5. November 2026

Can our Scales Handle the Weight? Testing the POP Tools on Large Systems

Valentin Seitz *(valentin.seitz@bsc.es)*





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Disclaimers

- Get ready for some tables!
- I will not go through every number, but rather focus on some things.
- I will not get into the details of the performance data
- I am a biased presenter

Overview

- Introduction (A bit of measurement fundamentals)
  - How to produce measurement data
  - How to store the data
  - What to do with the data once we have it?
- Test setup
  - Tools tested
  - Applications and machine
  - Tool configurations
- Execution time overheads
- IO sizes and rates
- Post Processing and Visualisation Costs
  - CPU runs
  - GPU runs
- Summary



What is a program?



How to measure?



How to measure?



Hybrid:

- Software: In-Band Instrumentation
- Hardware: Out of band recording

Event Recorder Board



Taken from:
https://hpc.fau.de/2025/10/16/nhr-perflab-seminar-memories-of-a-performance-engineer-38-years-of-tools-building-hybrid/

How to measure?



- Software In-band recording
  - Hardware counters
  - Time
  - Energy
  - Correctness

Score-P
Scalable performance measurement infrastructure for parallel codes

Extrae

dynamic load balancing

C P T

MAQAO

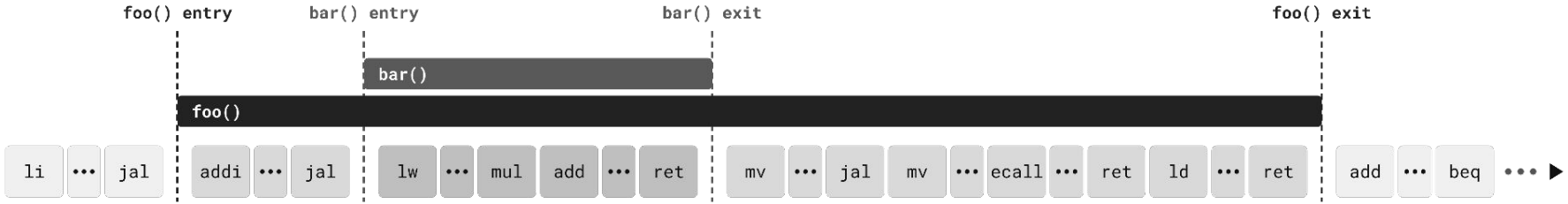
MUST

MERIC

Partition the stream



by programming language

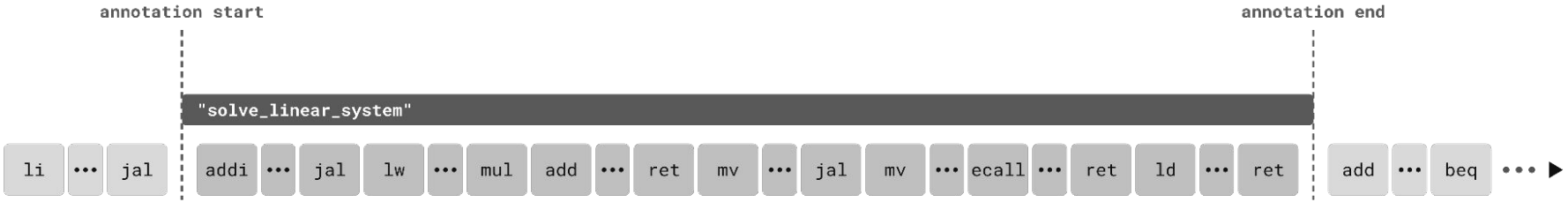


Using compilers, dynamic instrumentation, …

Partition the stream



... or user instrumentation

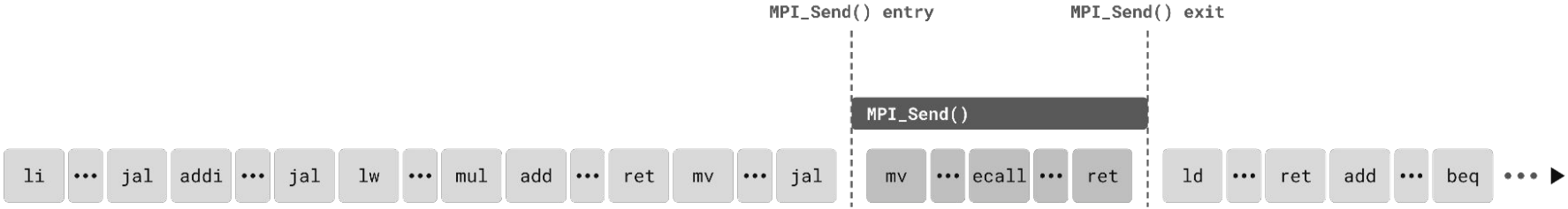


... or explicit user regions (NVTX, Caliper, etc)

Partition the stream



by programming model calls



Using interposition or callbacks

Partition the stream



by interruptions

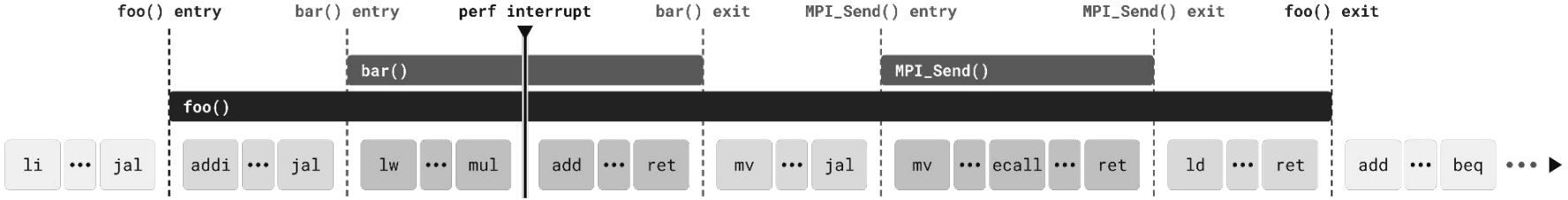


Using operating system utilities like perf, ptrace.

Partition the stream

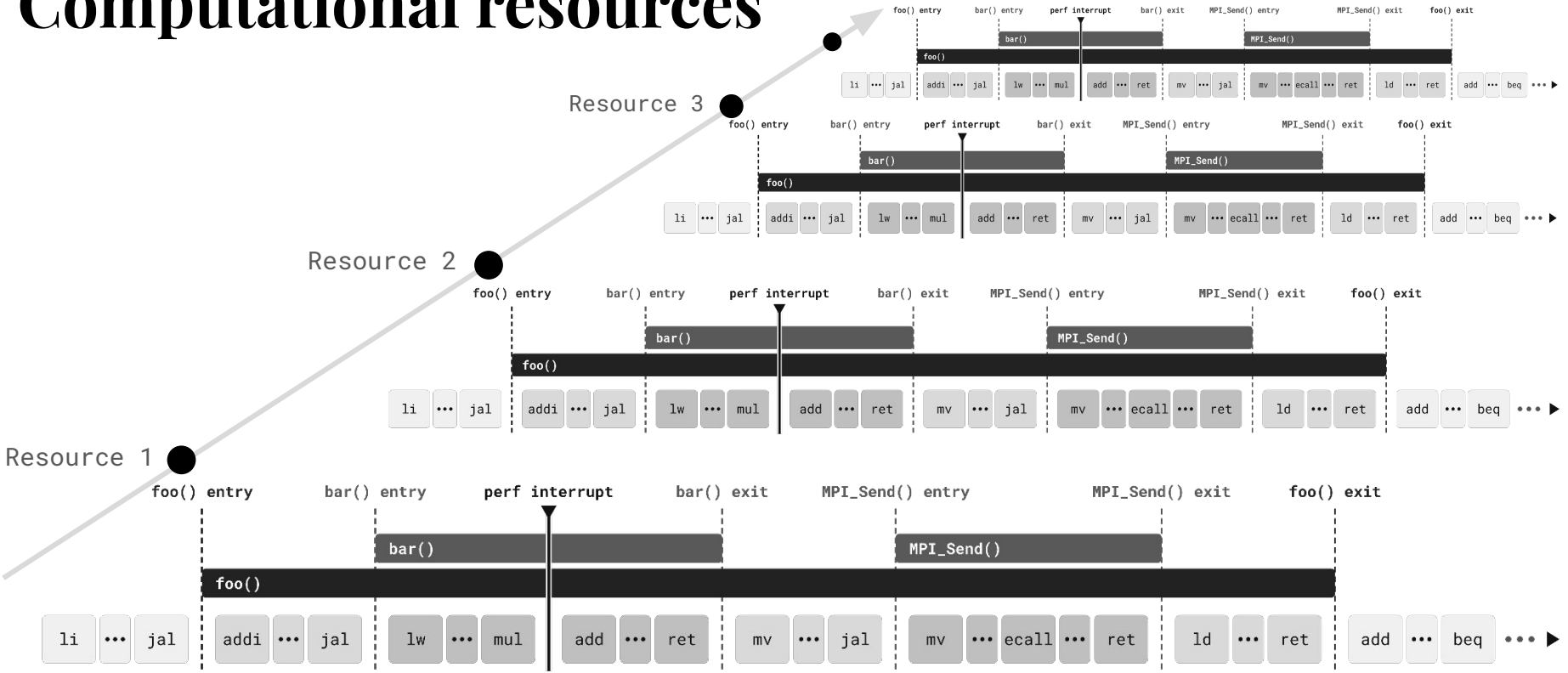


This can results in:



Here the tool gets runtime 7 times. -> granularity of 7 ***events per second***

Computational resources



How to store all that data?

- Data is fundamentally ordered (e.g. in **time**) and emitted by different **resources**
  - We can **sample** (if done in **time** it's also called **selective recording**)
  - We can **reduce** or **aggregate**
  - Or **keep as is**

- We can do this reductions
  - Online (TALP, CPT)
  - Post mortem (CUBE, BasicAnalysis)

Resource \ Time	As is	Reduced
As is	Trace	Profile
Reduced	"Trace"	Efficiency Metrics

Now I want to get insight!

Analysis / Postprocessing

scalasca

MAQAO

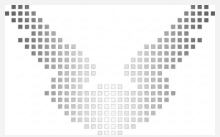
Dimemas

BasicAnalysis

Data visualization

Trace

Paraver



Profile

cube
scalasca

Test Setup

- Test setup
  - Tools tested
  - Applications and machine
  - Tool configurations

Machine

Marenostrum 5 Accelerated Partition

- 2x Intel Xeon Platinum 8460Y+
- 4x NVIDIA Hopper H100 64 HBM2
- 16x DIMM 32GB 4800MHz DDR5
- 4x ConnectX-7 NDR200
- 1120 nodes = 4480 GPUs

Marenostrum 5 General Purpose Partition

- 2x Intel Xeon Platinum 8480+
- 16x DIMM 16GB 4800MHz DDR5
- ½ ConnectX-7 NDR200 (shared between 2 nodes)
- 6192 nodes = 693504 CPU cores



GPU Application

- SOD2D
- Has NVTX annotations we redirected into the different tools
- Fortran + MPI + OpenACC
- Taylor-Green Vortex case with Re=1600
- Fully periodic structured hexahedral mesh with 544 elements per side and fifth-order polynomials

Compiled with NVIDIA HPC SDK 25.3 which provided MPI and OpenACC.

Selective recording of the sixth time step.

Strong scaling from 1024 GPUs to 2048 GPUs.



CPU Application

- Vlasiator
- Has Phiprof regions we redirect into the different tools
- C++ & MPI & OpenMP
- 3D magnetosphere production run
- 4.5TB restart file

Compiled with Clang 20 and OpenMPI 4.1.5

Selective recording of the third time step after the restart.

Strong scaling of from 56k cores to 112k cores.

Shameless plug if you want to know more about the performance of Vlasiator



CPU tools configuration

- OpenMP
  - OMPT for all tools except Extrae
  - Locks enabled
- MPI
  - Intercept the default functions
  - PMPI for all tools
- Hardware counters
  - PAPI used at each event (PAPI_TOT_INS, PAPI_TOT_CYC)



GPU tools configuration

- OpenAcc
  - Not used!
- CUDA
  - If possible driver API enabled, default behavior
- MPI
  - Intercept the default functions
  - PMPI for all tools
- Hardware counters
  - PAPI used at each event on host
  - On device no tool supports them right now.



General configuration

- Trace buffers sized so no IO during measurement
- Tool is switched of directly after MPI_Init and turned on for timestep
- Default parameters used for everything else.
- CPT does not allow for user instrumentation
  - We just enable it for the timestep
- TALP does not allow to be turned off:
  - We enumerate the regions, so it does not average out the timestep measurements
- Tools are executed in the same nodes sequentially
- All memory numbers are maximum resident set size reported by the OS



Results for data recording tools

- Execution time overheads
- IO sizes and rates



CPU Execution time overheads

	Memory	Runtime
	±10 MB	±1 %
Score-P (P)	174 MB	0.7 %
Score-P (T)		
Extrae	936 MB	5.4 %
MERIC	13 MB	-1.1 %
MAQAO	21 MB	-1.2 %
CPT	11 MB	-1.0 %
TALP	23 MB	1.2 %

Execution time overhead in terms of additional runtime in percent and memory in MB.
The intervals provided are the standard deviation of 3 uninstrumented runs for 112k cores

Tool	MPI	OpenMP	Interrupts	User
Score-P	52	91		5
Extrae	52	332		5
MERIC				2
MAQAO			50	
CPT	52*	52*		
TALP	2233	52*		5

*Granularities of the **events/s** per thread split by category and tool for the third timestep of Vlasiator on 112k Cores*
** Are estimated based on intercepted functions and Score-P data*



GPU Execution time overheads

	Memory	Runtime
	±2 MB	±2 %
Score-P (P)	56 MB	2.0 %
Score-P (T)		
Extrae	130 MB	9.7 %
MERIC	19 MB	3.0 %
TALP	36 MB	2.7 %

Execution time overhead in terms of additional runtime in percent and memory in MB.
The intervals provided are the standard deviation of 2 uninstrumented runs for 2048 GPUs

Tool	MPI	CUDA Host	CUDA Device	User
Score-P	16209	22646	2592	945
Extrae	16179	5246	1654	945
MERIC				2
TALP	16110	6321	50	945

*Granularities of the **events/s** per GPU split by category and tool for the third sixth timestep of SOD2D on 2048 GPUs*



IO rates and sizes

	I/O size	I/O rate
Score-P (P)	2.1 GB	
Score-P (T)	16 GB	0.73 GB/s
Extrae	210 GB	9.6 GB/s
MERIC	310 MB	14 MB/s
MAQAO	29 GB	1.3 GB/s
CPT	<1 MB	
TALP	<1 MB	

Size dumped to disk and IO rate of data collection using the timing of the timestep at 112k cores

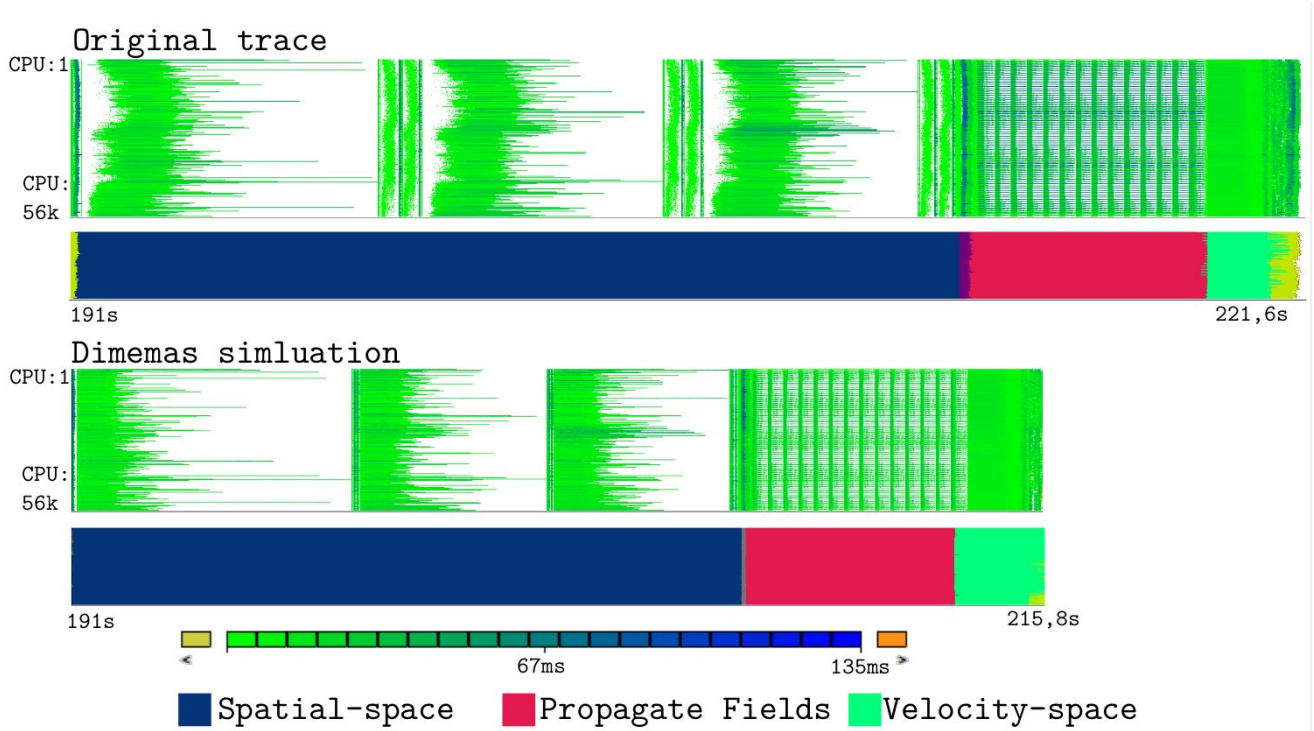
	I/O size	I/O rate
Score-P (P)	30 GB	
Score-P (T)	0.95 GB	3.6 GB/s
Extrae	2.8 GB	11 GB/s
MERIC	2 GB	7.6 GB/s
TALP	<1 MB	

Size dumped to disk and effective IO rate of data collection using the timing of the timestep at 2048 GPUs

Post Processing and Visualisation Costs: CPU

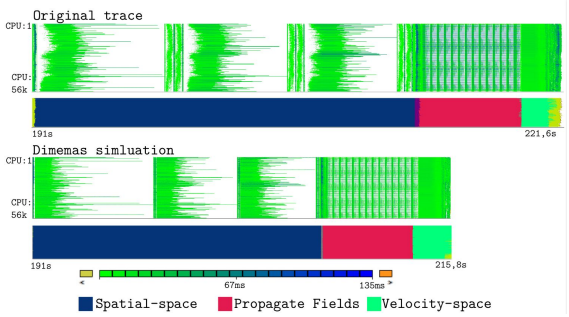
- Post Processing and Visualisation Costs
  - CPU runs

Paraver and Dimemas



Granularity between runtime calls of the Vlasiator time step on 56k cores with the simulated trace on an ideal network below.

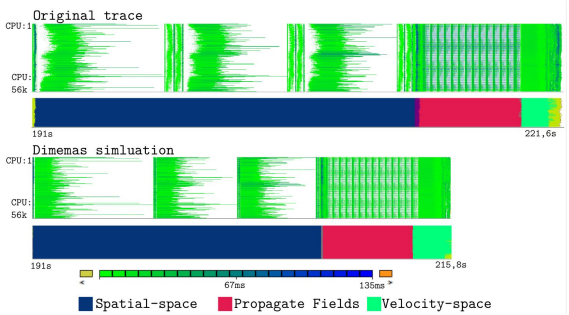
Paraver and Dimemas



Metric \ Tool	Extrae merger	Dimemas
Input Size	210GB	120GB
Time	4.5 h	5.1 h
CPU h	360 h	5.1 h
Memory	300 GB	30 GB
HDD	120 GB	160 GB

Post-processing costs for the 112k core execution

Paraver and Dimemas



Tool	Input size		Time		CPU h		Memory	
	56k	112k	56k	112k	56k	112k	56k	112k
Paraver	81GB	120GB	0.40 h		0.40 h		210 GB	

Needed wallclock time, CPU hours, and memory requirements for visualising the data of the Vlasiator runs.

BasicAnalysis

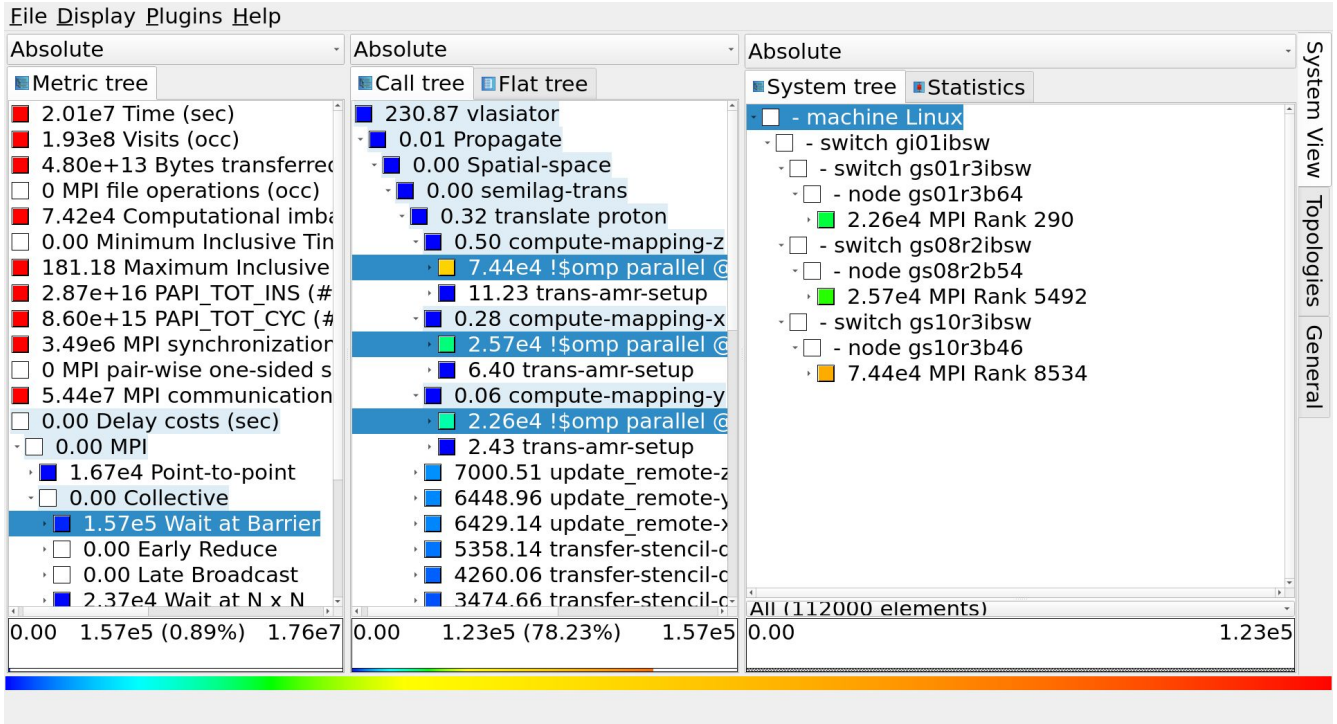
Metric	112000 (14000×8)
Hybrid Parallel efficiency	8.90%
↳ MPI Parallel efficiency	11.16%
↳ MPI Load balance	28.03%
↳ MPI Communication efficiency	39.82%
↳ MPI Serialization efficiency	63.32%
↳ MPI Transfer efficiency	62.88%
↳ OpenMP Parallel efficiency	79.77%
↳ OpenMP Load balance	81.30%
↳ OpenMP Communication efficiency	98.12%

HTML output of BasicAnalysis for the 112k core run

Metric \ Tool	Extrae merger	BasicAnalysis
Input Size	210GB	120GB
Time	4.5 h	10.7 h
CPU h	360 h	10.7 h
Memory	300 GB	303 GB
HDD	120 GB	314 GB

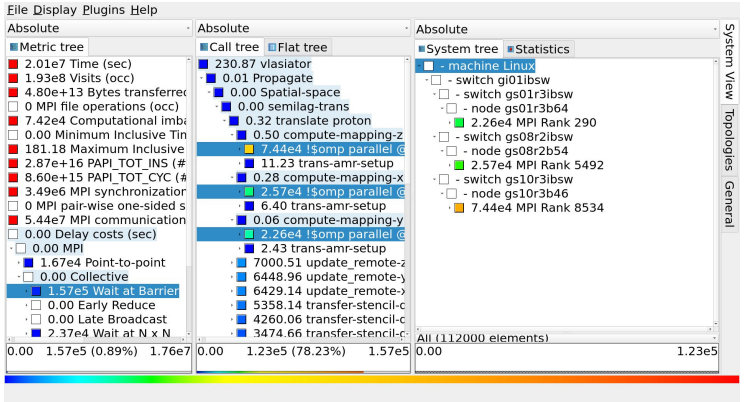
Post-processing costs for the 112k core execution

CUBE and Scalasca



Root cause analysis of the Vlasiator 112k core execution provided by Scalasca.
Cube visualises the MPI Collective Delay cost which stems from the highlighted imbalanced OpenMP parallels and the respective ranks.

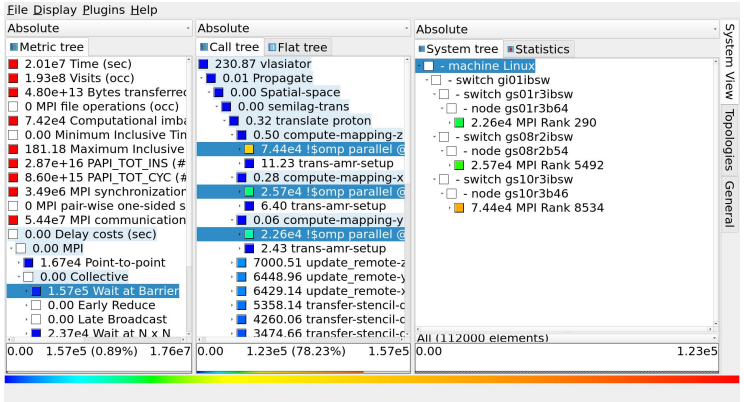
CUBE and Scalasca



Metric \ Tool	Scalasca	CUBE postprocess
Input Size	16GB	6.4GB
Time	49 s	1.3 h
CPU h	260 h	23 h
Memory	3.8 TB	130 GB
HDD	4.3GB	65GB

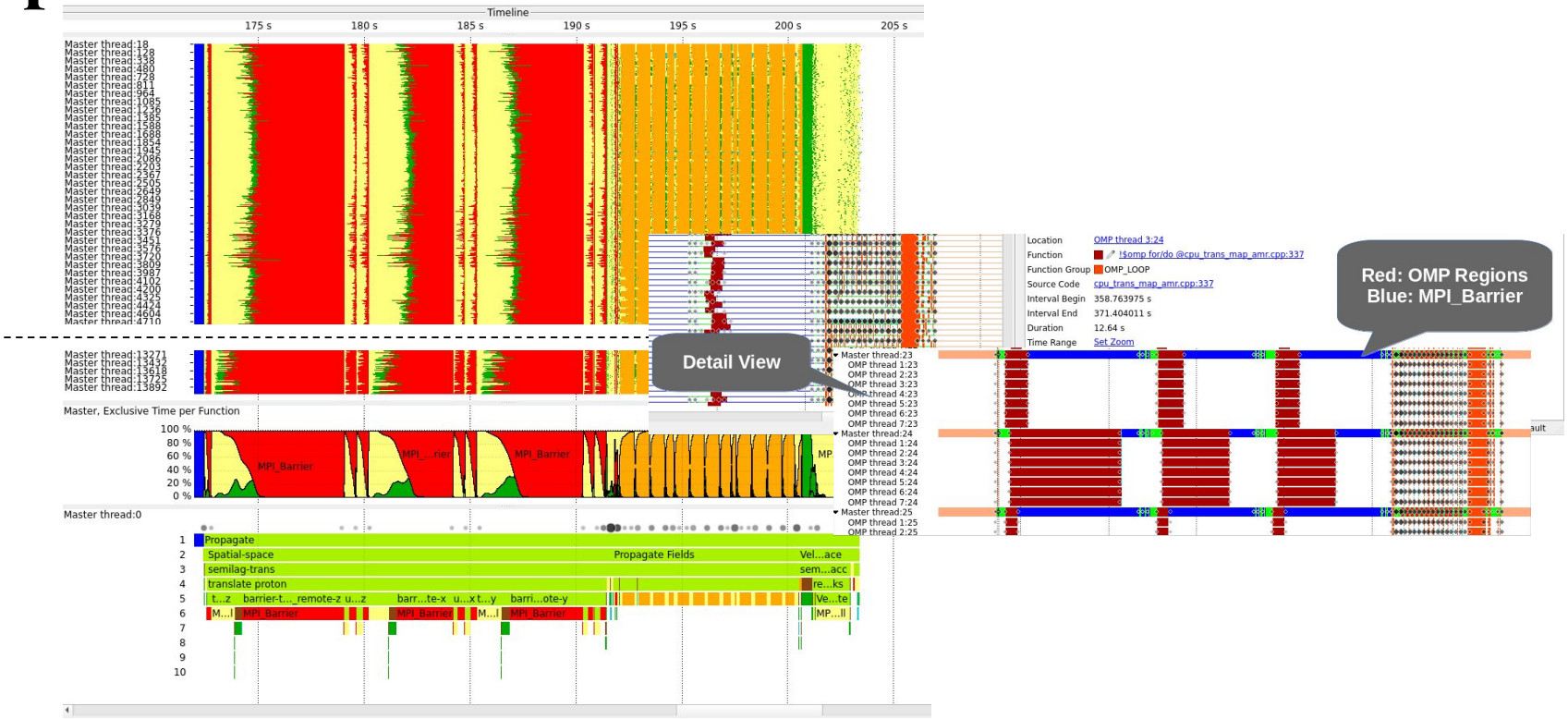
Post-processing costs for the 112k core execution

CUBE and Scalasca



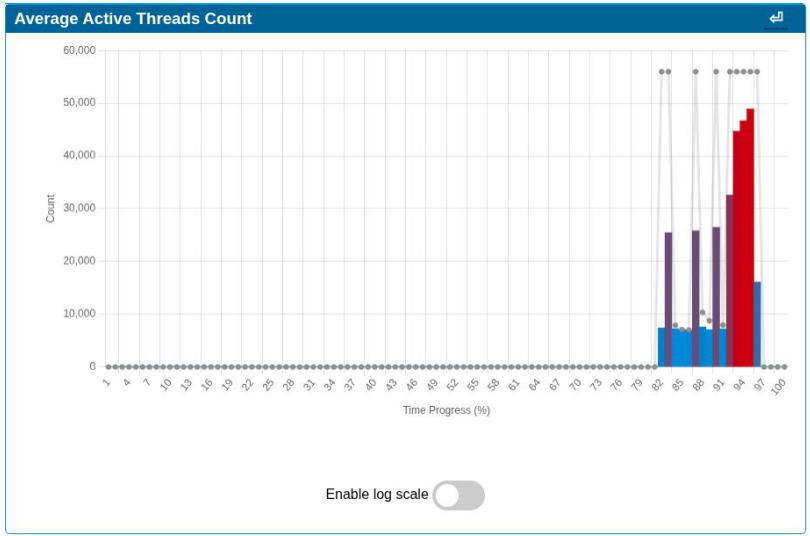
Tool	Input size		Time		CPU h		Memory	
	56k	112k	56k	112k	56k	112k	56k	112k
CUBE: GUI	32GB	65GB	30 s	30 s	1.1 h	1.1 h	2.0 GB	2.9 GB

Vampir



Timeline of the Vlasiator time step on 112k cores using Vampir.
Thankfully provided by Brian Wylie and Marc Schlütter

MAQAO



Average active threads "trace" of the Vlasiator execution on 56k cores

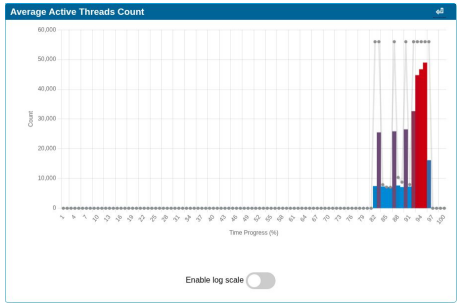
Loop id	Source Location	Source Function	▼Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)	Speedup If FP Vectorized	Speedup If Fully Vectorized
5585	vlasiator - cpu_trans_map_amr.cpp:20-144 [...]	propagatePencil(float*, Vec16f*, unsigned int, unsigned int, float, vmesh::VelocityMesh const*, int, float, float**, float*, unsigned int const*)	12.10	53.3	43.89	1.28	1.64
5610	vlasiator - cpu_trans_map_amr.cpp:359-369 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	1.62	5.26	7.65	1	33.05
3109	vlasiator - fsgrid.hpp:822-1007 [...]	SBC::SysBoundaryCondition::fieldBoundaryCopyFromSolvingNbrMagneticField(FsGrid<std::array<double, 3ul>, 2>&, FsGrid<fsgrids::technical, 2>&, int, int, int, unsigned int, unsigned int)	1.58	83	32.16	1	4.11
5607	vlasiator - cpu_trans_map_amr.cpp:386-395 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	1.08	38.46	21.69	1	6.64
5541	vlasiator - cpu_acc_map.cpp:571-623 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.88	57.6	60.25	1	1.04
3107	vlasiator - fsgrid.hpp:822-1012 [...]	SBC::SysBoundaryCondition::fieldBoundaryCopyFromSolvingNbrMagneticField(FsGrid<std::array<double, 3ul>, 2>&, FsGrid<fsgrids::technical, 2>&, int, int, int, unsigned int, unsigned int)	0.80	1.75	7.57	1	15.43
5540	vlasiator - cpu_acc_map.cpp:533-623 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.56	86.81	83.86	1.01	1.02
5609	vlasiator - vectorf512.h:127-127 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	0.49	70.63	47.33	1	1.7
5542	vlasiator - cpu_1d_pqm.hpp:98-140 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.49	6.67	10.63	1.7	4.58

Loops report of the 56k core execution.

Explore it yourself here:

https://datafront.maqao.org/public/VLASIATOR/Magnetosphere/SPR-MARENOSTRUM/clang/vlasiator_magnetoshphere_spr_08_m14_n500_ov1_valentin_v3/index.html

MAQAO

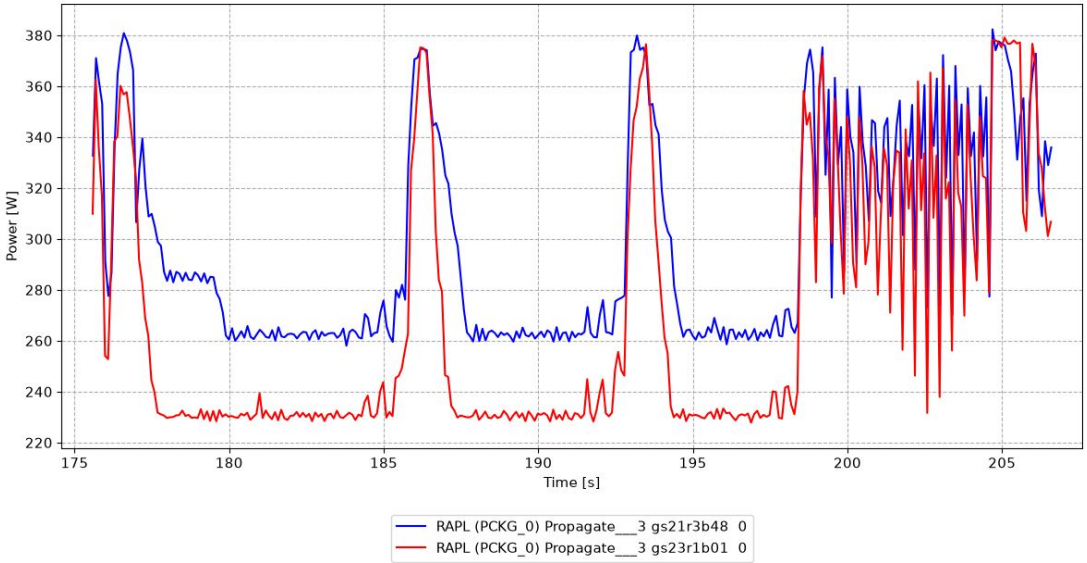


Loop Id	Source Location	Source Function	▼ Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)	Speedup If FP Vectorized	Speedup If Fully Vectorized
5585	vlasiator - cpu_trans_map_amr.cpp: 20-144 [...]	propagateParcel(float*, Vec16f*, unsigned int, unsigned int, float, vmesh::VelocityMesh const*, int, float, float**, float*, unsigned int const*)	12.10	53.3	43.89	1.28	1.64
5610	vlasiator - cpu_trans_map_amr.cpp: 358-369 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	1.62	5.26	7.65	1	33.05
3109	vlasiator - fsgrid.hpp:822-1007 [...]	SBC::SysBoundaryCondition::fieldBoundaryCopyFromSolvingNbrMagneticField(FsGrid<std::array<double, 3ul>, 2>&, FsGrid<fsgrids::technical, 2>&, int, int, int, unsigned int, unsigned int)	1.58	83	32.16	1	4.11
5607	vlasiator - cpu_trans_map_amr.cpp: 386-395 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	1.08	38.46	21.69	1	6.64
5541	vlasiator - cpu_acc_map.cpp:571-625 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.88	57.6	60.25	1	1.04
3107	vlasiator - fsgrid.hpp:822-1012 [...]	SBC::SysBoundaryCondition::fieldBoundaryCopyFromSolvingNbrMagneticField(FsGrid<std::array<double, 3ul>, 2>&, FsGrid<fsgrids::technical, 2>&, int, int, int, unsigned int, unsigned int)	0.80	1.75	7.57	1	15.43
5540	vlasiator - cpu_acc_map.cpp:533-625 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.56	86.81	83.86	1.01	1.02
5609	vlasiator - vector/512.h:127-127 [...]	trans_map_1d_amr(dccrg::Dccrg<spatial_cell::SpatialCell, dccrg::Cartesian_Geometry, std::tuple<>, std::tuple<> > const&, std::vector<unsigned long, std::allocator<unsigned long> > const&, std::vector<unsigned long, ...	0.49	70.63	47.33	1	1.7
5542	vlasiator - cpu_1d_ppm.hpp:98-140 [...]	map_1d(spatial_cell::SpatialCell*, unsigned int, double, double, double, unsigned int)	0.49	6.67	10.63	1.7	4.58

Metric \ Tool	MAQAO report
Input Size	29GB
Time	1.9 h
CPU h	1.9 h
Memory	40 GB
HDD	110 MB

Post-processing costs for the ONE VIEW report generation

MERIC



RAPL reported power during the timestep of on 56k cores. Plot showing CPUs with the highest and the lowest energy consumption

CPT and TALP

#Ranks x #Threads	7000x8	14000x8
Global Efficiency	13.0	9.4
Parallel Efficiency	13.0	9.0
Load Balance	26.8	21.7
Communication Efficiency	48.5	41.4
Serialisation Efficiency	64.7	65.4
Transfer Efficiency	74.9	63.3
MPI Parallel Efficiency	14.4	9.7
MPI Load Balance	28.4	23.3
MPI Communication Efficiency	50.7	41.6
MPI Serialisation Efficiency	67.8	65.7
MPI Transfer Efficiency	74.9	63.3
OMP Parallel Efficiency	90.2	92.6
OMP Load Balance	94.4	93.1
OMP Communication Efficiency	95.5	99.5
OMP Serialisation Efficiency	95.5	99.5
OMP Transfer Efficiency	100.0	100.0
Computational Scalability	100.0	104.7



CPT output for the 56k and 112k cores runs

Metric	Spatial-space		Propagate Fields		Velocity-space	
CPU cores	56k	112k	56k	112k	56k	112k
Parallel efficiency	0.050	0.030	0.080	0.080	0.55	0.54
MPI Parallel eff.	0.10	0.070	0.18	0.19	0.78	0.79
MPI Load balance	0.19	0.16	0.21	0.22	0.82	0.82
MPI Comm. eff.	0.55	0.45	0.85	0.88	0.95	0.96
OpenMP Parallel eff.	0.55	0.55	0.57	0.57	0.79	0.78
OpenMP Load balance	1.0	1.0	0.99	0.99	1.0	1.0
OpenMP Sched. eff.	1.0	1.0	1.0	1.0	1.0	1.0
OpenMP Serial. eff.	0.55	0.55	0.57	0.57	0.79	0.78
Computation scalability	1.0	1.1	1.0	1.0	1.0	0.99
IPC scalability	1.0	1.1	1.0	1.0	1.0	1.0
Instruction scalability	1.0	1.0	1.0	1.0	1.0	0.99
Frequency scalability	1.0	1.0	1.0	1.0	1.0	1.0

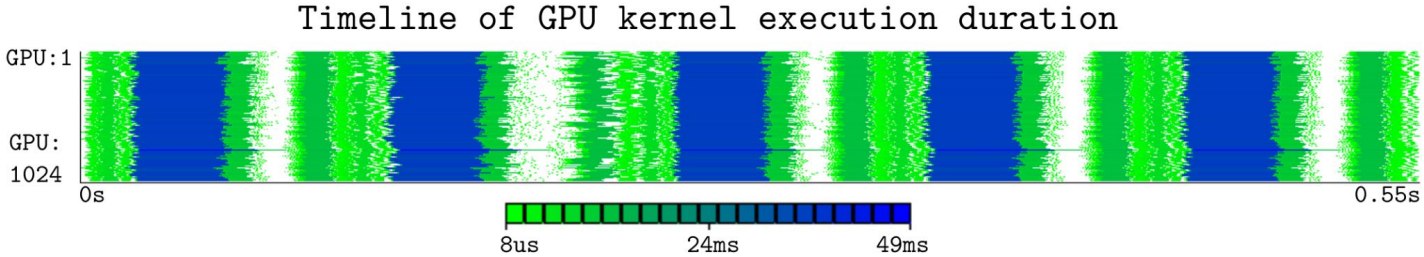
Table generated from TALP data obtained scaling from 56 to 112k cores

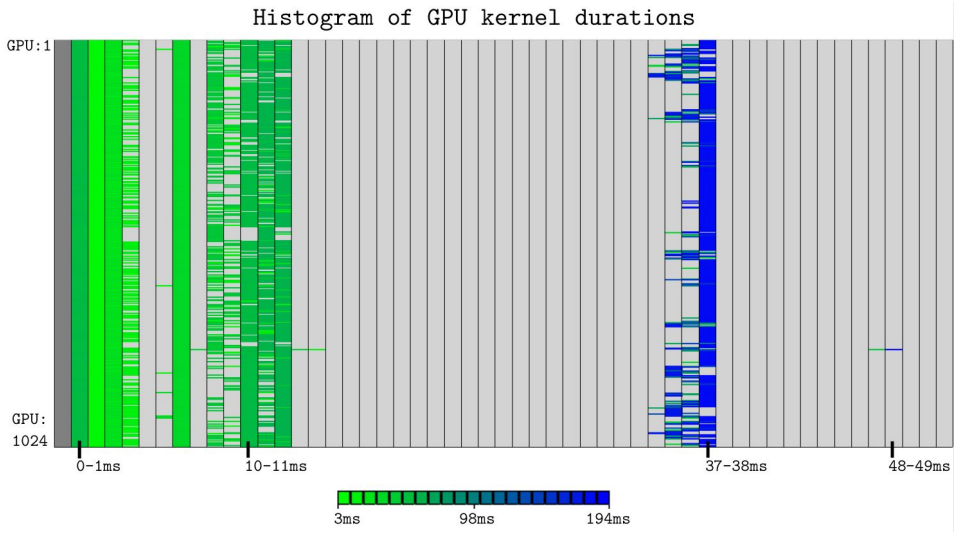
Post Processing and Visualisation Costs: GPU

- Post Processing and Visualisation Costs
  - GPU runs

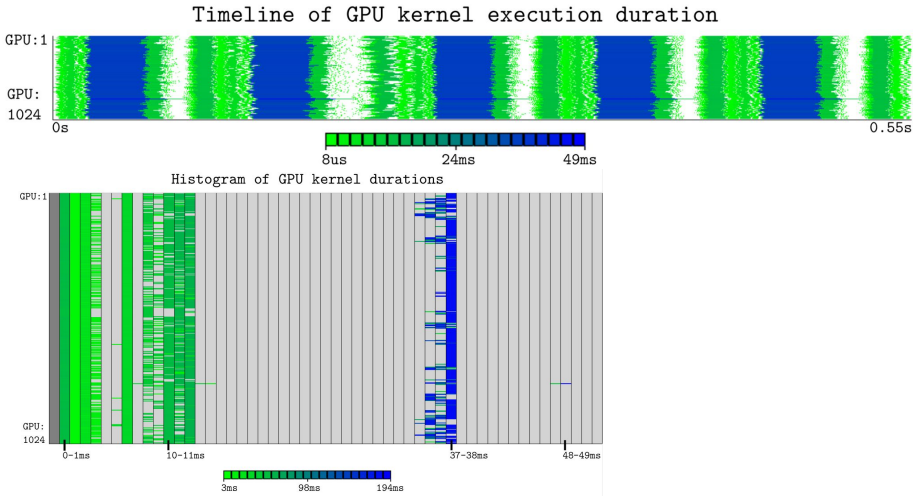


Paraver


Timeline of GPU kernel execution duration
GPU:1
GPU:
1024
0s
0.55s
8us
24ms
49ms


Histogram of GPU kernel durations
GPU:1
GPU:
1024
0-1ms
10-11ms
37-38ms
48-49ms
3ms
98ms
194ms

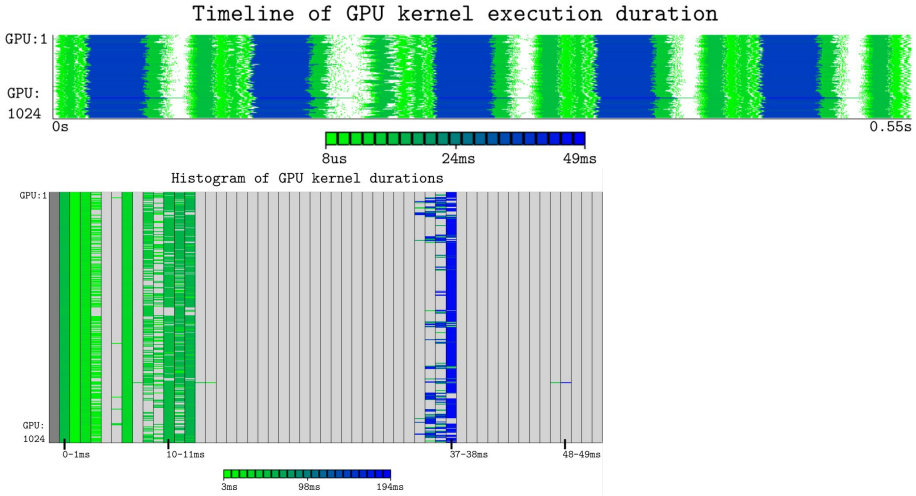
Paraver



Metric \ Tool	Extrae merger
Input Size	2.8 GB
Time	0.93 h
CPU h	0.93 h
Memory	15 GB
HDD	1.4 GB

Post-processing costs for the 2048 GPU execution

Paraver



Tool	Input size		Time		CPU h		Memory	
	1024	2048	1024	2048	1024	2048	1024	2048
Paraver	0.7GB	1.4GB	15 s	25 s	15 s	25 s	2.3 GB	4.2 GB

Needed wallclock time, CPU hours, and memory requirements for visualising the data of the SOD2D runs.

BasicAnalysis

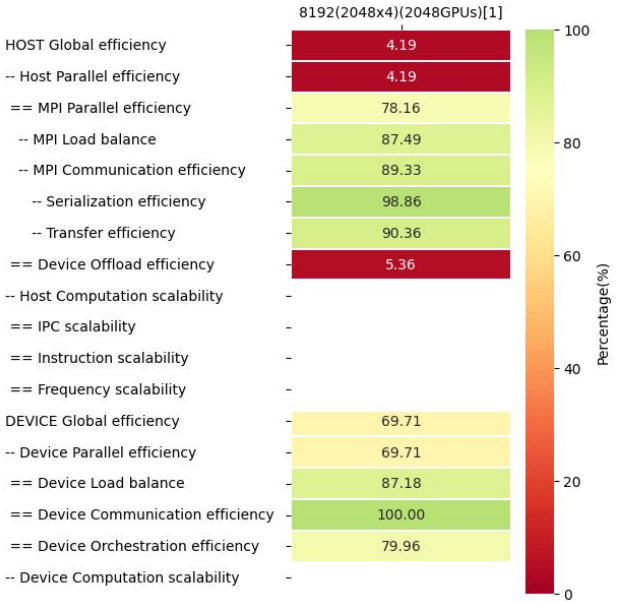
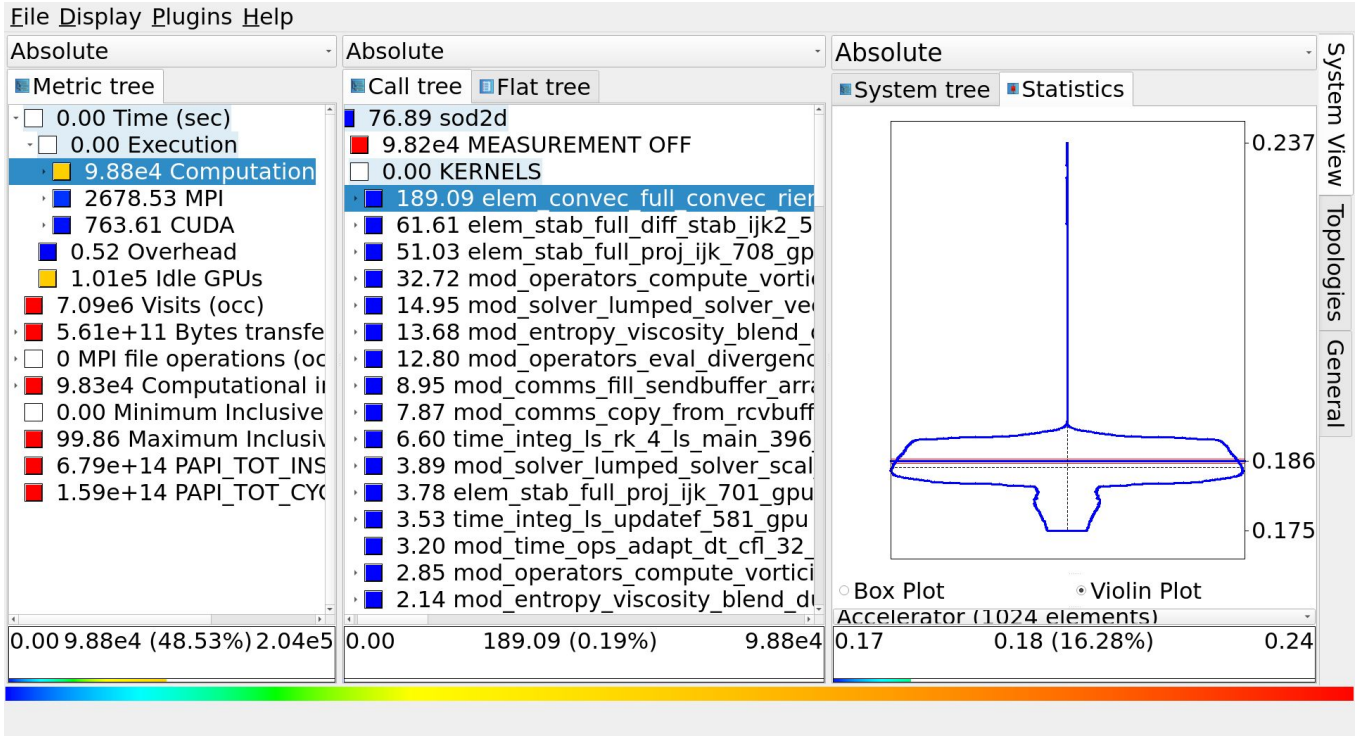


Table produced by BasicAnalysis using the 2048 GPU run

Metric \ Tool	Extrae merger	BasicAnalysis
Input Size	2.8 GB	1.4 GB
Time	0.93 h	0.3 h
CPU h	0.93 h	0.3 h
Memory	15 GB	15 GB
HDD	1.4 GB	7.4 GB

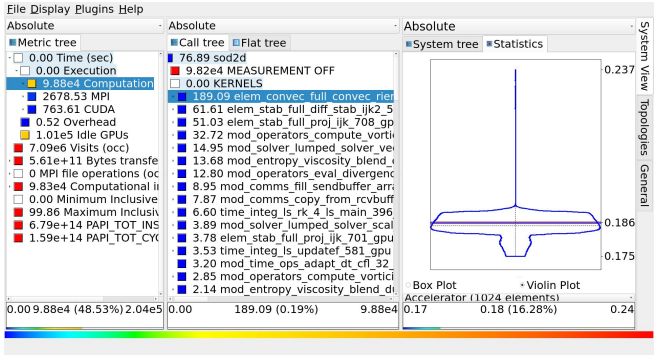
Post-processing costs for the 2048 GPU execution

CUBE



Score-P recorded profile of the 1024 GPU SOD2D run showing that the longest kernels on average takes 186 ms while one GPU takes 237 ms.

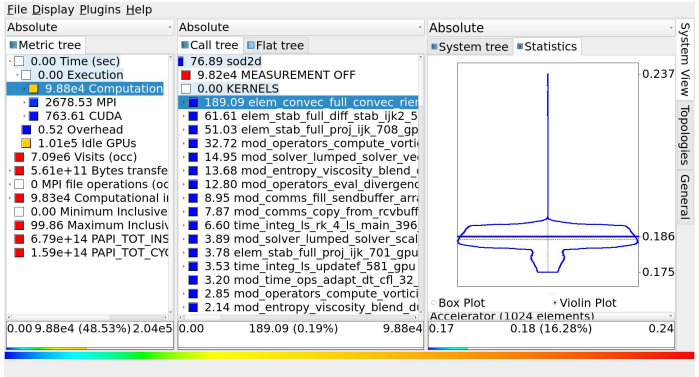
CUBE



Metric \ Tool	CUBE remap
Input Size	30 GB
Time	13 h
CPU h	760 h
Memory	390 GB
HDD	320 GB

Post-processing costs for the 2048 GPU execution

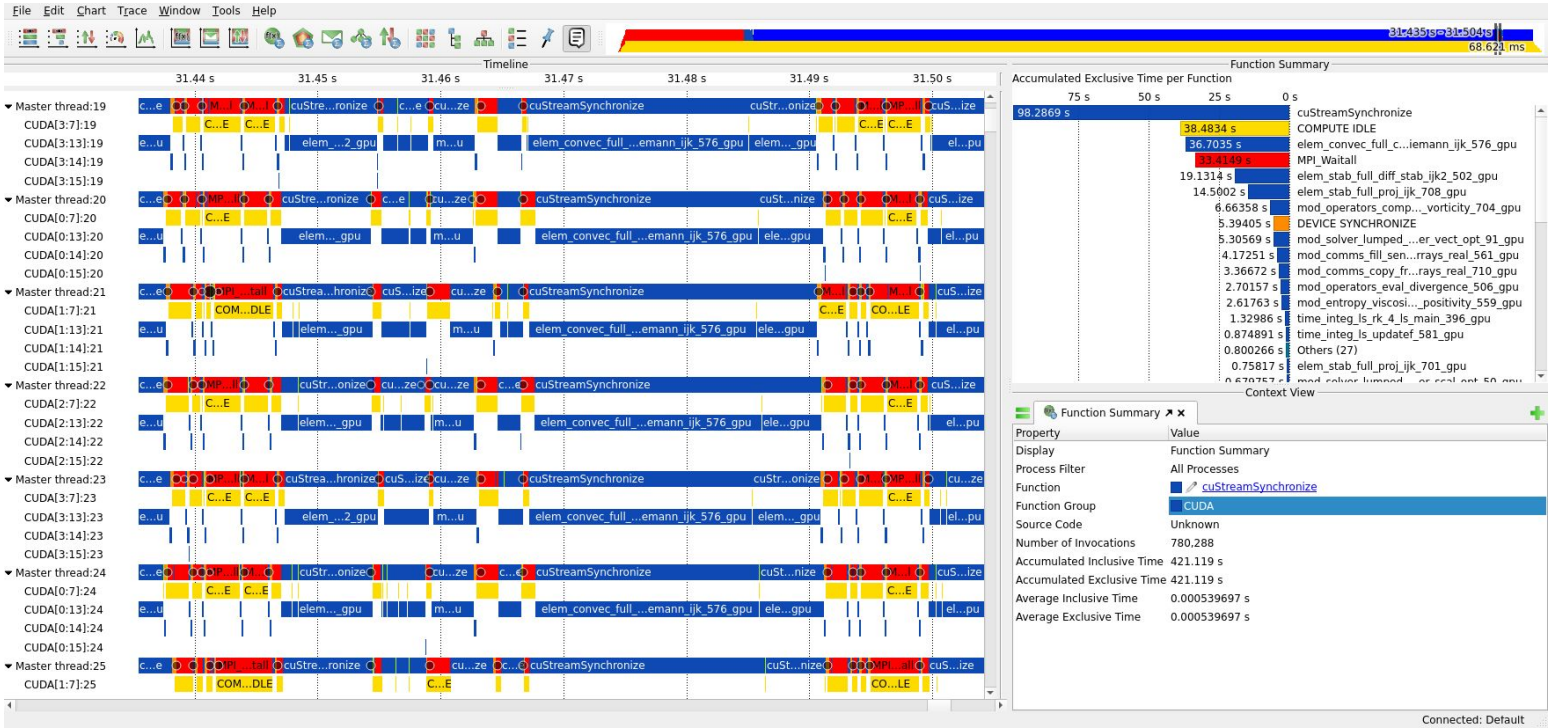
CUBE



Tool	Input size		Time		CPU h		Memory	
	1024	2048	1024	2048	1024	2048	1024	2048
CUBE: GUI	138GB	322GB	90 s	90 s	0.19 h	0.53 h	15 GB	15 GB

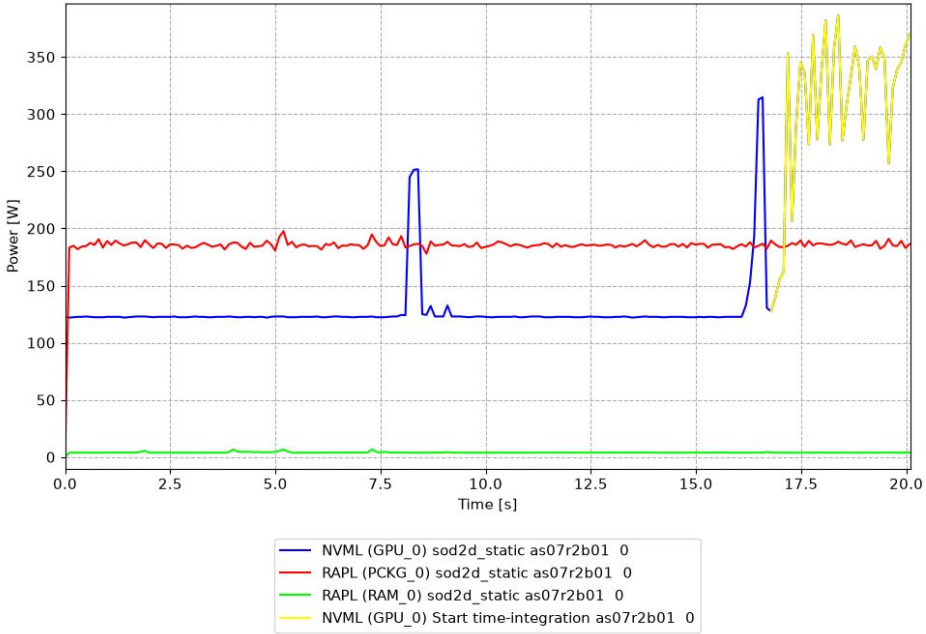
Needed wallclock time, CPU hours, and memory requirements for visualising the data of the SOD2D runs.

Vampir



Timeline visualization of 7 GPUs with MPI calls and kernels running in 4 different CUDA Streams.
Thankfully provided by Marc Schlütter

Meric



CPU and GPU power consumption during SOD2D execution in 2048 GPUs

TALP

Metric	Global		Start time-integration	
	1024 GPUs	2048 GPUs	1024 GPUs	2048 GPUs
Host parallel efficiency	0.83	0.83	0.030	0.040
MPI parallel efficiency	0.97	0.96	0.81	0.80
MPI Load balance	0.98	0.99	0.87	0.89
MPI Comm. eff.	0.98	0.97	0.94	0.90
Device Offload eff.	0.86	0.87	0.030	0.050
Host Comp. scalability	1.0	0.88	1.0	0.64
IPC scalability	1.0	1.1	1.0	0.82
Instruction scalability	1.0	0.74	1.0	1.5
Frequency scalability	1.0	1.1	1.0	0.52
Device Parallel efficiency	0.13	0.11	0.80	0.74
Device Load balance	0.89	0.88	0.86	0.88
Device Comm. eff.	0.97	0.97	1.0	1.0
Device Orch. eff.	0.15	0.13	0.92	0.84
Device Comp. scalability	1.0	1.0	1.0	1.0

Table generated from TALP data obtained scaling from 1024 to 2048 GPUs

Summary:

- With our tested applications the tools:
  - Did **not produce prohibitive overhead**
  - Did produce data
- With the amount of produced data, we were able to:
  - Run post-processing tools
  - Run simulation and analysis tools
- And finally we were able to **get insight** through different visualisation modes and tools



Some Lessons learned:

- General
  - IO, especially metadata operations is expensive and a lot of tools rely on: one or more files per traced resource.
- Data producing tools
  - We see very heterogeneous IO sizes and memory requirements (4-13x) for very “similar” configured tools
- Post-processing
  - Sparsity in dense formats can become expensive
  - Reducing across time and resources post-mortem is expensive as well
- Visualisation
  - It’s a good idea to not slurp the whole dataset into memory



Acknowledgements

We especially thank the BSC User-Support Team headed by David Vicente Dorca.

Without their help we could not have obtained the measurements of high GPU counts during the EPICURE High-Scalability Workshop.

We also thank the Vlasiator and SOD2D teams for providing the inputs to run at this scale.

This work has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931 (POP3 project)

The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal, the Czech Republic, Sweden, Finland, Greece, and Slovenia.

The POP3 project (PCI2024-153419) is also co-funded by MICIU/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR.

Access to the CPU computing resources was provided by the EuroHPC Development Access Call EHPC-DEV-2025D05-038

