm based on GPU in CUDA is developed, named as GPU-PSO for short in this paper, which greatly speeds up the computing.

## 2. Review of PSO

PSO is a stochastic global optimization technique inspired by the social behaviour of bird flocking or fish schooling. With PSO, each particle in the swarm adjusts its position in the search space based on its best position found so far as well as the position of the known best-fit particle of the entire swarm, and finally converges to the global best point. The search of the whole design space is done by a swarm with a specific number of particles. During each of the optimization iteration, the position and velocity of each particle are both updated according to its current best position ( $P_{gDb}(t)$ ) and best position of the entire swarm ( $P_{pDb}(t)$ ). The position $X_{ij}$ and velocity $V_{ij}$ of each particle on one dimension are updated as follows:

$$\begin{aligned} V_{ij}(t+1) &= wV_{ij}(t) + c_1 r_1 \left(P_{pDb}(t) - X_{ij}(t)\right) + c_2 r_2 \left(P_{gDb}(t) - X_{id}(t)\right), \\ X_{ij}(t+1) &= X_{ij}(t) + V_{ij}(t), \quad i = 1,2,...,n;\ j = 1,2,...,d. \end{aligned} \tag{1}$$

where $n$ is the number of particles in the swarm; $d$ is the optimization dimension; $c_1$ and $c_2$ are learning factors, which are non-negative constants; $r_1$ and $r_2$ are random numbers uniformly located in the interval [0, 1]; $w$ is the inertia weight used to balance the global and local search abilities of PSO, which is a constant lies between 0 and 1; $V_{ij} \in [-V_{\max},\ V_{\max}]$ is the velocity on the $j^{th}$ dimension of the $i^{\text{th}}$ particle with $V_{max}$ as a constant pre-specified according to the objective function. If $|V_{ij}| \geq V_{max}$, it will be set as $V_{ij} = V_{\max}$ or $V_{ij} = -V_{\max}$. The convergence rate is impacted by $V_{max}$, which can avoid premature of PSO. The general procedure of the PSO algorithm is described step by step in Figure 1 as below.

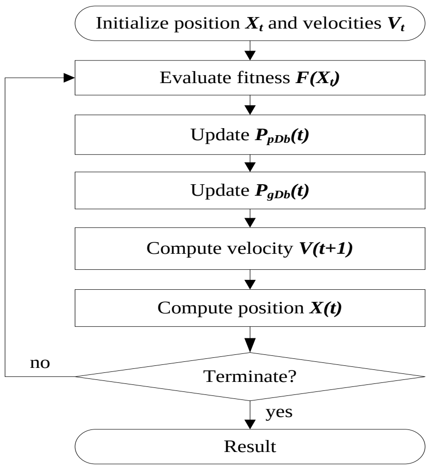

Initialize position $X_t$ and velocities $V_t$
Evaluate fitness $F(X_t)$
Update $P_{pDb}(t)$
Update $P_{gDb}(t)$
Compute velocity $V(t+1)$
Compute position $X(t)$
no
Terminate?
yes
Result

Figure 1: Flowchart of the particle swarm optimization algorithm

## 3. Introduction of GPU based parallel computing

GPU was originally designed especially for the purpose of image and graphic processing on computers, where computational intensive and highly parallel computing is required. It has been reported that the floating-point computation speed is 10 times of CPU, and the memory bandwidth is 5 times of the general memory compared to the cotemporary CPU [3]. Therefore, the GPU has been widely applied to general-purpose computing, such as scientific computation, fluid mechanics simulation, molecular mechanics computation etc. [5-6].

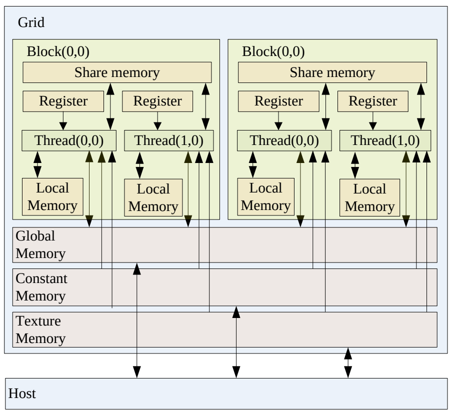


Figure 2: Memory Model of CUDA [7]

CUDA is a parallel computing platform based on the SIMD (Single Instruction, Multiple Data) [4]. The memory model of CUDA is depicted in Figure 2, in which threads are divided into three levels: thread, block and grid. Each thread and block has its unique index. Each thread has a small and fast private register, while each block has a faster shared memory. The operation of memory read and write can be done by all threads in the same block. Both global memory and constant memory existing in grid are visible to all threads.

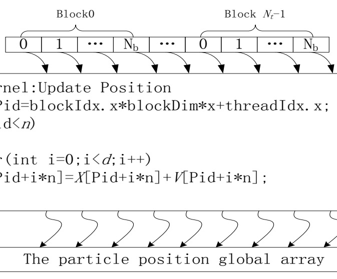
As an extended library in the environment of C/C++, CUDA C/C++ greatly facilitates the fast coding of kernel function running on GPU of NVIDIA for developers. Through calling the kernel function by CPU, the program can be run on the GPU parallelly. The program execution process of CUDA is divided into three steps: copy data to

GPU, execute kernel function and copy data to CPU.

## 4. GPU-Based PSO

Clearly, the evaluation of fitness function and updating of each particle in PSO is independent to each other. Therefore, PSO has the basic architecture of parallel computing. The whole process of parallel computing can be achieved through the one-one correspondence between each particle in PSO and the thread of GPU. The detailed procedure is: (1) thread is set on GPU with the same number as that of particles; (2) storage space is set up for each particle to store its velocity, position and other related data; (3) fitness evaluation and updating for all the particles are done simultaneously using GPU. Here, the execution model for position updating of particle in Figure 3 is used as a demonstration to show the parallel computing procedure of PSO. For the parallel computing of velocity updating and fitness evaluation, the procedure is the basically same.

In Figure 3, each block contains ($N_b$+1) thread. The smallest unit in CUDA is half-warp (16 threads). Therefore, ($N_b$+1) is generally set as the integer multiple of 16. Pid is the index number of thread, *i.e.* the serial number of each particle in PSO. $n$ is the total number of particles, $d$ is the optimization dimension. $N_t$ is the total number of thread block, which is determined by $N_t = \left(n + N_b - 1\right) / N_b$ .



```
//kernel:Update Position
int Pid=blockIdx.x*blockDim*x+threadIdx.x;
if(Pid<n)
{
  for(int i=0;i<d;i++)
  X[Pid+i*n]=X[Pid+i*n]+V[Pid+i*n];
}
```

Figure 3: The execution model based on CUDA

Based on the procedure of PSO and the parallel scenario introduced above, GPU-PSO is established, of which the flowchart is illustrated in Figure 4. Clearly, the processes of initiation, updating and fitness evaluation are all paralleled, of which the fitness evaluation is the prime component for parallelization.

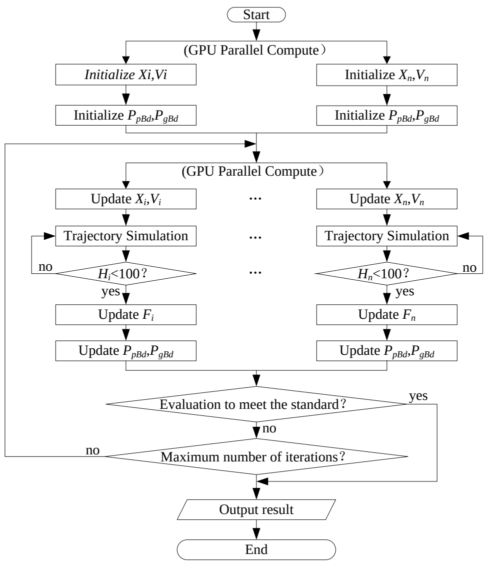


Figure 4: Flowchart of GPU-PSO

## 5. Comparative studies

In this section, four commonly used functions listed in Table 1 are used to verify the effectiveness of the developed PSO-GPU. The PSO algorithm without parallel computing (CPU-PSO) is also employed to optimize these functions. The speedup ratio $s_p$ of GPU-PSO is calculated by taking the ratio of the computational time $t_{cpu}$ of CPU-PSO and $t_{gpu}$ of GPU-PSO.

All the parameters in PSO are set as follows as is commonly done in practice: the inertia weight $w = 0.7298$, learning factor $c_1 = c_2 = 2.05$ , velocity of updating particle is the right bound of each function. In order to effectively investigate the speedup capability of GPU-PSO over CPU-PSO, the convergent criterion is set as that when the number of optimization iterations reaches 5000, the optimization is stopped. The design dimension is set as *d*=50 for all the four functions, and different number of particles *n* are tested to investigate its impact on the speed-up ratio. The computational environment employed in this work is shown as Table 2.

Table 1: Test functions

| Names | Functions |
|---|---|
| Sphere | $f_1 = \sum_{i=1}^{d} x_i^2, \quad -100 \le x_i \le 100$ |
| Rosenbrock | $f_2 = \sum_{i=1}^{d} \left[ x_i^2 - 10\cos(2\pi x_i + 100) \right], \ -10 \le x_i \le 10$ |
| Griewank | $f_3 = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{k=1}^{d} \cos\left( x_i / \sqrt{i} \right) + 1, \ -600 \le x_i \le 600$ |
| Ackley | $f_4 = -20 * \exp\left( -0.2 * sqrt\left( (1/d) * \left( \sum_{i=1}^{d} x_i^2 \right) \right) \right) - \exp\left( (1/d) * \left( \sum_{i=1}^{d} \cos(2\pi x_i) \right) \right) + \exp(1) + 20, -8 \le x_i \le 8$ |

Table 2: The computational environment

| CPU | Intel i7-4770k |
|---|---|
| GPU | NVIDIA GTX TITAN BLACK |
| Memory | 16G |
| OS | Windows 7 X64 |
| Platform | Visual Studio 2013, CUDA6.5 |

Table 3: The results of Sphere function (*d*=50)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 0.775 | 0.000 | 11.342 | 1.638 | 6.924 |
| 2500 | 0.000 | 0.000 | 18.235 | 1.703 | 10.708 |
| 3600 | 0.000 | 0.000 | 24.820 | 1.841 | 13.482 |
| 4900 | 0.000 | 0.000 | 40.404 | 1.888 | 21.400 |
| 6400 | 0.000 | 0.000 | 59.657 | 1.950 | 30.593 |

Table 4: The results of Rosenbrock function (*d*=50)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 1.636 | 11.851 | 11.996 | 1.638 | 7.324 |
| 2500 | 33.599 | 5.272 | 21.606 | 1.716 | 12.591 |
| 3600 | 48.648 | 3.988 | 26.645 | 1.825 | 14.600 |
| 4900 | 30.998 | 2.679 | 42.295 | 1.918 | 22.052 |
| 6400 | 32.283 | 0.080 | 61.261 | 1.966 | 31.460 |

Table 5: The results of Griewank function (*d*=50)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 0.693 | 0.000 | 42.323 | 1.935 | 21.872 |
| 2500 | 0.000 | 0.000 | 66.628 | 1.747 | 38.139 |
| 3600 | 0.000 | 0.000 | 98.093 | 1.872 | 52.400 |
| 4900 | 0.000 | 0.000 | 141.198 | 1.950 | 72.409 |
| 6400 | 0.000 | 0.000 | 193.207 | 1.950 | 99.081 |

Table 6: The results of Ackley function (*d*=50)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 0.031 | 0.000 | 43.025 | 1.310 | 32.844 |
| 2500 | 0.000 | 0.000 | 75.536 | 1.342 | 56.281 |
| 3600 | 0.000 | 0.000 | 98.186 | 1.389 | 70.688 |
| 4900 | 0.000 | 0.000 | 143.191 | 1.435 | 99.7429 |
| 6400 | 0.000 | 0.000 | 198.465 | 1.435 | 138.303 |

Table 7: The results of Griewank function (*d*=100)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 0.000 | 0.000 | 99.391 | 3.454 | 28.776 |
| 2500 | 0.001 | 0.000 | 173.736 | 3.624 | 47.940 |
| 3600 | 0.000 | 0.000 | 250.893 | 3.770 | 66.550 |
| 4900 | 0.000 | 0.000 | 315.421 | 3.913 | 80.681 |
| 6400 | 0.000 | 0.000 | 464.017 | 4.013 | 115.628 |

Table 8: The results of Ackley function (*d*=100)

| *n* | *f** | | *t* /s | | $s_p$ |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 1600 | 0.018 | 0.000 | 86.429 | 2.449 | 35.292 |
| 2500 | 0.003 | 0.000 | 168.461 | 2.538 | 66.376 |
| 3600 | 0.000 | 0.000 | 221.264 | 2.620 | 84.566 |
| 4900 | 0.000 | 0.000 | 341.765 | 2.701 | 126.533 |
| 6400 | 0.000 | 0.000 | 434.846 | 2.767 | 157.154 |

The optimal objective function values (*f**), the computing time (*t*) and the speedup ratio (*sp*) are listed in Tables 3-8, from which it is clearly observed that the GPU-PSO runs much faster than CPU-PSO with very high speed-up ratio. Meanwhile, the optimal solutions of GPU-PSO and CPU-PSO are basically the same, which show great agreements to the real optimal solutions (*f*=0), indicating the good convergence property of GPU-PSO. It is also noticed that the complexity of test functions has large impact on the speed-up ratio, which can be derived from the results in Tables 3-6. Clearly, from the Sphere to Ackley functions, the complexity is basically increasing, and the corresponding speed-up ratio is increasing as well. Meanwhile, the speed-up ratio is also increased with the increase of the number of particles *n*. This is self-evident since the evaluation of all particles are executed serially in CPU-PSO, thus more particles yields more computational time, while this is done all at once in GPU-PSO. However, once *n* reaches to a certain value, the speedup ratio is increased very slowly. The reason is that GPU only contains 2880 CUDA CORE.

To further study the impact of design dimension on the speed-up ratio, *d*=100 are also tested. Considering the space limit, only results of the last two functions are shown (see Tables 7 and 8). It is found that for all the functions, the speed-up ratio with *d*=100 is larger than that with *d*=50, indicating that the speed-up ratio is improved with the increase of optimization dimension. It can be concluded that generally the higher of the dimension and the more of the particles, the larger of the speed-up ratio obtained by GPU-PSO. All these results demonstrate the effectiveness and good speed-up capability of GPU-PSO.

## 6. Application of GPU-PSO to trajectory optimization

### 6.1. Description of trajectory optimization

Trajectory optimization of aerocraft is actually an optimal control problem, which is generally solved by the direct method [8]. As a traditional direct method, the direct shooting method is frequently used for solving practical trajectory optimization due to its simplicity and convenience. With this method, the optimal control is transcribed into a nonlinear programming problem (NLP) through parameterizing the control on certain time nodes and treated as design variables. For engineering applications, it is necessary to generate trajectory as fast as possible. However, oftentimes, a large number of discreted nodes are required to parameterize the control variable in order to ensure high accuracy especially for complex and long-time flight mission, which may increase the time of optimal trajectory generation. Therefore, GPU-PSO is applied to the glide trajectory optimization to save computational time. The optimal control of the glide trajectory optimal problem is formulated as [9]:

$$
\begin{aligned}
&\mathit{Find} \quad \alpha(t), \quad 0^\circ \le \alpha \le 10^\circ \\
&\mathit{Max} \quad J = L(\tau_f) \\
&\mathit{s.t.} \quad h(\tau_f) \le 100 \\
&\qquad \begin{cases} m\dfrac{dV}{dt} = -C_x qS - mg\sin\theta \\ mV\dfrac{d\theta}{dt} = C_y qS - mg\cos\theta \\ \dfrac{dL}{dt} = V\cos\theta, \ \dfrac{dh}{dt} = V\sin\theta \end{cases}
\end{aligned}
\tag{2}
$$

where $V,\theta,L,h$ are respectively velocity, trajectory angle and height, with initial values as $V_0$=542.725m/s, $\theta_0 = 10^\circ$, $L_0$=0.0m and $H_0$=28541m; $q, C_x, C_y$ are the dynamic pressure, drag coefficient and lift coefficient, respectively; *S*=0.126m$^2$ is the reference area; *g* = 9.8m/s$^2$ is the acceleration of gravity; *m*=210kg is the mass of vehicle. $C_x$ and $C_y$ are calculated using linear interpolation with respect to *h*, *Ma* and $\alpha$.

The design variable of the problem is the law of the angle of attack $\alpha(t)$, which is constrained during flight due to the structural and control requirements. The objective function is to maximize the range of the vehicle $L(\tau_f)$ without any power by control as much as possible. The terminal boundary constraint is $h(\tau_f) \le 100m$, *i.e.* the flight task is completed when the flight height is less than 100m. The direct shooting method is employed to solve the gliding trajectory optimization in Eq. (2) and GPU-PSO is employed to solve the transcribed NLP. The flight time $[\tau_0, \tau_f]$ is divided into *N* sub-intervals as $\tau_0 < \tau_1 < \cdots < \tau_{N-1} < \tau_N = \tau_f$. Correspondingly, the angle of attack $\alpha(t)$ is discreted at the *N*+1discreted time nodes as $\alpha_0, \alpha_1, \cdots, \alpha_i, ..., \alpha_{N-1}, \alpha_N$. The control variable $\alpha(t)$ at anytime point is predicted by the spine interpolation over the *N*+1 discreted time nodes and the corresponding angle of attack. Clearly, the flight time $\tau_f$ is unknown. In this work, it is not considered as a design variable, and is only used for the calculation of trajectory and interpolation of angle of attack. If the flight time $\tau > \tau_f$, then $\alpha(\tau) = \alpha_{\tau_f}$.

### 6.2. Results

For this problem, the optimization process is terminated when the variation of the maximum flight range is less than 0.01*m*. The number of particles is set as *n*=10000 and *d*=51 (*N*=50).The parameters in PSO are set as the same as those in the above four mathematical examples. The optimal angle of attack is shown in Figure 5. It is found that both methods yield almost the same optimal solutions, while GPU-PSO (1464.146s) needs much shorter time than CPU-PSO (9.064s), with about 161X speedup. The optimal angle of attack and range of GPU-PSO are slightly different from those of CPU-PSO, which is caused by the difference of the floating point computing method between CPU and GPU. This can be reduced and even eliminated by screwing up the code [10].

Although, the speed-up capability is remarkable, it is necessary to verify the accuracy of GPU-PSO. From the theory of lift-to-drag ratio, the maximum range can be approximately obtained when gliding with maximum lift-to-drag ratio [11]. Therefore, the trajectory with maximum lift-to-drag ratio is calculated by plugging different angle of attack $\alpha(\tau)=1^{\circ},...,10^{\circ}$ into the dynamic model in Eq. (2) and Rung-Kutta numerical integration is used to obtain the lift-to-drag ratio and range. It is found that $\alpha(\tau)=10^{\circ}$ yields the largest lift-to-drag ratio, so as the range. Clearly, the optimal angle of attack by GPU-PSO is equal to $10^{\circ}$during almost the whole flight mission, which is basically the same as that derived from the lift-to-drag theory. The trajectories of $\alpha(\tau)=10^{\circ}$, GPU-PSO and CPU-PSO are shown in Figure 6. It is noticed that the three trajectories are almost the same, yielding very similar range (129.02, 129.06 and 129.70km). These results demonstrate the effectiveness of GPU-PSO.

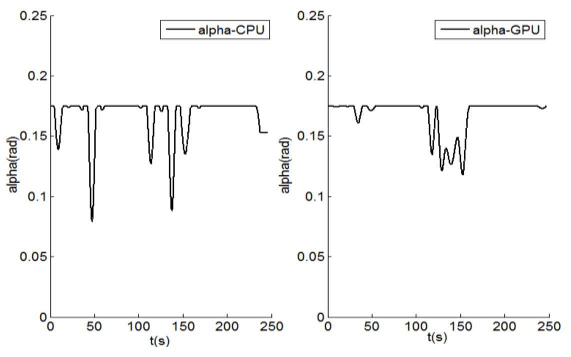


Figure 5: The optimal $\alpha(\tau)$ of both methods

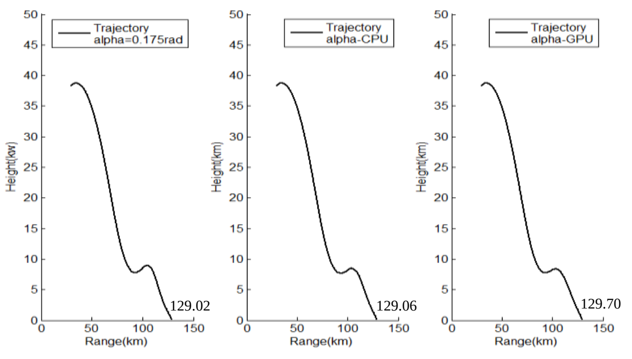


Figure 6: Optimal trajectories

## 7. Conclusions

In this paper, based on the standard PSO algorithm, the parallel computing of PSO is developed using the GPU technique in the CUDA platform. Through the comparison of GPU-PSO and CPU-PSO by four mathematical functions, it is observed that the developed GPU-PSO can greatly save computational time, while yields good accuracy and convergence properties. The application of GPU-PSO to a glide trajectory optimization problem further demonstrates the effectiveness and advantage of GPU-PSO, as well as its great potential to practical engineering optimization.

## References

[1] J. Kennedy, R C.Eberhart, Particle Swarm Optimization, *Proc of IEEE International Joint Conference on Neural Networks,*Washington DC：IEEE Computer Society, 1995: 1942 - 1948.

[2] F. Bergh, A. P. Engelbrecht, Cooperative Learning in Neural Net Works Using Particle Swarm Optimizers, *South African Computer Journal*, 200 (011): 84 – 90.

[3] Y. Zhou, Y. Tan, et al, GPU-based Parallel Particle Swarm Optimization, *2009 IEEE Congress on Evolutionary Computation* (CEC 2009).

[4] NVIDIA, *NVIDIA CUDA C Programming Guide 6.5*, 2014.

[5] J. D. Owens, M. Houston, D. S. Luebke, et al, *GPU Computing*, IEEE,2008,96(5):879 – 899.

[6] J. D. Owens, D. Luebke, N. Govindaraju, et al, A Survey of General-purpose Computation on Graphics Hardware, *Computer Graphics Forum*, 2007, 26(1): 80 – 113.

[7] J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, *Addison-Wesley Professional*, 2010

[8] E. Yong, L. Chen, G. Tang, A Survey of Numerical Methods for Trajectory Optimization of Spacecraft, *Journal Of Astronautics*, 2008, 29(2):398-406.

[9] X. F. Qian, R. X. Lin, and Y. N. *Zhao, Flight Dynamics of Missile*, Beijing: Beijing Institute of Press, 2000. (in Chinese)

[10] D. B. Kirk, W. Hu, *Programming Massively Parallel Processors*, Elsevier Science Ltd, 2010.

[11] J. Shi, Z. Wang, X. Cao, W. Liu, K. Yang, Design of glide trajectory for gliding range-extended projectile, *Journal of Nanjing university of Science and Technology*, 2007, 31(2):147 – 153.