Design of an autotuning plug-in for Periscope:
Initial design and manual optimization baseline

Antonio Pimenta

July, 2012

Advisors

Anna Sikora Eduardo César

Contents

- Introduction
- Periscope
  - Current capabilities
  - Plug-in
- Test applications
- Manual optimizations
- Experimental results
- Conclusions

Introduction

AUTOTUNE
www.autotune-project.eu

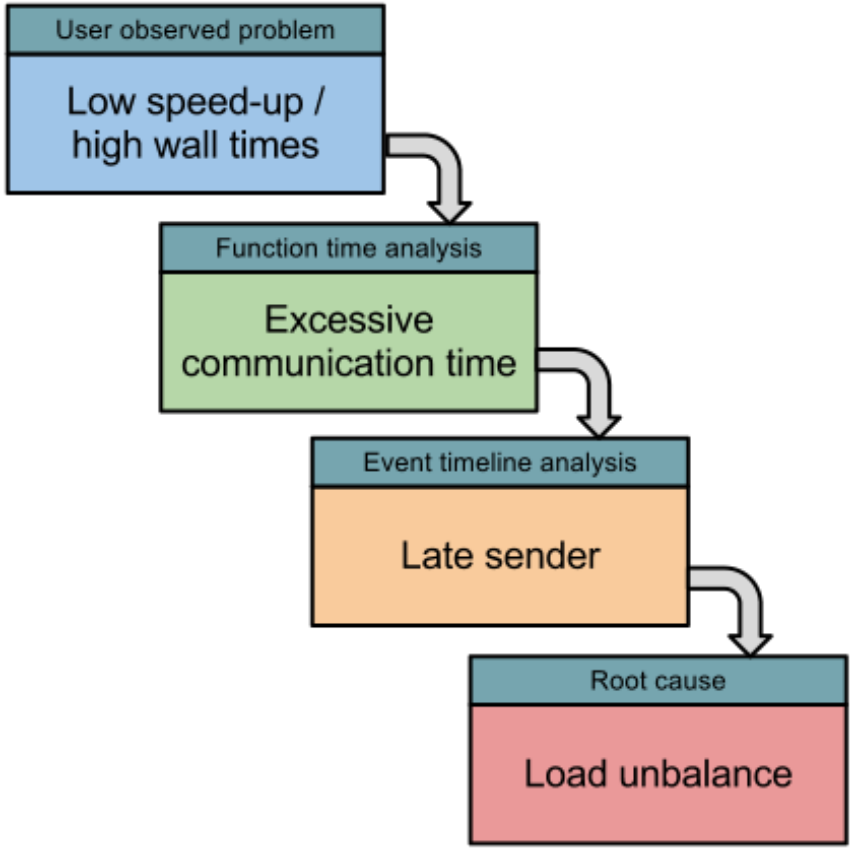
- Autotune project (http://www.autotune-project.eu/)
  - Partners: TUM, Universität Wien, ICHEC, CAPS, LRZ
  - Funding: European Union Seventh Framework Programme
  - Period: 2011-2014
- Main objectives
  - Extend Periscope to integrate plug-ins that add functionality to it
  - Create several plug-ins to tackle different performance problems (GPU pipelining, CPU efficiency, MPI automatic optimization)
  - Disseminate the results and make Periscope known to the scientific community

Introduction

- Scalasca
- TAU
- Paradyn
- MATE
- Periscope
- …

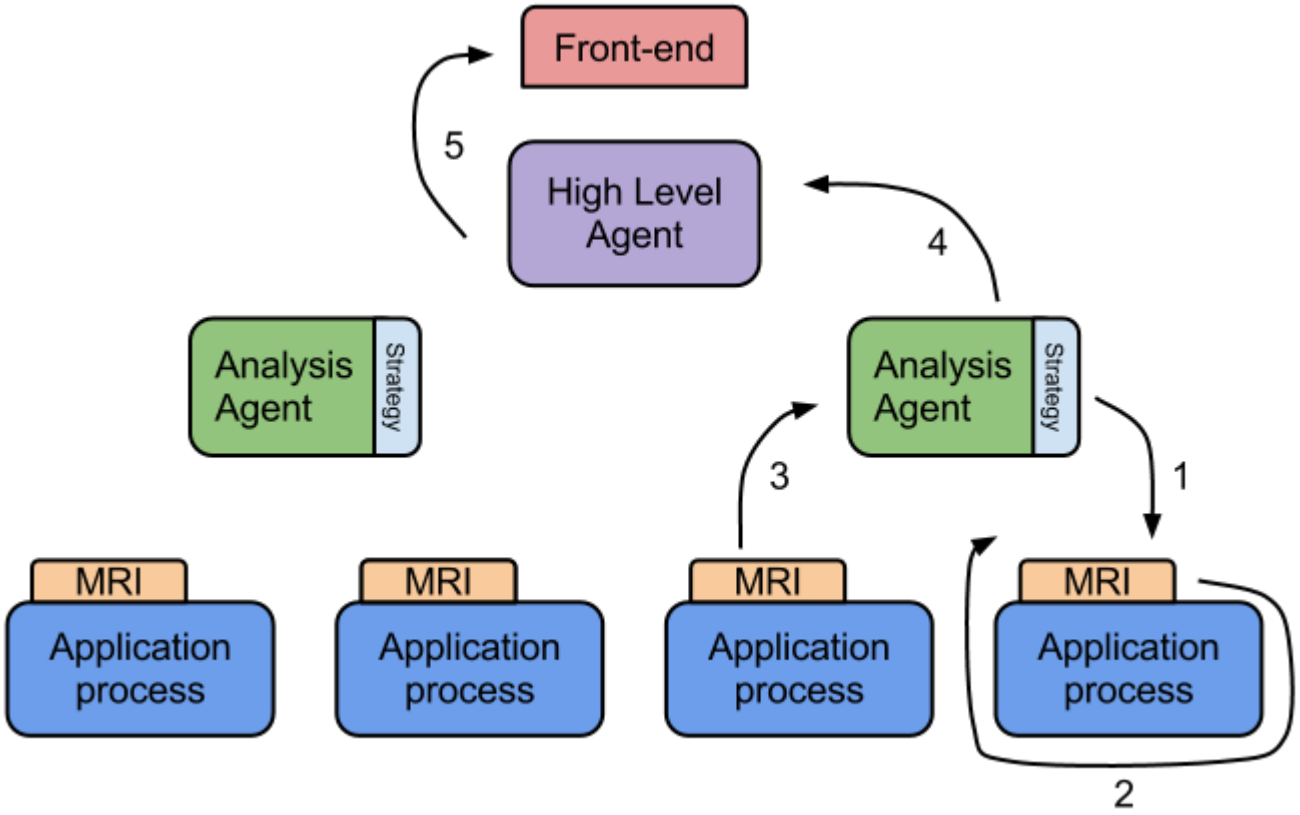
Periscope: current capabilities

- Property detection
  - Properties can be added by the user
- Online analysis
  - No trace files created in this mode
- Comprehensive output
  - The properties represent performance problems the user can understand



Periscope: current capabilities

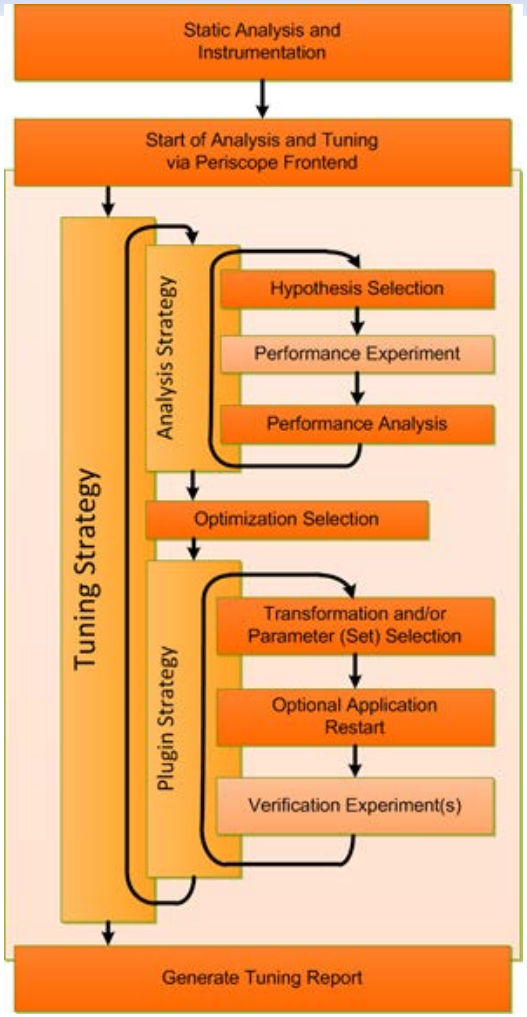
AUTOTUNE
www.autotune-project.eu



Periscope: plug-in

AUTOTUNE
www.autotune-project.eu

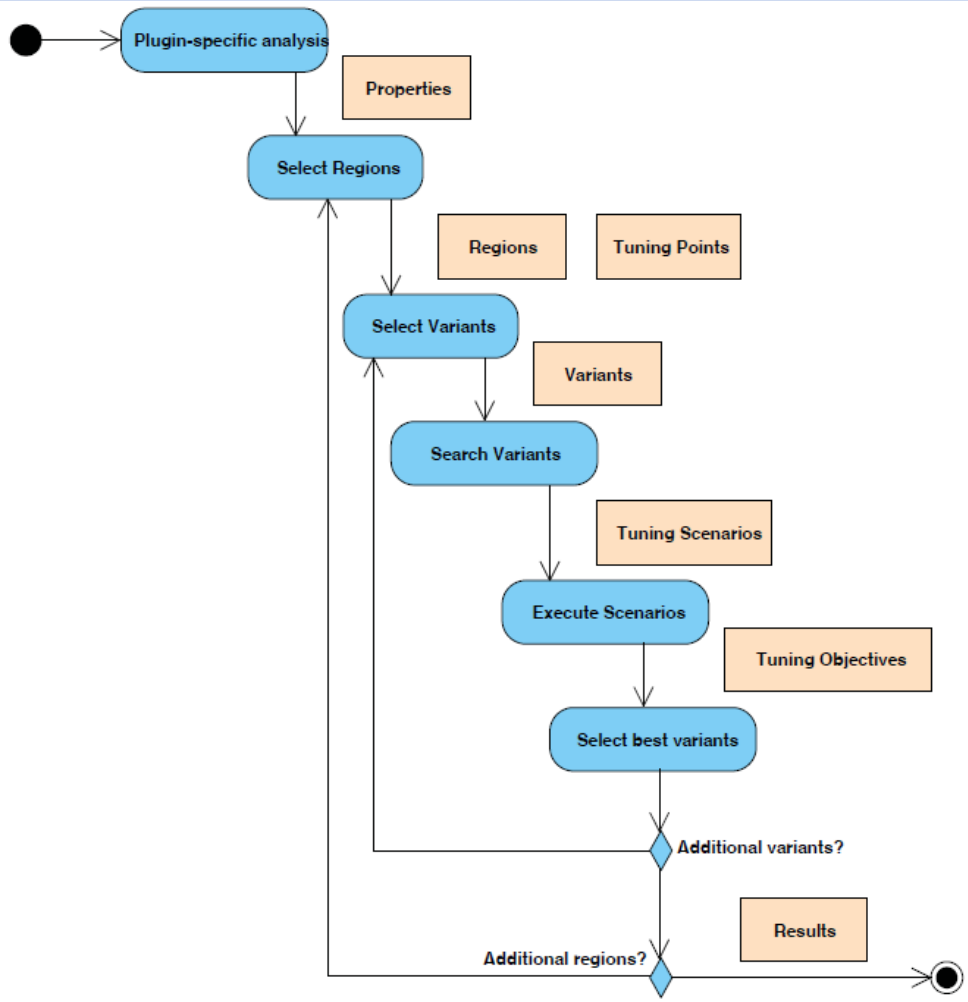
- Two main tuning approaches
  - General MPI tuning
    - Optimizing the MPI library usage
    - Configurations done by changing environment variables
  - Custom model optimizations
    - Master-Worker
    - SPMD
    - Pipeline



Periscope: plug-in

AutoTune
www.autotune-project.eu

- Basic plug-in structure
  - Analyzer module at the level of the Periscope front end
  - Tuning integrated into the agent hierarchy and MRI monitor
- Basic strategy (Master-Worker)
  - Standard MPI analysis
  - Parallel regions
  - Partition factor and number of workers as variants
  - Reduce wall time



Test applications

- Necessary to evaluate automatic tuning
- WRF
  - Numerical weather prediction
  - Developed by a group of US institutions
  - MPI + OpenMP parallelism, SPMD
- S2F2M
  - Statistical fire propagation prediction
  - MPI, Master-Worker
- FSSIM
  - Biological behavior simulator
  - MPI, SPMD

Test applications

- Fish School simulator
  - Initial distribution of load
  - Iterative time step simulation
    - Gather necessary data in each node
    - Compute the new position and direction of each individual
    - Perform necessary migrations
- MPI communications
  - Initial distribution
  - Neighbor fish exchange and interprocess migrations

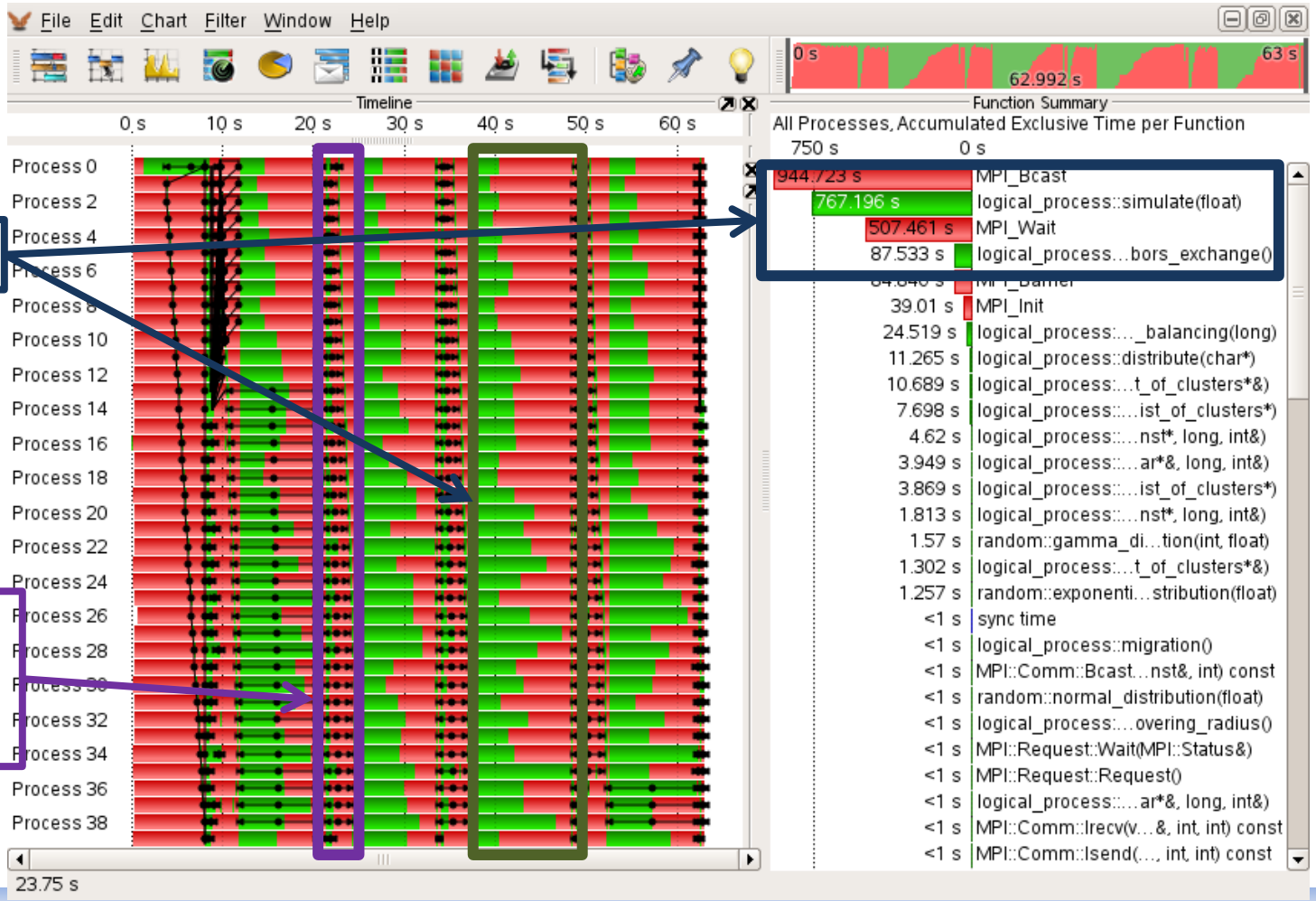
Test applications: First analysis

AUTOTUNE

www.autotune-project.eu

Load unbalance

Exesive communication time



Manual optimizations

- FSSIM
  - Optimize the inter process communication phases
    - Neighbor exchange
    - Migrations
  - Balance the load
    - Find proper balancing criteria
    - Develop balancing function
    - Determine optimal values for the balancing process

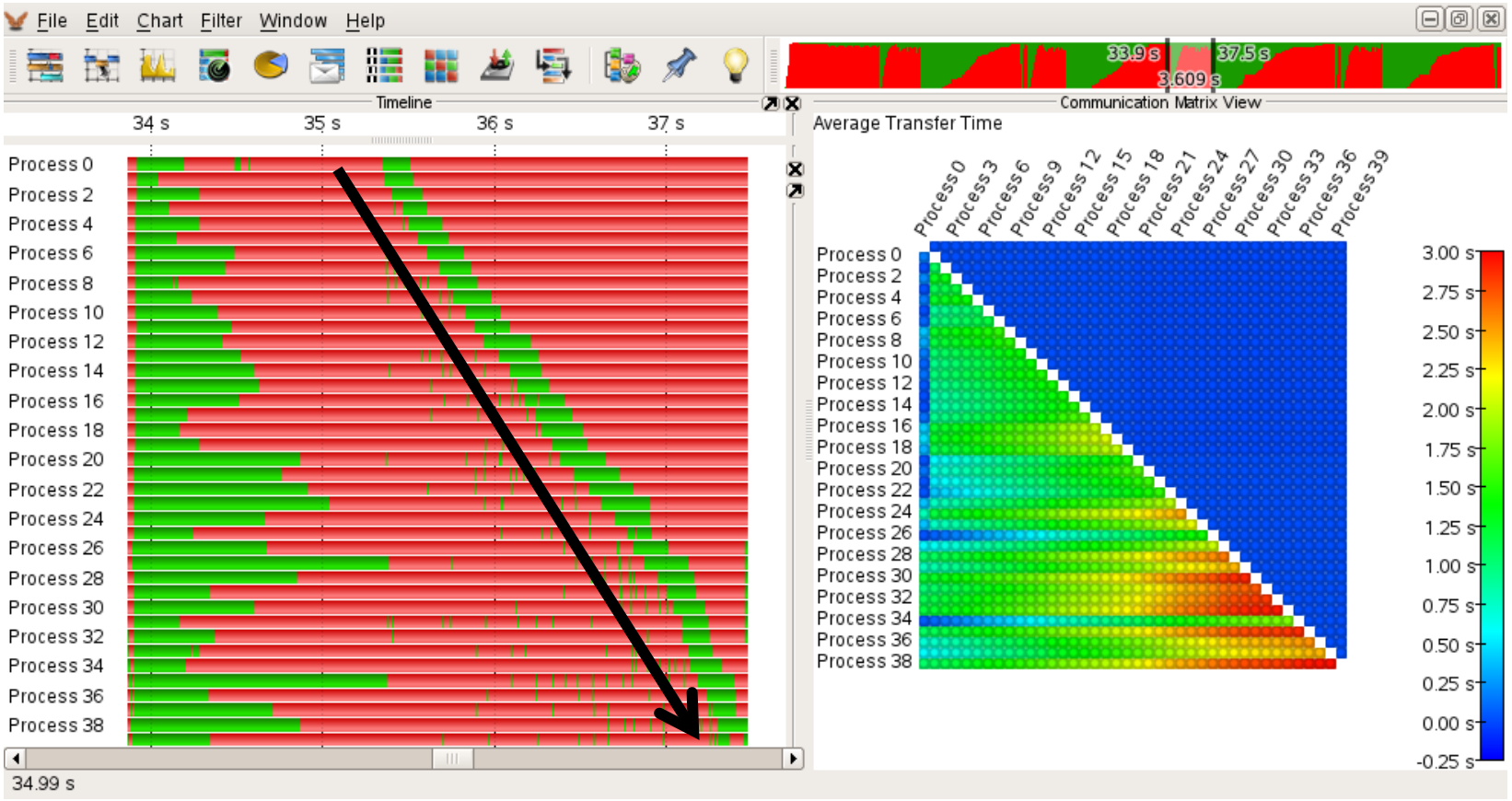
Experimental results

- Experiments performed on:
  - SuperMIG computer
  - LRZ, Garching (Germany)
  - Fat nodes
    - Westmere-EX Intel Xeon E7-4870 10 core
    - 4 processor per node
    - 256 GB memory per node
  - Infiniband QDR interconnect
- Experiments ranging from 40 to 160 cores.

Manual optimizations: MPI restructuring

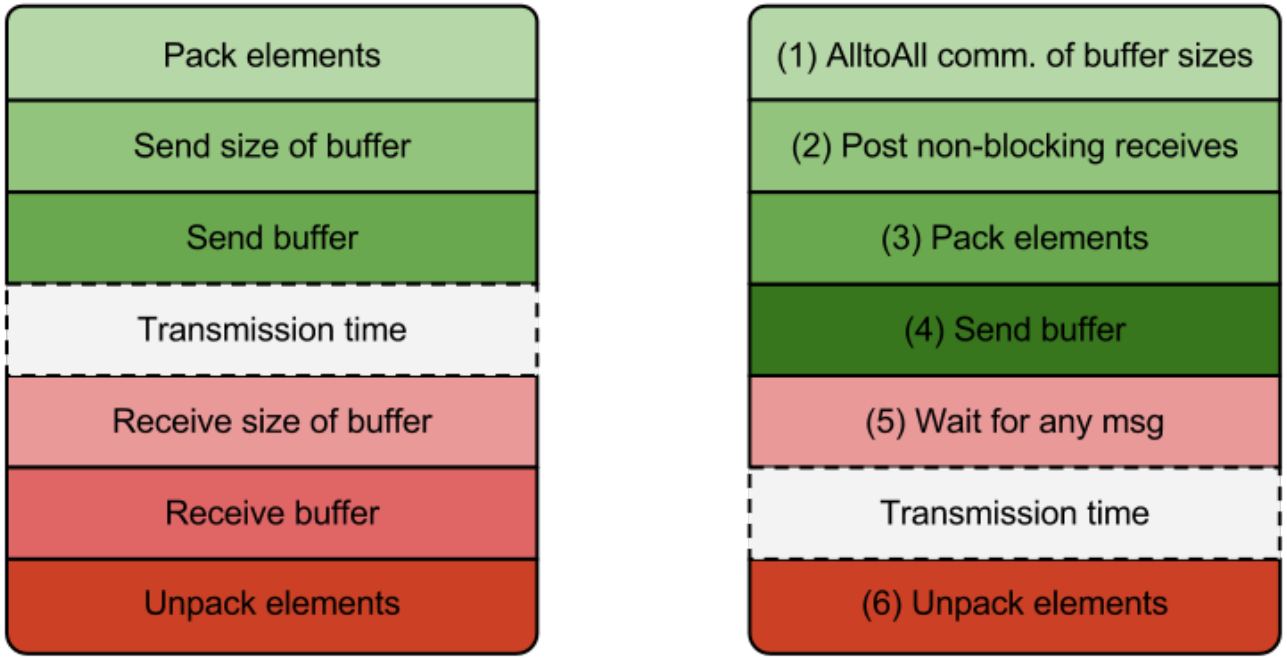
AUTOTUNE

www.autotune-project.eu



Manual optimizations: MPI restructuring

- Proposed solution
  - Use of non blocking MPI directives
  - Restructure the communication pattern



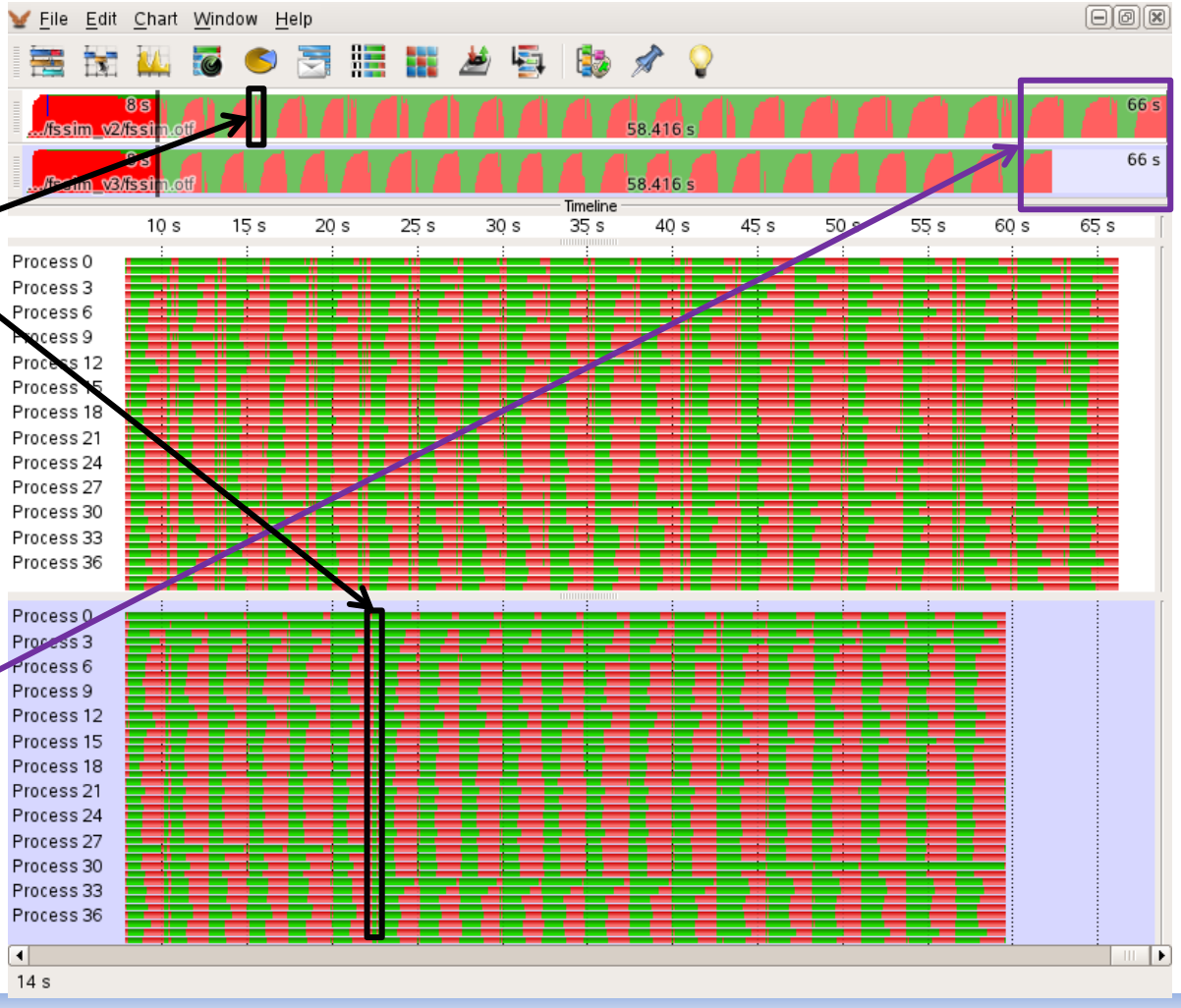
Experimental results: MPI restructuring

AUTOTUNE

www.autotune-project.eu

Reduced tiome for inter process communications

10% total time reduction

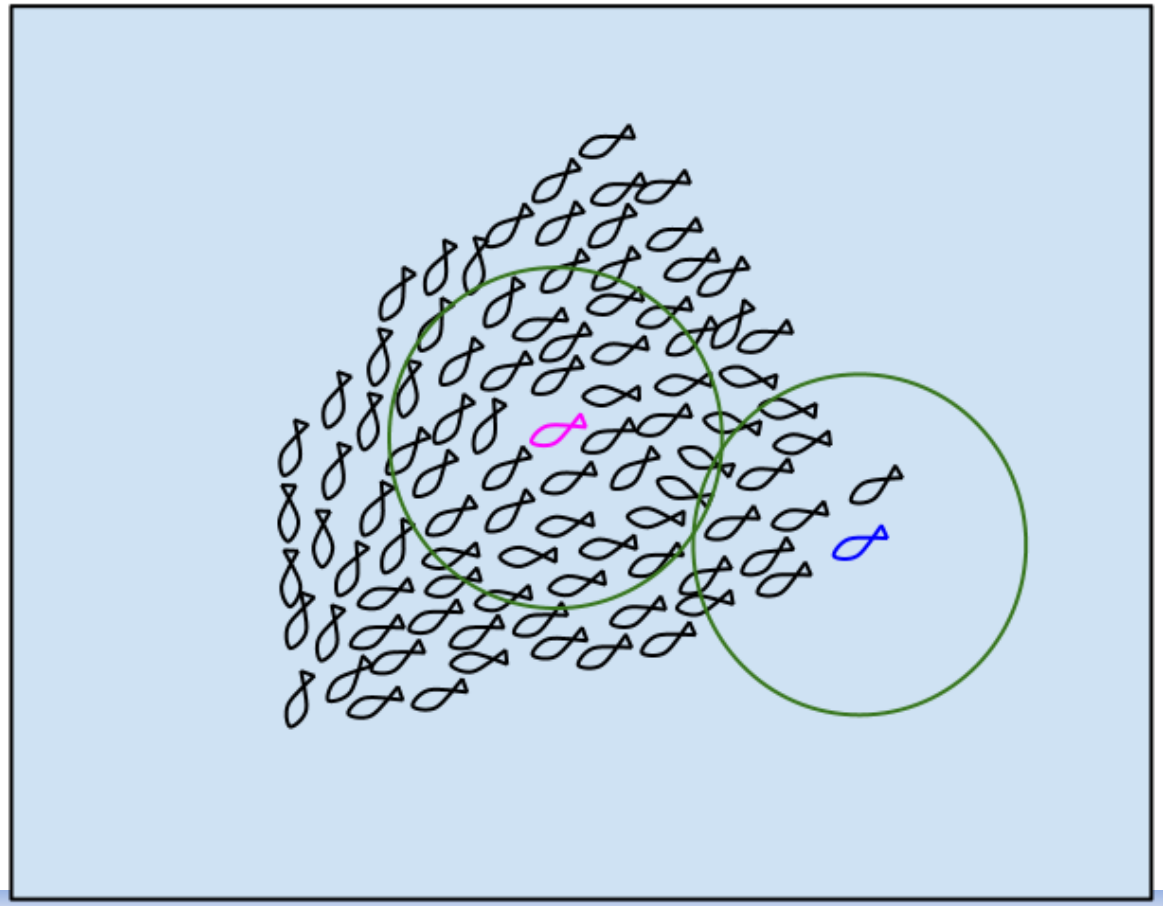


Manual optimizations: Load balancing

- Notable unbalance between processes
  - Idle times in underloaded nodes
  - Overall performance degradation
- The unbalance happens in function update_buckets
  - Depends on the amount of fish to process, but not excusively
  - Some fish require more computing time, due to their neghbors

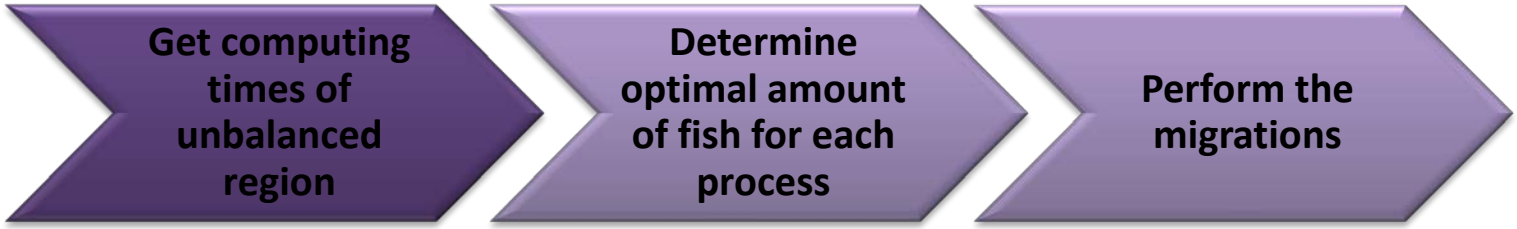
Manual optimizations: Load balancing

- Heavy vs light fish



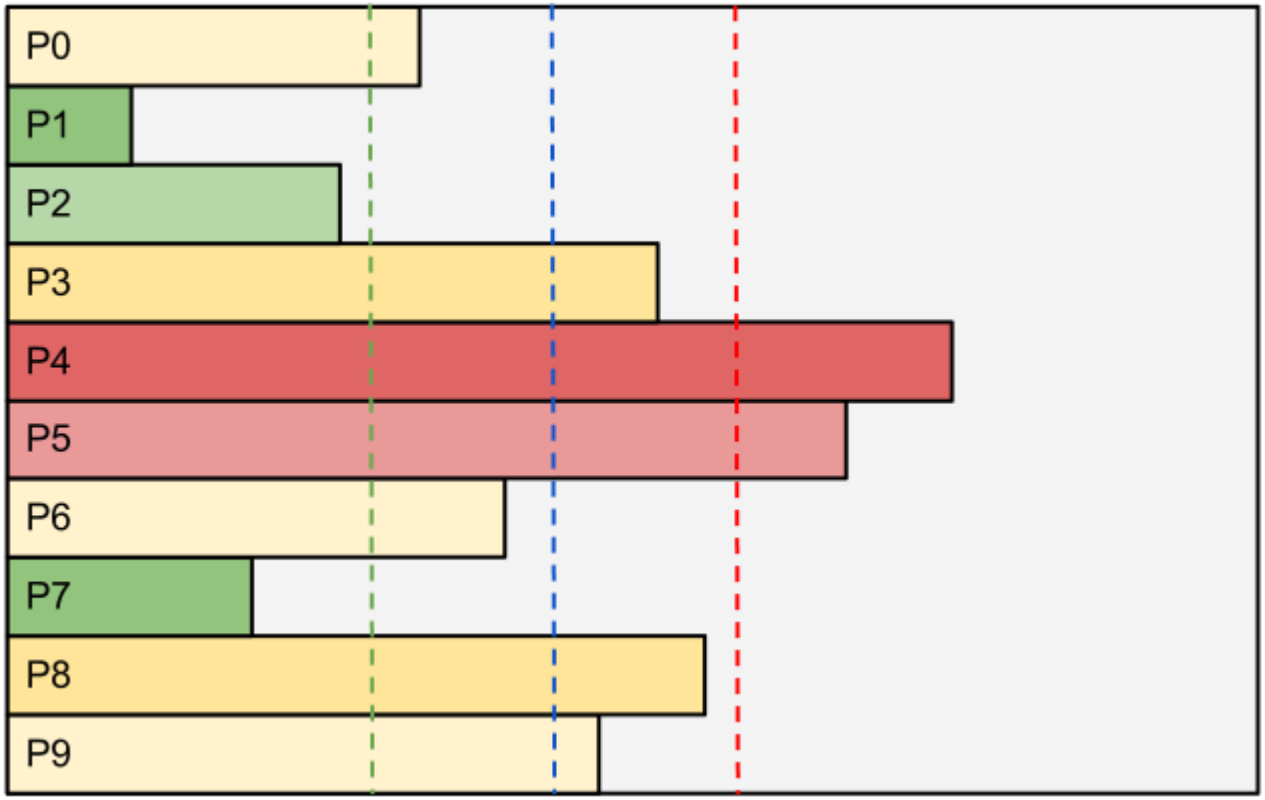
Manual optimizations: Load balancing

- Proposed solution
  - Change the existing balancing function
  - Computing time of previous timestep as criteria
  - Tolerance of unbalance, to only apply the balancing method when strictly necessary



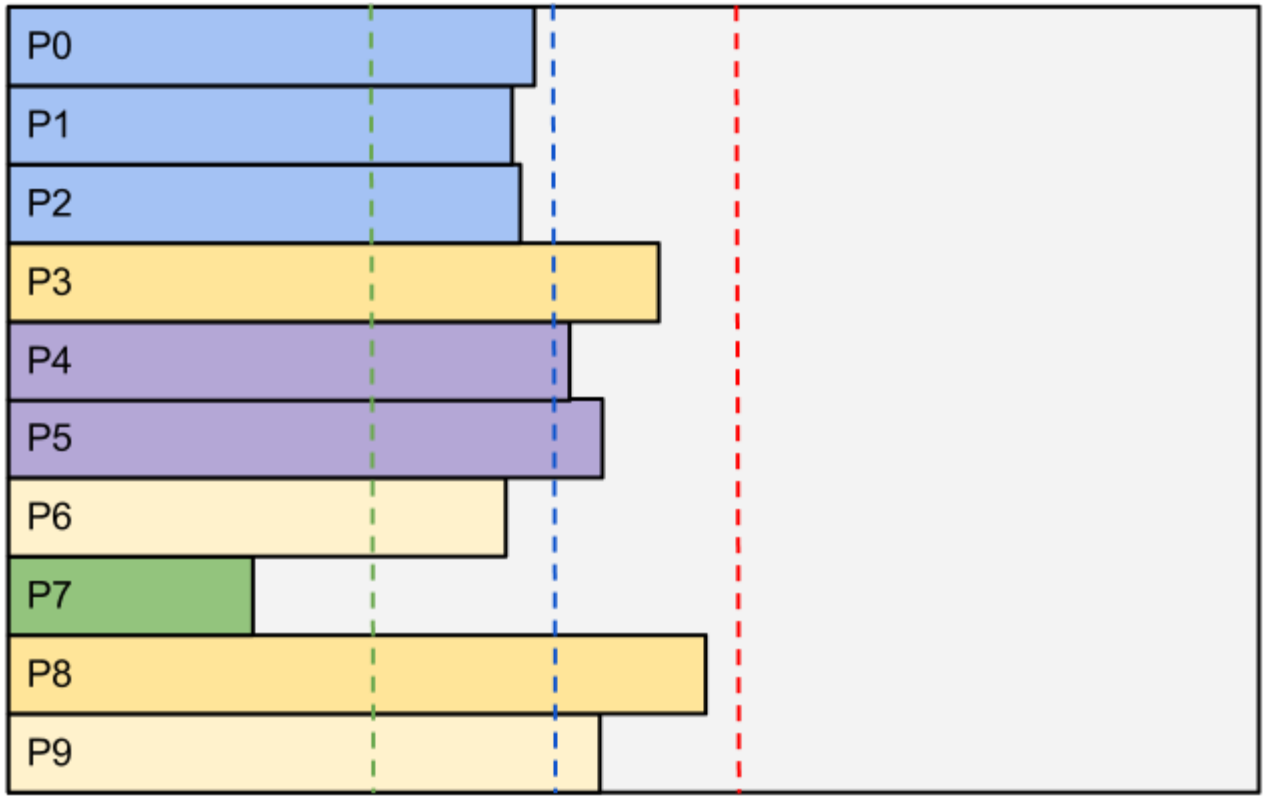
Manual optimizations: Load balancing

- Unbalanced state



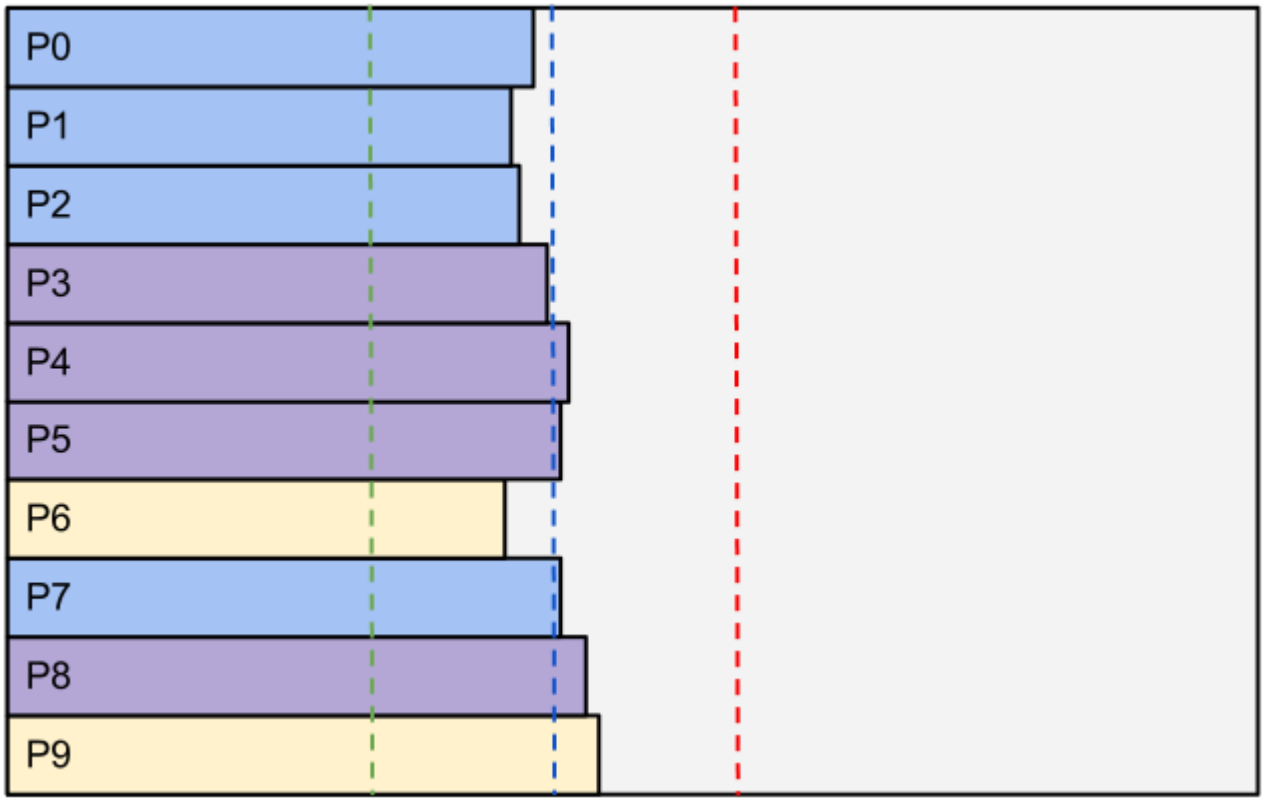
Manual optimizations: Load balancing

- Balancing phase 1



Manual optimizations: Load balancing

- Balancing phase 2



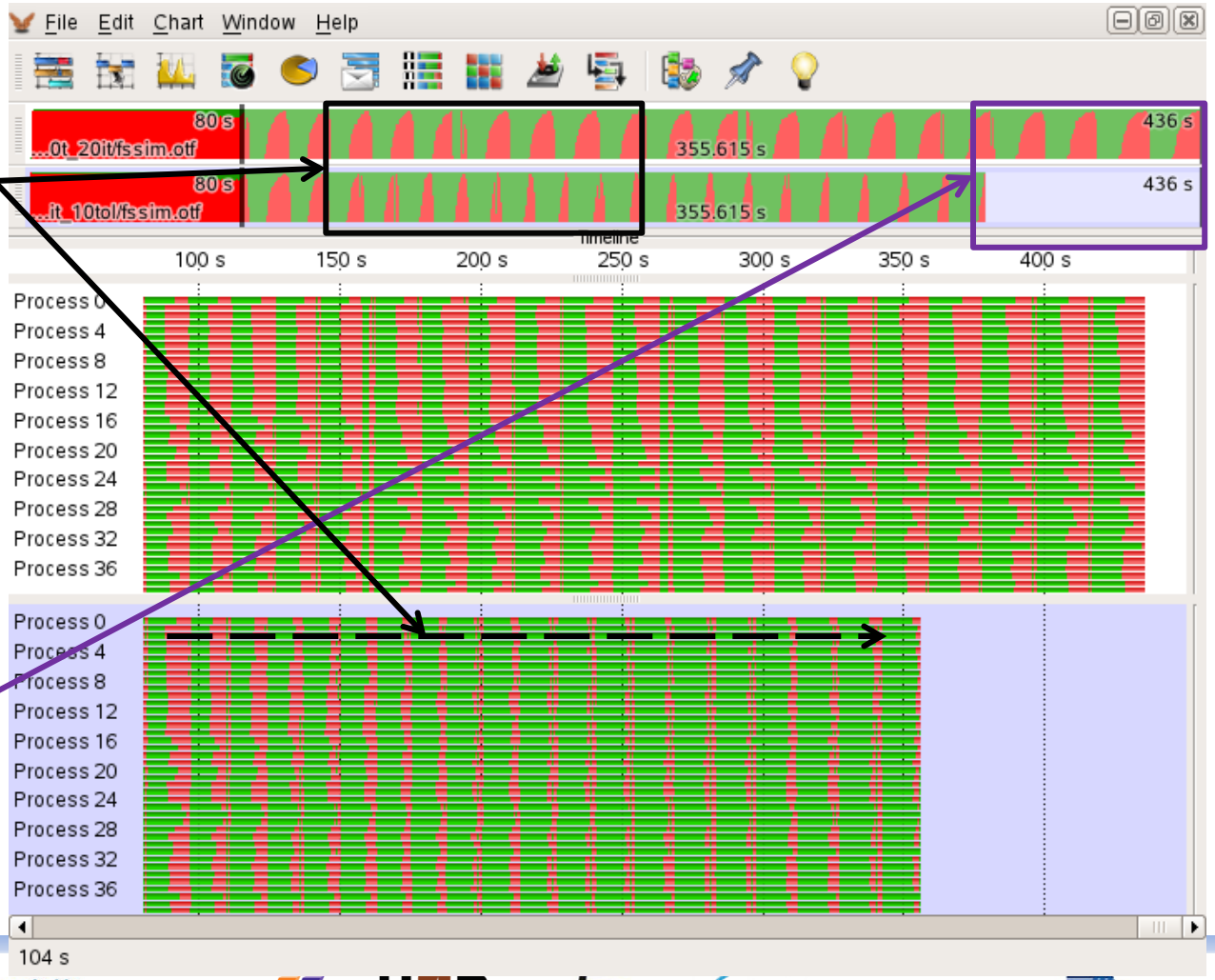
Experimental results: Load balancing

AUTOTUNE

www.autotune-project.eu

Gradual increase in load balance

From 436s to 356s, 22% total time reduction



Experimental results: Combined optimizations

AUTOTUNE
www.autotune-project.eu

- Appling both optimizations together

(40 threads, 512k fish, 20 iterations)

- Large simulation results:

(160 threads, 100 iterations)

Workload	Original time	Opt. time	Speed-up
131072	248.7	117.3	2.12
262144	594.7	289.9	2.05
524288	1741.9	917.9	1.89

Conclusions and future work

AUTOTUNE
www.autotune-project.eu

- We've designed the basic structure of the tuning plug-in
- We've determined the scope of the tuning
- We've chosen test applications appropriate to our tuning expectations
- We've manually optimized one of the applications obtaining a baseline for the automatic tuning

Conclusions and future work

- Future work within the project
  - Test the generic MPI tuning configurations
  - Finish the design of the plug-in
  - Develop the plug-in software
  - Integrate the plug-in with Periscope
  - Dissemination of the results

Thank you for your attention

Any questions?