Aftermath:
Interactive Visualization of Cross-Layer Performance Anomalies in Dynamic Task-Parallel Applications and Systems

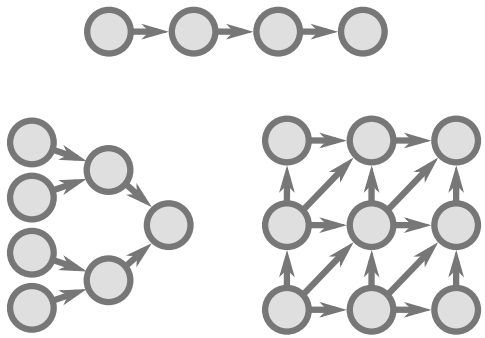
Andi Drebes

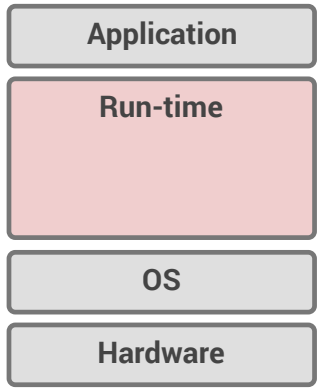
The University of Manchester
School of Computer Science
Advanced Processor Technologies
andi.drebes@manchester.ac.uk

Joint work with:
Antoniu Pop, Karine Heydemann, Albert Cohen

ISPASS 2016

Task-Parallel Programming

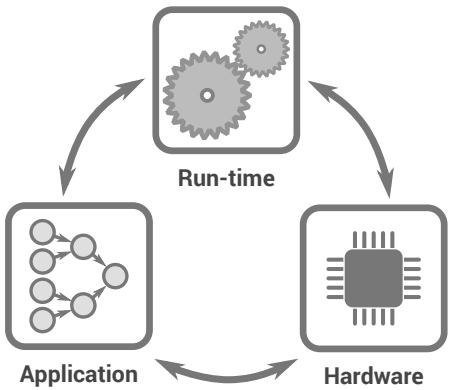




Principles of task-parallel programming

- Small units of work (tasks)
- Inter-task data dependences
- Expose large amounts of parallelism
- Efficient execution by a run-time system
- OpenMP 4, StarSs, OpenStream, . . .

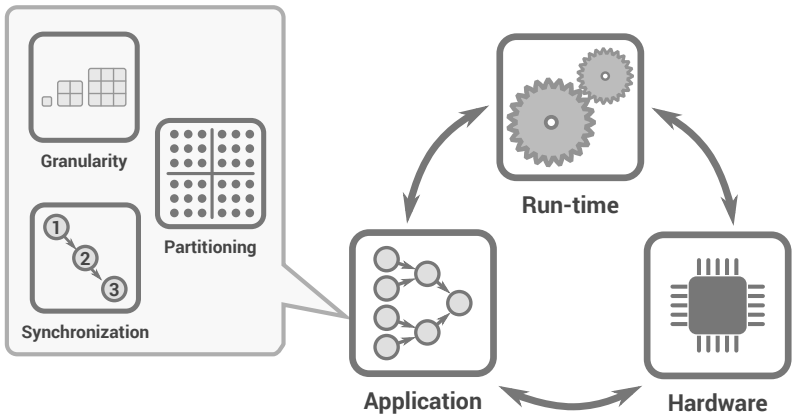
Performance of Task-Parallel Programs



Performance analysis difficult

- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

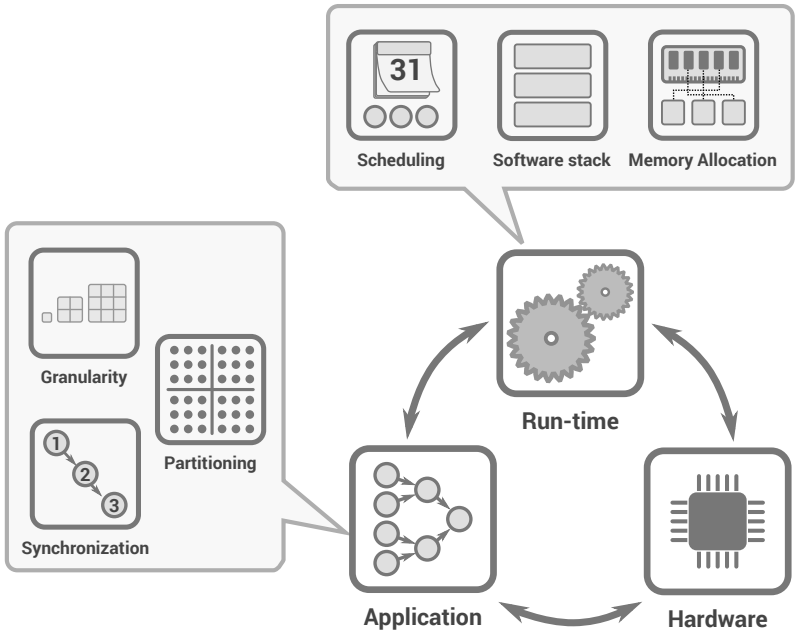
Performance of Task-Parallel Programs



Performance analysis difficult

- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

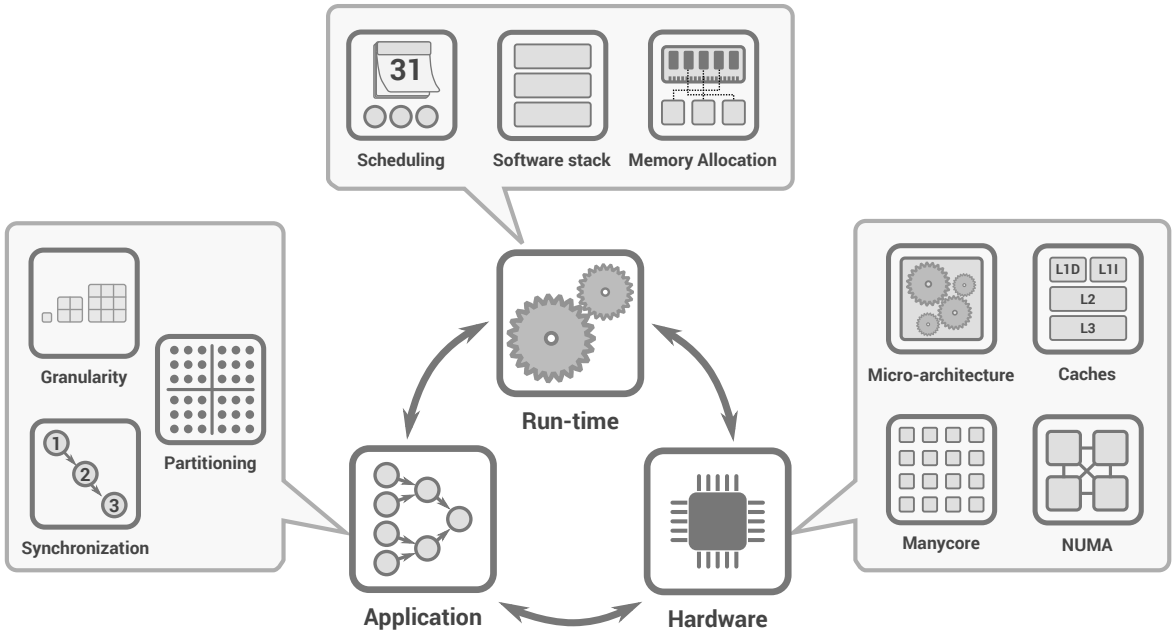
Performance of Task-Parallel Programs



Performance analysis difficult

- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

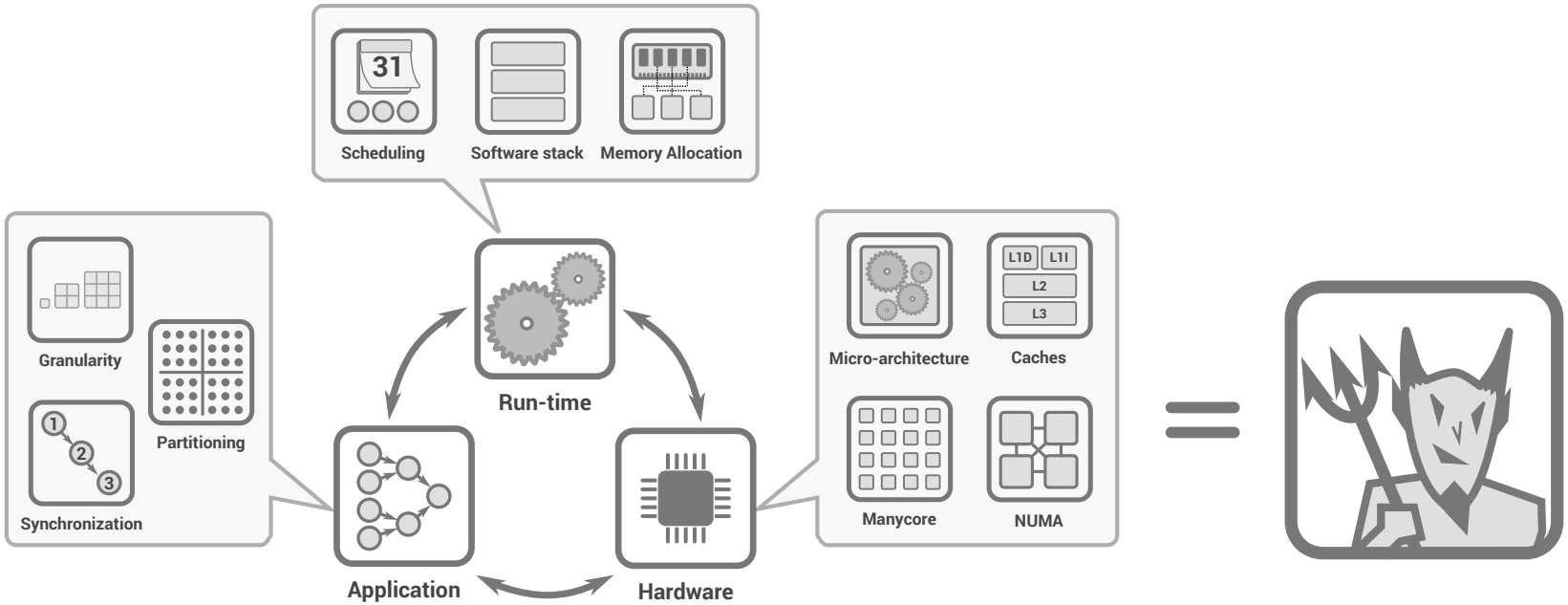
Performance of Task-Parallel Programs



Performance analysis difficult

- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

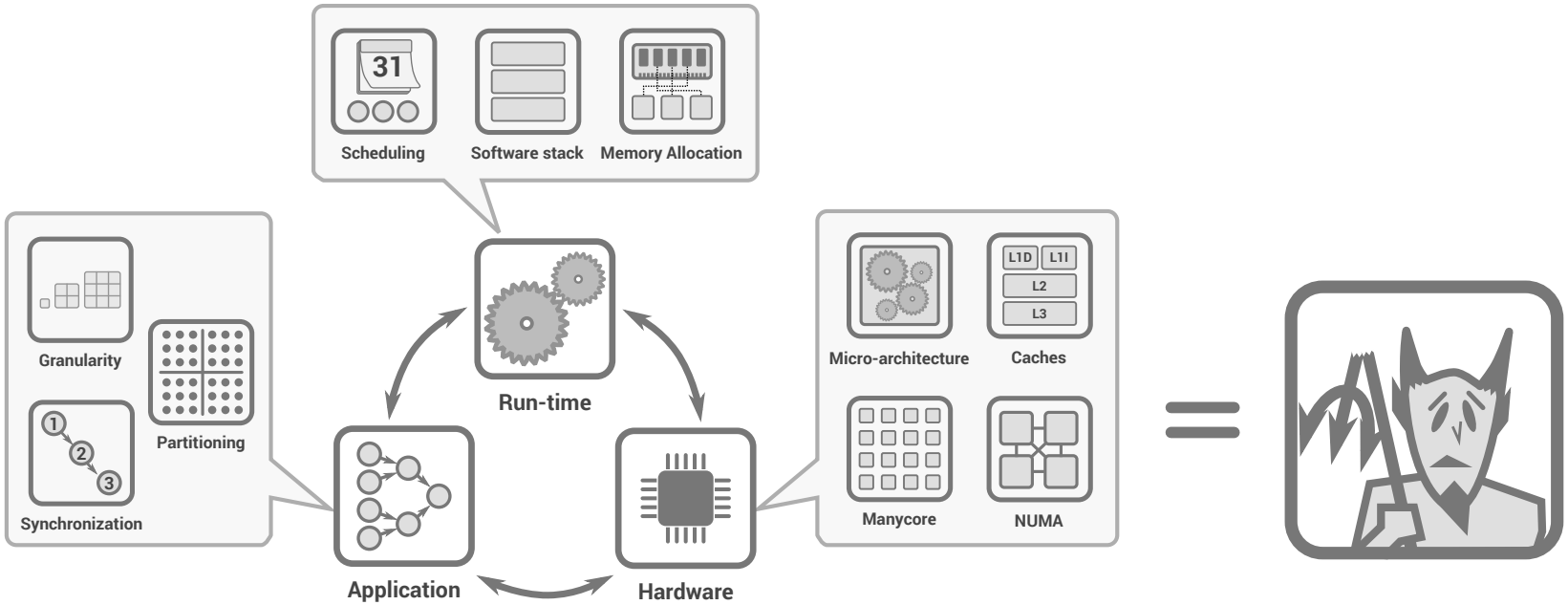
Performance of Task-Parallel Programs



Performance analysis difficult

- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

Performance of Task-Parallel Programs



Performance analysis difficult

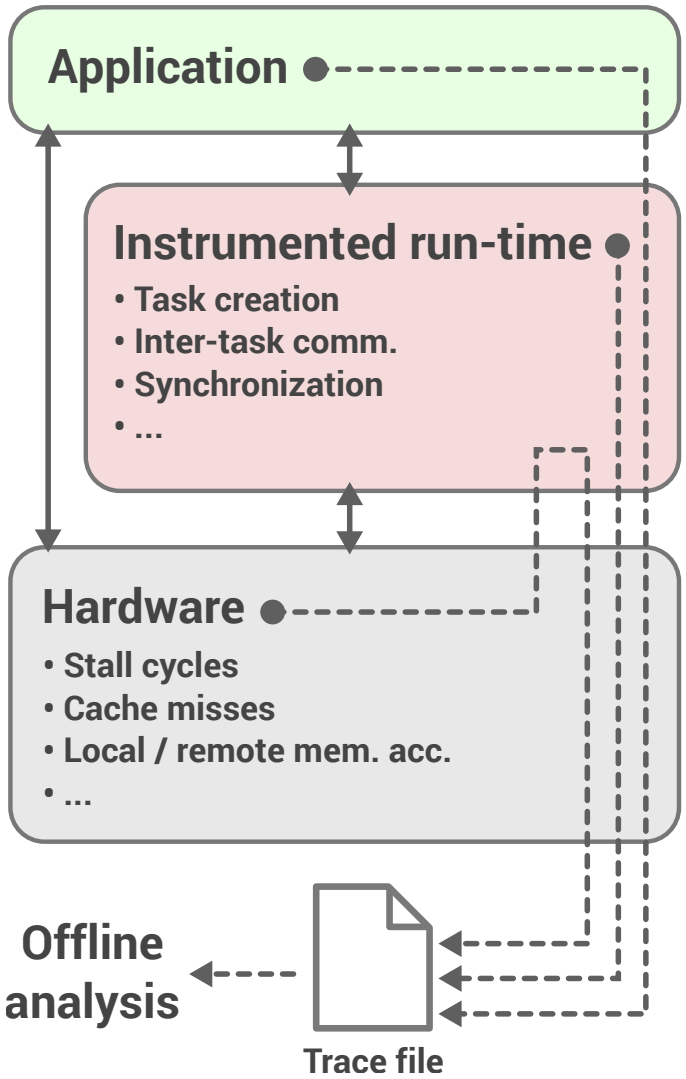
- Every component complex in itself
- Lots of complex interactions
- Performane anomalies? Which component(s)?
- Relation to execution model: Task-centric analysis

Aftermath: Make things easier with the right tool

Outline

- Trace-based Analysis
- Overview of Aftermath
- Demo
- Conclusion

Trace-Based Analysis

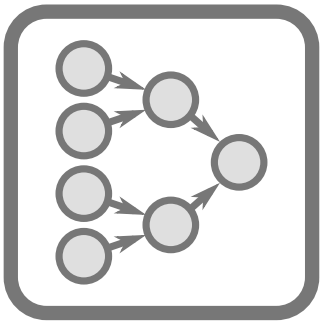


Trace data

- Captured during execution
- Run-time / hardware / application events
- Event types
  - Discrete events
  - State changes
  - Counter values
- Dump to trace file for post-mortem analysis

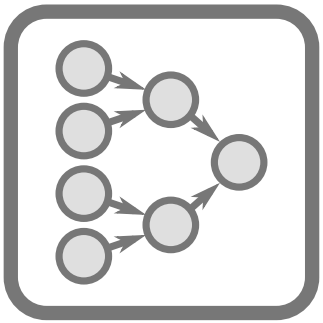
Aftermath: Why a new Tool?

Aftermath: Why a new Tool?

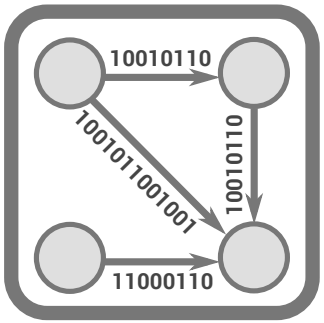


Support for Dependent Tasks

Aftermath: Why a new Tool?

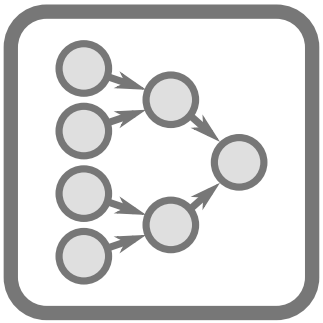


Support for Dependent Tasks

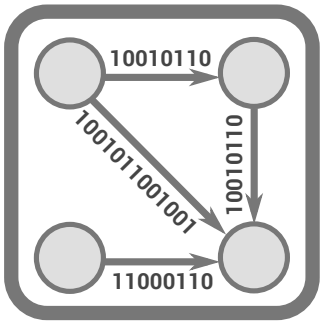


Data-flow Analysis

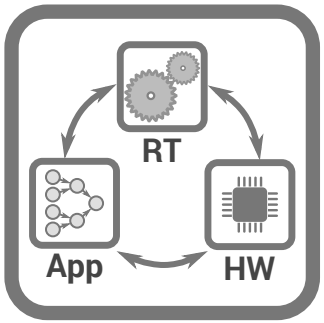
Aftermath: Why a new Tool?



Support for Dependent Tasks

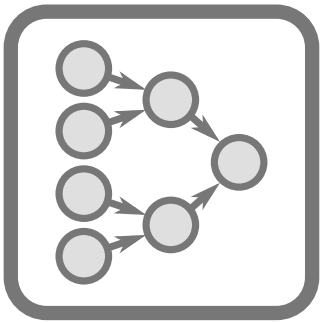


Data-flow Analysis

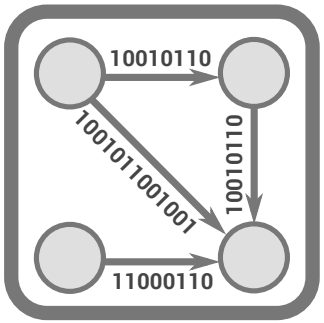


Cross-Layer Analysis

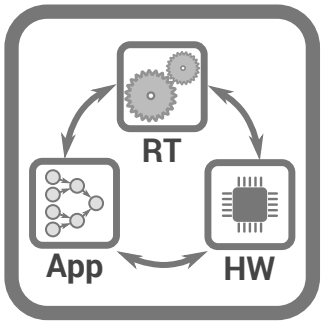
Aftermath: Why a new Tool?



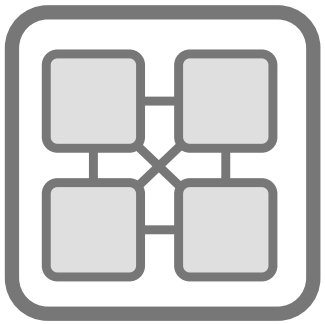
Support for Dependent Tasks



Data-flow Analysis

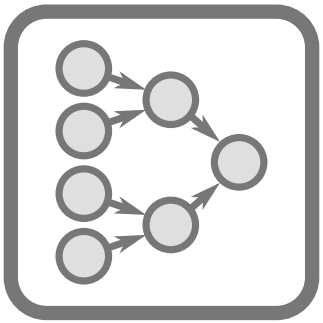


Cross-Layer Analysis

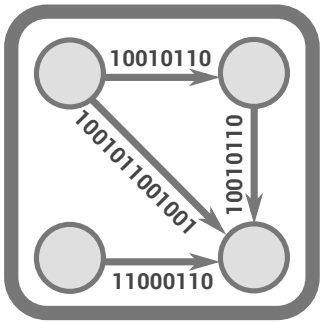


Support for NUMA

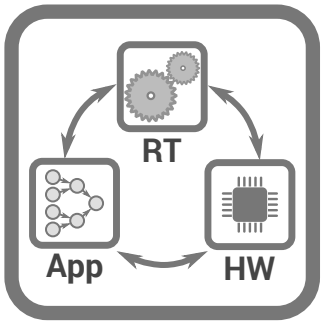
Aftermath: Why a new Tool?



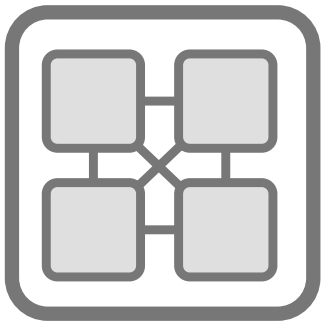
Support for Dependent Tasks



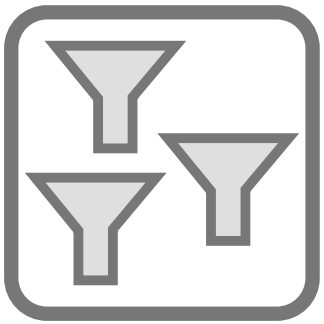
Data-flow Analysis



Cross-Layer Analysis

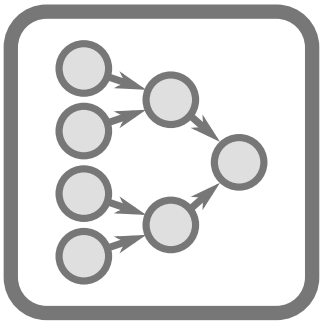


Support for NUMA

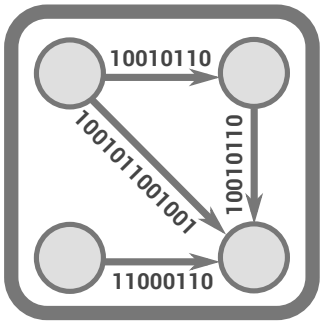


Powerful Filters

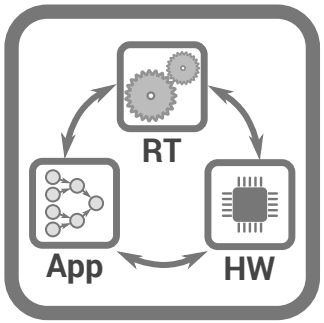
Aftermath: Why a new Tool?



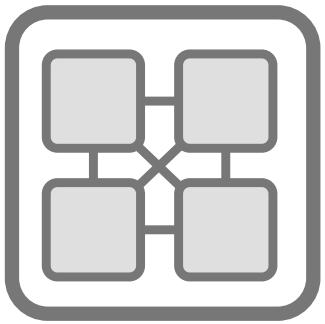
Support for Dependent Tasks



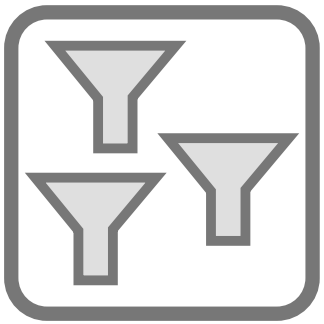
Data-flow Analysis



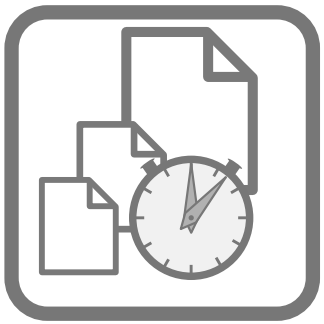
Cross-Layer Analysis



Support for NUMA

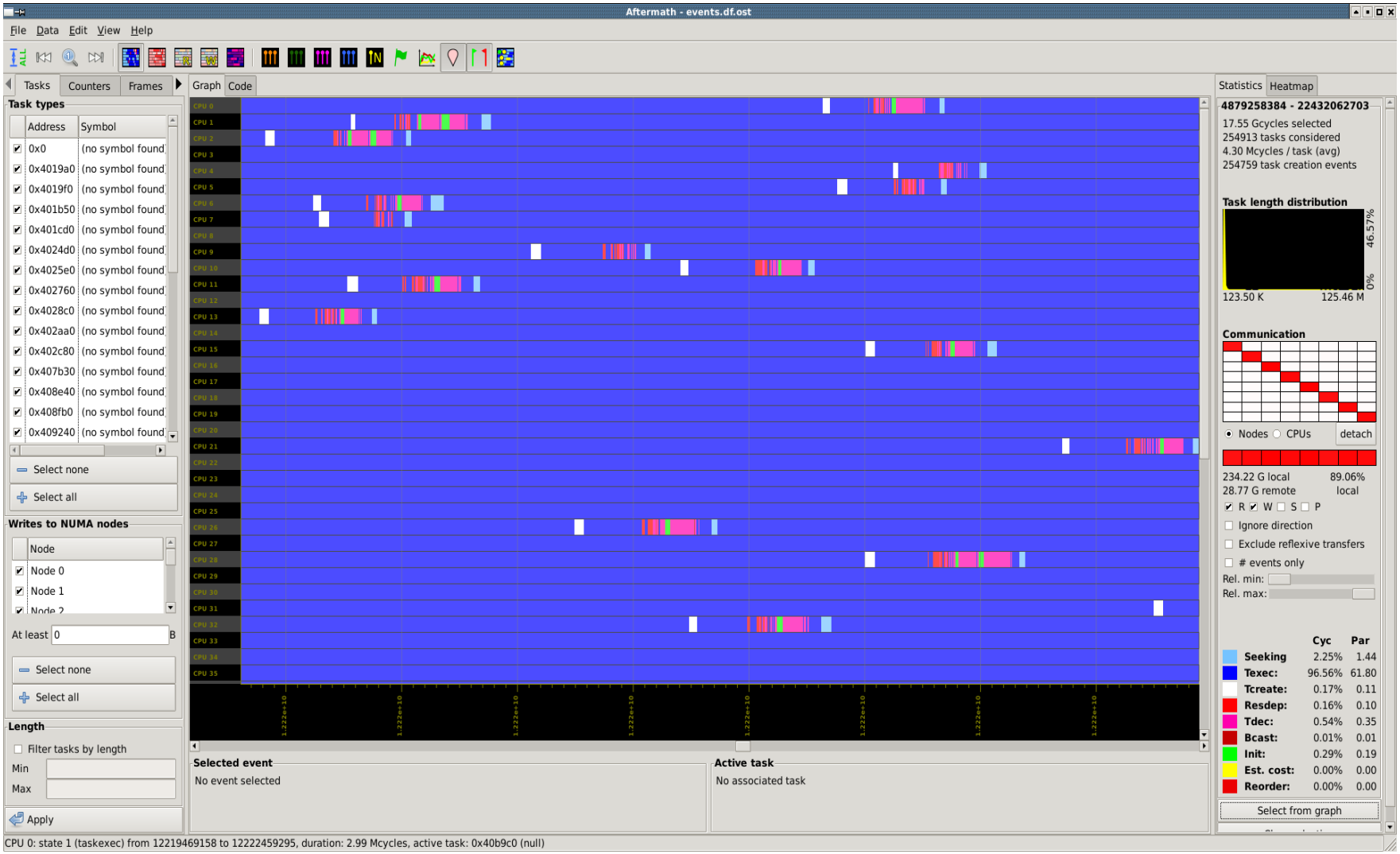


Powerful Filters

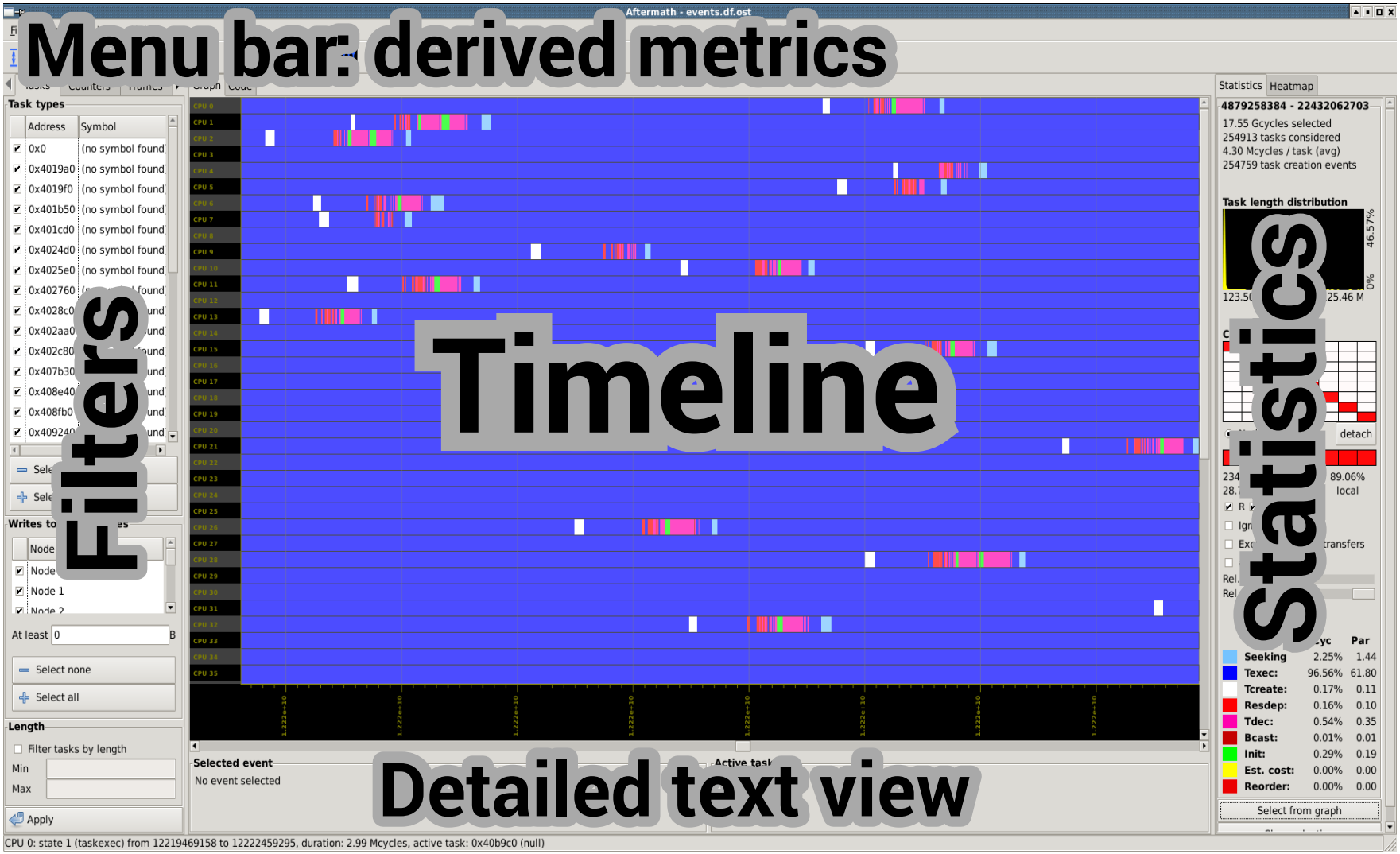


Reactive UI for large Files

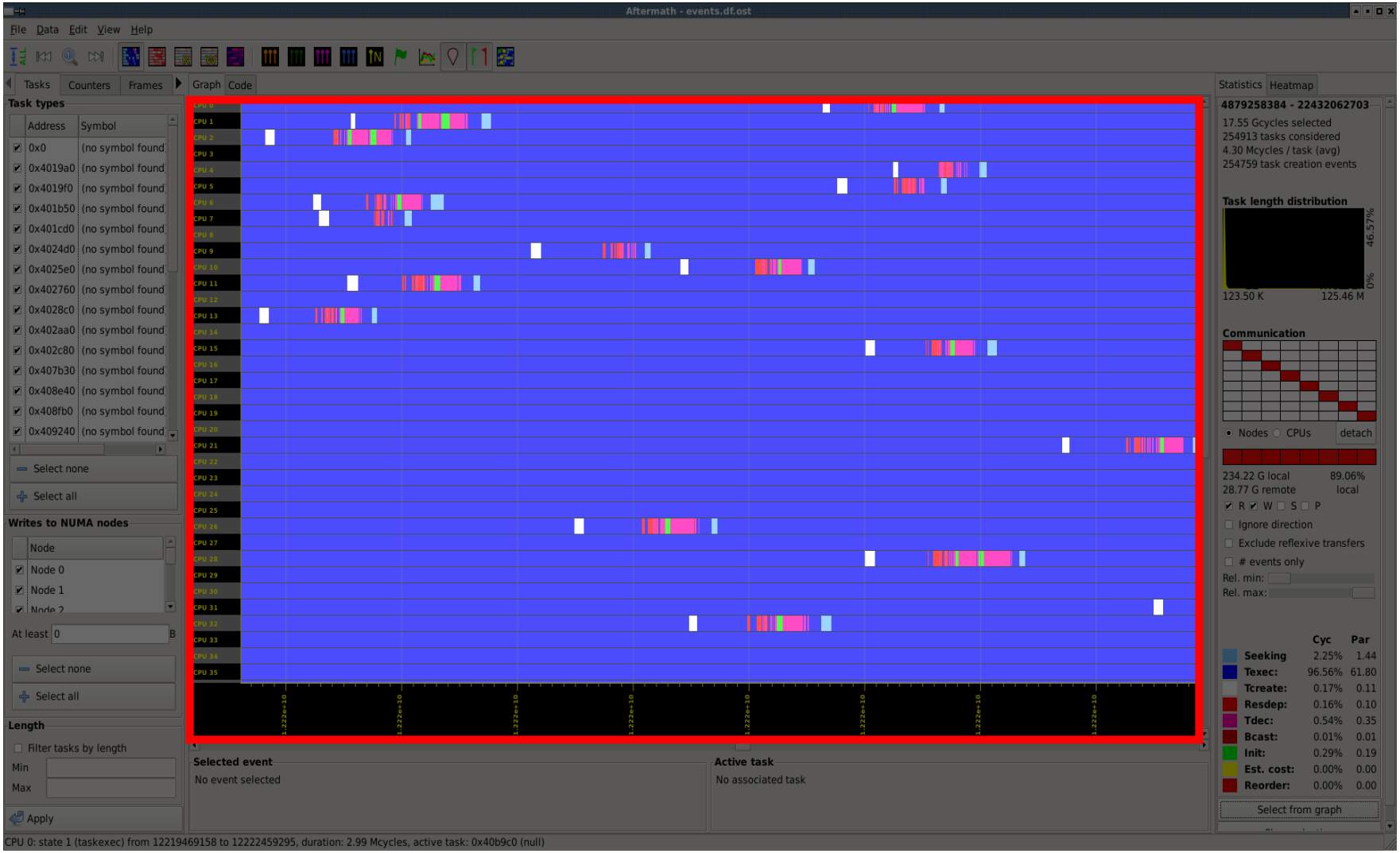
Aftermath: Overview of the GUI



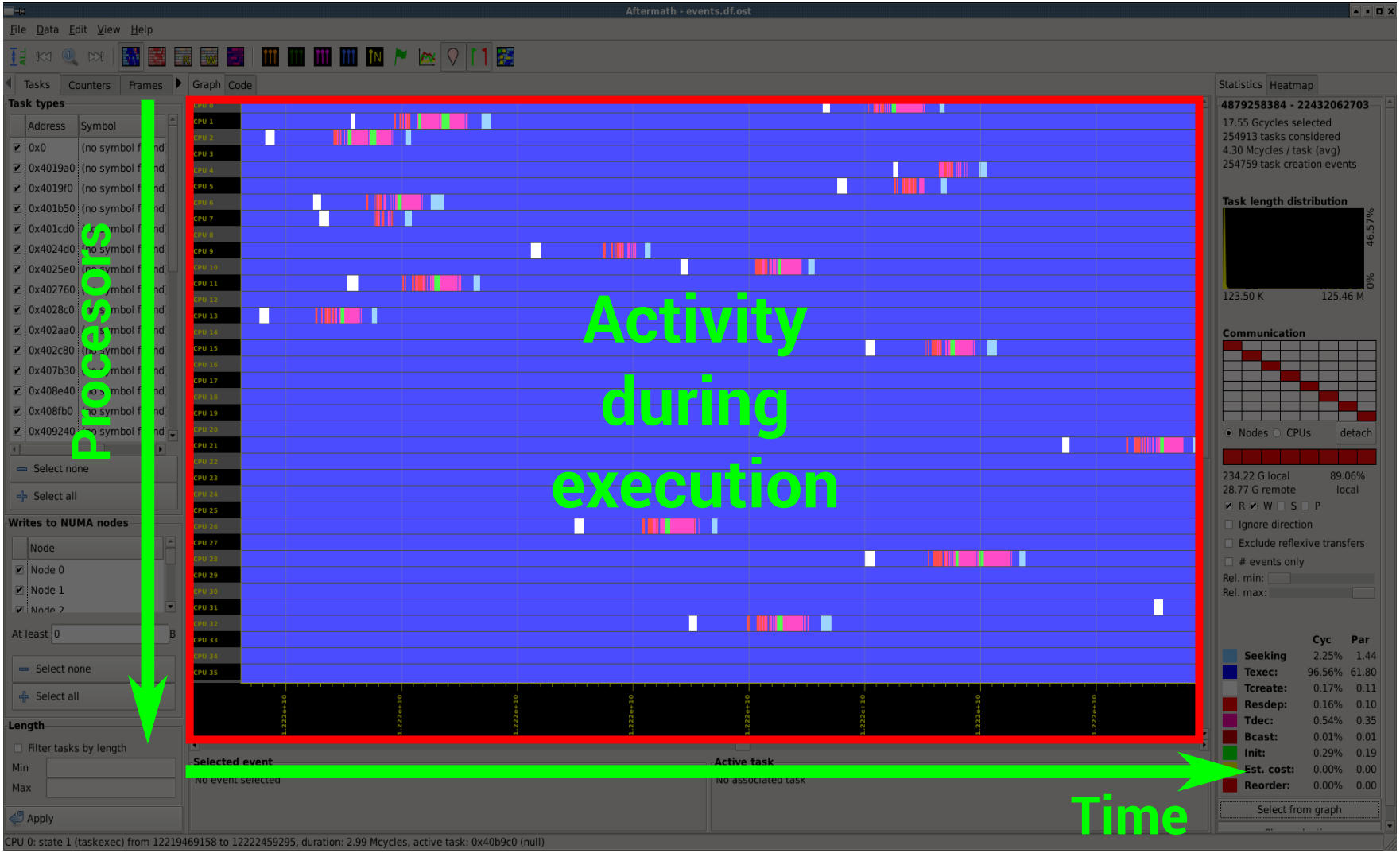
Aftermath: Overview of the GUI



Aftermath: Overview of the GUI

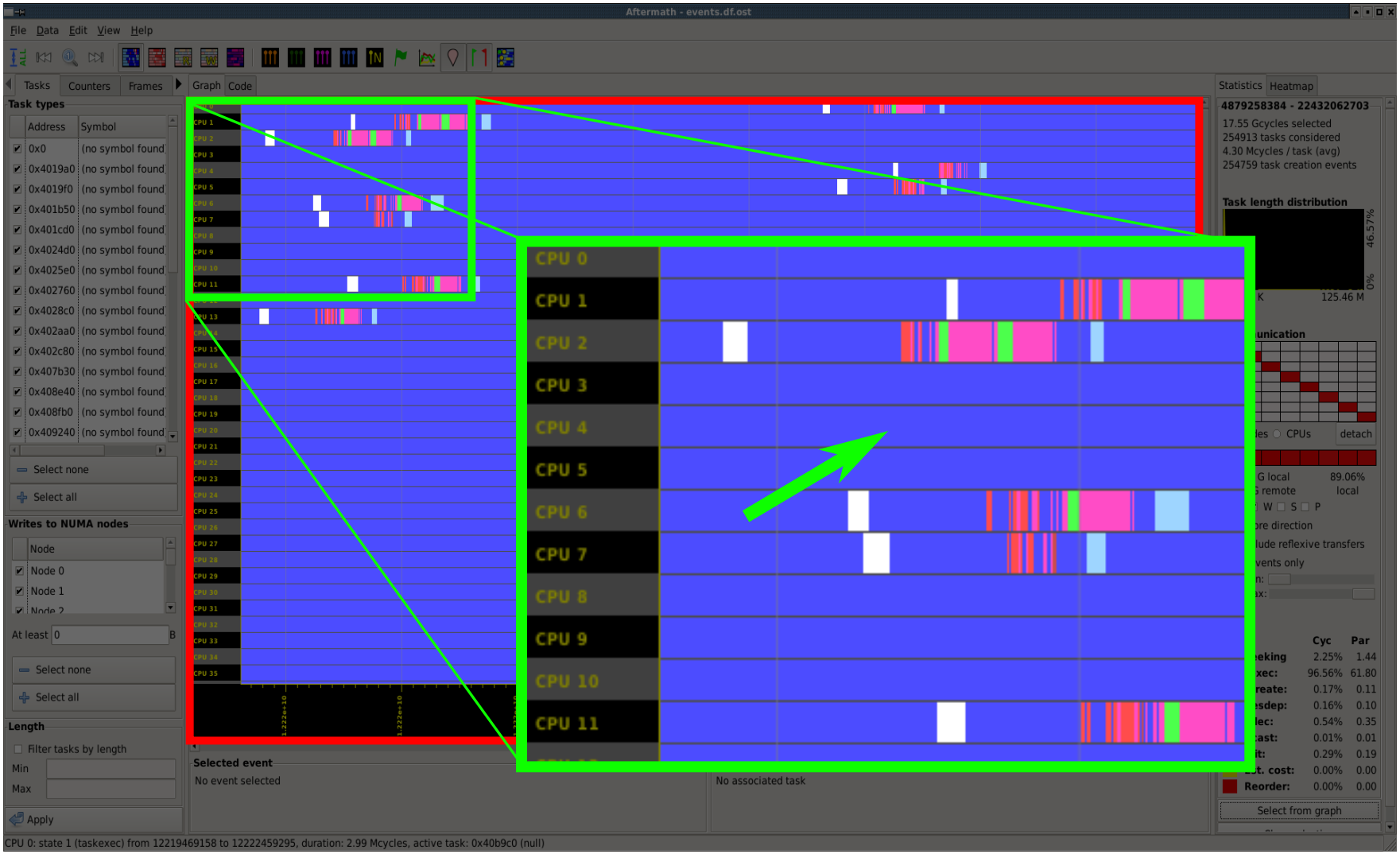


Aftermath: Overview of the GUI



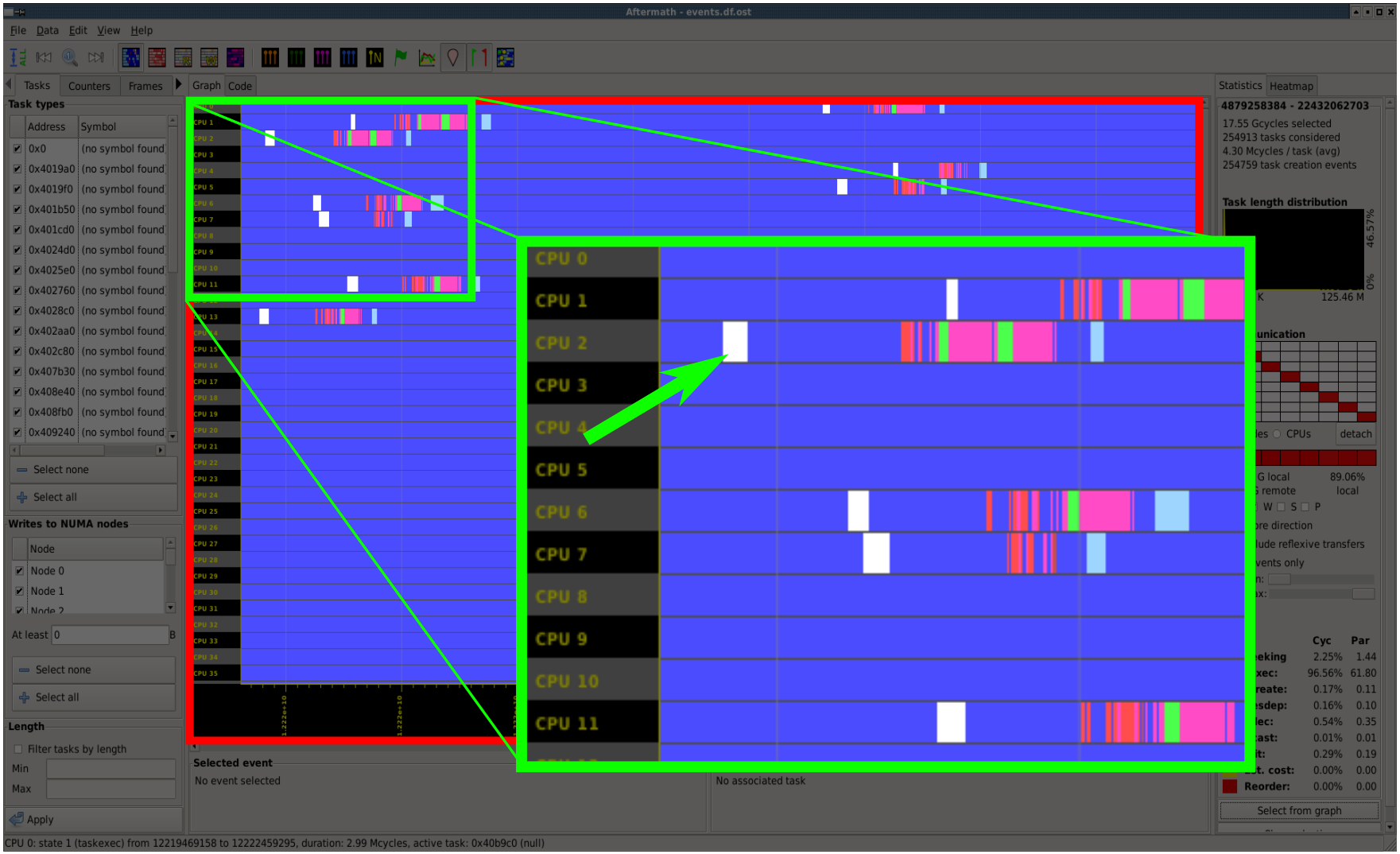
Aftermath: Overview of the GUI

Aftermath: Overview of the GUI



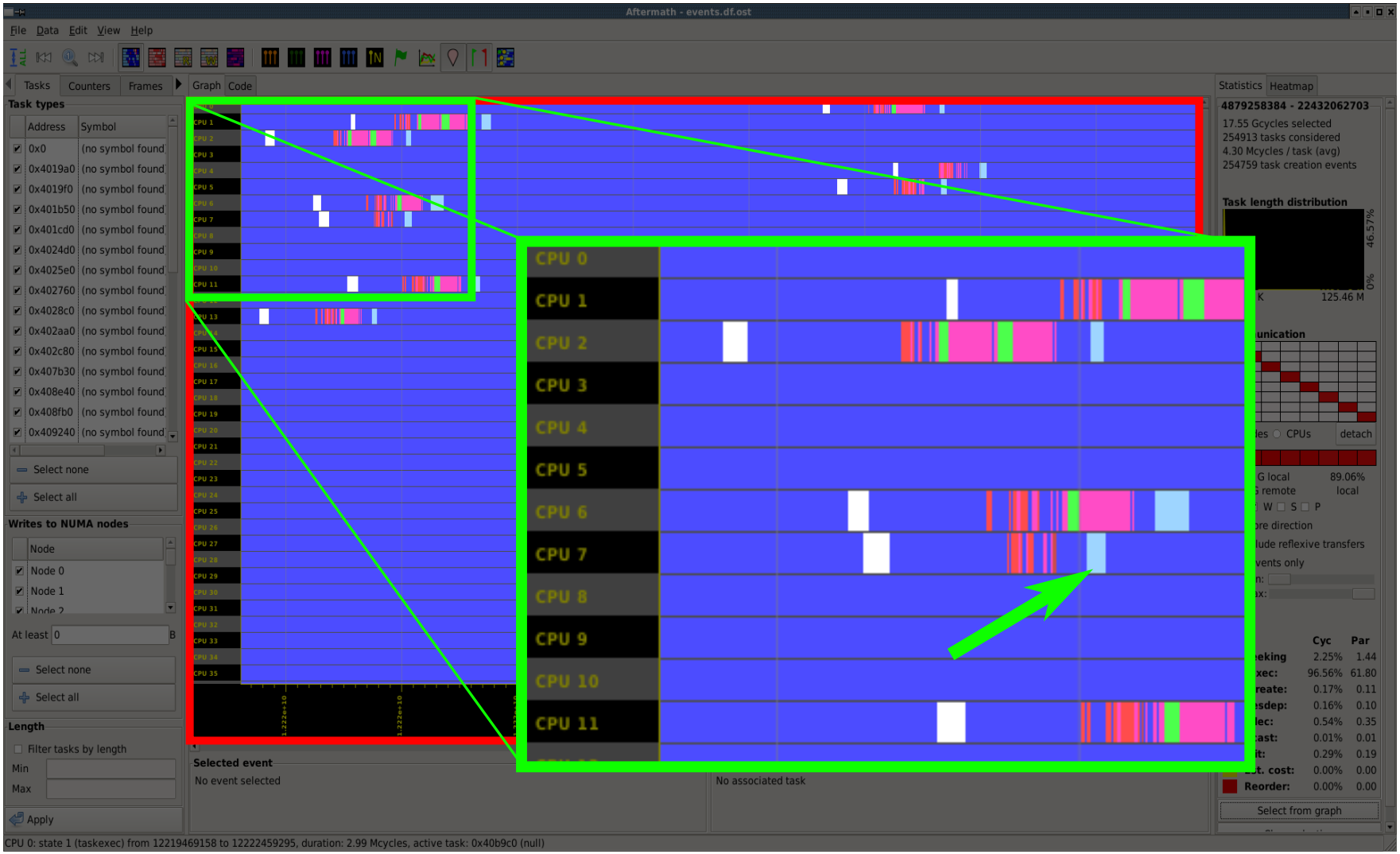
Task execution (dark blue)

Aftermath: Overview of the GUI



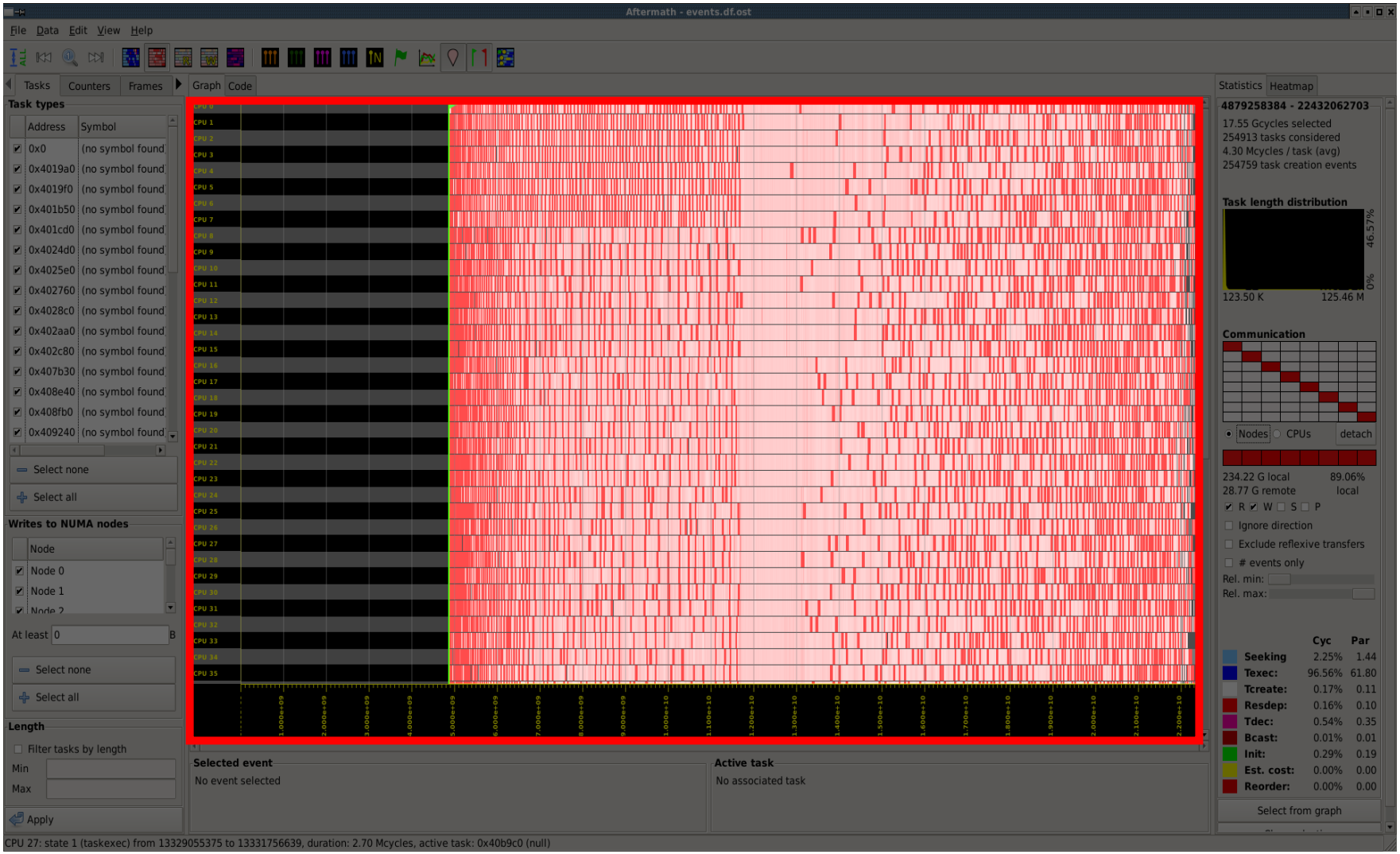
Task creation (white)

Aftermath: Overview of the GUI



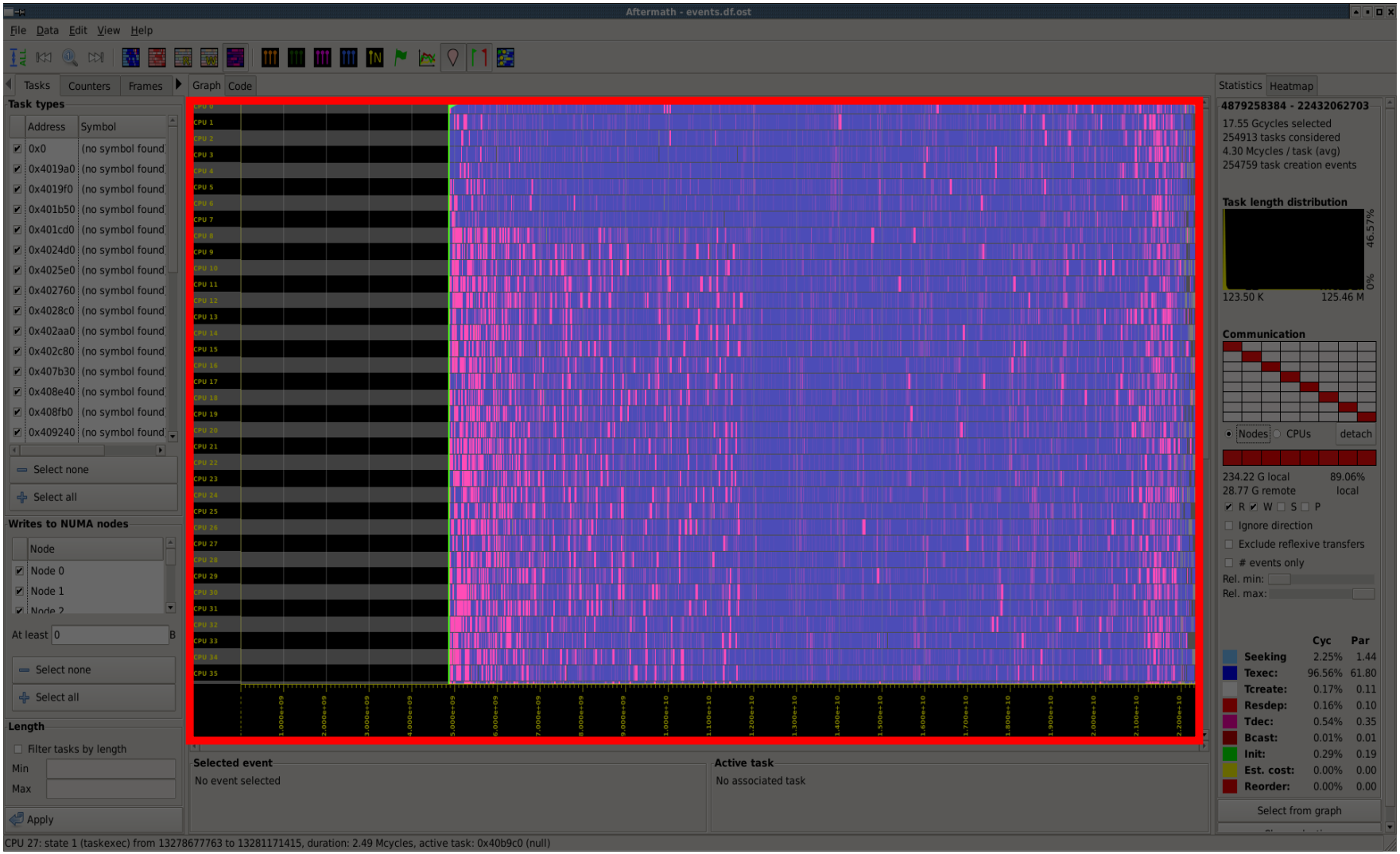
Searching for work (light blue)

Aftermath: Overview of the GUI



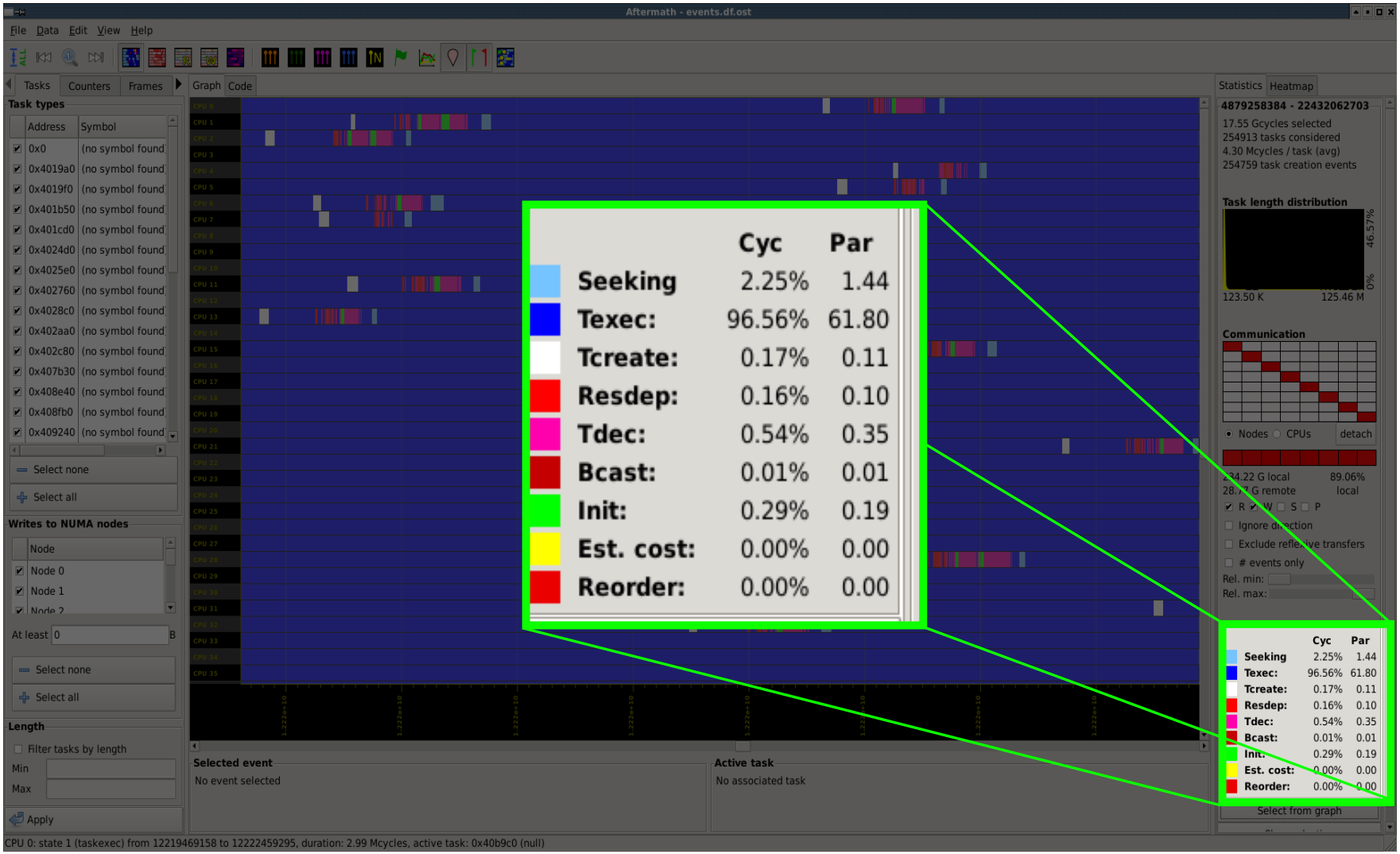
Heatmap indicating task duration (white: short, red: long)

Aftermath: Overview of the GUI



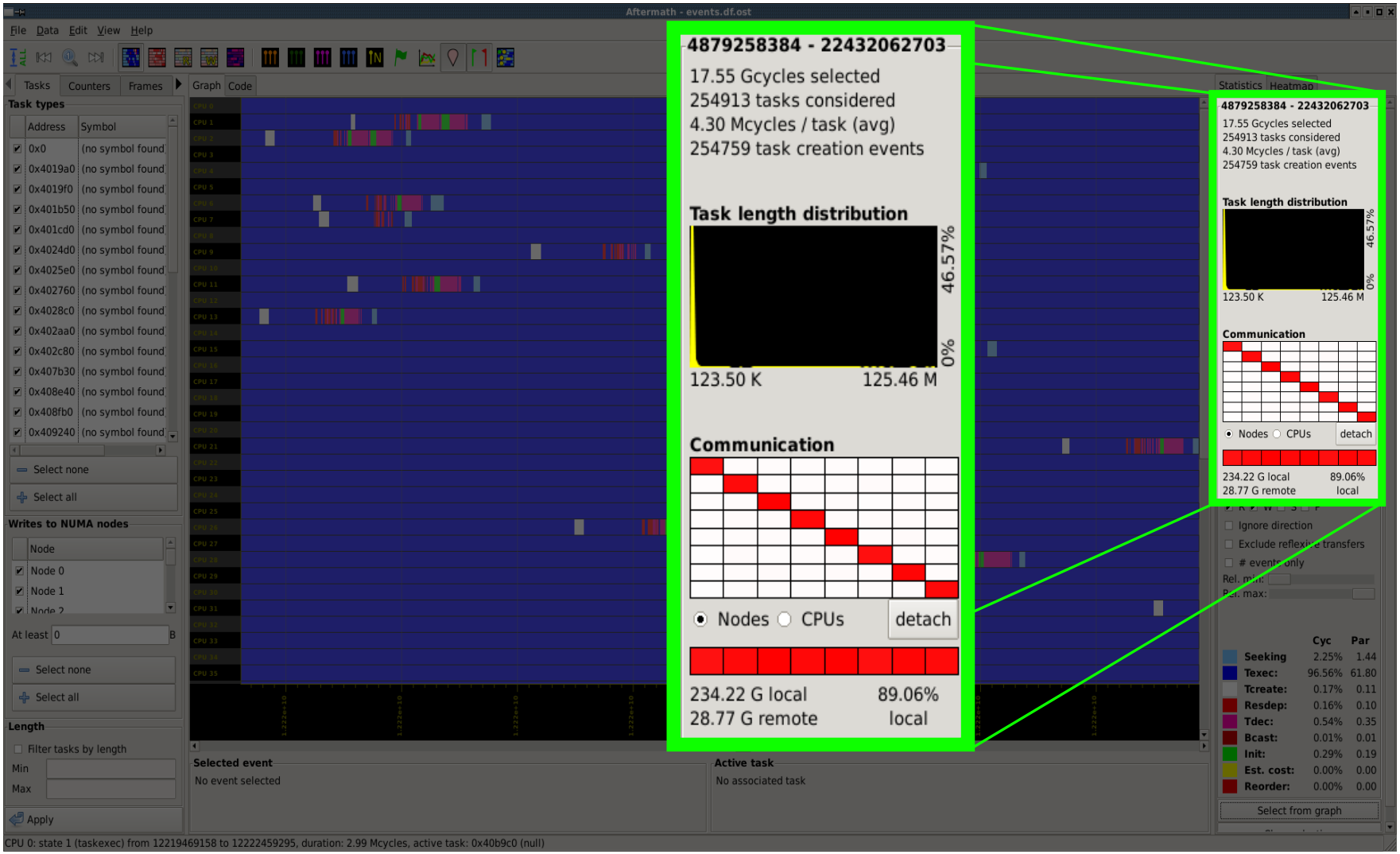
NUMA heatmap (blue: local accesses, pink: remote accesses)

Aftermath: Overview of the GUI



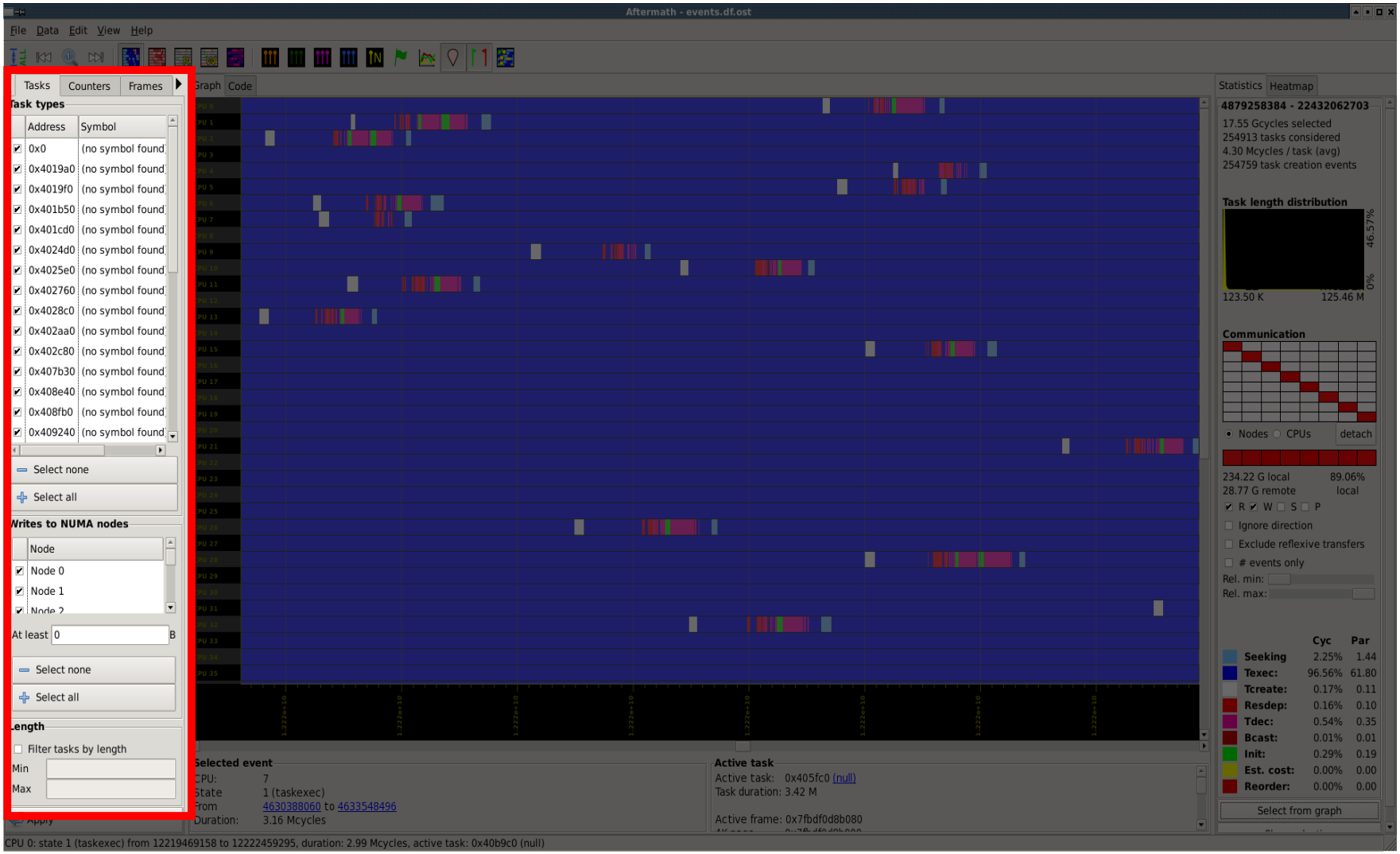
Basic statistics for run-time states

Aftermath: Overview of the GUI



Histogram for task duration + NUMA statistics

Aftermath: Overview of the GUI

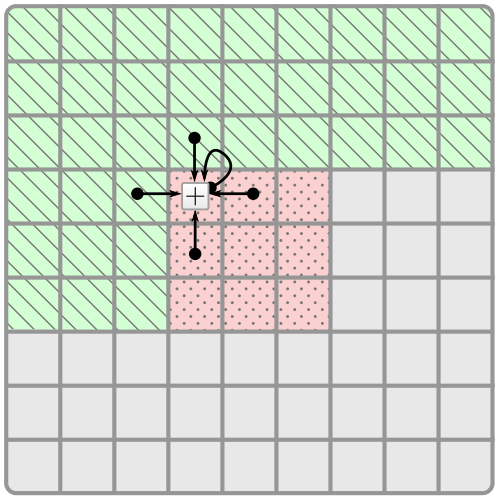


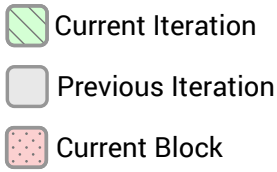
Filters: task type / duration, CPUs, event types, . . .

Aftermath: Overview of the GUI

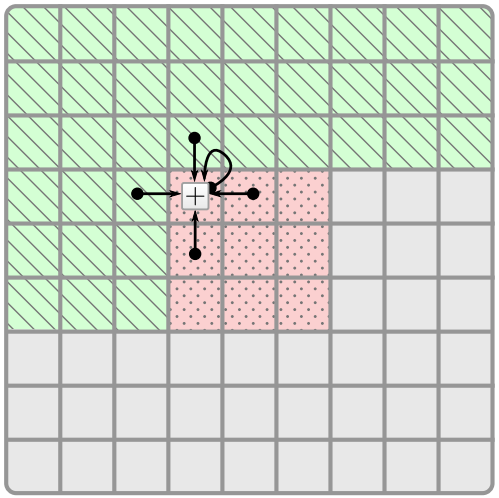
Detailed task view: duration, dependences, accessed NUMA nodes

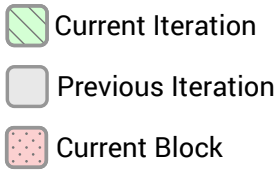
Demo: Seidel-2d



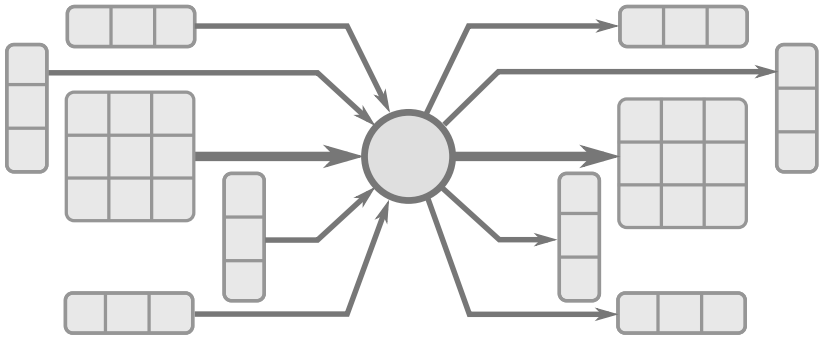


Demo: Seidel-2d

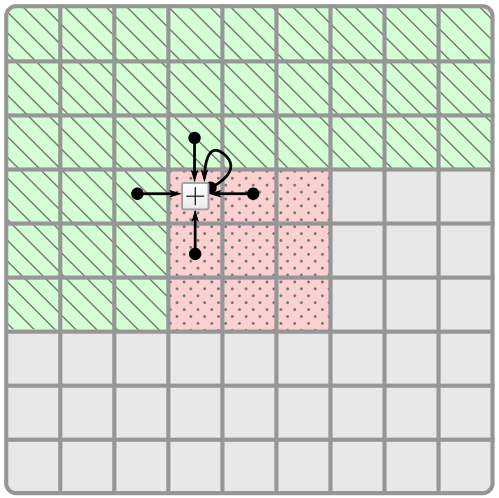


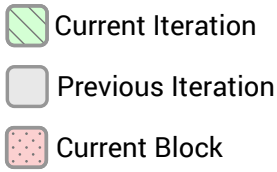


OpenStream implementation

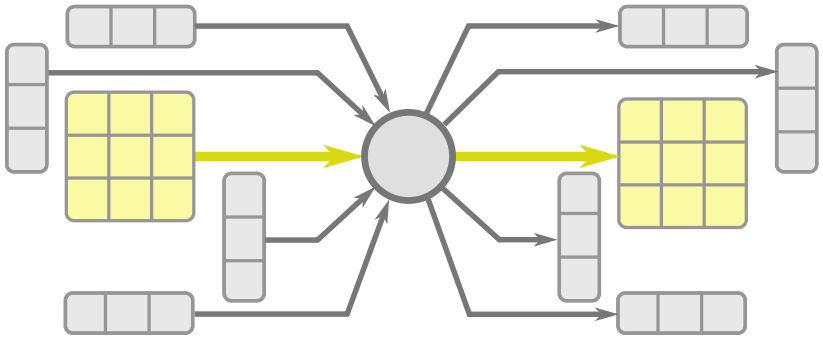


Demo: Seidel-2d

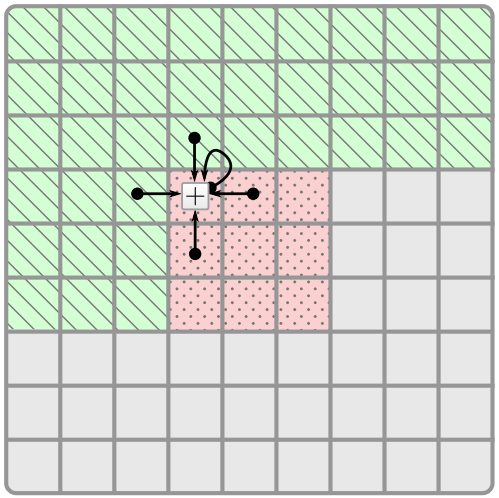


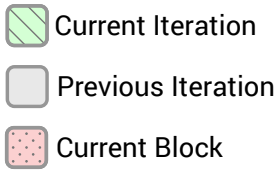


OpenStream implementation

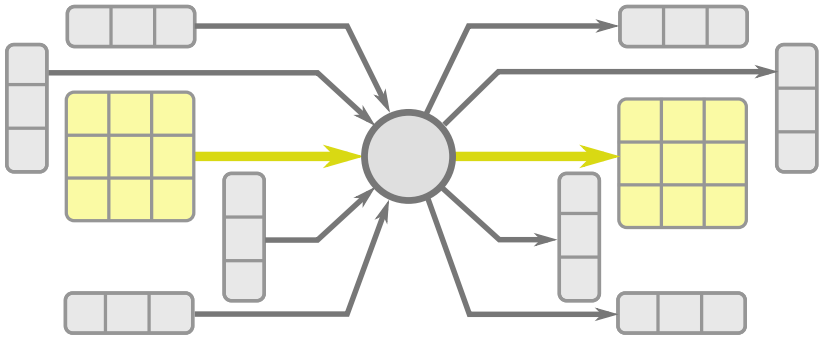


Demo: Seidel-2d





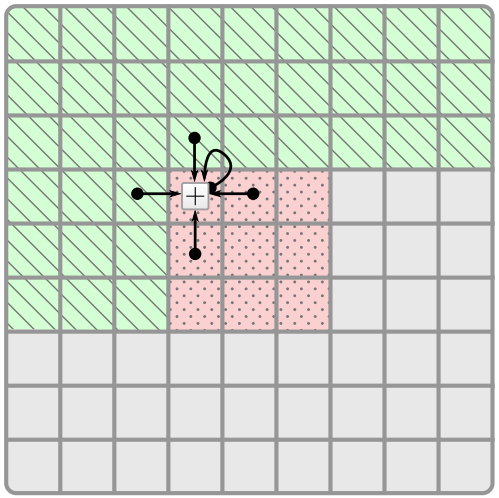
OpenStream implementation



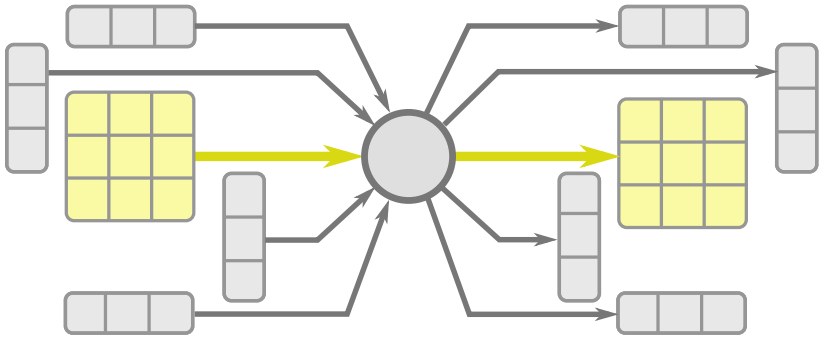
Benchmark parameters

- Double precision floats
- Matrix: $2^{14} \times 2^{14}$ (2 GiB)
- Block: $2^{8} \times 2^{8}$ (512 KiB)
- 60 iterations
- 254,977 tasks

Demo: Seidel-2d



OpenStream implementation



Benchmark parameters

- Double precision floats
- Matrix: $2^{14} \times 2^{14}$ (2 GiB)
- Block: $2^{8} \times 2^{8}$ (512 KiB)
- 60 iterations
- 254,977 tasks

Test system

- SGI UV 2000
- 24×Intel Xeon E5-4640
- Hyperthreading disabled
- 192 cores
- 24 NUMA nodes, 756 GiB RAM

DEMO

1. Default, random work-stealing
2. Run-time optimized for NUMA

Impact on OpenStream

Run-time development & performance debugging

- NUMA-aware task and data placement
- Topology-aware work-stealing
- Overhead analysis
  - Memory allocation
  - Task creation

Application development & performance debugging

- Task granularity / available parallelism
- Partitioning: main tasks / auxiliary tasks
- Task creation
- Dependence patterns
- Hardware performance counter analysis

Model-Centric Analysis of OpenMP Programs*

Analysis of OpenMP programs

- Parallel loops
  - Examine Partitioning
  - Execution time per worker
  - Filtering
- OpenMP 4 tasks
- OpenMP constructs: Barriers, single, . . .
- Hardware performance counters

*Joint work with Jean-Baptiste Bréjon

Model-Centric Analysis of OpenMP Programs*

Analysis of OpenMP programs

- Parallel loops
  - Examine Partitioning
  - Execution time per worker
  - Filtering
- OpenMP 4 tasks
- OpenMP constructs: Barriers, single, . . .
- Hardware performance counters

Easy-to-use scripts and library

- Instrumented Clang / LLVM OpenMP run-time (KMP / Intel)

```
$ aftermath-openmp-trace -o events.ost -- <program> <args>
$ aftermath events.ost
```

*Joint work with Jean-Baptiste Bréjon

Summary

Aftermath at a glance

- Fast and responsive UI, multiple connected views
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Joint analysis applications + run-time
- Successfully used for performance debugging of OpenStream

Summary

Aftermath at a glance

- Fast and responsive UI, multiple connected views
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Joint analysis applications + run-time
- Successfully used for performance debugging of OpenStream

Source code and tutorial available at

`http://www.openstream.info/aftermath`

- Free software license, small code base ($<$ 30k lines of C code)
- Few dependences: GTK+, Cairo, Pango, Glibc, . . .

Summary

Aftermath at a glance

- Fast and responsive UI, multiple connected views
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Joint analysis applications + run-time
- Successfully used for performance debugging of OpenStream

Source code and tutorial available at

`http://www.openstream.info/aftermath`

- Free software license, small code base ($<$ 30k lines of C code)
- Few dependences: GTK+, Cairo, Pango, Glibc, . . .

VM with Aftermath + OpenStream + sample traces available at

`http://www.openstream.info/vm`

Summary

Aftermath at a glance

- Fast and responsive UI, multiple connected views
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Joint analysis applications + run-time
- Successfully used for performance debugging of OpenStream

Source code and tutorial available at

`http://www.openstream.info/aftermath`

- Free software license, small code base ($<$ 30k lines of C code)
- Few dependences: GTK+, Cairo, Pango, Glibc, . . .

VM with Aftermath + OpenStream + sample traces available at

`http://www.openstream.info/vm`

OpenMP support very soon!