Particle Swarm Optimization
A parallelized approach

Samuele Bortolotti   Federico Izzo

University of Trento

December 14, 2022

Introduction

Particle Swarm Optimization

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

In PSO, a particle is characterized by:

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

The solution is achieved by perturbing each particle according to the neighbors:

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

Particle Swarm Optimization

Easom function

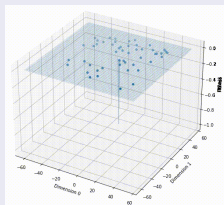$$f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2)$$
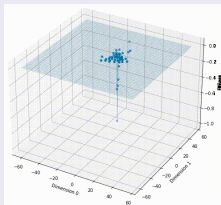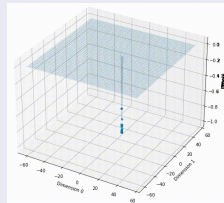

Figure 1: PSO start

Figure 2: PSO mid

Figure 3: PSO end

Pipeline

The proposed solution is provided with a pipeline for containers creation and usage suitable for a cluster environment.
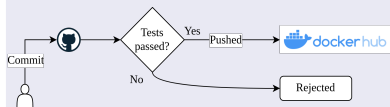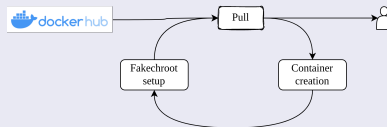

Figure 4: Container creation

Figure 5: Container pull

In order to know each process and thread state and visualize a thread-safe logging library has been employed: The logs follows a common pattern so as to be easily processed.

```
15:27:58 DEBUG : PSODATA    :: problem dimension :: 2
...
15:27:58 DEBUG : New best global solution found
...
15:27:58 INFO  : COMPUTING  :: iteration 10/10
Best fitness 4.690962
```

To recover the particles' positions during the entire program execution, we have stored each particle position at each iteration within a SQLite database.

Serial version of the algorithm

Algorithm 1 Particle Swarm Optimization (Nearest Neighbors)

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

Hybrid parallelization

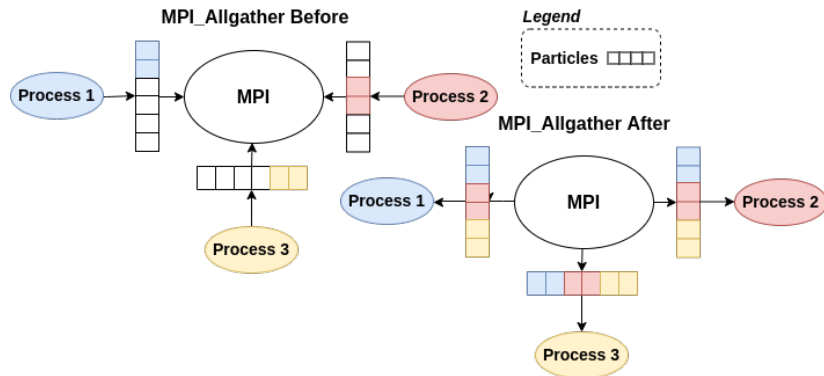We propose an all-to-all parallel computational solution using
`MPI_Allgather`.

Figure 6: Parallel Architecture

Hybrid parallelization (cont'd)

Once each process knows everything about the others, PSO considers the neighbor contributions.

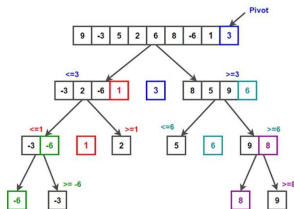To compute the particle's neighboring positions we have employed the quicksort algorithm.

Figure 7: Parallel Quicksort

Finally, the algorithm evolves by updating velocity and position.

Benchmarking, first conclusions

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

- 50 particle dimensions;
- 500 iterations;
- 5000 particles.

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

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

Benchmarking, time exceed

Many of the submitted experiments failed due to time exceeded errors. At a first sight, it seems that the failure rate is correlated with the increasing number of processes used for the computation.
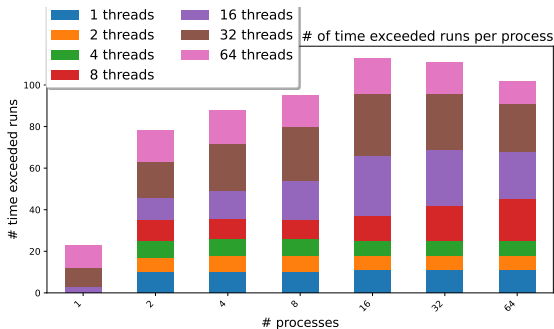

Figure 8: Number of failed run per process

Benchmarking, threads fault

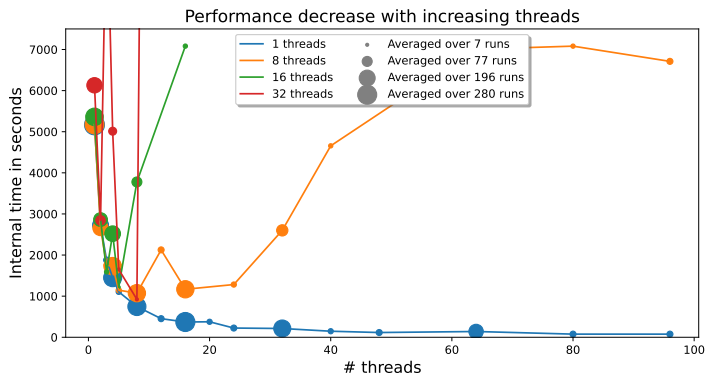A more depth analysis highlights that the problem is related to threads' overhead.

Figure 9: Thread and time exceeded correlation

Benchmarking, single thread solution

- The time required for the execution decreases if the number of processes is increased;
- The proposed solution is influenced neither by the network overhead nor exclusive nodes.
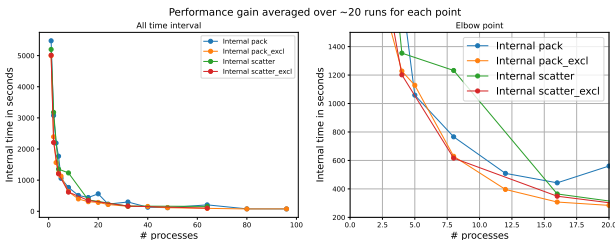

Figure 10: Processes performance

State of the Art Analysis

Ref.	Year	Type	Code	Note
Kennedy et al. (1995)	1995	Serial	No	-
toddguant (2019)	2019	Serial	Yes	1
souusouho (2019)	2019	Serial	Yes	1
kkentzo (2020)	2020	Serial	Yes	1
fisherling (2020)	2020	Serial	Yes	1
Moraes et al. (2015)	2014	MPI	No	-
Nedja et al. (2017)	2017	MPI/MP	No	-
abhi4578 (2019)	2019	MPI/MP,CUDA	Yes	1
LaSEEB (2020)	2020	OpenMP	Yes	2
pg443 (2021)	2021	Serial,OpenMP	Yes	1

only global neighborhood (1) no distance-based implementation (2)

Figure 11: Speedup

Figure 12: Efficiency

Conclusion and future work
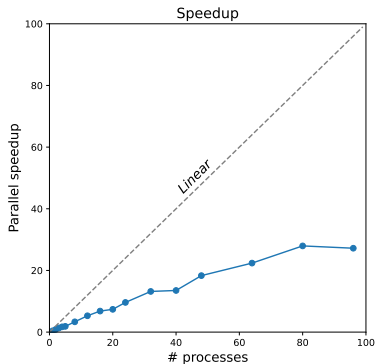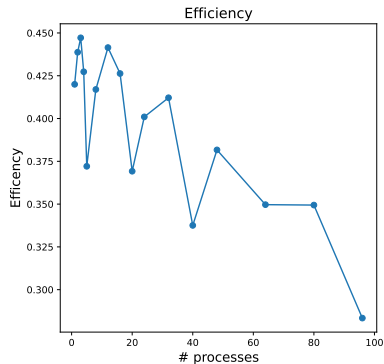
Up until this point, we produced a hybrid OpenMP-MPI algorithm to solve complex continuous optimization problems.

From the benchmarking analysis we claim:

- thread parallelization does not fit well our solution;
- benchmarking the algorithm is far from being trivial;
- the program provides its best result when the number of processes is limited.

As a future work, it would be interesting to:

- complement the already present architecture with different types of neighborhoods;
- analyze which configuration brought the best results.

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.

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.