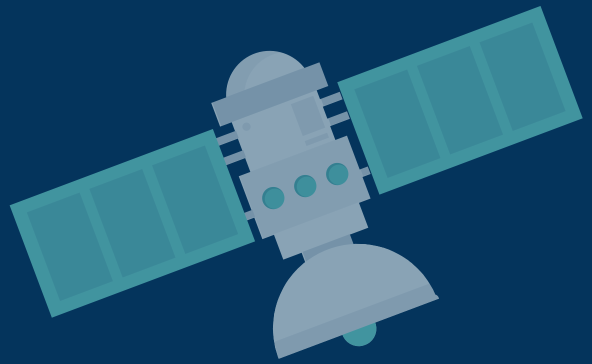
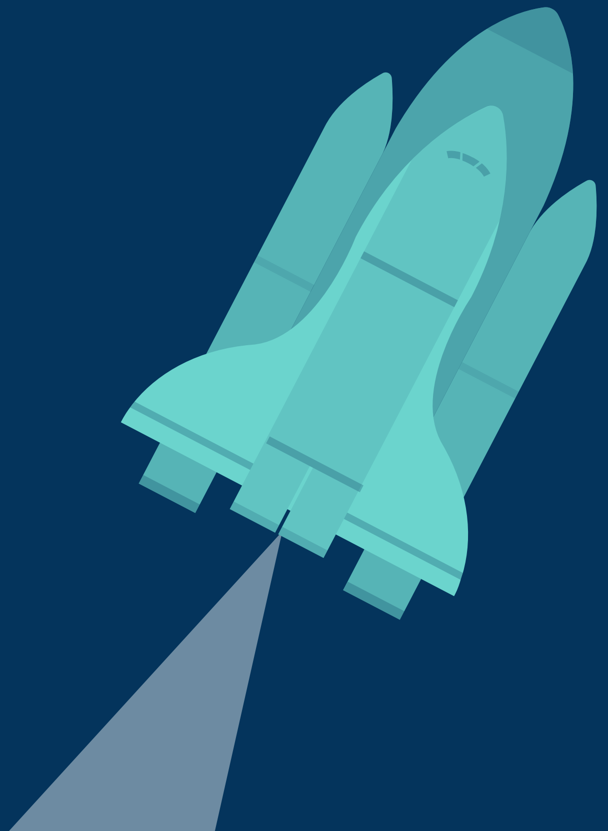



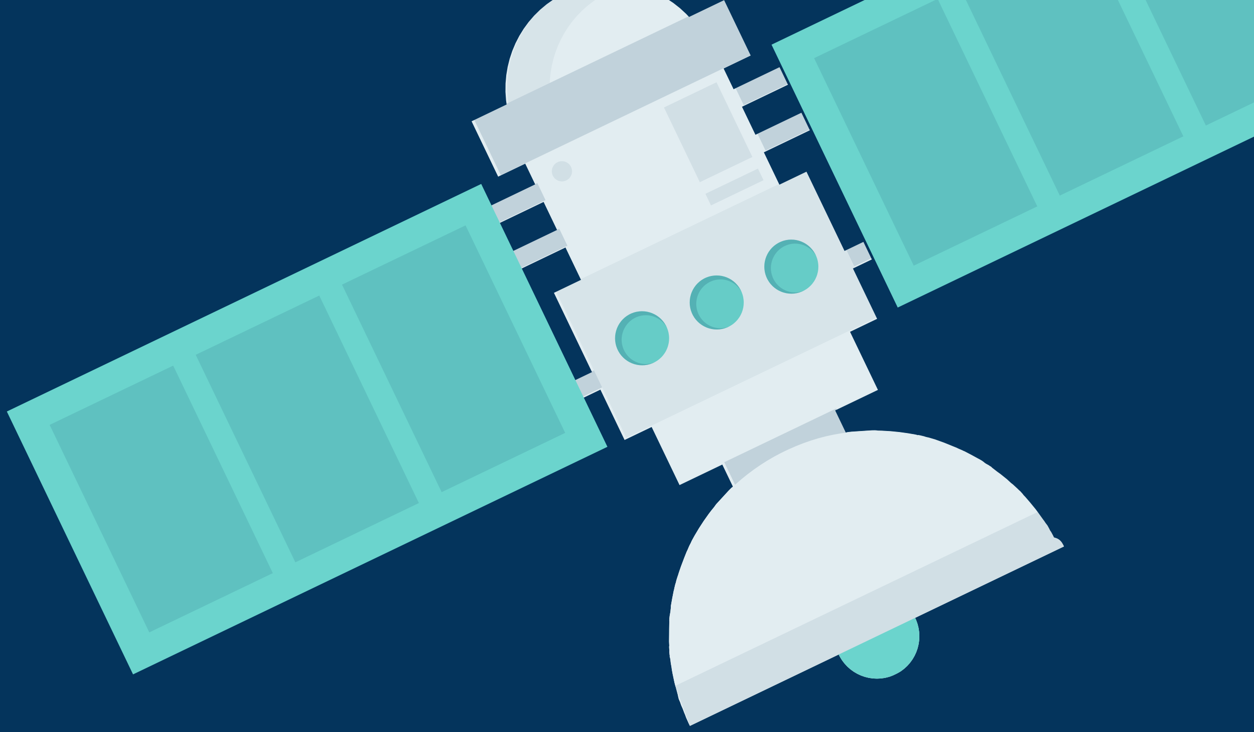
HACC

The Hardware Accelerated Cosmology Code framework

Curated by Roberto Stagi

PARALLEL PROGRAMMING TOOLS AND MODELS - 2020 @ UPC

Presentation Outline



Part 1

Introduction and set-up

Why HACC?

What is HACC?

The Hardware Accelerated Cosmology Code (HACC) framework uses N-body techniques to **simulate the formation of structure** in collisionless fluids under the influence of gravity in an expanding universe.

The main scientific goal is to **simulate the evolution of the Universe** from its early times to today and to **advance our understanding of dark energy and dark matter**, the two components that make up 95% of our Universe.

The code is hybrid **MPI-OpenMP** and depends on external FFT library.

Steps to build HACC

All the source code is available on my GitHub:
github.com/rstagi/HACC_MareNostrum_analysis/

TAKE THE ENVIRONMENT EXAMPLE FILE

HACC provides some example bash files for environment setup

FILL WITH THE LOCATION OF THE DEPENDENCIES

The environment variables need to point to MPI and FFTW libraries

LAUNCH THE BUILD

Once the environment is set up, just run the build using the Makefile

Part 2

Prepare for the analysis

Input files, job definition, run scripts

Command and options

The executable, called "hacc_tpm", receives some input parameters:

```
hacc_tpm   <INDAT>
           cmbM000.tf
           m000 INIT
           ALL_TO_ALL
           -w -R -N 512
           -a final -f refresh
           -t <GEOMETRY>
```

- INDAT is the path to the input configuration file

- GEOMETRY represents the cartesian decomposition of the MPI processes

Input file and constraints

Inside the INDAT file there are 3 values related to the problem size:

- **np**: number of particles per dimension ("alive" particles)
- **ng**: number of grid points per dimension
- **Physical Box size**: size of the space reference

Constraints:

- **np** must be equal to **ng**
- **np/ng** must be **divisible by any 2D decomposition** of the GEOMETRY (e.g. if 8x4x2, np must be divisible by 8x4, 8x2 and 4x2)

Script to run the jobs

To better design the scaling analysis, the **job** was designed to get the **data from arguments** and **environment variables**:

```
OMP_NUM_THREADS=<OPENMP_THREADS_COUNT> \
sbatch    --job-name=<JOB_NAME> \
          --ntasks=<MPI_TASKS> \
          --chdir=<WORKING_DIRECTORY> \
          --time=<TIME> \
          --qos=<QUEUE_NAME> \
          --get-user-env \
          job_hacc.sh <EXECS_DIRECTORY> <INDAT_FILE> <GEOMETRY>
```

Script to run the jobs (cont.)

- A **script** was designed **for each type of analysis** (strong, weak, openmp, etc.), by just varying environment variables and input arguments
- Each job was running in **its own working directory**
- Each job was instrumented using **Extrae**

All the source code is available on my GitHub:
github.com/rstagi/HACC_MareNostrum_analysis/

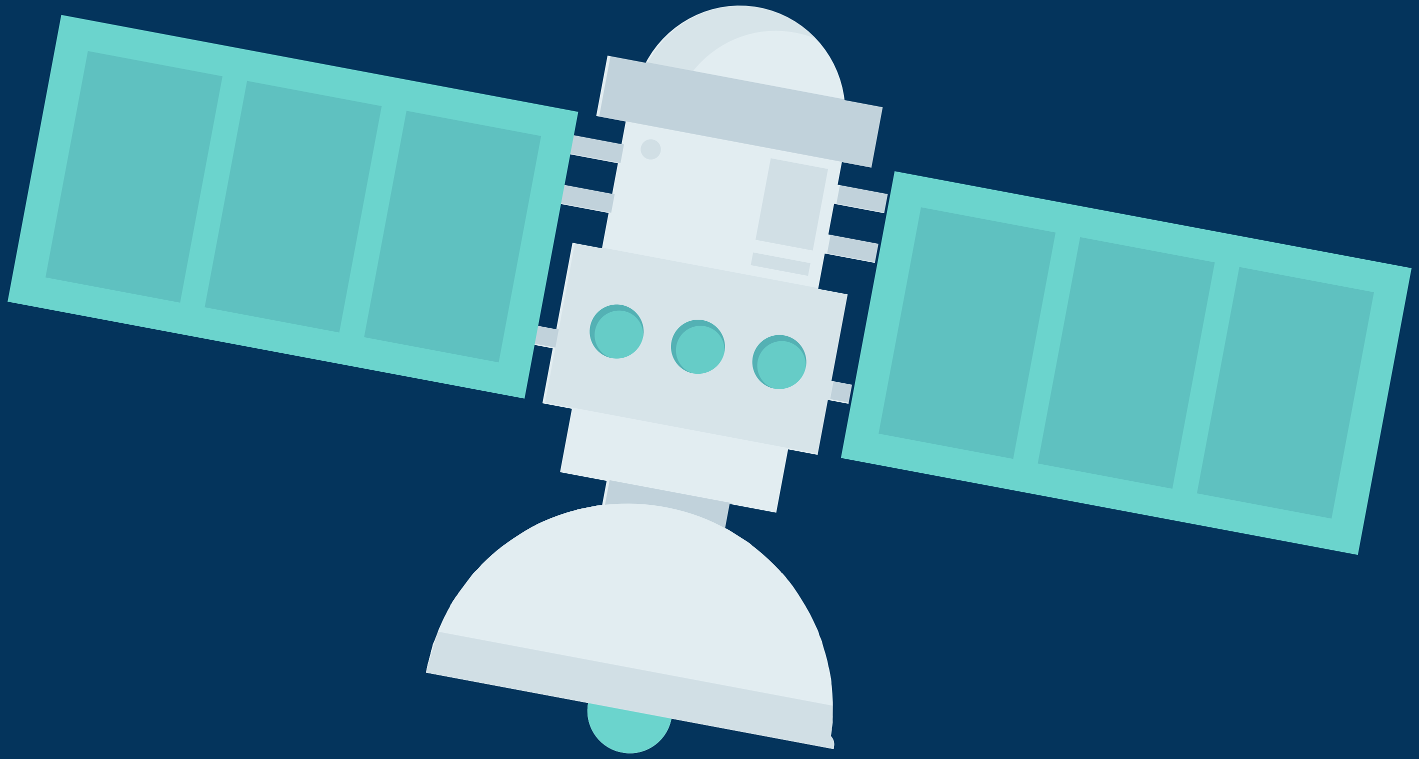
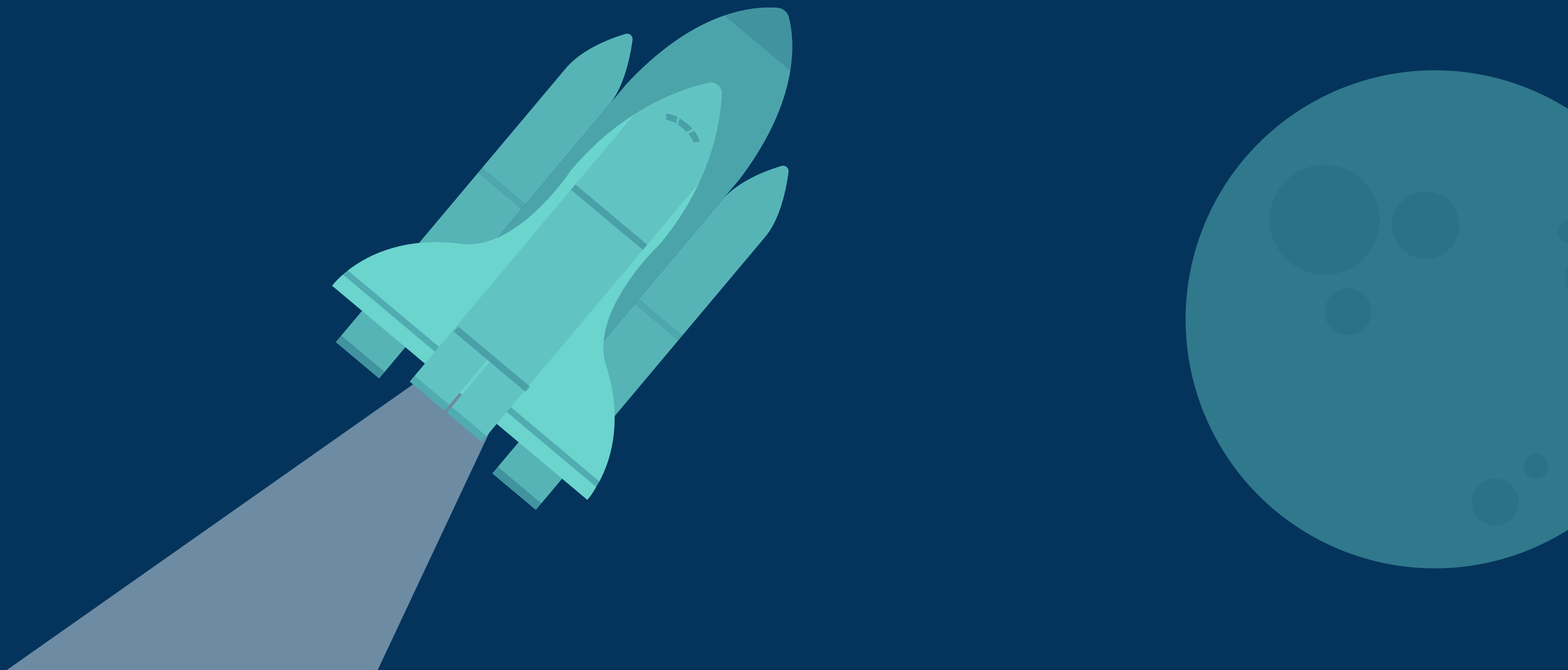
Problem size scaling

- The size of the example jobs was too large
  - np and ng reduced by at least 13x (problem size 13^3 times smaller)
- Scaling the problem size needed some extra care
- Problem size is defined by the total number of particles
- The space reference was a cube; **np** and **ng** refers to one dimension
- Scaling **X times** the number of particles means scaling **np** and **ng** by the cubic root of X

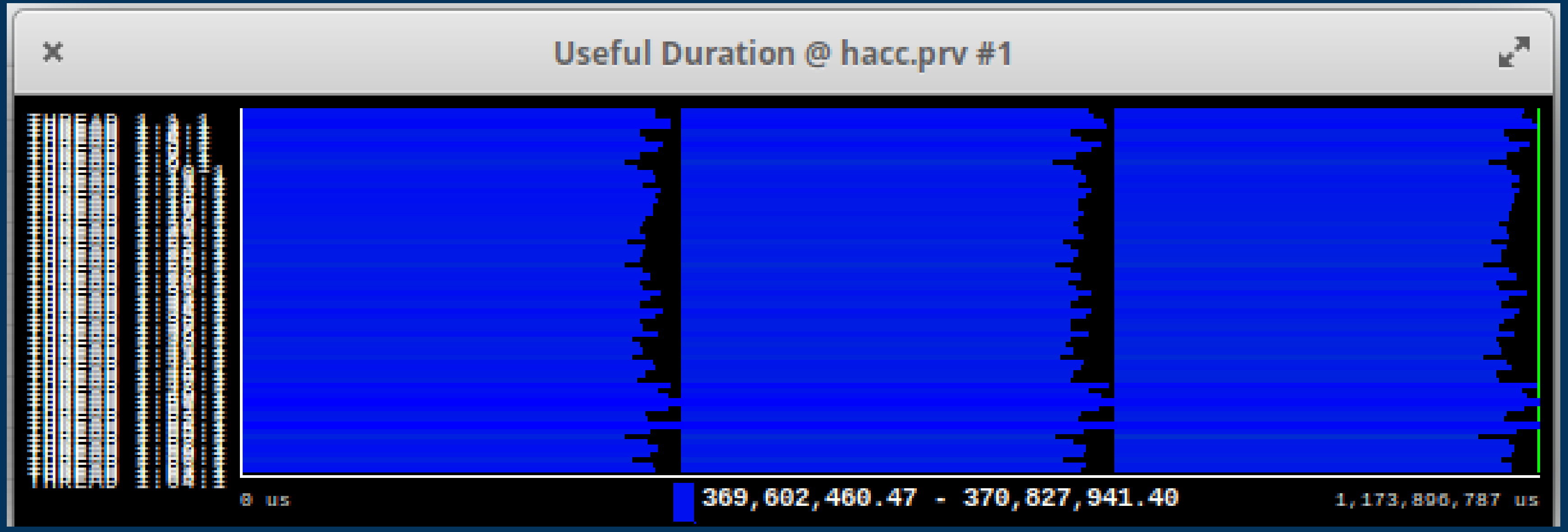
Part 3

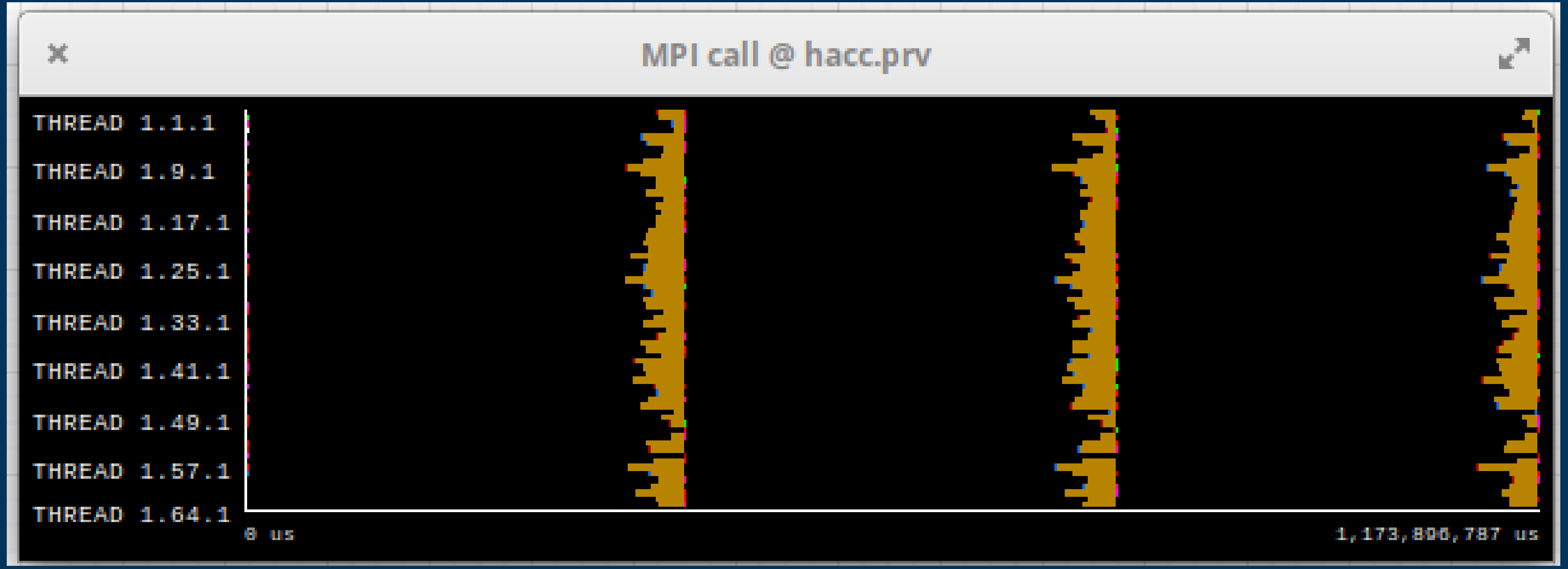
Analysis Traces

Structure, Scalability, Efficiency Model

Overall Structure

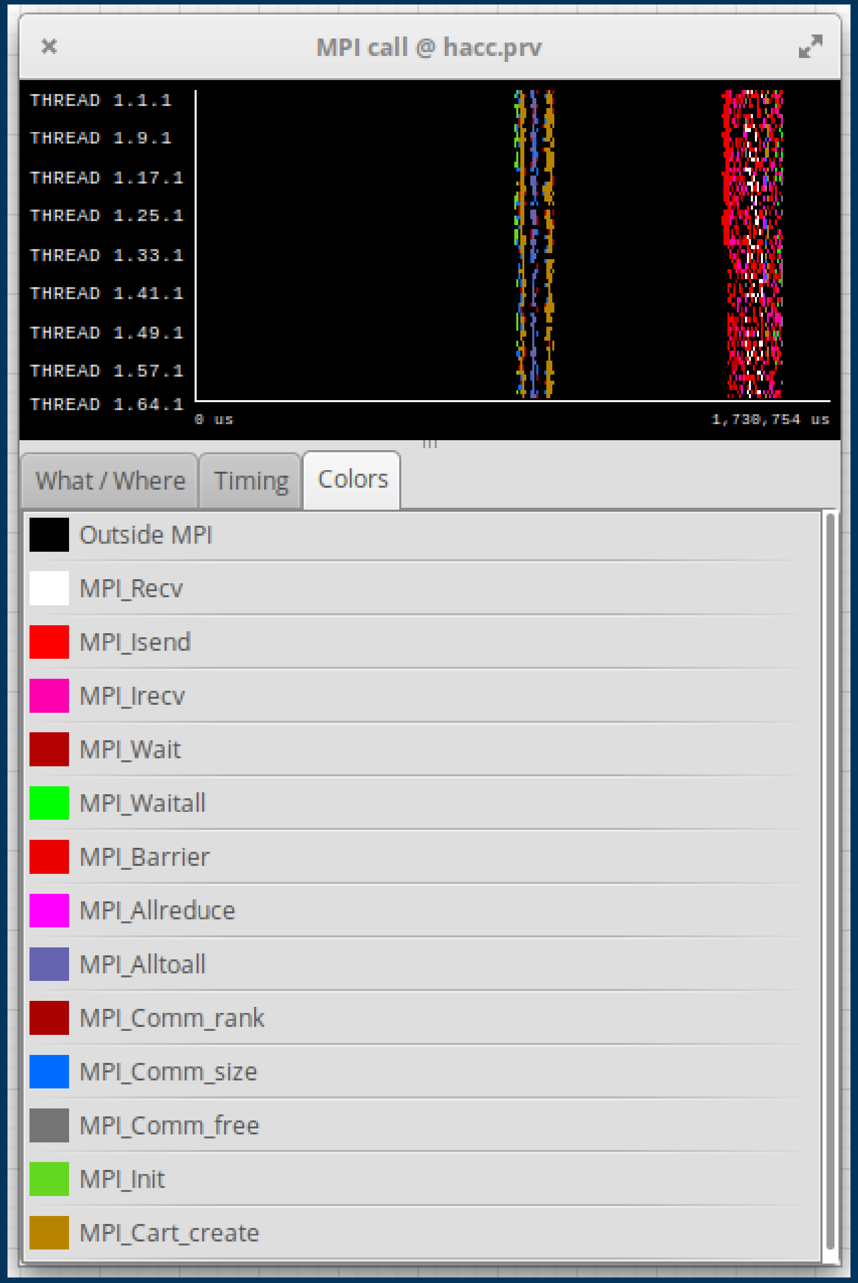




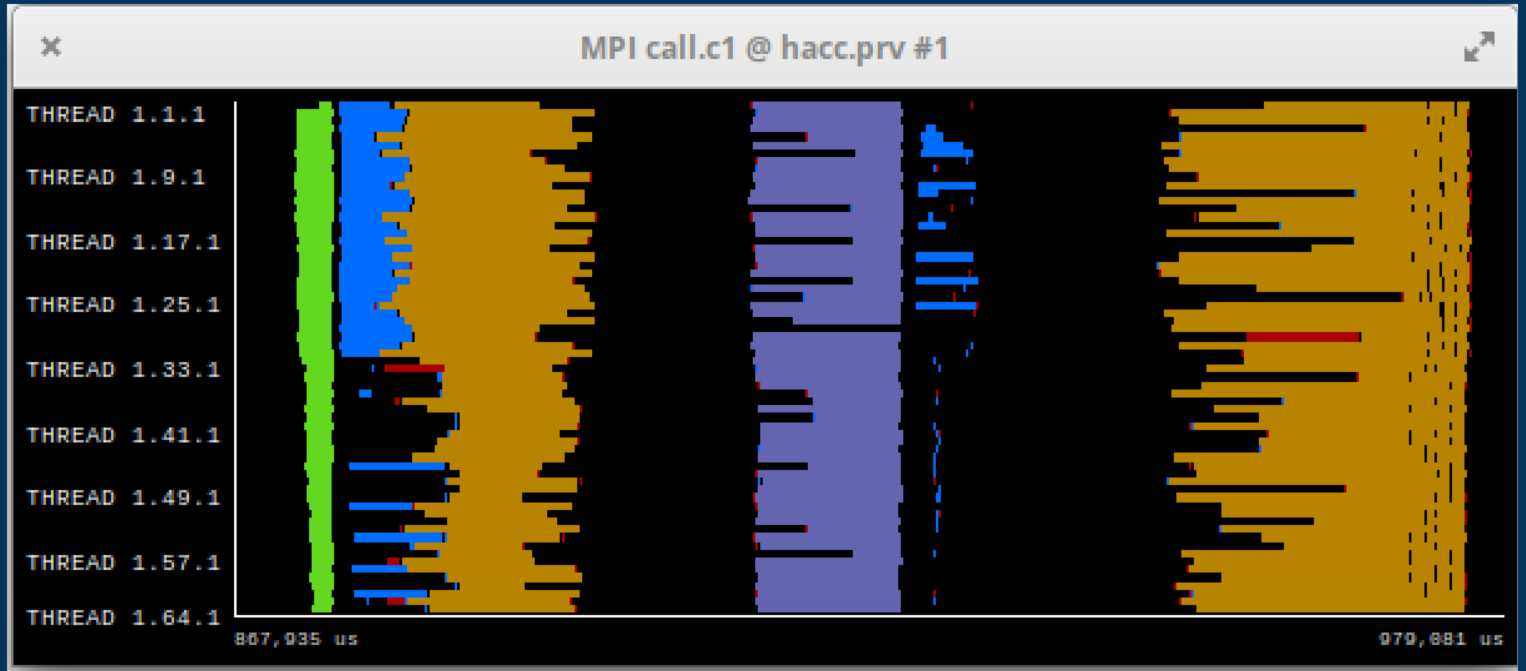
Based on a 64 MPI processes and 1 OpenMP thread execution

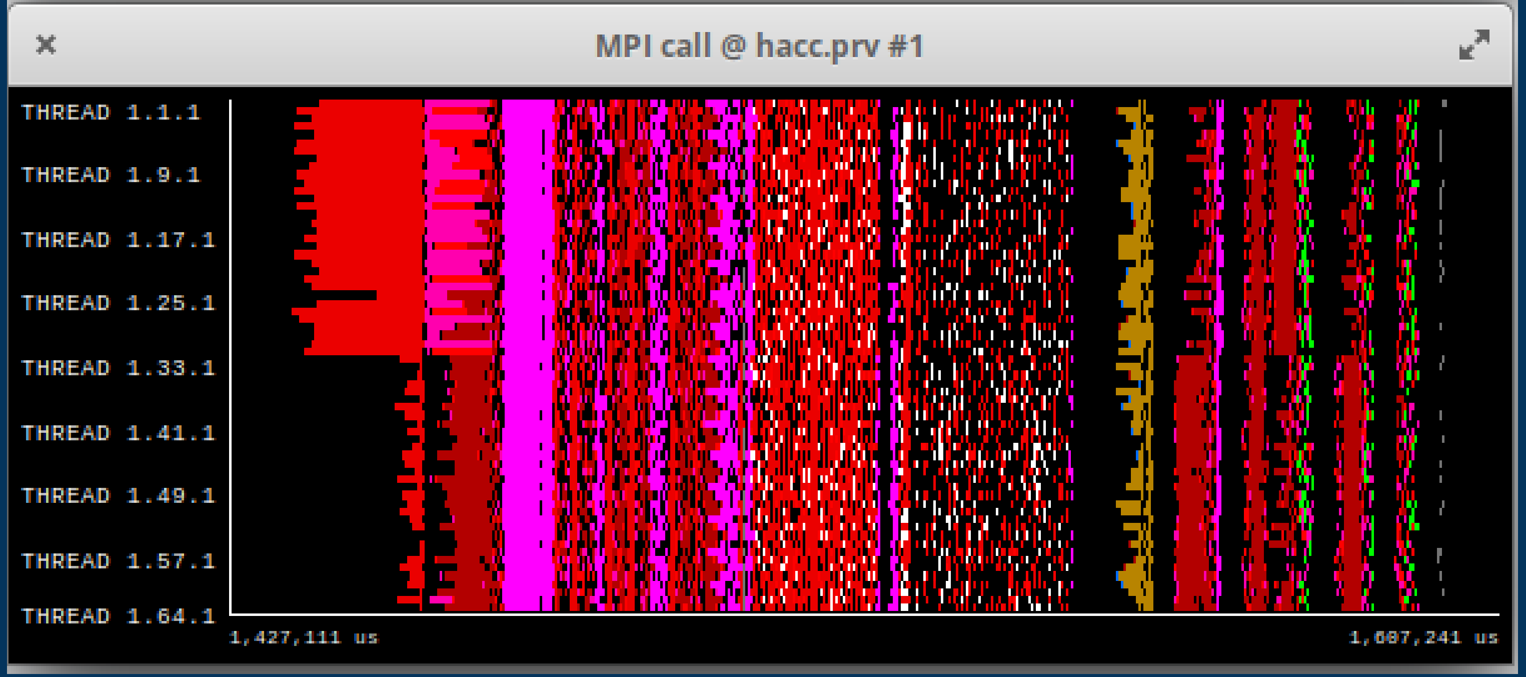
- Iterative execution
- Long duration runs for each iteration
- MPI calls at the end of each iteration

Structure - Initialization



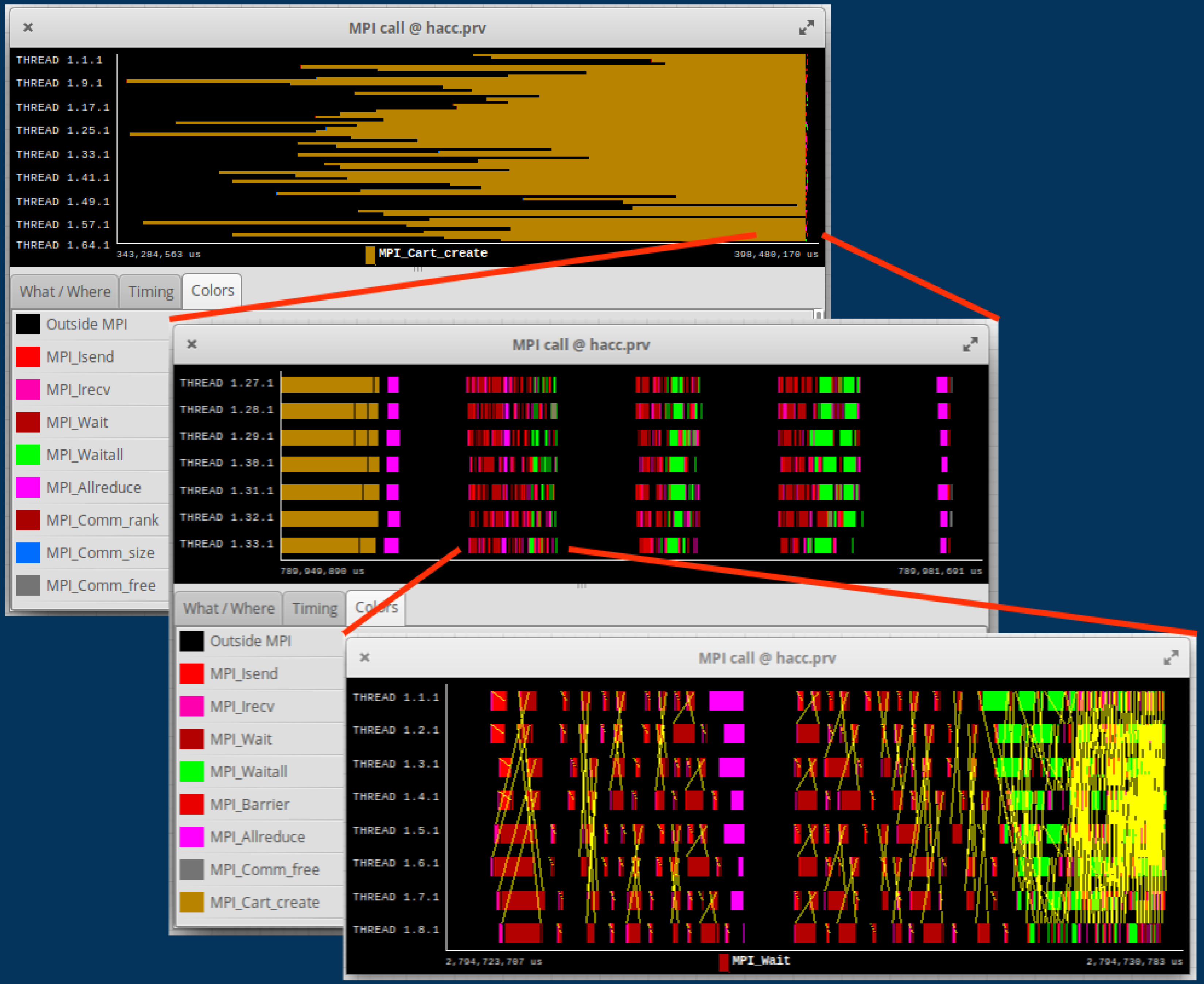
⬛	Outside MPI
⬜	MPI_Recv
🟥	MPI_Isend
🟪	MPI_Irecv
🟥	MPI_Wait
🟩	MPI_Waitall
🟥	MPI_Barrier
🟪	MPI_Allreduce
🟪	MPI_Alltoall
🟥	MPI_Comm_rank
🟦	MPI_Comm_size
⬛	MPI_Comm_free
🟩	MPI_Init
🟧	MPI_Cart_create





- MPI Cartesian grid init
- MPI All-to-All initialization
- Cartesian 1D, 2D and 3D decomposition init
- Particles load (input data)

Structure - Iteration

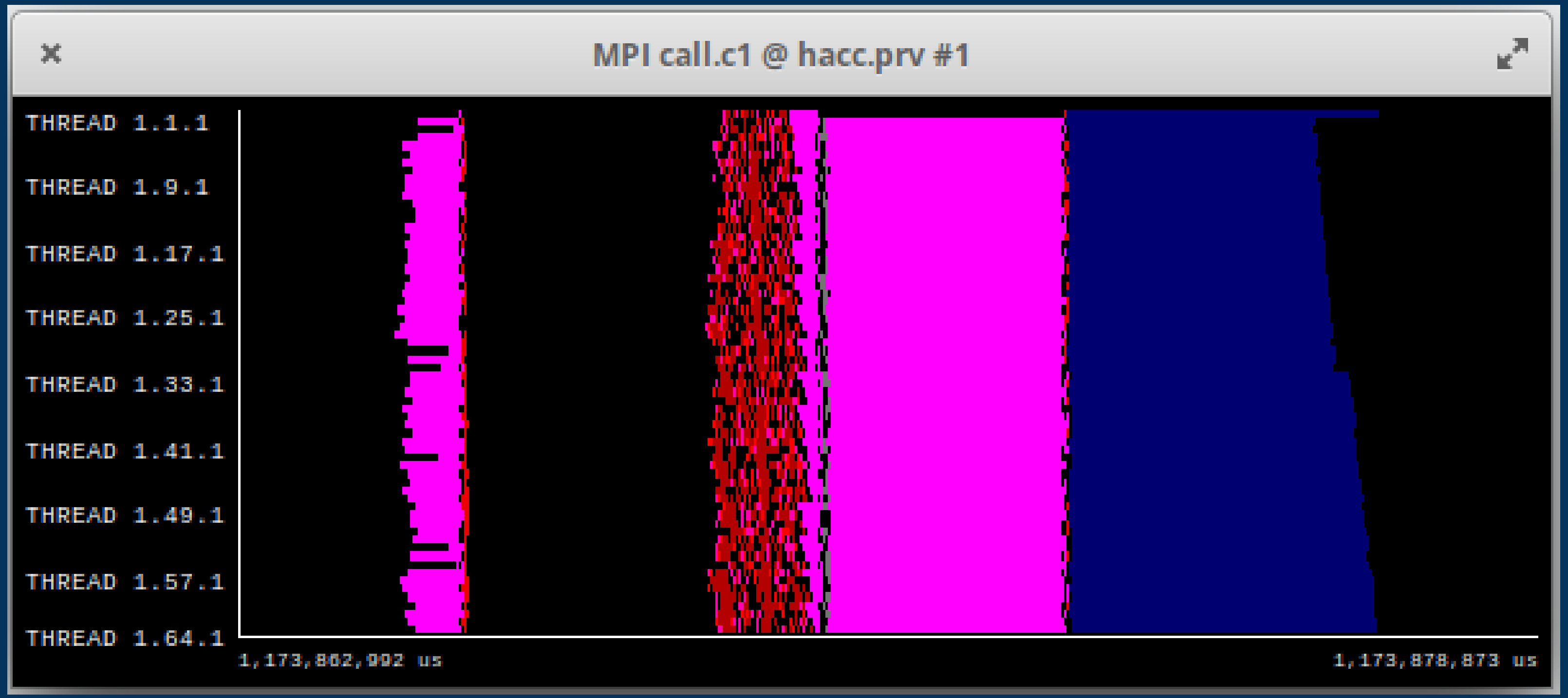


- Long MPI_Cart calls

- Much messaging among all the processes
  - Updating each process on global data

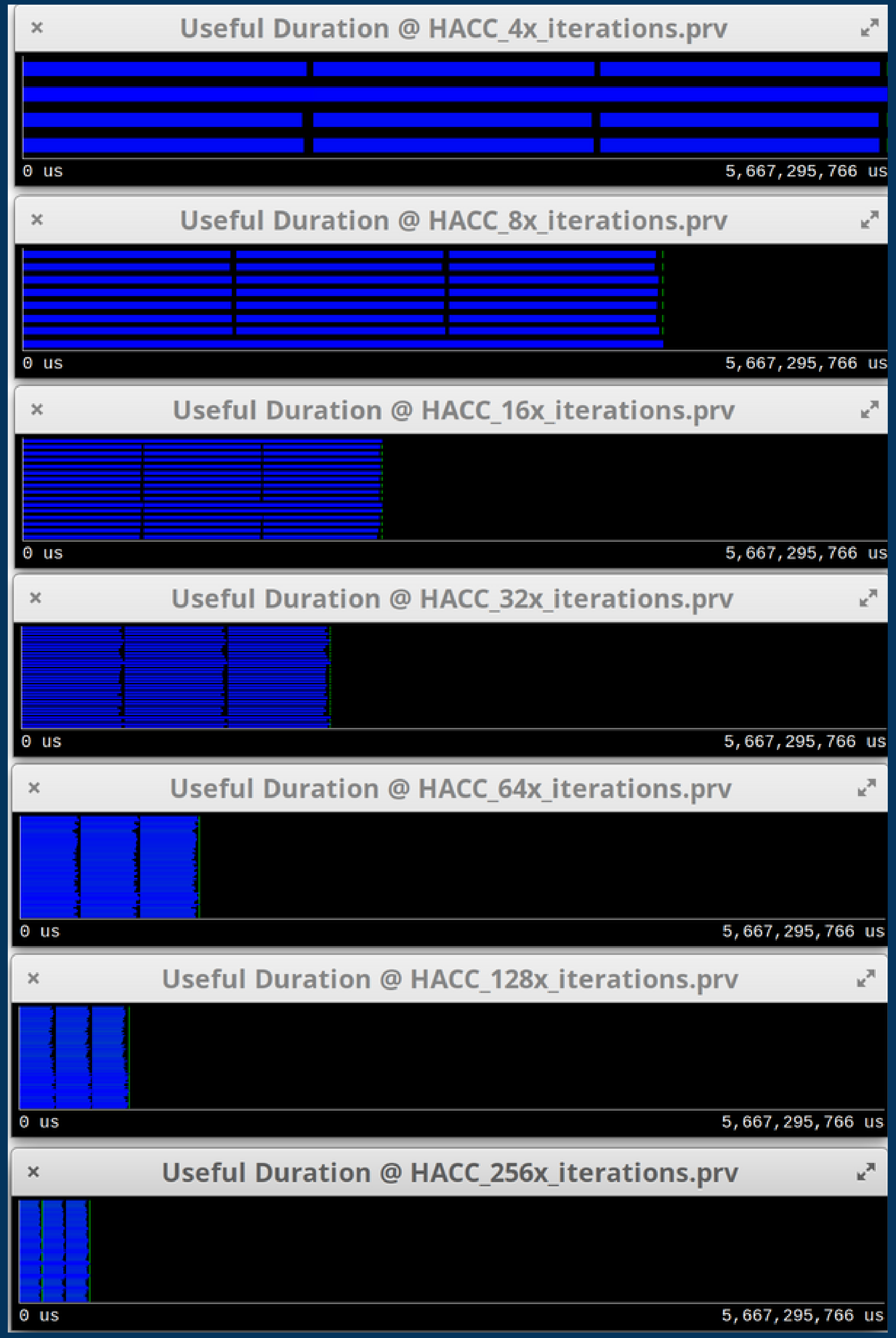
- 3 main data-exchange regions

Structure - Ending



- Final additional MPI_Allreduce
- MPI Finalize

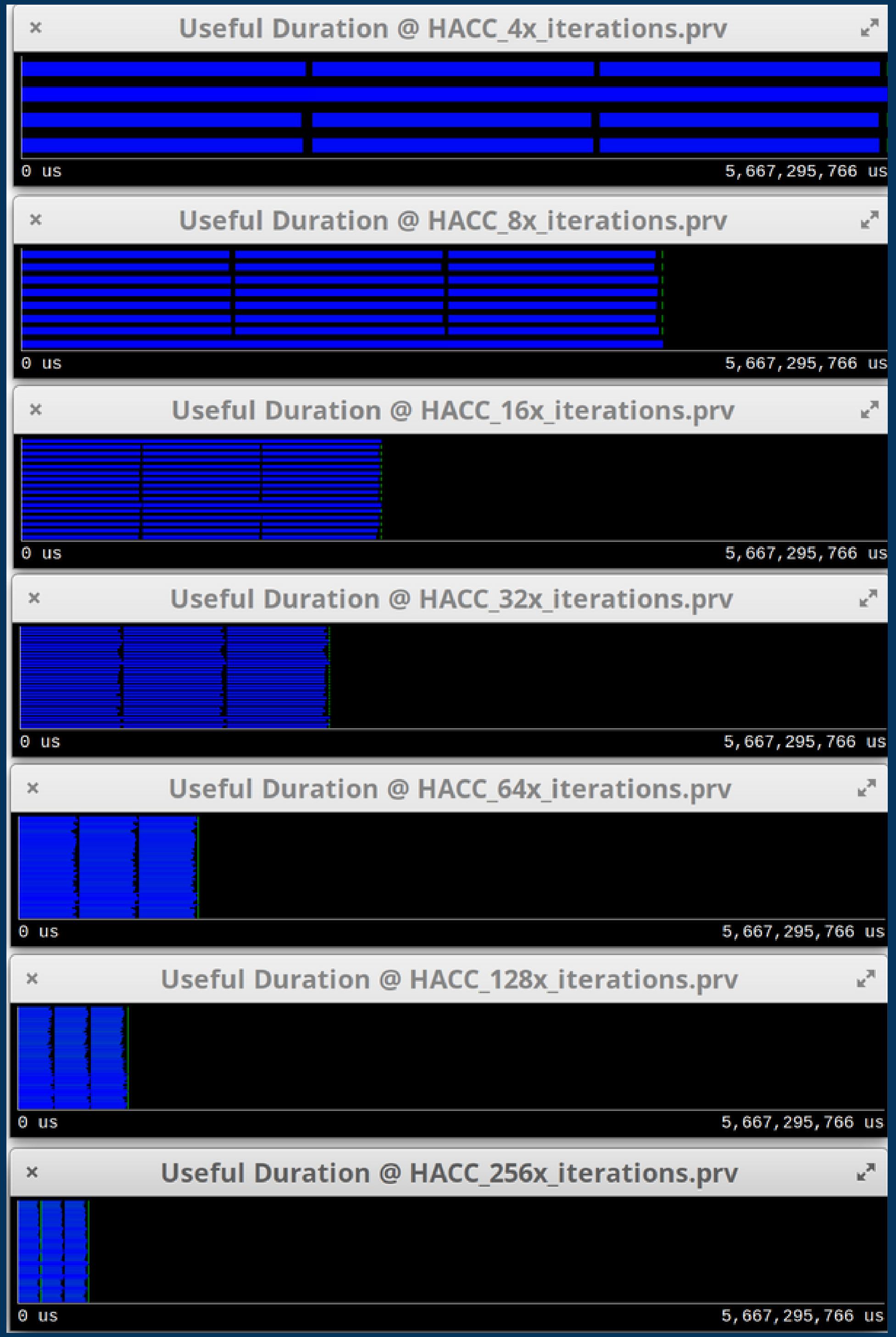
Strong Scaling analysis


Useful Duration @ HACC_4x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_8x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_16x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_32x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_64x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_128x_iterations.prv
0 us
5,667,295,766 us
Useful Duration @ HACC_256x_iterations.prv
0 us
5,667,295,766 us

4x	2x2x1
8x	2x2x2
16x	4x2x2
32x	4x4x2
64x	4x4x4
128x	8x4x4
256x	8x8x4

- Could not find an input good for each number of processes
- The one used in the end covered most executions, but it was
  - too large for 1x and 2x
  - too small for 512x
- Input file has:
  - np = ng = 64
  - Physical Box = 40

Strong Scaling - model factors


Useful Duration @ HACC_4x_iterations.prv
Useful Duration @ HACC_8x_iterations.prv
Useful Duration @ HACC_16x_iterations.prv
Useful Duration @ HACC_32x_iterations.prv
Useful Duration @ HACC_64x_iterations.prv
Useful Duration @ HACC_128x_iterations.prv
Useful Duration @ HACC_256x_iterations.prv
0 us
5,667,295,766 us

4x 2x2x1

8x 2x2x2

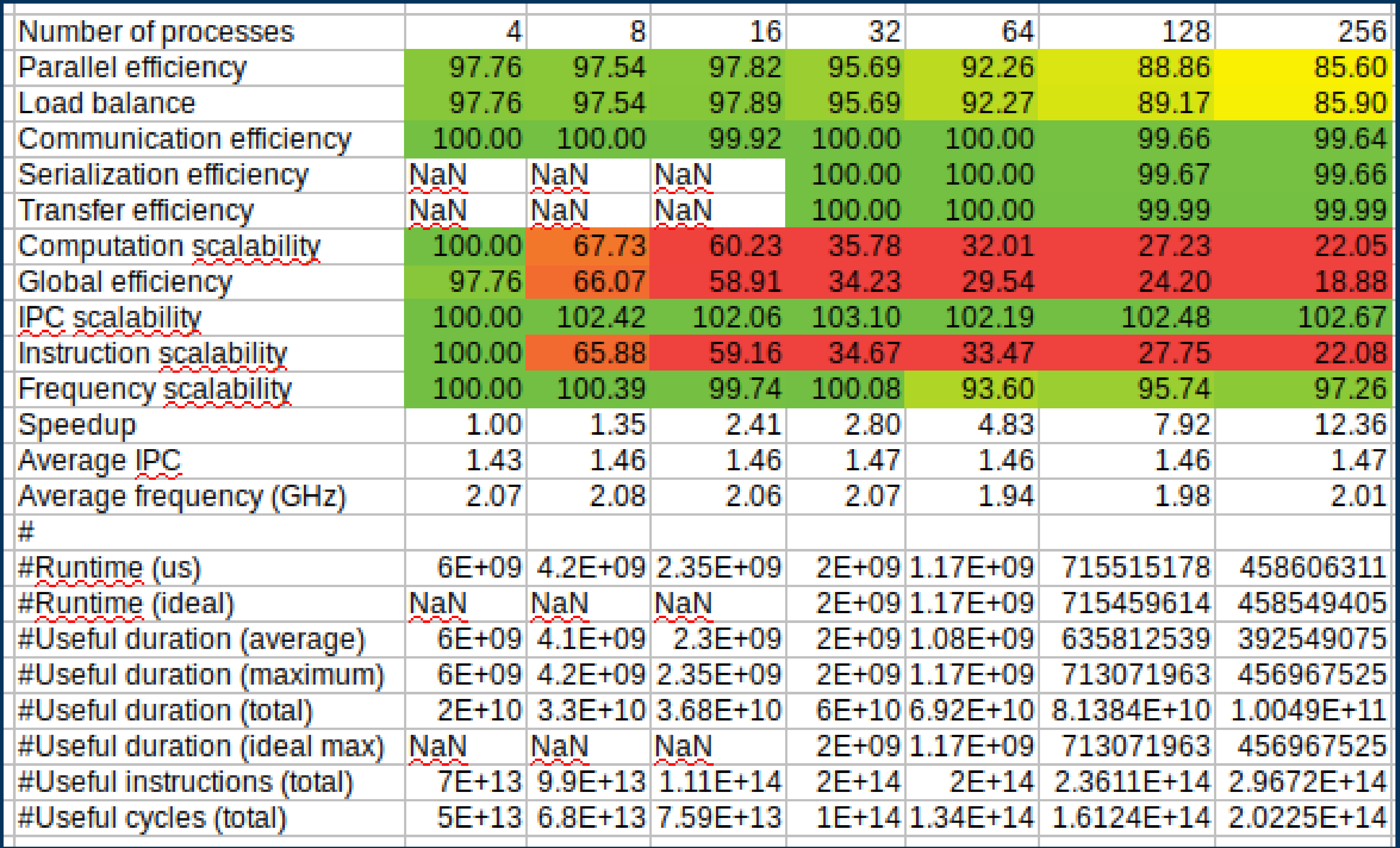
16x 4x2x2

32x 4x4x2

64x 4x4x4

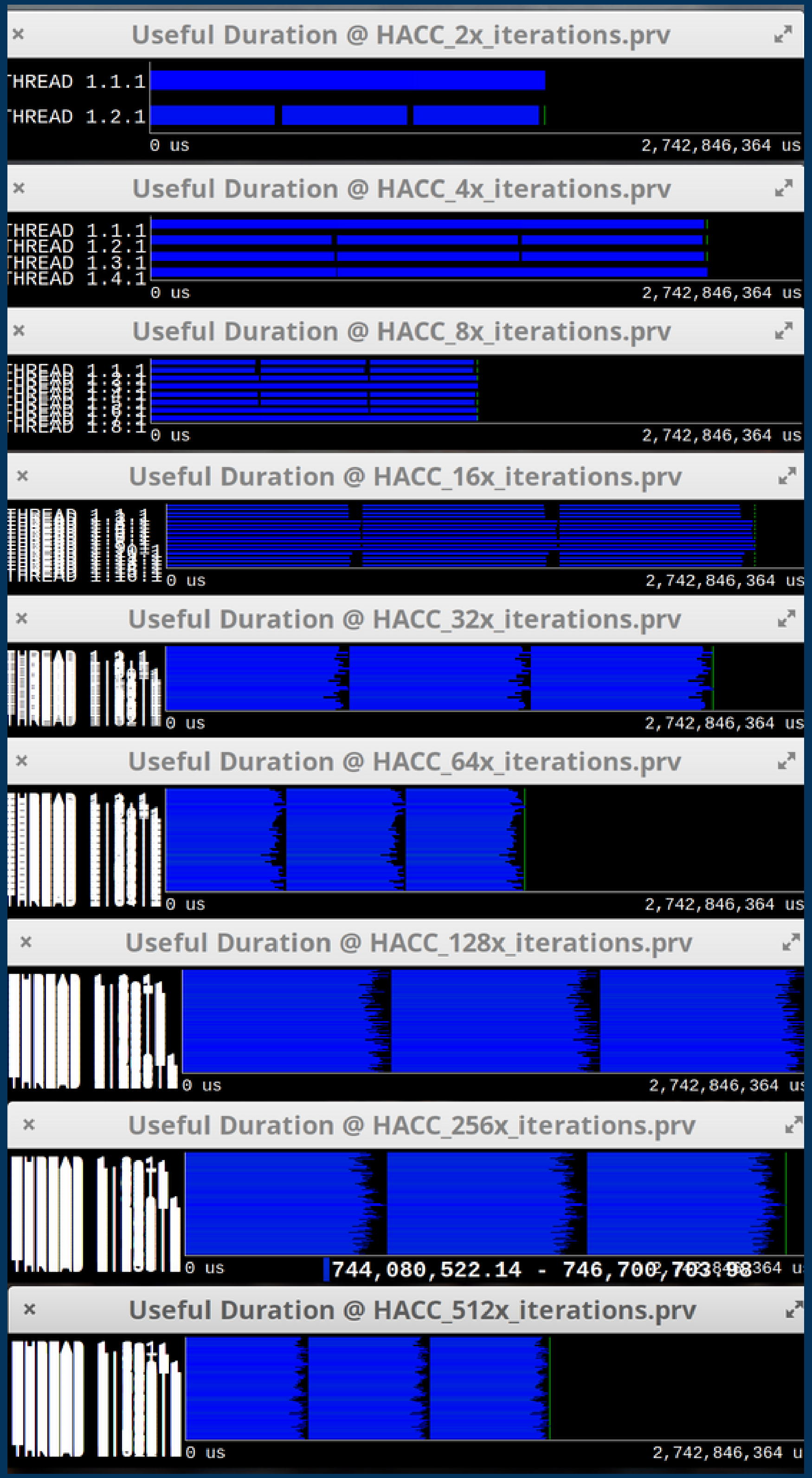
128x 8x4x4

256x 8x8x4



Number of processes	4	8	16	32	64	128	256
Parallel efficiency	97.76	97.54	97.82	95.69	92.26	88.86	85.60
Load balance	97.76	97.54	97.89	95.69	92.27	89.17	85.90
Communication efficiency	100.00	100.00	99.92	100.00	100.00	99.66	99.64
Serialization efficiency	NaN	NaN	NaN	100.00	100.00	99.67	99.66
Transfer efficiency	NaN	NaN	NaN	100.00	100.00	99.99	99.99
Computation scalability	100.00	67.73	60.23	35.78	32.01	27.23	22.05
Global efficiency	97.76	66.07	58.91	34.23	29.54	24.20	18.88
IPC scalability	100.00	102.42	102.06	103.10	102.19	102.48	102.67
Instruction scalability	100.00	65.88	59.16	34.67	33.47	27.75	22.08
Frequency scalability	100.00	100.39	99.74	100.08	93.60	95.74	97.26
Speedup	1.00	1.35	2.41	2.80	4.83	7.92	12.36
Average IPC	1.43	1.46	1.46	1.47	1.46	1.46	1.47
Average frequency (GHz)	2.07	2.08	2.06	2.07	1.94	1.98	2.01
#							
#Runtime (us)	6E+09	4.2E+09	2.35E+09	2E+09	1.17E+09	715515178	458606311
#Runtime (ideal)	NaN	NaN	NaN	2E+09	1.17E+09	715459614	458549405
#Useful duration (average)	6E+09	4.1E+09	2.3E+09	2E+09	1.08E+09	635812539	392549075
#Useful duration (maximum)	6E+09	4.2E+09	2.35E+09	2E+09	1.17E+09	713071963	456967525
#Useful duration (total)	2E+10	3.3E+10	3.68E+10	6E+10	6.92E+10	8.1384E+10	1.0049E+11
#Useful duration (ideal max)	NaN	NaN	NaN	2E+09	1.17E+09	713071963	456967525
#Useful instructions (total)	7E+13	9.9E+13	1.11E+14	2E+14	2E+14	2.3611E+14	2.9672E+14
#Useful cycles (total)	5E+13	6.8E+13	7.59E+13	1E+14	1.34E+14	1.6124E+14	2.0225E+14

Weak Scaling analysis


Useful Duration @ HACC_2x_iterations.prv
THREAD 1.1.1
THREAD 1.2.1
0 us
2,742,846,364 us
Useful Duration @ HACC_4x_iterations.prv
THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
0 us
2,742,846,364 us
Useful Duration @ HACC_8x_iterations.prv
0 us
2,742,846,364 us
Useful Duration @ HACC_16x_iterations.prv
0 us
2,742,846,364 us
Useful Duration @ HACC_32x_iterations.prv
0 us
2,742,846,364 us
Useful Duration @ HACC_64x_iterations.prv
0 us
2,742,846,364 us
Useful Duration @ HACC_128x_iterations.prv
0 us
2,742,846,364 us
Useful Duration @ HACC_256x_iterations.prv
0 us
744,080,522.14 - 746,700,703.98
Useful Duration @ HACC_512x_iterations.prv
0 us
2,742,846,364 us

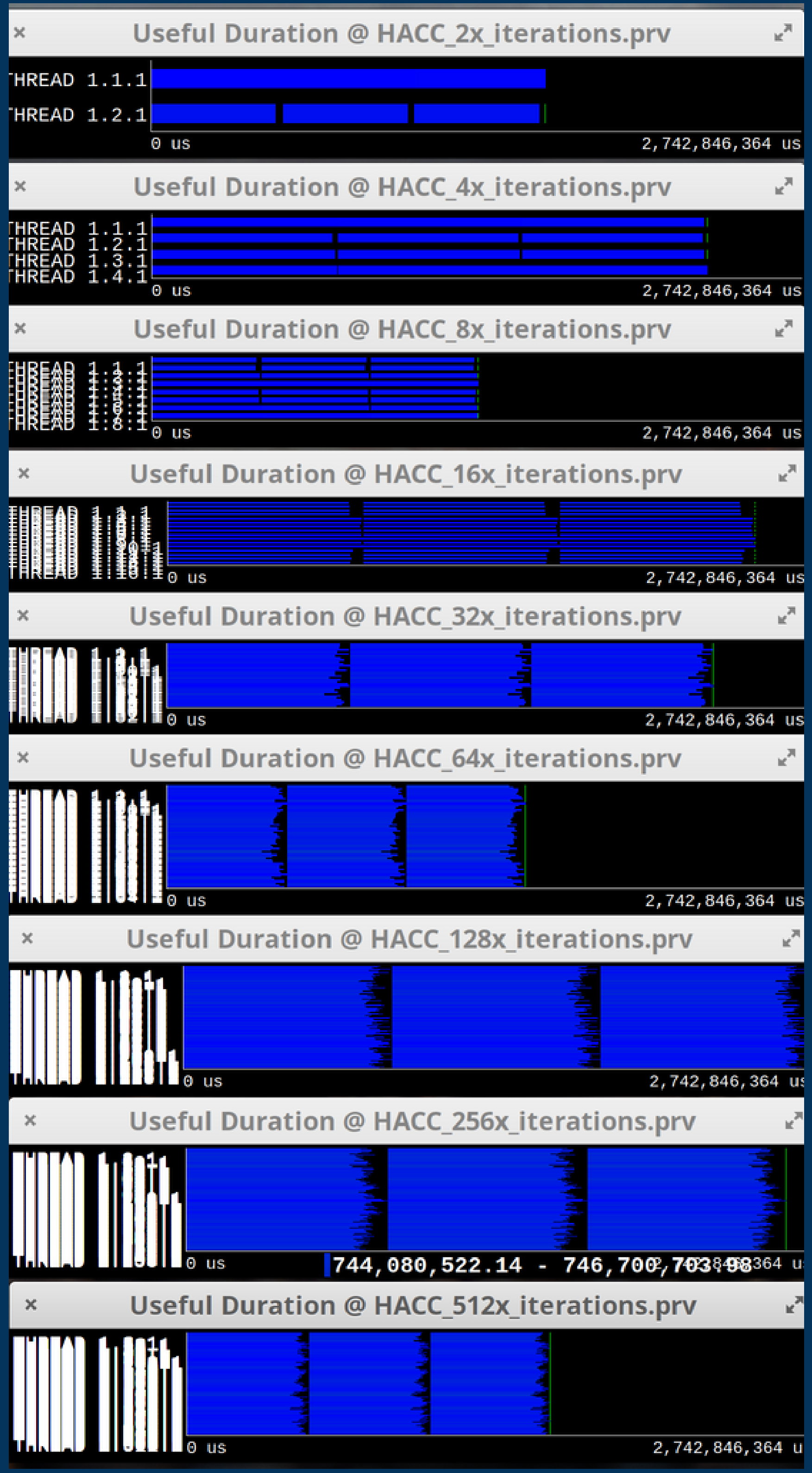
2x	2x1x1
4x	2x2x1
8x	2x2x2
16x	4x2x2
32x	4x4x2
64x	4x4x4
128x	8x4x4
256x	8x8x4
512x	8x8x8

- Could not generate first trace
  - Extrae indefinitely waiting for processes to end(?)
  - Time taken from SLURM job data

- Input files sizes started from:
  - np = ng = **24**
  - Physical Box = **20**
- ...arriving to:
  - np = ng = **192**
  - Physical Box = **160**

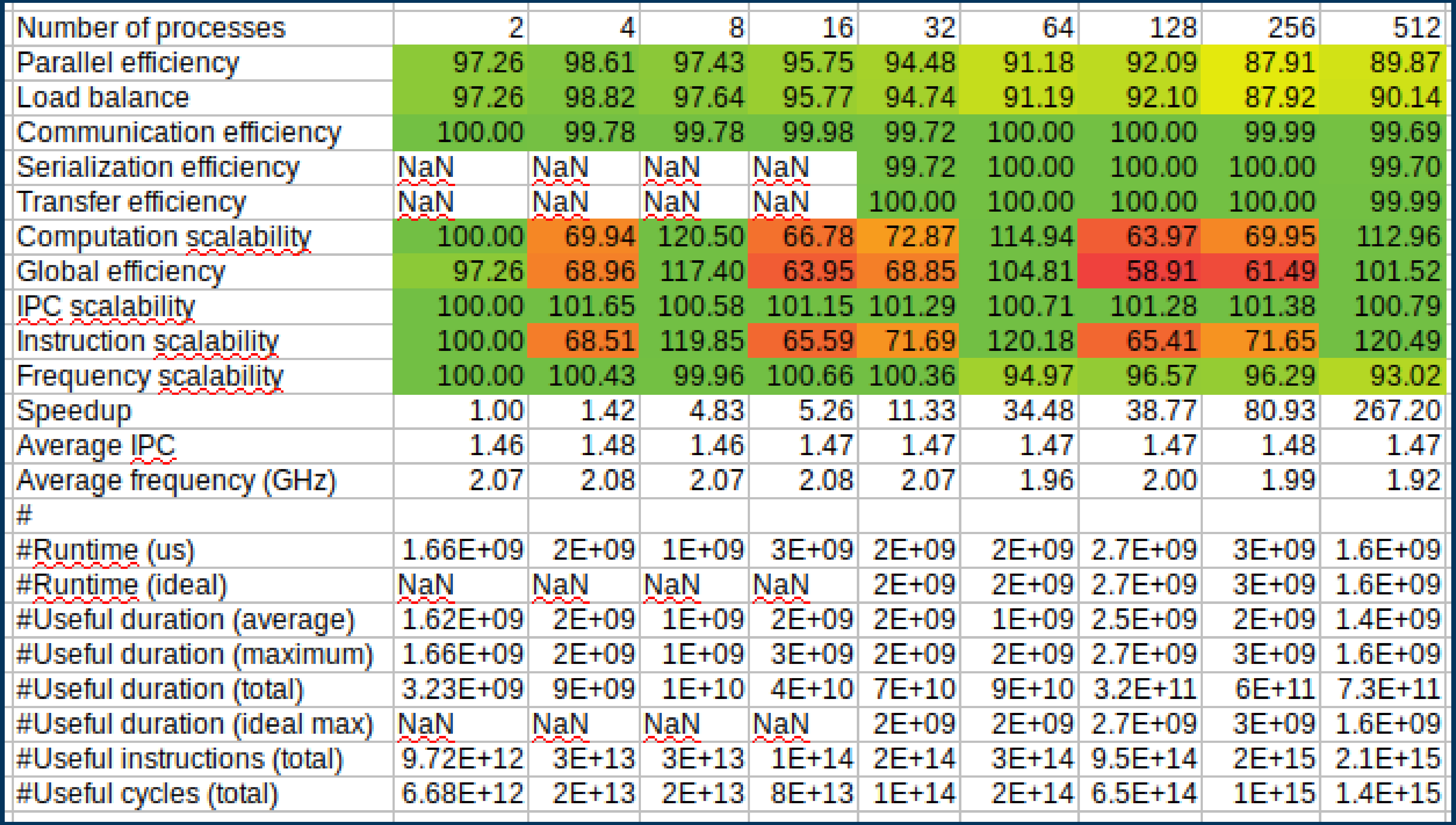
Weak Scaling - model factors

2x
4x
8x
16x
32x
64x
128x
256x
512x



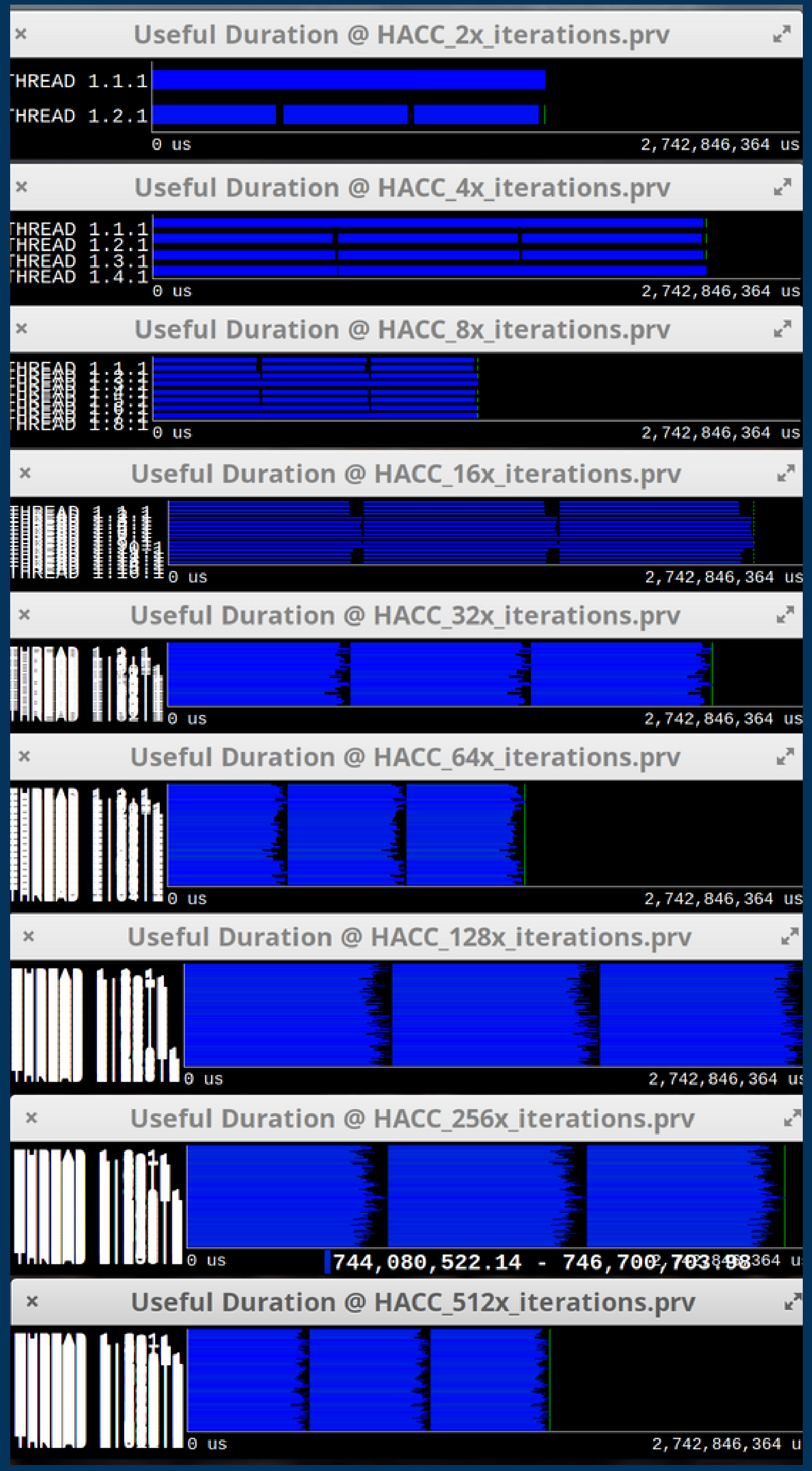
Useful Duration @ HACC_2x_iterations.prv
THREAD 1.1.1
THREAD 1.2.1
0 us
2,742,846,364 us
Useful Duration @ HACC_4x_iterations.prv
THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
Useful Duration @ HACC_8x_iterations.prv
Useful Duration @ HACC_16x_iterations.prv
Useful Duration @ HACC_32x_iterations.prv
Useful Duration @ HACC_64x_iterations.prv
Useful Duration @ HACC_128x_iterations.prv
Useful Duration @ HACC_256x_iterations.prv
744,080,522.14 - 746,700,703.98
Useful Duration @ HACC_512x_iterations.prv

2x1x1
2x2x1
2x2x2
4x2x2
4x4x2
4x4x4
8x4x4
8x8x4
8x8x8



Number of processes	2	4	8	16	32	64	128	256	512
Parallel efficiency	97.26	98.61	97.43	95.75	94.48	91.18	92.09	87.91	89.87
Load balance	97.26	98.82	97.64	95.77	94.74	91.19	92.10	87.92	90.14
Communication efficiency	100.00	99.78	99.78	99.98	99.72	100.00	100.00	99.99	99.69
Serialization efficiency	NaN	NaN	NaN	NaN	99.72	100.00	100.00	100.00	99.70
Transfer efficiency	NaN	NaN	NaN	NaN	100.00	100.00	100.00	100.00	99.99
Computation scalability	100.00	69.94	120.50	66.78	72.87	114.94	63.97	69.95	112.96
Global efficiency	97.26	68.96	117.40	63.95	68.85	104.81	58.91	61.49	101.52
IPC scalability	100.00	101.65	100.58	101.15	101.29	100.71	101.28	101.38	100.79
Instruction scalability	100.00	68.51	119.85	65.59	71.69	120.18	65.41	71.65	120.49
Frequency scalability	100.00	100.43	99.96	100.66	100.36	94.97	96.57	96.29	93.02
Speedup	1.00	1.42	4.83	5.26	11.33	34.48	38.77	80.93	267.20
Average IPC	1.46	1.48	1.46	1.47	1.47	1.47	1.47	1.48	1.47
Average frequency (GHz)	2.07	2.08	2.07	2.08	2.07	1.96	2.00	1.99	1.92
#									
#Runtime (us)	1.66E+09	2E+09	1E+09	3E+09	2E+09	2E+09	2.7E+09	3E+09	1.6E+09
#Runtime (ideal)	NaN	NaN	NaN	NaN	2E+09	2E+09	2.7E+09	3E+09	1.6E+09
#Useful duration (average)	1.62E+09	2E+09	1E+09	2E+09	2E+09	1E+09	2.5E+09	2E+09	1.4E+09
#Useful duration (maximum)	1.66E+09	2E+09	1E+09	3E+09	2E+09	2E+09	2.7E+09	3E+09	1.6E+09
#Useful duration (total)	3.23E+09	9E+09	1E+10	4E+10	7E+10	9E+10	3.2E+11	6E+11	7.3E+11
#Useful duration (ideal max)	NaN	NaN	NaN	NaN	2E+09	2E+09	2.7E+09	3E+09	1.6E+09
#Useful instructions (total)	9.72E+12	3E+13	3E+13	1E+14	2E+14	3E+14	9.5E+14	2E+15	2.1E+15
#Useful cycles (total)	6.68E+12	2E+13	2E+13	8E+13	1E+14	2E+14	6.5E+14	1E+15	1.4E+15

Weak Scaling - efficiency


Useful Duration @ HACC_2x_iterations.prv
Useful Duration @ HACC_4x_iterations.prv
Useful Duration @ HACC_8x_iterations.prv
Useful Duration @ HACC_16x_iterations.prv
Useful Duration @ HACC_32x_iterations.prv
Useful Duration @ HACC_64x_iterations.prv
Useful Duration @ HACC_128x_iterations.prv
Useful Duration @ HACC_256x_iterations.prv
Useful Duration @ HACC_512x_iterations.prv
THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
0 us
2,742,846,364 us
744,080,522.14 - 746,700,703.98

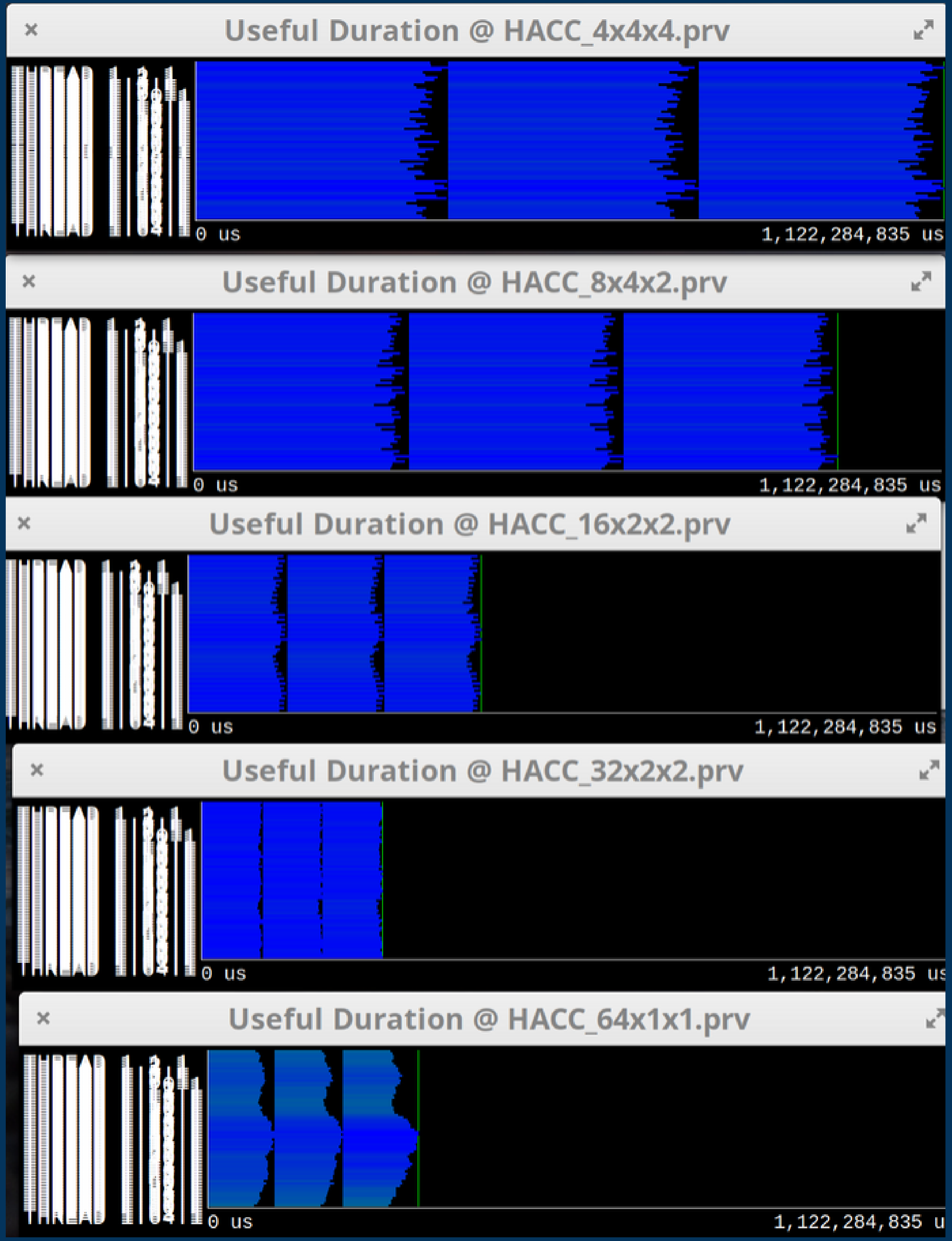
2x	2x1x1 →	0.82
4x	2x2x1 →	0.58
8x	**2x2x2** →	**0.99**
16x	4x2x2 →	0.54
32x	4x4x2 →	0.58
64x	**4x4x4** →	**0.88**
128x	8x4x4 →	0.49
256x	8x8x4 →	0.52
512x	**8x8x8** →	**0.85**

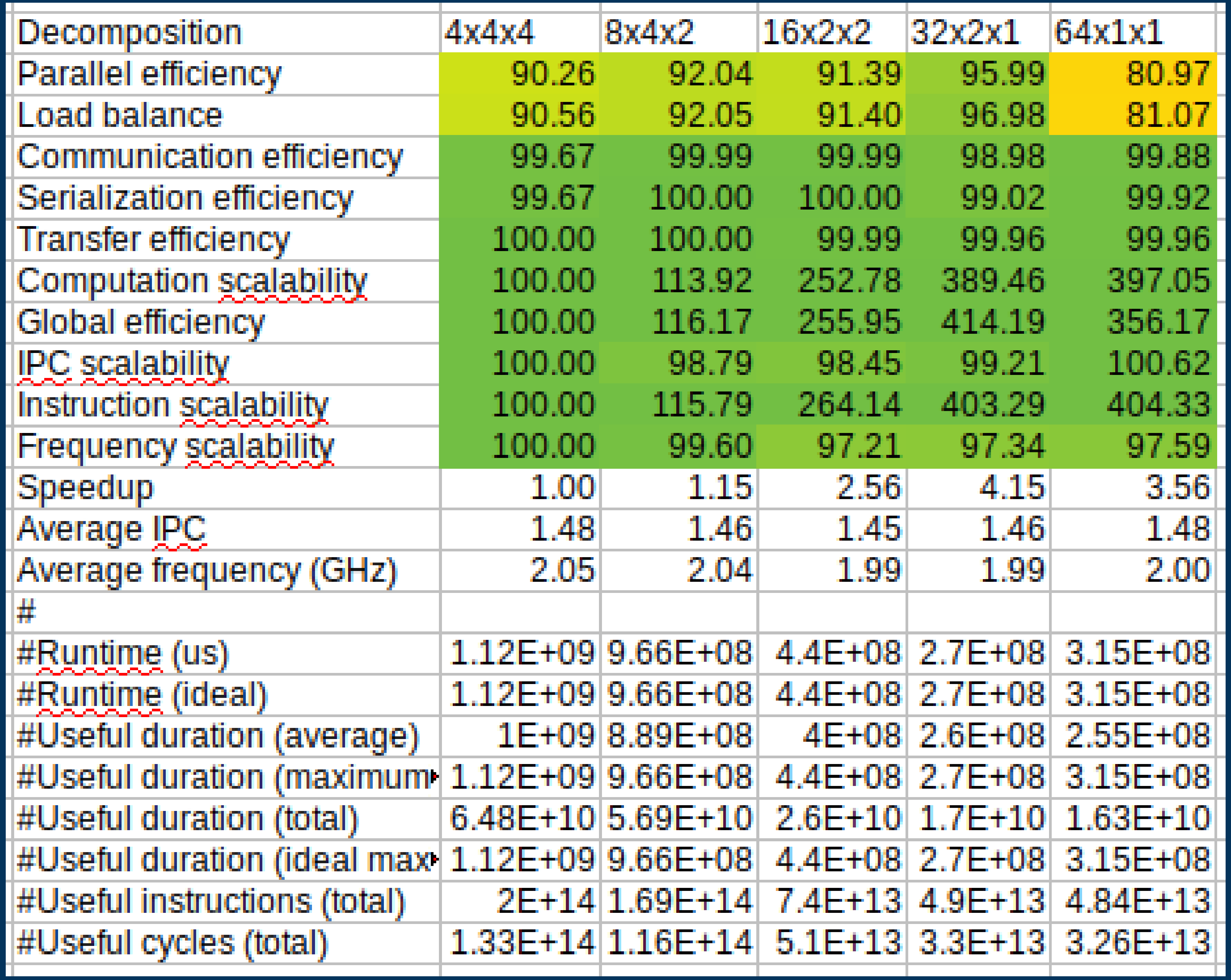
<u>Perfect cubic decomposition!</u>

↓

According to HACC documentation

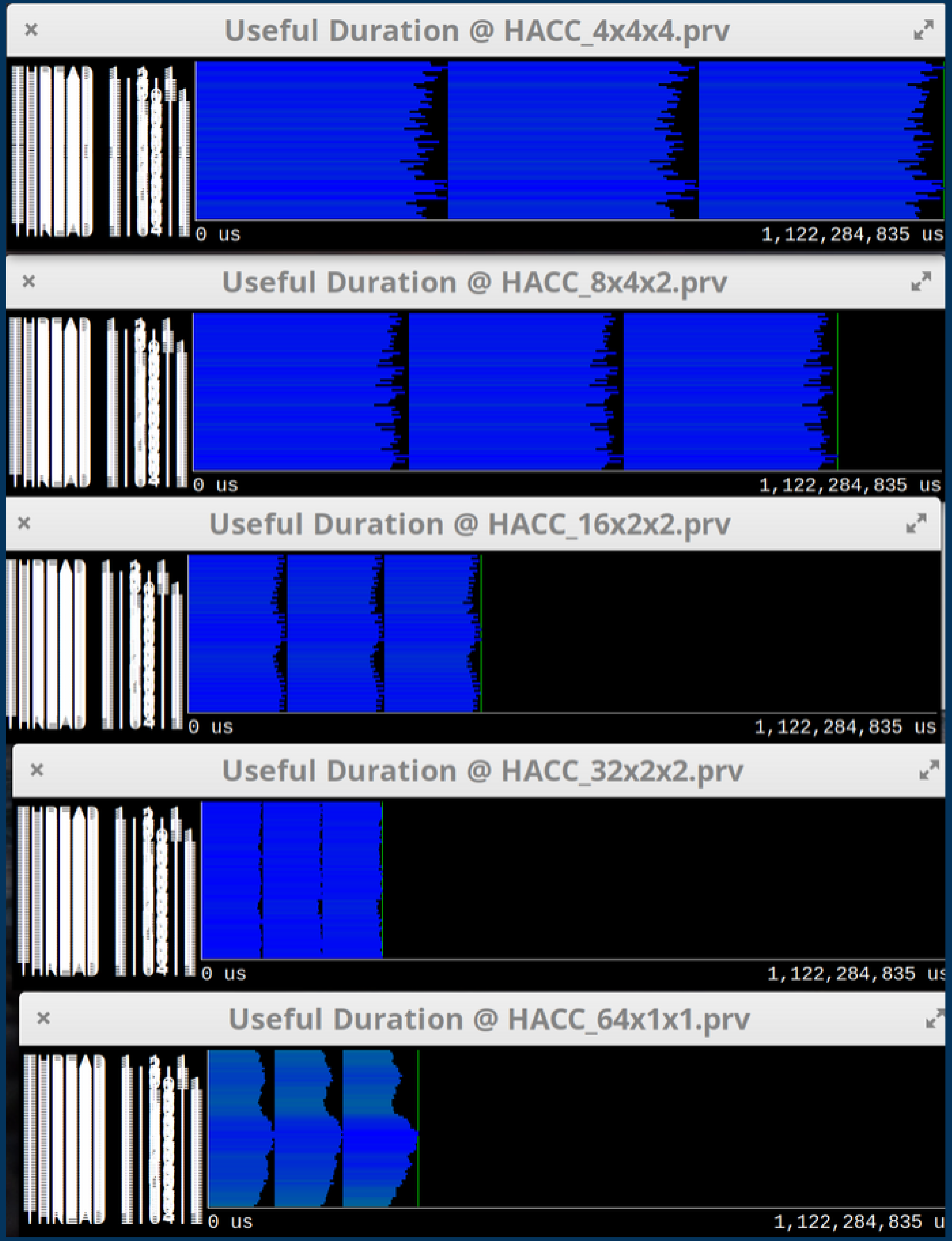
Decomposition analysis


4x4x4
Useful Duration @ HACC_4x4x4.prv
0 us
1,122,284,835 us
8x4x2
Useful Duration @ HACC_8x4x2.prv
0 us
1,122,284,835 us
16x2x2
Useful Duration @ HACC_16x2x2.prv
0 us
1,122,284,835 us
32x2x1
Useful Duration @ HACC_32x2x2.prv
0 us
1,122,284,835 us
64x1x1
Useful Duration @ HACC_64x1x1.prv
0 us
1,122,284,835 u



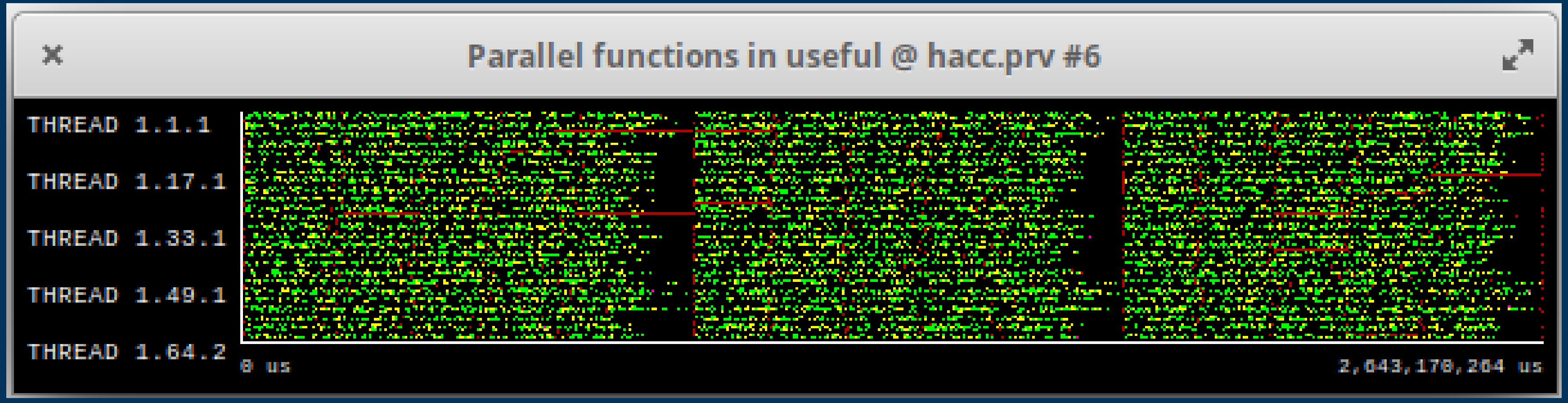
Decomposition	4x4x4	8x4x2	16x2x2	32x2x1	64x1x1
Parallel efficiency	90.26	92.04	91.39	95.99	80.97
Load balance	90.56	92.05	91.40	96.98	81.07
Communication efficiency	99.67	99.99	99.99	98.98	99.88
Serialization efficiency	99.67	100.00	100.00	99.02	99.92
Transfer efficiency	100.00	100.00	99.99	99.96	99.96
Computation scalability	100.00	113.92	252.78	389.46	397.05
Global efficiency	100.00	116.17	255.95	414.19	356.17
IPC scalability	100.00	98.79	98.45	99.21	100.62
Instruction scalability	100.00	115.79	264.14	403.29	404.33
Frequency scalability	100.00	99.60	97.21	97.34	97.59
Speedup	1.00	1.15	2.56	4.15	3.56
Average IPC	1.48	1.46	1.45	1.46	1.48
Average frequency (GHz)	2.05	2.04	1.99	1.99	2.00
#					
#Runtime (us)	1.12E+09	9.66E+08	4.4E+08	2.7E+08	3.15E+08
#Runtime (ideal)	1.12E+09	9.66E+08	4.4E+08	2.7E+08	3.15E+08
#Useful duration (average)	1E+09	8.89E+08	4E+08	2.6E+08	2.55E+08
#Useful duration (maximum	1.12E+09	9.66E+08	4.4E+08	2.7E+08	3.15E+08
#Useful duration (total)	6.48E+10	5.69E+10	2.6E+10	1.7E+10	1.63E+10
#Useful duration (ideal max	1.12E+09	9.66E+08	4.4E+08	2.7E+08	3.15E+08
#Useful instructions (total)	2E+14	1.69E+14	7.4E+13	4.9E+13	4.84E+13
#Useful cycles (total)	1.33E+14	1.16E+14	5.1E+13	3.3E+13	3.26E+13

Decomposition analysis (cont.)


4x4x4
Useful Duration @ HACC_4x4x4.prv
0 us
1,122,284,835 us
8x4x2
Useful Duration @ HACC_8x4x2.prv
0 us
1,122,284,835 us
16x2x2
Useful Duration @ HACC_16x2x2.prv
0 us
1,122,284,835 us
32x2x1
Useful Duration @ HACC_32x2x2.prv
0 us
1,122,284,835 us
64x1x1
Useful Duration @ HACC_64x1x1.prv
0 us
1,122,284,835 u

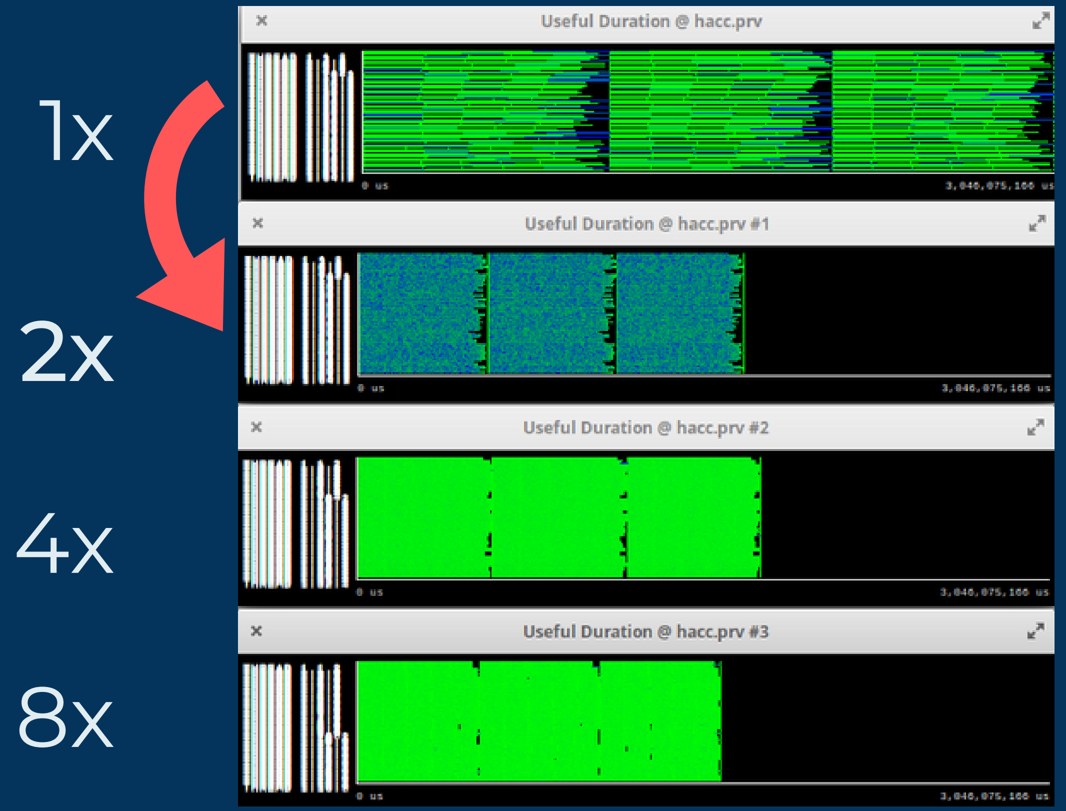
- Input file was:
  - np = ng = **64**
  - Physical Box = **50**
- Maybe because of a better distribution on a single dimension?
- ...or maybe on 2D (32x2x1)?

OpenMP Scaling Analysis



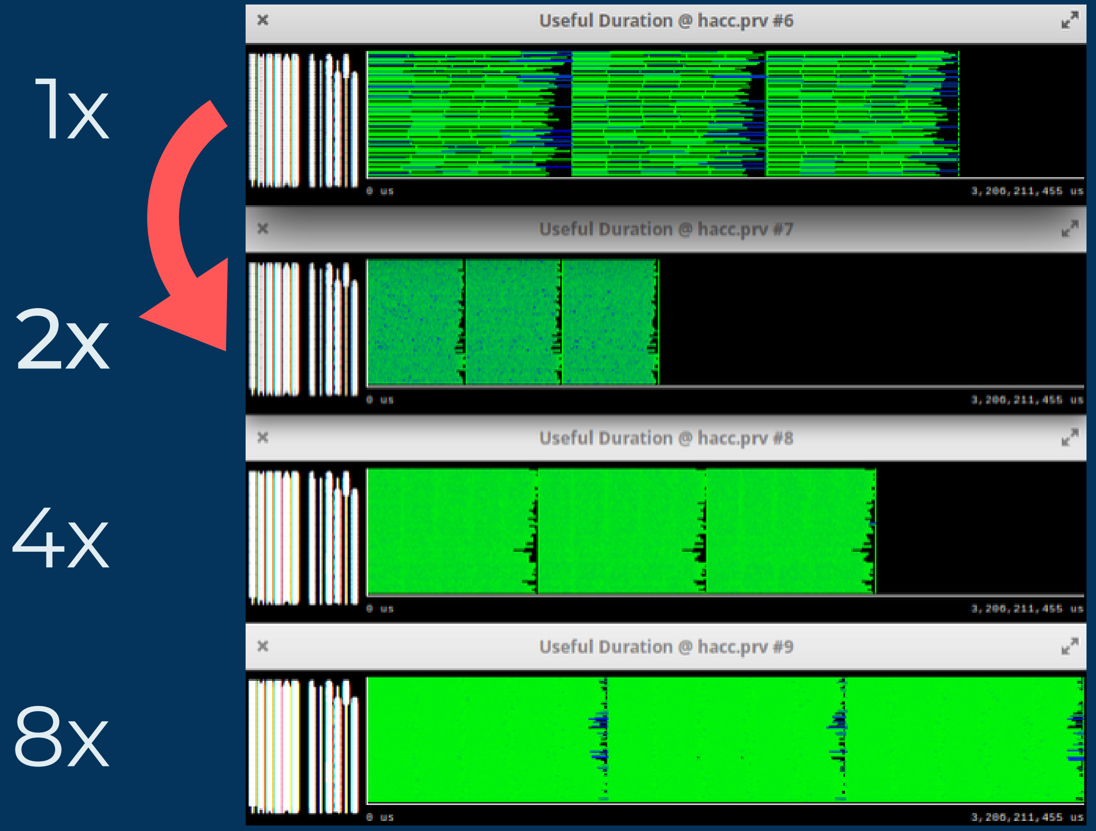
→ Iterations' inner computations benefit of OpenMP parallelization

Strong Scaling


Useful Duration @ hacc.prv
Useful Duration @ hacc.prv #1
Useful Duration @ hacc.prv #2
Useful Duration @ hacc.prv #3
1x
2x
4x
8x

2.14x speedup!

Weak Scaling


Useful Duration @ hacc.prv #6
Useful Duration @ hacc.prv #7
Useful Duration @ hacc.prv #8
Useful Duration @ hacc.prv #9
1x
2x
4x
8x

1.93 of efficiency!

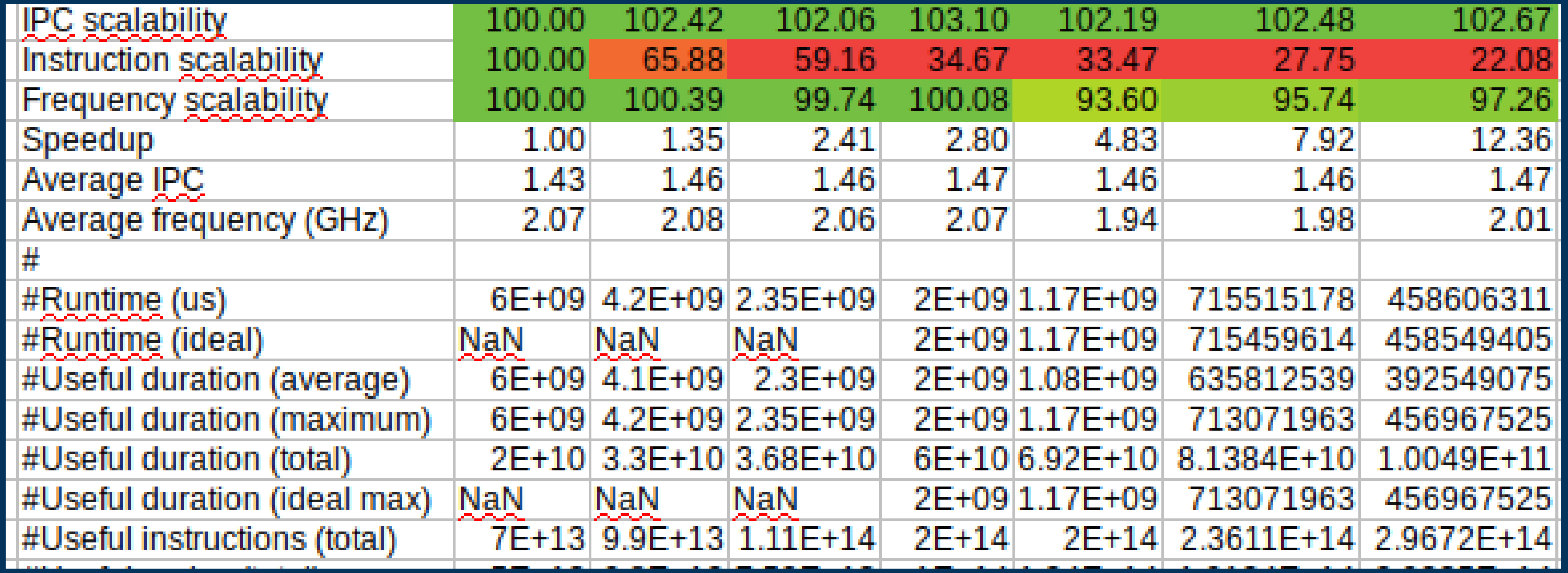
Part 4

Further Discussions

Where is it improvable?

Poor instruction scalability

Recalling the strong scaling analysis...

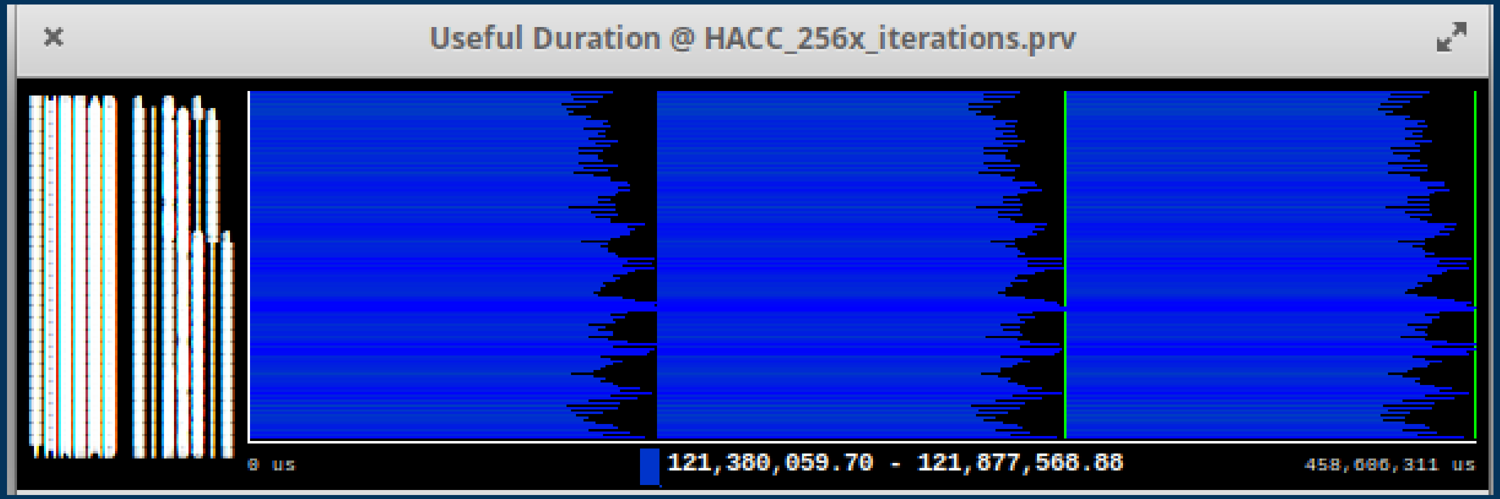


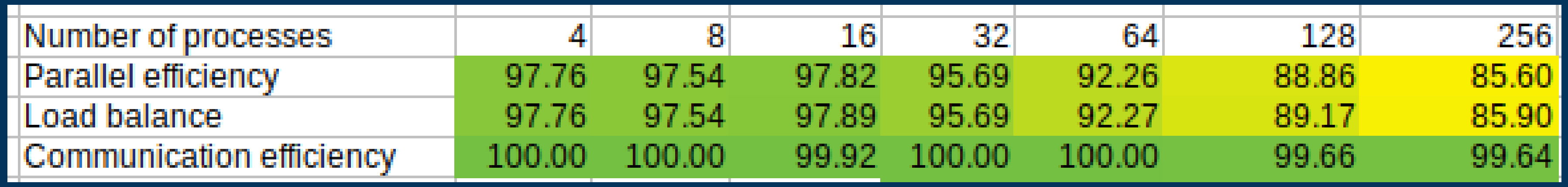
IPC scalability	100.00	102.42	102.06	103.10	102.19	102.48	102.67
Instruction scalability	100.00	65.88	59.16	34.67	33.47	27.75	22.08
Frequency scalability	100.00	100.39	99.74	100.08	93.60	95.74	97.26
Speedup	1.00	1.35	2.41	2.80	4.83	7.92	12.36
Average IPC	1.43	1.46	1.46	1.47	1.46	1.46	1.47
Average frequency (GHz)	2.07	2.08	2.06	2.07	1.94	1.98	2.01
#							
#Runtime (us)	6E+09	4.2E+09	2.35E+09	2E+09	1.17E+09	715515178	458606311
#Runtime (ideal)	NaN	NaN	NaN	2E+09	1.17E+09	715459614	458549405
#Useful duration (average)	6E+09	4.1E+09	2.3E+09	2E+09	1.08E+09	635812539	392549075
#Useful duration (maximum)	6E+09	4.2E+09	2.35E+09	2E+09	1.17E+09	713071963	456967525
#Useful duration (total)	2E+10	3.3E+10	3.68E+10	6E+10	6.92E+10	8.1384E+10	1.0049E+11
#Useful duration (ideal max)	NaN	NaN	NaN	2E+09	1.17E+09	713071963	456967525
#Useful instructions (total)	7E+13	9.9E+13	1.11E+14	2E+14	2E+14	2.3611E+14	2.9672E+14

- Probably, there are many synchronization instructions
- Additionally, code might be replicated

Load imbalance

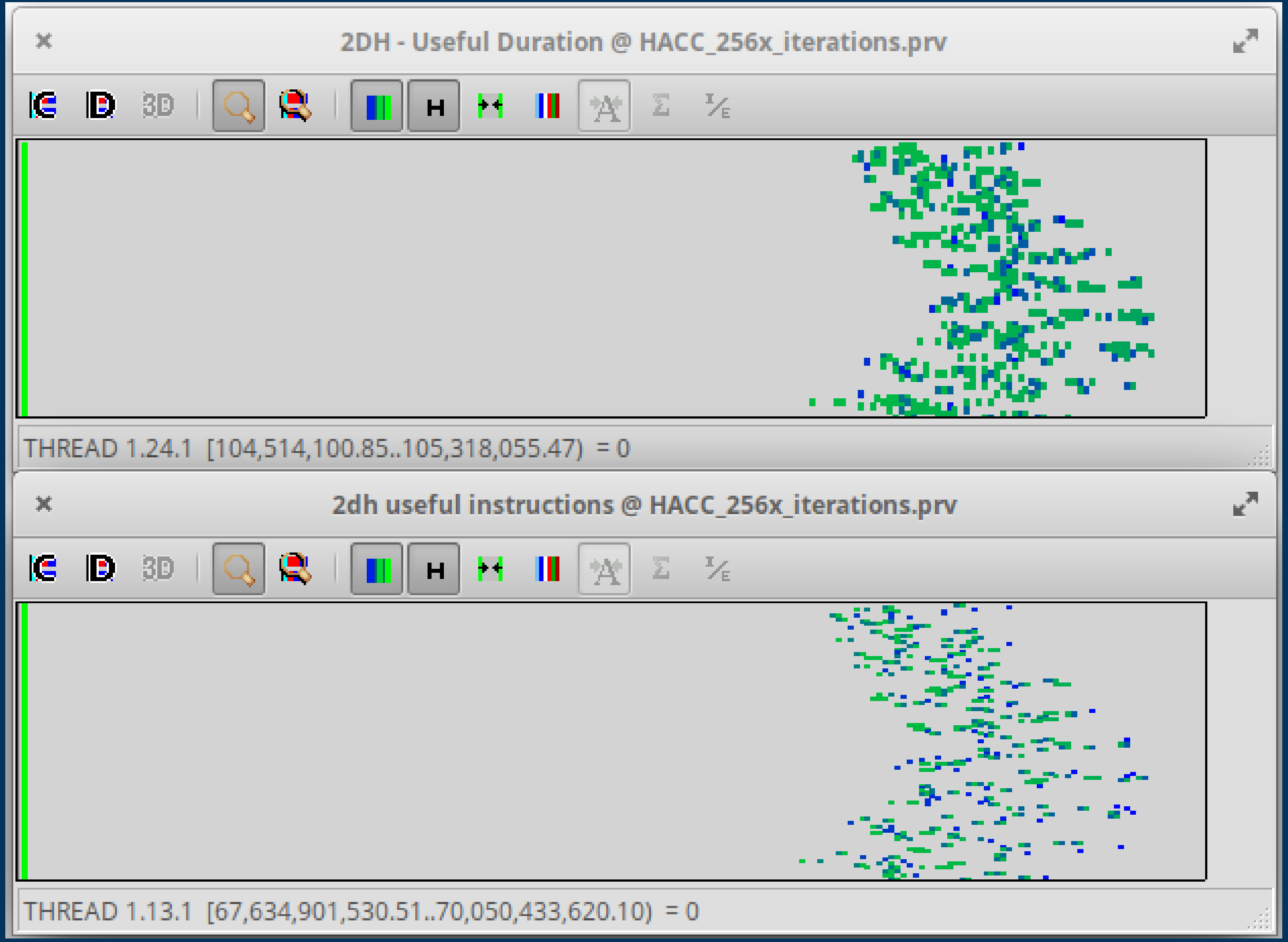
Recalling the strong scaling analysis...





Number of processes	4	8	16	32	64	128	256
Parallel efficiency	97.76	97.54	97.82	95.69	92.26	88.86	85.60
Load balance	97.76	97.54	97.89	95.69	92.27	89.17	85.90
Communication efficiency	100.00	100.00	99.92	100.00	100.00	99.66	99.64

Load imbalance (cont.)

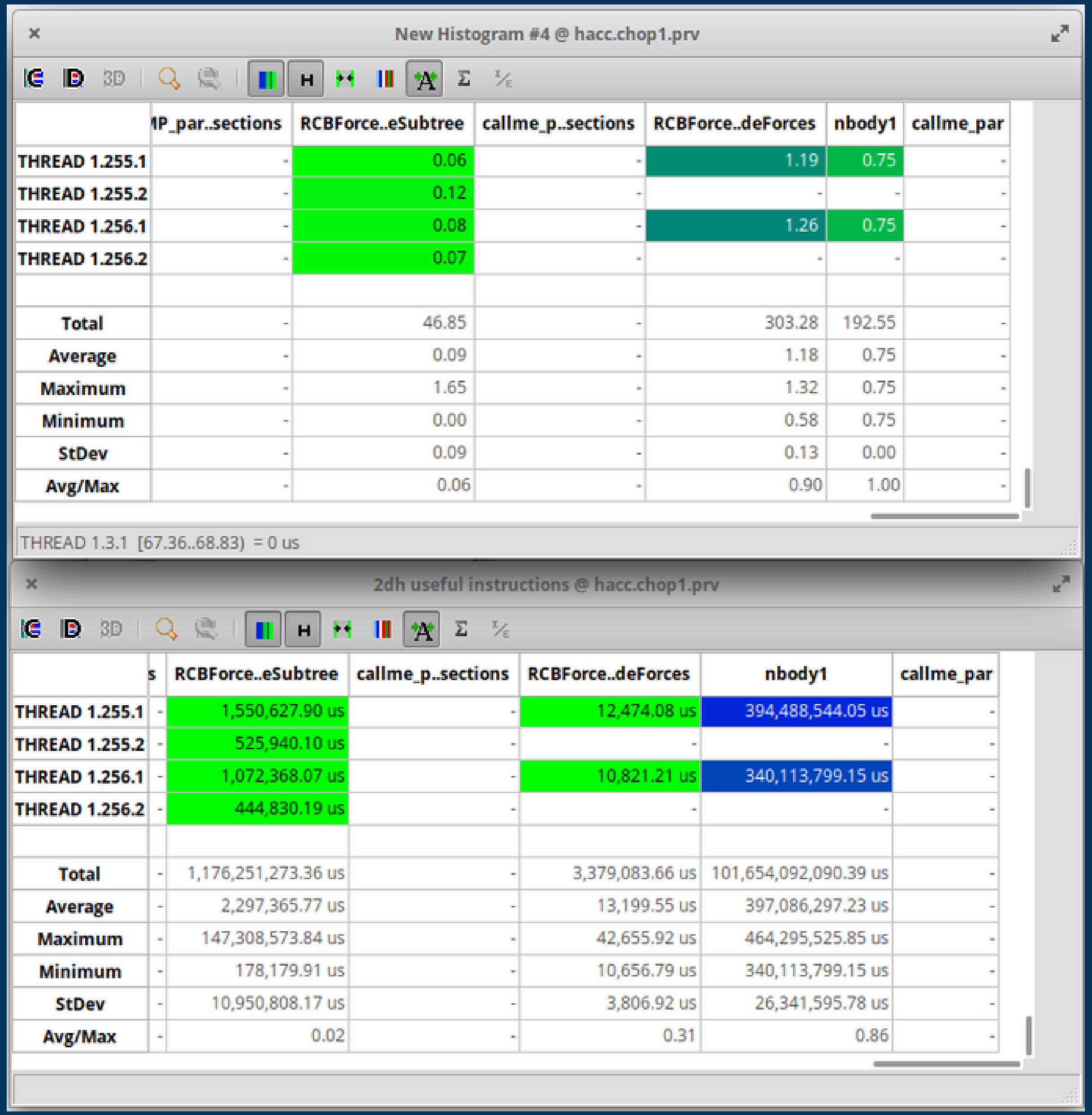

2DH - Useful Duration @ HACC_256x_iterations.prv
2dh useful instructions @ HACC_256x_iterations.prv

THREAD 1.24.1 [104,514,100.85..105,318,055.47) = 0

THREAD 1.13.1 [67,634,901,530.51..70,050,433,620.10) = 0

- Useful duration correlated to useful instructions
- Dynamic load balance could be a possible improvement

Low IPC



New Histogram #4 @ hacc.chop1.prv

	IP_par..sections	RCBForce..eSubtree	callme_p..sections	RCBForce..deForces	nbody1	callme_par
THREAD 1.255.1	-	0.06	-	1.19	0.75	-
THREAD 1.255.2	-	0.12	-	-	-	-
THREAD 1.256.1	-	0.08	-	1.26	0.75	-
THREAD 1.256.2	-	0.07	-	-	-	-
Total	-	46.85	-	303.28	192.55	-
Average	-	0.09	-	1.18	0.75	-
Maximum	-	1.65	-	1.32	0.75	-
Minimum	-	0.00	-	0.58	0.75	-
StDev	-	0.09	-	0.13	0.00	-
Avg/Max	-	0.06	-	0.90	1.00	-

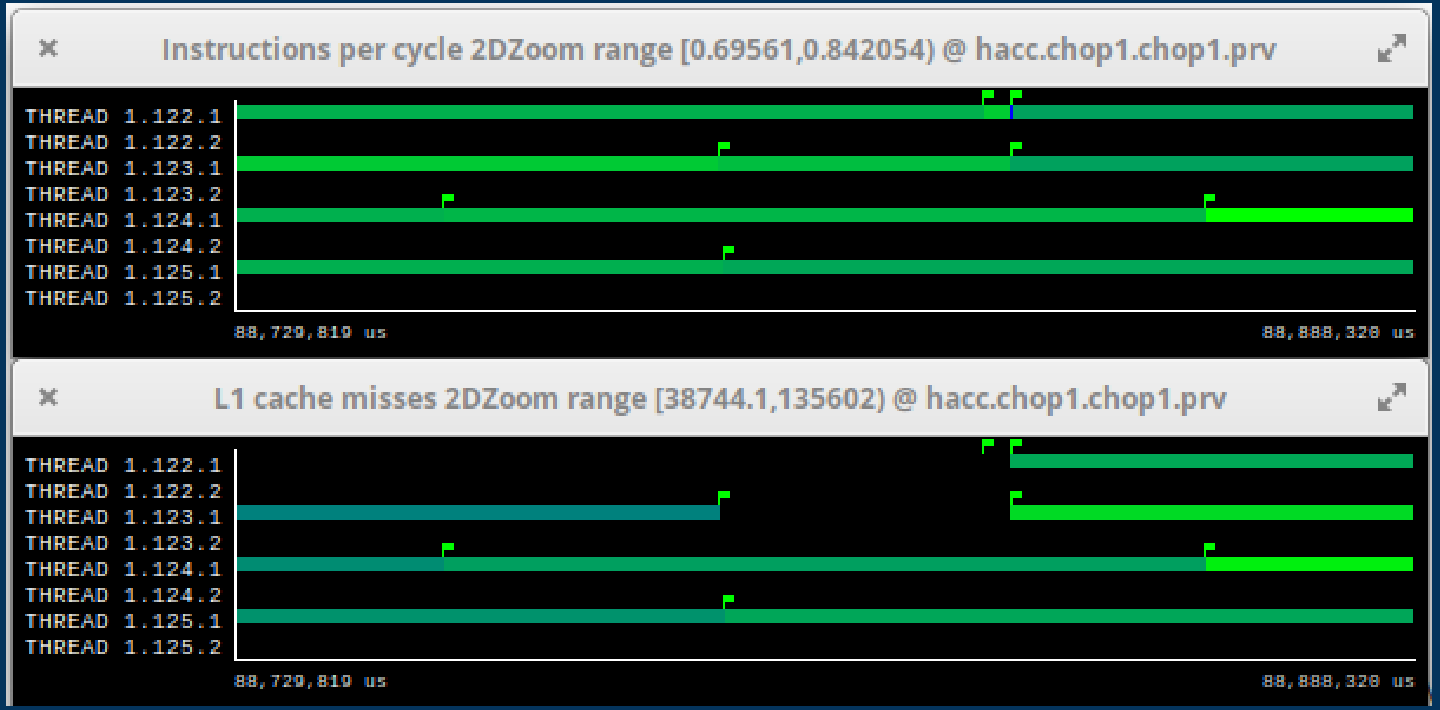
THREAD 1.3.1 [67.36..68.83) = 0 us

2dh useful instructions @ hacc.chop1.prv

	s	RCBForce..eSubtree	callme_p..sections	RCBForce..deForces	nbody1	callme_par
THREAD 1.255.1	-	1,550,627.90 us	-	12,474.08 us	394,488,544.05 us	-
THREAD 1.255.2	-	525,940.10 us	-	-	-	-
THREAD 1.256.1	-	1,072,368.07 us	-	10,821.21 us	340,113,799.15 us	-
THREAD 1.256.2	-	444,830.19 us	-	-	-	-
Total	-	1,176,251,273.36 us	-	3,379,083.66 us	101,654,092,090.39 us	-
Average	-	2,297,365.77 us	-	13,199.55 us	397,086,297.23 us	-
Maximum	-	147,308,573.84 us	-	42,655.92 us	464,295,525.85 us	-
Minimum	-	178,179.91 us	-	10,656.79 us	340,113,799.15 us	-
StDev	-	10,950,808.17 us	-	3,806.92 us	26,341,595.78 us	-
Avg/Max	-	0.02	-	0.31	0.86	-

- While inspecting the reasons behind load imbalance…
- Looked at the call stack and related hardware counters
- The most used function, has a very low IPC (0.75)

Low IPC - cache misses

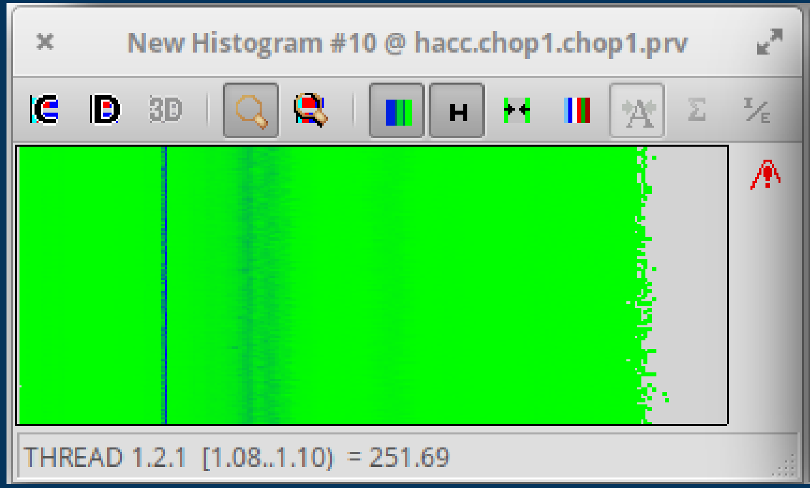


Instructions per cycle 2DZoom range [0.69561,0.842054) @ hacc.chop1.chop1.prv

```
THREAD 1.122.1
THREAD 1.122.2
THREAD 1.123.1
THREAD 1.123.2
THREAD 1.124.1
THREAD 1.124.2
THREAD 1.125.1
THREAD 1.125.2
88,729,819 us                                   88,888,320 us
```

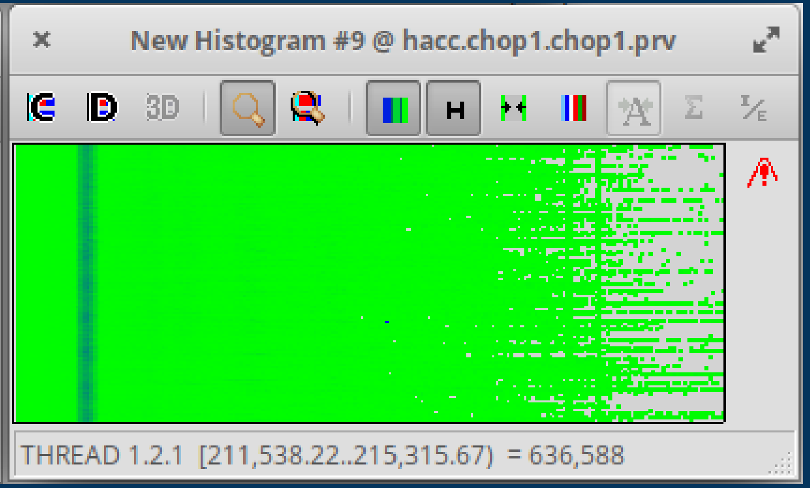
L1 cache misses 2DZoom range [38744.1,135602) @ hacc.chop1.chop1.prv

```
THREAD 1.122.1
THREAD 1.122.2
THREAD 1.123.1
THREAD 1.123.2
THREAD 1.124.1
THREAD 1.124.2
THREAD 1.125.1
THREAD 1.125.2
88,729,819 us                                   88,888,320 us
```



New Histogram #10 @ hacc.chop1.chop1.prv

THREAD 1.2.1 [1.08..1.10) = 251.69



New Histogram #9 @ hacc.chop1.chop1.prv

THREAD 1.2.1 [211,538.22..215,315.67) = 636,588

- Despite of low IPC, cache misses and IPC look uncorrelated
- As a consequence, the low IPC is probably due to long instructions
- Long instructions might be divisions or square roots
  - Could not verify, related PAPI counters not available

Part 5

Conclusions

Small wrap-up

Conclusions

- HACC has an iterative structure, based on multiple steps of the same computation, whose scaling capability is not bad (neither outstanding)
- It uses MPI Cartesian decomposition, whose geometry affects application performance (even if not in a clearly defined way)
- OpenMP offers a very important aid to computation efficiency
- Presents some load imbalance and poor instruction scalability
- The most used function has a low IPC, probably due to long instructions (e.g. divisions or square roots)

Thank you!

Any questions?