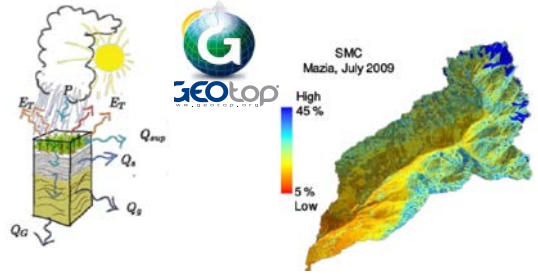


An empirical study on the GEOtop hydrological model optimal estimation and uncertainty reduction using supercomputers

Giacomo Bertoldi ¹, Stefano Campanella ^{2,3}, Emanuele Cordano ⁴, and Alberto Sartori ^{2,5}

Background and aims

- **GEOtop**: distributed integrated hydrological model
- **Calibration challenges**: large parameter space + computational expensive model
- **GOAL**: develop a HPC tool for the calibration of **GEOtop**, including all the available parameters.



Approach: multiple evolutionary algorithms (particle swarm ...)

Tools: [GEOtoPy](#) (wrapper) + [Jupyter](#) ecosystem + [Nevergrad](#) + [Dask](#)



Application

- Application for soil moisture time series optimization for agricultural sites in South Tyrol (Italy)
- Deployment on [HPC cluster](#) (up to 1024 CPU cores).
- **Challenge**: dealing with code failures.

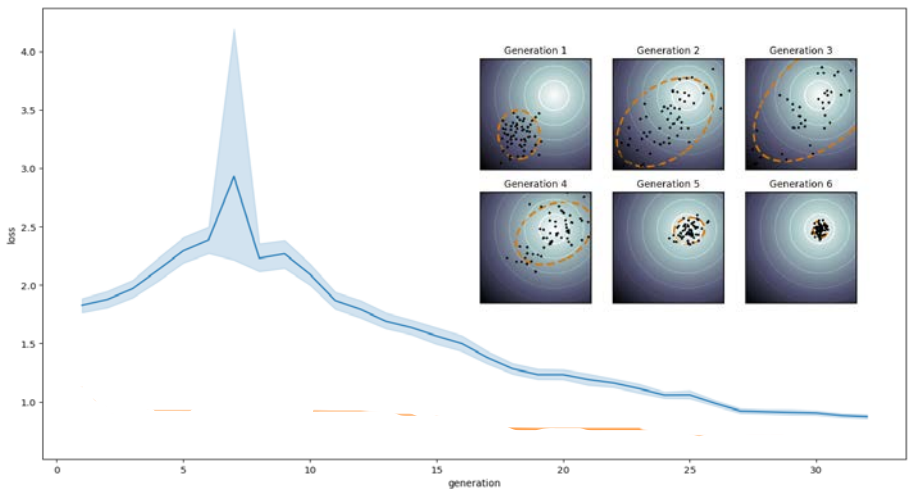
Results

- Parallel execution using futures
- Speculative execution.
- Scaling analyses.

Conclusions and outlook

- A HPC tool for **GEOtop** calibration has been developed,
- **libraries & tools** used in ML have been adapted to the purpose,
- it has been proven to **scale up** to 1024 CPUs (Ulysses V2),
- **performance models** have been proposed and discussed.

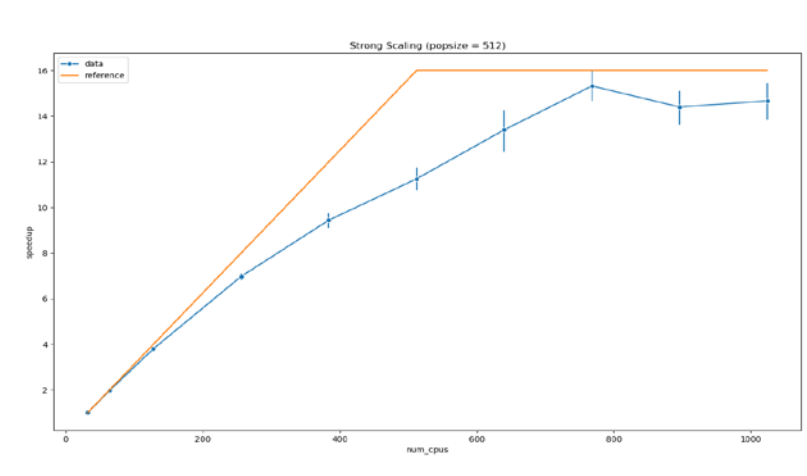
Evolutionary algorithm convergence example



Soil moisture optimization at 2 cm depth

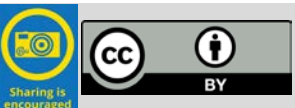


Observed (blue) vs. theoretical scaling behavior (orange)



References: Campanella, S.; [Calibration of the GEOtop model using evolutionary algorithms on supercomputers](#), MHPC Thesis, SISSA and ICTP. Code on Github: [MHPC-project](#)

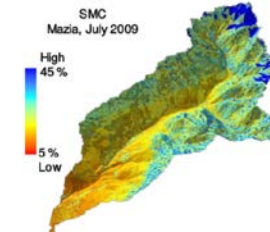
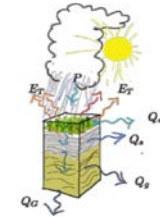
Acknowledgments: The research was supported by OGS, Eurac and CINECA under HPC-TRES program award 2019-33 and by the EU ERDF Project DPS4ESLAB number 1094.



Do you want to use the optimization approach?

The Model:

- [GEOtop](#): an open-source distributed integrated hydrological model.
- Code and documentation available on [GitHub](#).
- Last development version [V3.0](#) in C++.



The Tools:

[GEOtoPy](#) (Dead simple, paper-thin wrapper to work with GEOtop from Python.)

[Jupyter](#) ecosystem for easy Python scripting

[Papermill](#) a tool for parameterizing and executing Jupyter Notebooks in background.

[Scrapbook](#) library to produce notebook's workflows.

[Nevergrad](#): A gradient-free optimization platform.

[Dask](#): advanced parallelism for analytics, enabling HPC in python.

The Optimization framework

Code and documentation available on Github: [MHPC-project](#)



Do you want to know more on the HPC calibration approach used?

GEOTOP CALIBRATION USING EVOLUTIONARY ALGORITHMS ON SUPERCOMPUTERS

Stefano Campanella

Giacomo Bertoldi, Alberto Sartori



You can download the whole MHPC thesis here:

Campanella, S.; [Calibration of the GEOTop model using evolutionary algorithms on supercomputers](#)



OUTLINE

1. **Intro & Motivations**

1. GEOtop and its calibration
2. Evolutionary algorithms and need for HPC

2. **Methodology & Implementation**

1. Workflow: Jupyter, Nevergrad, GEOtoPy
2. Parallel Computing in Python and Dask
3. The Optimization Loop

3. **Results**

1. Example of Calibration
2. Scaling Analysis and Modeling

GEOTOP CALIBRATION

10'000 FOOT VIEW OF GEOTOP CALIBRATION

GEOTop: meteorological forcing \rightarrow soil water content

loss = $\|\text{predictions} - \text{observations}\|$ (es. 1 - KGE)

N° of parameters $\approx O(\text{volume}) + O(\text{surface})$

EXAMPLE: SOIL MOISTURE BEFORE CALIBRATION



CHALLENGES

large search space, high sensitivity to parameters

1D Workload: CPU-bounded, serial, ETA \approx 1 min

GOAL OF THIS PROJECT

Develop a tool for calibration of GEOtop, including all the available parameters and in the least possible time

EVOLUTIONARY ALGORITHMS AND NEED FOR HPC

RANDOM OPTIMIZATION PROCESS

search space $D \subseteq \mathbb{R}^n$, compact

objective $f : D \mapsto [0, +\infty)$, continuous

optimization $X_i : \Omega \mapsto D \mid f(X_i) \leq f(X_{i-1})$

EXAMPLE: RANDOM SEARCH

$$X_0 = U_0, X_i = \begin{cases} U_i & \text{if } f(U_i) \leq f(X_{i-1}) \\ X_{i-1} & \text{otherwise} \end{cases}$$

EVOLUTIONARY ALGORITHMS

The complexity of the fitness landscape suggest the use of Evolutionary Algorithms (EA)

In EA, search space is sampled multiple times each generation, hence f can be evaluated in parallel ← **HPC kicks in**

Speedup with n CPU is $\min(p, n)$, **in theory**

DEALING WITH FAILURES

The algorithms/implementation must use a re-sampling strategy

TOOLS

CODE USED IN THIS PROJECT

1. GEOtoPy: A Python wrapper developed for this project
2. Jupyter Ecosystem: Papermill, Scrapbook
3. Facebook Nevergrad and SALib: CMA-ES, etc.
4. Dask: distributed, jobqueue

PYTHON FOR HPC

PARALLEL COMPUTING IN PYTHON

- The GIL: a mutex on CPython interpreter execution
- Multithreading: may make sense if code release the GIL
- Multiprocessing: serialize state, and spawn new interpreters

DASK DISTRIBUTED

- Part of the Dask library
- Able to scale on large systems (1/3 of Summit)
- Implements and extend the `concurrent.futures` interface

OPTIMIZATION LOOP

ASK & TELL OOP INTERFACE

```
while not optimizer.stop():  
    x = optimizer.ask()  
    y = f(x)  
    optimizer.tell(x, y)
```

PARALLEL EXECUTION USING FUTURES

```
1  for _ in range(num_generations):
2      for i in range(popsize):
3          x[i] = optimizer.ask()
4
5      for i in range(popsize):
6          futures[i] = client.submit(lambda x: (x, f(x)), x[i])
7
8      to_tell = []
9      for future in as_completed(futures):
10         to_tell.append(future.result())
```

SPECULATIVE EXECUTION

```
1 completed_queue = as_completed(futures)
2 to_tell = []
3 for future in completed_queue:
4     x, y = future.result()
5     if isfinite(y):
6         to_tell.append(future.result())
7     if len(to_tell) + completed_queue.count() < popsize:
8         for _ in range(num_new_futures):
9             new_x = optimizer.ask()
10            new_future = client.submit(lambda x: (x, f(x)),
11                                       new_x)
12            completed_queue.add(new_future)
```

RUNNING ON THE CLUSTER

SETUP & RUN SCRIPT

```
1  #!/usr/bin/env bash
2  #SBATCH --tasks-per-node=1
3  #SBATCH --cpus-per-task=32
4  #SBATCH --mem-per-cpu=400MB
5  #SBATCH --hint=nomultithread
6  #SBATCH --hint=compute_bound
7  #SBATCH --mail-type=ALL
8  #SBATCH --verbose
9
10 SITE=$1
11 PARAMETERS_PATH=$2
12 ALGORITHM=$3
13 POPSIZE=$4
14 NUM_GENERATIONS=$5
15 TIMEOUT=$6
16 OUTPUT_DIR=$7
17
18 export TMPDIR=/dev/shm
19 export PYTHONPATH=$PYTHONPATH:$MHPCPROJECT_ROOT
20 export DASK_CONFIG=$MHPCPROJECT_ROOT/config/dask
21
22 INPUT="$MHPCPROJECT_ROOT/notebooks/calibration.ipynb"
23 MODEL_PATH="$MHPCPROJECT_ROOT/data/$SITE/inputs"
24 OBSERVATIONS_PATH="$MHPCPROJECT_ROOT/data/$SITE/observations/
obs.csv"
25 NUM_PROCS=4
26 NUM_CPUS=$((SLURM_CPUS_PER_TASK * SLURM_NTASKS))
27 NUM_WORKERS=$((NUM_PROCS * SLURM_NTASKS))
28 BUDGET=$((NUM_GENERATIONS * POPSIZE))
29 SCHEDULER_WAIT_RETRY_MAX=10
30
31 mkdir -p "$OUTPUT_DIR"
```

```
32 OUTPUT="$(mktemp -p "$OUTPUT_DIR" \  
33 "$SITE-$ALGORITHM-$BUDGET-$NUM_CPUS-XXX.ipynb")" \  
34 \  
35 mkdir -p "$MHPCPROJECT_ROOT/scheduler_files" \  
36 SCHEDULER_FILE="$(mktemp -p \  
37 "$MHPCPROJECT_ROOT/scheduler_files" \  
38 scheduler-XXX.json)" \  
39 ulimit -n 128000 \  
40 \  
41 dask-scheduler --scheduler-file="$SCHEDULER_FILE" \  
42 --no-dashboard \  
43 --no-show \  
44 --interface=ib0 & \  
45 sleep 30 \  
46 \  
47 SCHEDULER_WAIT_RETRY_COUNT=0 \  
48 while \  
49 ! python "$MHPCPROJECT_ROOT"/scripts/check_address.py \  
50 "$SCHEDULER_FILE" && \  
51 [[ $SCHEDULER_WAIT_RETRY_COUNT -lt $SCHEDULER_WAIT_RETRY_MAX \  
52 ]] \  
53 do \  
54 SCHEDULER_WAIT_RETRY_COUNT=$((SCHEDULER_WAIT_RETRY_COUNT + \  
55 1)) \  
56 sleep 30 \  
57 done \  
58 python "$MHPCPROJECT_ROOT"/scripts/check_address.py \  
59 "$SCHEDULER_FILE" \  
60 \  
61 srun dask-worker --scheduler-file="$SCHEDULER_FILE" \  
62 --local-directory="/tmp" \  
63 --death-timeout=180 \  
64 --no-dashboard \  
65 --no-show \  
66 --nprocs=$NUM_PROCS \  
67 --nthreads= \  
68 $((SLURM_CPUS_PER_TASK / NUM_PROCS)) \  
69 --interface=ib0 &
```

```
67 papermill --no-progress-bar \  
68     -p model_path "$MODEL_PATH" \  
69     -p timeout "$TIMEOUT" \  
70     -p observations_path "$OBSERVATIONS_PATH" \  
71     -p parameters_path "$PARAMETERS_PATH" \  
72     -p algorithm "$ALGORITHM" \  
73     -p popsize "$POPSIZE" \  
74     -p num_generations "$NUM_GENERATIONS" \  
75     -p scheduler_file "$SCHEDULER_FILE" \  
76     -p num_cpus $NUM_CPUS \  
77     -p num_workers $NUM_WORKERS \  
78     -p performance_report_filename \  
79     "${OUTPUT%.ipynb}-performance-report.html" \  
80     "$INPUT" "$OUTPUT" \  
81 jupyter nbconvert --to html "$OUTPUT" \  
82 rm "$SCHEDULER_FILE"
```

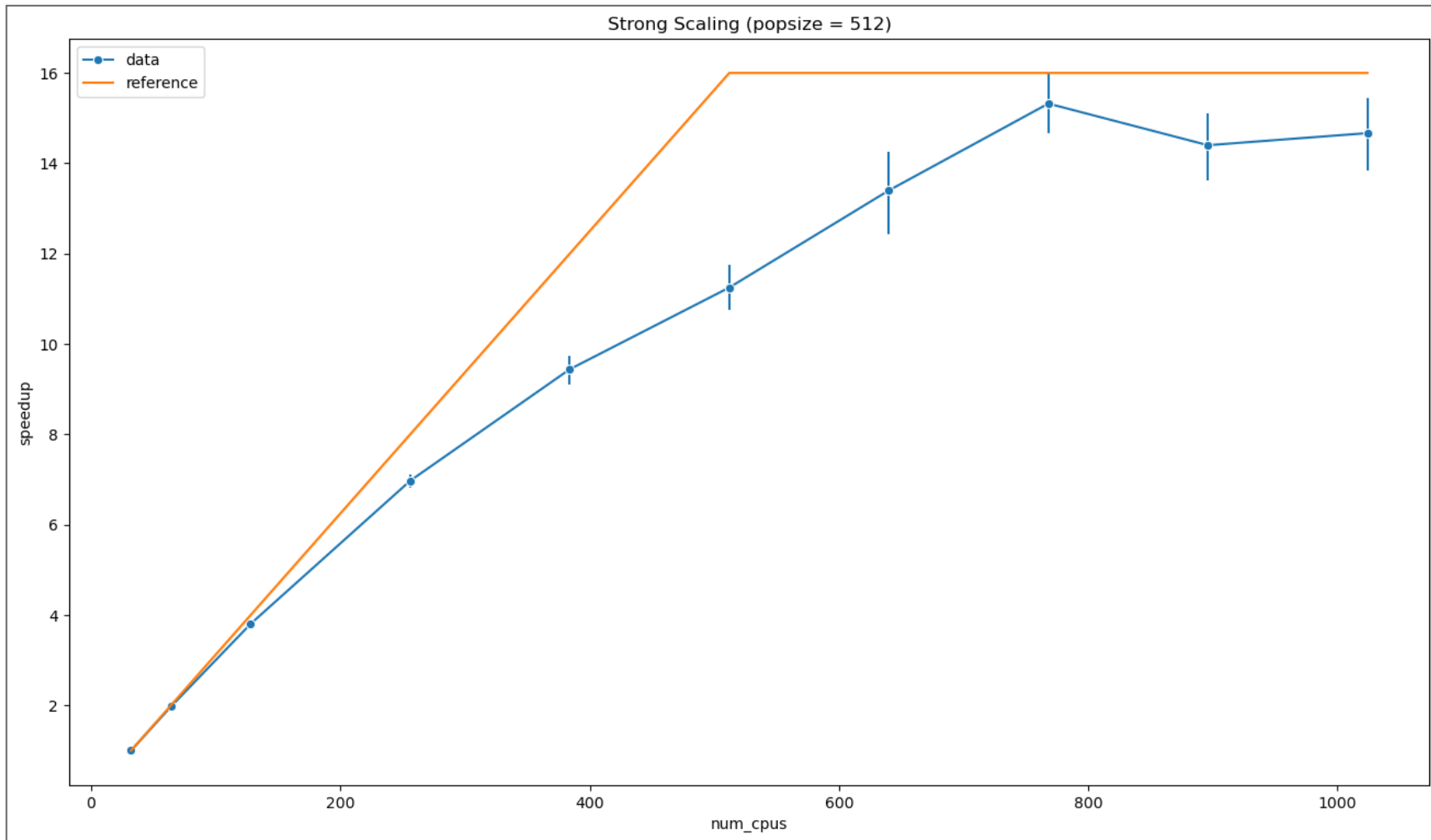
SCALING ANALYSIS

$T(p, n)$ calibration time with p individuals and n CPUs

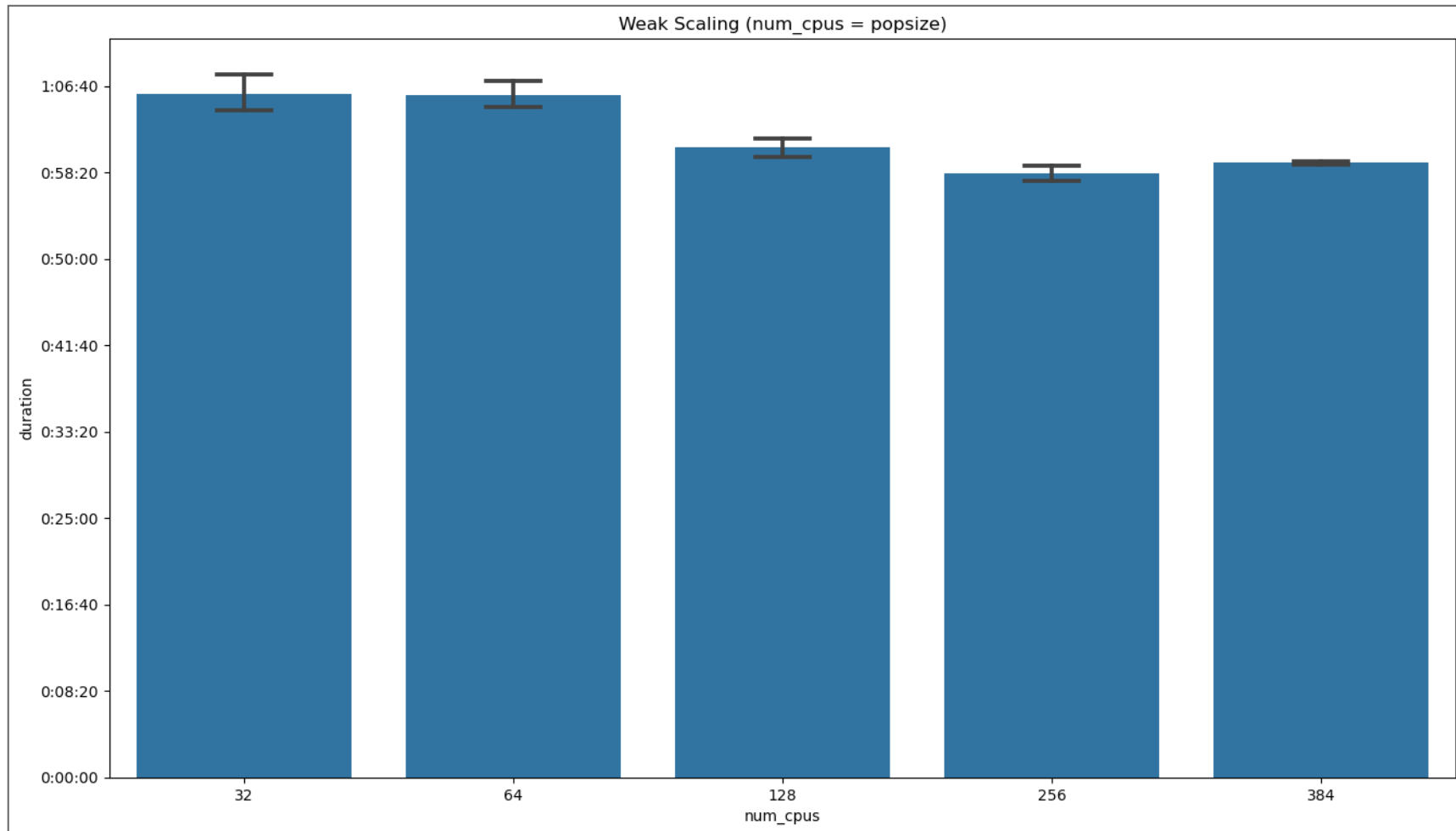
$S(p, n) := \frac{T(p, n_0)}{T(p, n)}$, speedup using $n_0 = \mathbf{32}$ CPUs as reference

$S(p, n) = \frac{\min(p, n)}{n_0}$, perfect speedup

STRONG SCALING



WEAK SCALING



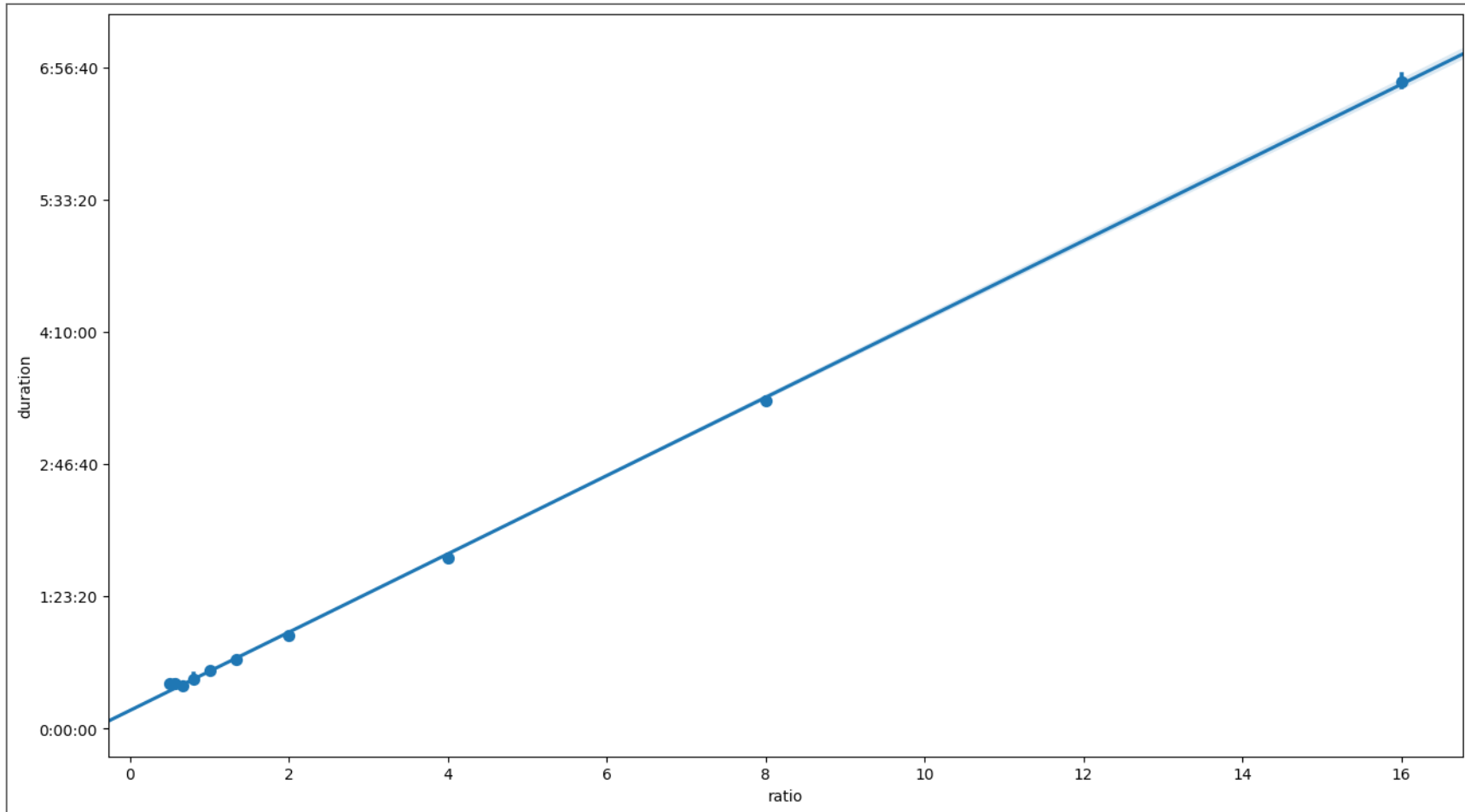
PERFORMANCE MODELING

LINEAR MODEL

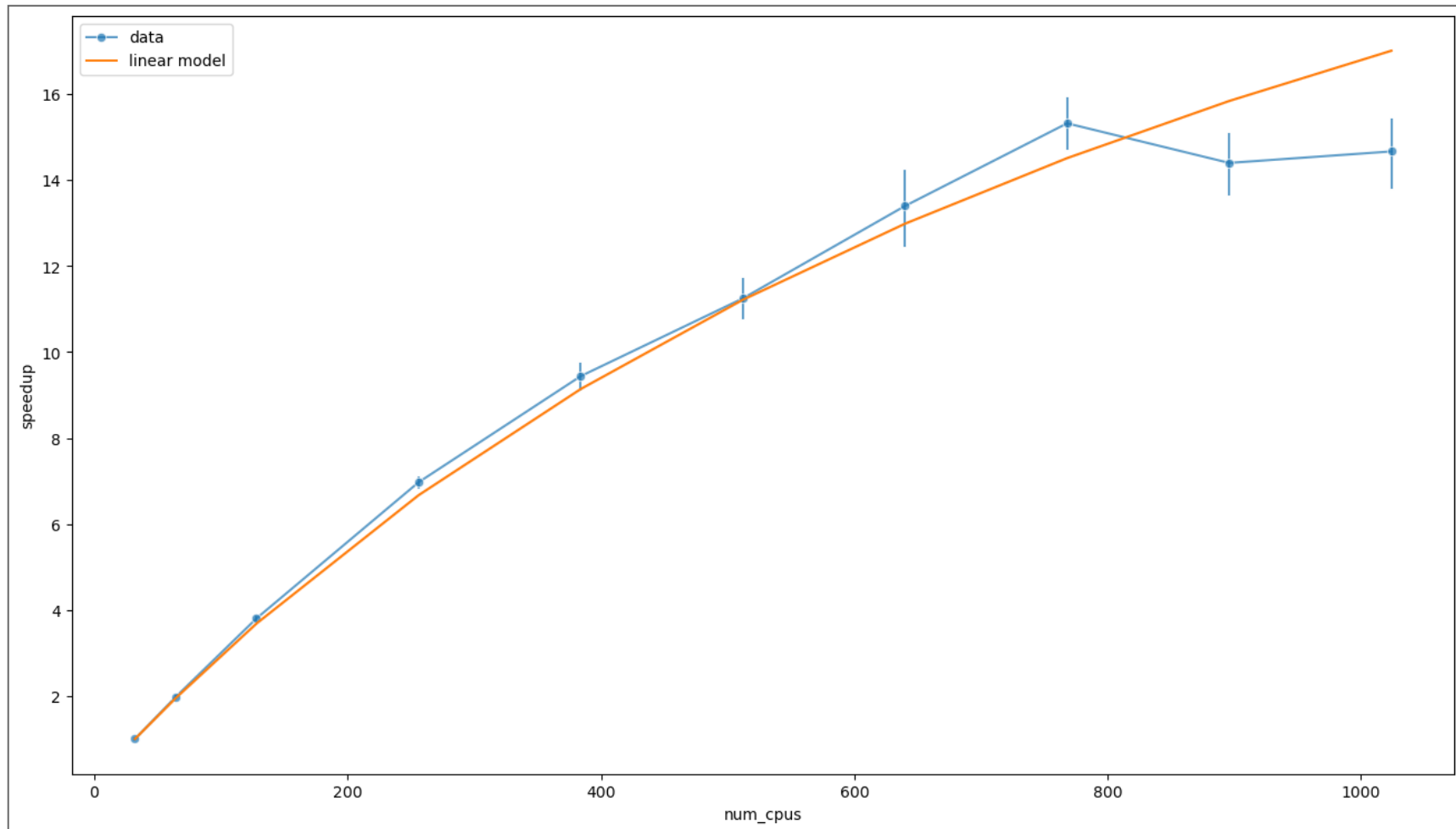
The simplest performance model compatible with perfect weak scaling
 $T(p, n) = T(\frac{p}{n})$ and $T(p, n) \sim \mathcal{O}(p)$ is

$$T(p, n) = T_0 + T_1 \frac{p}{n}$$

LINEAR MODEL FIT



LINEAR MODEL SPEEDUP



Comparing the linear model speedup

$$S(p, n) = \frac{T_0 + T_1 \frac{p}{n_0}}{T_0 + T_1 \frac{p}{n}} = \frac{1}{\frac{T_0}{T_0 + T_1 \frac{p}{n_0}} + \frac{T_1 \frac{p}{n_0}}{T_0 + T_1 \frac{p}{n_0}} \frac{n_0}{n}}$$

with the Amdahl law $S(n) = \frac{1}{(1-f) + f \frac{n_0}{n}}$ we get

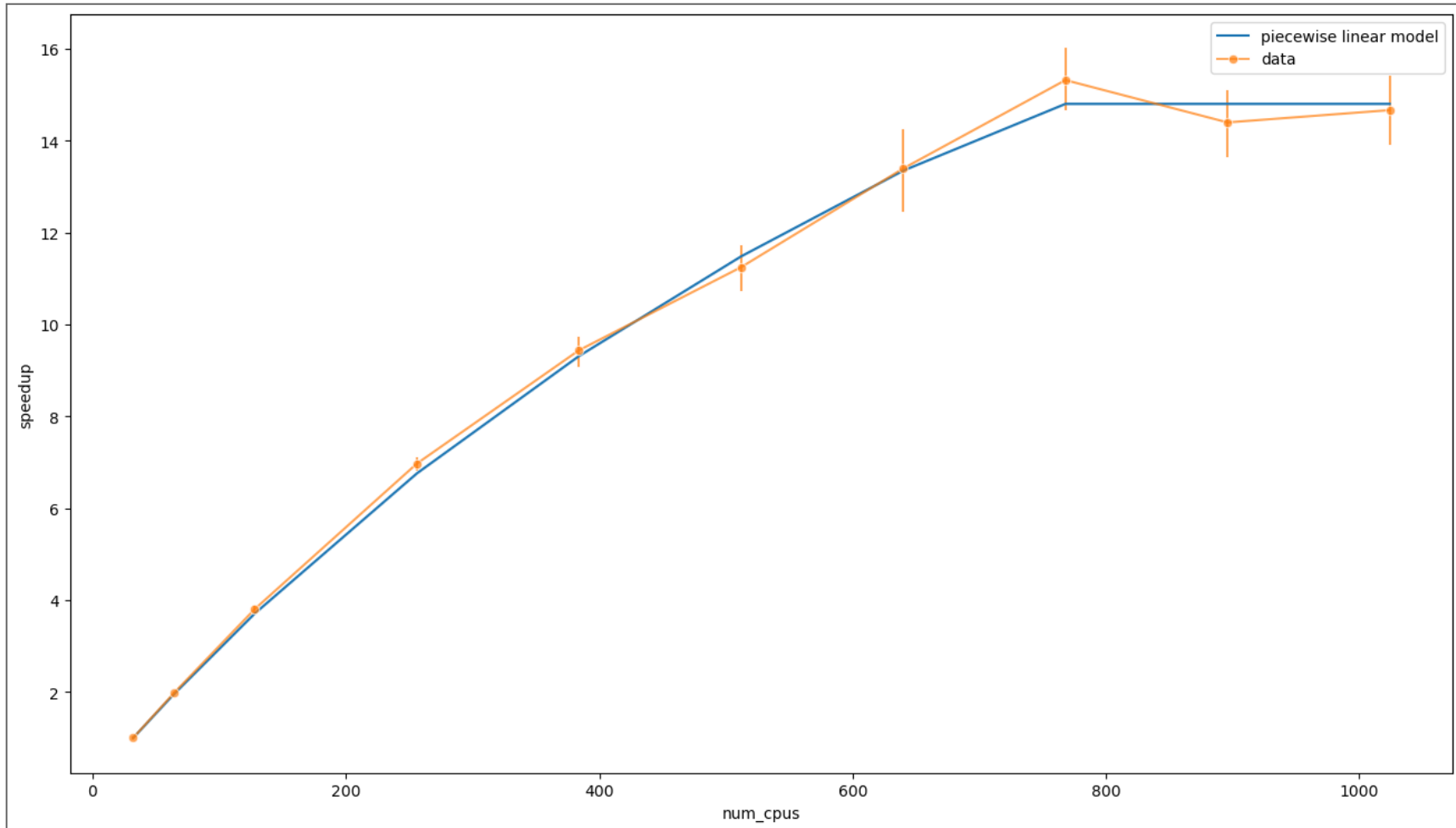
$$f = \frac{T_1 \frac{p}{n_0}}{T_0 + T_1 \frac{p}{n_0}} \approx 97\%$$

A MODEL FOR LARGE CPUS NUMBERS

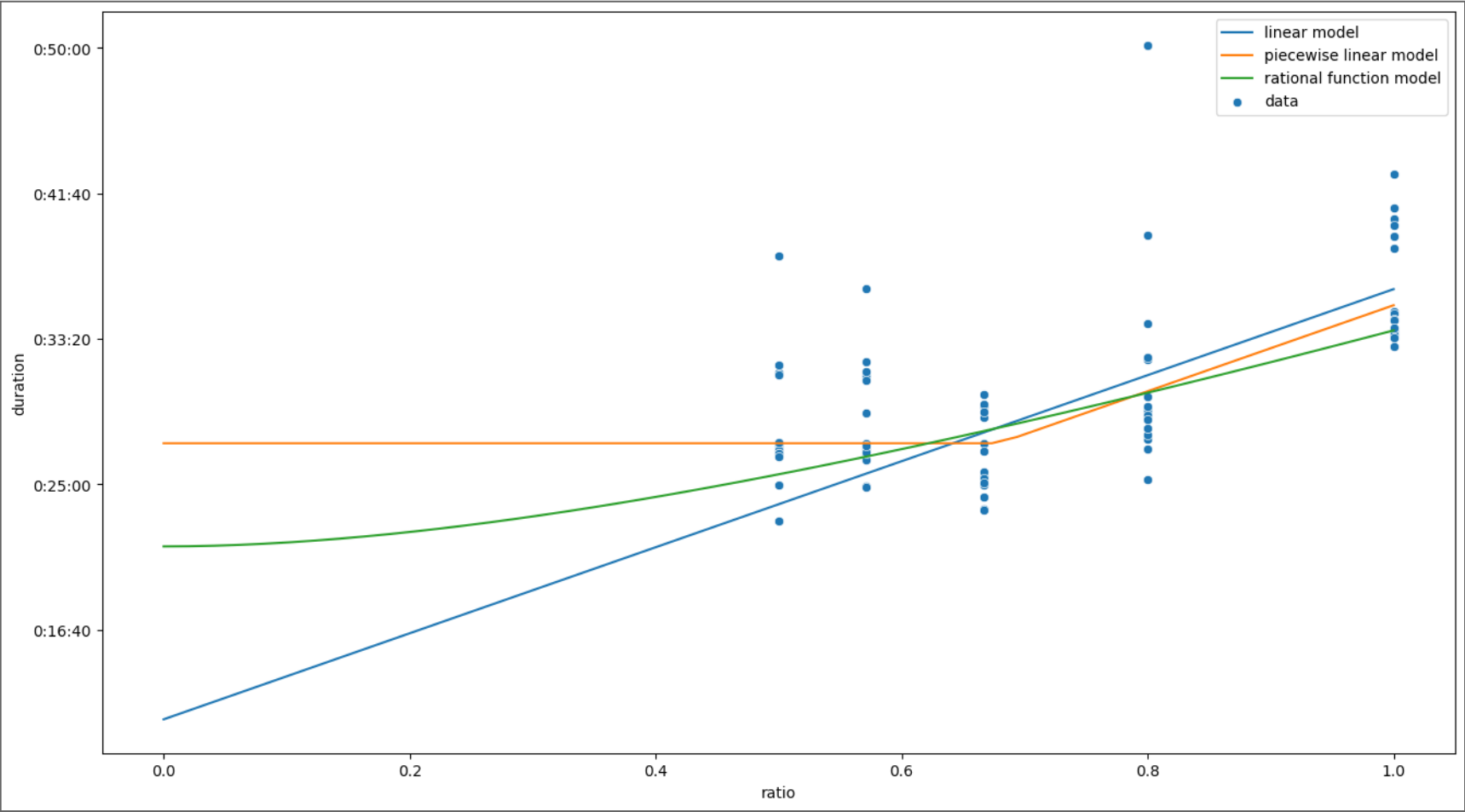
$$T_{\text{piecewise}}(p, n) = \begin{cases} T_0 + T_1 \left(\frac{p}{n} - x_0 \right) & \text{if } \frac{p}{n} \geq x_0 \\ T_0 & \text{otherwise} \end{cases},$$

where the effective population size is $p_{\text{eff}} \approx \frac{p}{x_0}$

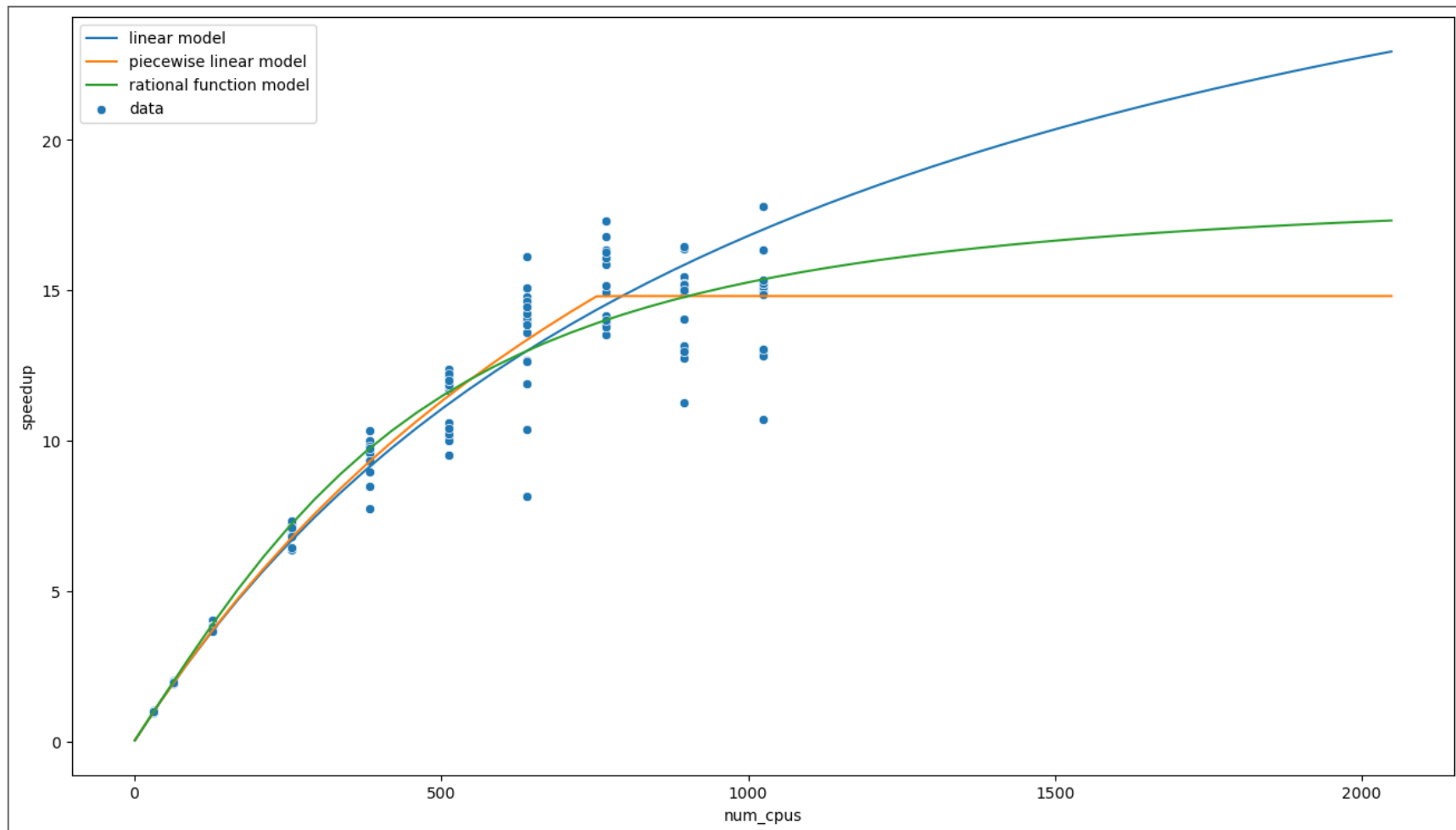
PIECEWISE LINEAR MODEL SPEEDUP



LARGE CPUS NUMBERS COMPARISON



LARGE CPUS NUMBERS COMPARISON



WRAP UP

- An automatic tool for GEOtop calibration has been developed,
- libraries & tools used in ML have been adapted to the purpose,
- it has been proven to scale up to 1024 CPUs (Ulysses V2),
- performance models have been proposed and discussed.

ACKNOWLEDGMENTS

The research reported in this work will be supported by OGS, Eurac and CINECA under HPC-TRES program award number 2019-33
EURAC supported this work using funding from the European Regional Development Fund, Operational Programme Investment for growth and jobs ERDF 2014-2020 under Project number ERDF1094, Data Platform and Sensing Technology for Environmental Sensing LAB – DPS4ESLAB.