Performance optimization of EUROFusion HPC code ERO2.0

Joan Vinyals-Ylla-Català
Marta Garcia-Gasulla

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 and 2019-2020 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Context

- EuroFusion Advanced Computing Hub at BSC
  - Provides computer science, performance, optimization and software engineering support to EuroFusion program
  - 3 BSC teams and departments involved
  - STELLA, SPICE, KNOSOS, and **ERO2** among others
- In collaboration with ERO2.0 developers from JSC, at TSVV-7

ERO2

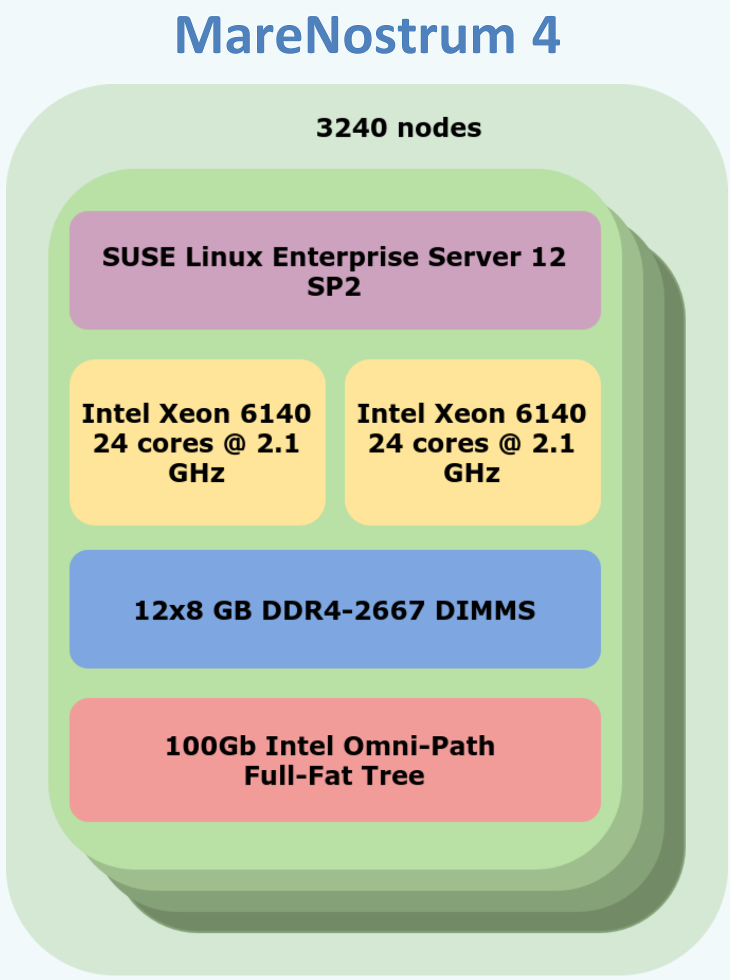
- Monte Carlo 3D code for global erosion and deposition modelling
- Written in C++ and parallelized using MPI and OpenMP
- Set up
  - Input files from:
    - https://jugit.fz-juelich.de/ero/runs/jromazanov/jet/run03/seq01
  - JET simulation
  - Random Seed set to false
  - 500k Particles
  - 250 maxMpiChunkSize
  - 0.1 maxTracingTime

- More information on the simulation setup:
  - Juri Romazanov <j.romazanov@fz-juelich.de>

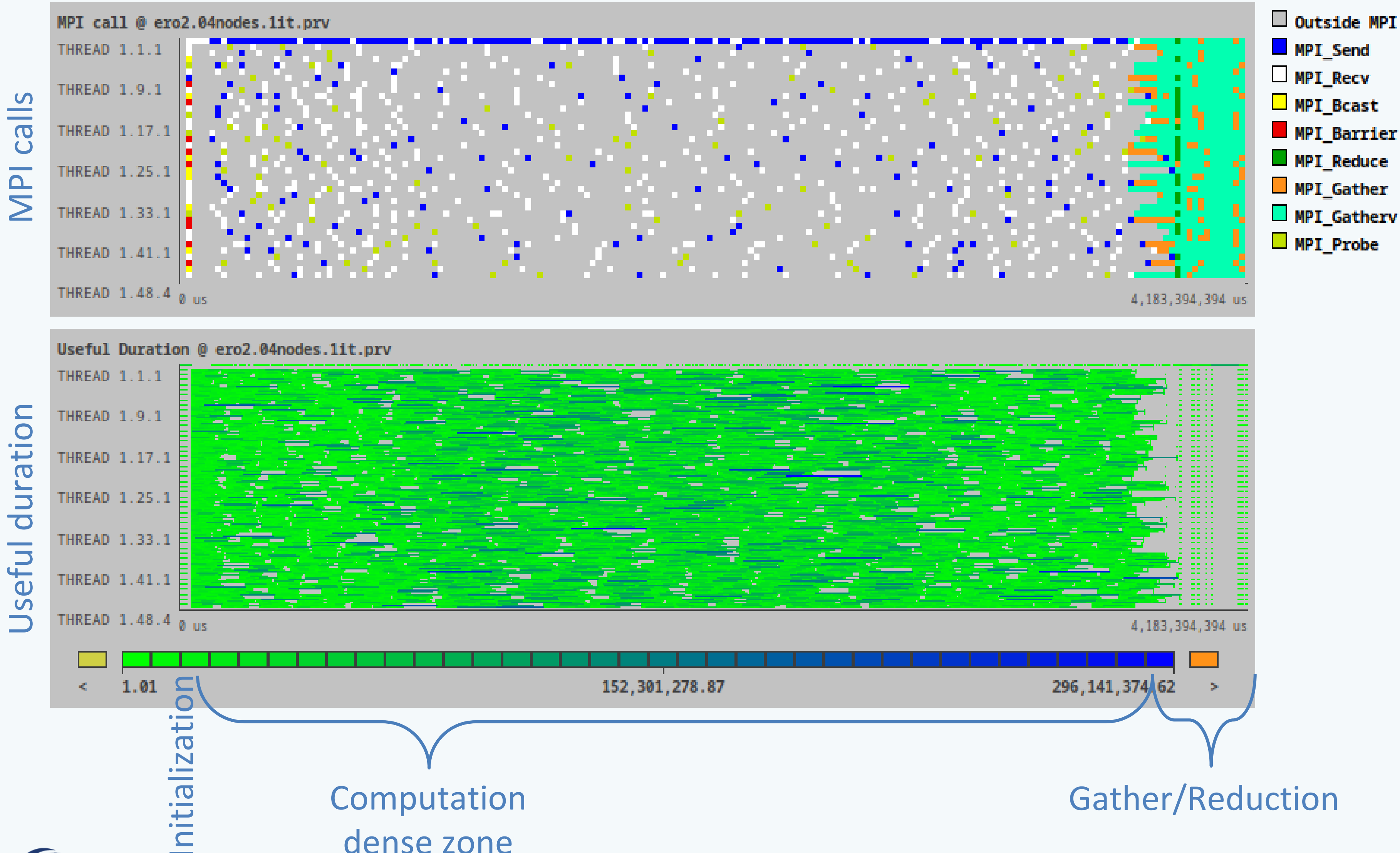
Environment

- Intel Compiler 2017.4
- MPI: IMPI 2017.4
- Libraries:
  - MKL 2017.4
  - HDF5 1.8.19
  - BOOST 175.0Z

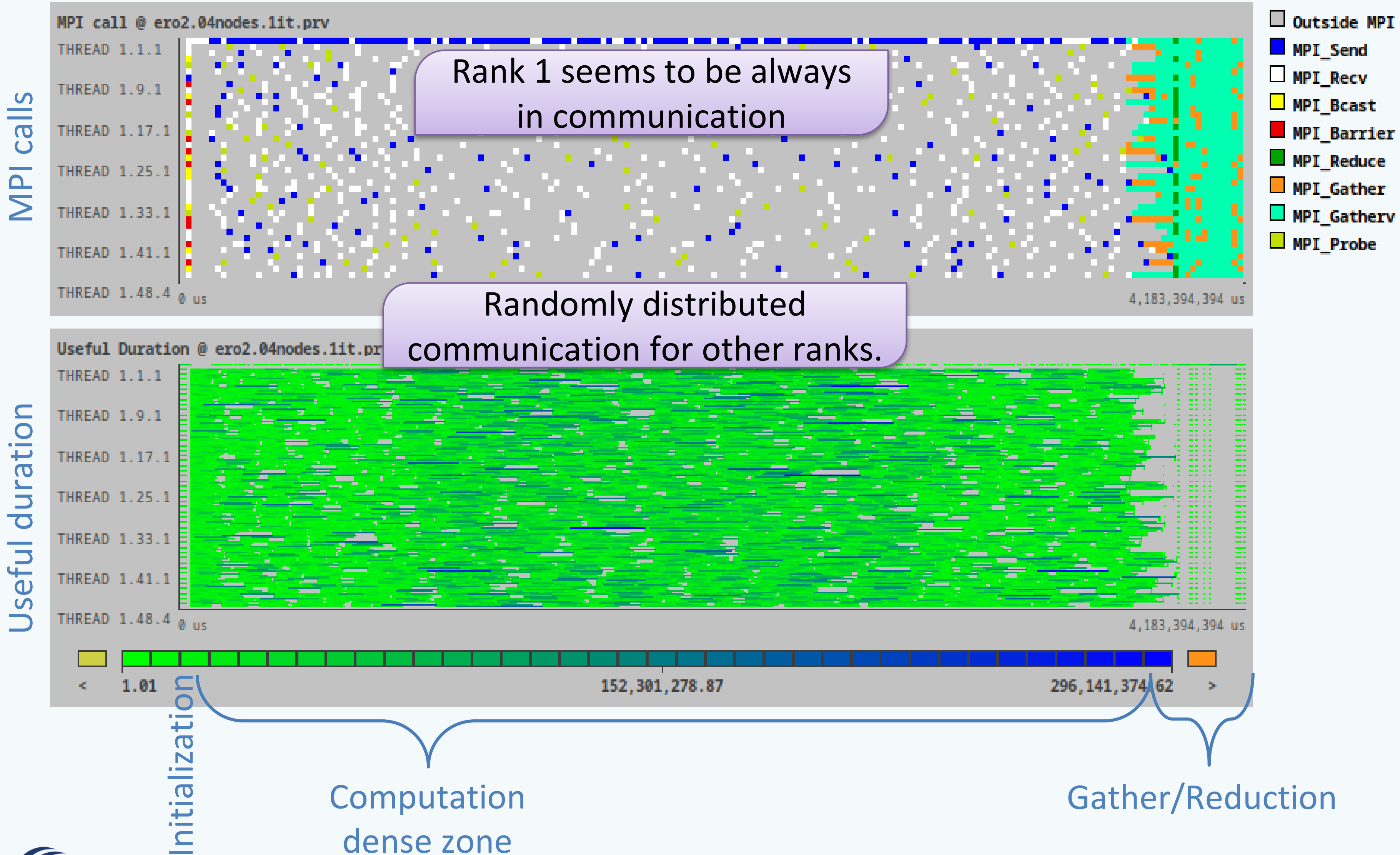
- For performance analysis:
  - Extrae to get traces
  - Paraver to visualize traces



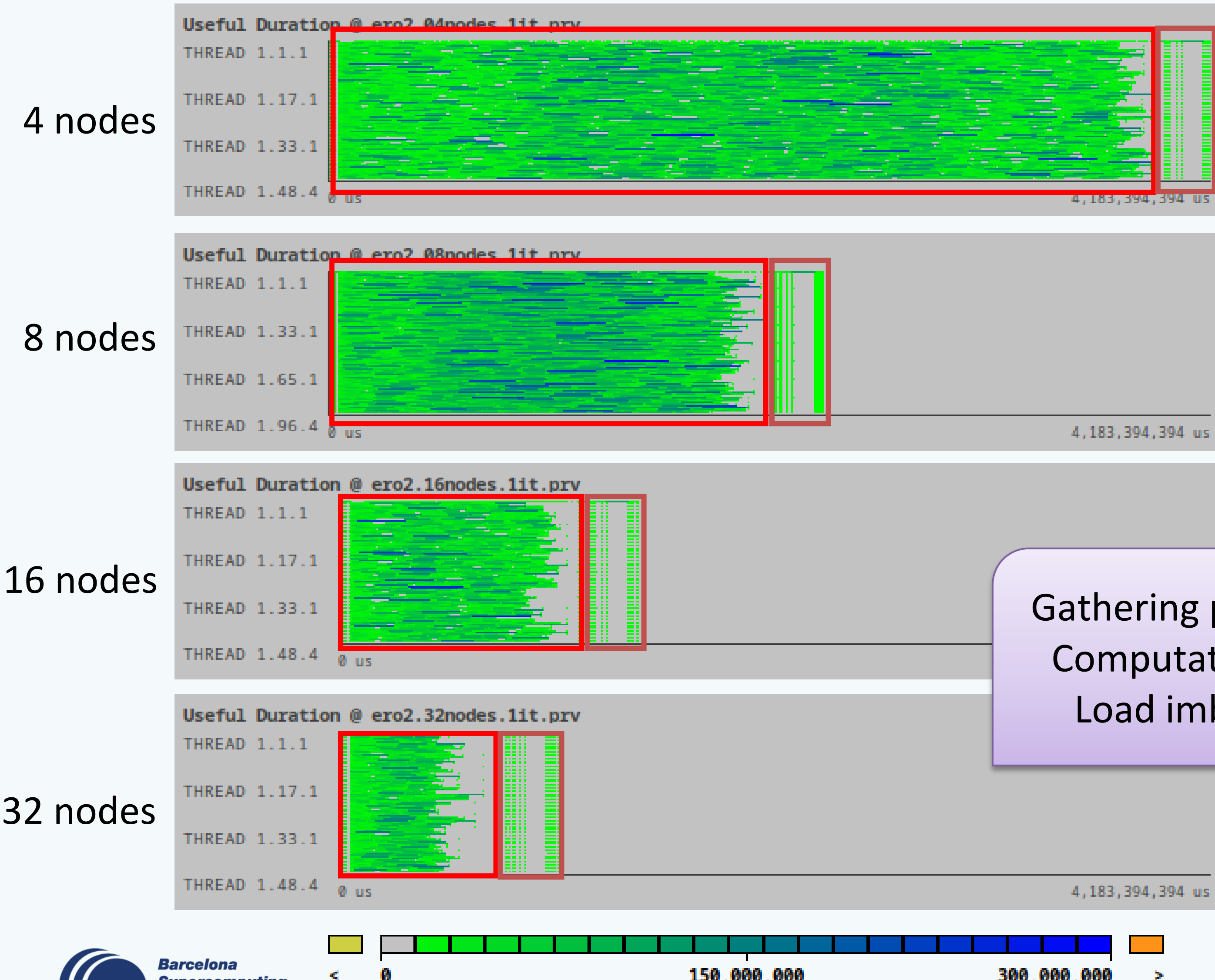
Structure - 1 step execution



Structure - 1 step execution

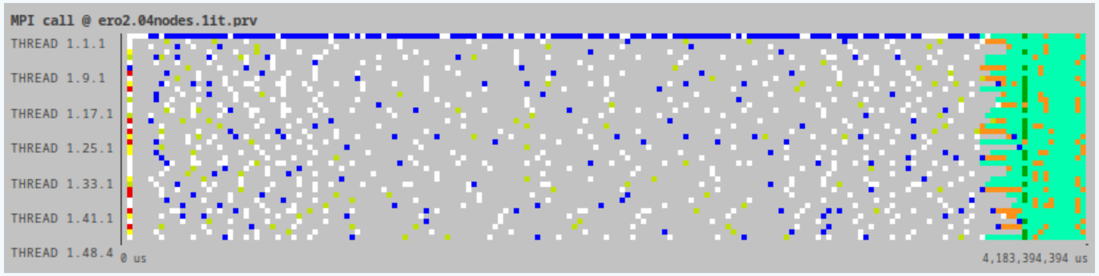


Scalability



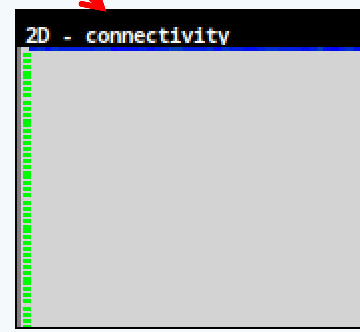
Gathering phase does not scale
Computation seems to scale.
Load imbalance increases.

Structure - Master/Slave

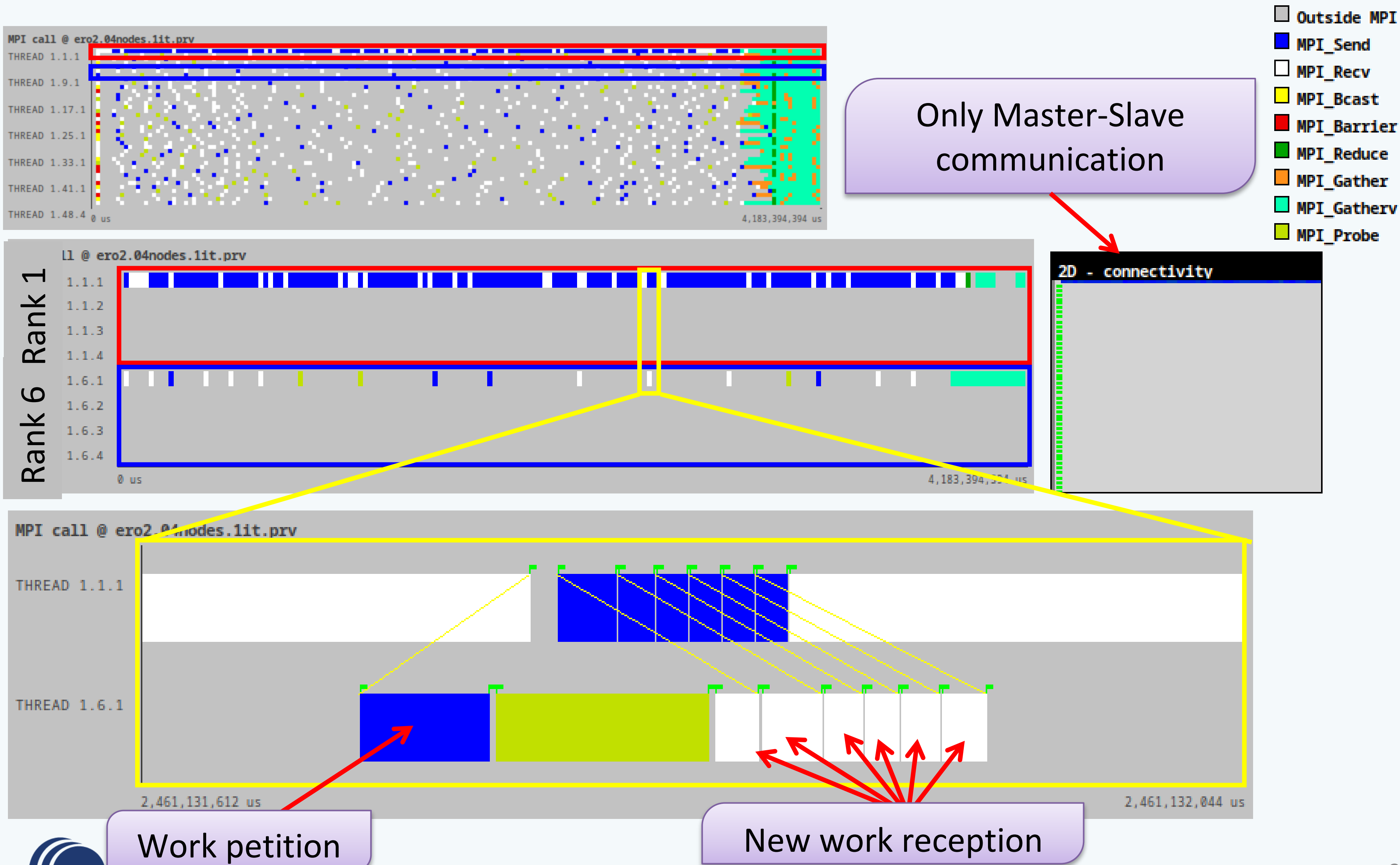


Outside MPI
MPI_Send
MPI_Recv
MPI_Bcast
MPI_Barrier
MPI_Reduce
MPI_Gather
MPI_Gatherv
MPI_Probe

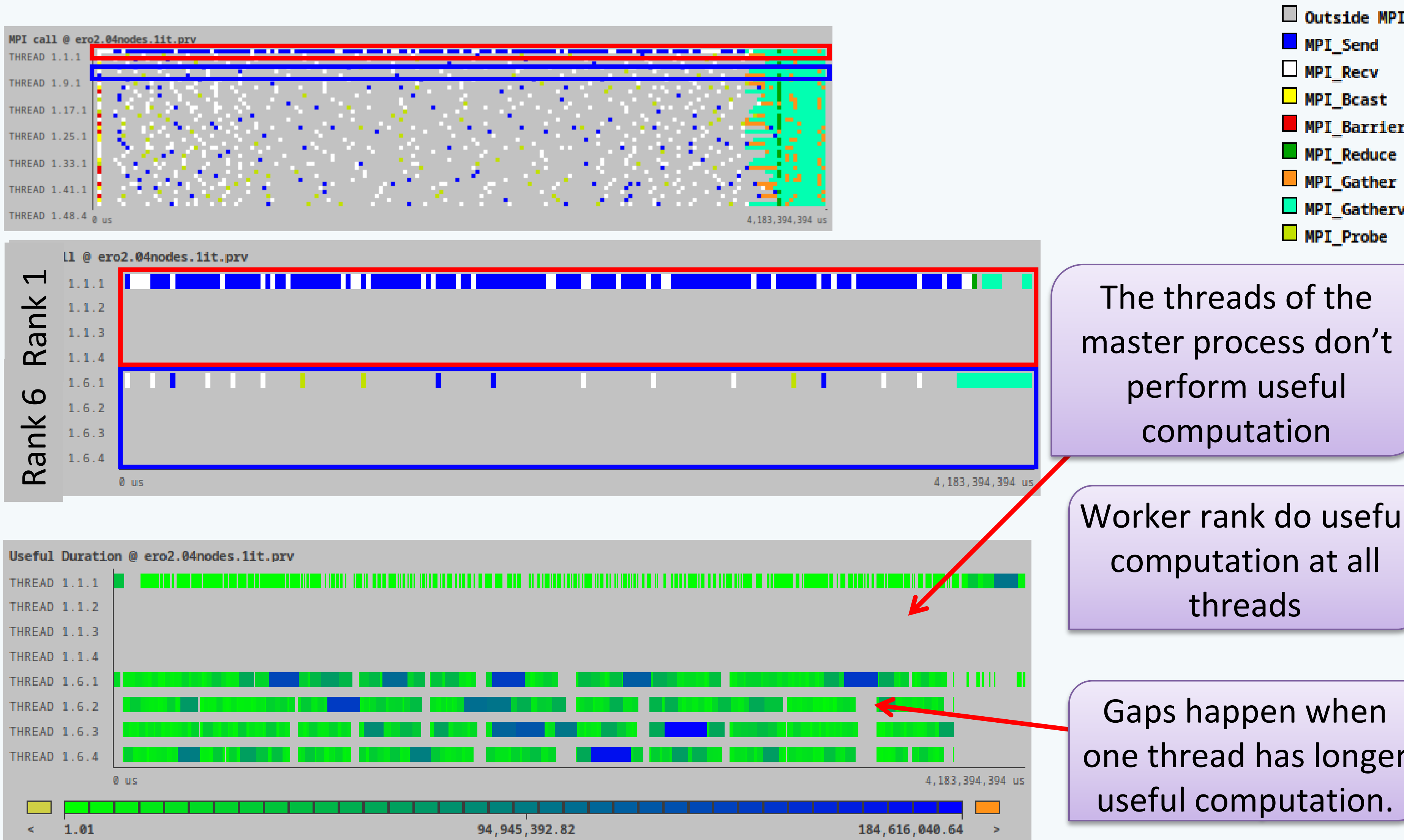
Only Master-Slave communication



Structure - Master/Slave



Structure - Master/Slave

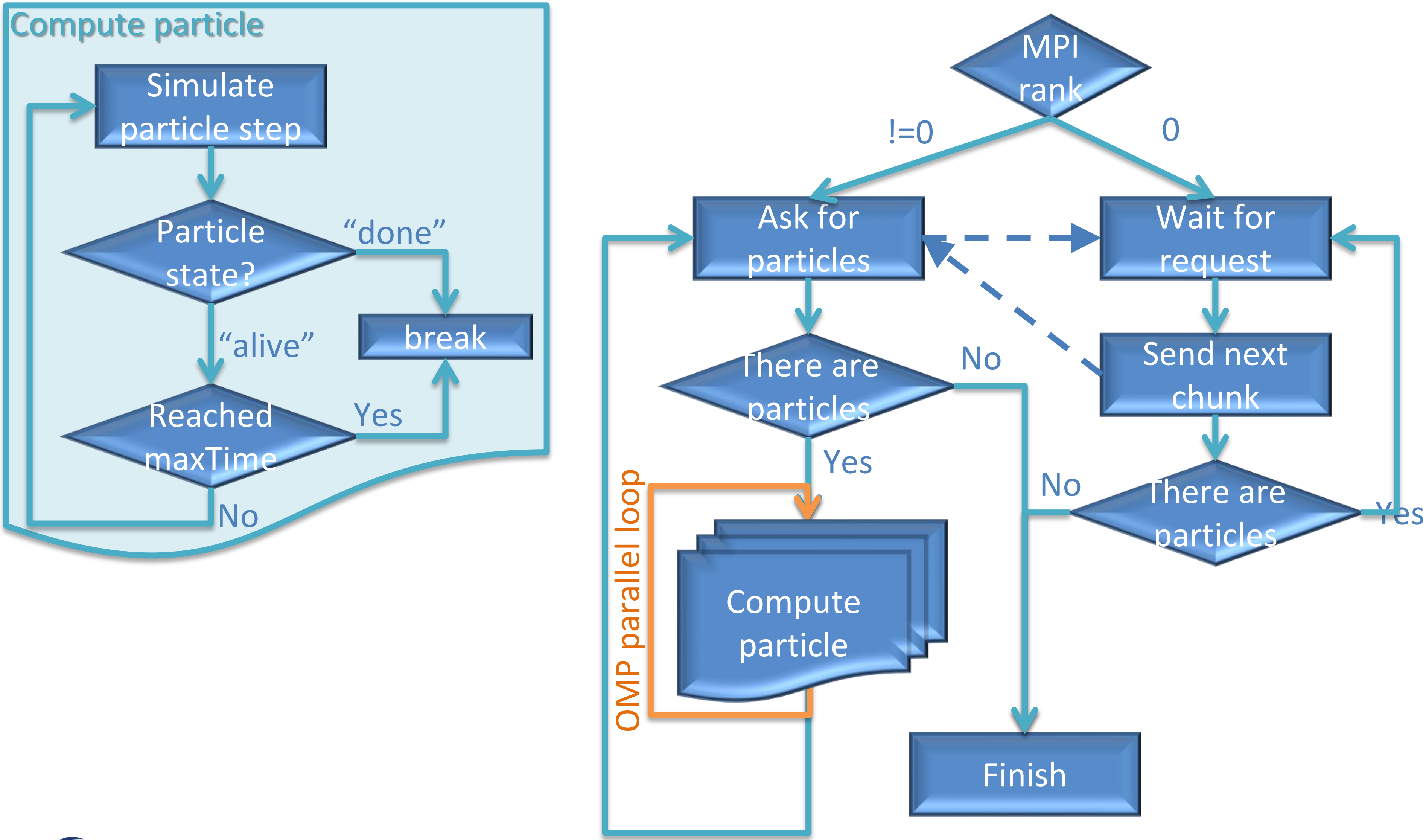


The threads of the master process don't perform useful computation

Worker rank do useful computation at all threads

Gaps happen when one thread has longer useful computation.

Summary structure



Efficiency metrics

Nodes:	4	8	16	32
	192(48x4)[1]	384(96x4)[2]	768(192x4)[3]	1536(384x4)[4]
Global efficiency	80.20	70.95	57.81	39.51
-- Parallel efficiency	80.20	70.59	58.33	40.43
-- Load balance	91.52	86.73	76.55	62.27
-- Communication efficiency	87.64	81.39	76.21	64.93
-- Computation scalability	100.00	100.52	99.10	97.73
-- IPC scalability	100.00	99.94	100.23	100.35
-- Instruction scalability	100.00	100.58	98.91	97.67
-- Frequency scalability	100.00	100.00	99.96	99.71

Percentage(%) scale: 0, 20, 40, 60, 80, 100

Very low global eff. due to bad parallel efficiency.

Very good computation scalability.

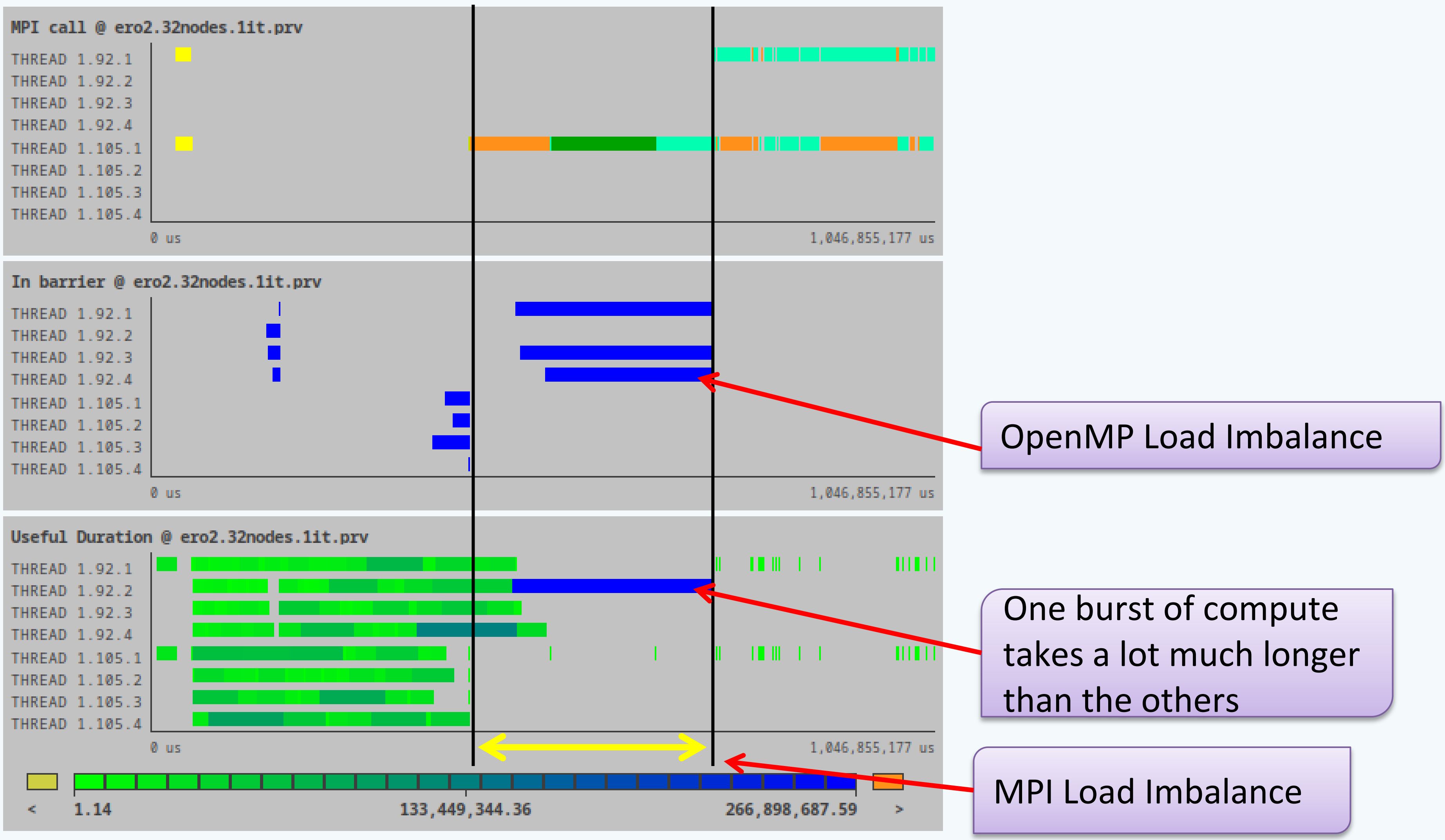
	192(48x4)[1]	384(96x4)[2]	768(192x4)[3]	1536(384x4)[4]
-- Hybrid Parallel efficiency	80.20	70.59	58.33	40.43
-- MPI Parallel efficiency	88.95	79.11	66.42	47.71
-- MPI Load balance	95.46	90.80	84.69	68.30
-- MPI Communication efficiency	93.18	87.13	78.42	69.86
-- Serialization efficiency	93.20	87.18	78.53	70.03
-- Transfer efficiency	99.98	99.94	99.86	99.75
-- OpenMP Parallel efficiency	90.17	89.23	87.83	84.74
-- OpenMP Load Balance	95.87	95.53	90.39	91.17
-- OpenMP Communication efficiency	94.05	93.41	97.17	92.94

Percentage(%) scale: 0, 20, 40, 60, 80, 100

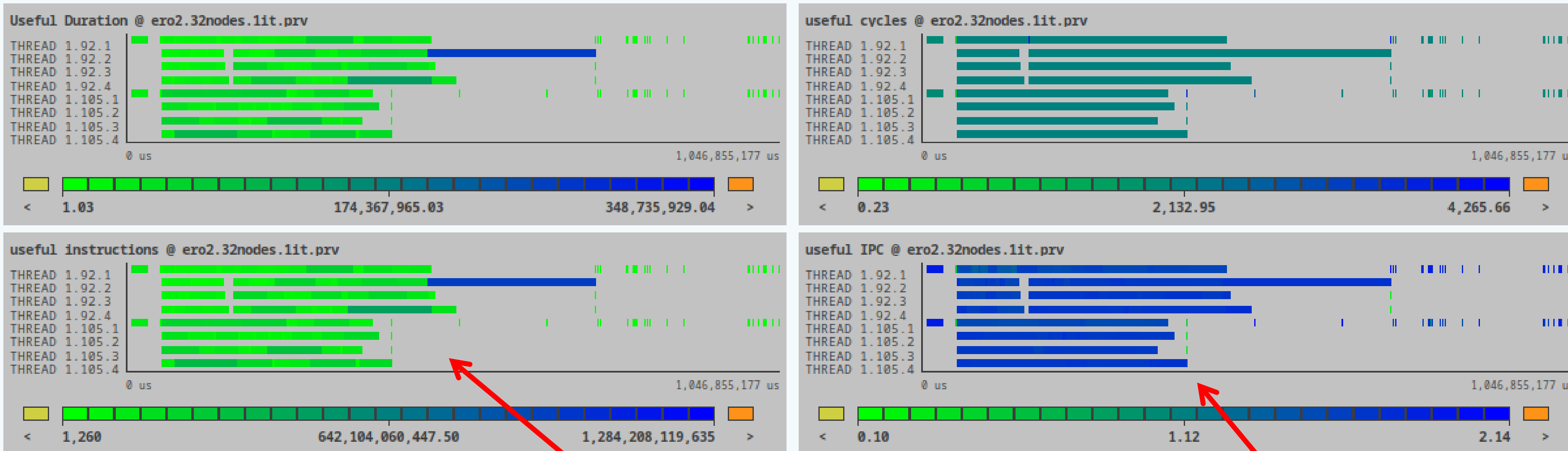
Very low MPI Parallel Eff. Due to Load Balance and Serialization.

OpenMP Parallel efficiency shows a tendency to decrease.

Load imbalance in detail



Load imbalance



The amount of instructions correlates with the burst duration.

Load balance seems to come due to more instructions. Hence more code execution.

On the other hand, the cycles and IPC are stable for most bursts.

Load Imbalance – grain

maxMpiChunkSize: 50 10

	1536(384x4)[1]	1536(384x4)[2]
Global efficiency	40.43	37.57
-- Parallel efficiency	40.43	33.39
-- Load balance	62.27	62.37
-- Communication efficiency	64.93	53.53
-- Computation scalability	100.00	112.51
-- IPC scalability	100.00	100.31
-- Instruction scalability	100.00	112.47
-- Frequency scalability	100.00	99.73

	1536(384x4)[1]	1536(384x4)[2]
-- Hybrid Parallel efficiency	40.43	33.39
-- MPI Parallel efficiency	47.71	53.22
-- MPI Load balance	68.30	74.46
-- MPI Communication efficiency	69.86	71.48
-- Serialization efficiency	70.03	71.65
-- Transfer efficiency	99.75	99.77
-- OpenMP Parallel efficiency	84.74	62.73
-- OpenMP Load Balance	91.17	83.77
-- OpenMP Communication efficiency	92.94	74.89

Percentage(%) color scale: 0, 20, 40, 60, 80, 100

MPI grain reduction

Overall Worst Load Balance

We tradeoff the OpenMP Load Balance for the MPI

Already using lowest OpenMP granularity

```
#pragma omp for shcedule(dynamic)
  for (int i=0; i<chunkSize; i++) {
     …
     transportParticleLoop (…);
     …
  }
}
```

Load Imbalance summary

- MPI
  - Only present at the end of the execution
  - Due to:
    - Heterogeneous distribution of “particle chunks”
    - “Long” particles

- OpenMP
  - Present at the end of every “particle chunk”
  - Due to:
    - Implicit OpenMP barrier at the end of parallel
    - “Long” particles

Proof of concepts

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

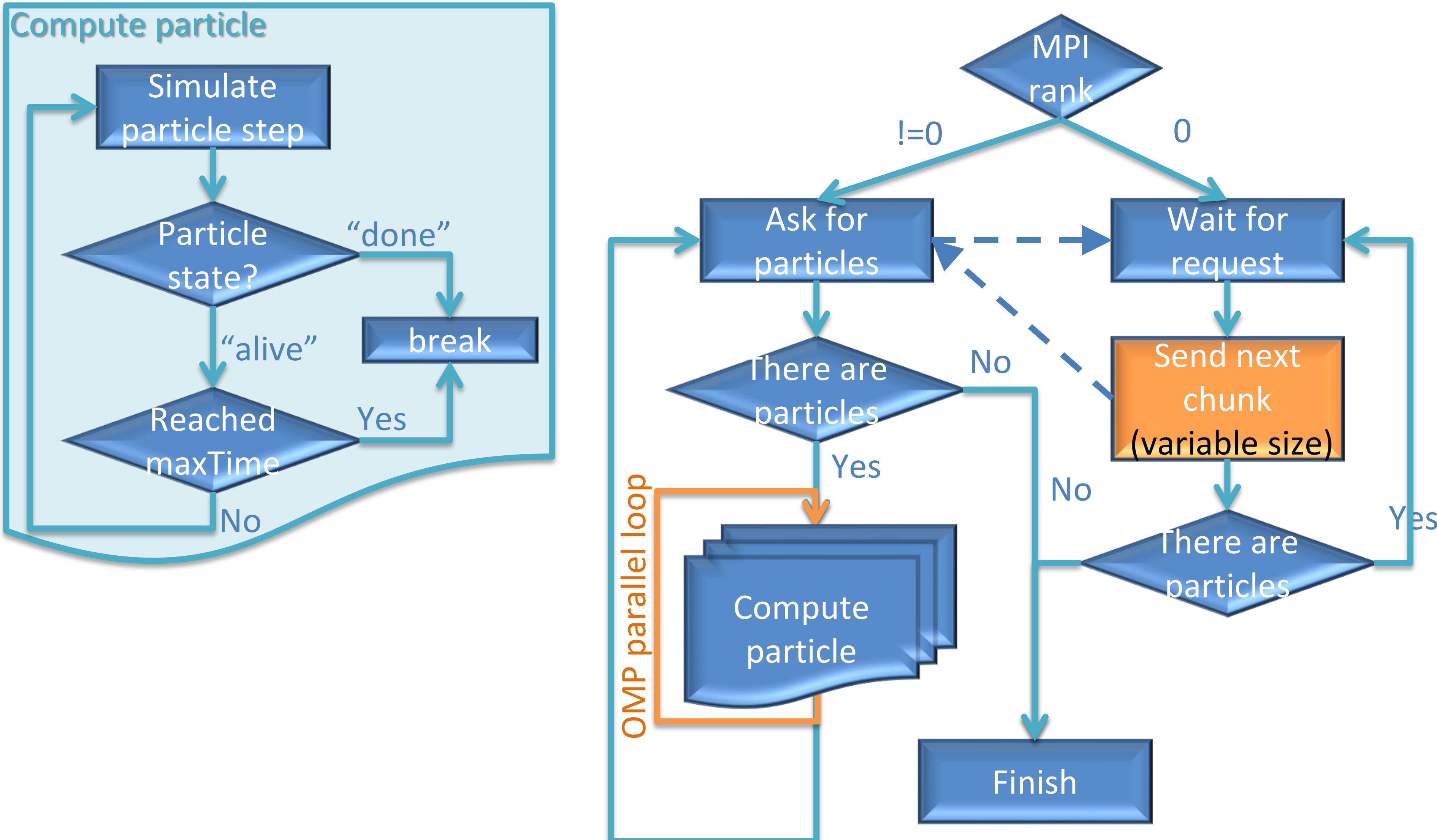
PoC1: Guided-like particle chunk size

Objective:
- Start with a big granularity so we reduce the overhead and improve OpenMP Load balance.
- Reduce the granularity towards the end to MPI load imbalance.

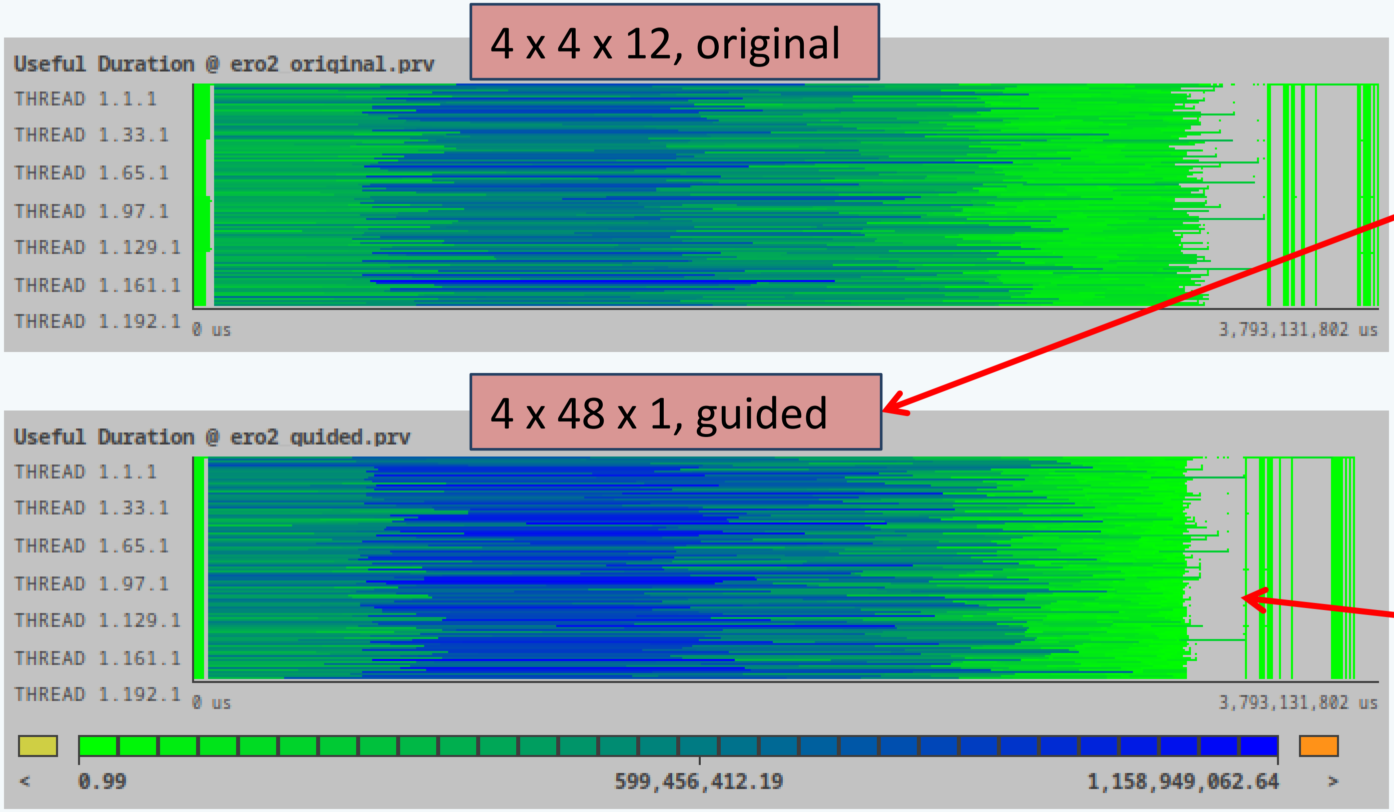
How we do it:
- We assign the chunk size to be the 1/3 of particles remaining (divided by number of processes).
- We set the minimum to 1.
- Default set to input value

$$nRemaining = nParticles - nSent$$
$$CS_{guided} = max(1, (nRemaining / comm\_size) * 0.3)$$
$$CS = min(nRemaining, CS_{guided})$$

PoC1: Guided-like particle chunk size



PoC1: Guided-like particle chunk size



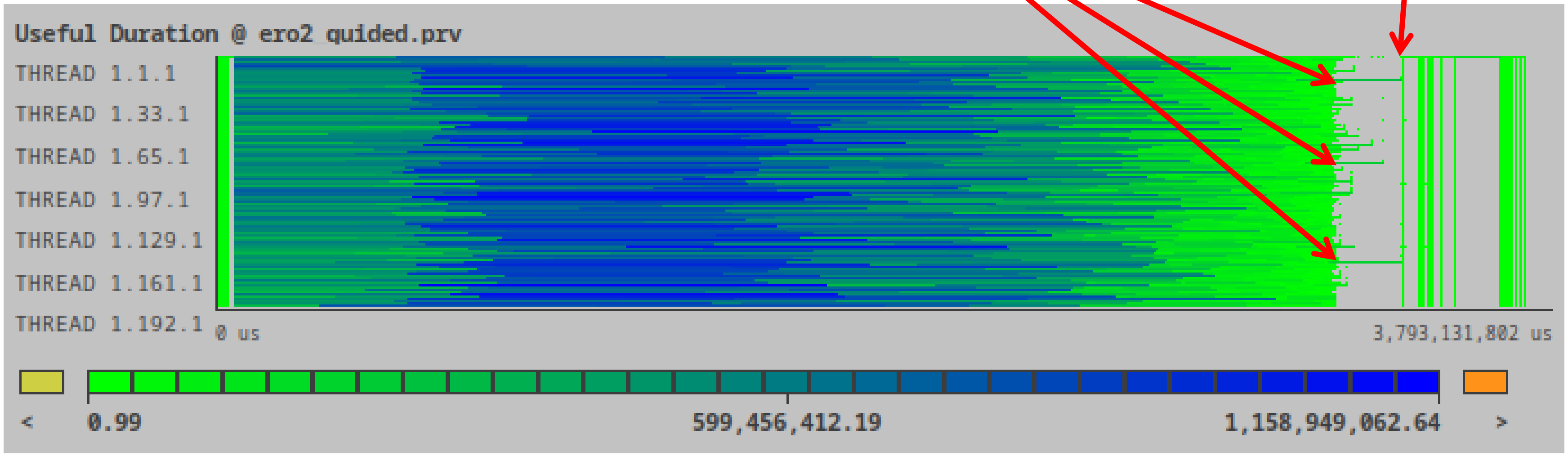
The load imbalance gets better with this version.

Even though we have improved the load balance, we still observe some outliers that keep us from further improving.

PoC2: Dynamic maxTracingTime

Is it worth to wait for these 3 particles?

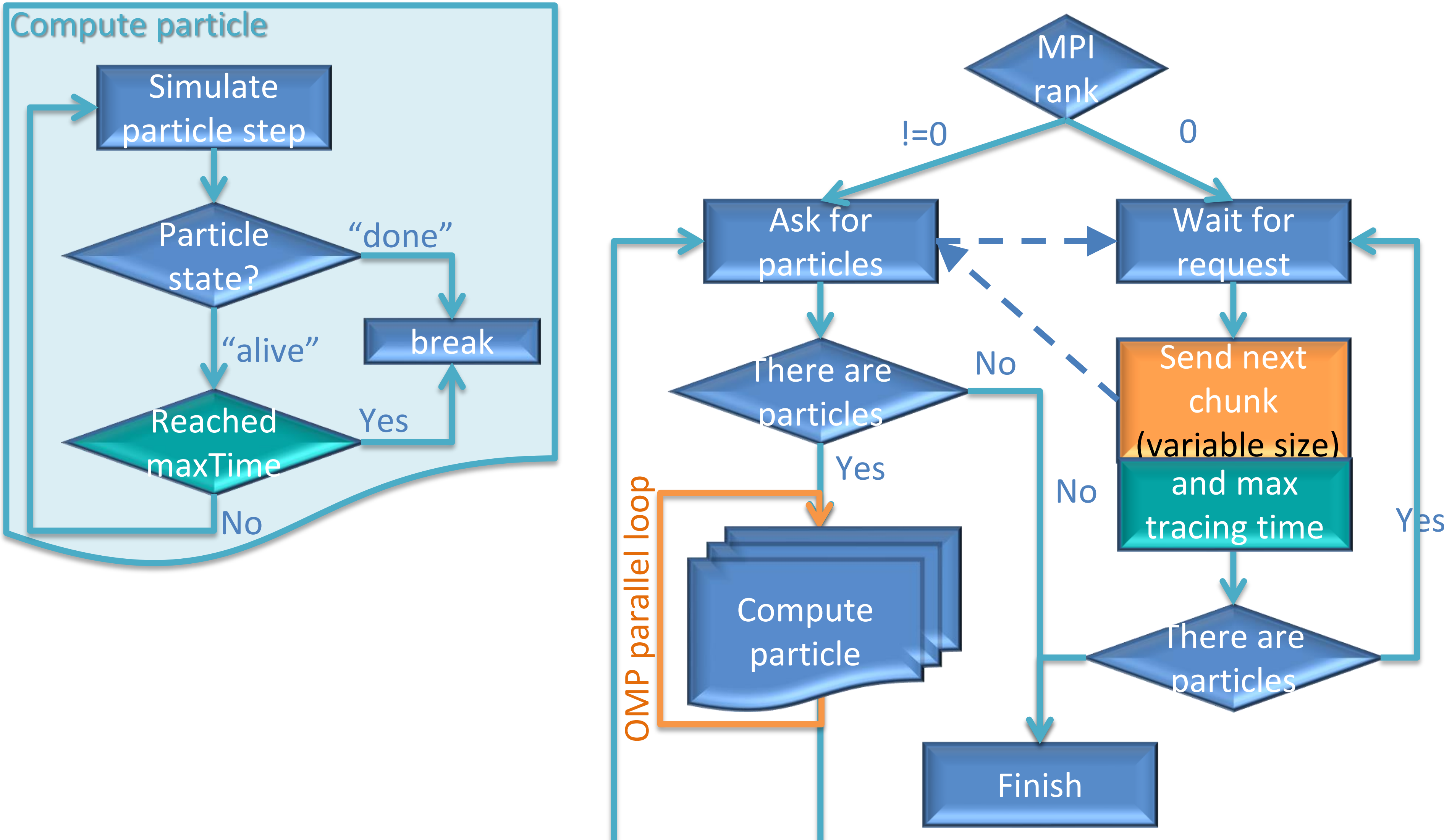
They will probably be killed here can we kill them earlier?



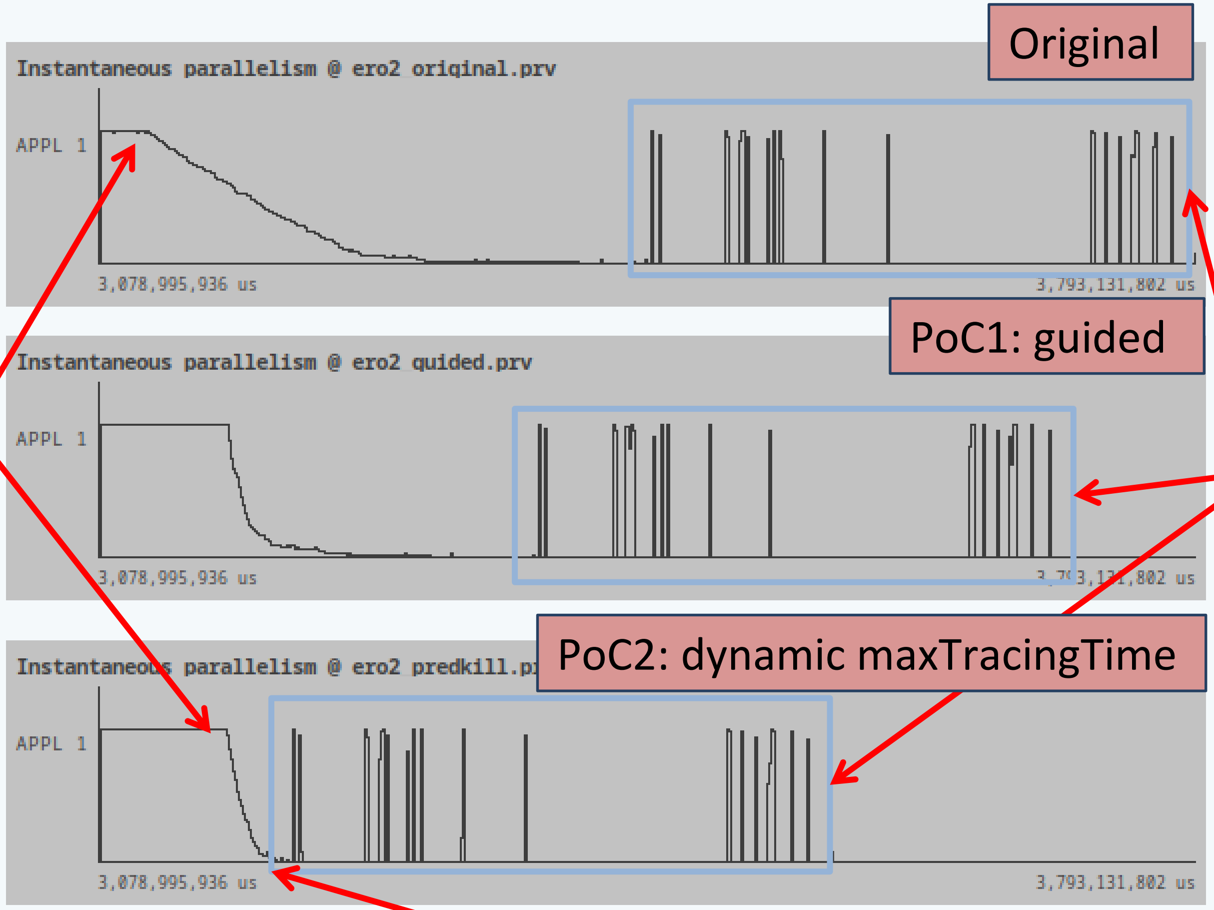
Objective:
Detect early that the particle is lost and kill it earlier

How:
We keep track of the average tracing time of particles, and kill particles that are above that threshold

PoC2: Dynamic maxTracingTime



Comparing Original, PoC1 and PoC2

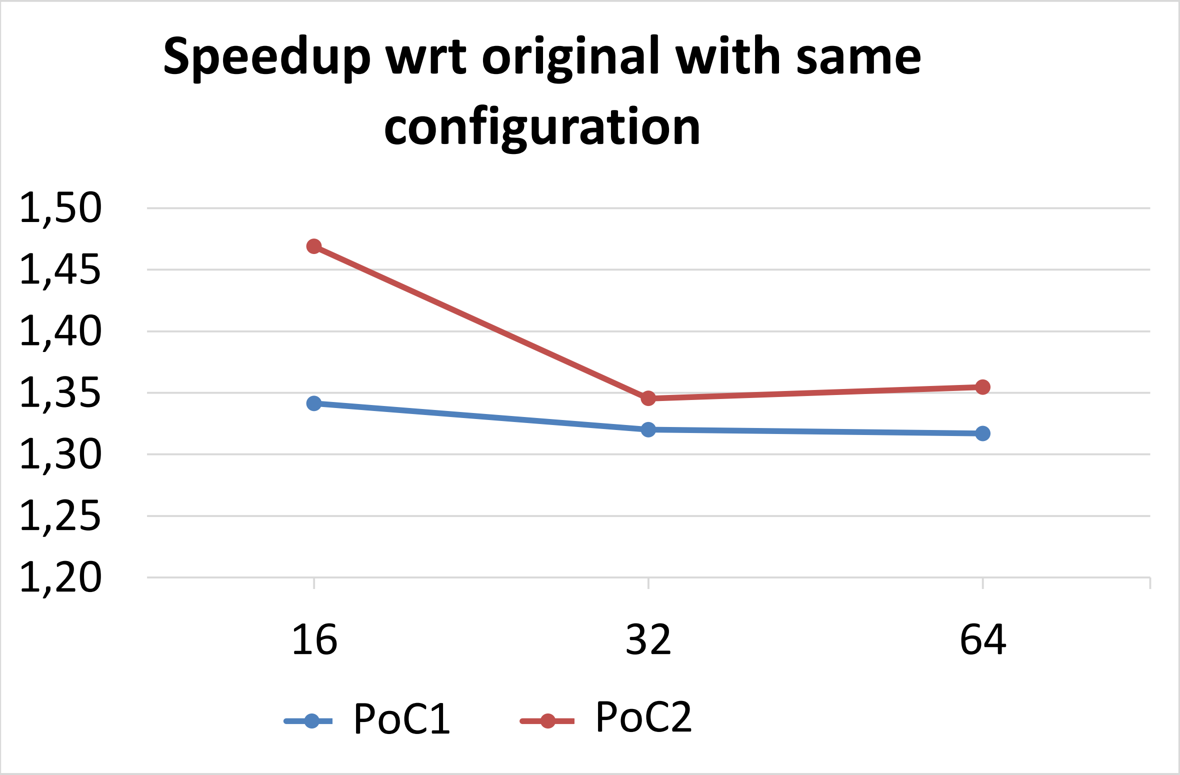


We start losing efficiency later, for the two new versions

Gather phase

PoC2 (dynamic maxTracingTime) ends the simulation when running inefficiently.

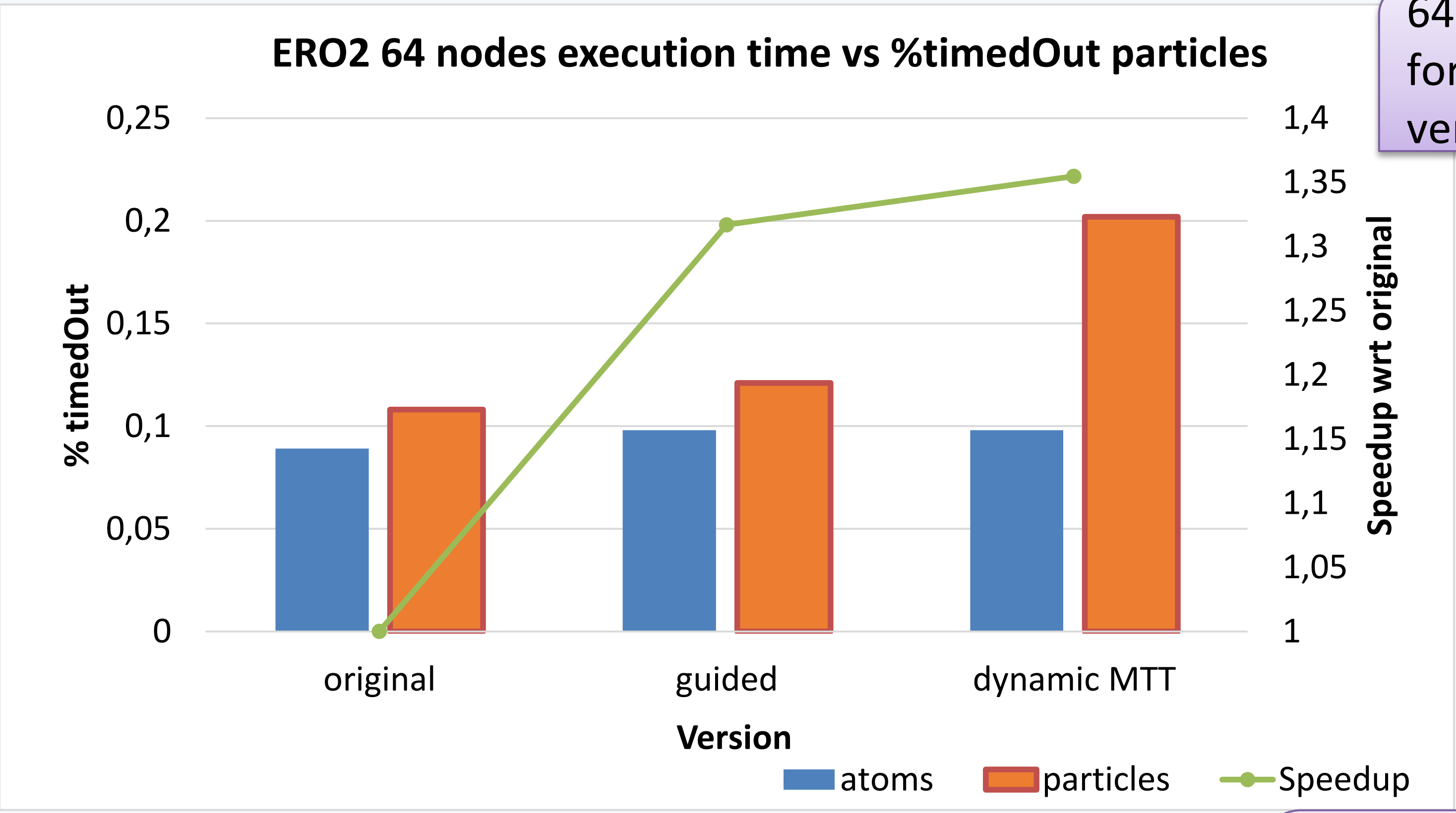
Comparing Original, PoC1 and PoC2



For executions with low number of nodes we observe huge improvement with the "dynamic maxTracingTime".

We observe at least a 1.3 speedup for all node counts with PoC1 and 1.35 for PoC2.

Comparing Original, PoC1 and PoC2



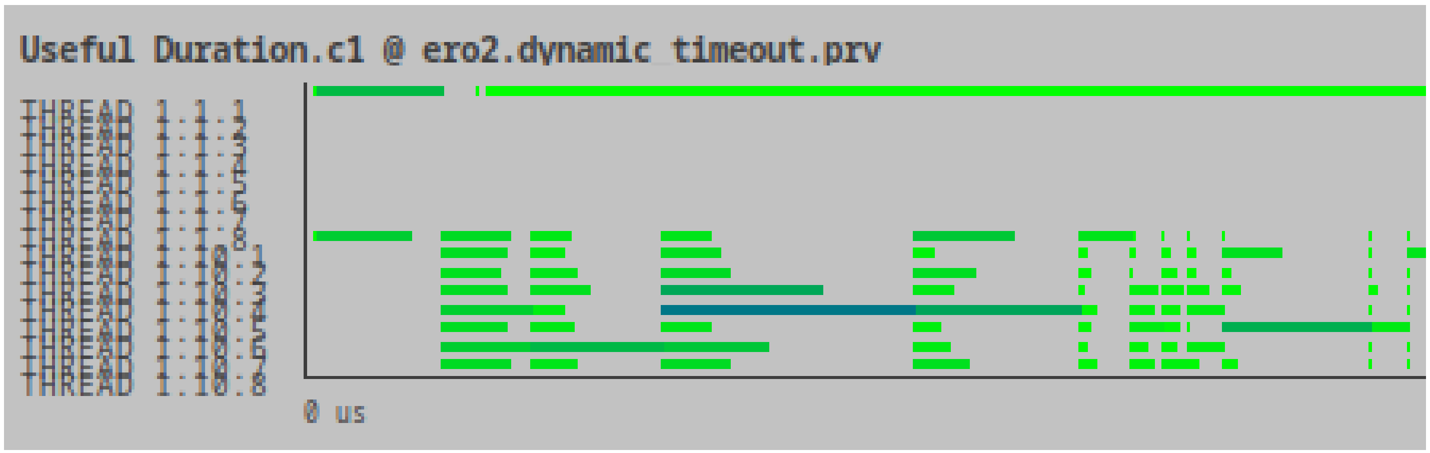
64 node execution for the 3 different versions.

PoC1 (guided)
- 1.32 speedup
- Insignificant number of atoms and particles timed out.

PoC2 (dynamic MTT)
- 1.35 speedup
- ~1% increase of timed out particles.

PoC3: Outer parallel

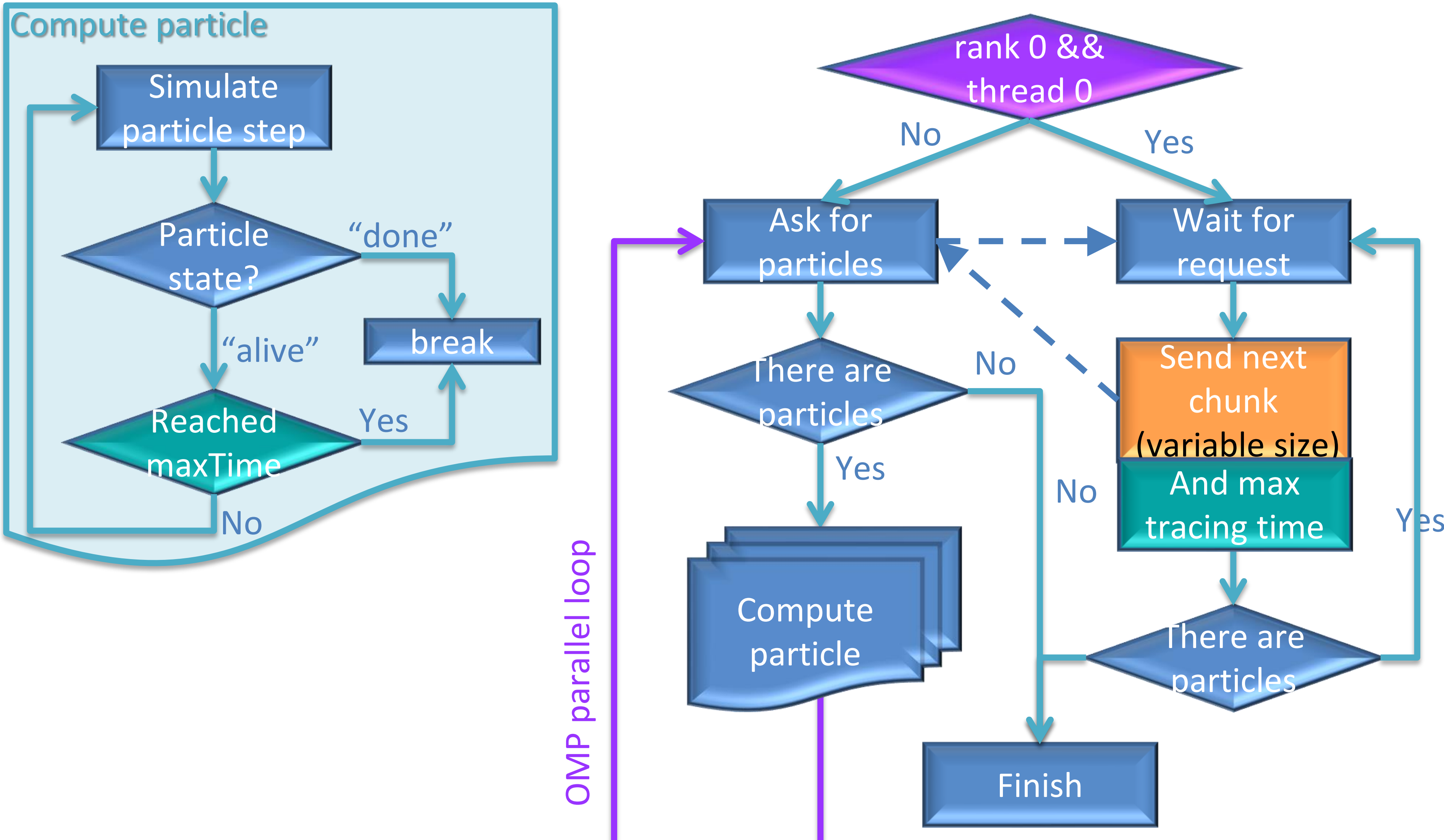
MPI load imbalance has been almost completly removed, only OpenMP load imbalance is remaining



Objective:
avoid OpenMP threads waiting for other threads

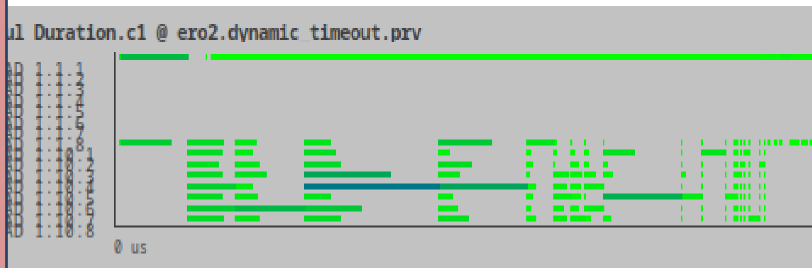
How:
Allow threads to start computing next chunk of particles before all threads finish their particles

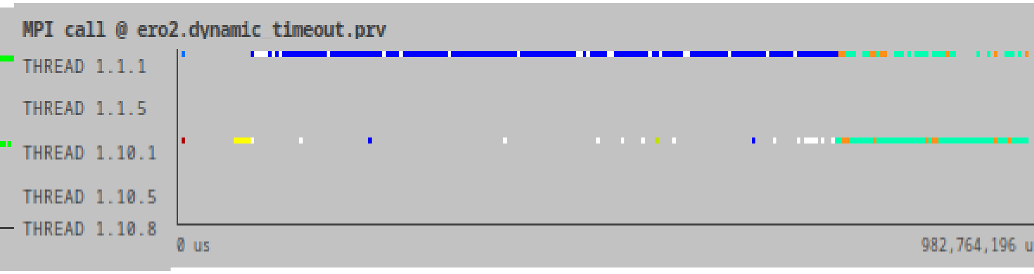
PoC3: Outer parallel



PoC 3: Outer Parallel

64 x 6 x 8, guided

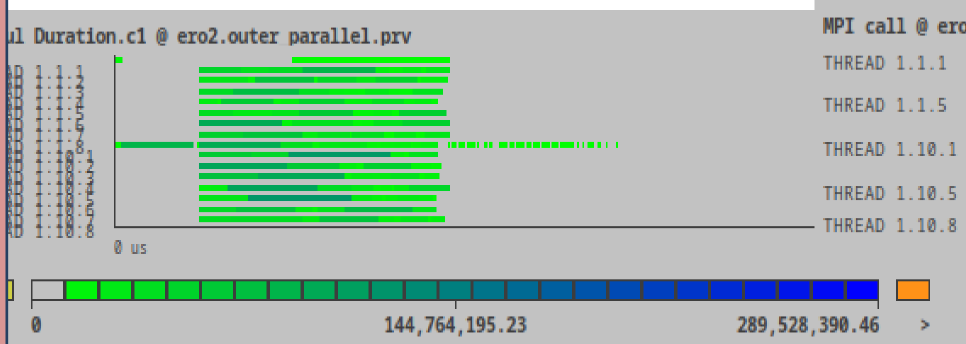


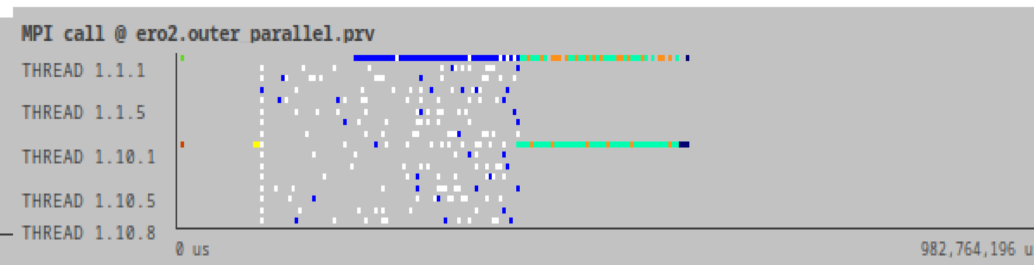


Useful computation

MPI calls

64 x 6 x 8, outer parallel





No load imbalance is observed among different chunks.

All threads perform communication, even those from rank 0.

Threads from Rank0 also compute particles

Summary - Conclusions

- Analyzed and optimized ER2.0 code
  - Main issues affecting efficiency:
    - OpenMP load imbalance
    - MPI load imbalance
  - 3 incremental optimizations:
    - Guided-like distribution of particles among MPI processes
    - Dynamic max tracing time for particles
    - Outer OpenMP loop
  - Speedup achieved 1.35

- Successful interdisciplinary work

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

THANK YOU!

joan.vinyals@bsc.es
marta.garcia@bsc.es

www.bsc.es