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

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 :

B. PSO in Real Number Space

C. 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

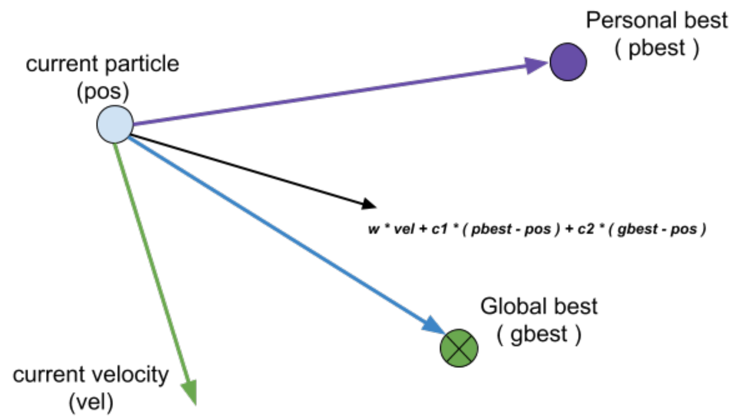

Fig. 1. PSO overview

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
    $p.bestpos = p.pos$

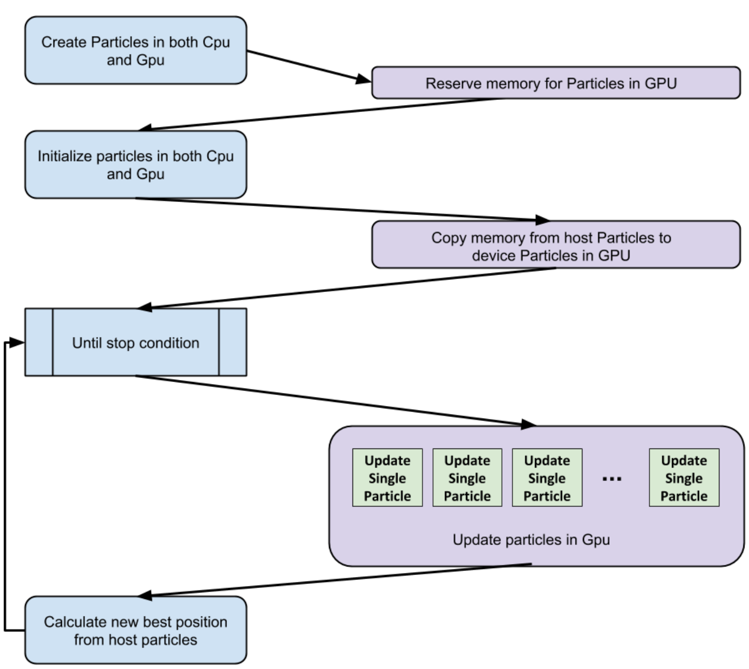

Fig. 2. PSO algorithm in GPU