Particle Swarm Optimization

A parallelized approach

Samuele Bortolotti Federico Izzo

University of Trento

December 11, 2022

Introduction

Particle Swarm Optimization

Particle Swarm Optimization is an optimization algorithm for nonlinear functions based on bird swarms.

A particle is characterized by:

- position x;
- velocity v;
- performance measure $f(x)$;
- personal best y;
- global best positions z.

The solution is achieved by perturbing each particle:

- $v' = w \cdot v + \phi_1 U_1 \cdot (y - x) + \phi_2 U_2 \cdot (z - x)$
- $x' = x + v'$

Particle Swarm Optimization

Easom function

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2)$$

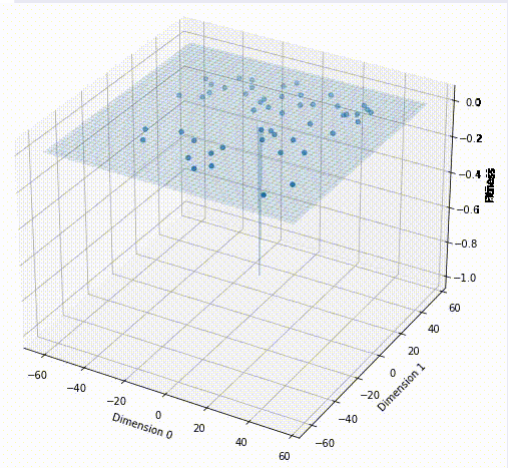


Figure 1: PSO start

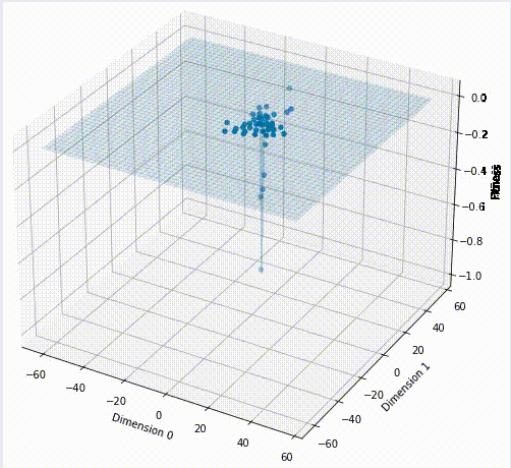


Figure 2: PSO mid

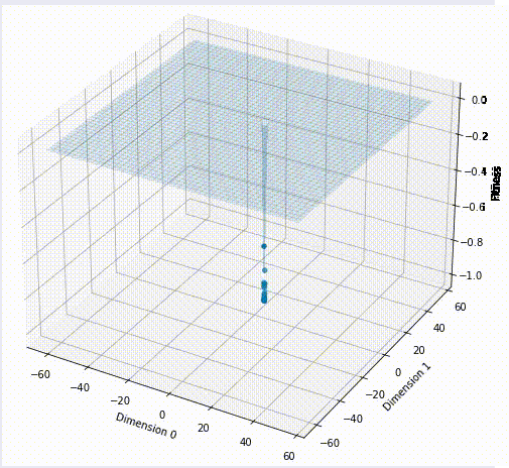


Figure 3: PSO end

State of the Art Analysis

Ref.	Year	Type
(Kennedy and Eberhart 1995)	1995	Serial
(toddguant 2019)	2019	Serial
(souusouho 2019)	2019	Serial
(kkentzo 2020)	2020	Serial
(fisherling 2020)	2020	Serial
(Moraes et al. 2015)	2014	MPI
(Nedjah, Moraes Calazan, and Macedo Mourelle 2017)	2017	MPI/MP
(abhi4578 2019)	2019	MPI/MP
(LaSEEB 2020)	2020	OpenMP
(pg443 2021)	2021	Serial,Op

1. provides only global neighborhood implementation.

State of the Art Analysis

Ref.	Year	Type
(Kennedy and Eberhart 1995)	1995	Serial
(toddguant 2019)	2019	Serial
(souusouho 2019)	2019	Serial
(kkentzo 2020)	2020	Serial
(fisherling 2020)	2020	Serial
(Moraes et al. 2015)	2014	MPI
(Nedjah, Moraes Calazan, and Macedo Mourelle 2017)	2017	MPI/MP
(abhi4578 2019)	2019	MPI/MP
(LaSEEB 2020)	2020	OpenMP
(pg443 2021)	2021	Serial,Op

1. provides only global neighborhood implementation.
2. provides PSO with different neighborhood versions but without a distance based approach.

DevOps

Work in progress...

Serial version of the algorithm

Algorithm 1 Particle Swarm Optimization (Nearest Neighbors)

1: **function** PSO($\mathcal{S}$, $\mathcal{D}$, MAX_IT, n, f, v, x, x_{min}, x_{max}, v_{max})
2: INITIALIZE($\mathcal{S}$, $\mathcal{D}$, f, v, x, x_{min}, x_{max}, v_{max})
3: $it = 0$
4: **repeat**
5: **for each** particle $i \in \mathcal{S}$ **do**
6: **if** $f(x_i) < f(pb_i)$ **then**
7: $pb_i \leftarrow x_i$
8: **end if**
9: **end for**
10: $\mathcal{S}' = \text{COPY}(\mathcal{S})$
11: **for each** particle $i \in \mathcal{S}$ **do**
12: $\mathcal{S}' = \text{SORT}(\text{S'}, \text{i})$
13: **for each** particle $j \in \mathcal{S}'$ **do**
14: **if** $f(x_j) < f(gb_i)$ **then**
15: $gb_i \leftarrow x_j$
16: **end if**
17: **end for**
18: **end for**
19: **for each** particle $i \in \mathcal{S}$ **do**
20: **for each** dimension $d \in \mathcal{D}$ **do**
21: $v_{i,d} = v_{i,d} + C_1 \cdot Rnd(0,1) \cdot [pb_{i,d} - x_{i,d}] + C_2 \cdot Rnd(0,1) \cdot [gb_d - x_{i,d}]$
22: $x_{i,d} = x_{i,d} + v_{i,d}$
23: **end for**
24: **end for**
25: $it \leftarrow it + 1$
26: **until** it < MAX_ITERATIONS
27: **return** x
28: **end function**

Hybrid parallelization

We propose an all-to-all parallel computational pattern using `MPI_Allgather`.

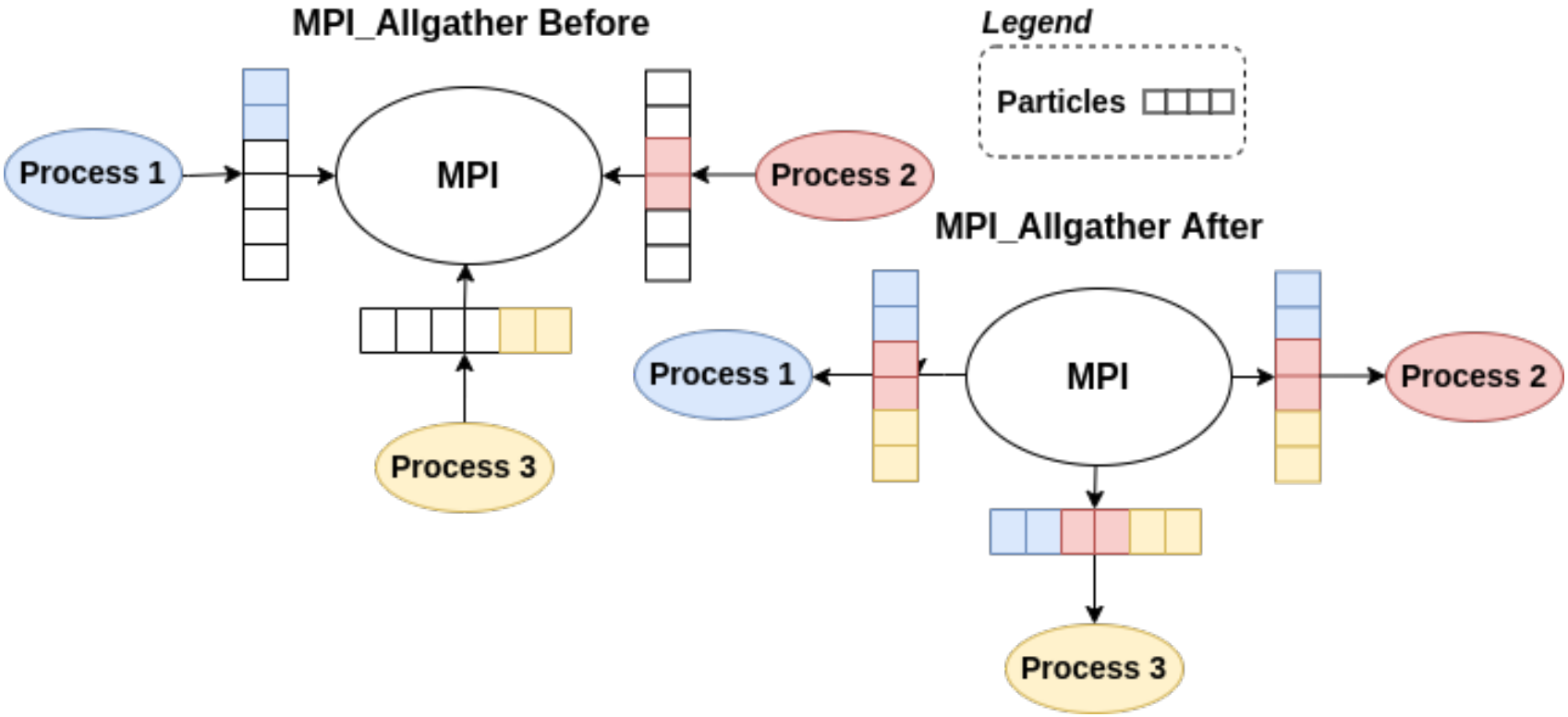
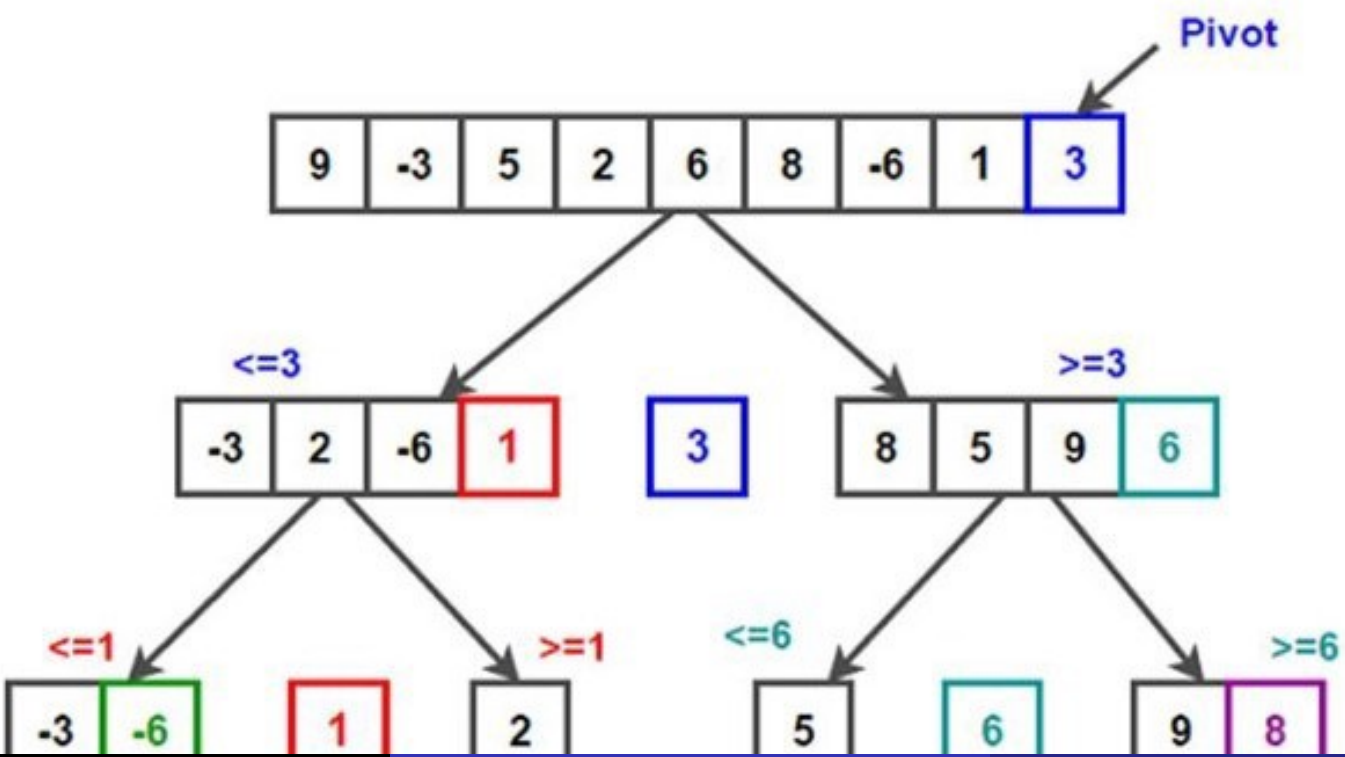


Figure 4: Parallel Architecture

Hybrid parallelization (cont'd)

Once each process knows everything about the others, PSO considers the neighbor contributions in order to update the process particles' position and velocity.

To compute the particle's neighboring positions we have employed the quicksort algorithm.



Benchmarking, first conclusions

The problem we have decided to solve consists in solving the sphere function ($f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2$) with:

- 50 particle dimensions
- 500 iterations
- 5000 particles

We have run around 1280 tests considering every possible combination of different parameters:

- processes: chosen between `[1 2 4 8 16 32 64]`;
- threads: chosen between `[1 2 4 8 16 32 64]`;
- select: chosen between `[1 2 3 4 5]`;
- places: chosen between `[pack scatter pack:excl scatter:excl]`.

Benchmarking, first conclusions (cont'd)

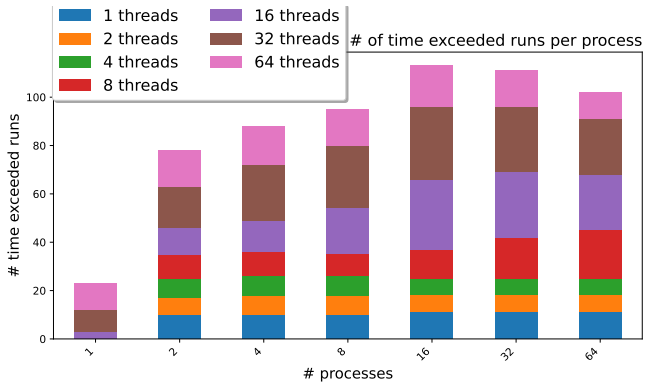


Figure 6: Number of failed run per process

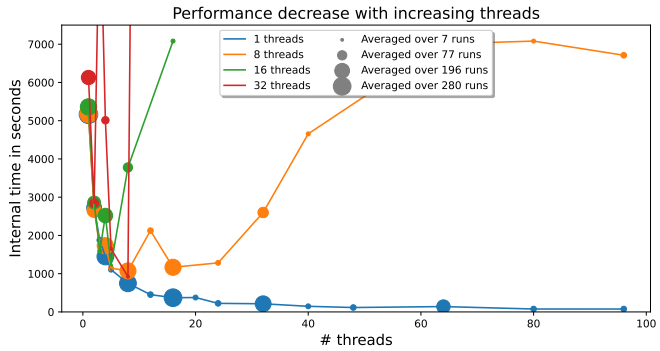


Figure 7: Thread and time exceeded correlation

Benchmarking, first conclusions (cont'd)

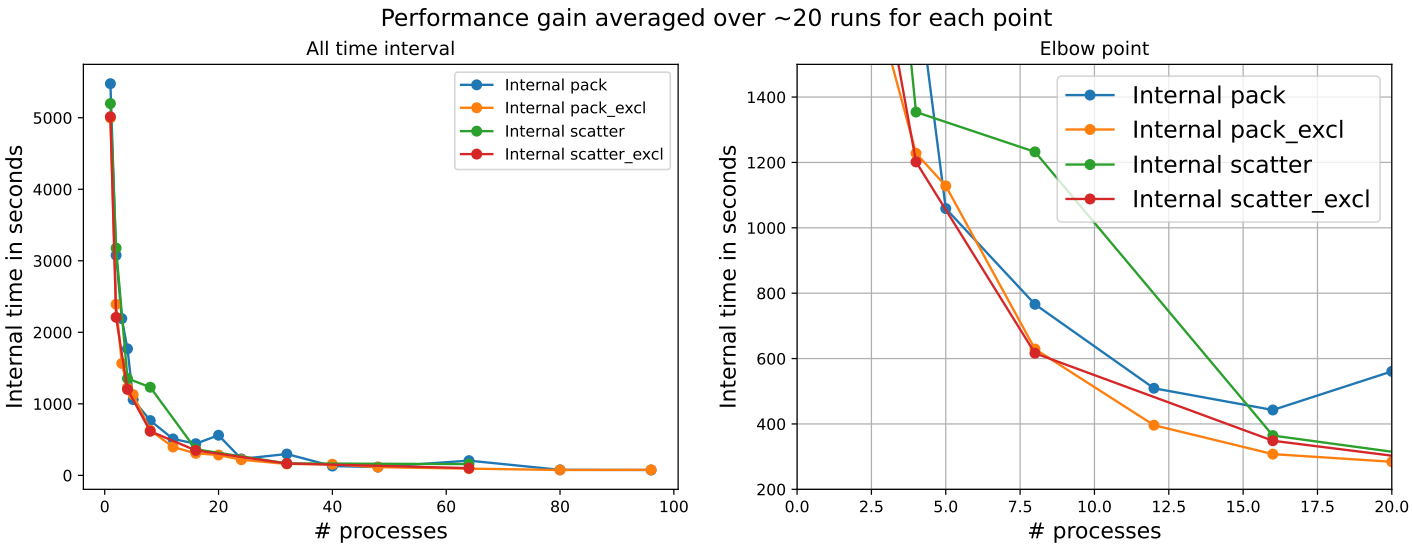


Figure 8: Processes performance

Benchmarking, final remarks

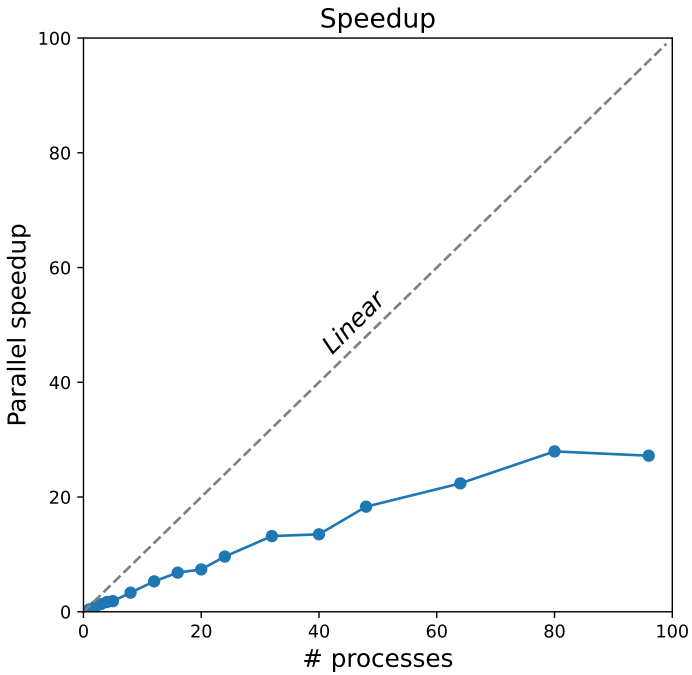


Figure 9: Speedup

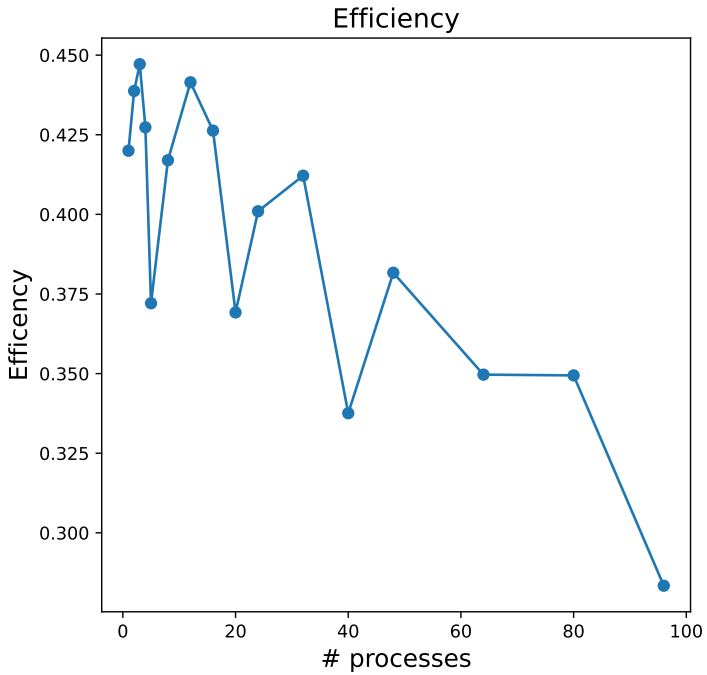


Figure 10: Efficiency

Conclusion

From our experiments we claim:

- thread parallelization does not fit well our problem;
- the program provides its best result when the number of processes is limited.

References I

abhi4578. 2019. "Parallelization-of-PSO." https://github.com/abhi4578/Parallelization-of-PSO.

fisherling. 2020. "Pso." https://github.com/fisherling/pso.

Kennedy, J., and R. Eberhart. 1995. "Particle Swarm Optimization." In *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4:1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968.

kkentzo. 2020. "Pso." https://github.com/kkentzo/pso.

LaSEEB. 2020. "Openpso." https://github.com/abhi4578/openpso.

Moraes, Antonio O. S., João F. Mitre, Paulo L. C. Lage, and Argimiro R. Secchi. 2015. "A Robust Parallel Algorithm of the Particle Swarm Optimization Method for Large Dimensional Engineering Problems." *Applied Mathematical Modelling* 39 (14): 4223–41. https://doi.org/https://doi.org/10.1016/j.apm.2014.12.034.

References II

Nedjah, Nadia, Rogério de Moraes Calazan, and Luiza de Macedo Mourelle. 2017. "A Fine-Grained Parallel Particle Swarm Optimization on Many-Core and Multi-Core Architectures." In *Parallel Computing Technologies*, edited by Victor Malyshkin, 215–24. Cham: Springer International Publishing.

pg443. 2021. "Particle-Swarm-Optimization-OpenMP." https://github.com/pg443/Particle-Swarm-Optimization-OpenMP.

souusouho. 2019. "Succing PSO." https://github.com/sousouhou/succinctPSO.

toddguant. 2019. "PSO Library for c." https://github.com/toddgaunt/cpso.