Particle Swarm Optimization

Wilbert Pumacay, *Catholic University San Pablo*, wilbert.pumacay@ucsp.edu.pe
Gerson Vizcarra, *Catholic University San Pablo*, gerson.vizcarra@ucsp.edu.pe

***Abstract*—Heuristics-based swarm algorithms emerged as a powerful family of optimization techniques, inspired by the collective behavior of social animals. Particle Swarm Optimization (PSO) , part of this family, is known to solve large-scale nonlinear optimization problems using particles as set of candidate solutions. In this paper we test the PSO on 3 common benchmarks functions and compare it with the results from other metaheuristics using the Wilcoxon rank sum test.**

We also implemented a parallel version of the algorithm in CUDA to speed up the optimization process and allow more particles to be used.

***Index Terms*—Metaheuristics, Particle Swarm Optimization, Convergence Behavior, CUDA.**

I. Introduction

THE Particle Swarm Optimization algorithm is a stochastic population-based optimization method. It is based on the research of bird and fish flock movement behavior. While searching for food, birds are either scattered or go together before they locate the place where they can find the food. While the birds are searching for food from one place to another there is always a bird that can smell the food very well. Because they are transmitting the information, they can search conduced by the good information, the birds eventually flock to the place where food can be found.
In the algorithm the birds are equal to the particles, the place where there are food is equal the solution of the problem, and the good information is equal to the most optimist solution.

PSO has been applied to many areas such as artificial neural network training, function optimization, fuzzy control, and pattern classification because of its ease of implementation and fast convergence to acceptable solutions.

II. Particle Swarm Optimization (PSO)

A. Basic concepts

Particle Swarm Optimization, first introduced by Kennedy and Eberhart [1] is a stochastic optimization technique htat is based on two fundamental diciplines [3]: social science and computer science. In addition, PSO uses the swarm intelligence concept: the collective behavior of unsophisticated agents that are interacting locally with their environment create coherent global functional patterns.

1) Social concepts : Human intelligence resulting of social interaction consist in evaluate, compare and imitate to others, as well to learn of experience allow to humans to adapt to the environment and find optimal patterns of behavior

2) Swarm intelligence : Swarm intelligence can be described by considering principles, this principles are based on biological agents; according to [2] there are four main principles.

1) Coordination: Information is shared among the agents.
2) Collaboration: Agents can do different tasks in parallel.
3) Deliberation: Agents can determine priorities if they have more than one option.
4) Cooperation: Agents combine their efforts to successfully solve a problem.

3) Computational Characteristics : Swarm intelligence provides a useful paradigm for implementing adaptative systems. In particular, PSO has particles that can be updated in parallel, their values depends of the previous value and the neighbors and all updates are performed using the same rules.

B. Description of the algorithm

PSO is a population based optimization technique, where the population is called a *swarm*. An explanation of the operations performed is as follows. Earch particle represents a posible solution to the optimization task at hand, during each iteration each particle accelerates in the direction of its own personal best solution found so far, as well as the direction of the global best solution discovered so far by any of the particles in the swarm. This means if a particle found a promising new solution, the other particles will move closer to it, it helps to explore the region more deeply.

$$\begin{aligned} v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,i}(t)[y_{i,j}(t) - x_{i,j}(t)] + \\ c_2 r_{2,i}(t)[\hat{y}_j(t) - x_{i,j}(t)] \end{aligned} \tag{1}$$

Let s detote the swarm size, Each individual $1 \leq i \leq s$ has the following attributes. A current position in search space x_i, a current velocity v_i, and a personal best position in the search space y_i. During each iteration, each particle in the swarm is updated using (1) and (2). In (1) for all $j \in 1...n$, $v_{i,j}$ is the velocity of the jth dimension of the ith particle, and c_1 and c_2 denote the *acceleration coefficients*. The new position of a particle is a calculated using

$$x_i(t+1) = x_i(t) + v_i(t+1). \tag{2}$$

The personal best position of each particle is updated using

$$y_i(t+1) = \left\{ \begin{array}{l} y_i(t), \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1), \text{if } f(x_i(t+1)) < f(y_i(t)) \end{array} \right\} \tag{3}$$

and the global best solution found by any particle during all previous steps, $\hat{y}$, is defined as

$$\hat{y}(t+1) = arg f(y_i(t+1)), 1 \leq i \leq s \tag{4}$$

The value of each component in every v_i vector can be clamped to the range $[-v_{max}, v_{max}]$ to reduce the likelihood of particles leaving the search space, another option is to change the direction of the velocity before it leaves the search space. The acceleration coefficients c_1 and c_2 control how far the particle will move in a single iteration, tipically these are bot setted to a value of 2.0, although variations sometimes increases the performance of the algorithm.

C. PSO in Real Number Space

In real number space we use the formulation of above and the following procedure.

1) Initialize the swarm by assigning a random position in the problem hyperspace to each particle.
2) Evaluate the fitness function of each particle.
3) For each particle, compare the particle's fitness value with its p_{best}. If the current value is better than the p_{best} value, is setted as p_{best}.
4) Identify the particle with best fitness value and set it to global best.
5) Update the velocities and the positions of each particle using (1) and (2).
6) Repeat steps 2-5 until stopping criterion (e.g. maximum number of iterations or a suffiently good fitness value).

1) Computational Characteristics :

D. Implementation details

For the CPU implementation we followed this simple variation of the asynchronous PSO algorithm :

For the GPU implementation we followed the same variation of the PSO algorithm in a synchronous way, by making the comparison with the global best outside of the particles update loop :

III. Results

IV. Conclusions and Future improvements

References

[1] James Kennedy and Russell Eberhart.
Particle Swarm Optimization. - 1995

[2] Simon Garnier, Jacques Gautrais, Guy Theraulaz
The biological principles of swarm intelligence. - 2007

[3] Yamille del Valle, Ganesh Kumar, Salman Mohagheghi, Jean Hernandez, Ronald Harley
Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. - 2008

[4] Zhengyou Zhang
A Flexible New Technique for Camera Calibration. - 2000

[5] Derek Bradley, Gerhard Roth
Adaptive Thresholding Using the Integral Image. - 2011

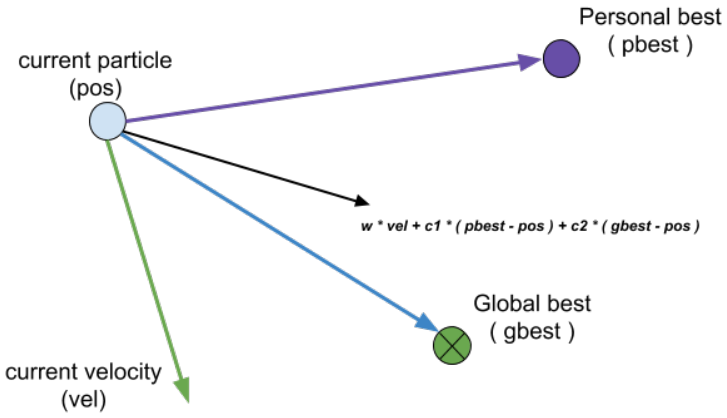

Fig. 1. PSO overview

Algorithm 1 Particle Swarm Optimization - Asynchonous Serial Version

In : *objFunction, populationSize, w,* c_1*,* c_2*, k, dimensions, domain*
Out : *Solution optima* $P_{g.cost}$
$vMin = -k * (domain.max - domain.min)/2.0$
$vMax = k * (domain.max - domain.min)/2.0$
$P_{g.pos} =$ *zeros(dimensions)* ▷ Position that gives best cost
$P_{g.cost} = Inf$ ▷ Best cost so far

$Particles = \{\}$ ▷ Empty array of particles
for $i = 1$ **to** $PopulationSize$ **do** ▷ Particles Initialization
    $p =$ **Particle()**
    $p.pos =$ **randUniform(domain)**
    $p.vel =$ **zeros(dimensions)**
    $p.cost =$ **objFunction(p.pos)**
    $p.bestpos = p.pos$
    $p.bestcost = p.cost$
    if $p.cost \leq P_{g.cost}$ **then**
        $P_{g.cost} = p.cost$
        $P_{g.pos} = p.pos$

while *!stopCondition()* **do** ▷ Optimization process
    for p **in** $Particles$ **do**
        $p.vel = w * p.vel + c_1 * (p.bestpos - p.pos) + c_2 * (P_{g.pos} - p.pos)$ ▷ Velocity update
        $p.vel =$ **ClampVector**(*p.vel, vMin, vMax*)
        $p.pos = p.pos + p.vel$ ▷ Position update
        $p.pos =$ **ClipPosition**(*p.pos, domain*)

        if $p.cost \leq p.bestcost$ **then**
            $p.bestcost = p.cost$ ▷ Update personal best
            $p.bestpos = p.pos$
            if $p.cost \leq P_{g.cost}$ **then**
                $P_{g.cost} = p.cost$ ▷ Update global best
                $P_{g.pos} = p.pos$
return $P_{g.cost}$

Algorithm 2 Particle Swarm Optimization - Synchonous Parallel Gpu Version

In : *objFunction, populationSize, w,* c_1, c_2, *k, dimensions, domain*
Out : *Solution optima* $P_{g.cost}$
$vMin = -k * (domain.max - domain.min)/2.0$
$vMax = k * (domain.max - domain.min)/2.0$
$P_{g.pos} =$ *zeros(dimensions)* ▷ Position that gives best cost
$P_{g.cost} = Inf$ ▷ Best cost so far

$hostParticles = \{\}$
$deviceParticles = \{\}$
GpuCreateParticles(*hostParticles, deviceParticles, objFunction, populationSize, dimensions, domain*)
GpuInitParticles(*hostParticles, deviceParticles, objFunction*)

while *!stopCondition()* **do** ▷ Optimization process
    GpuUpdateParticles(*hostParticles, deviceParticles, w,* c_1, c_2, *k*)s
    for p **in** $hostParticles$ **do**
        if $p.cost \leq P_{g.cost}$ **then**
            $P_{g.cost} = p.cost$ ▷ Update global best
            $P_{g.pos} = p.pos$
return $P_{g.cost}$

Algorithm 3 Particle Swarm Optimization - GpuUpdateParticles

In : *deviceParticles, coreIndx, w,* c_1, c_2, *k,* P_g
$p =$ **getDeviceParticle**(*coreIndx*)
$p.vel = w * p.vel + c_1 * (p.bestpos - p.pos) + c_2 * (P_{g.pos} - p.pos)$ ▷ Velocity update
$p.vel =$ **ClampVector**(*p.vel, vMin, vMax*)
$p.pos = p.pos + p.vel$ ▷ Position update
$p.pos =$ **ClipPosition**(*p.pos, domain*)

if $p.cost \leq p.bestcost$ **then**
    $p.bestcost = p.cost$ ▷ Update personal best
    $p.bestpos = p.pos$

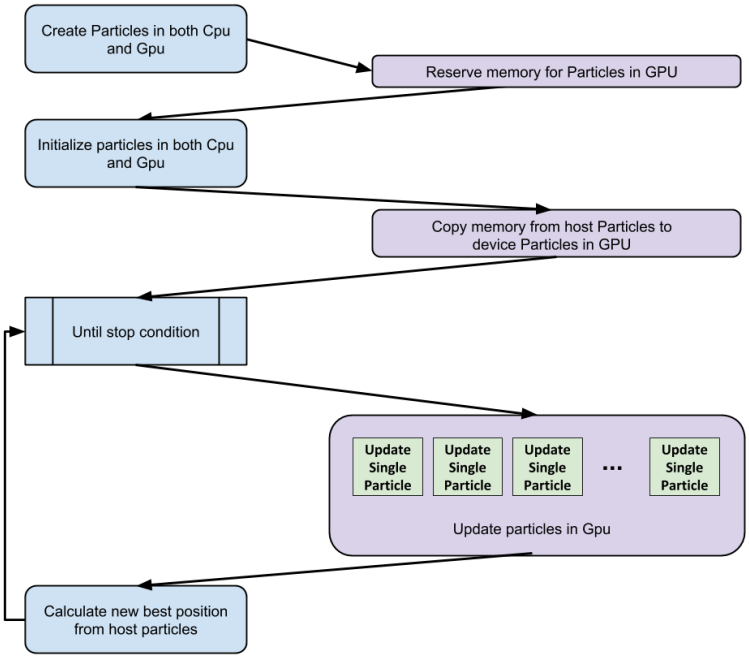

Fig. 2. PSO algorithm in GPU

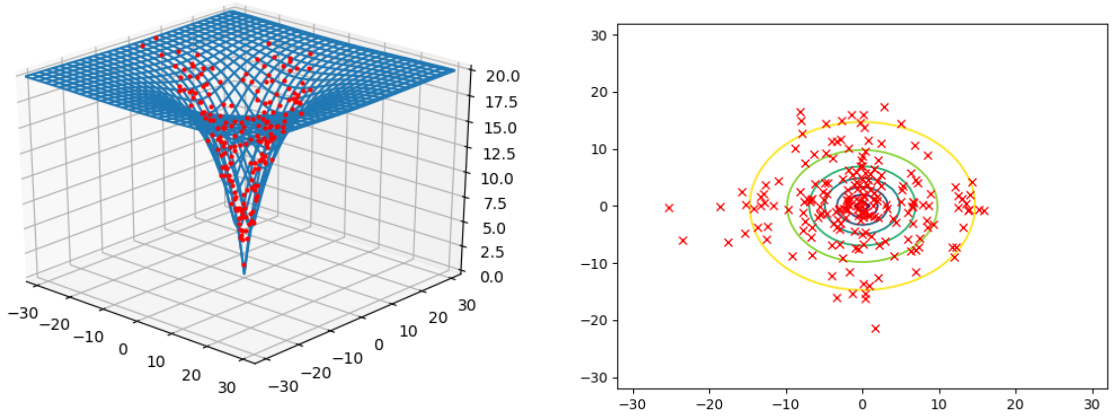

Fig. 3. PSO overview

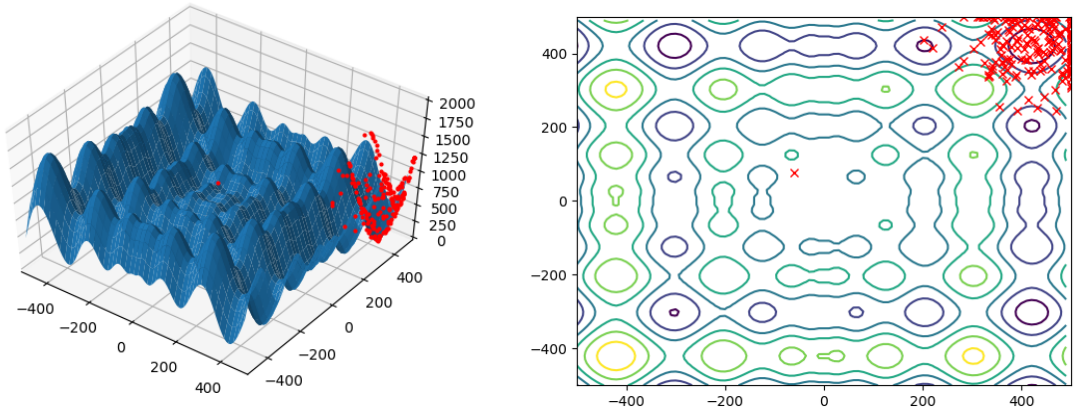

Fig. 4. PSO overview

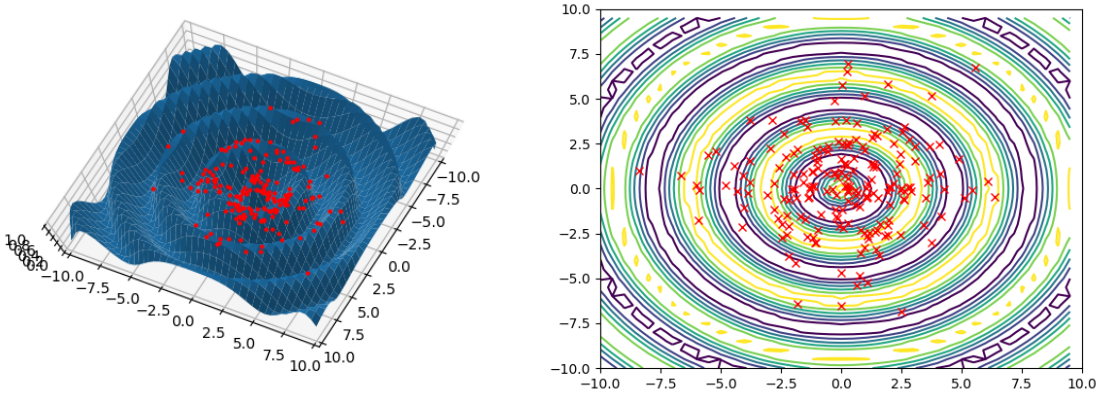

Fig. 5. PSO overview