Value Iteration on a Grid World using Python.

Parallel Computing Laboratory

Carlo Lucchesi

February, 2026

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Da un secolo, oltre.

UNIVERSITÀ DEGLI STUDI FIRENZE
Da un secolo, oltre.

Goal of The project

Implementing Synchronous and Asynchronous Value Iteration (VI) algorithms, and then parallelizing them with different approaches. Python 3.14.2 free threading build was used.

Versions Implemented.

- Synchronous VI.
  - Joblib (with processes and threads).
  - ThreadPoolExecutor.
- Asynchronous VI.
  - Correctly managing race conditions.
  - Neglecting locks.

Figure: Python 3.14.2 (https://www.python.org/downloads/release/python-3142/)

Grid World

The experiments focused in finding the optimal value function for a grid world of a give size.

Possible cell inside a grid world.

- Goals (absorbing state).
- Traps (absorbing state).
- Free cells.
- Walls.

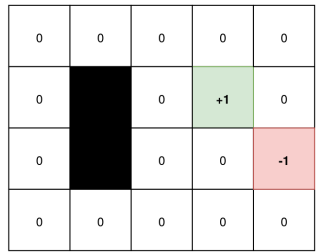

Figure: Example of a Grid World environment.

Versions Implemented

Synchronous Version

Due to how synchronous update is done, the application of the Bellman operator can be easily parallelized.

Different parallel implementations:

- Joblib (Loky backend).
- Joblib (Thread backend).
- ThreadPoolExecutor.

Figure: Joblib logo. (https://joblib.readthedocs.io/en/stable/)

Versions Implemented

Asynchronous Version

To remain adherent to the implementation, asynchronous VI requires synchronization to avoid race conditions. However, neglecting them doesn't have meaningful effect on the result. Different parallel implementations:

- ThreadPoolExecutor (properly lock acquiring).
- ThreadPoolExecutor (ignoring locks).

Experiments conditions

Computer used for tests.

- **CPU**: AMD Ryzen 5 7520U 4 core (8 threads) 2.8 GHz.
- **Cache**: 64 B of cache line, 256KB L1, 2MB L2, 4MB L3.
- **RAM**: 8GB.
- **OS**: GNU/Linux (Pop_OS! specifically, an Ubuntu derivative).
- **Compiler**: Clang 14.0.0-1ubuntu1.1.
- **OpenMP**: version 4.5.

Every test was conducted 6 times, and the first result was excluded. The remaining 5 was mediated.
During all test execution, CPU boosting **was disabled**.

Results

Synchronous VI

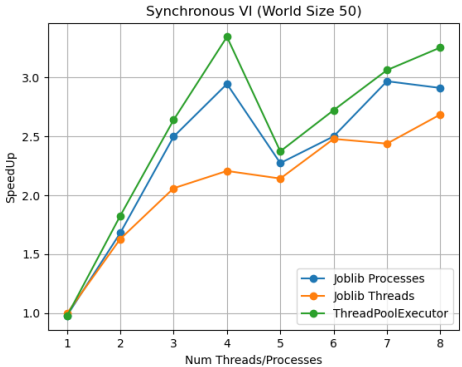

Figure: Speed-up for increasing number of parallel streams.

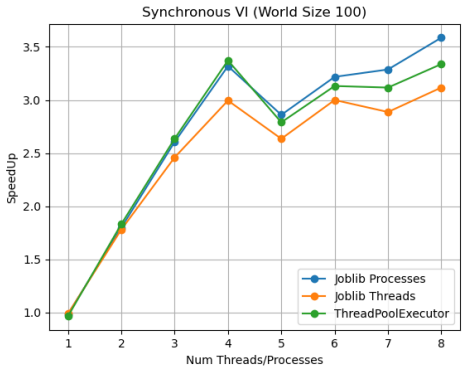

Figure: Speed-up trend for a bigger world.

Results

Synchronous VI

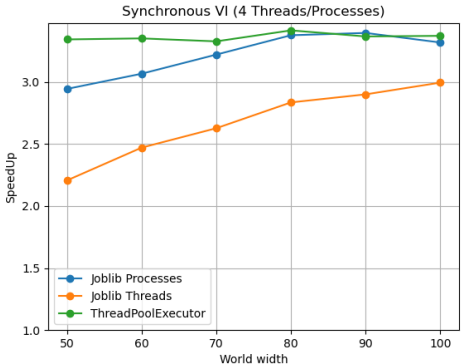

Figure: Speed-up on different world sizes, 4 parallel streams.

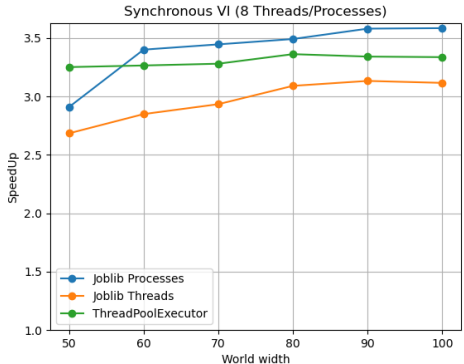

Figure: Speed-up on different world sizes, 8 parallel streams.

Results

Asynchronous VI

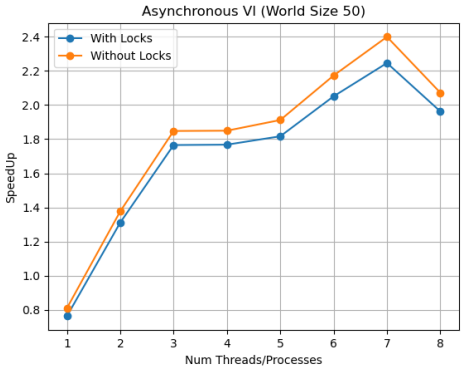

Figure: Speed-up for increasing number of parallel streams.

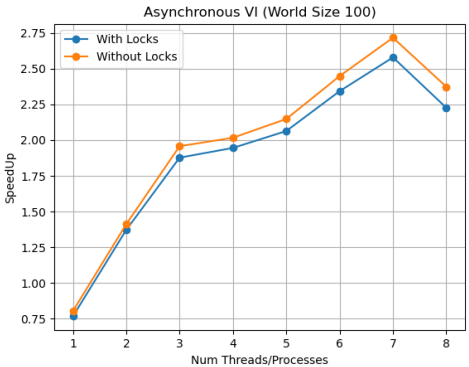

Figure: Speed-up trend for a bigger world.

Results

Asynchronous VI

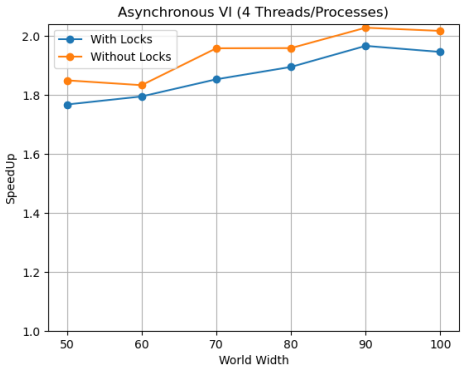

Figure: Speed-up on different world sizes.

Conclusions

From these results, we can conclude that:

- Synchronous update is much easier than asynchronous, and brings more benefits.
- Free threading python works, but is not competitive with previous state-of-the-art parallelism frameworks.
- Neglecting python locks doesn’t change the performance much.