ST7 EI - High Performance Simulation

Midterm defense

Low-cost optimization of the performance of an acoustic wave propagation code

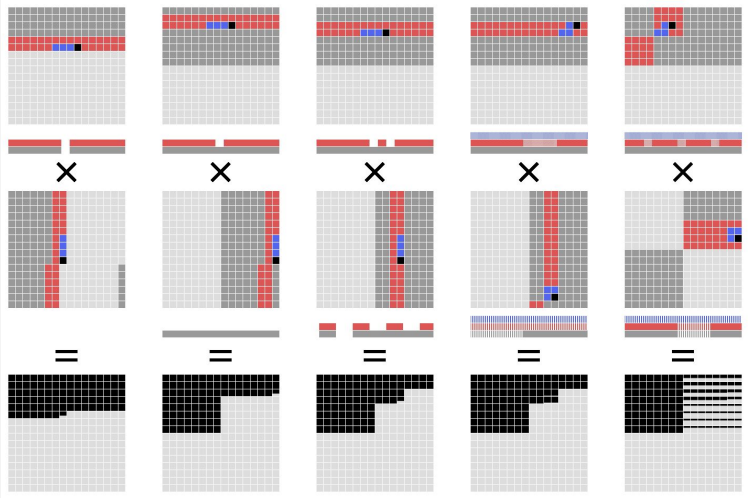
Benjamin Feldman
João Pedro Regazzi
Lucas Martins
Mateus Goto

Technical context

- The base code is a seismic application modeling wave propagation CPU-intensive application (using the finite differences method).
- As seen in the course, cache blocking plays an important role in code performance.
- Goal: apply parallelization and HPC principles to **cache blocking size optimization**.
- In particular, our interest lies in **local methods.**

Cache blocking optimization

- Ensures efficient use of cache memory, reducing the need for frequent main memory access.
- Tailors block size to specific CPU/GPU cache dimensions for optimal performance.
- Minimizes the management overhead associated with handling too many small blocks.
- Avoids large blocks that lead to rapid cache evictions and subsequent performance degradation.



Initial steps

The first step was the development of a python code allowing to execute the C++ code and measure the correspondent performance in GFlops.

In order to do so, the **subprocess** library was used for compiling and running the C++ code. Additionally, the Python **regular expressions** library was used for extracting the GFlops value from the C++ code output.

```python
def extract_fitness(text):
    pattern = r'(\d+(\.\d+)?)\s*GFlops'
    fitness = re.search(pattern, text).group()
    return fitness
```

```python
def run_process():

    command = ['bin/iso3dfd_dev13_cpu_avx2.exe']
    command.extend(parse_arguments())

    result = subprocess.run(command,
                            stdout=subprocess.PIPE,
                            text=True)
    result = result.stdout
    gflops = float(extract_fitness(result)[:5])

    print(f"GFlops: {gflops}")
    return gflops
```

First Optimization Strategy

- **Simple Hill Climbing** Strategy: choose a **random starting point**, then look for neighbouring points. Therefore, at each iteration, a neighbor three-dimensional point (one dimension for each cache blocking size) is chosen.

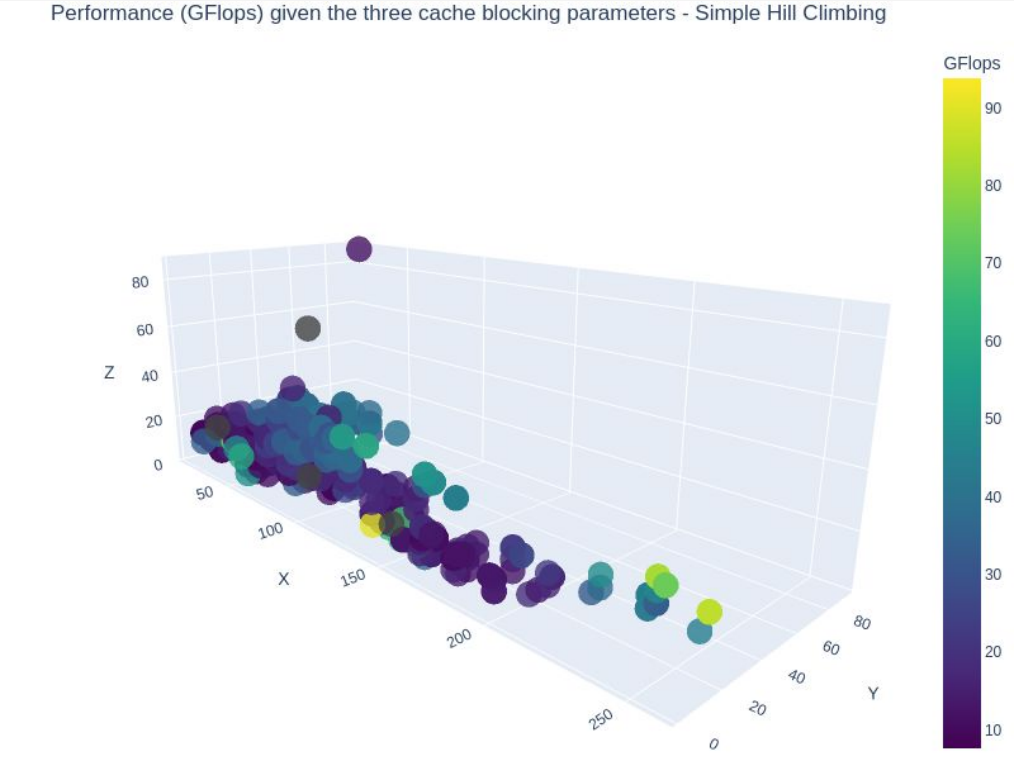
- Subsequently, the C++ code is executed and performance is measured. If performance beats that of the original point, the algorithm **moves to this new poin**t and repeats the process.

- The algorithm stops after **10 stable runs** (10 failed attempts to find a better neighbour)

  Usage of **MPI**: parallelism can be applied in order to separate the search space in different regions. For instance, the 3D-space is divided into N equal-sized regions, and each node is responsible for searching this subrregion.

  Advantages: gain in performance, as the search space for each node is divided by the number of nodes N.

Simple Hill Climbing results



Optimum values:

[240, 24, 17]

93.87 GFlops

Greedy Hill Climbing

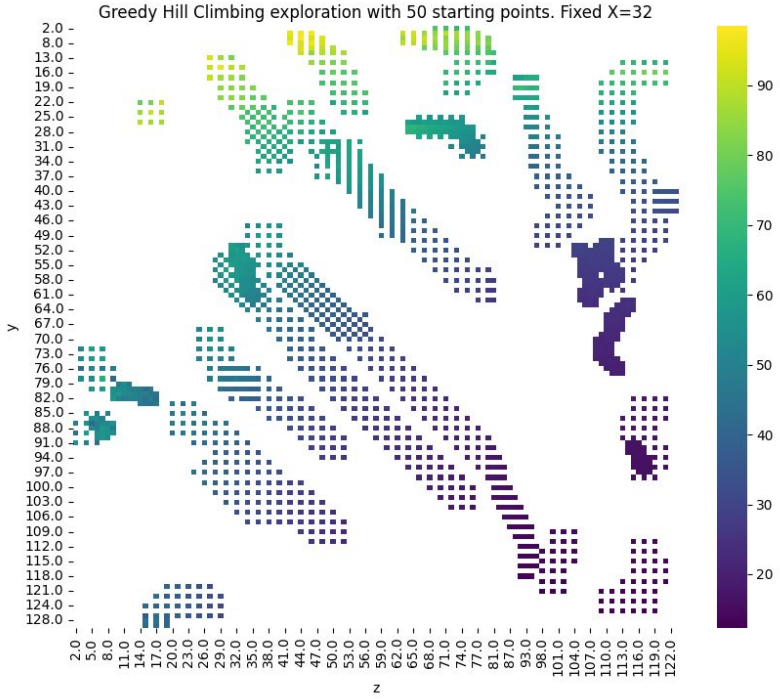
- **Starting Point Selection:** Begins with a random initial point in a three-dimensional space, representing different cache blocking sizes.

- Iteratively explore neighboring cache configurations within a step size. Evaluate performance of each configuration using the fitness function. Update to the better performing configuration, tracking improvements.

- **Termination**: Conclude after a set number of iterations or when no better configurations are found.

- Parallelize using **MPI**, distributing the search space across different processes to expedite finding the best solution.
- **Advantages:** simplicity and efficiency.
- **Disadvantages:** lack of exploration (terminates quickly) ; local optima ; high dependency on the initial conditions (random seed)

Greedy Hill Climbing results



Optimum values:

[32, 44, 19]

98.52 GFlops

Guided Hill Climbing

- **Basic idea:** the cost function evolves as the optimization process evolves.

- After exploring a hill (local maxima), a constant baseline value is attributed to the cost function around the explored region, in order to prevent future iterations to return to this neighbourhood.

- Implementation: simple hill climbing is used as a basis, however the cost function becomes **dynamic**: once reaching a point, before executing the C++ code, the algorithm checks if that point belongs to any ‘forbidden’ region, in which case the cost function returns a constant value, equal to the local minima of that neighborhood.

Guided Hill Climbing results

Optimum values:

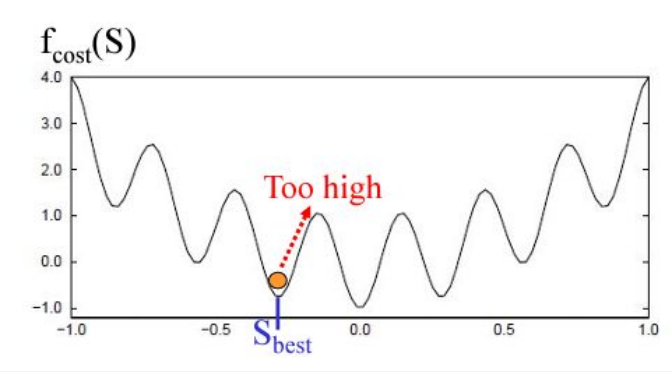
[176, 26, 18]

50.64 GFlops

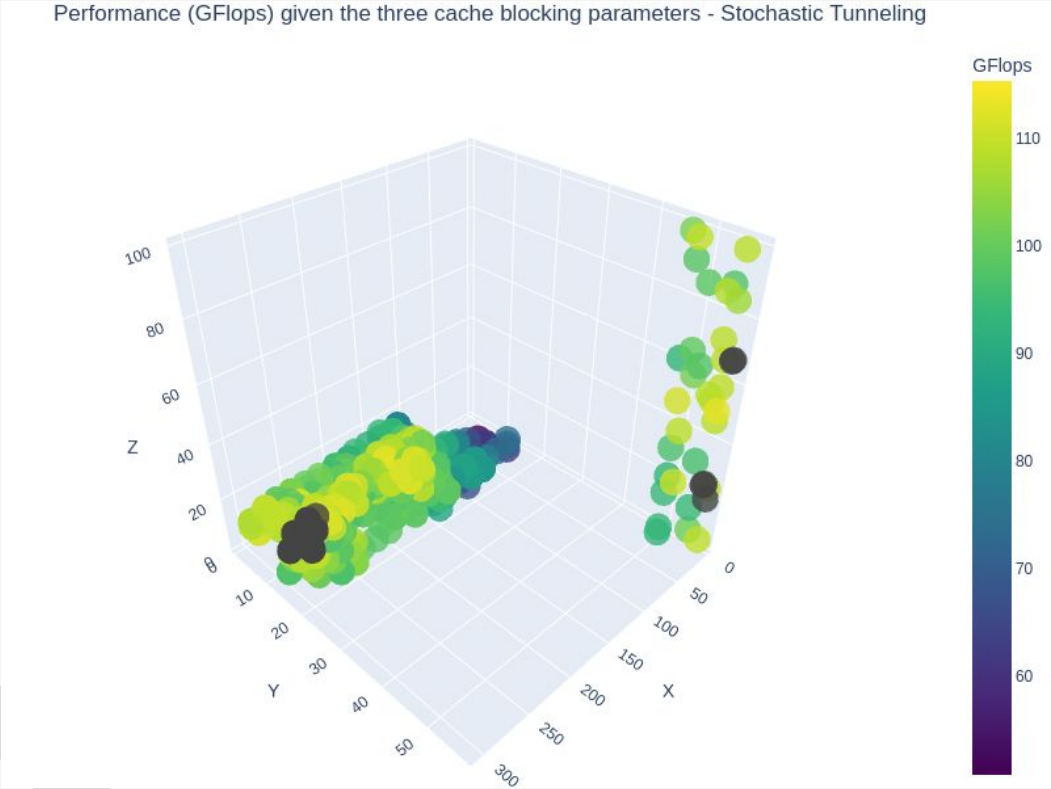
Stochastic Tunneling

- **Strategy:** In this approach, we are employing a **simulated annealing** technique to conduct the local search method instead of using **hill climbing**. Additionally**,** a adapted **stochastic tuning** is implemented to modify the cost function and escape potential local maxima.

- An **improvement in the quantity of Gflops obtained** has been observed compared to other methods due to the incorporation of two techniques aimed to get a global maxima.



Stochastic Tunneling results



Optimum values:

[240, 6, 6]

115.36 GFlops

Grid search

- **Strategy:** This is a **brute-force** approach that explores a predefined parameter space to find the optimal set of cache block parameters of the problem.

- MPI was employed to parallelize grid search, distributing the workload by giving each node their own subspace of the problem.

- Given a maximum cache block of 256, step sizes of (16, 4, 4) and the deployment of 8 nodes, we have that:

$$\text{total\_time} = \frac{256}{16} \cdot \frac{256}{4} \cdot \frac{256}{4} \cdot \frac{1}{8} \cdot \text{average\_runtime} \approx 3h52min$$

Advantages: With the capability to systematically explore the entire grid exhaustively, we ensure the identification of the optimal cache block for the problem.

Grid search results

- First we used big steps (32) and navigated through the grid from 32 to 256. The following are the best parameters found, in ascending order:

  (064 032 032) (096 032 032) (128 032 032) (160 032 032)

- We concluded that a more detailed search, with fine-grained increments, was needed where n1 thrd_block is high [240, 512] while n2 and n3 thr_blocks are low [1, 32].

- The best found result was **102.52 GFlops** with parameters (240 007 021)

Method comparison

Method	Optimum cache blocking size 1	Optimum cache blocking size 2	Optimum cache blocking size 3	Max GFlops
Hill Climbing	240	24	17	**93.87**
Greedy Hill	32	44	19	**98.53**
Guided Hill Climbing	176	26	18	**50.64**
Stochastic Tunneling	240	6	6	**115.36**
Grid Search	240	7	21	**102.52**

Conclusions

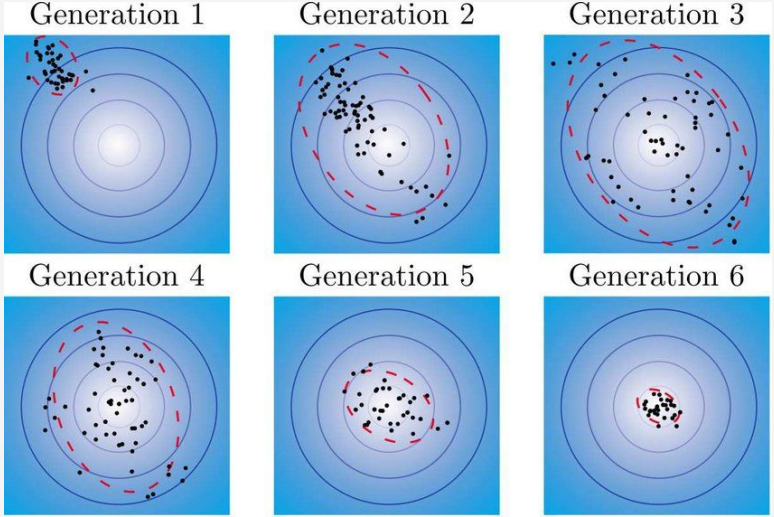
- Stochastic tunneling has proven itself to be the best algorithm among the four approaches tested.

- Even though algorithms yielded significantly different results, we observe that all of them converge to the same optimal zone:

- **High** values of cache blocking size 1
- **Low** values of cache blocking size 2 and 3

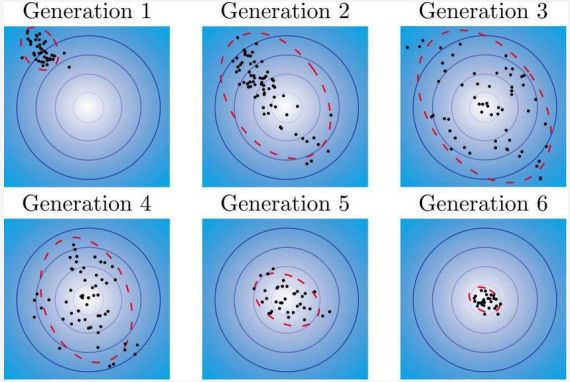
Roadmap

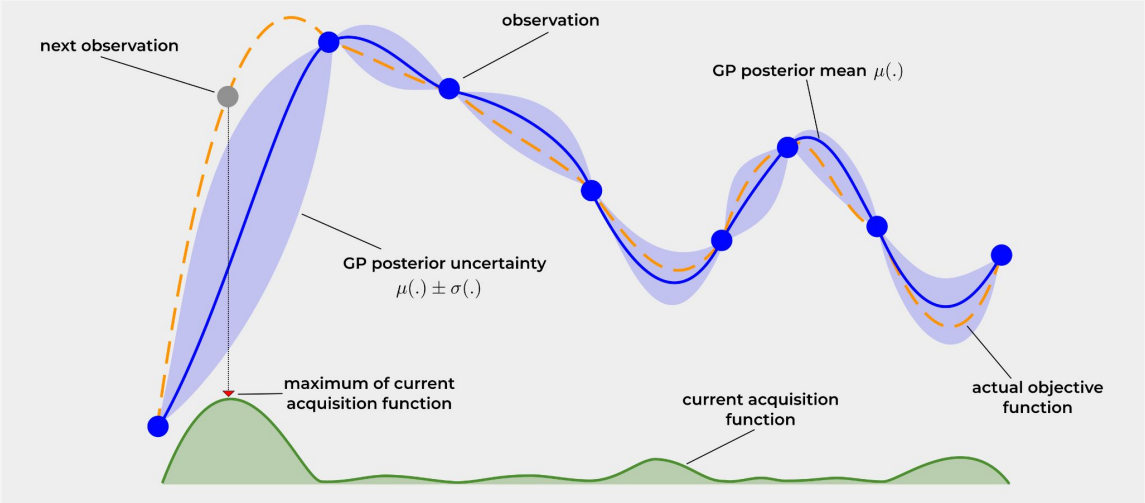
- Test more sophisticated methods, such as **CMA-ES**



Roadmap

- Test more sophisticated methods, such as **CMA-ES**
- **Combinatorial Bayesian optimization**





Roadmap

- Test more sophisticated methods, such as **CMA-ES**
- **Combinatorial Bayesian optimization**
- Keep refining our current approaches, tuning parameters in order to achieve better performances