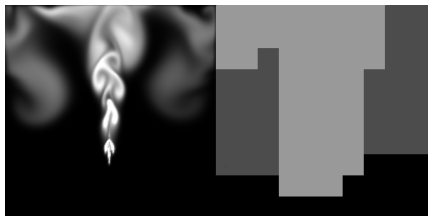
Adaptive Code Refinement

César Sabater

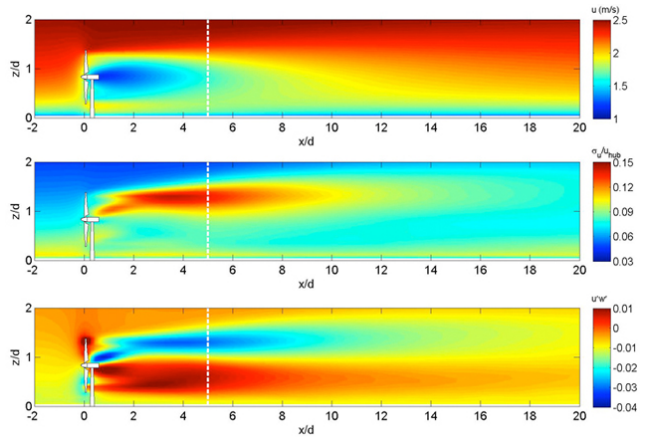


13 de enero de 2017

Presentación de Tesina

Simulation

- A simulation is a program that immitates the behavior of a system over time
- It implements the abstract model describing the system
- **Usssually, it requires a big ammount of computing power**



Optimizing Simulation

- Preserving accuracy:
  - paralellization
  - data locality
  - loop reordering
  - vectorization
  - well addressed by automatic approaches
- Not preserving accuracy
  - less precise models
  - less accurate computations
  - adaptive mesh refinement
  - not well addressed by automatic approaches

Our goal: to design a compiler approach to optimize simulation codes

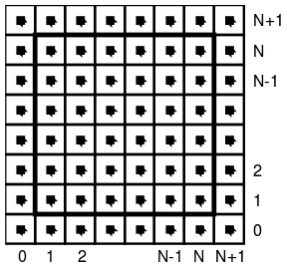
- by tuning accuracy
- through adaptive techniques

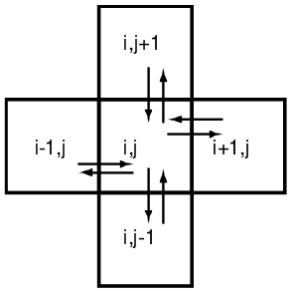
Outline

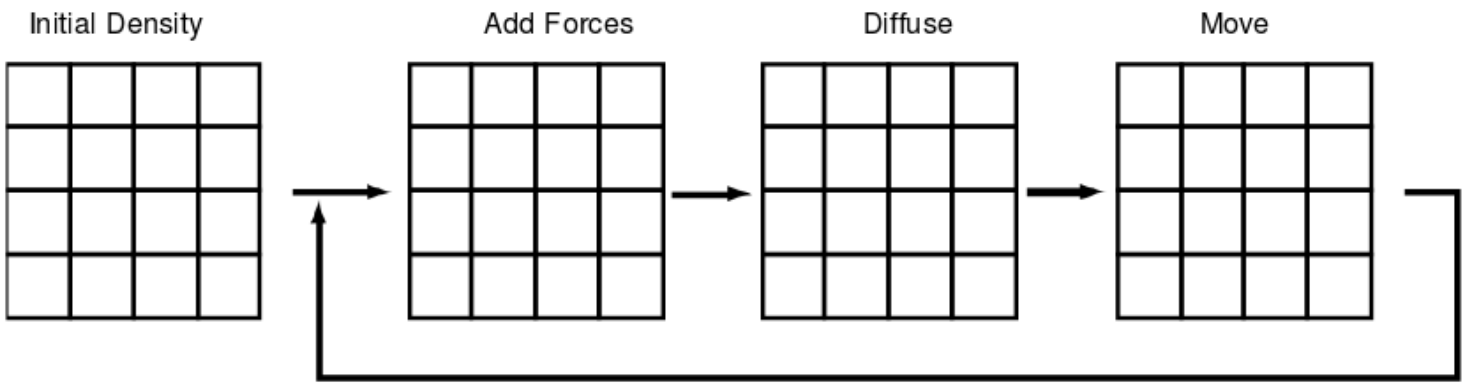
- Aimed Simulation: Eulerian Fluid Simulation
- Adaptive Techniques
- Static Tool: Spot
- Dynamic Tool: Adaptive Code Refinement
- Experiments

Eulerian Fluid Simulation

- Simulates the behavior of a fluid over time
- A rectangle is divided into cells, and every cell represent a particle
- At every time iteration, every cell actualizes its density value and its velocity vector







Adaptive Techniques

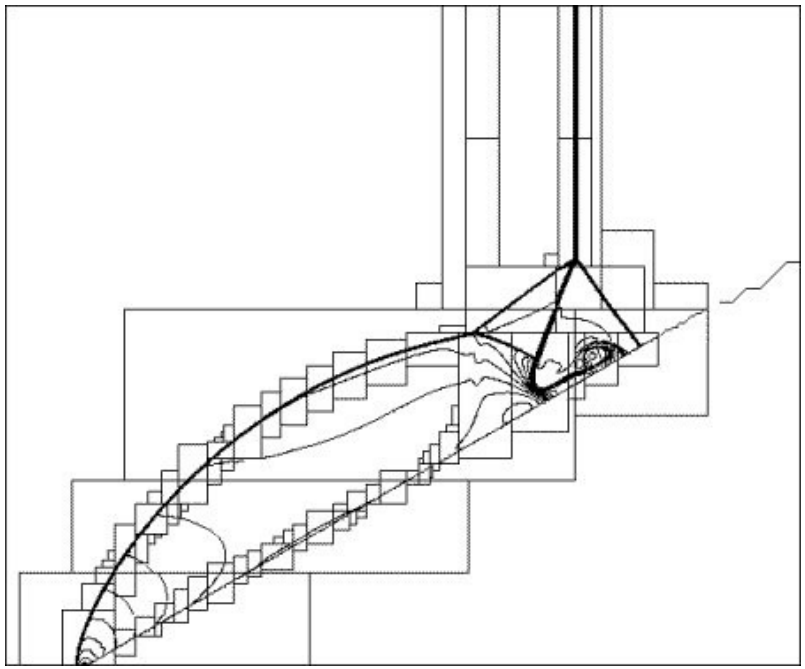
An adaptive technique only performs computations only where it is needed in the iteration space

- It changes the way it operates depending on the input
- It doesn't spends unnecesary computations
- It develops different regions of interest on computation
- It changes its behavior over time to fit the new states of the execution

Example: Adaptive Mesh Refinement

AMR is a Numerical Analysis technique for changing the accuracy of a solution in certain regions, while the solution is being calculated

- It computes hierarchical grid wich specifies the complexity of the computation
- It refines the precision of the calculation in intresting regions
- It performs basic calculations in regions where almost nothing or nothing hapens



First Approach: Static Accuracy Tuning

- we want to apply a filter the image
- the important region of the image is the flower
- the rest of the image can be processed by some simple calculations

SPOT

Simple Polyhedral loop Transformer

- source-to-source compilation tool for transforming loops
- allows us to input to a program information of different regions of a computation space
- manipulates the iteration space of a loop to change/eliminate computations in regions specified in pragmas

SPOT

Example: Image Processing

```
#pragma spot 1
   "[H, W]->{[x,y]: x > 3*H/4 or y > 3*W/4 or x < H/4 or y > W
   "simpleFilter(x, y, IMG);"
for (x = 0; x < H; x++)
   for (y = 0; y < W; y++)
      complexFilter(x, y, IMG);
```

SPOT

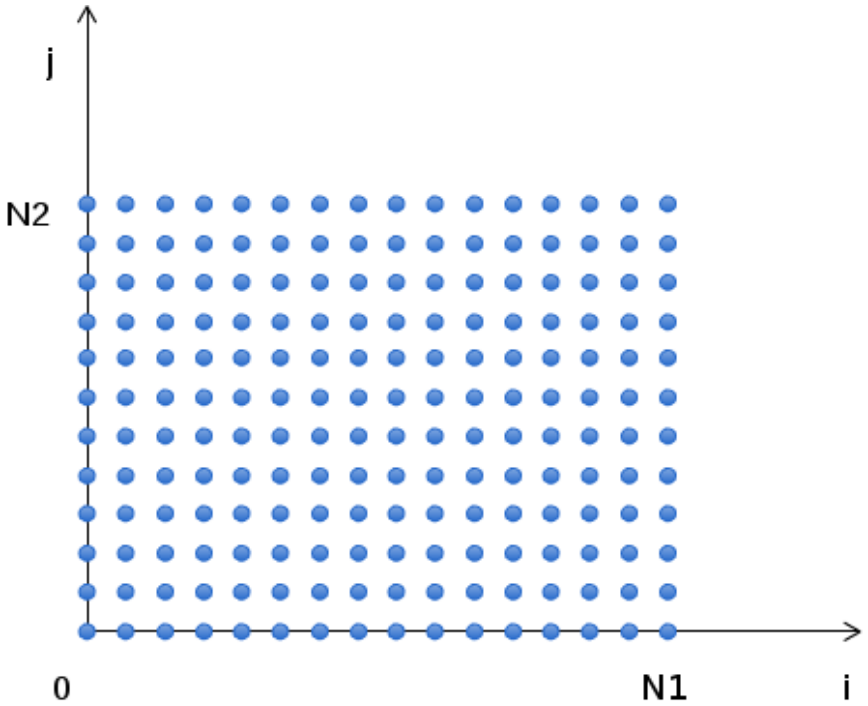
Simple Polyhedral loop Transformer

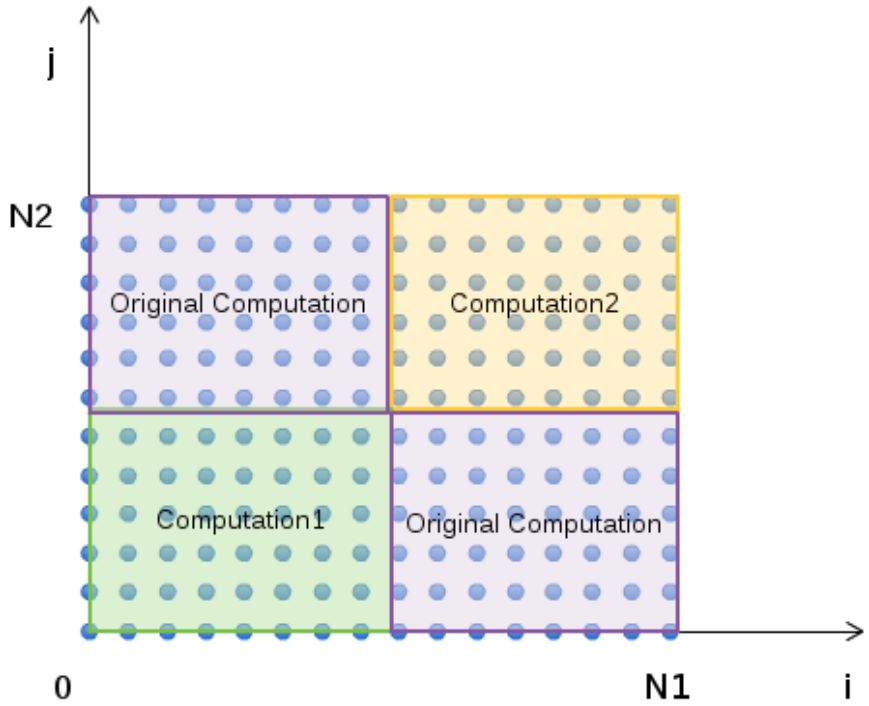
- uses ISL to specify input regions in pragmas as integer sets
- relies on polyhedral tools to analyze and process the sets
- it generates new code that performs the computations specified in the different regions

SPOT

Simple Polyhedral loop Transformer

```
#pragma spot 1
    "[N1, N2]->{[i,j]: i < N1/2 and j < N2/2}"
    "COMPUTATION1"
#pragma spot 2
"[N1, N2]->{[i,j]: i > N1/2 and j > N2/2}"
"COMPUTATION2"
for (i = 0; i < N1; i++)
   for (j = 0; j < N2; j++)
      ORIGINAL COMPUTATIONS;
```


j
N2
0
N1
i


j
N2
Original Computation
Computation2
Computation1
Original Computation
0
N1
i

Adaptive Code Refinement

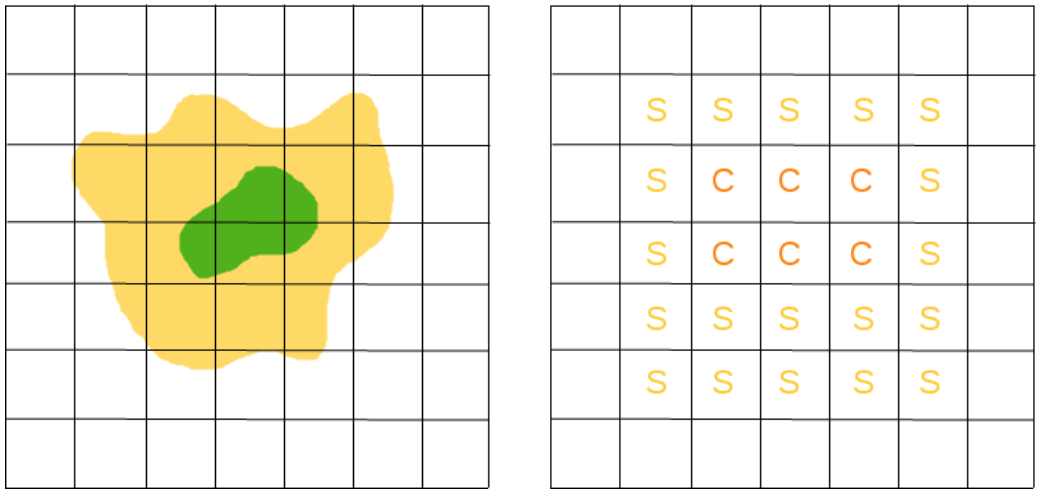
- provide adaptive capabilities to simulation codes
- uses domain-specific knowledge
  - save unnecesary computations
  - achieves "good enough results"
- regenerates the code in execution time
- uses SPOT to process iteration portions and to generate code

Adaptive Code Refinement: How it works

- uses a grid to gather useful information about the simulation state
  - complexity of computation needed in every region
  - grid processing is lightweight
- generates an optimized version of code according to the grid
- when the grid changes, the code is regenerated

ACR

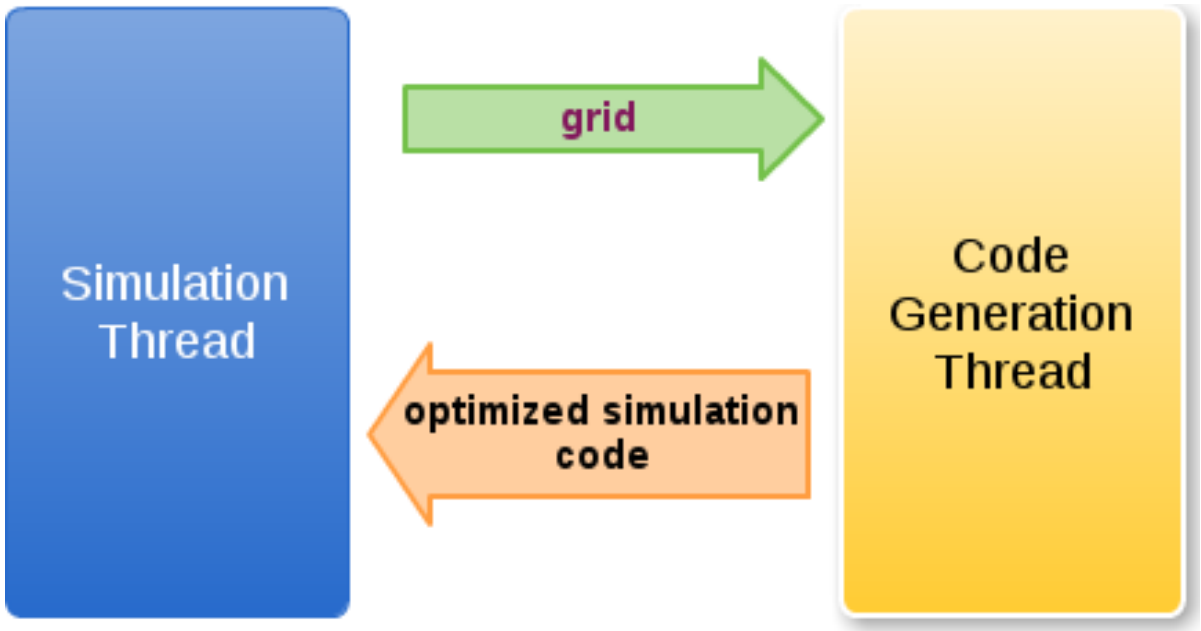
```
for (time = 0; time < T; time++)
 #pragma ACR gridsize=6
 {S1 -> Complex(i,j,t), S2 -> Simple(i,j,t),
   S3 -> AlmostNothing(i,j,t)}
 for (i = 0; i < N; i++)
   for (j = 0; j < N; j++) {
     Original(i,j,t);
   }
```



ACR

Threads

- Simulation and Code generation are done in two parallel threads
- One thread executes the simulation code available and refreshes the grid
- The other thread recompiles the simulation code when the grid changes



ACR

Ensuring "Safety"

- there is a gap between grid change and new code available
- if the available optimized code doesn't fit the current grid the simulation thread executes the original code
- to avoid to many switches to the original code, the surroundings of intrest regions are also covered by the grid

Experiments

2D Eulerian Fluid Simulation

- Simulates the behavior of a fluid over time
- A rectangle is divided into cells, and every cell represent a particle
- At every time iteration, every cell actualizes its density value and its velocity vector

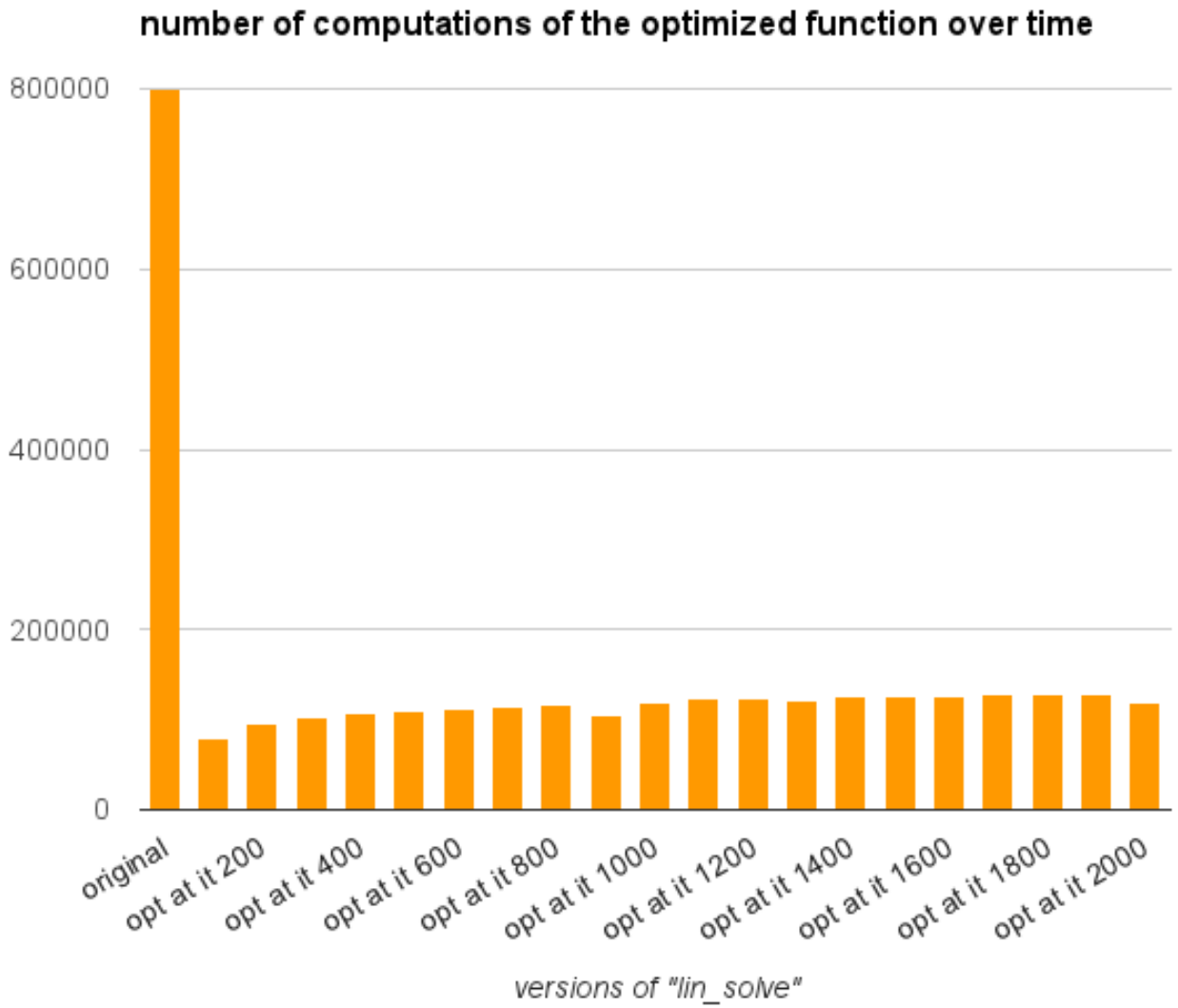
Experiments

Applying ACR

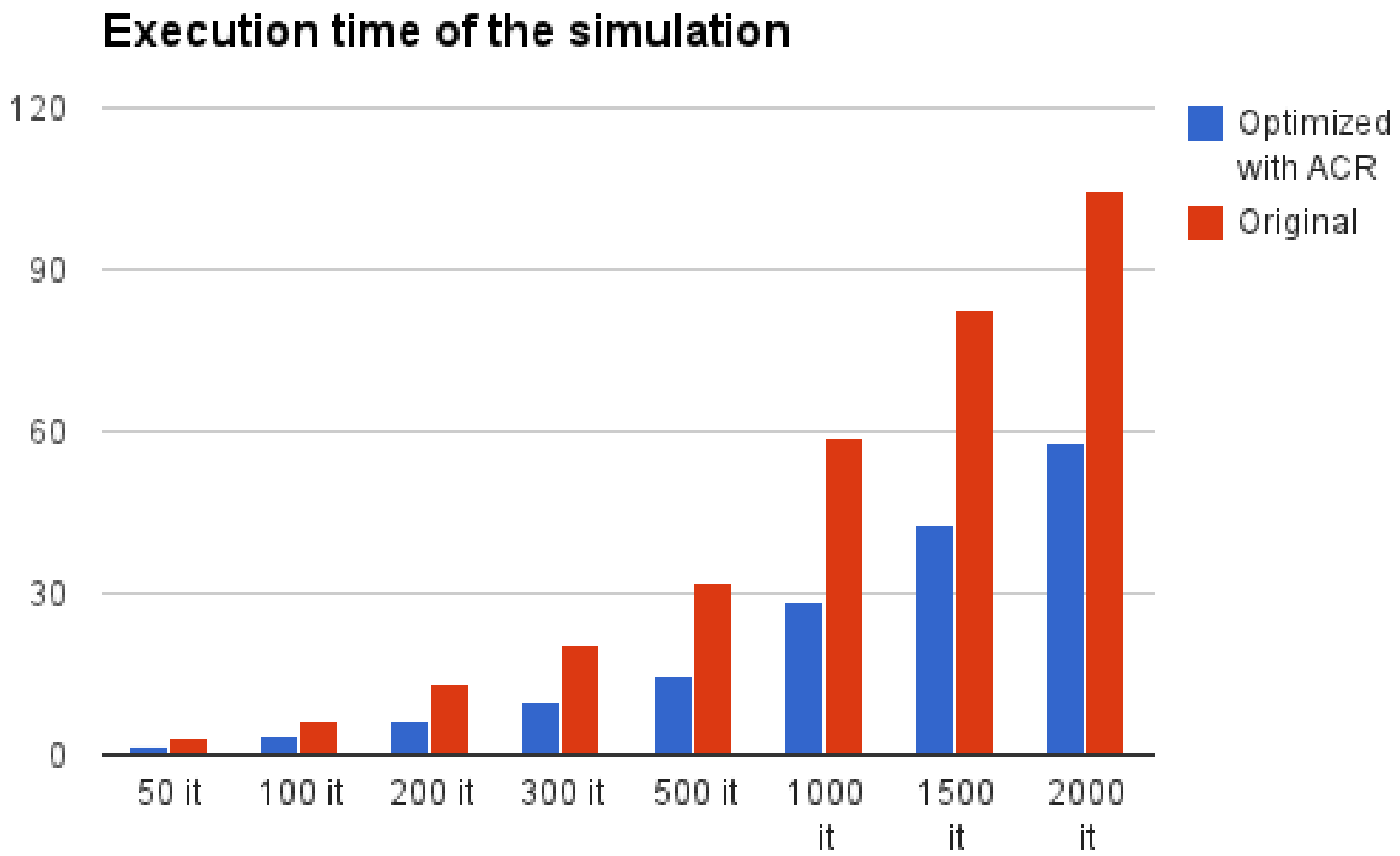
- The complexity of the computation in a region depends on the density of the fluid
- As the simulation evolves, the grid will adapt to fit complex calculations where the fluid is
- More simpler calculations will be done where there is a little amount of fluid, and almost any where there is not fluid

Experiments

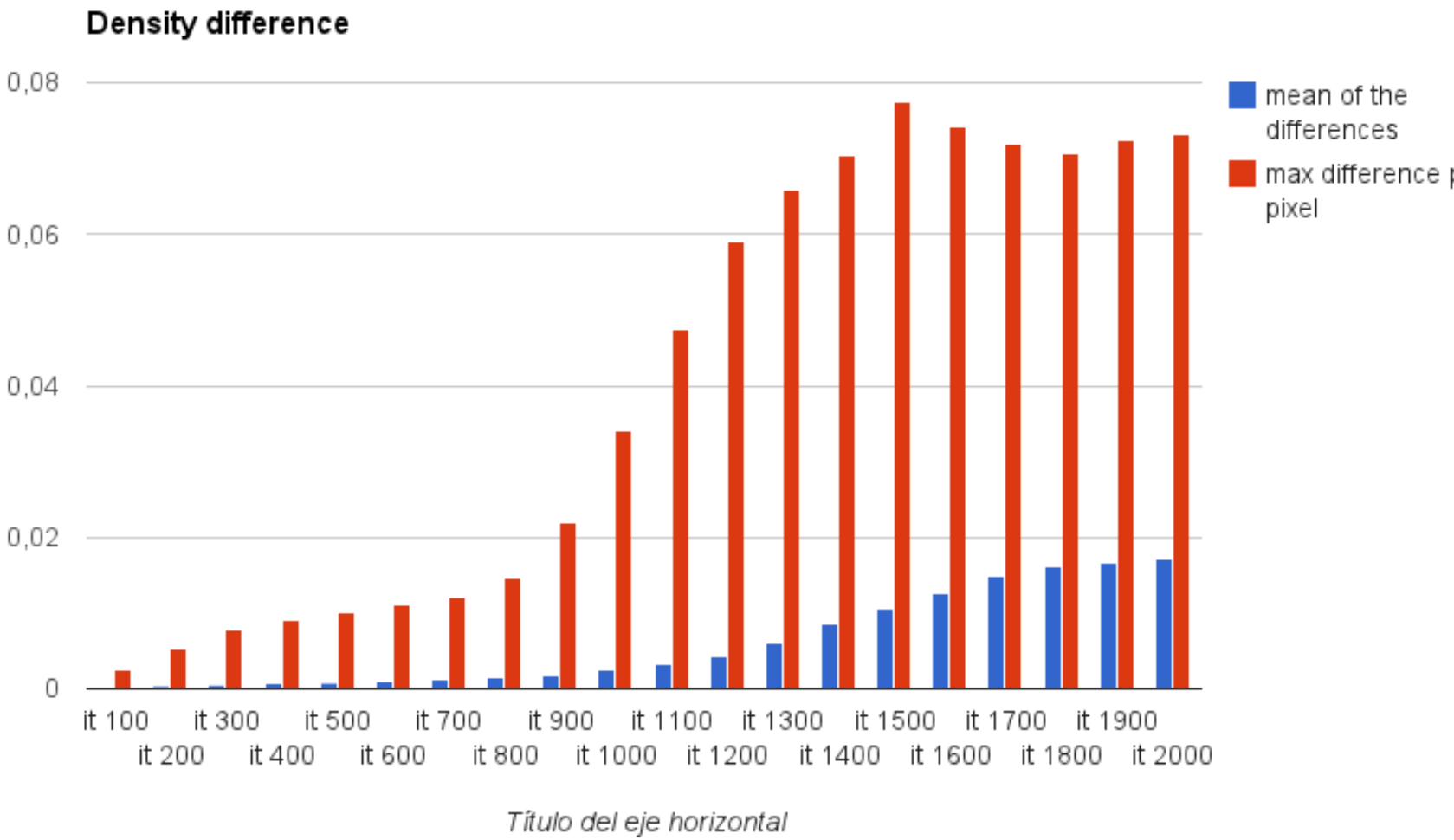
Results: Saved Computations



Results: Performance over time



Results: Accuracy



Questions?