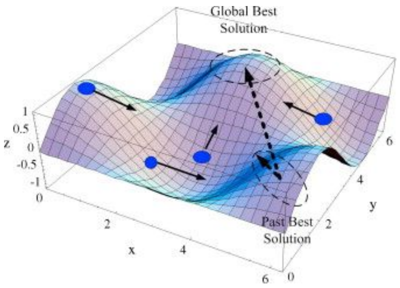
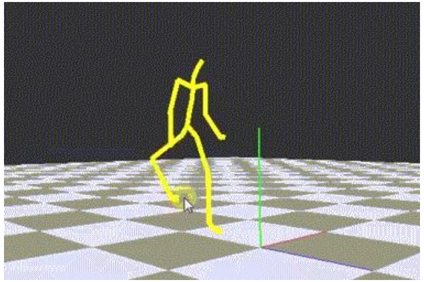
Parallel Particle Swarm Optimization on Inverse kinematics

第六組
林宗佑 10967222
劉禮榮 10967241





Outline

Introduction to Inverse Kinematics

Introduction to Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

Experiment

Demo

Reference

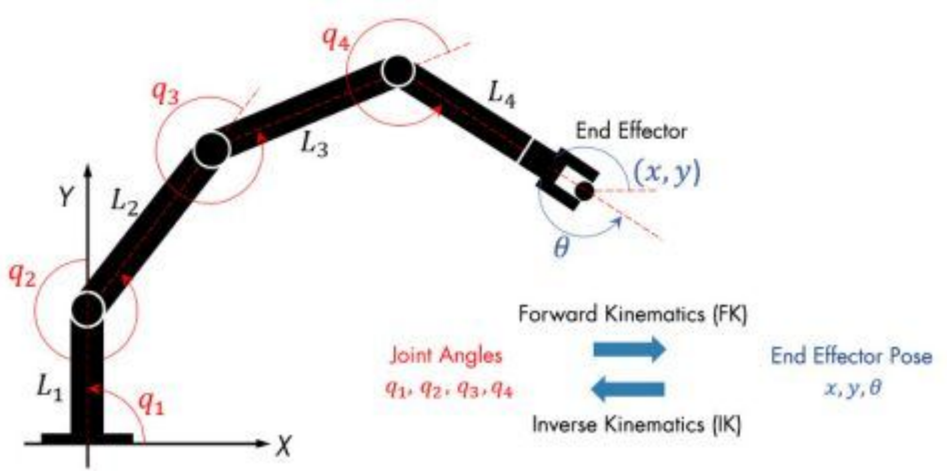
Outline

- Introduction to Inverse Kinematic
- Introduction to Particle Swarm Optimization
- Related Work
- Paralle Particle Swarm Optimization
- Experiment
- Demo
- Conclusion and Future work
- Reference

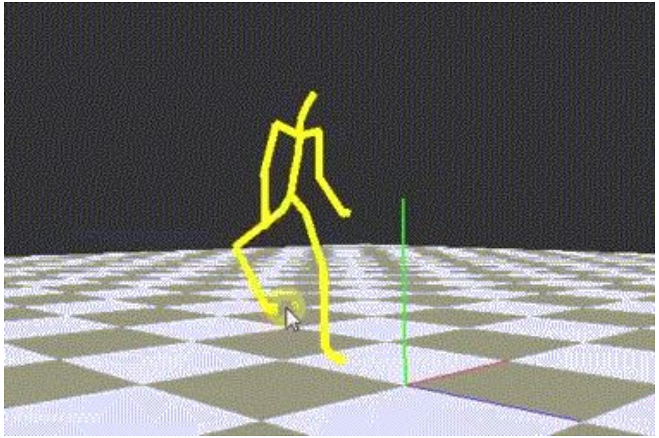
Introduction to Inverse Kinematics

[Neutrino's Blog：電腦動畫中的反向動力法 (Inverse Kinematics)]()

> 常見的方法包含 Cyclic Coordinate Descent 方法、Jacobian Pseudoinverse 方法、Jacobian Transpose 方法、Levenberg-Marquardt Damped Least Squares 方法、Quasi-Newton and Conjugate Gradient 方法、神經網路方法



https://www.mathworks.com/discovery/inverse-kinematics.html



https://nagachiang.github.io/assets/images/IK.gif

Outline

Introduction to Inverse Kinematic

Introduction to Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

Experiment

Demo

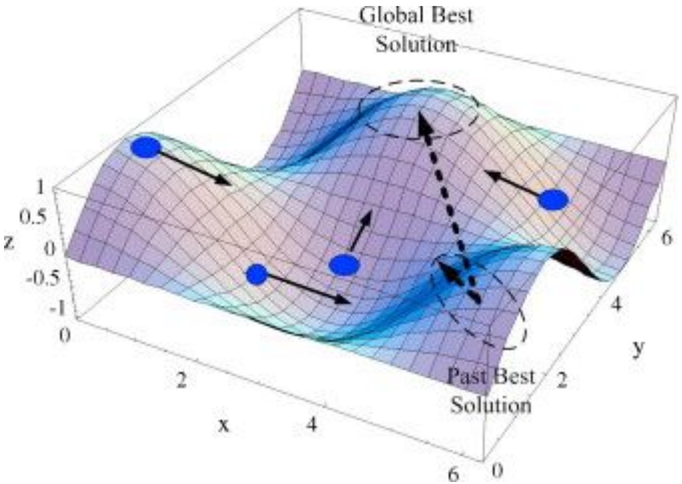
Conclusion and Future work

Reference

Introduction to Particle Swarm Optimization

$$V_{ij}^{t+1} = wV_{ij}^{t} + c_1 r_1^t \left(pbest_{ij} - X_{ij}^{t}\right) + c_2 r_2^t \left(gbest_j - X_{ij}^{t}\right)$$

$$X_{ij}^{t+1} = X_{ij}^{t} + V_{ij}^{t+1}$$



https://de.mathworks.com/matlabcentral/fileexchange/43541-particle-swarm-optimization-pso

1. Initialization
   1.1. For each particle i in a swarm population size P:
      1.1.1. Initialize X_i randomly
      1.1.2. Initialize V_i randomly
      1.1.3. Evaluete the fitness $f(X_i)$
      1.1.4. Initialize $\boldsymbol{pbest}_i$ with a copy of X_i
   1.2. Initialize $\boldsymbol{gbest}$ with a copy of X_i with the best fitness
2. Repeat until a stopping criterion is satisfied:
   2.1. For each particle i:
      2.1.1. Update V_i^t and X_i^t according to Eqs. (1) and (2)
      2.1.2. Evaluete the fitness $f(X_i^t)$
      2.1.3. $\boldsymbol{pbest}_i \longleftarrow X_i^t$ if $f(\boldsymbol{pbest}_i) < f(X_i^t)$
      2.1.4. $\boldsymbol{gbest} \longleftarrow X_i^t$ if $f(\boldsymbol{gbest}) < f(X_i^t)$

Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems
by Bruno Seixas Gomes de Almeida and Victor Coppo Leite

Outline

Introduction on Inverse Kinematic

Introduction on Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

Experiment

Demo

Conclusion and Future work

Reference

Related Work

Parallel global optimization with the particle swarm algorithm, by J. F. Schutte , J. A. Reinbolt. Int J Numer Methods Eng. 2004 December 7

Particle swarm optimization within the CUDA architecture,by Luca Mussi,Stefano Cagn ,Conference: Genetic and Evolutionary Computation Conference - GECCO 2009

Parallel asynchronous particle swarm optimization, by Byung-Il Koh , Alan D. George. Int J Numer Methods Eng. 2006 July 23

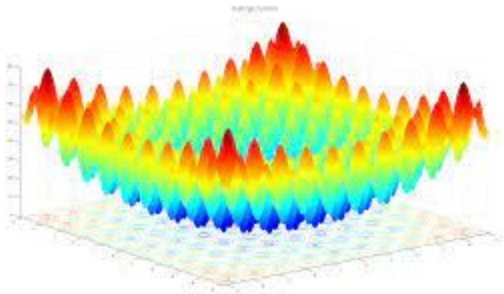
1. Initialization
   1.1. For each particle i in a swarm population siz
      1.1.1. Initialize X_i randomly
      1.1.2. Initialize V_i randomly
      1.1.3. Evaluete the fitness $f(X_i)$
      1.1.4. Initialize $pbest_i$ with a copy of X_i
   1.2. Initialize $gbest$ with a copy of $X_i$with the be

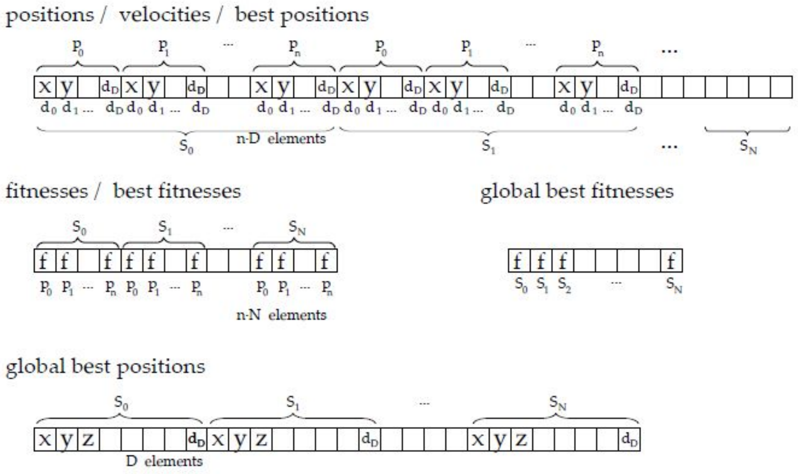
Related Work

- Master processor
  1. Initializes all optimization parameters and particle positions and velocities;
  2. Holds a queue of particles for slave processors to evaluate;
  3. Updates particle positions and velocities based on currently available information p^i, p^g;
  4. Sends the position x^i of the next particle in the queue to an available slave processor;
  5. Receives cost function values from slave processors;
  6. Checks convergence.
- Slave processor
  7. Receives a particle position from the master processor;
  8. Evaluates the analysis function $f(x^i)$ at the given particle position x^i;
  9. Sends a cost function value to the master processor.

Related Work

Rastrigin's function





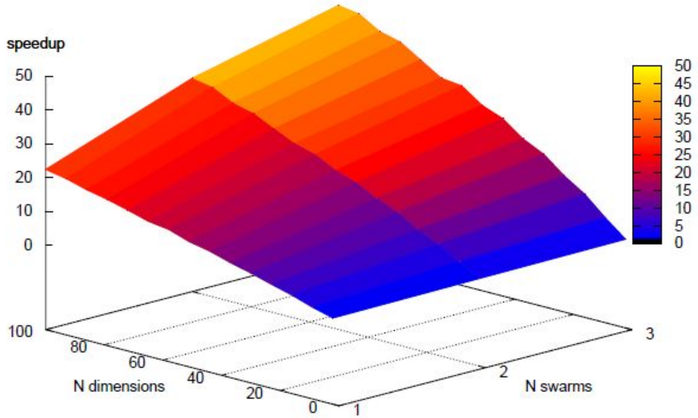


Figure 3: CUDAPSO vs sequentialPSO: Speedup

Outline

Introduction on Inverse Kinematic

Introduction on Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

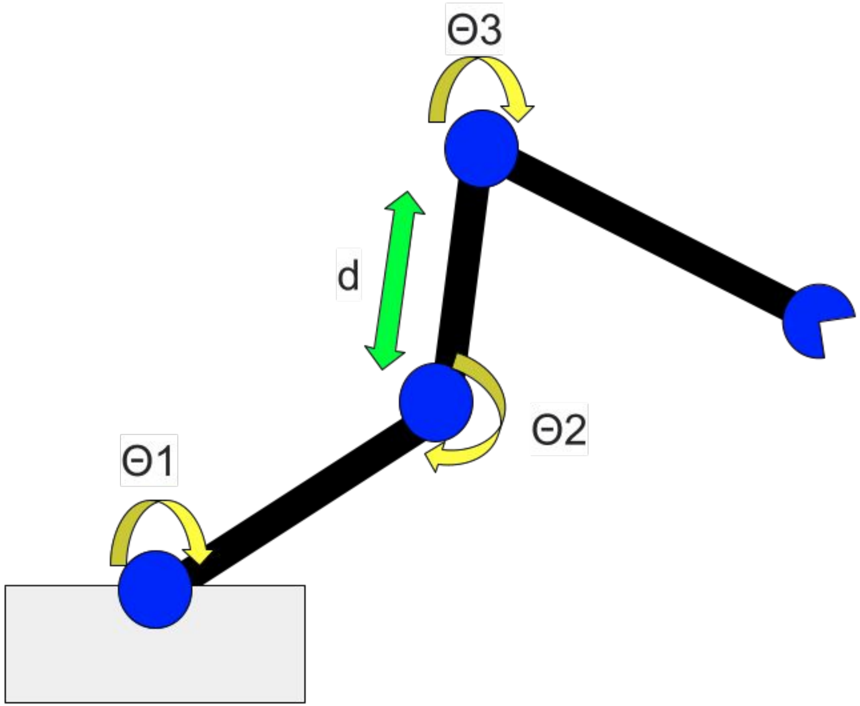
- OpenMP
- CUDA

Experiment

Demo

Reference

Problem Definition



Paralle Particle Swarm Optimization : CUDA

1. Introduce MTGP32 random number generator
2. Reduce memory copy between host to device
3. Reduction when evaluating pBestFit

Paralle Particle Swarm Optimization : CUDA

random number generator is needed when :

1. particle is out of bound
2. generating R1 R2

Generating 32768 * 128 * 256 random floats between 1000000 and 0 costs:

5.65640s for GPU with 128 * 256 threads

57.94887s for CPU

Paralle Particle Swarm Optimization : CUDA

Reduce memory copy between host to device

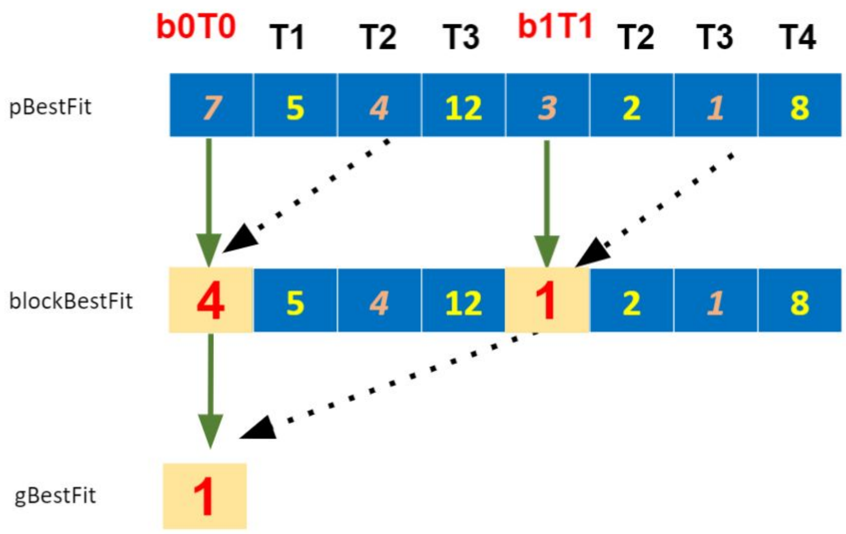
```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   97.63%  549.72ms     10000  54.972us  44.575us  104.93us  deviceSearch(double*, dou
e*, double*, double*, double*, double*, int*, int*, curandStateMtgp32*)
                    2.37%  13.345ms     20038     665ns     543ns  126.14us  [CUDA memcpy HtoD]
                    0.00%  4.5130us         6     752ns     672ns     928ns  [CUDA memcpy DtoH]
                    0.00%     832ns         1     832ns     832ns     832ns  [CUDA memset]
      API calls:   70.77%  567.56ms     10000  56.756us  2.0550us  510.99us  cudaDeviceSynchronize
                   16.23%  130.14ms        17  7.6553ms  1.2310us  130.00ms  cudaMalloc
                    9.16%  73.466ms     20017  3.6700us  2.2340us  947.63us  cudaMemcpyToSymbol
                    3.69%  29.599ms     10000  2.9590us  2.6690us  256.10us  cudaLaunchKernel
                    0.07%  557.79us        27  20.659us  4.9910us  35.661us  cudaMemcpy
                    0.03%  253.08us         1  253.08us  253.08us  253.08us  cuDeviceTotalMem
                    0.02%  182.71us        17  10.747us  1.3650us  95.969us  cudaFree
```

CUDA MEMCPY HtoD Only costs 2.37%

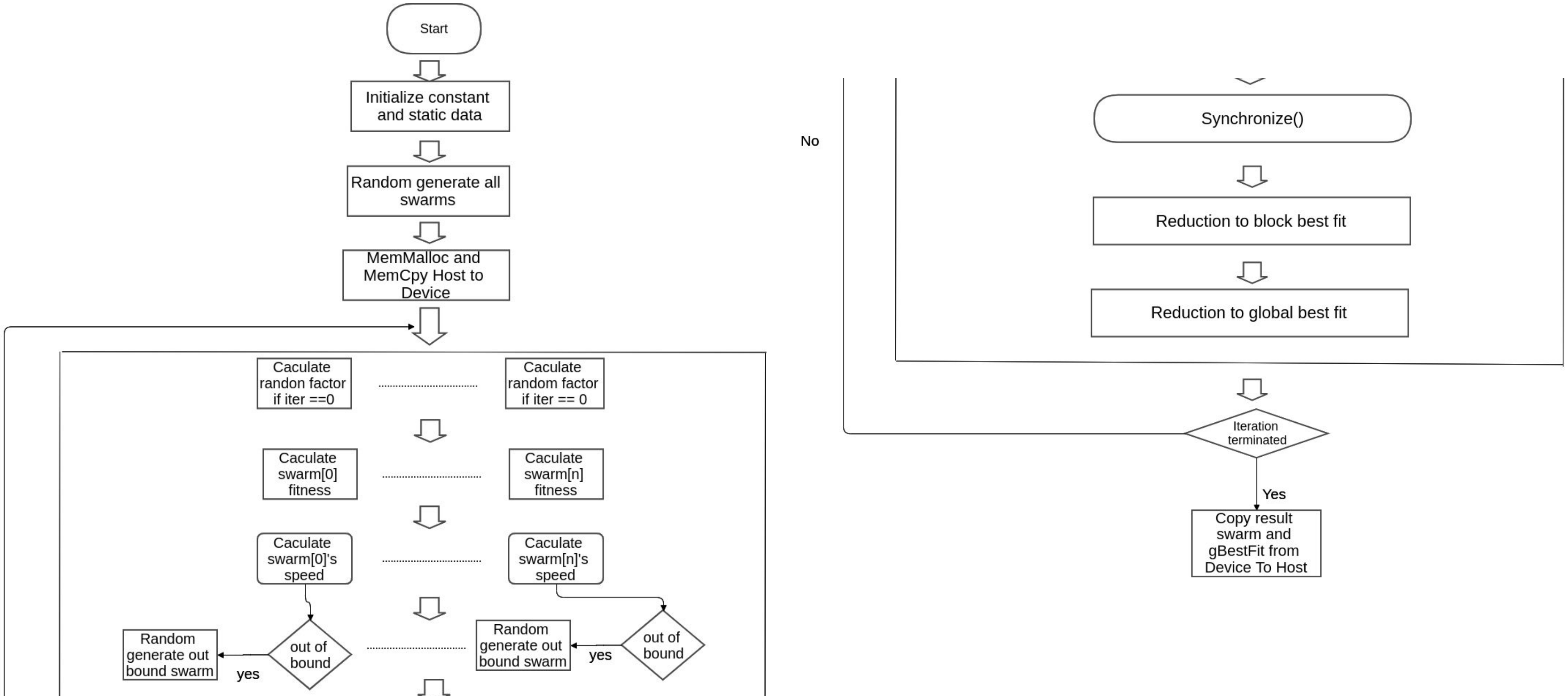
CUDA MEMCPY DtoH Only costs 0.00%

Paralle Particle Swarm Optimization : CUDA

1. Reduction when evaluating pBestFit



Paralle Particle Swarm Optimization : CUDA



Paralle Particle Swarm Optimization : OpenMP

work sharing:

#pragma omp parallel for

1. Initialization
   1.1. For each particle i in a swarm population size P:
      1.1.1. Initialize X_i randomly
      1.1.2. Initialize V_i randomly
      1.1.3. Evaluete the fitness $f(X_i)$
      1.1.4. Initialize $\boldsymbol{pbest}_i$ with a copy of X_i
   1.2. Initialize $\boldsymbol{gbest}$ with a copy of X_i with the best fitness
2. Repeat until a stopping criterion is satisfied:
   2.1. For each particle i:
      2.1.1. Update V_i^t and X_i^t according to Eqs. (1) and (2)
      2.1.2. Evaluete the fitness $f(X_i^t)$
      2.1.3. $\boldsymbol{pbest}_i \longleftarrow X_i^t$ if $f(\boldsymbol{pbest}_i) < f(X_i^t)$
      2.1.4. $\boldsymbol{gbest} \longleftarrow X_i^t$ if $f(\boldsymbol{gbest}) < f(X_i^t)$

Outline

- Introduction on Inverse Kinematic
- Introduction on Particle Swarm Optimization
- Related Work
- Paralle Particle Swarm Optimization
- Experiment
- Demo
- Conclusion and Future work
- Reference

Experiment--Platform

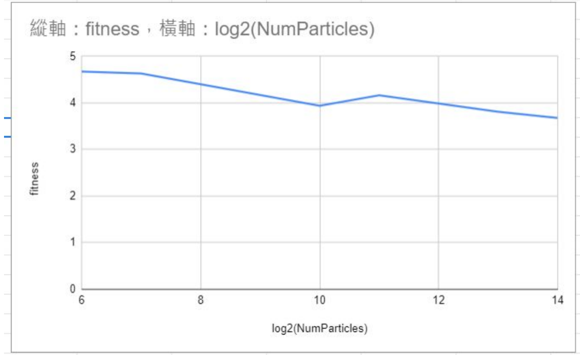
Parallel platform:

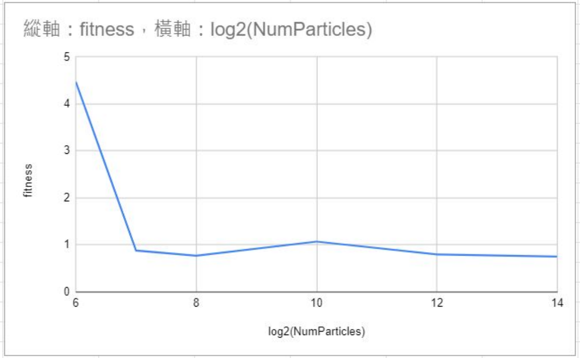
- OpenMp
- Cuda
- C++

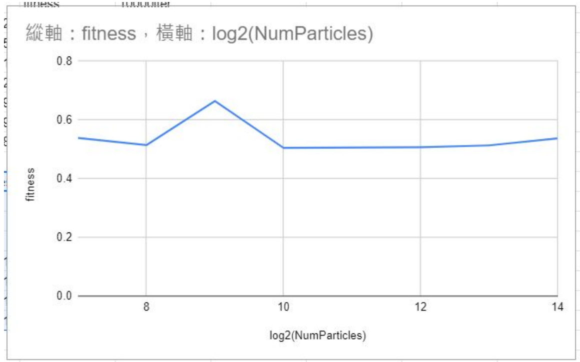
Demo platform:

- Qt

Experiment--CUDA evaluation

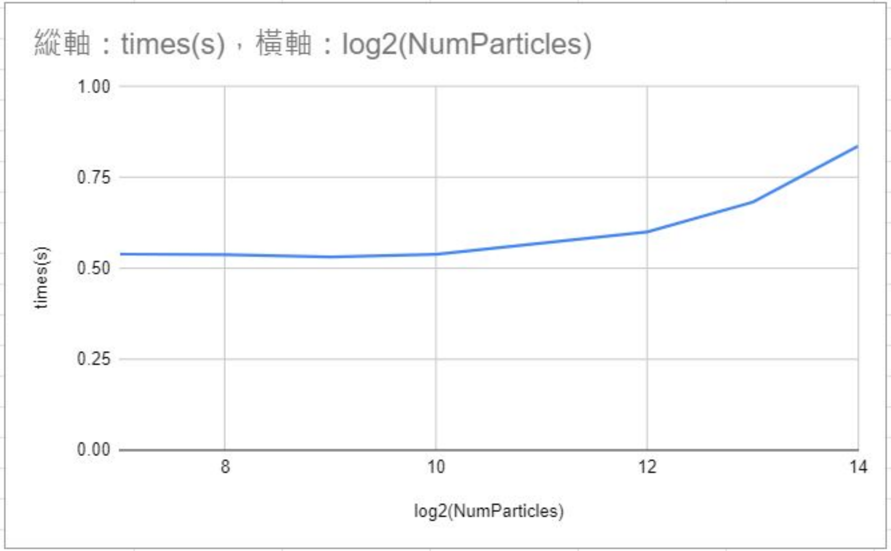


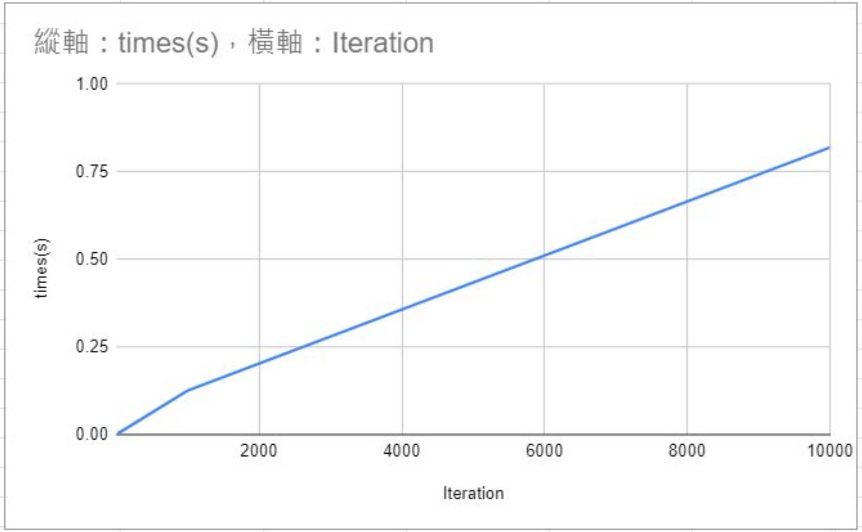




10, 100, 10000 iteration: log2(NumParticle) to fitness

Experiment--CUDA evaluation

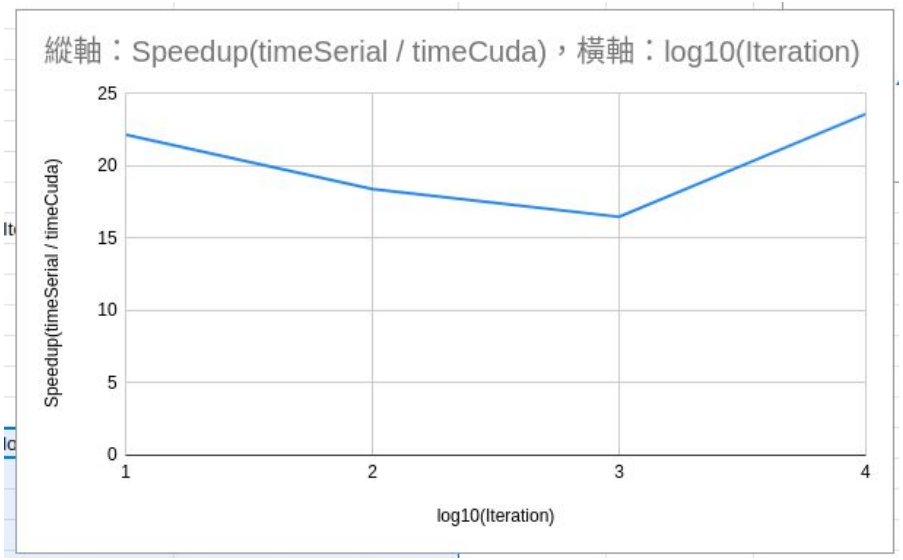




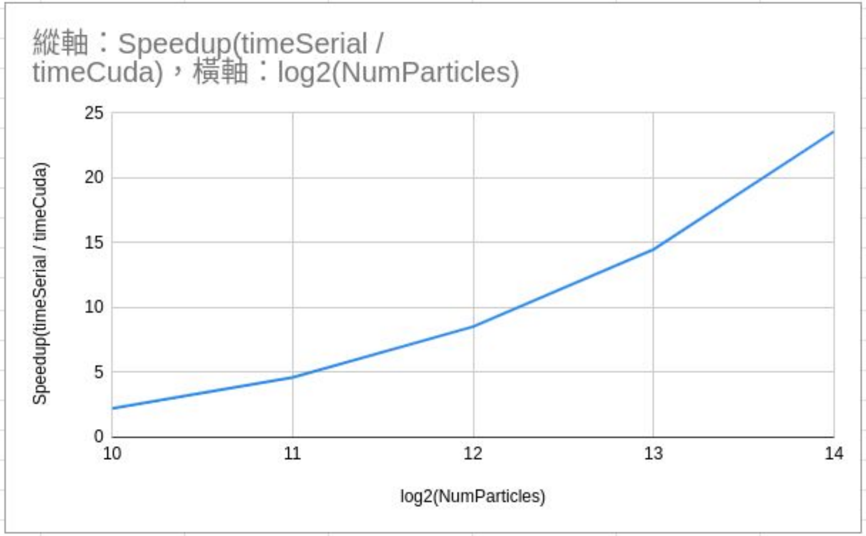
10000 iteration: log2(NumParticle) to time

16384NumParticles: iteration to time

Experiment--CUDA vs Serialize

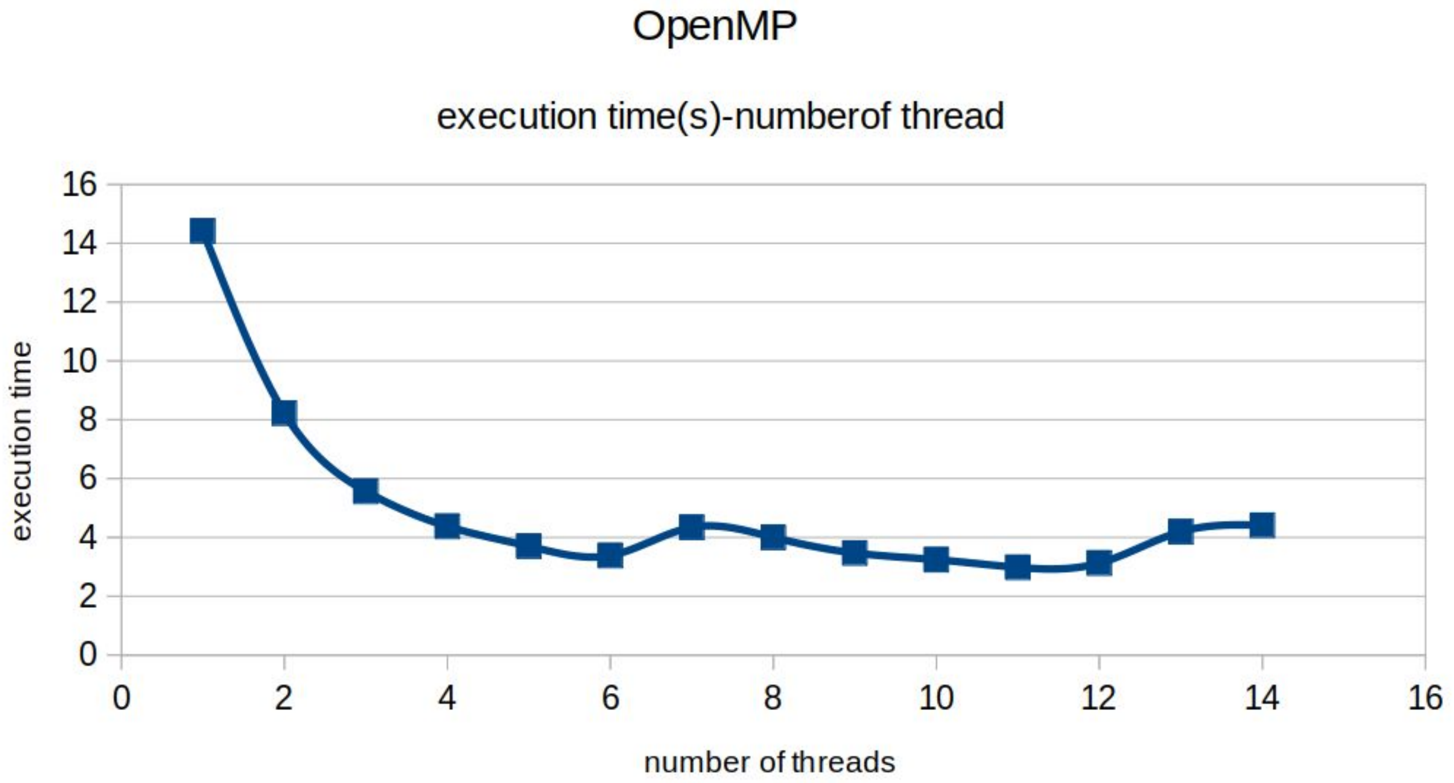


16384NumParticles: log10(Iterations) to time

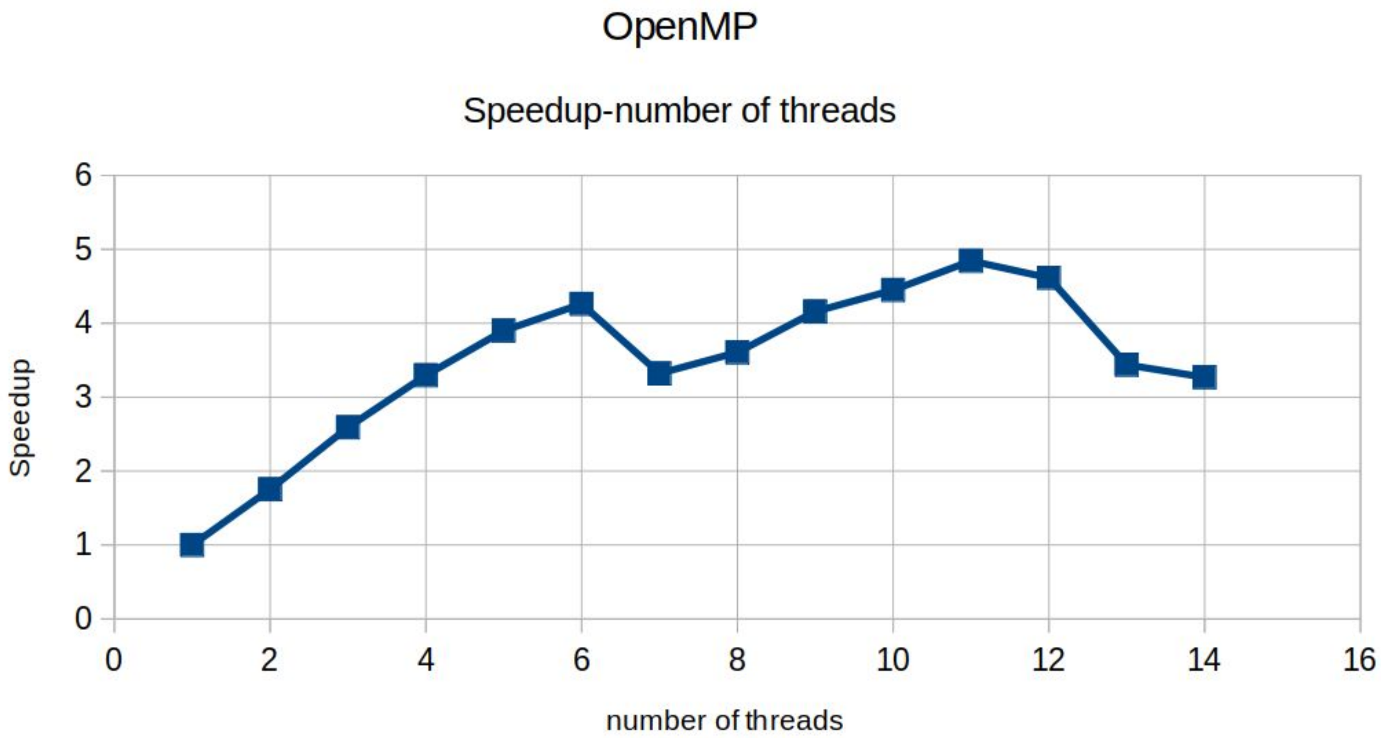


10000iteration: log2(NumParticles) to time

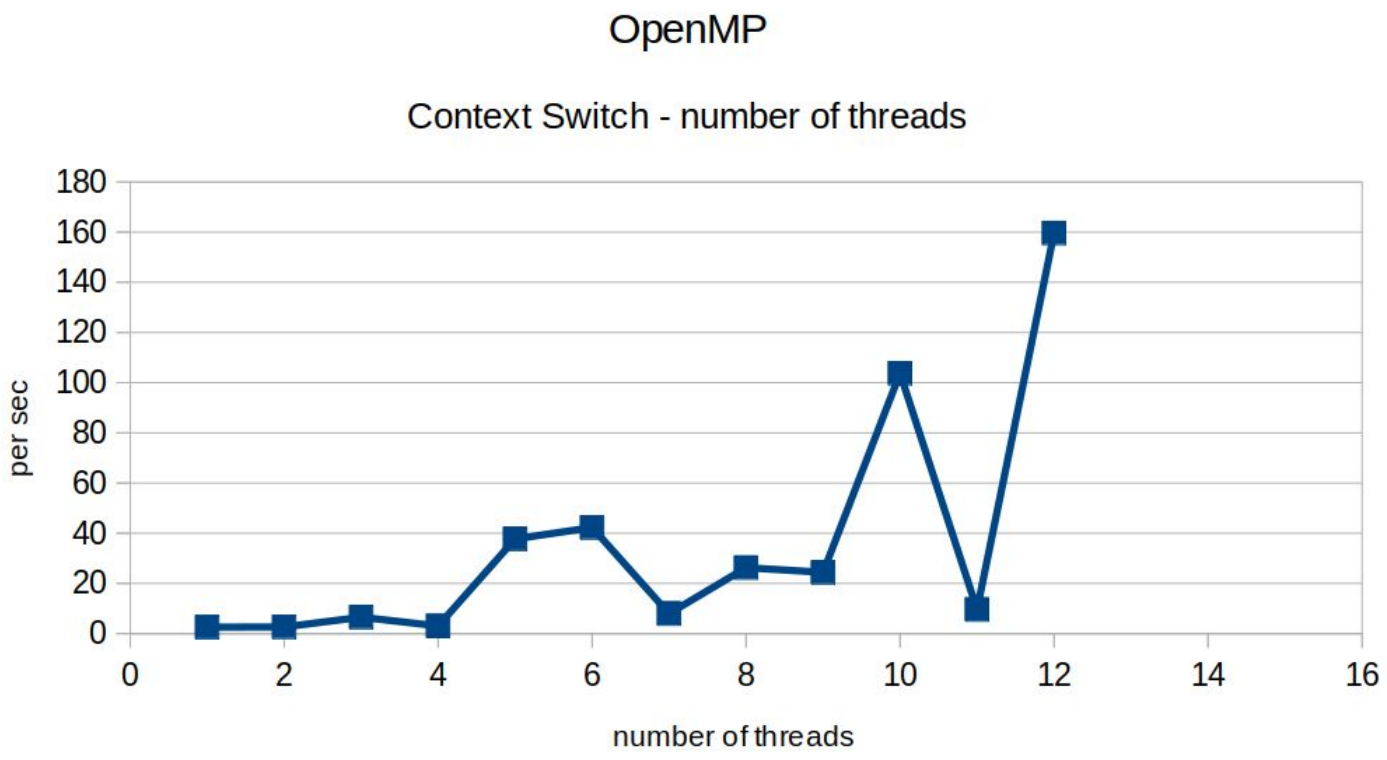
Experiment--OpenMP



Experiment--OpenMP

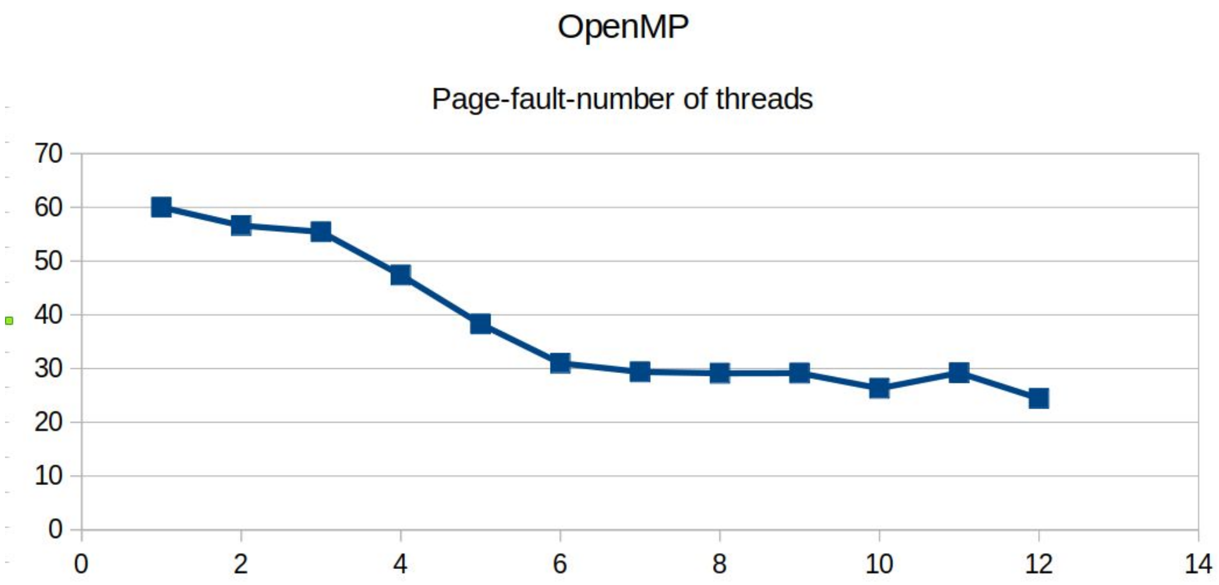


Experiment--OpenMP



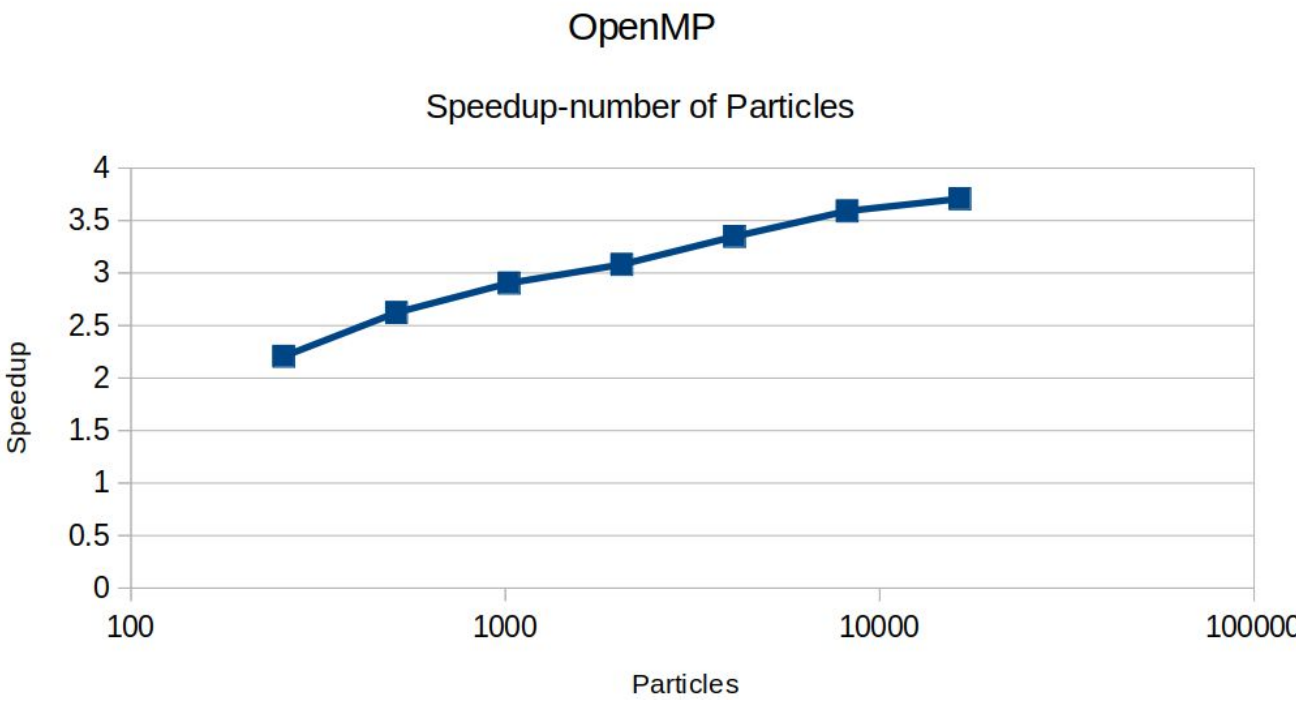
Experiment--OpenMP

4 Threads



Experiment--OpenMP

4 Threads



Outline

Introduction on Inverse Kinematic

Introduction on Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

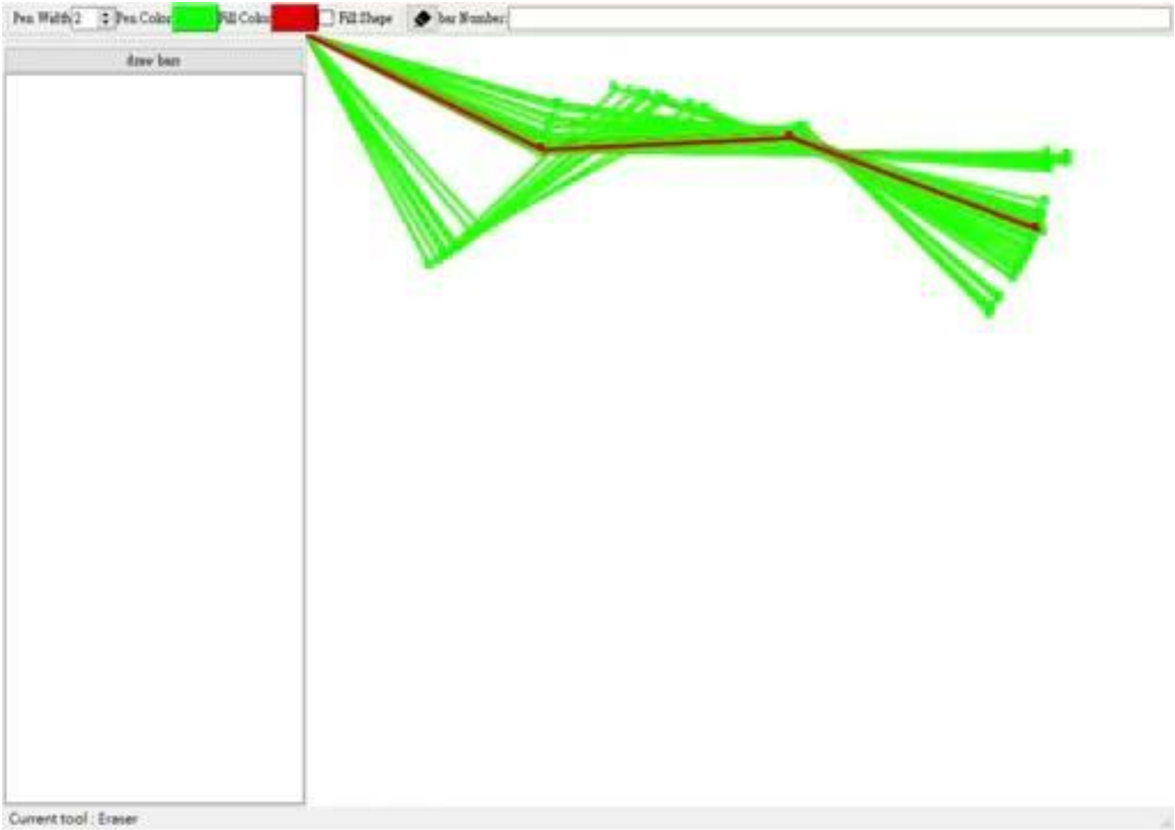
Experiment

Demo

Conclusion and Future work

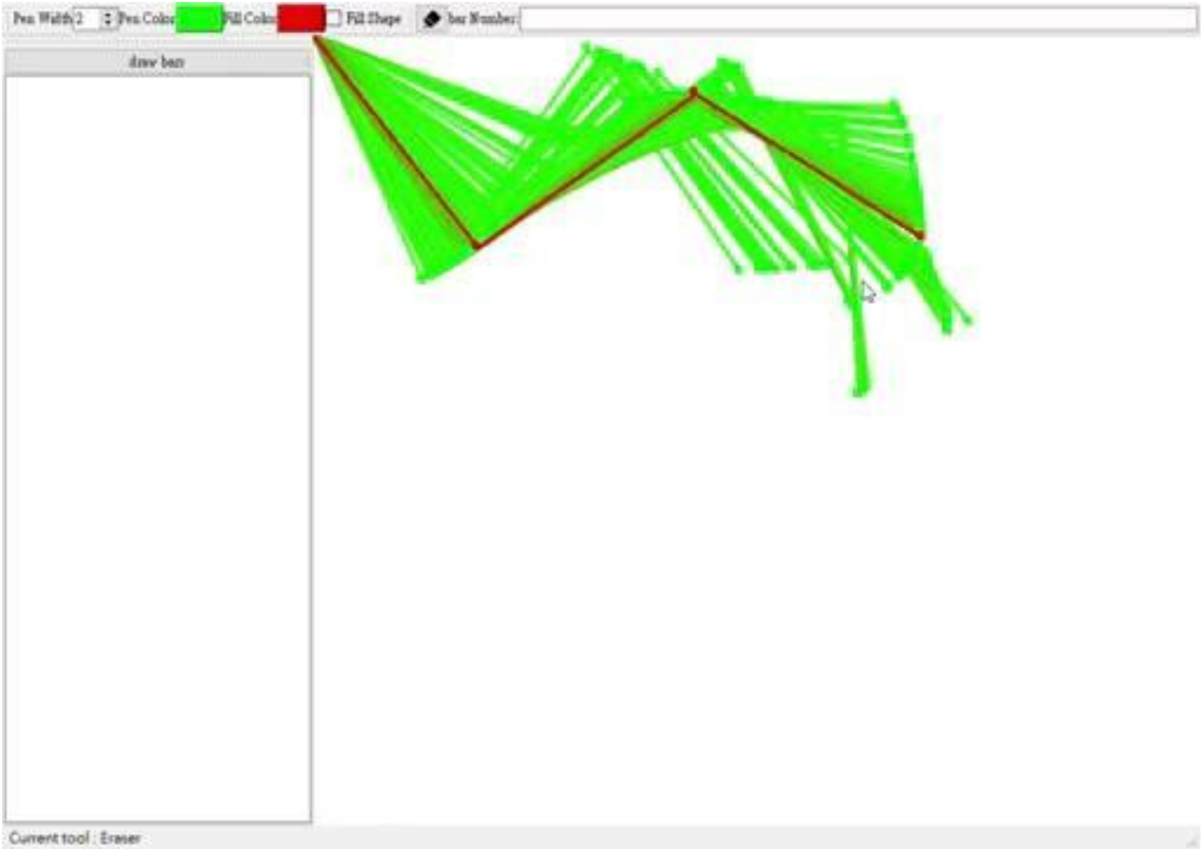
Reference

Demo--serialize 3000iter 15 times slow



gBest is at: a1:0.453, a2:0.489, a3:0.357,d2: 4.791
arm position:(6.0000, 3.000)
time elapsed: 3.53499s
fitness: 0.505182

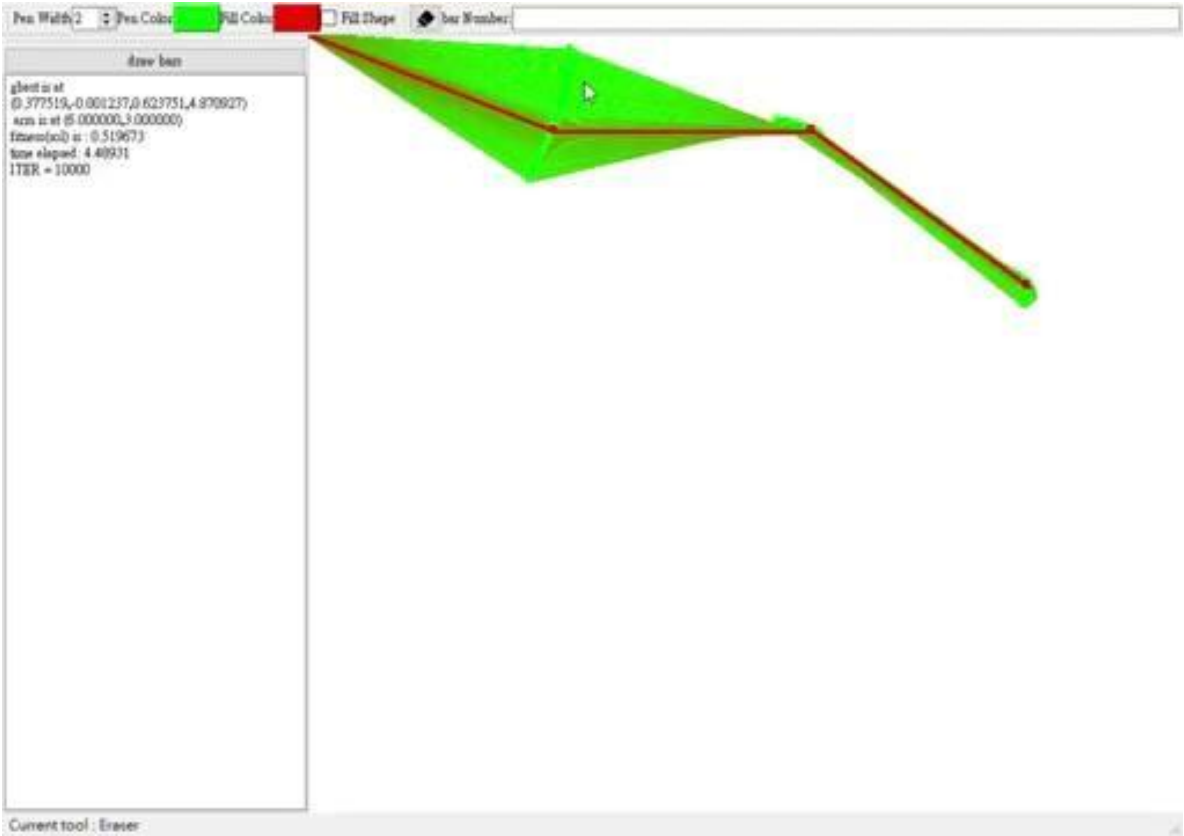
Demo--openMP 3000iter 15 times slow



gBest is at: a1:0.894, a2:-0.599, a3:0.599,d2: 4.960
arm position:(6.0000, 3.000)
time elapsed: 0.6425s
fitness: 0.505182

6.99745X

Demo--openMP 10000iter 15 times slow



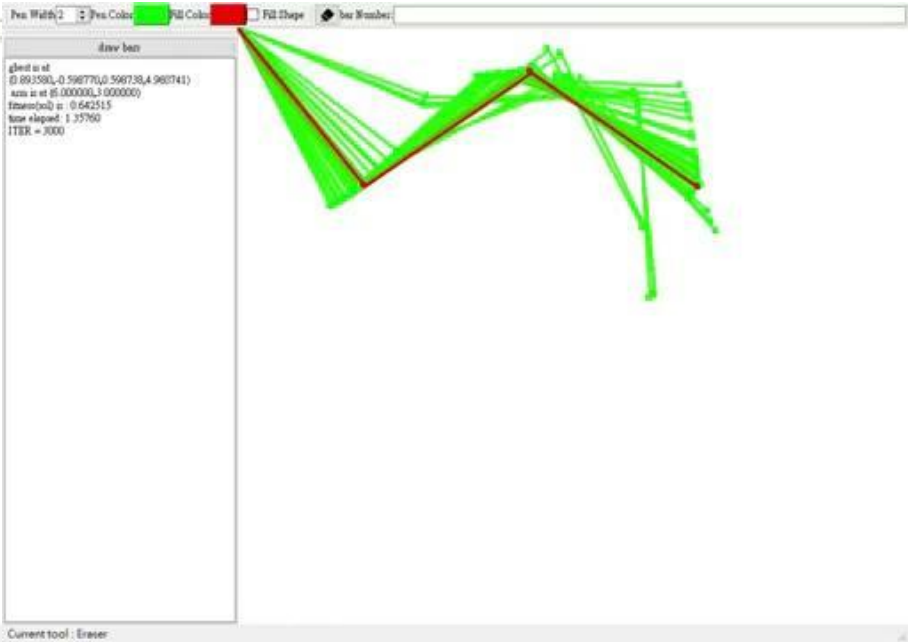
gBest is at: a1:0378, a2:-0.001, a3:0.624,d2: 4.871
arm position:(6.0000, 3.000)
time elapsed: 4.409s
fitness: 0.519673

Demo--cuda 3000 iter 15 times slow

gBest is at: a1:0.234, a2:0.119, a3:1.037,d2: 5.163
arm position:(6.0000, 3.000)
time elapsed: 0.332s
fitness: 0.505182

10.6468X

Demo--cuda and openMP 3000iter normal speed



Outline

Introduction on Inverse Kinematic

Introduction on Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

Experiment

Demo

Conclusion and Future work

Reference

Conclusion and Future work

Conclusion:

openMP and CUDA can significantly speedup PSO when solving Inverse Kinematics

CUDA done better when the problem size increases;

It is much more easier to parallelize PSO by openMP

Future work:

Real time tracing Inverse Kinematics

Outline

Introduction on Inverse Kinematic

Introduction on Particle Swarm Optimization

Related Work

Paralle Particle Swarm Optimization

Experiment

Demo

Conclusion and Future work

Reference

Reference

J. F. Schutte , J. A. Reinbolt., Parallel global optimization with the particle swarm algorithm, Int J Numer Methods Eng. 2004 December 7

Luca Mussi,Stefano Cagn ,Particle swarm optimization within the CUDA architecture, ,Conference: Genetic and Evolutionary Computation Conference - GECCO 2009

Byung-Il Koh , Alan D. George , Parallel asynchronous particle swarm optimization,.Int J Numer Methods Eng. 2006 July 23

Bruno Seixas Gomes de Almeida and Victor Coppo Leite (2019),

Particle Swarm Optimization:A Powerful Technique for Solving Engineering Problems. IntechOpen.
PySwarms (2021), Lj Miranda, https://github.com/ljvmiranda921/pyswarms.

Thank you for your attention