Language-Centric Performance Analysis of OpenMP Programs with Aftermath

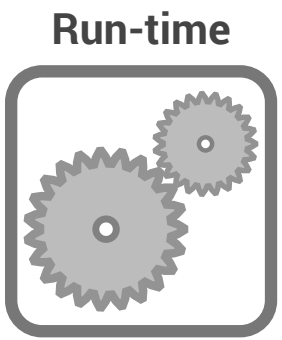
Andi Drebes

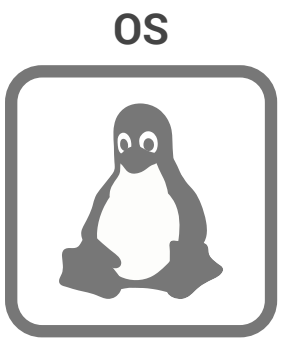
The University of Manchester
School of Computer Science
Advanced Processor Technologies
andi.drebes@manchester.ac.uk

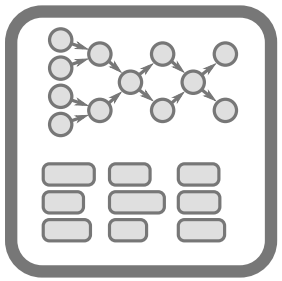
Joint work with:
Jean-Baptiste Bréjon, Antoniu Pop, Karine Heydemann, Albert Cohen

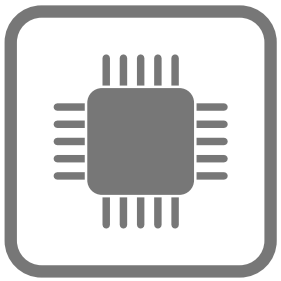
IWOMP 2016

Analysis of OpenMP Programs

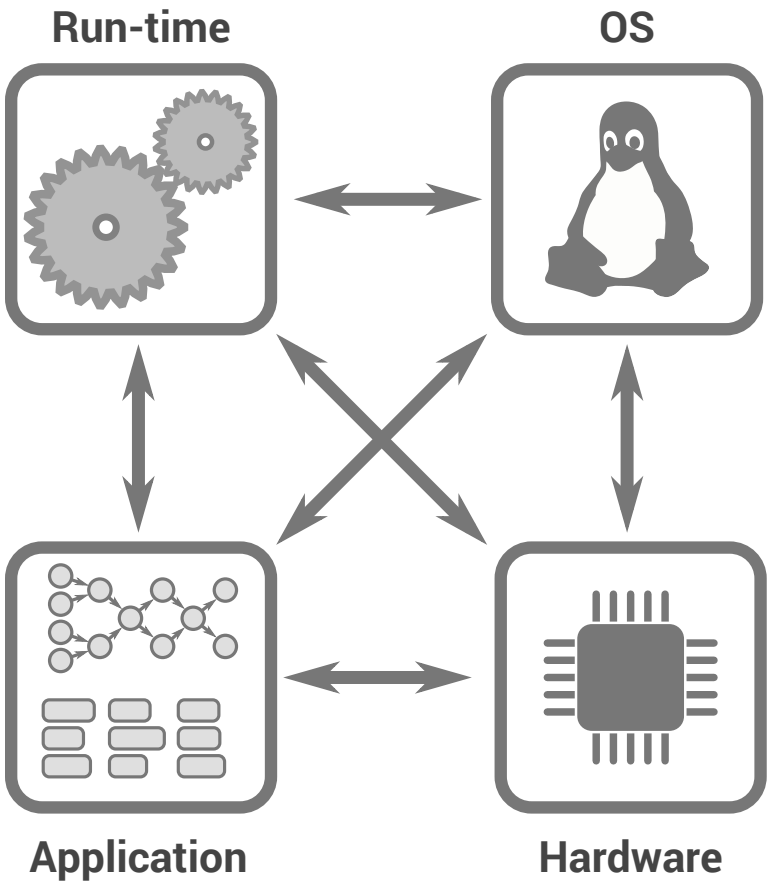




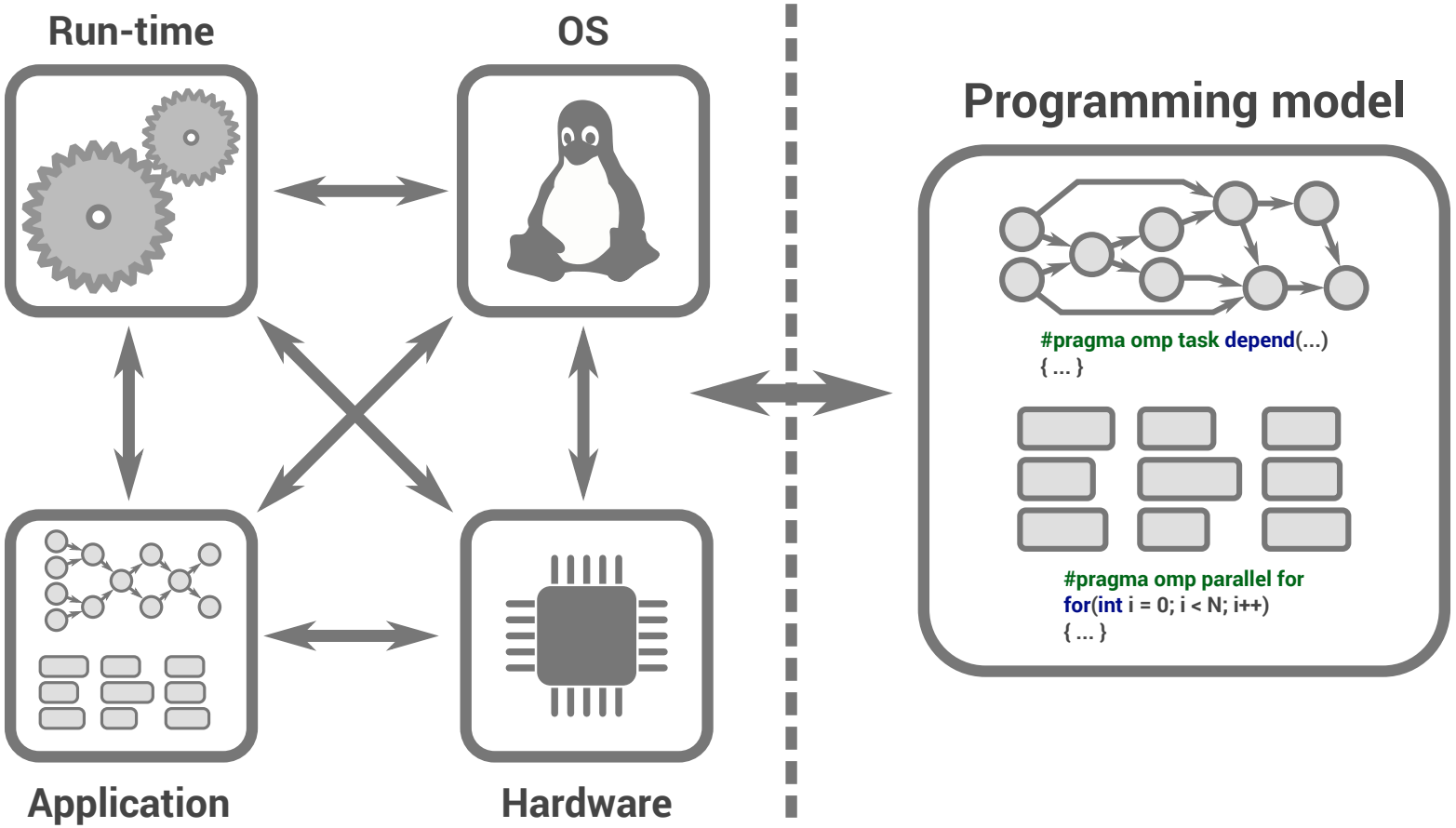




Analysis of OpenMP Programs



Analysis of OpenMP Programs



New Tools for Performance Analysis

Frequent topics for performance analysis:

- Amount of parallelism and load balacing
- Duration of execution phases
- Synchronization overhead (e.g., barriers)
- Choice of an appropriate loop schedule
- Data distribution on NUMA systems
- Relate hardware events to loops / tasks

New Tools for Performance Analysis

Frequent topics for performance analysis:

- Amount of parallelism and load balacing
- Duration of execution phases
- Synchronization overhead (e.g., barriers)
- Choice of an appropriate loop schedule
- Data distribution on NUMA systems
- Relate hardware events to loops / tasks

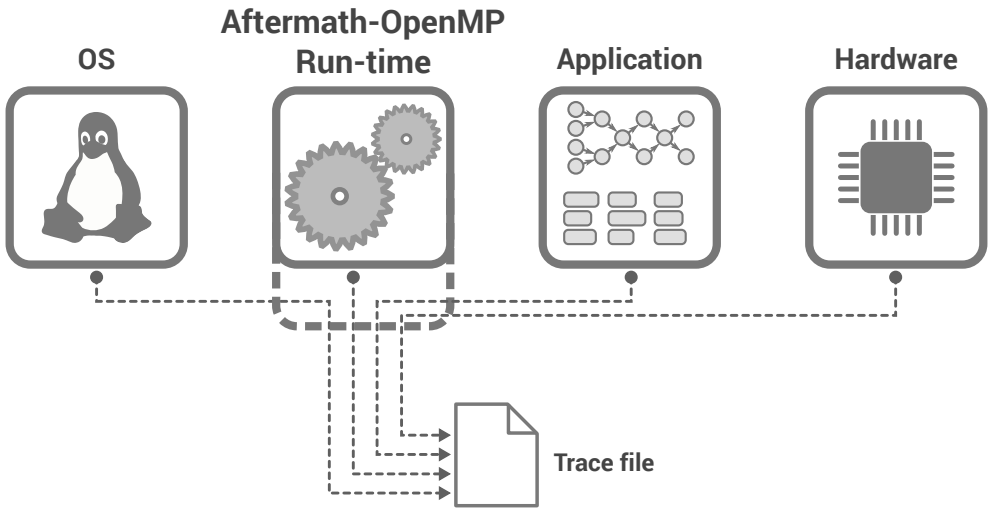
Our tools: Aftermath & Aftermath-OpenMP

- Aftermath: Graphical tool for performance analysis
- Aftermath-OpenMP: Instrumented LLVM/clang run-time

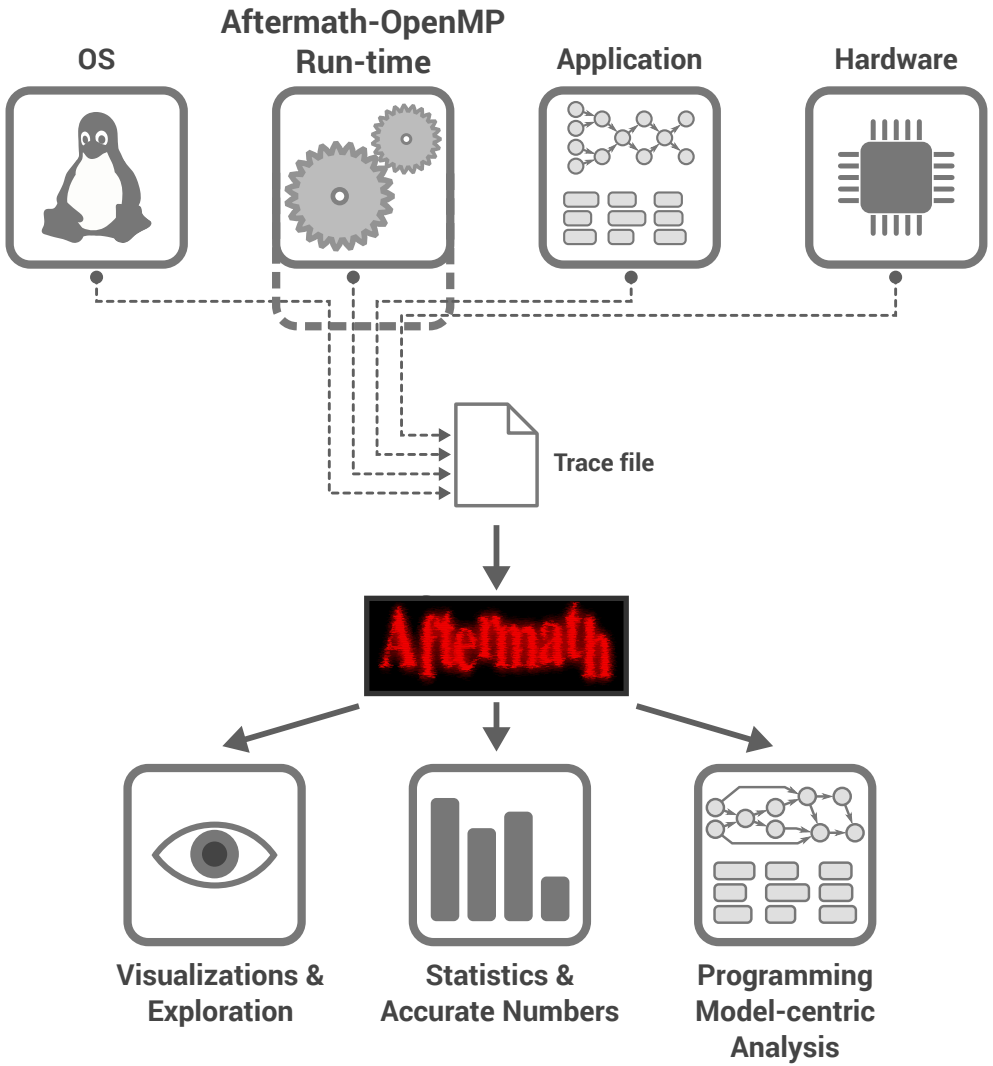
Outline

1. Overview of Trace-based Analysis
2. Overview of Aftermath’s GUI
3. Demo
4. Overhead of Tracing
5. Summary & Conclusion

Trace-based Analysis with Aftermath



Trace-based Analysis with Aftermath



Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

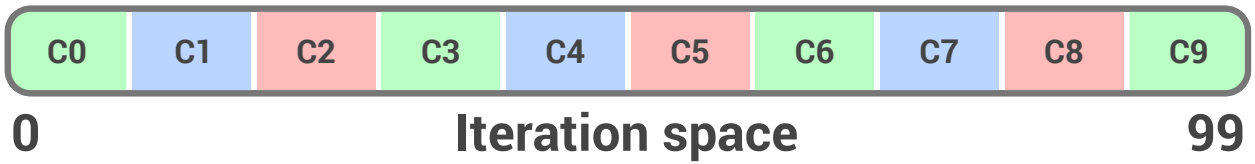
Loop construct

Loop

Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

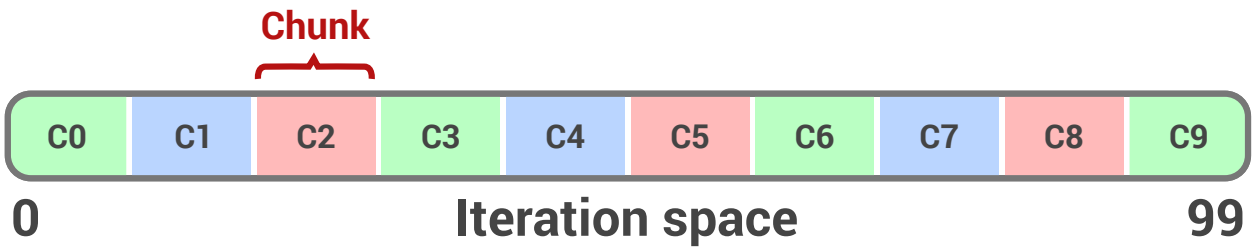


Loop

Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

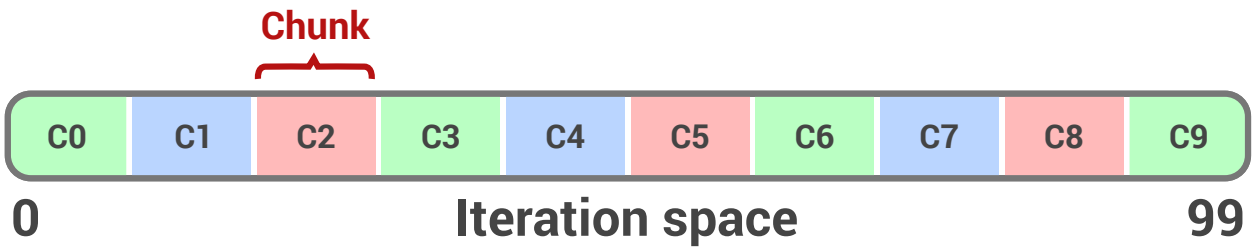


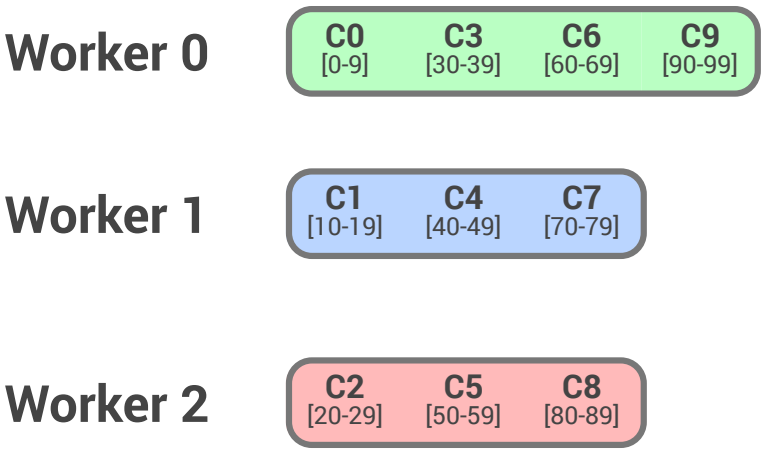
Loop

Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

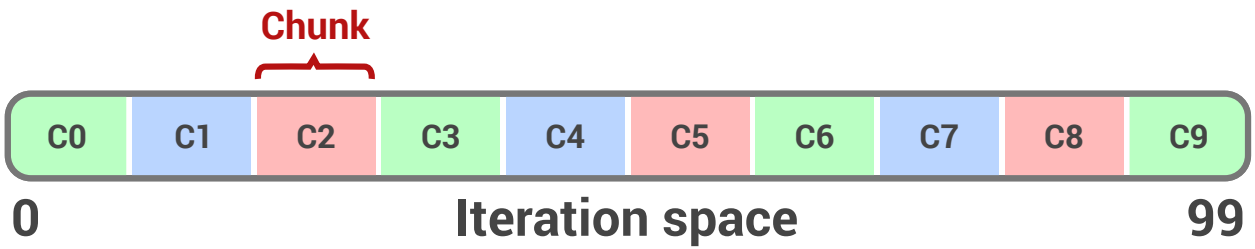


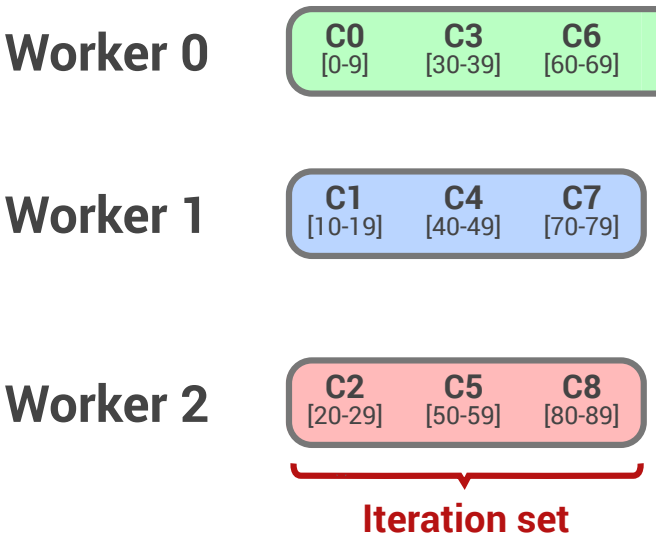


Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

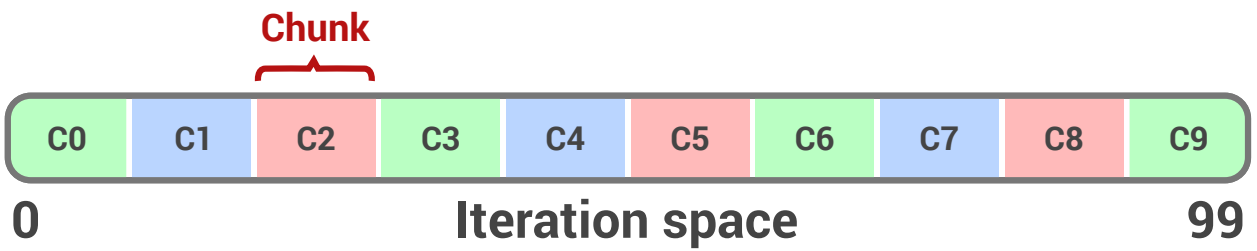


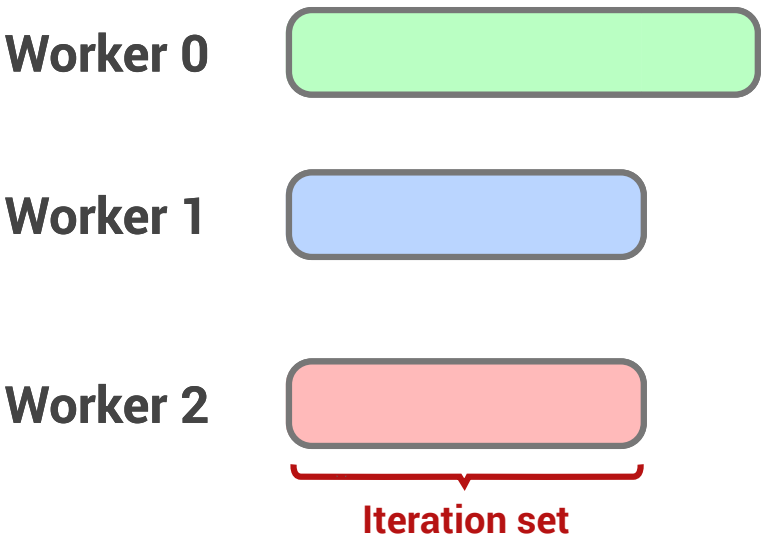


Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

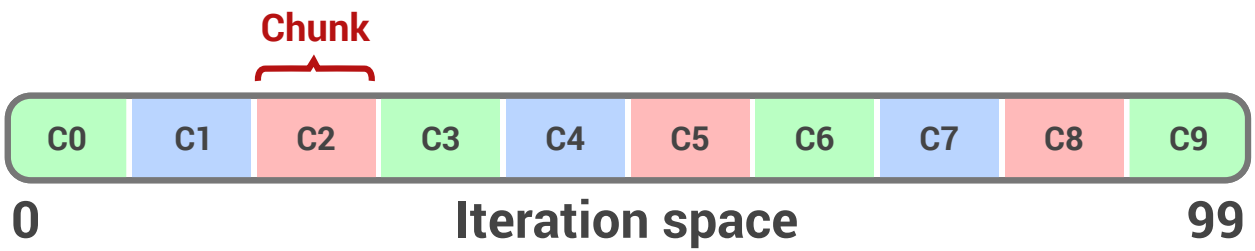


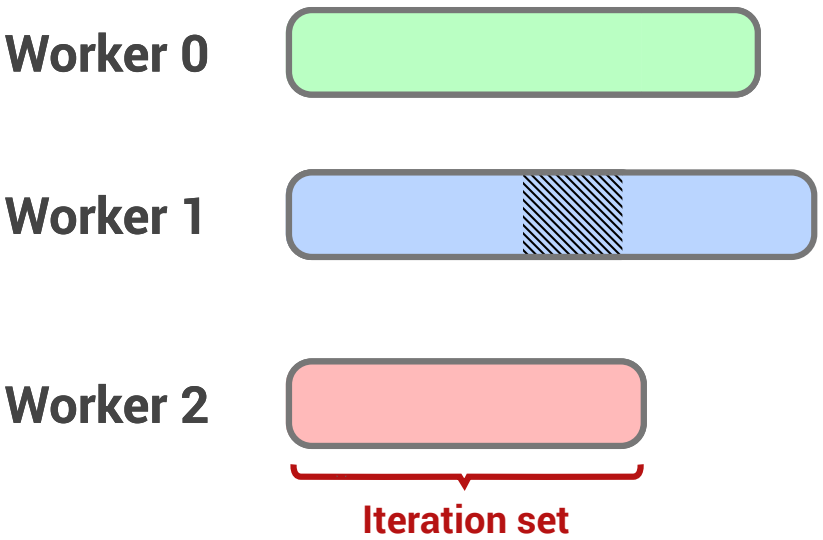


Terminology

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

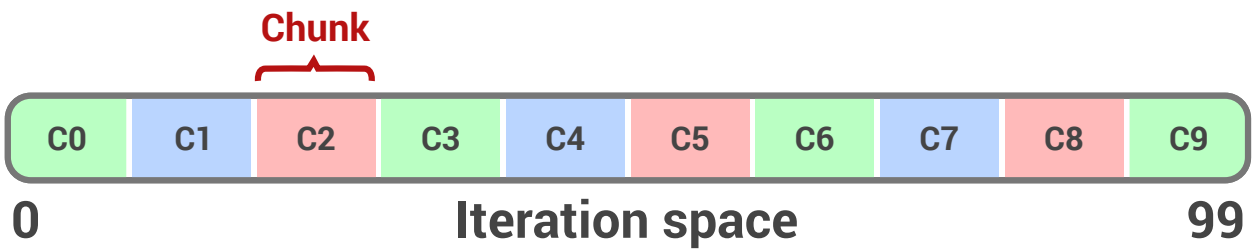


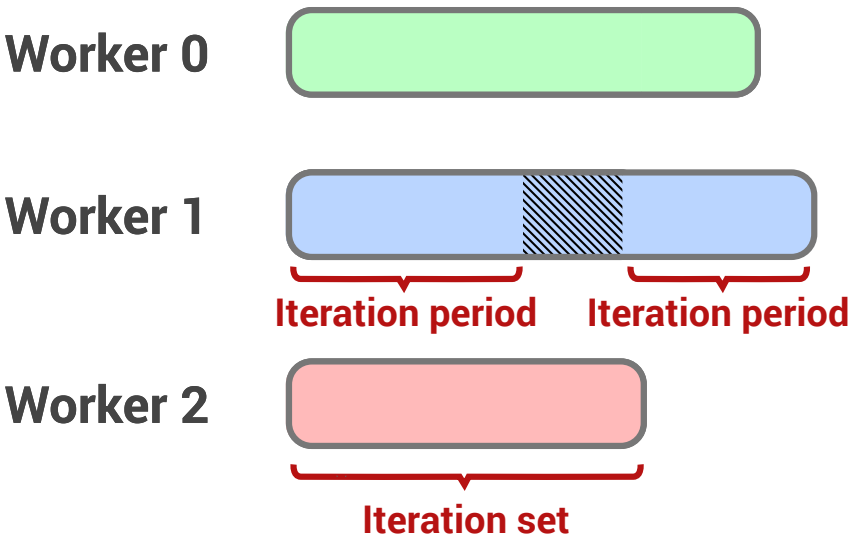


Terminology

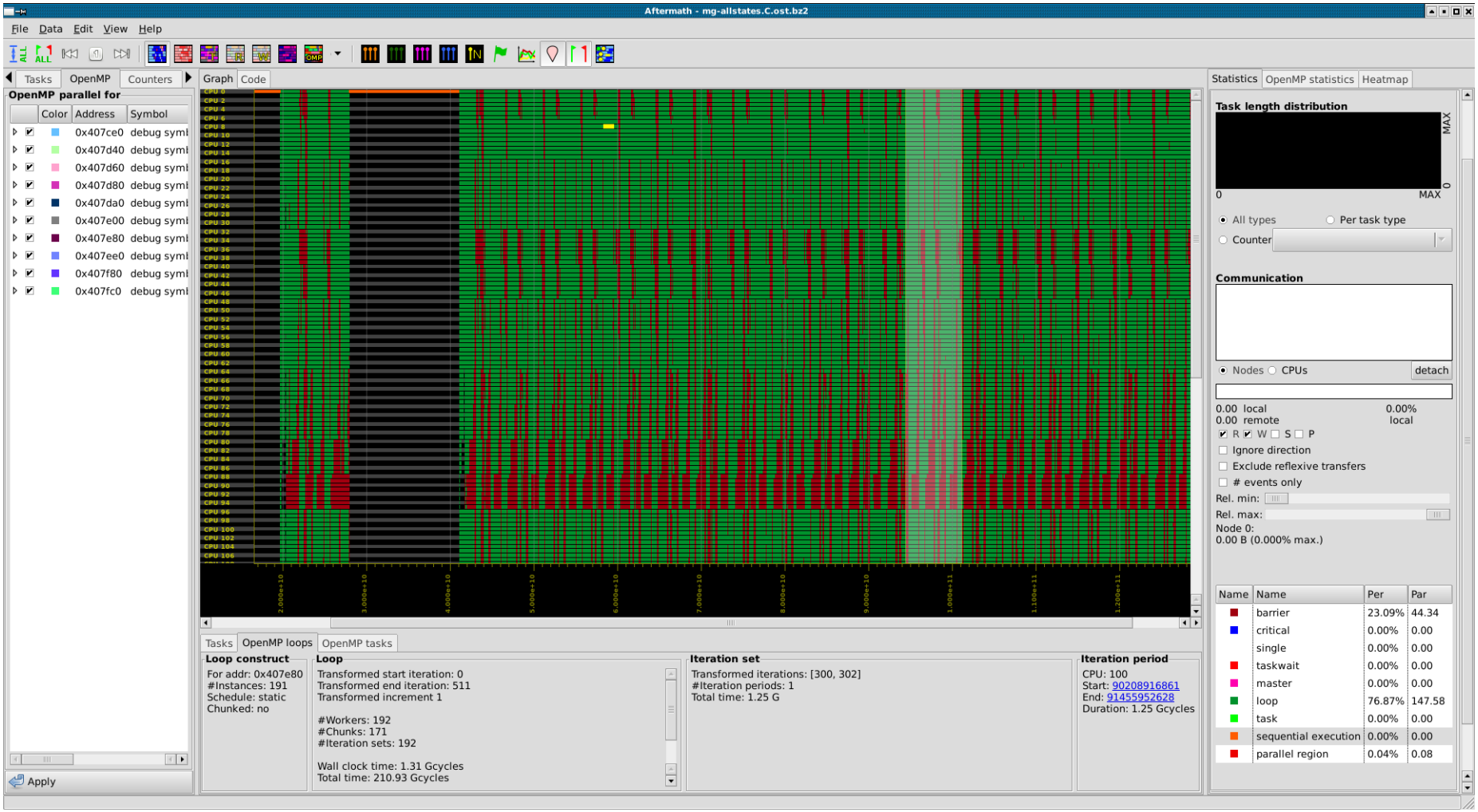
```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }
```

Loop construct

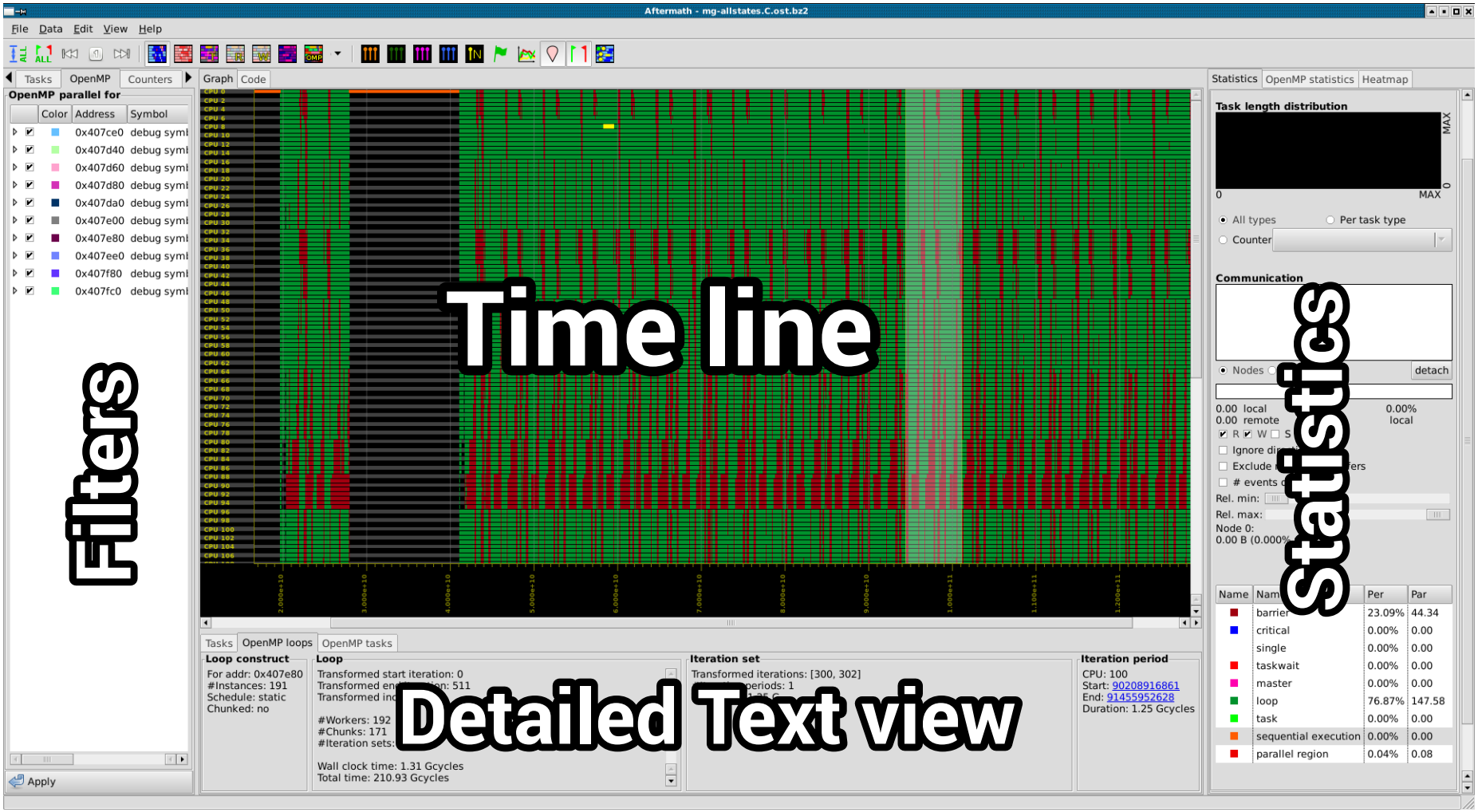




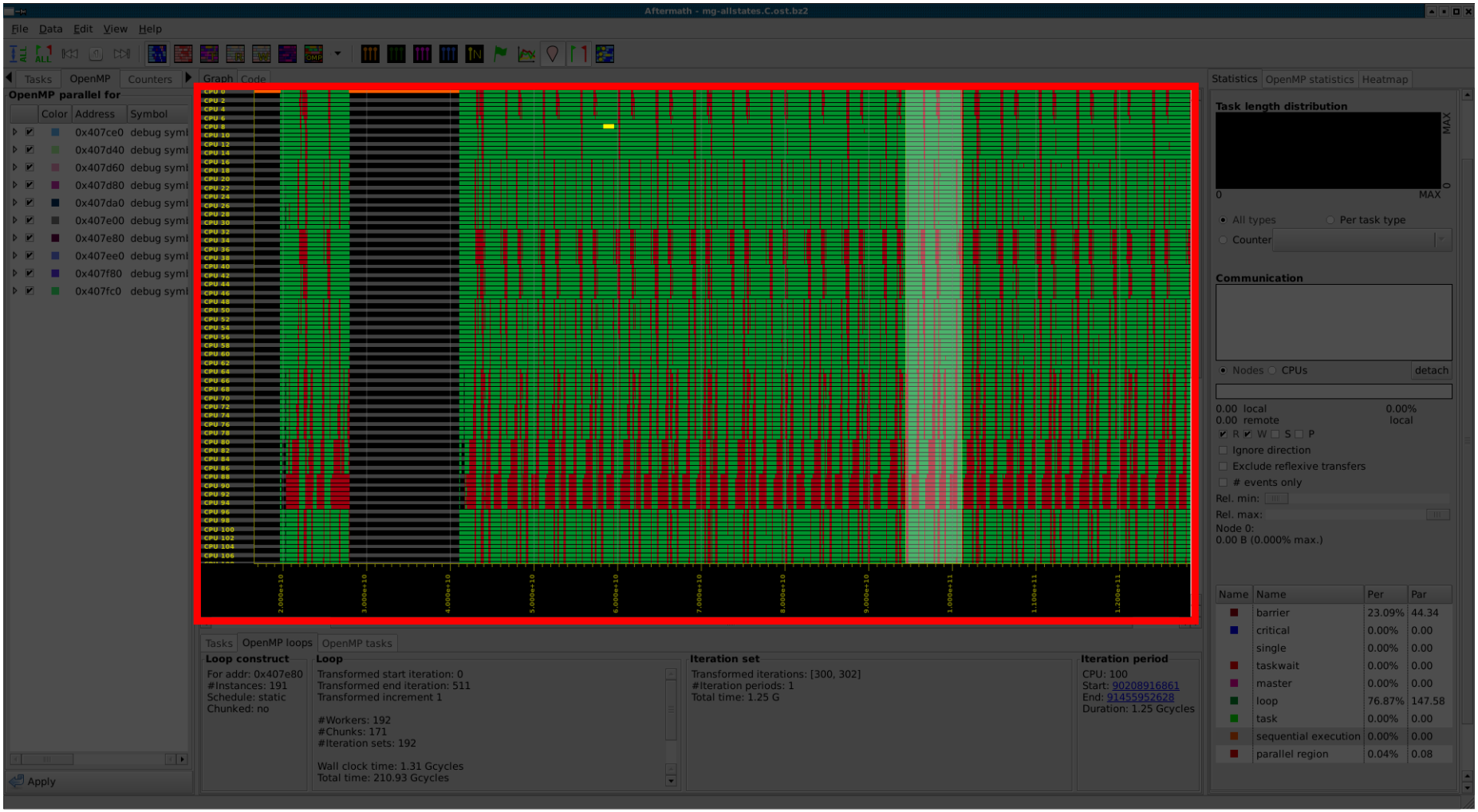
Aftermath: Overview of the GUI



Aftermath: Overview of the GUI

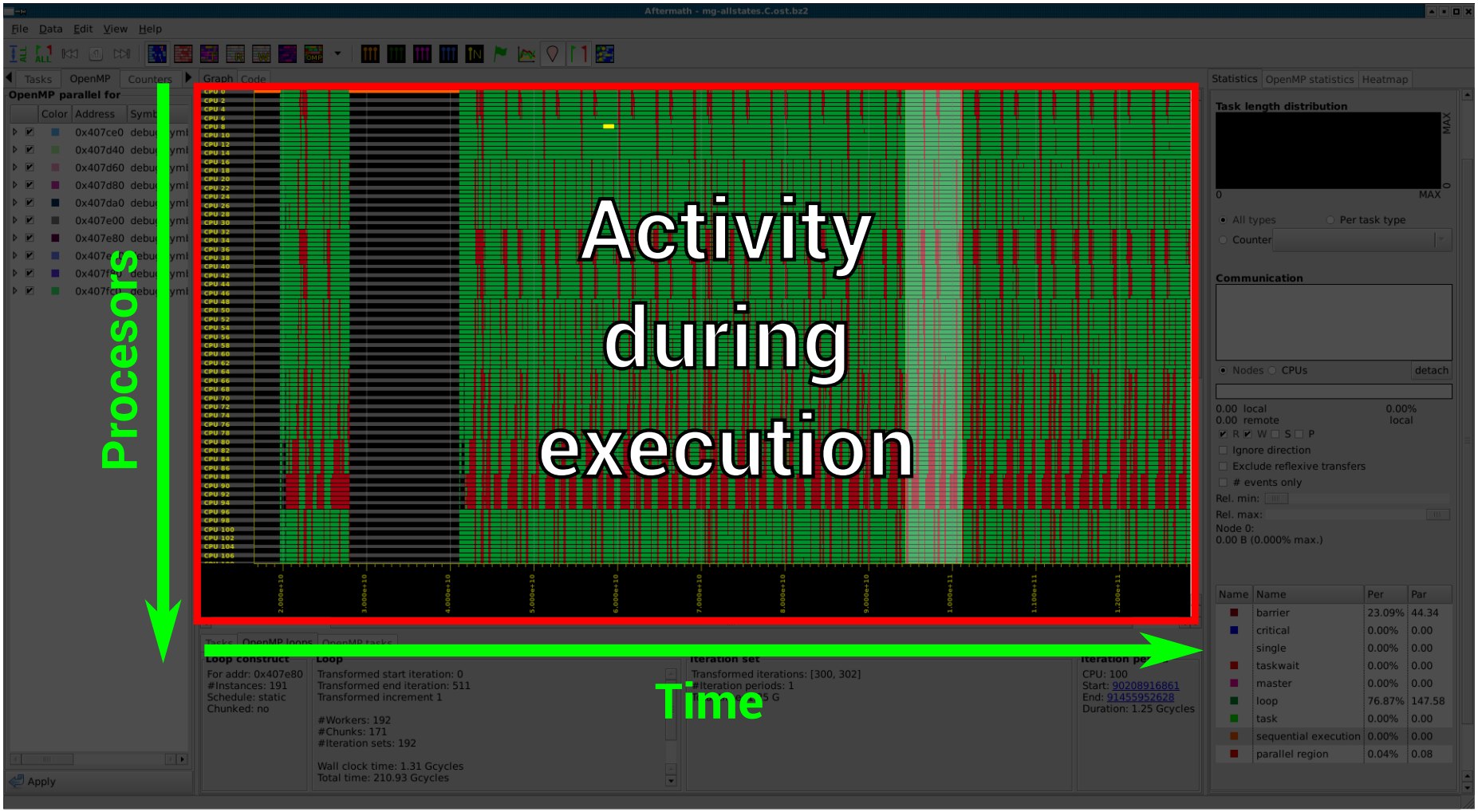


Aftermath: Overview of the GUI



Time line

Aftermath: Overview of the GUI

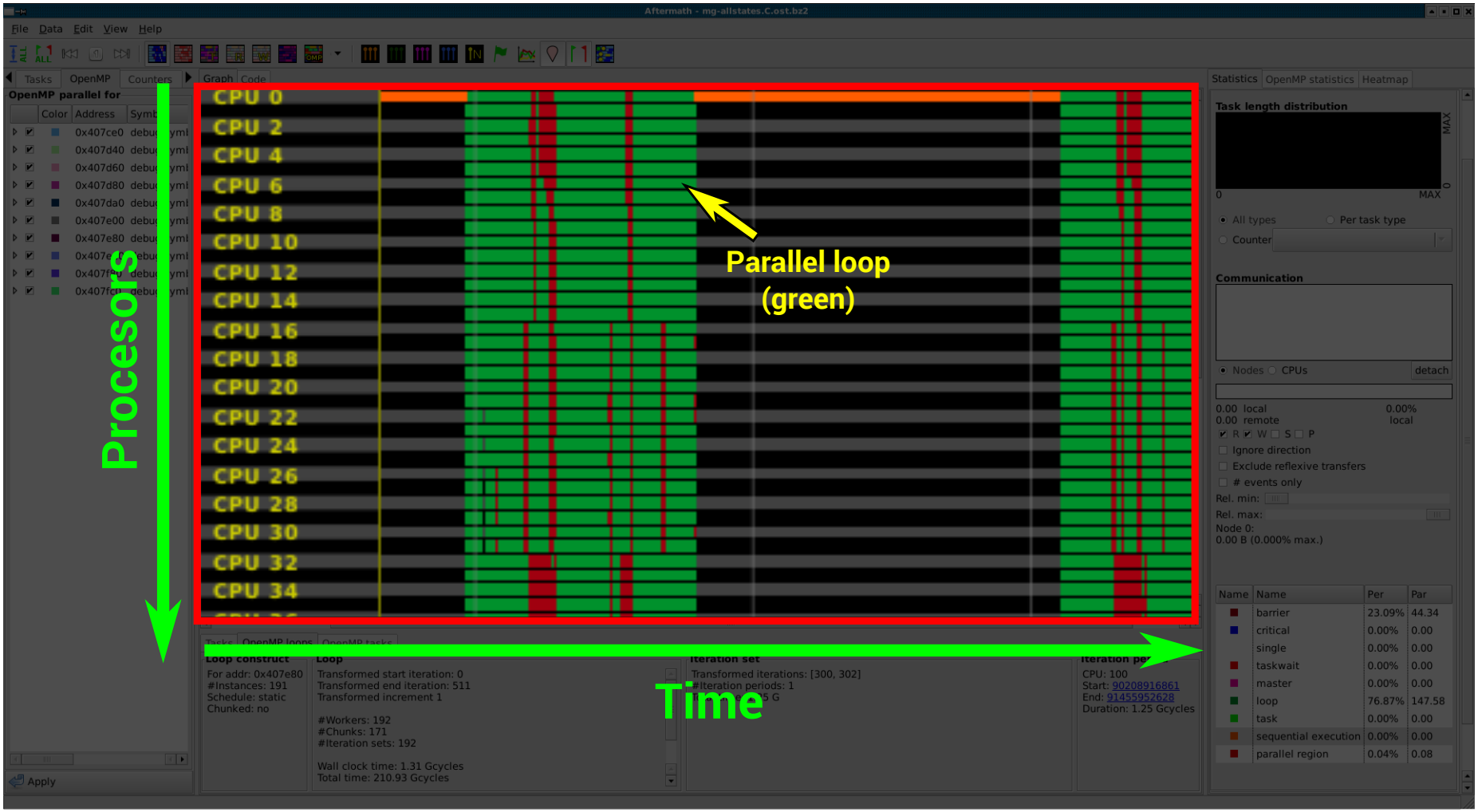


Time line

Aftermath: Overview of the GUI

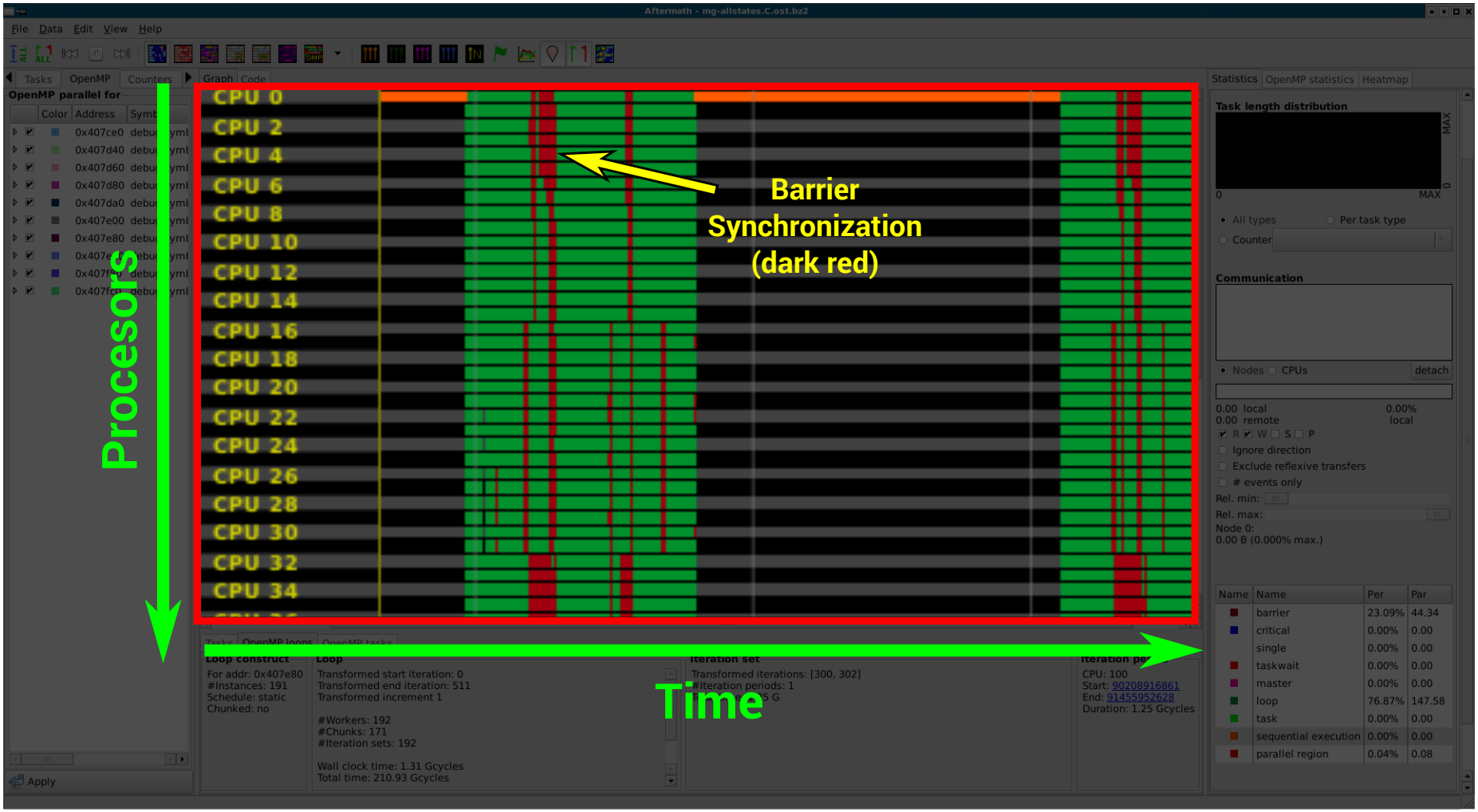
Time line: Run-time states

Aftermath: Overview of the GUI



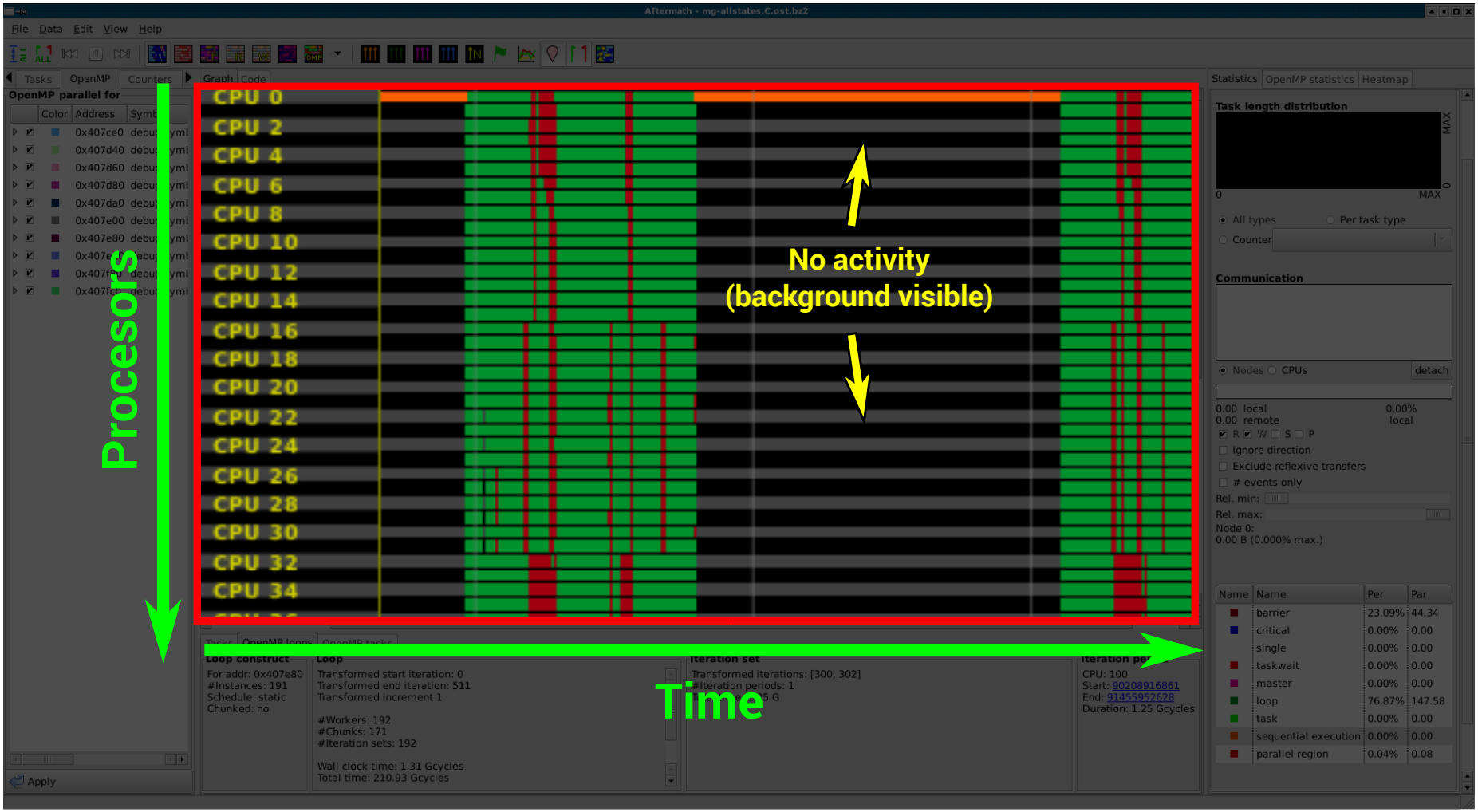
Time line: Run-time states

Aftermath: Overview of the GUI



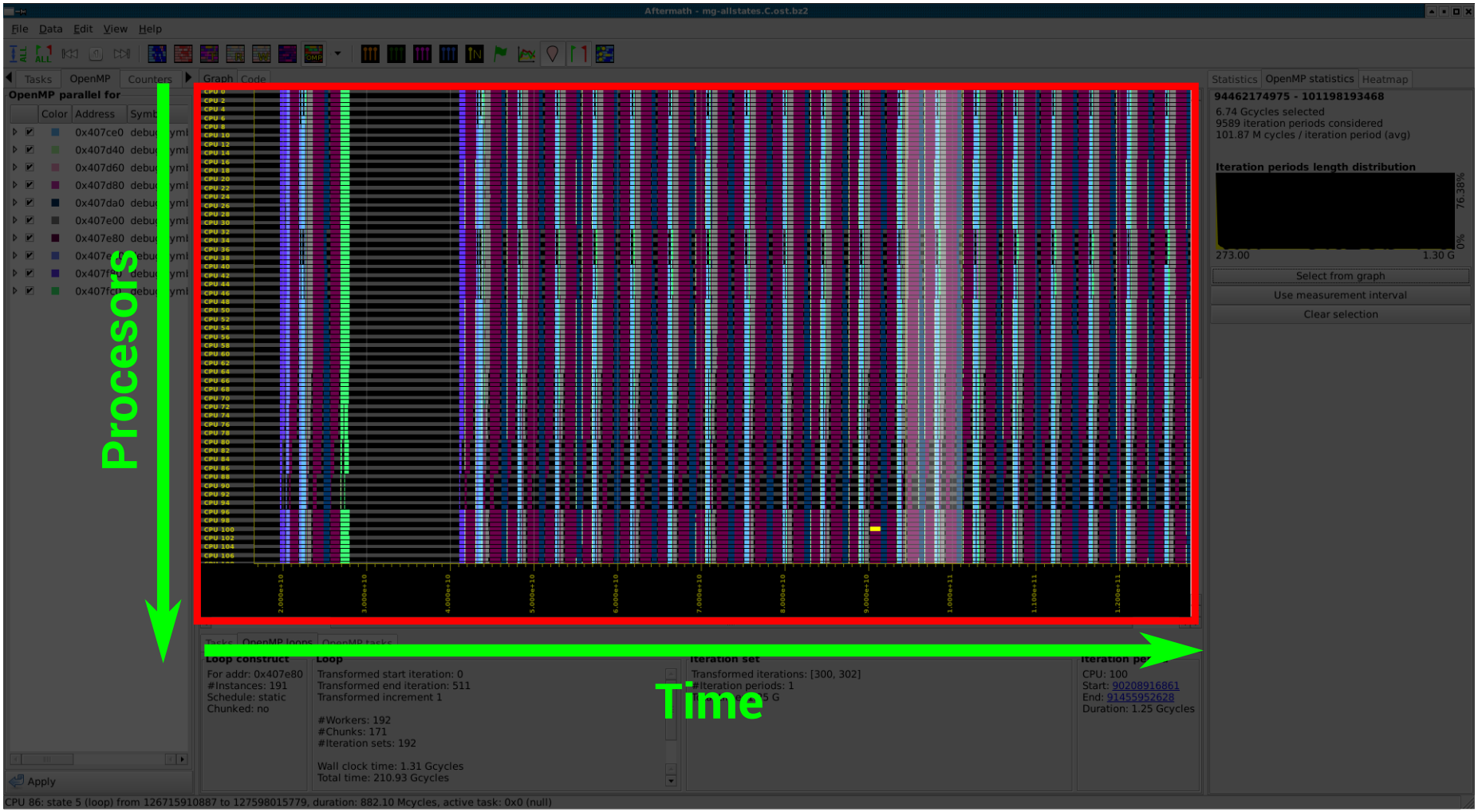
Time line: Run-time states

Aftermath: Overview of the GUI



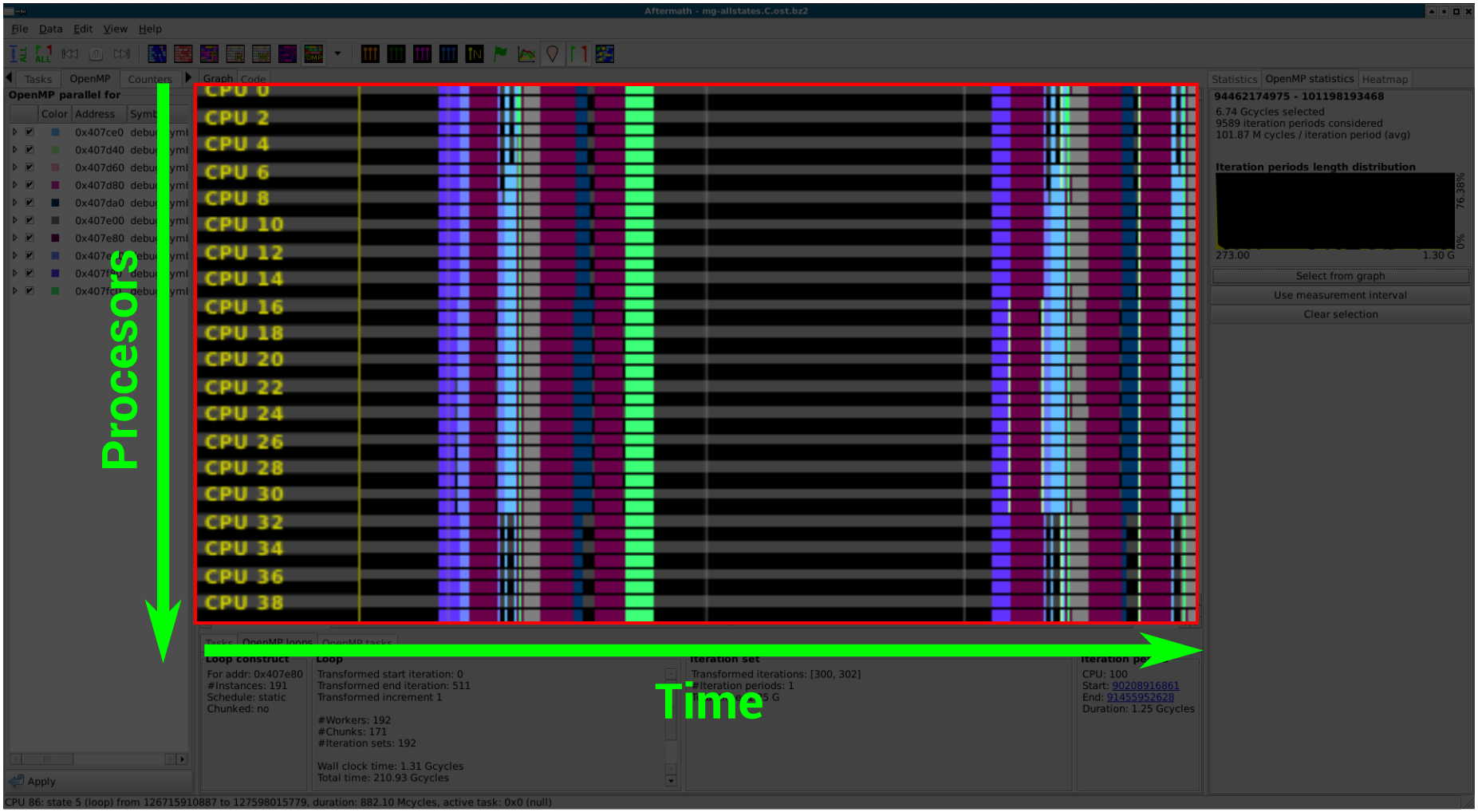
Time line: Run-time states

Aftermath: Overview of the GUI



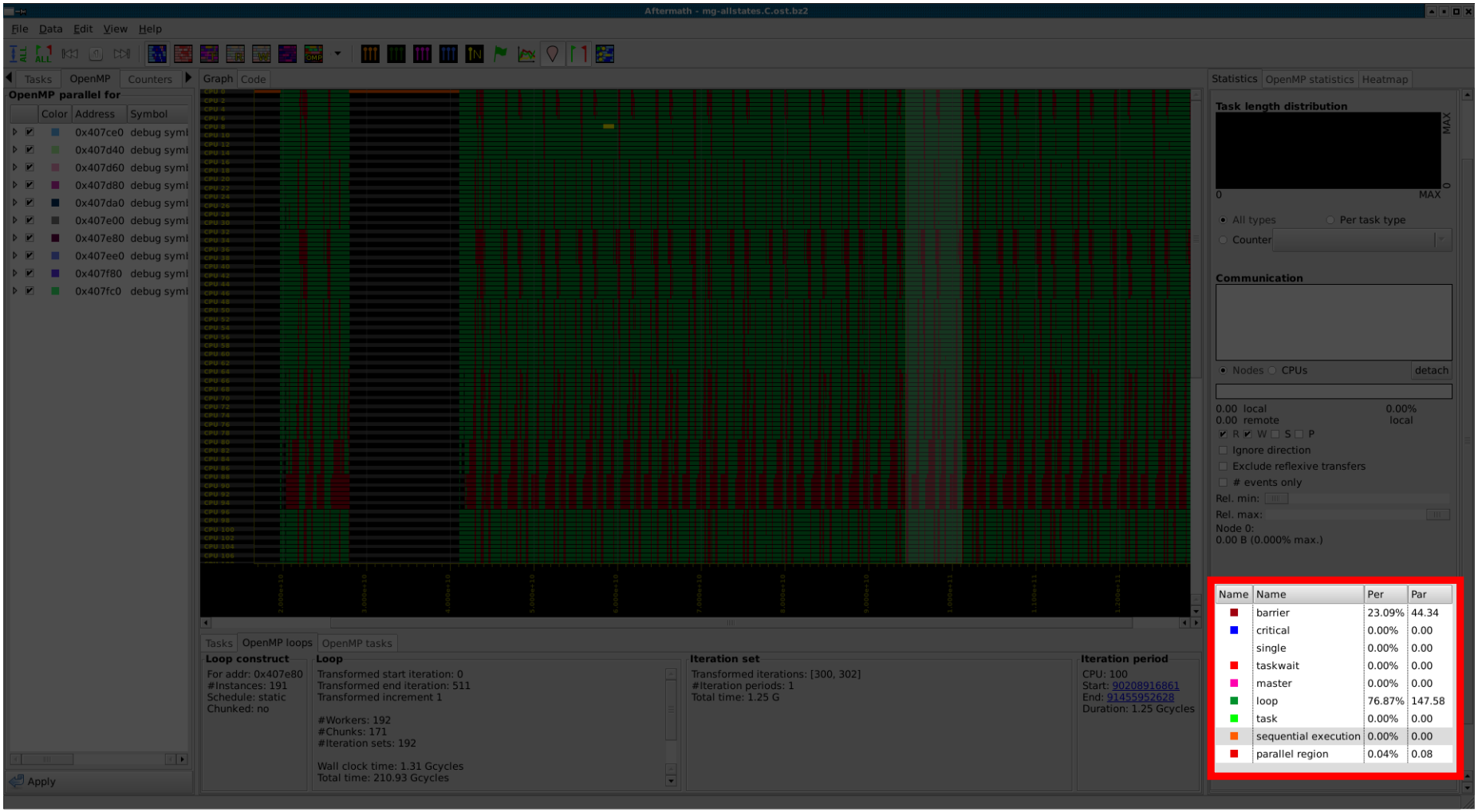
Time line: Loop constructs

Aftermath: Overview of the GUI



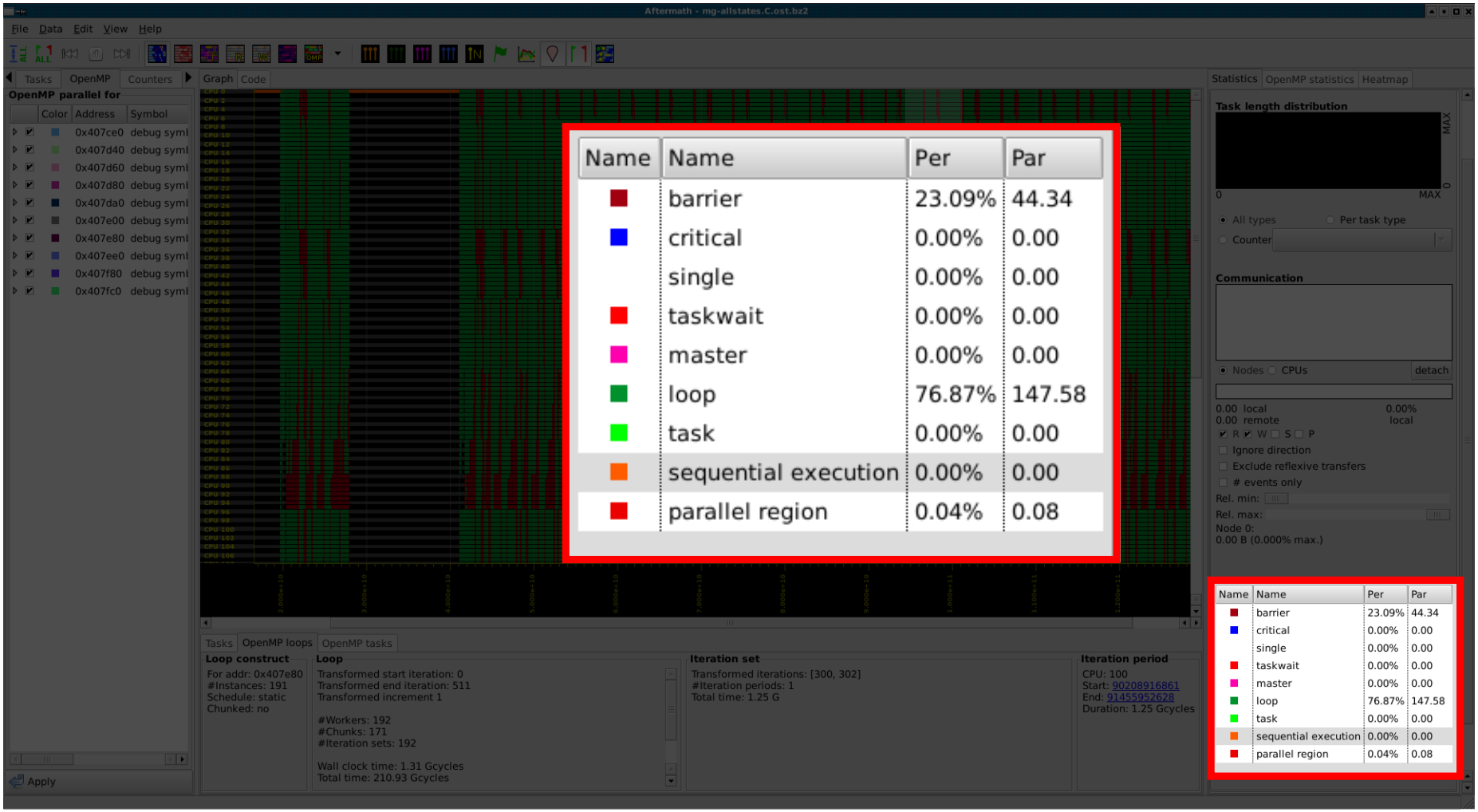
Time line: Loop constructs

Aftermath: Overview of the GUI



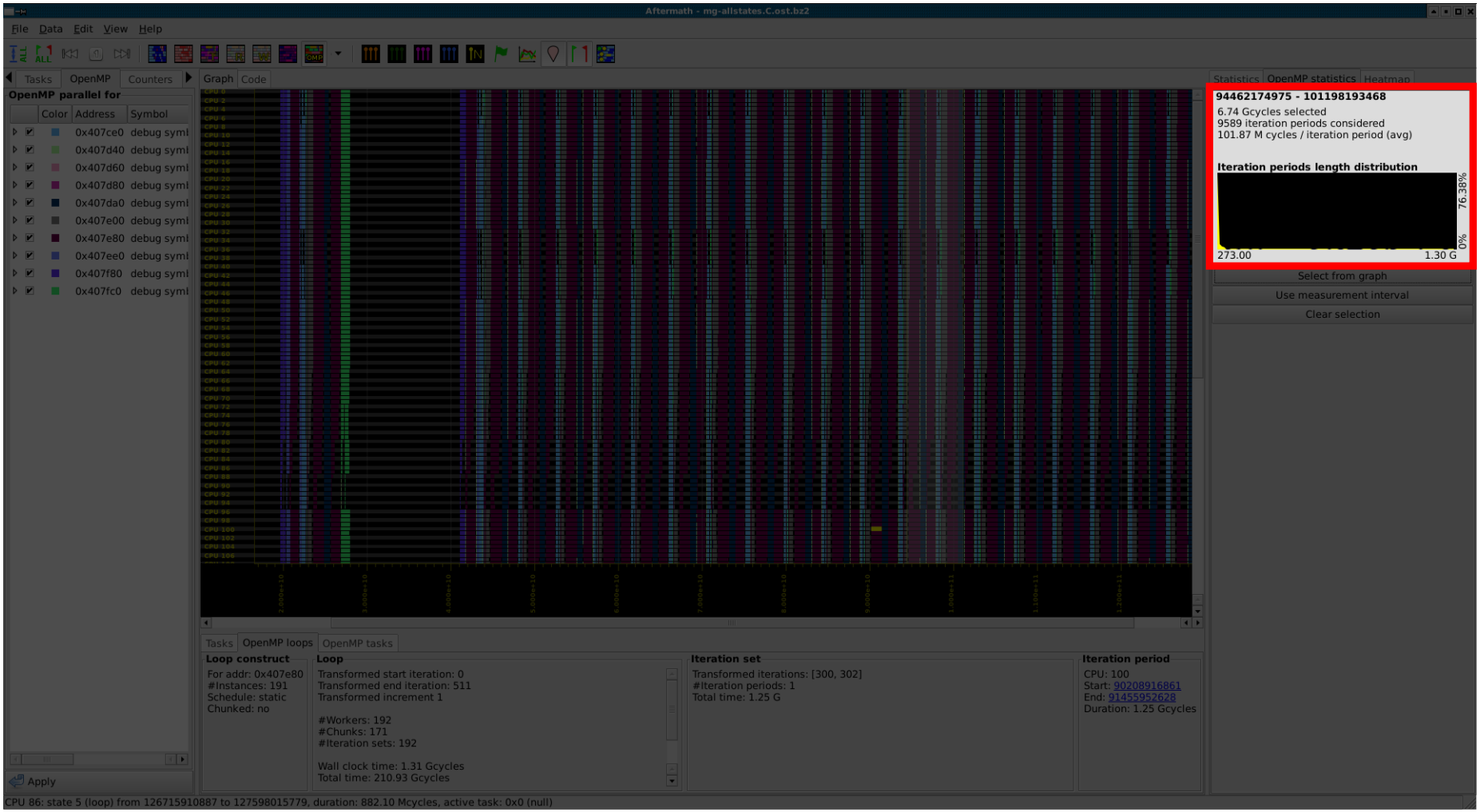
State statistics

Aftermath: Overview of the GUI



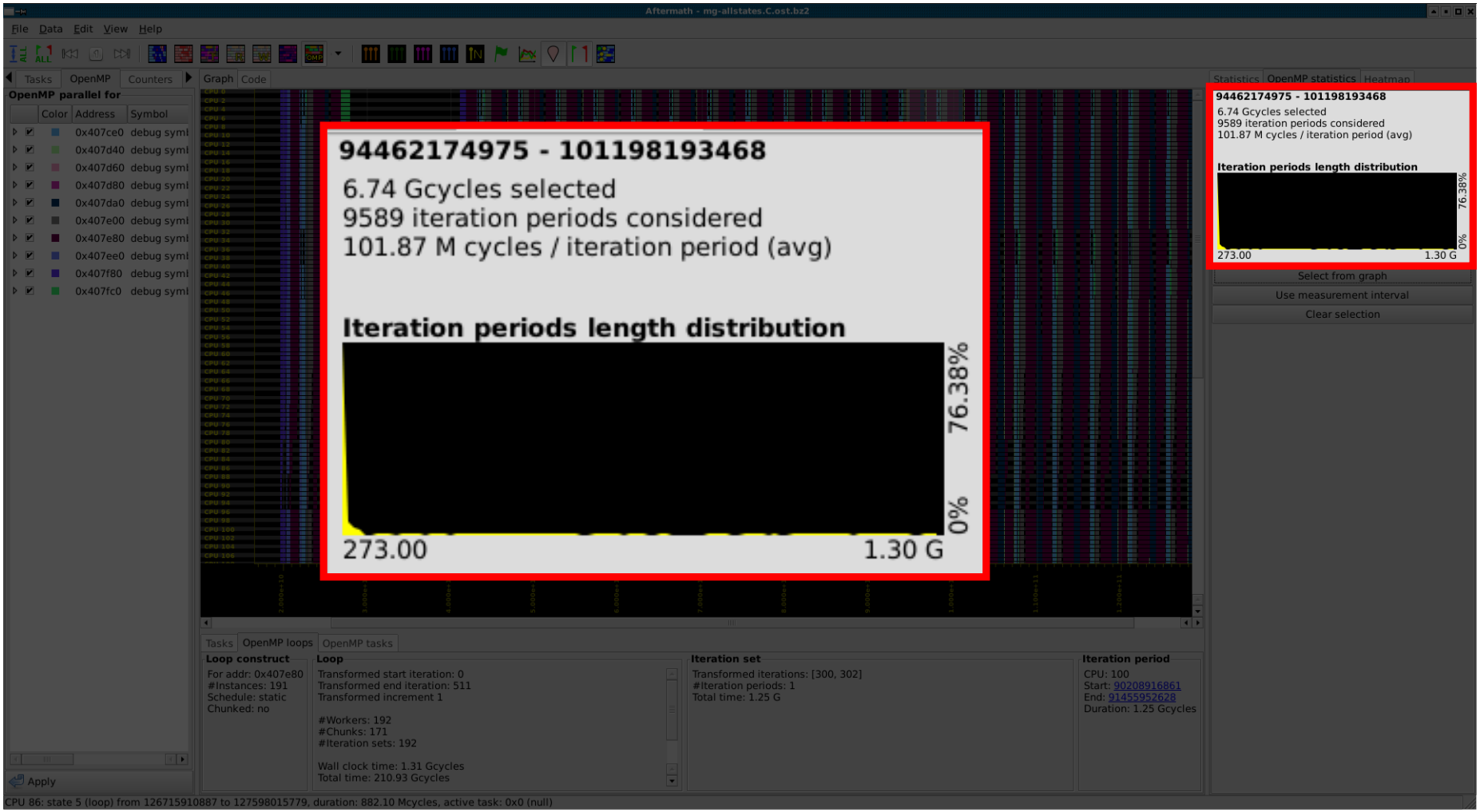
State statistics

Aftermath: Overview of the GUI



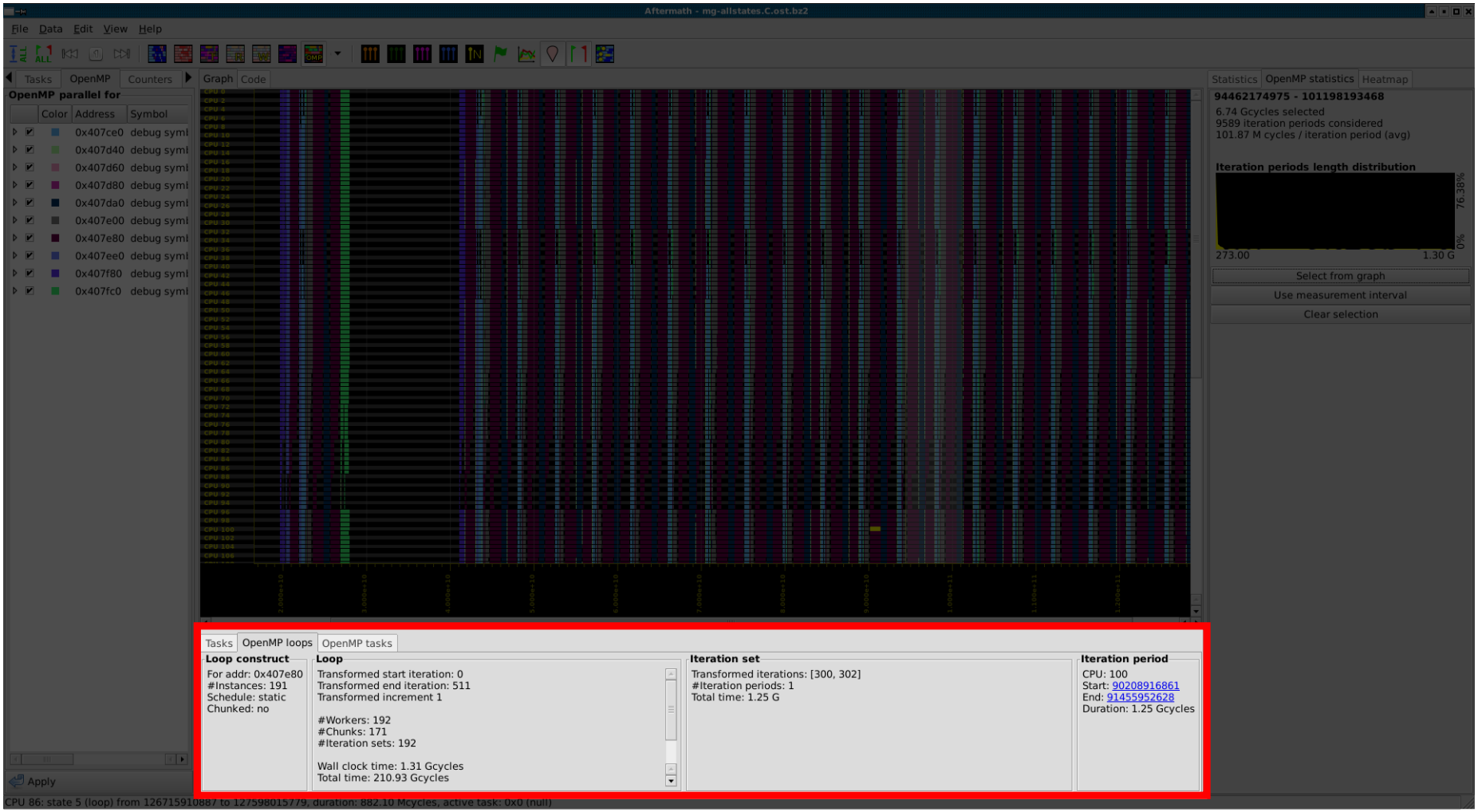
Histogram showing duration of iteration periods

Aftermath: Overview of the GUI



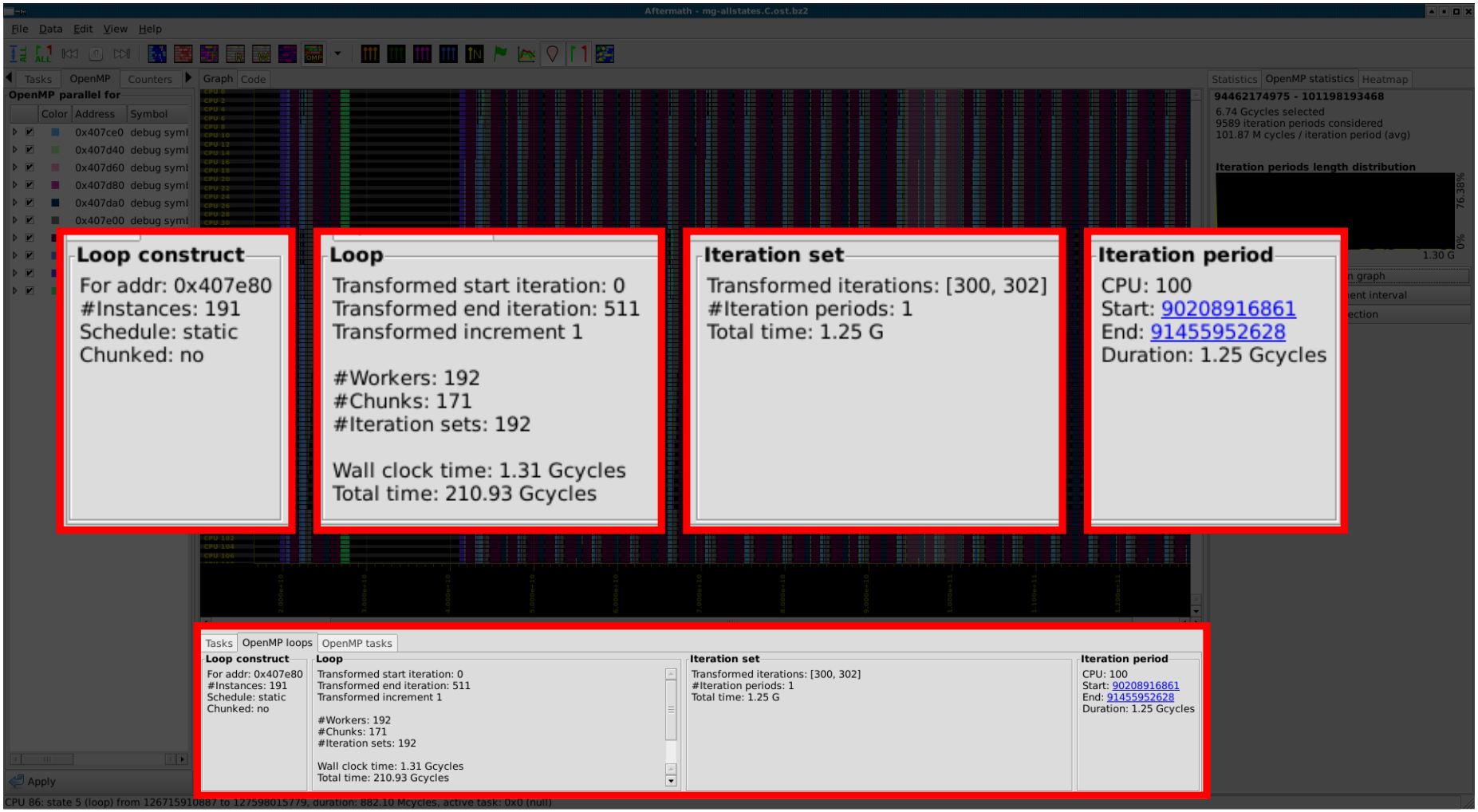
Histogram showing duration of iteration periods

Aftermath: Overview of the GUI



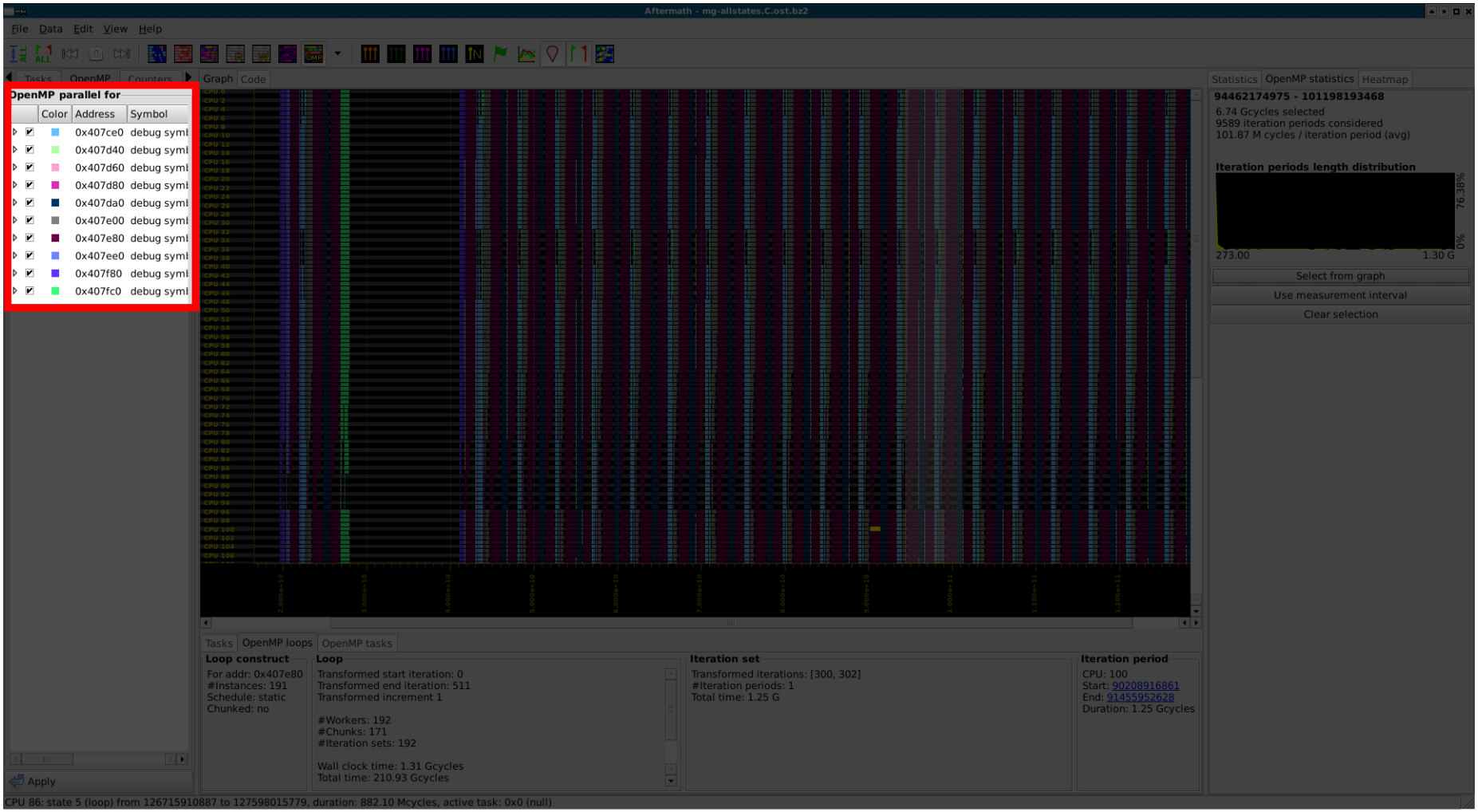
Detailed text view for parallel loops

Aftermath: Overview of the GUI



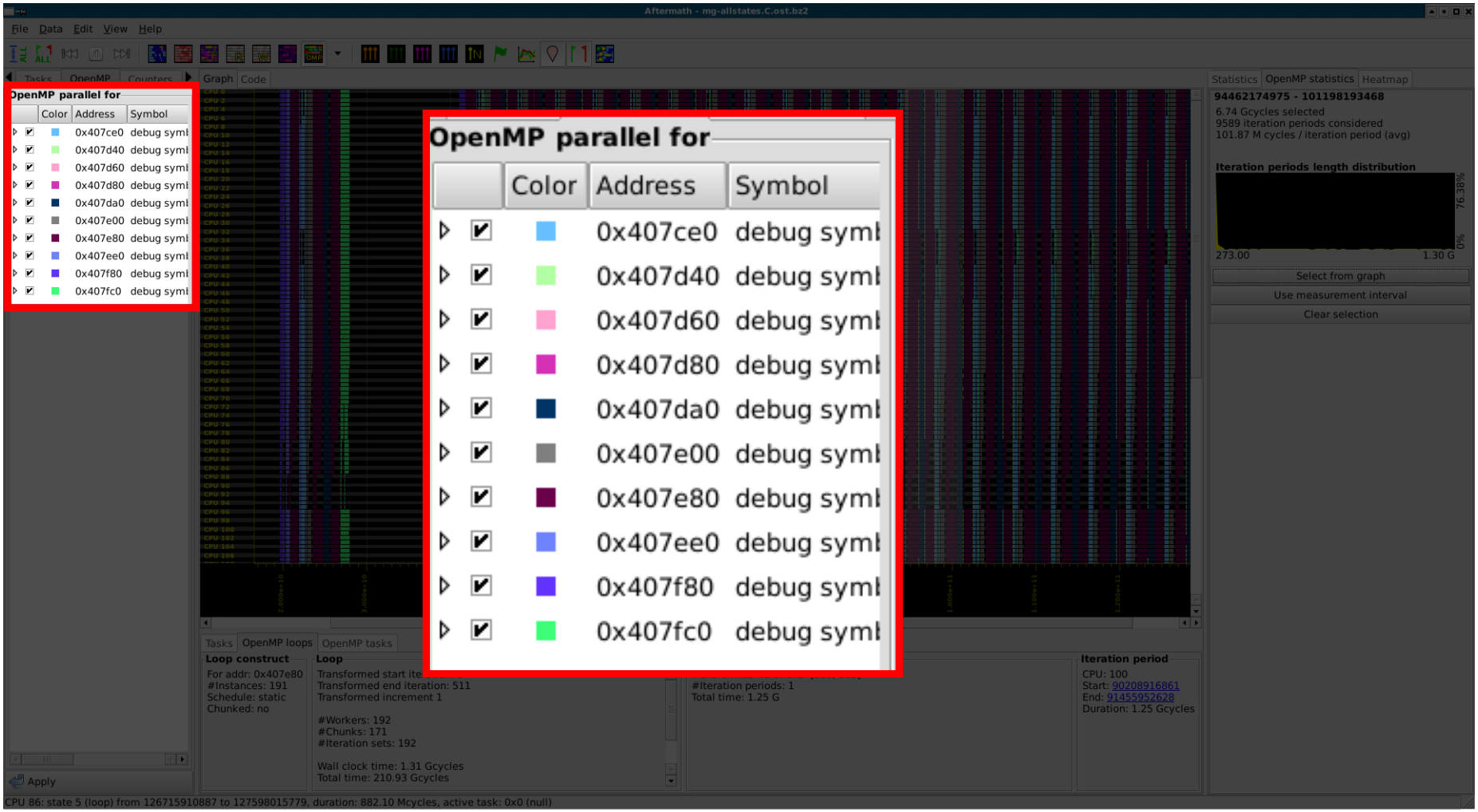
Detailed text view for parallel loops

Aftermath: Overview of the GUI



Filter for loop constructs

Aftermath: Overview of the GUI



Filter for loop constructs

Demo: NPB's MG benchmark

Benchmark: NPB MG

- NPB 2.3 C implementation from the Omni Compiler Project
- C input class (512×512 elements)

Test platform

- SGI UV 2000 (Xeon E5-4640)
- 192 cores (Hyperthreading disabled)
- 24 NUMA nodes, 756 GiB RAM
- LLVM/clang 3.8.0
- Aftermath-OpenMP for trace generation

DEMO

Demo: Summary

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Demo: Summary

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

- States on time line / statistics panel

Demo: Summary

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

- States on time line / statistics panel

Load imbalance

- Sufficient parallelism
- High load imbalance, but not due to partitioning / schedule
- Same NUMA node $\rightarrow$ Aprox. same execution time

Demo: Summary

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

- States on time line / statistics panel

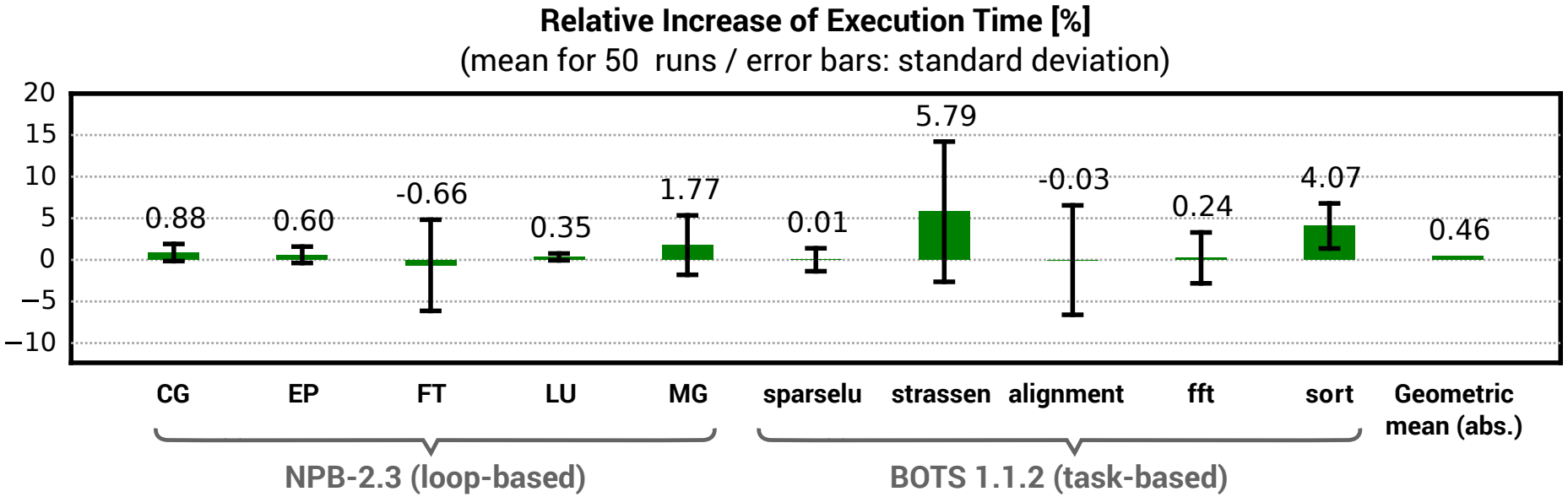
Load imbalance

- Sufficient parallelism
- High load imbalance, but not due to partitioning / schedule
- Same NUMA node $\rightarrow$ Aprox. same execution time

Solution

- Change allocation scheme: one big allocation
- Reduce number of workers: #iters = $n \times$ #workers
- Result: 35$\times$ speedup

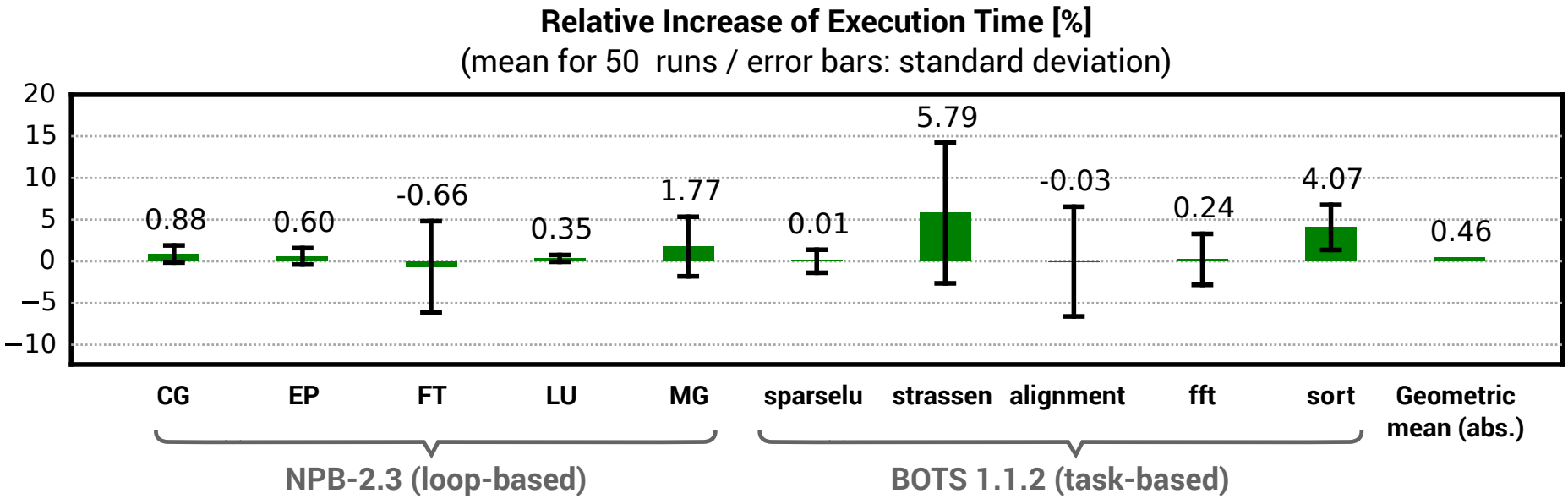
Overhead of Tracing



Test system

- SGI UV 2000 (192 cores, 24 NUMA nodes)

Overhead of Tracing



Test system

- SGI UV 2000 (192 cores, 24 NUMA nodes)

Missing benchmarks

- Outlier: *floorplan* (+380% execution time; very small tasks)
- Segfaults (*BT*, *nqueens*, *uts*) / Excessive Execution time (*IS*) / Verification Failure (*health*)

Using Aftermath & Aftermath-OpenMP

Drop-in replacement for libomp with wrapper script:

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

Using Aftermath & Aftermath-OpenMP

Drop-in replacement for libomp with wrapper script:

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

Source code and tutorial:

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

Virtual Machine
(Aftermath + Aftermath-OpenMP + sample traces + documentation):

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

Summary

Aftermath

- Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Summary

Aftermath

- Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Summary

Aftermath

- Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Future work

- Dependent tasks
- Automate recurring analyses

Summary

Aftermath

- Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Future work

- Dependent tasks
- Automate recurring analyses

On-line resources

`http://www.openstream.info/aftermath` (Main website)
`http://www.openstream.info/vm` (VM image)