Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Analysis of PFLOTRAN on Jaguar

Kevin Huck, Jesus Labarta, Judit Gimenez, Harald Servat, Juan Gonzalez, and German Llort

CScADS – Workshop on Performance Tools for Petascale Computing
August 2-5, 2010 – Snowbird, UT

How this got started...

- As part of a Dagstuhl seminar in May, 2010, the Performance Tools group performed large-scale performance analysis of two applications on two machines
  - PEPC – Pretty Efficient Parallel Coulomb-solver
    - Parallel tree code for rapid computation of long-range Couomb forces in N-body particle systems
  - PFLOTRAN
    - Modeling Multiscale-Multiphase-Multicomponent Subsurface Reactive Flows Using Advanced Computing
  - Jaguar – XT5 at Oak Ridge National Laboratory
  - Jugene – BG/P at Jülich Supercomputing Center

PFLOTRAN Overview

- Using PERI “Tiger Team” instructions*, build and run PFLOTRAN on the target systems
- Opportunity! - porting Extrae measurement system to a new architecture: XT5
- Challenge... - porting Extrae measurement system to a new architecture (ongoing)
- Using scalable trace collection methods, traces were collected using 8k, 12k and 16k processes on Jugene, 8k processes on Jaguar

* http://secure-water.org/wiki/index.php?title=PFLOTRAN_Tiger_Team_Instructions

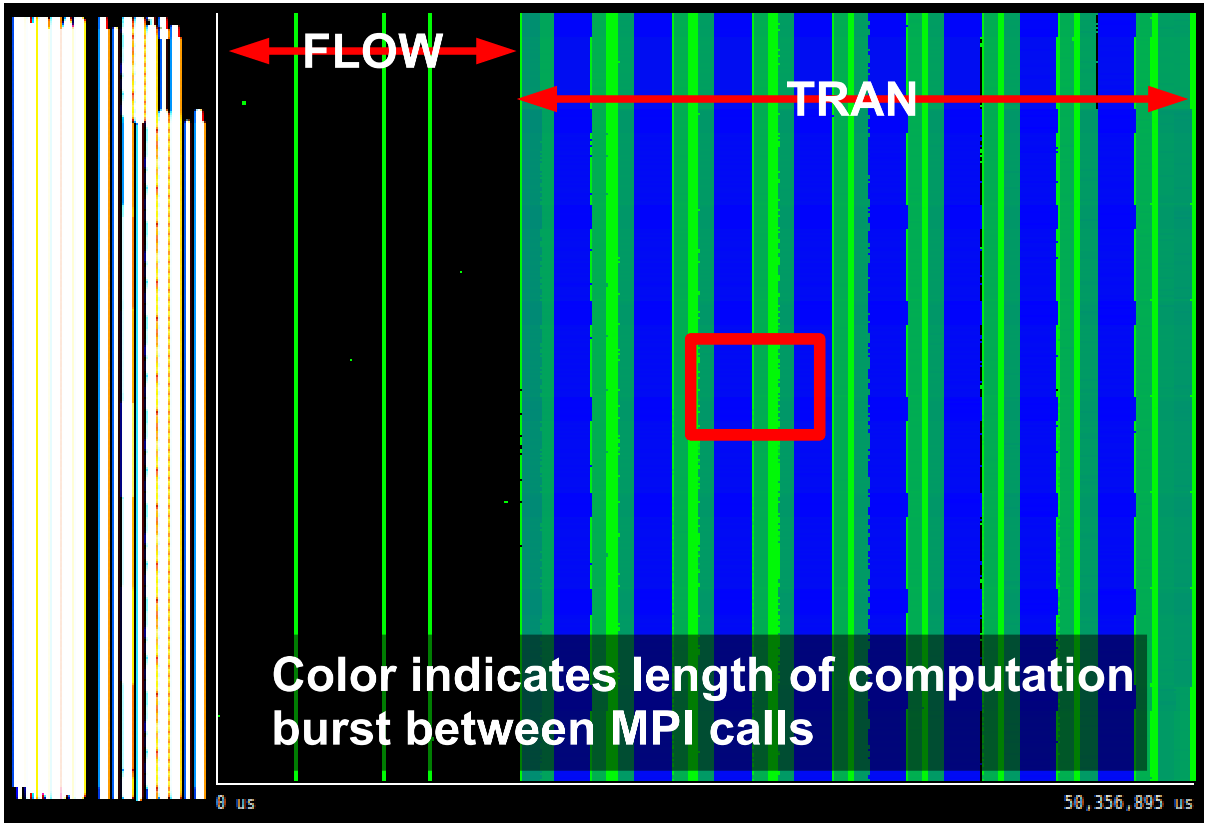
PFLOTRAN Overview, continued

- Iteration-based simulation with two main stages: **FLOW** and **TRAN**sfer
- Based on PETSc, uses BCGS solver (IBCGS solver developed, but not yet accepted)
- Input dataset computes with 1+Billion degrees of freedom in TRAN stage, relatively little computation in the FLOW stage
- FLOW scales poorly, TRAN scales well

BSC Performance Tools Overview

- Extrae 2.0 – measurement library
  - Default behavior: measures time between MPI calls or OpenMP synchronization points
    - Stores the callpath to the event
  - User Functions using instrumentation
    - Source or Dyninst
  - Sampling support
  - Options for scalable trace collection
- Paraver – trace visualizer (OTF supported)
- Our initial setup: “burst” traces with 20ms threshold
  - Some MPI statistics, but no detail
  - 10 iterations, 8K cores yields a ~1.8GB trace

Jaguar – Useful Duration with 8k cores

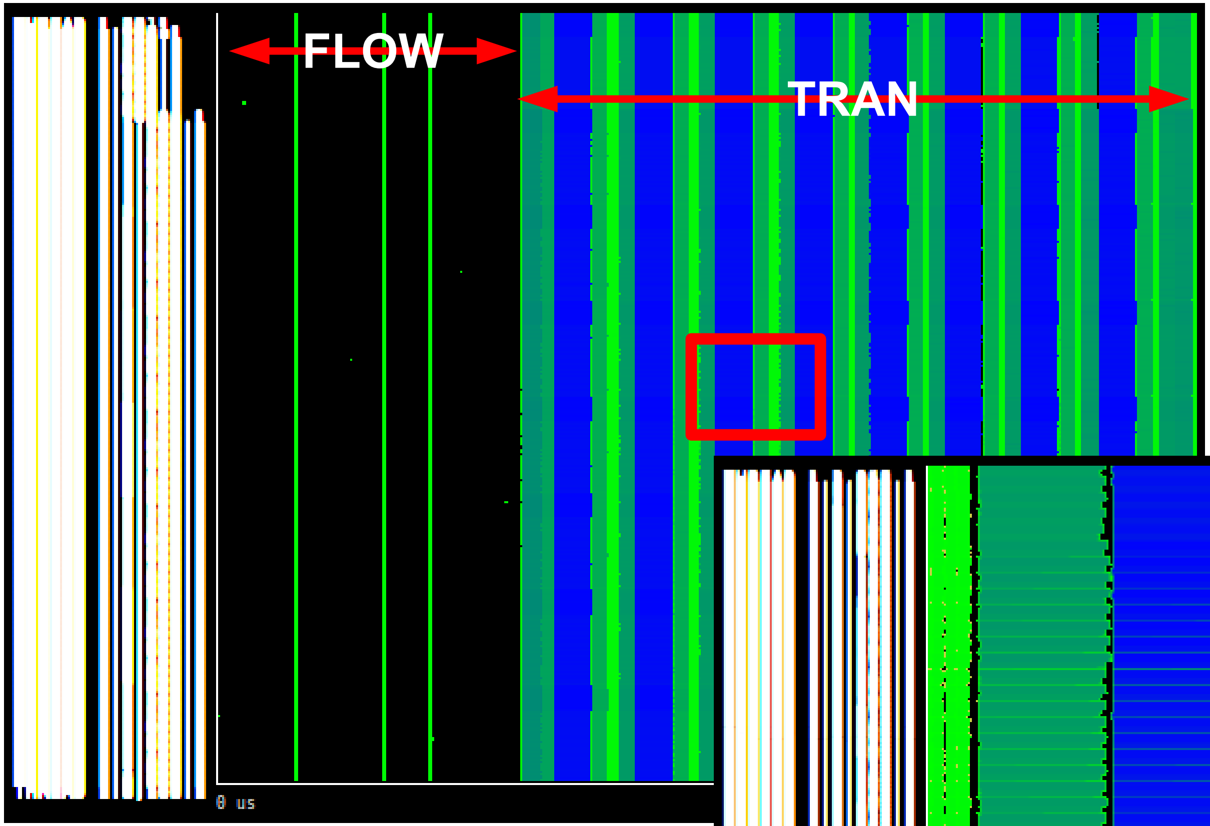


Appears balanced...

Paraver timeline view: x-axis is time, y-axis is process, compressed display which gives priority to larger values. This timeline is a gradient from bright green (smallest value) to deep blue (largest value).

...but zooming in reveals imbalance

Jaguar – Useful Duration with 8k cores

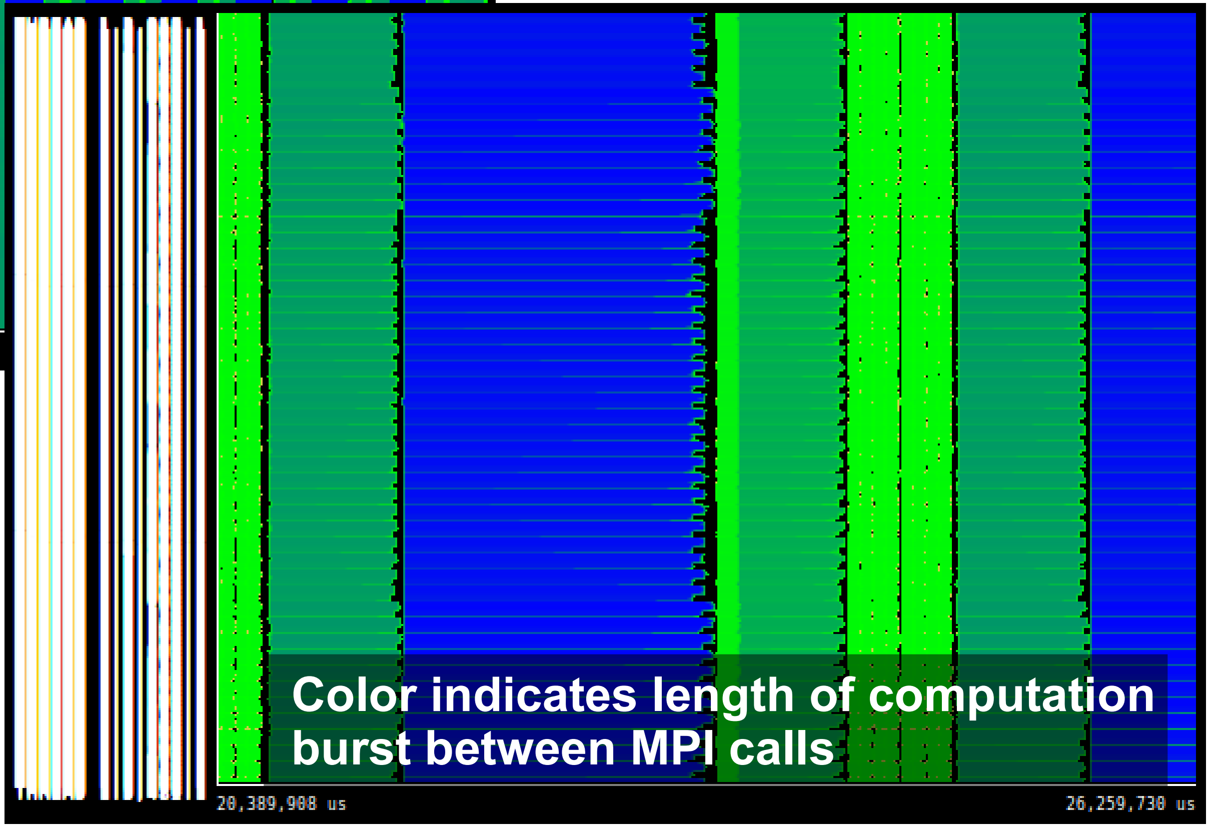


Paraver timeline view: x-axis is time, y-axis is process, compressed display which gives priority to larger values. This timeline is a gradient from bright green (smallest value) to deep blue (largest value).

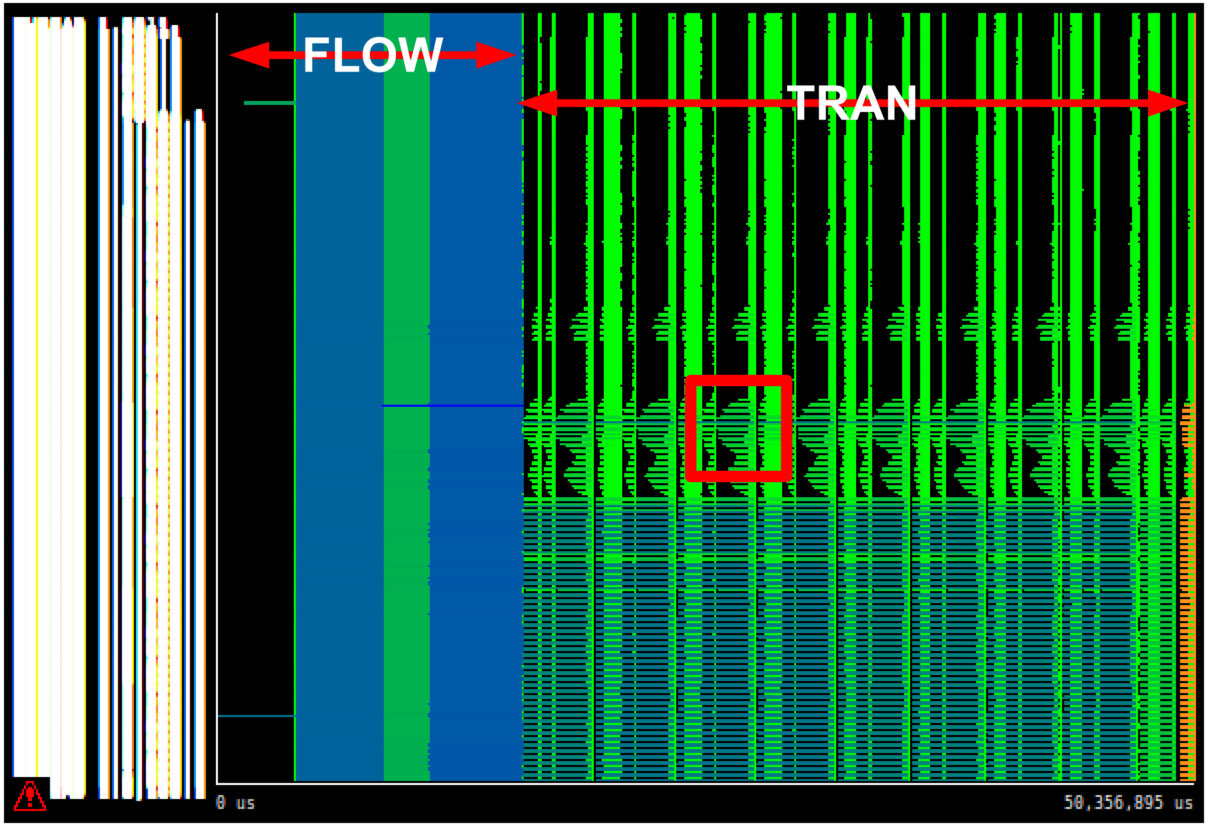
...but zooming in reveals imbalance

Appears balanced...

“Sawtooth” pattern among nodes, and some processes have very little work to do (horizontal bright green strips)



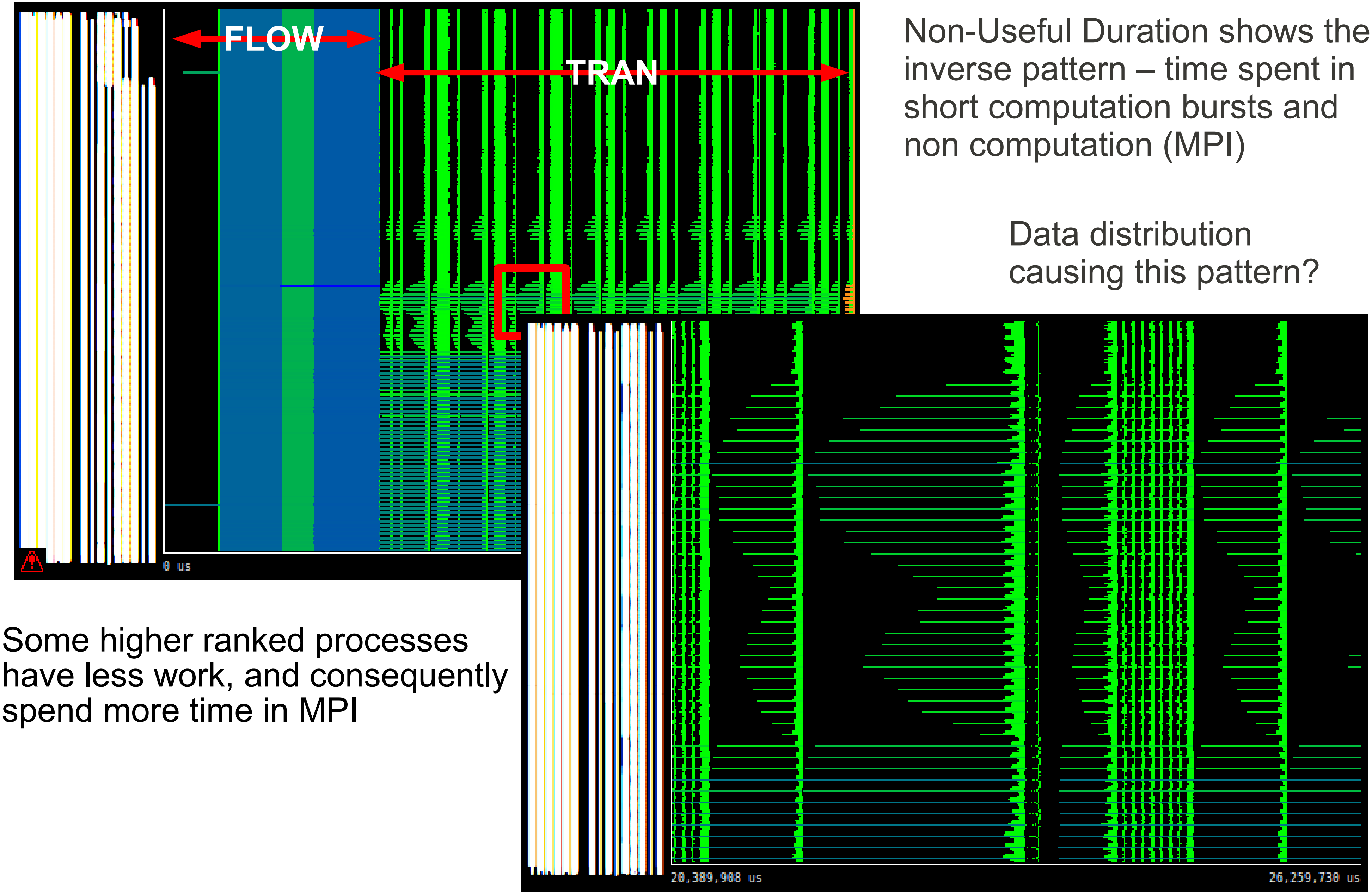
Jaguar – Non Useful Duration



Non-Useful Duration shows the inverse pattern – time spent in short computation bursts and non computation (MPI)

...zooming in reveals imbalance

Jaguar – Non Useful Duration



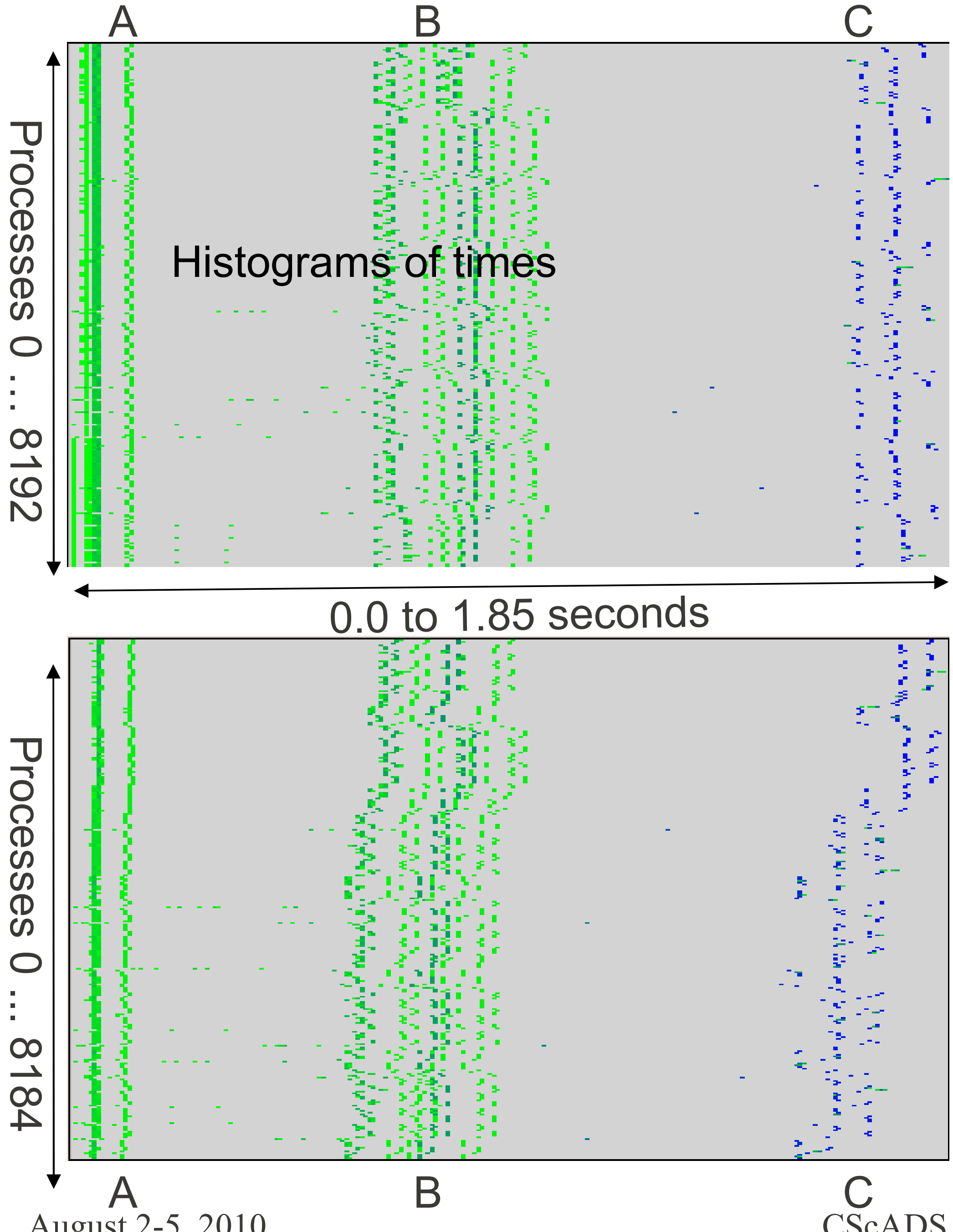
Non-Useful Duration shows the inverse pattern – time spent in short computation bursts and non computation (MPI)

Data distribution causing this pattern?

Some higher ranked processes have less work, and consequently spend more time in MPI

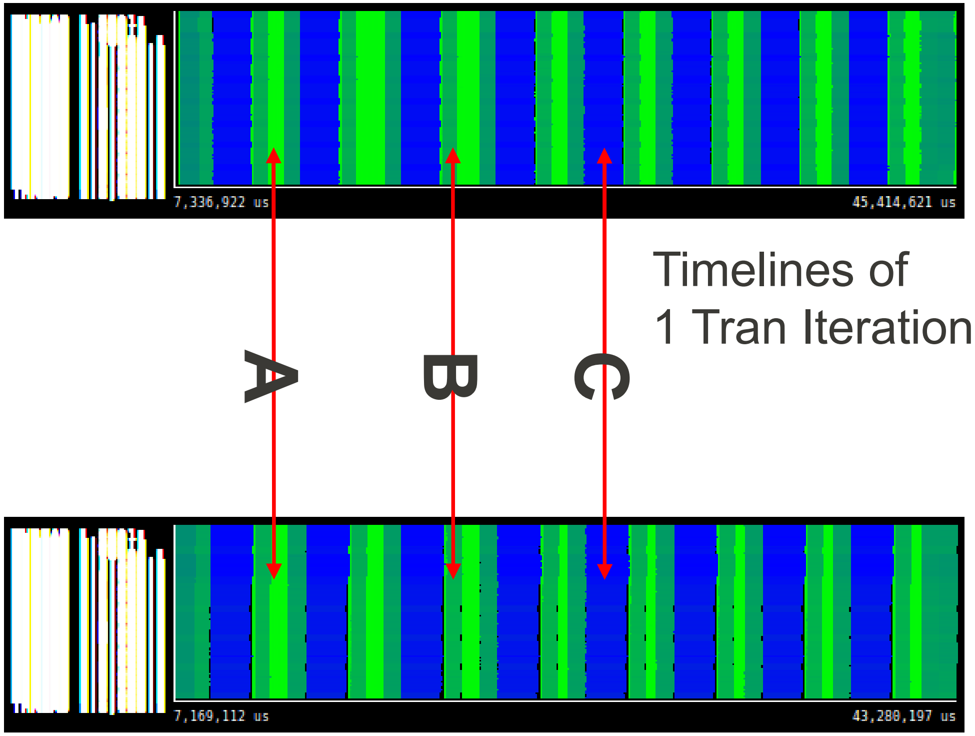
TRAN Analysis

Comparing imbalances: 8192 to 8184



total = X * Y * Z

8192 = 32 x 32 x 8



Timelines of
1 Tran Iteration

8184 = 31 x 44 x 6

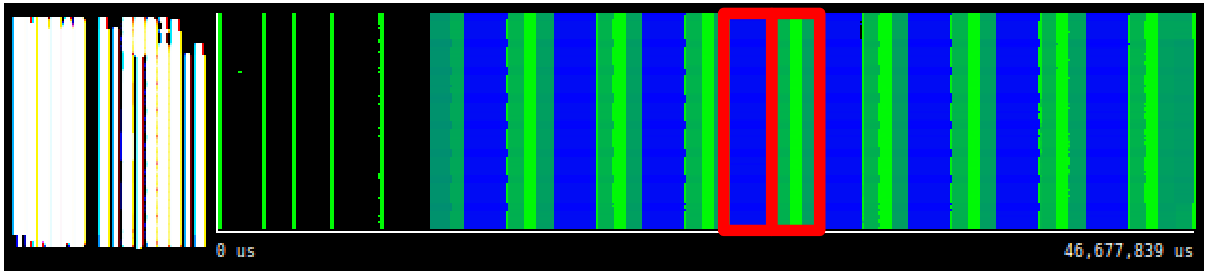
For longest bursts, more variance with 8184 – other processes have to wait

Load Balance Computation

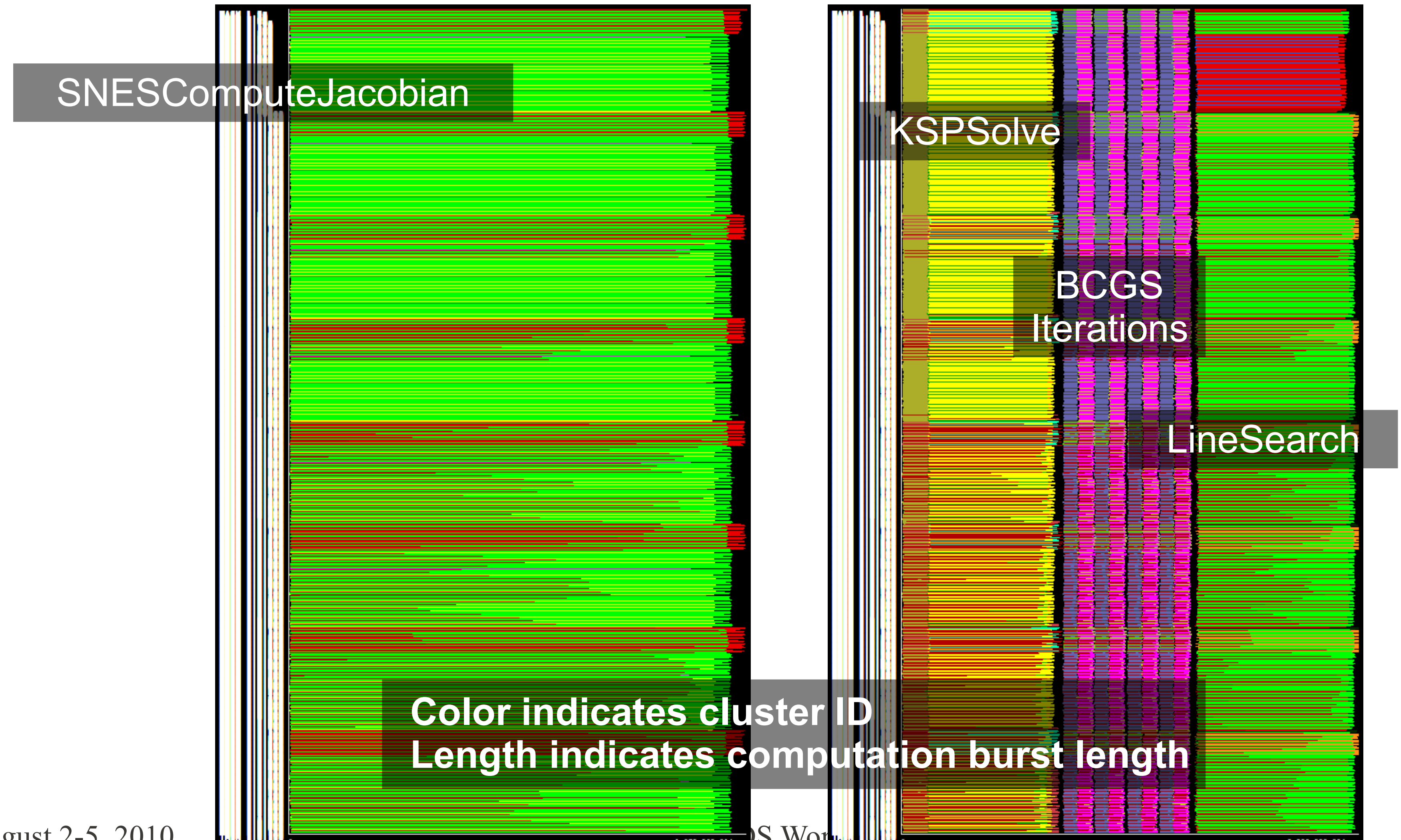
Stage	Procs	Solver	Total (sec)	Compute Avg	Compute Max	Compute Min	Comp. Avg/Max
Flow	8192	BCGS	10.21	40.47%	59.45%	**0.22%**	**0.68**
	8192	IBCGS	7.75	48.83%	55.33%	**0.58%**	**0.88**
	8184	IBCGS	7.23	47.43%	53.63%	**0.62%**	**0.88**
Tran	8192	BCGS	35.51	91.90%	99.04%	**9.37%**	**0.93**
	8192	IBCGS	36.76	91.78%	100.21%	**12.23%**	**0.92**
	8184	IBCGS	35.49	88.66%	99.55%	**10.37%**	**0.89**

Statistics from 1 selected iteration

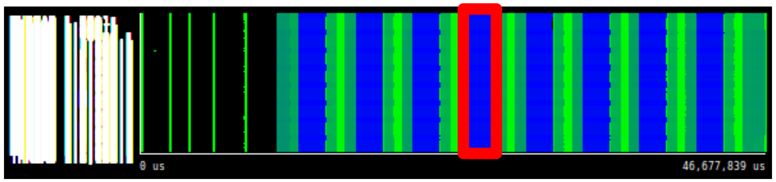
Jaguar – Clustering of 8k run

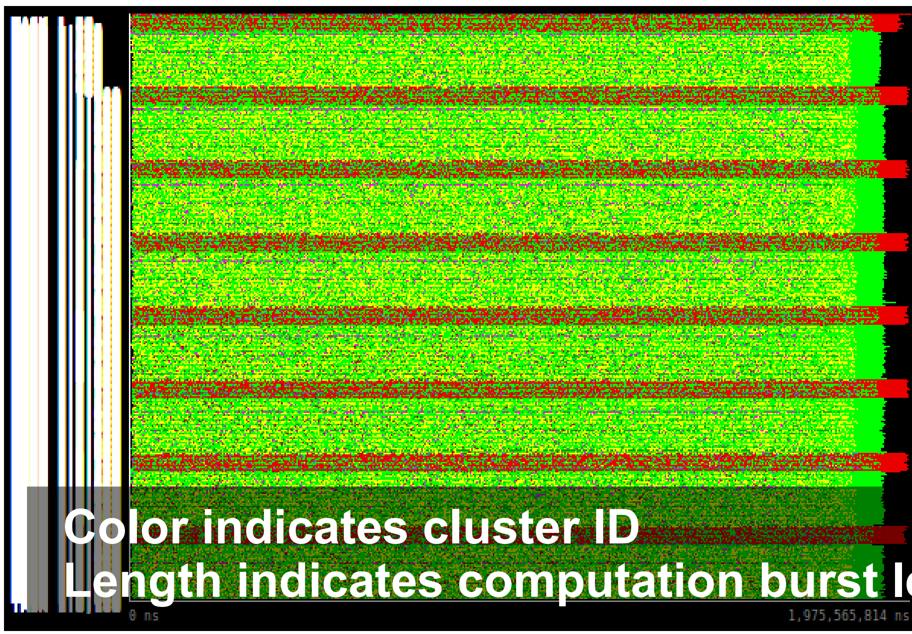


We chop one inner iteration from the Transport linear solver iterations, split it in half, and cluster each half

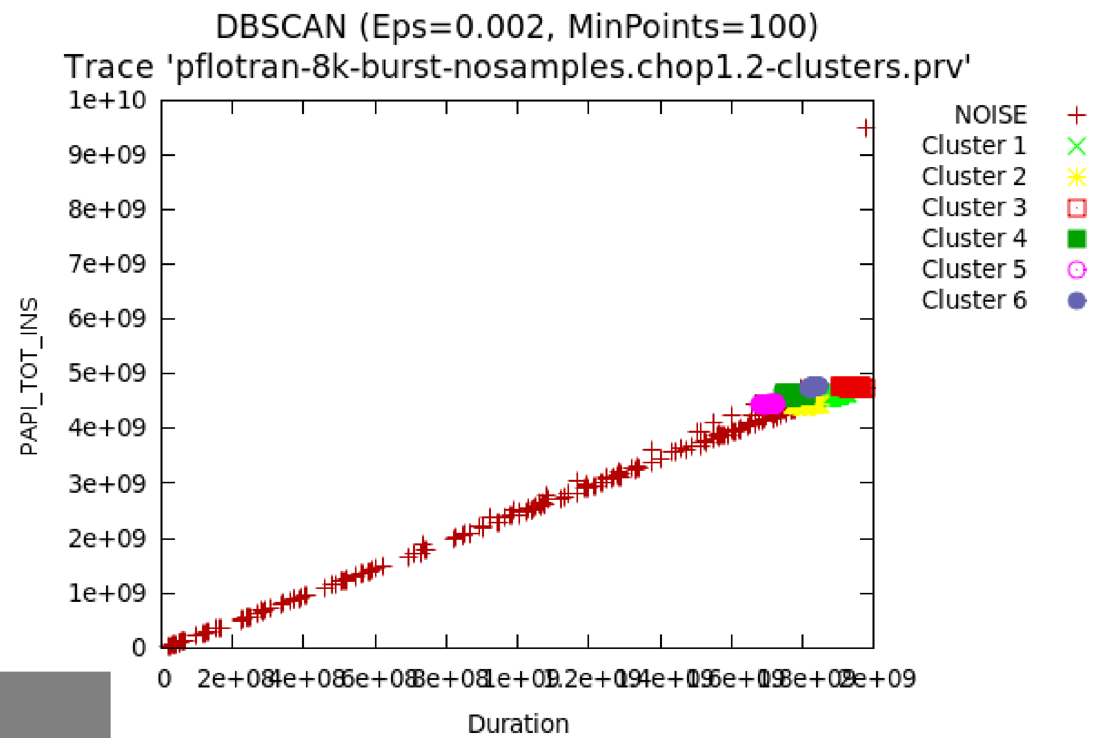


Jaguar – Clustering of Jacobian





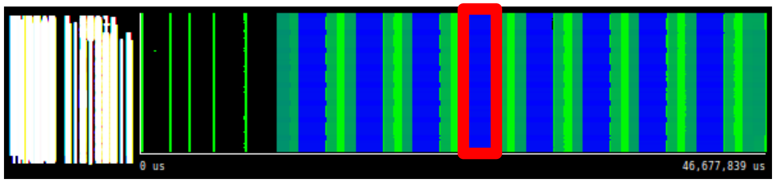
Color indicates cluster ID
Length indicates computation burst length

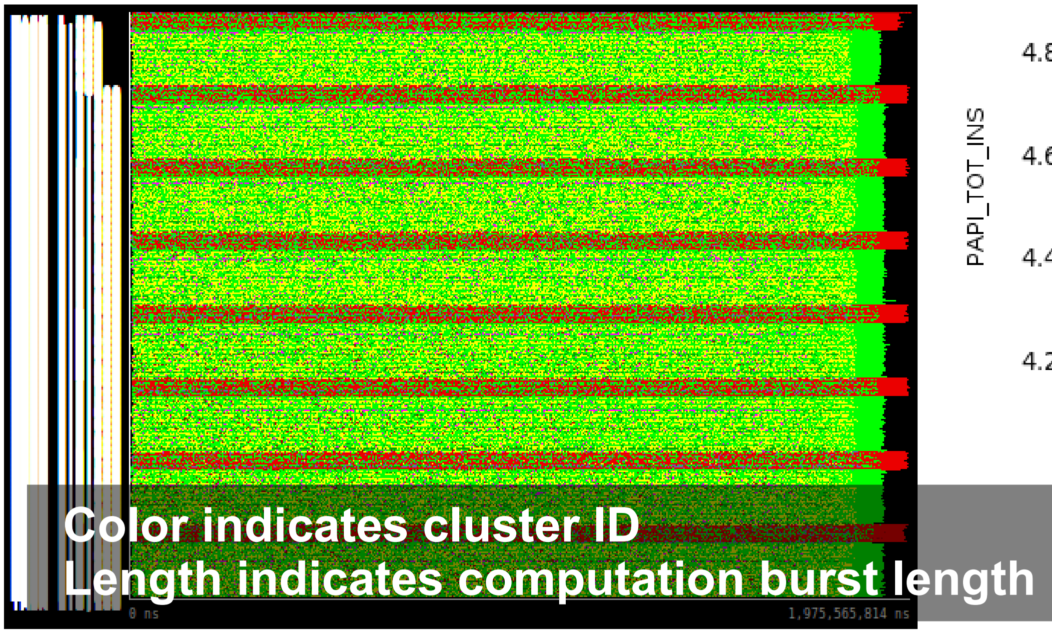


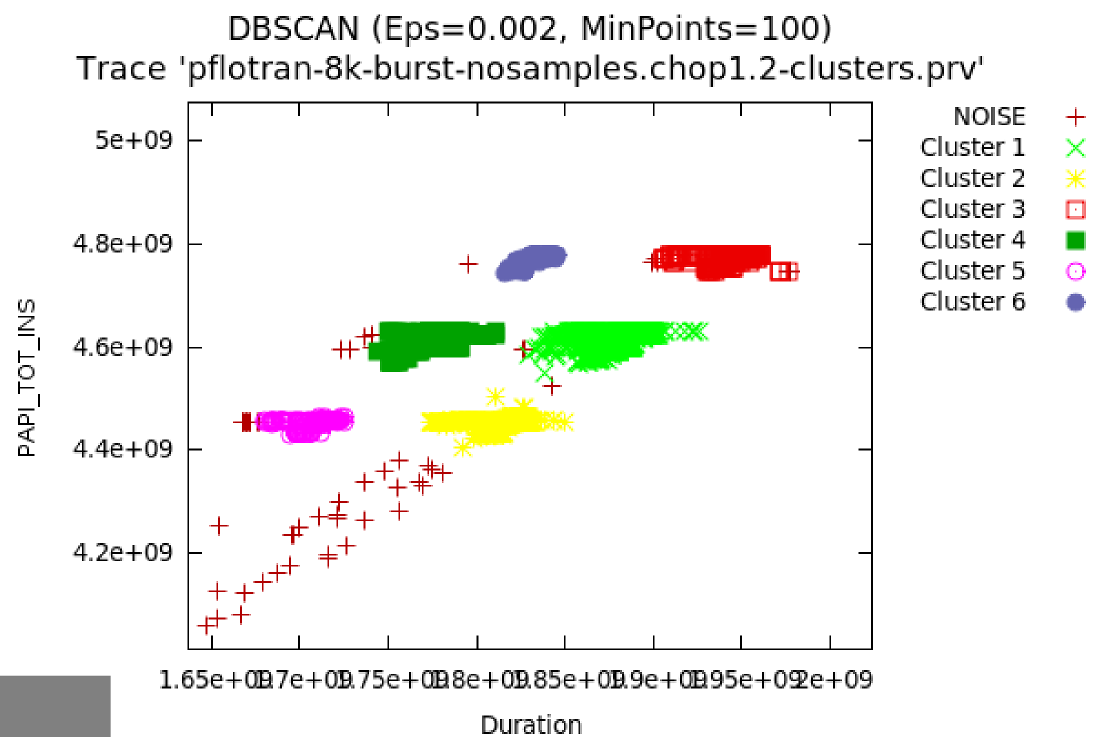
Outliers as small as ~0 seconds!

Cluster Name	NOISE	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Density	**237**	3685	2169	990	573	275	151
Total duration	2.70E+011	6.95E+012	3.94E+012	1.92E+012	1.02E+012	4.70E+011	2.77E+011
Avg. duration	1.14E+009	1.89E+009	1.82E+009	1.94E+009	1.78E+009	1.71E+009	1.83E+009
%Total duration	0.00%	47.66%	27.04%	13.20%	6.99%	3.22%	1.90%
mean TOT_INS	2.86E+009	4.62E+009	4.45E+009	4.77E+009	4.62E+009	4.45E+009	4.77E+009

Jaguar – Clustering of Jacobian



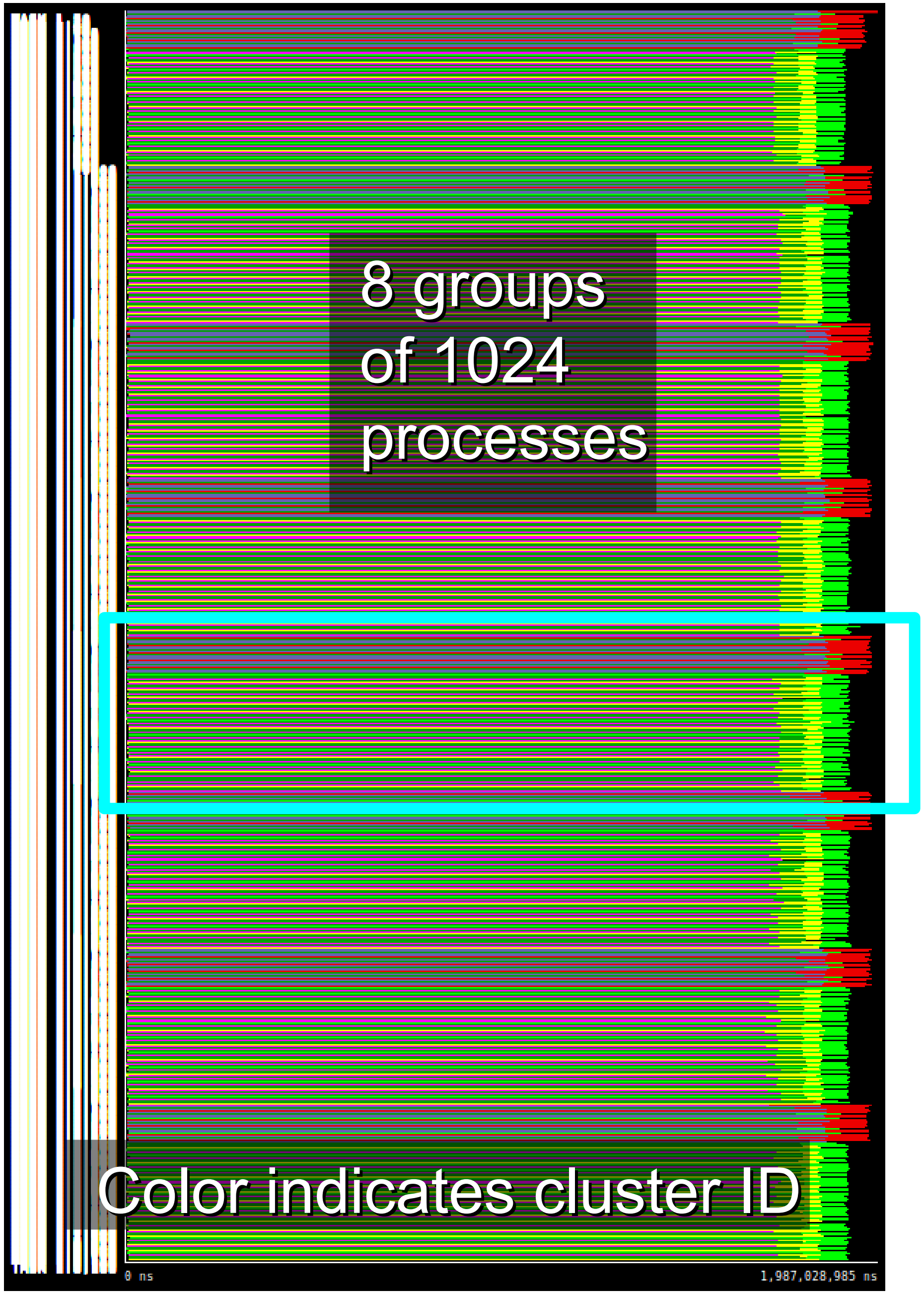


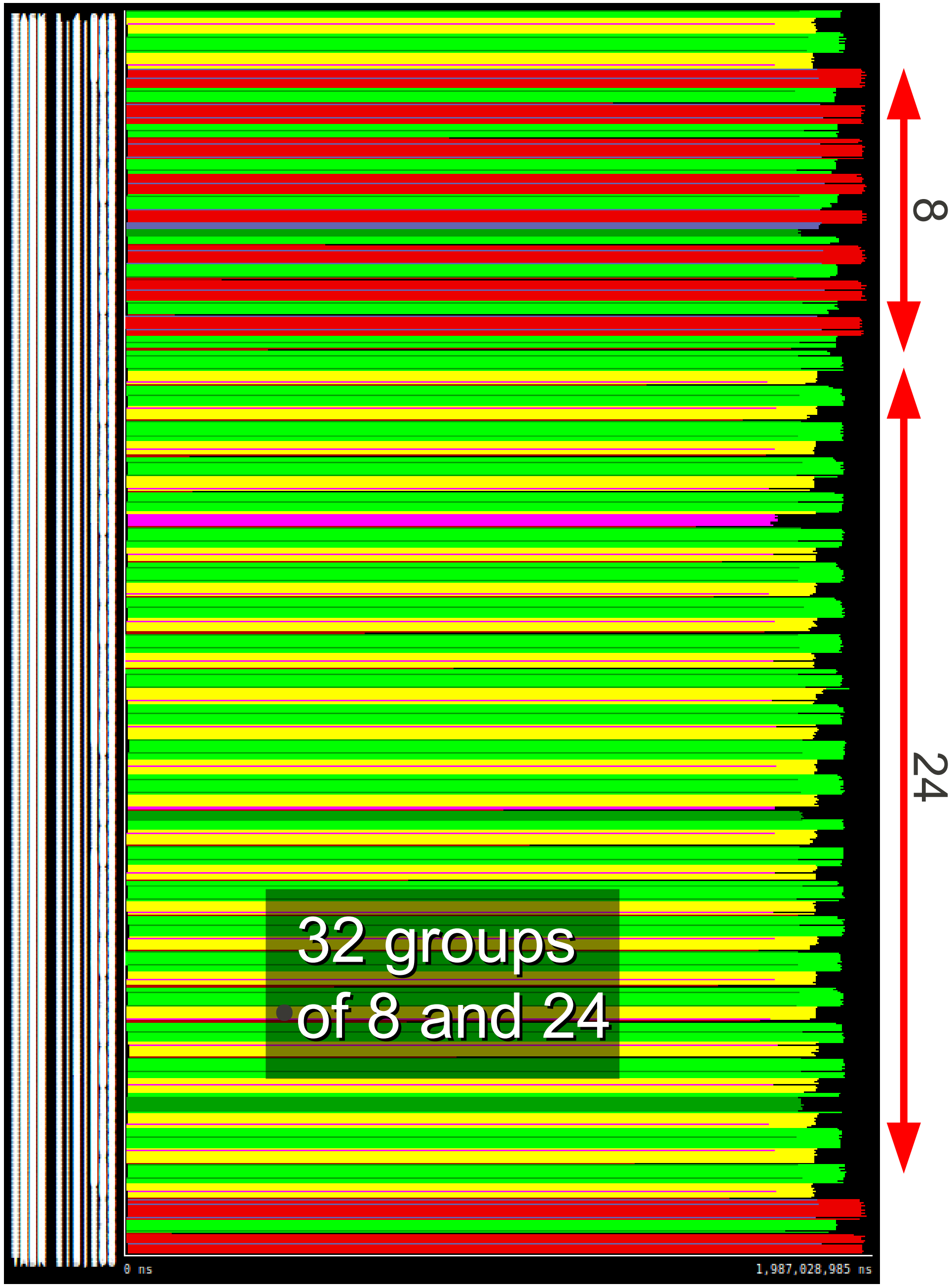


Range of ~1.66 to ~1.975 seconds

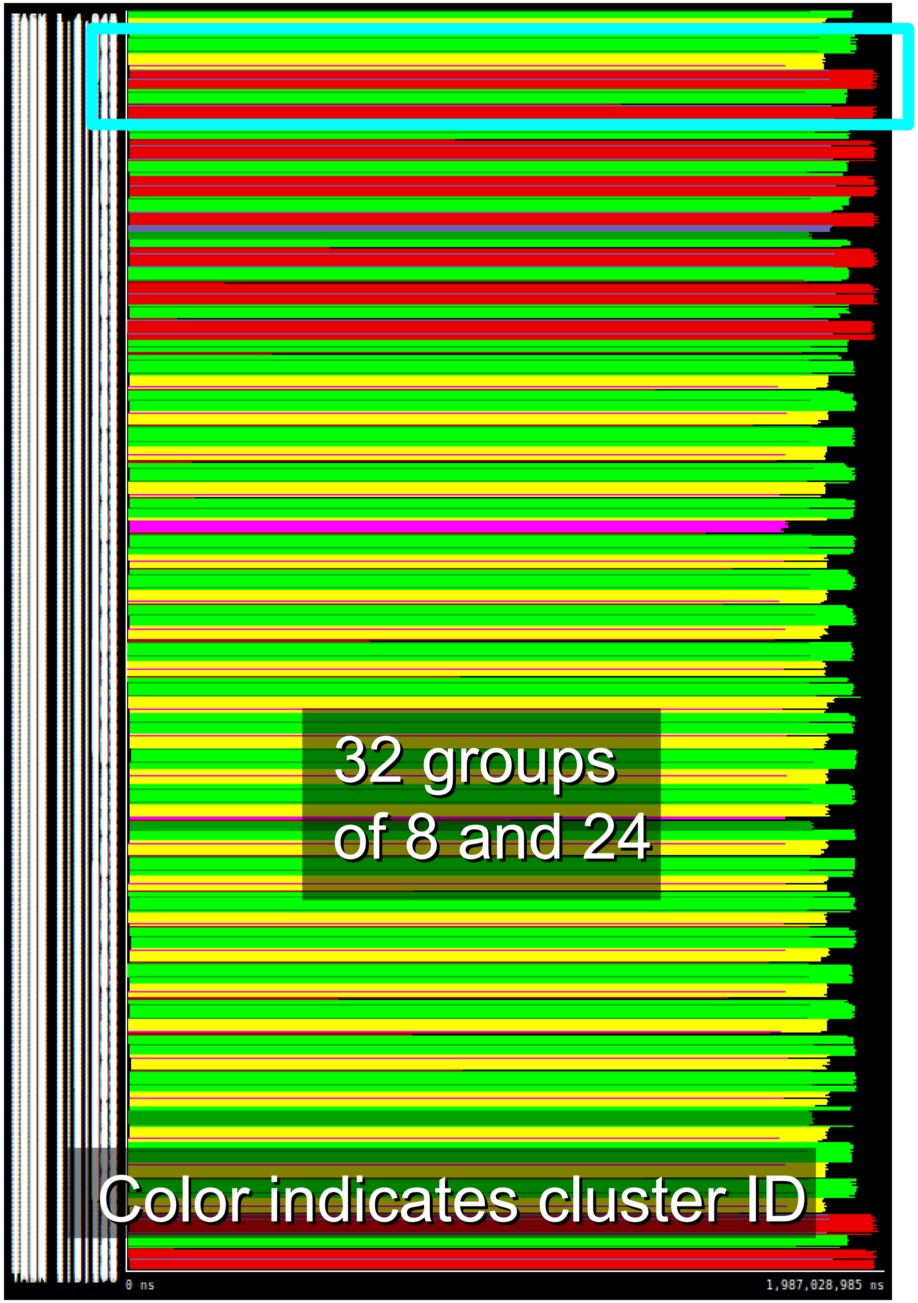
Cluster Name	NOISE	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Density	237	3685	2169	**990**	573	275	151
Total duration	2.70E+011	6.95E+012	3.94E+012	**1.92E+012**	1.02E+012	4.70E+011	2.77E+011
Avg. duration	1.14E+009	1.89E+009	1.82E+009	**1.94E+009**	1.78E+009	1.71E+009	1.83E+009
%Total duration	0.00%	47.66%	27.04%	**13.20%**	6.99%	3.22%	1.90%
mean TOT_INS	2.86E+009	4.62E+009	4.45E+009	**4.77E+009**	4.62E+009	4.45E+009	4.77E+009

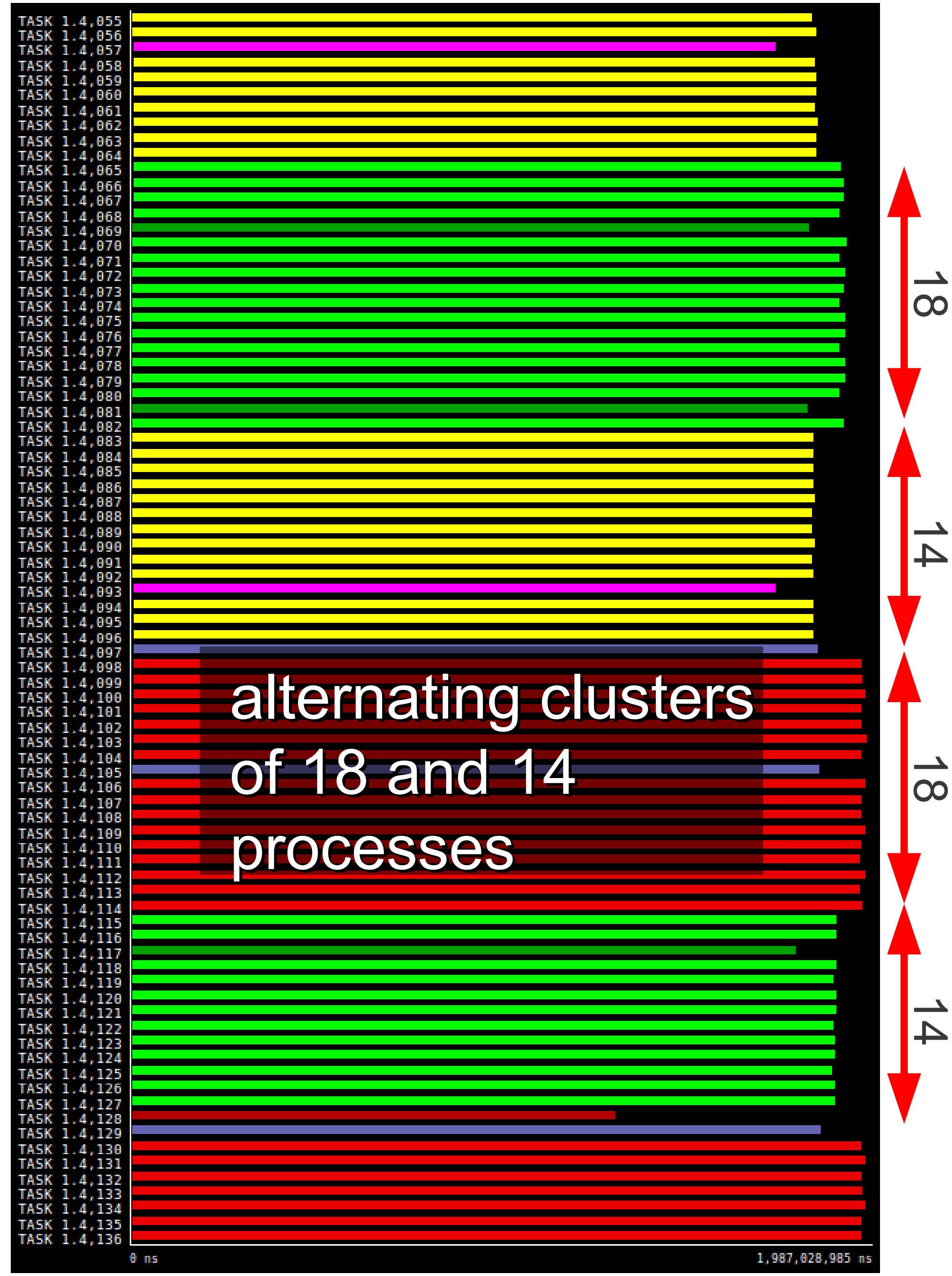
Clusters of Clusters – Z dimension



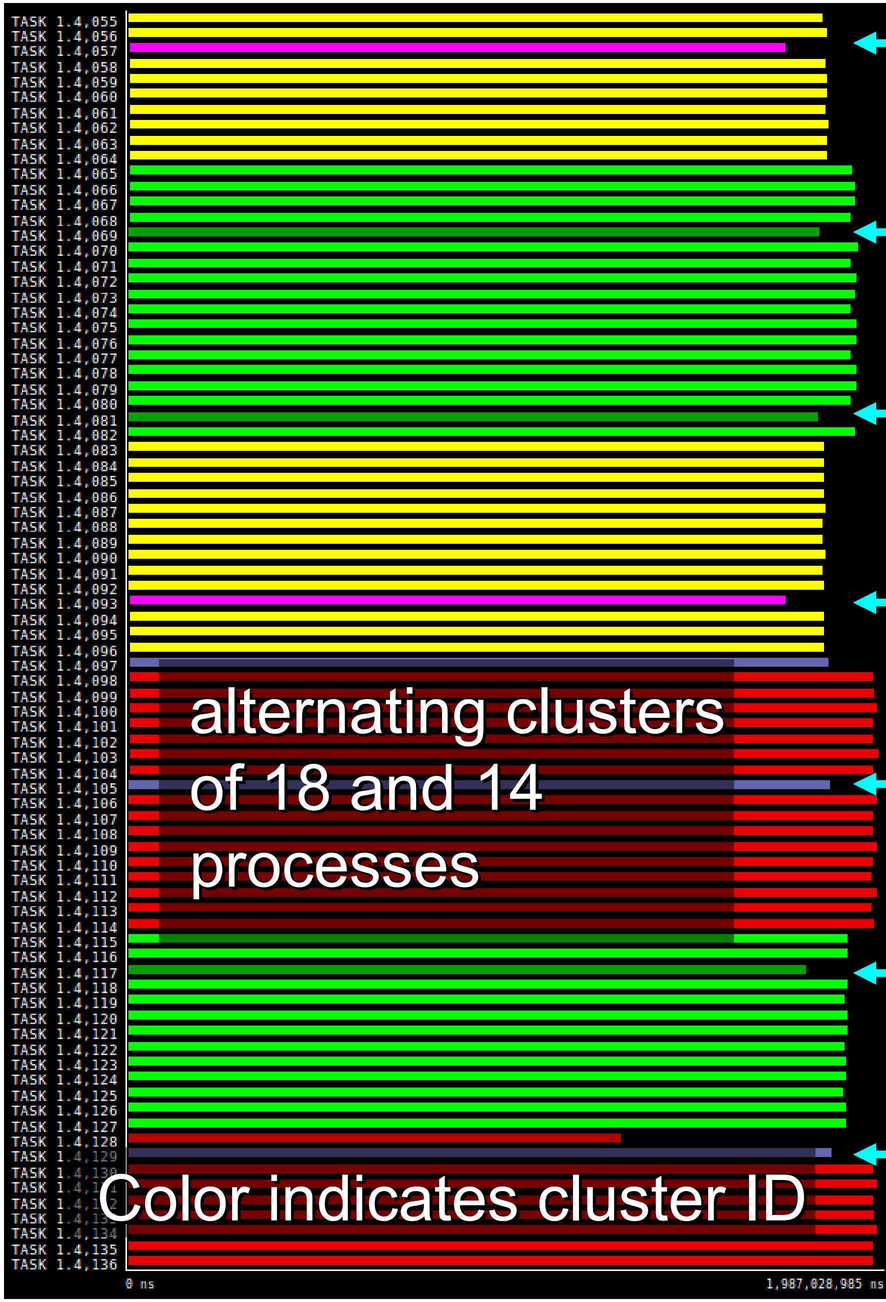


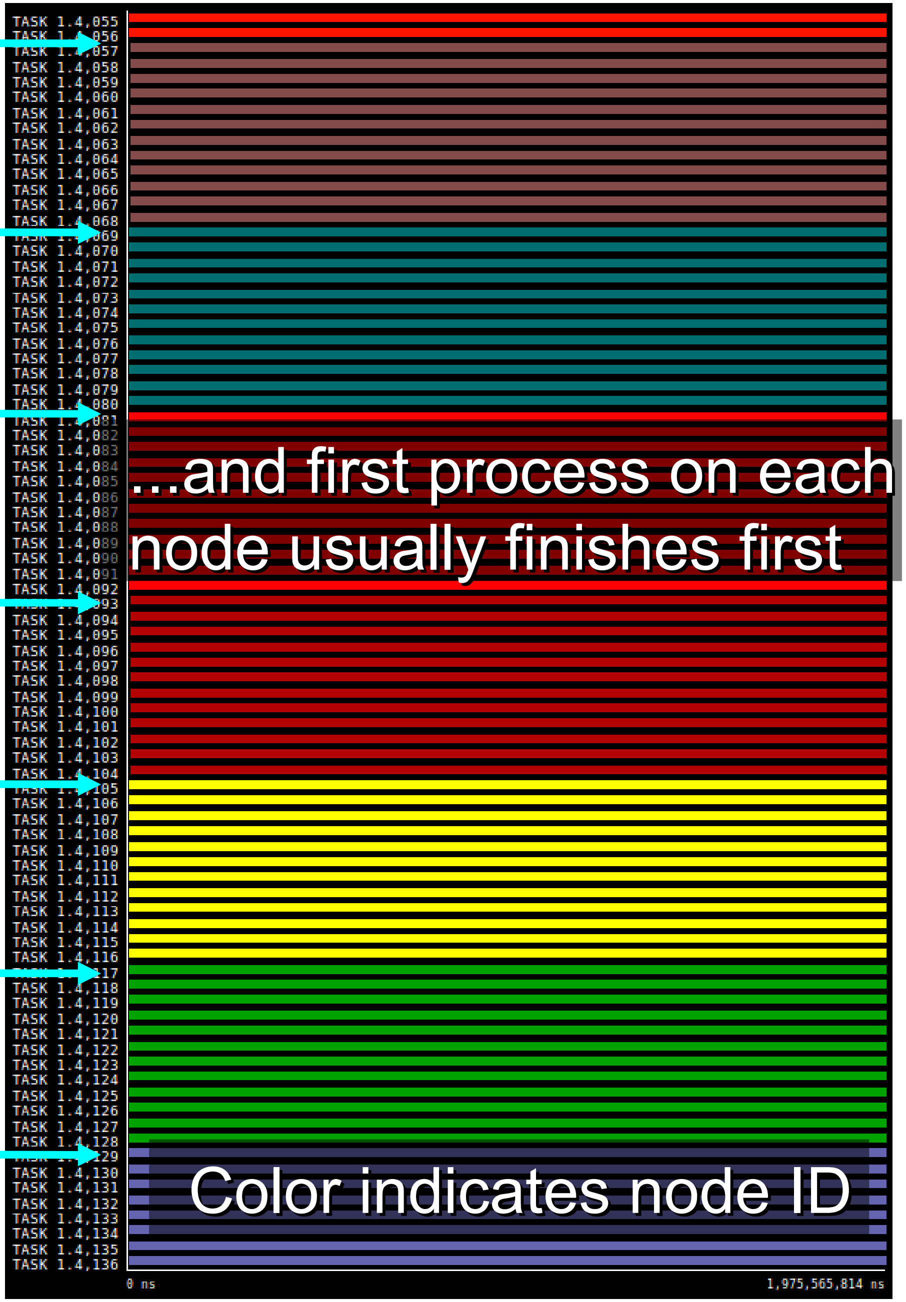
Clusters of Clusters – Y dimension



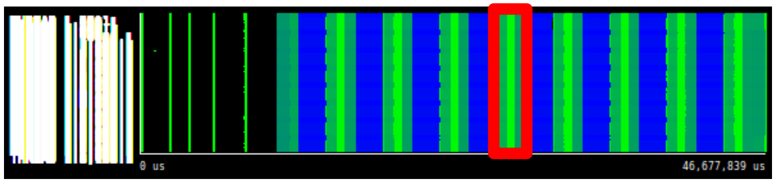


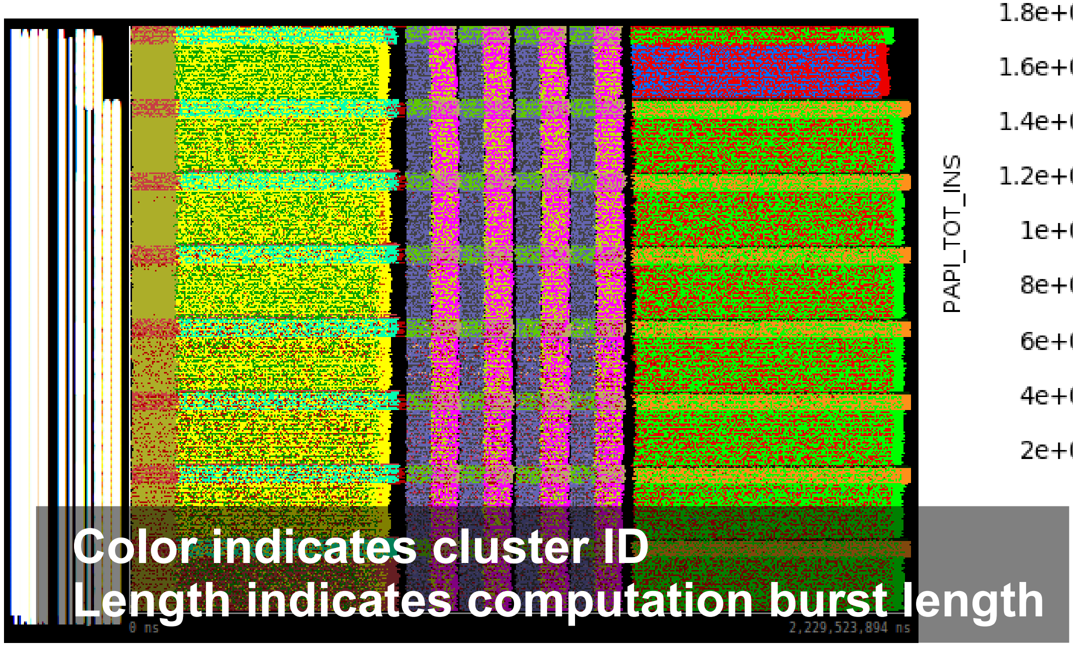
Clusters of Clusters – X dimension

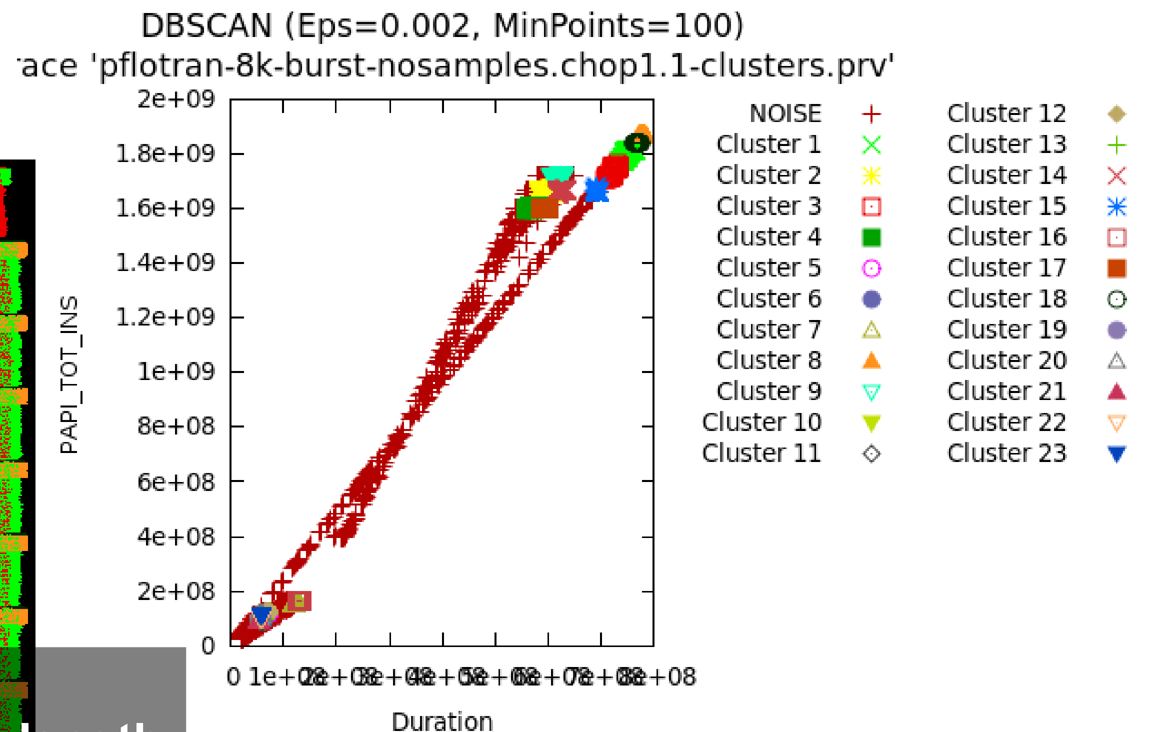




Clustering of KSPSolve



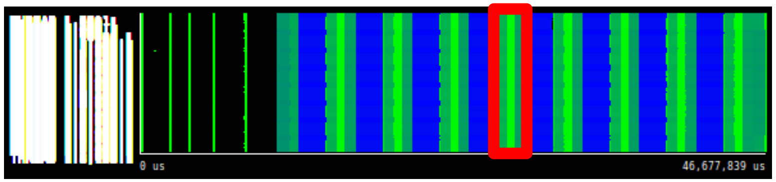


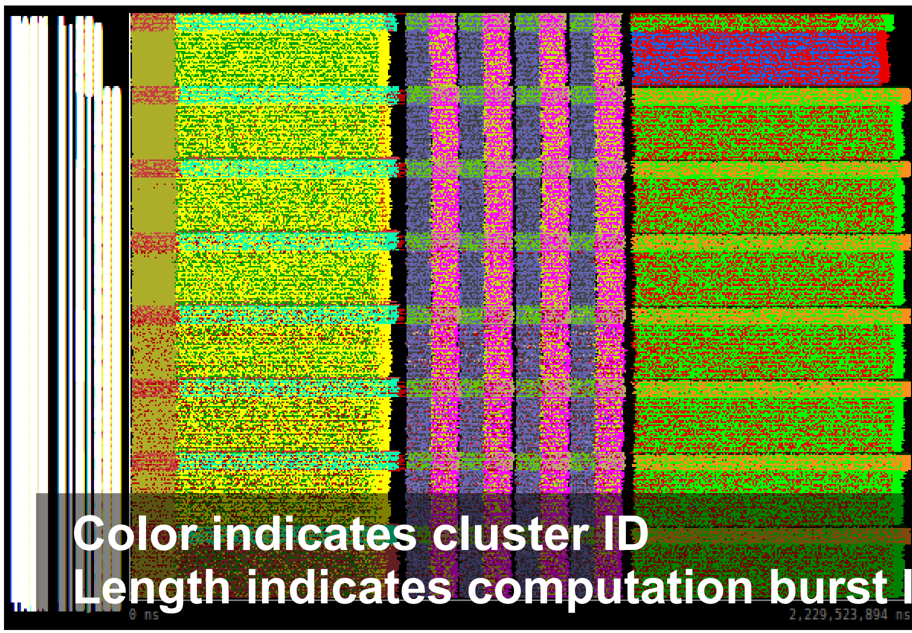


Outliers as small as ~0 seconds!

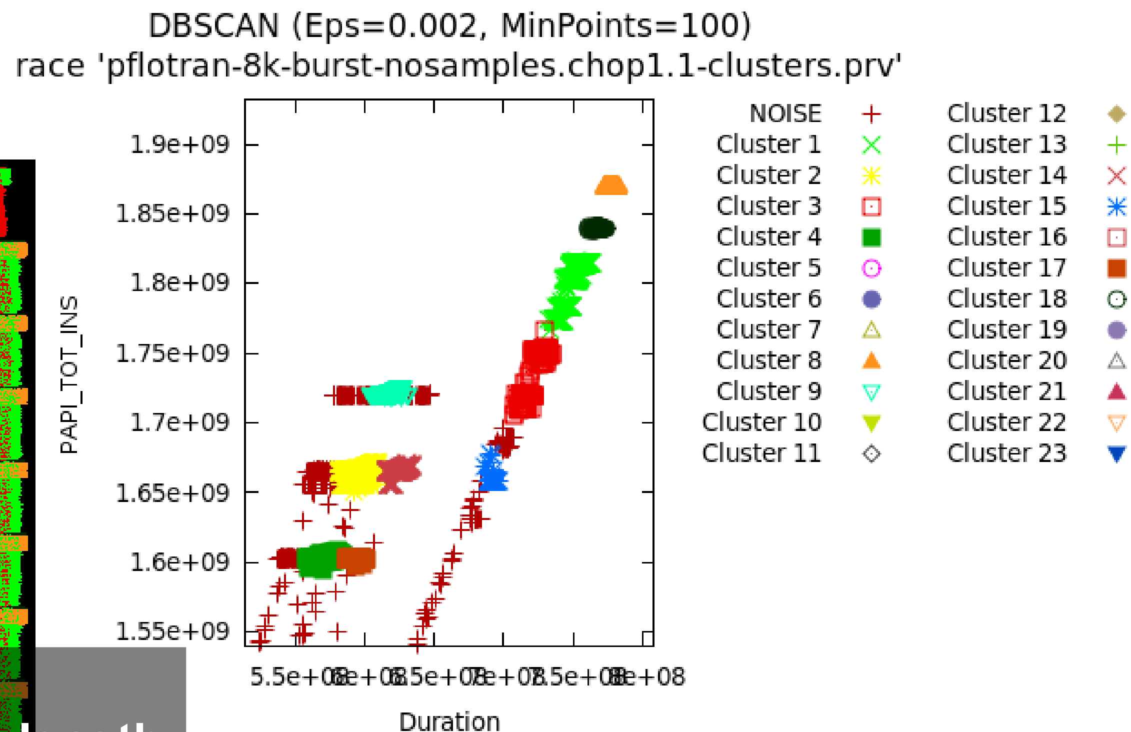
Cluster Name	NOISE	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
Density	4981	3849	3619	2660	2163	16007	16151	6281
Total duration	7.16E+011	2.90E+012	2.17E+012	1.93E+012	1.24E+012	1.10E+012	1.00E+012	7.82E+011
Avg. duration	1.44E+008	7.53E+008	5.99E+008	7.26E+008	5.74E+008	6.86E+007	6.22E+007	1.24E+008
%Total duration	0	0.19	0.14	0.13	0.08	0.07	0.07	0.05
PAPI_TOT_INS	3.39E+008	5.23E+008	1.81E+009	1.06E+007	1.66E+009	4.41E+006	1.74E+009	1.34E+007

Clustering of KSPSolve





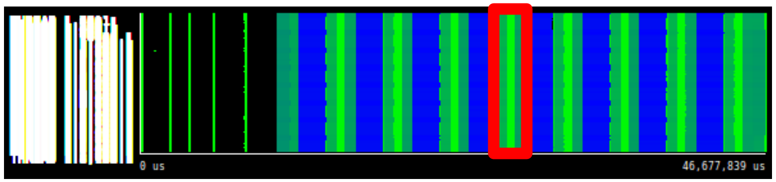
Color indicates cluster ID
Length indicates computation burst length

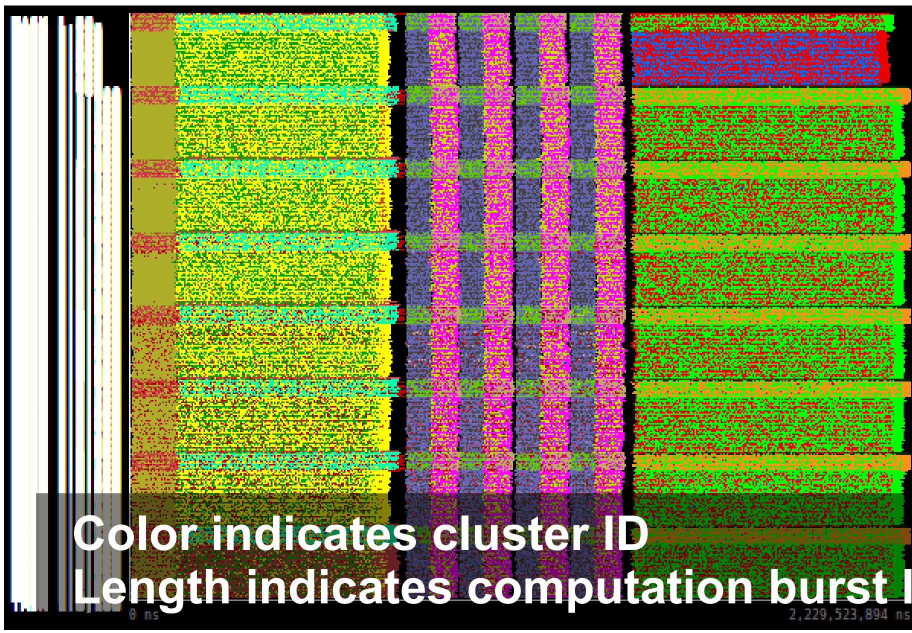


Outliers as small as ~0 seconds!

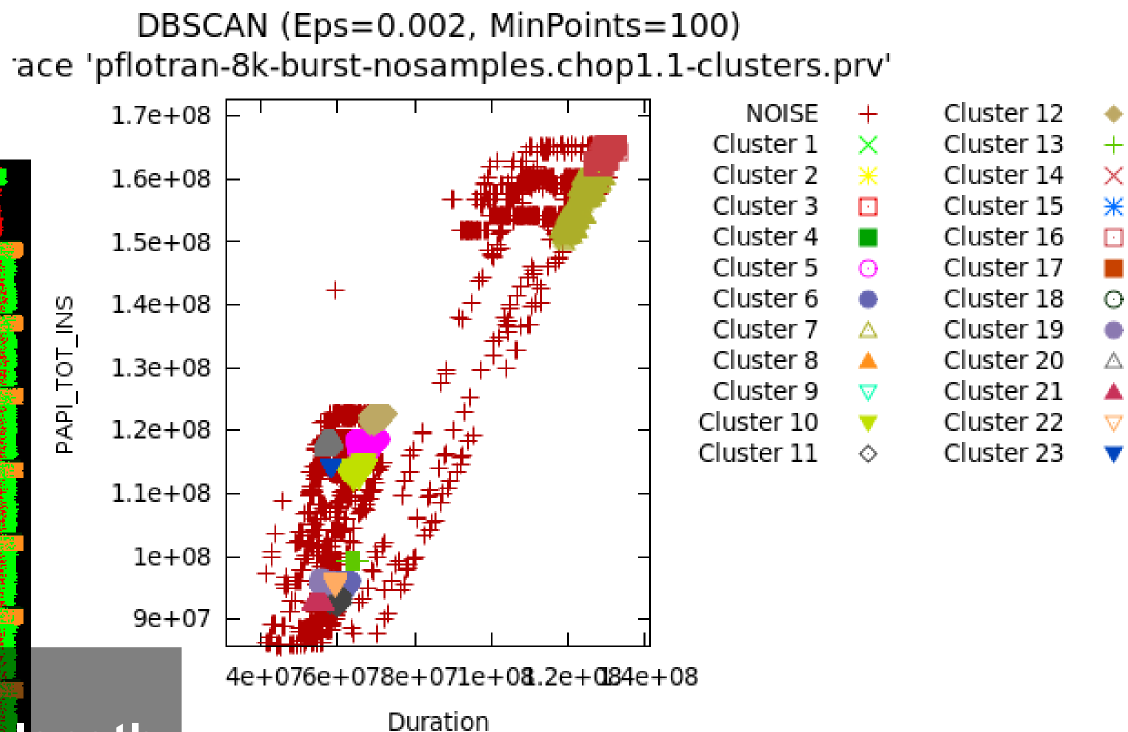
Cluster Name	NOISE	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
Density	4981	3849	3619	2660	2163	16007	16151	6281
Total duration	7.16E+011	2.90E+012	2.17E+012	1.93E+012	1.24E+012	1.10E+012	1.00E+012	7.82E+011
Avg. duration	1.44E+008	7.53E+008	5.99E+008	7.26E+008	5.74E+008	6.86E+007	6.22E+007	1.24E+008
%Total duration	0	0.19	0.14	0.13	0.08	0.07	0.07	0.05
PAPI_TOT_INS	3.39E+008	5.23E+008	1.81E+009	1.06E+007	1.66E+009	4.41E+006	1.74E+009	1.34E+007

Clustering of KSPSolve





Color indicates cluster ID
Length indicates computation burst length



Outliers as small as ~0 seconds!

Cluster Name	NOISE	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
Density	4981	3849	3619	2660	2163	16007	16151	6281
Total duration	7.16E+011	2.90E+012	2.17E+012	1.93E+012	1.24E+012	1.10E+012	1.00E+012	7.82E+011
Avg. duration	1.44E+008	7.53E+008	5.99E+008	7.26E+008	5.74E+008	6.86E+007	6.22E+007	1.24E+008
%Total duration	0	0.19	0.14	0.13	0.08	0.07	0.07	0.05
PAPI_TOT_INS	3.39E+008	5.23E+008	1.81E+009	1.06E+007	1.66E+009	4.41E+006	1.74E+009	1.34E+007

FLOW Stage

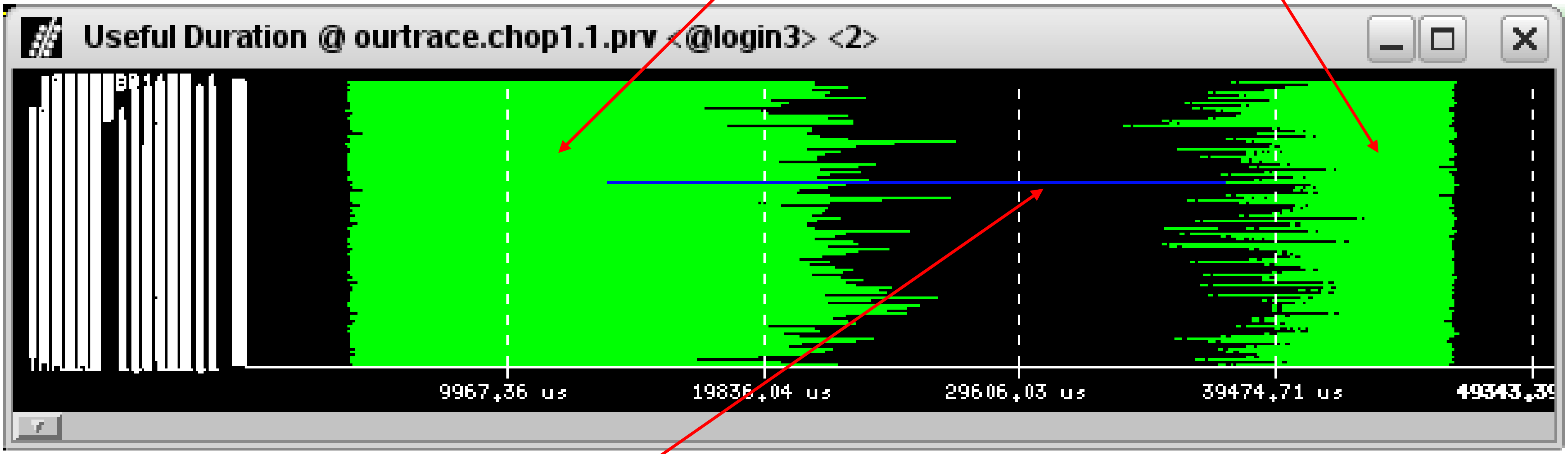
FLOW stage

- As also measured by PERI team, a lot of time is spent in MPI_Allreduce.
- Computation / Communication ratio is ~1
- **What is causing irregular iteration lengths in the Flow phase?**
  - Suspected process migration or preemption
- We collected a detailed trace of 1 iteration (9th iteration of 10 total) of a 8k process run to see if there is something causing long MPI_Allreduce times
  - 1 iteration of FLOW stage = ~45GB

Useful Duration Timeline View

Computing bursts

Color indicates length of computation burst between MPI calls



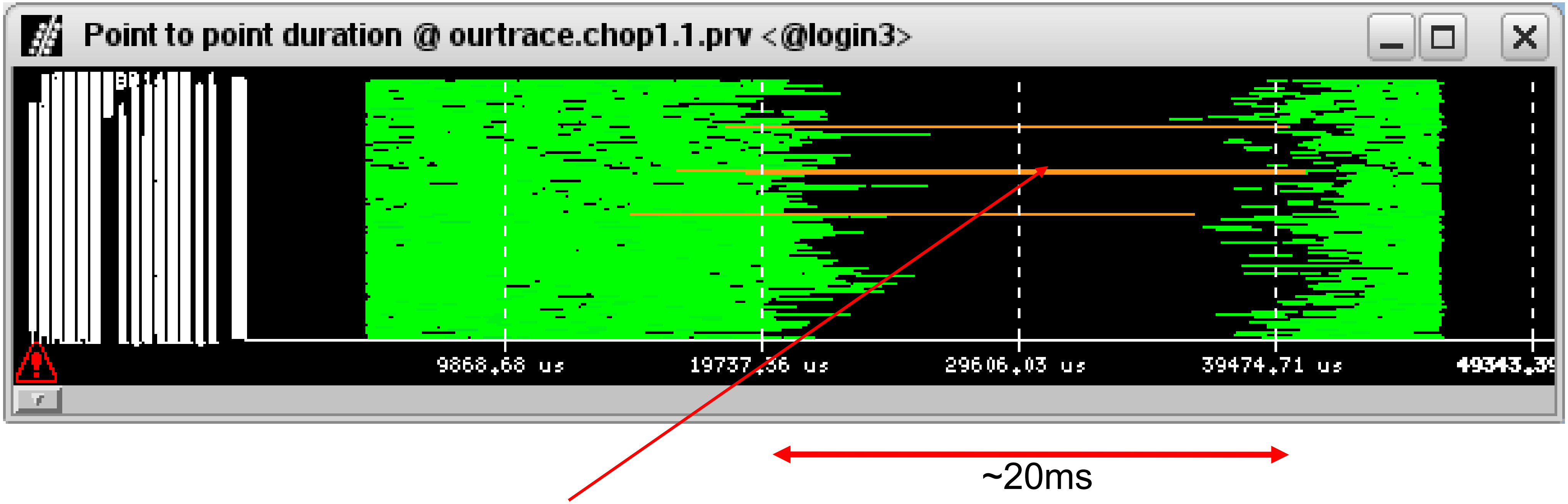
~20ms

Long collective communication delayed by single outlier(s)...

Point-to-Point Communication

Color indicates length of Point-to-point MPI call

...a few other processes are waiting...



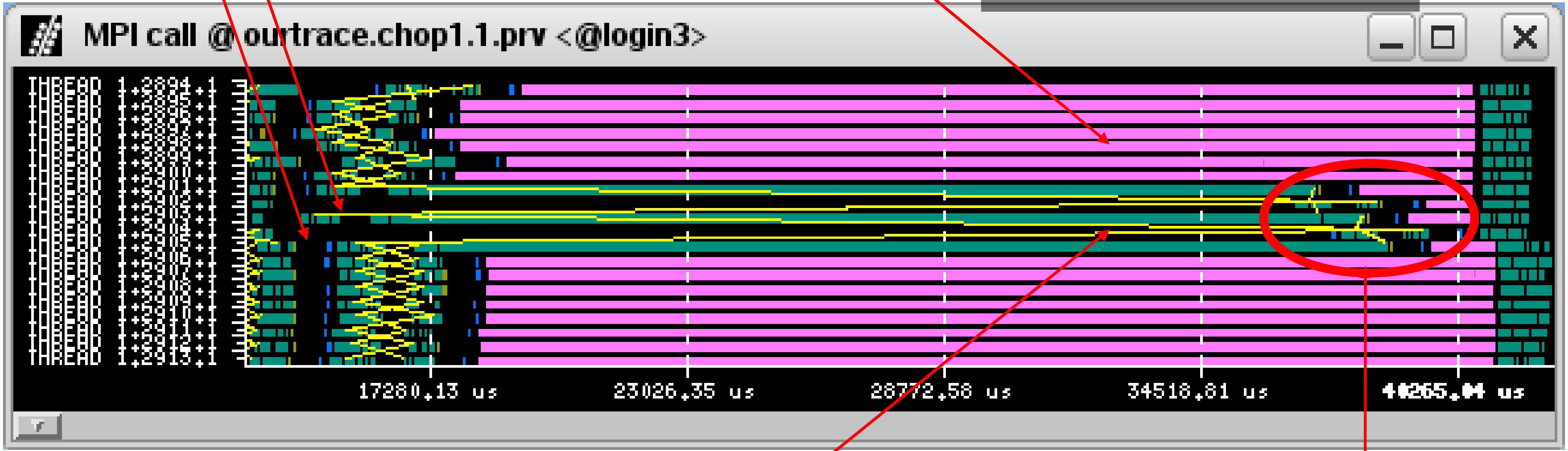
...which causes delays in MPI_Waitall()...

MPI Call View

Two delayed processes (black gaps)

All other processes are waiting in MPI_Allreduce() (pink)

Color indicates MPI call



Processes waiting in MPI_Waitall() (grey-green) (yellow lines are communication)

Five processes finally sync up

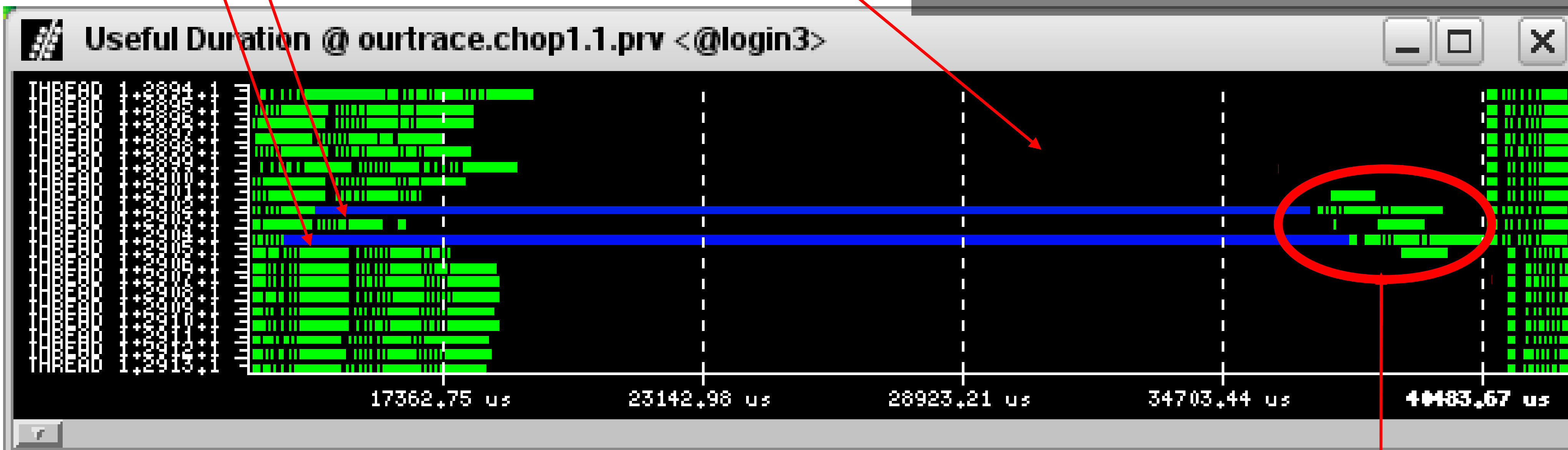
...and long delays in MPI_Waitall() result in long MPI_Allreduce() times

Useful Duration View (same zoom)

Two delayed processes (blue lines)

Processes waiting in MPI_Allreduce()

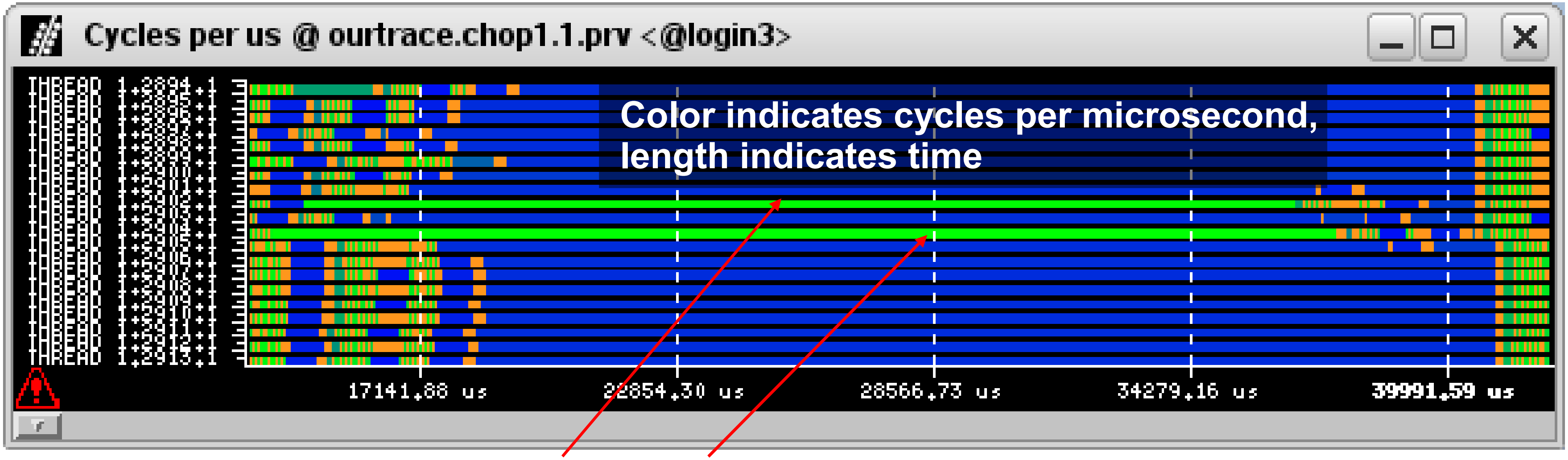
Color indicates length of computation burst between MPI calls



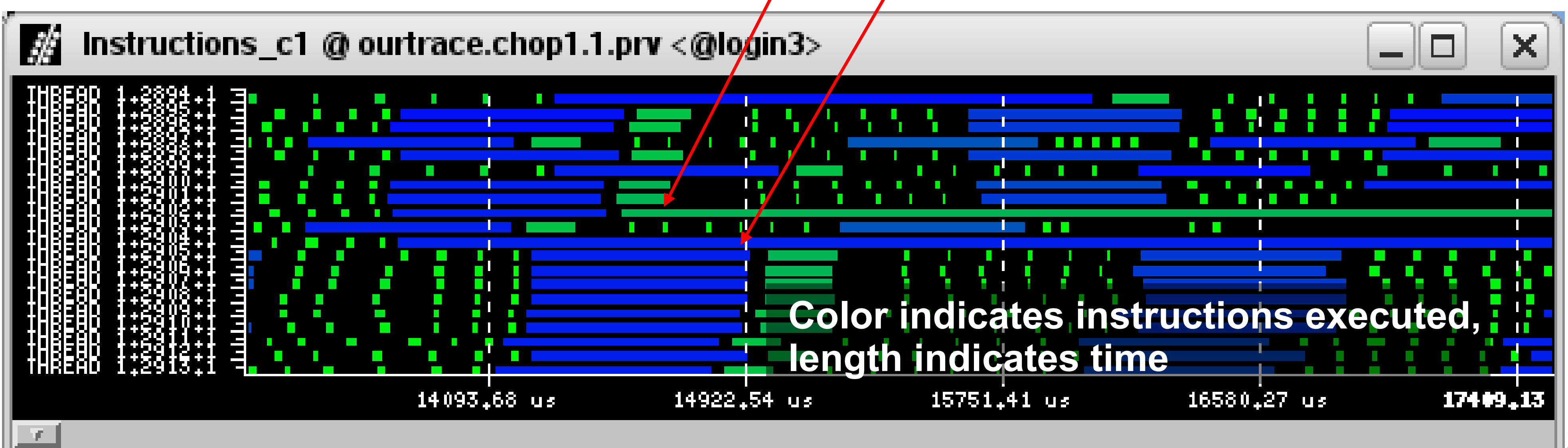
Five processes finally sync up

...but **WHY** are these two processes delayed?
Possibly a process migration or preemption problem...

Cycles per microsecond view



These two processes were allocated less cycles during these computation bursts, but executed the same instructions as their peers



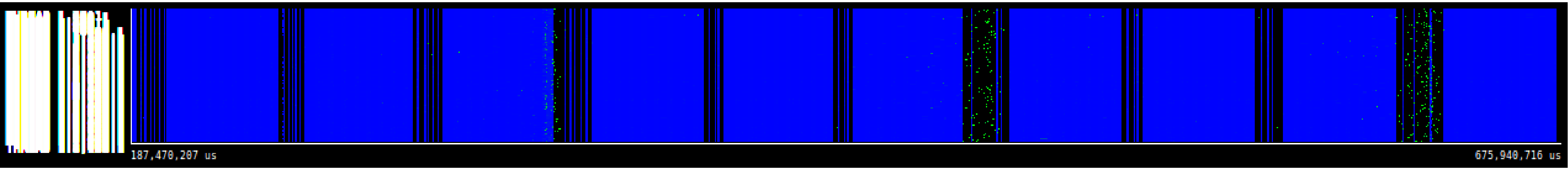
Further Investigation

- We theorized that the pre-emptions were not interrupts, but daemons or some kind of system process "starving" our MPI processes
- Could pinning the processes to the cores help?
- Three options: (none worked)
  - Pin to CPU
  - Pin to Core (default!)
  - Pin to Node (no pinning)
- What about leaving cores free for the OS?
  - Use only 11 cores
  - Use only 10 cores

"Reducing Application Runtime Variability On Jaguar XT5", Oral et al, CUG 2010

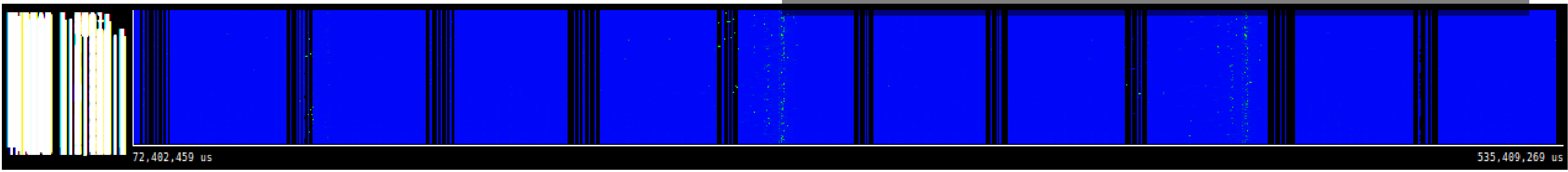
Pinning Results – 10 iterations

Default (pin to core) – 488 seconds

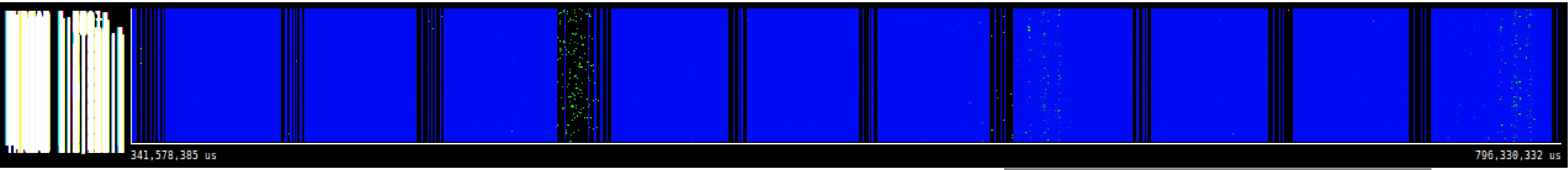


Explicit Pin to Core (“fastest”) – 463 seconds

Color indicates Cycles per microsecond

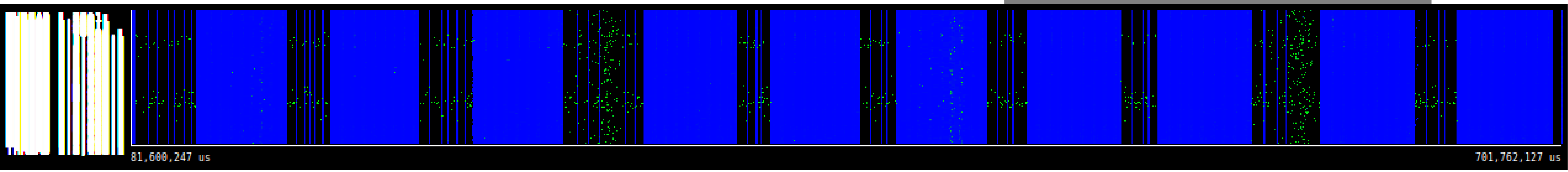


Pin to CPU (NUMA) – 455 seconds



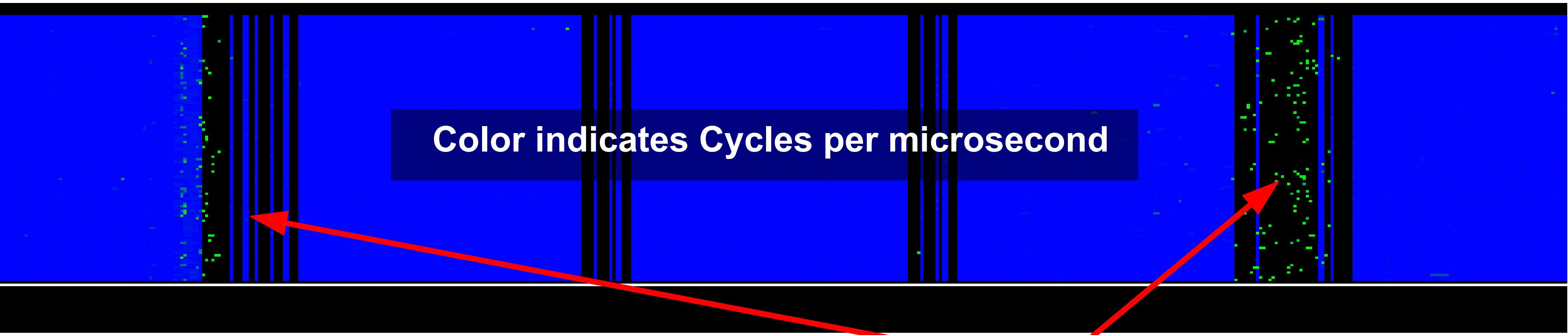
No pinning (slowest) – 620 seconds

(timelines not to scale)



Pinning – zoomed view

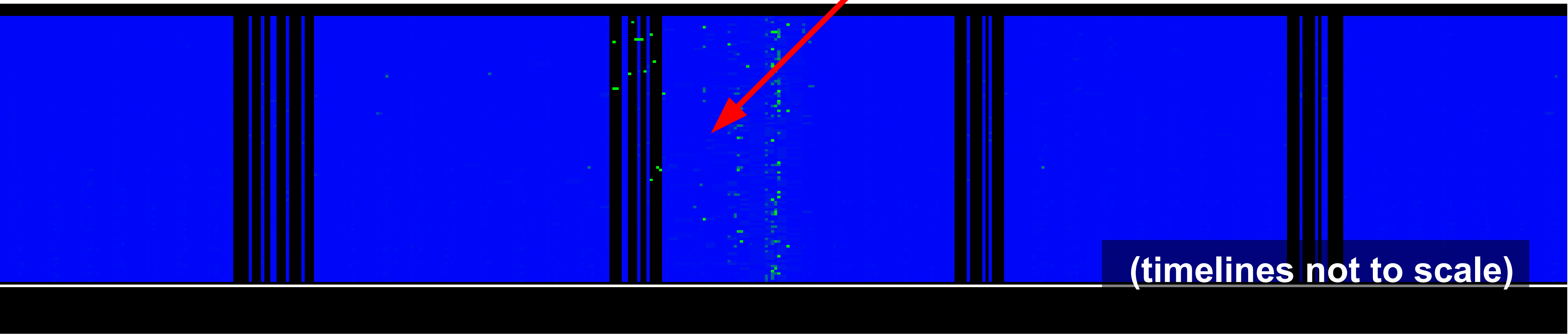
Default



Pre-emptions have significant effect In FLOW stage

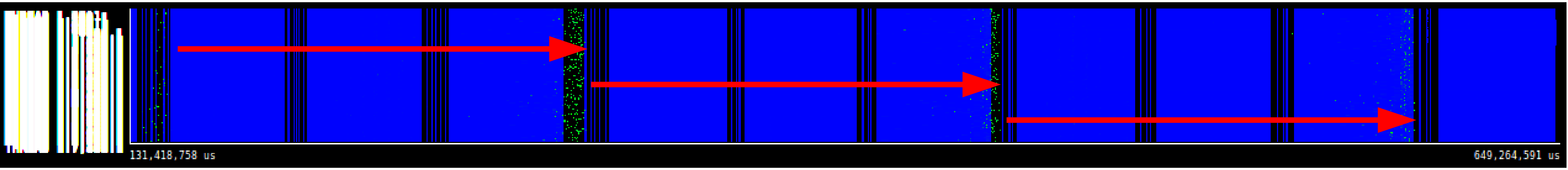
...but not in the TRAN stage

Pin to Core ("fastest")

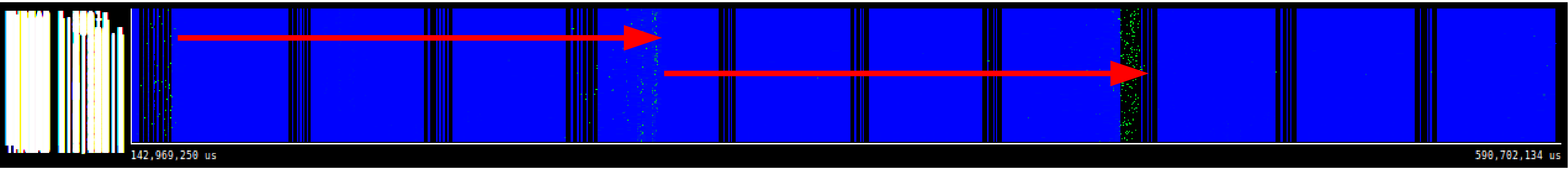


“Spare” core results – no improvement

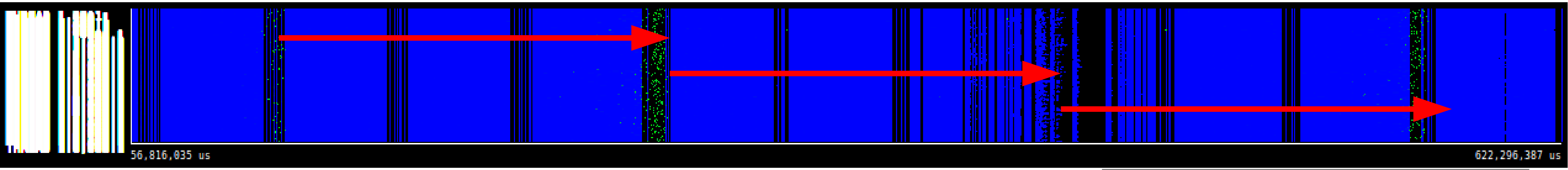
682 nodes, 7502 total cores – 538 seconds **150 Seconds!**



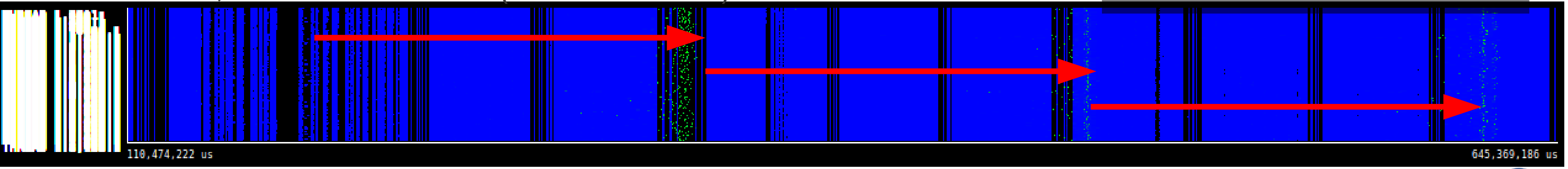
744 nodes, 8184 total cores – 448 seconds



682 nodes, 6820 total cores – 566 seconds



819 nodes, 8184 total cores (last 6 unused) – 536 seconds **(timelines not to scale)**



Pinning Results

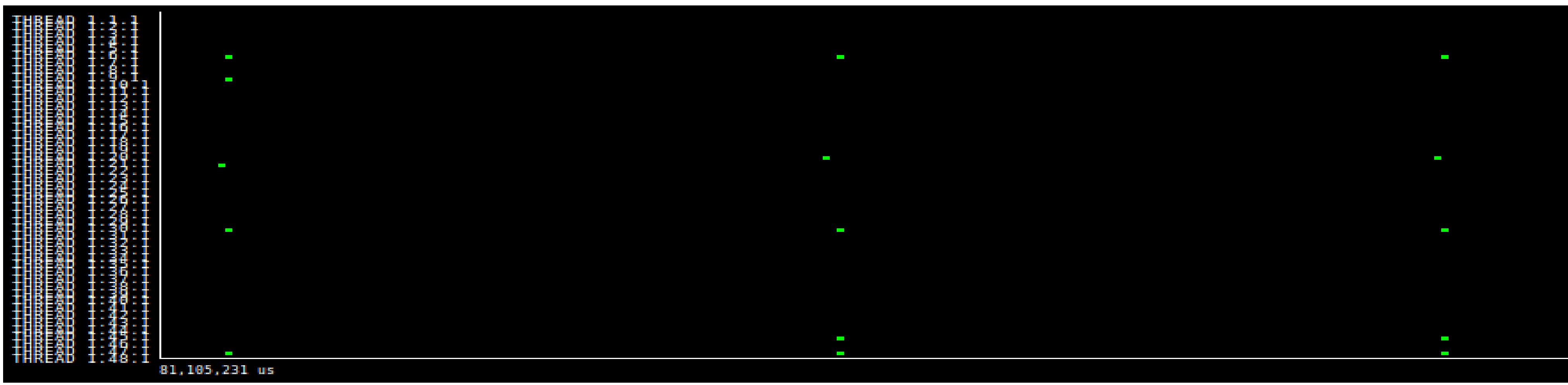
- Noise still occurred... but an insight was gained
- Clouds of noise occur every ~150 seconds
- Starts gradually, all nodes participate, then stops
- Always happened, regardless of whether there were extra cores or not – does not happen on the same core of each node
- Dramatic effect on runtime when noise was synchronized with FLOW stage of the iteration

Simple Kernel to Reproduce Problem

- We tried to reproduce preemptive behavior with a simple test
- 4 MPI_Start() calls, 1 MPI_Startall() call followed by a short computation, 4 MPI_Waitany() calls and 1 MPI_Waitall() call, followed by a short computation and an MPI_Allreduce() call – all repeated 835 times
- 48 Processes laid out in 2D grid, communication is with four neighbors

Simple Program Timeline

5 preemptions of >9ms every 150 seconds



150 seconds between preemptions

...not just *possible* to happen, but *guaranteed(?)* to happen on every node, every 150 seconds

Next Steps:

- Experiments with High Performance Linux (HPL)
- Modified current Linux kernel scheduler
  - Real Time Class
  - **HPC Class (new)**
  - Normal Class
  - Idle Class
- If an HPC process or thread is ready to run, it is given priority over all other normal tasks, including OS
- Improvement over standard scheduler, which is biased towards interactive responsiveness, not batch

“A Global Operating System for HPC Clusters”, Betti et al., CLUSTER 2009
“Designing OS for HPC Applications: Scheduling”, Gioiosa et al., draft

Conclusion

- Tran
  - Load balance can be improved
  - Clustering of bursts shows structure of load imbalance
    - Structure of clustering related to data decomposition
    - Many underutilized “noise” points
  - Recommend better decomposition strategy
  - Try StarSs/SMPSuperscalar* implementation – load balanced parallelism through runtime data flow analysis
- Flow
  - The solver is too synchronous for larger scales
  - Is there a potential for overlap – can an improved solver overlap communication and computation?
  - Large effect from system noise resonance

*http://www.bsc.es/plantillaG.php?cat_id=385

BSC Performance Tools

- For more information, contact:
  - tools@bsc.es
- To contact the presenter:
  - kevin.huck@bsc.es
- To download the open-source BSC Performance Tools:
  - http://www.bsc.es/paraver, select the “Downloads” link