Aftermath

A graphical tool for performance analysis and debugging of fine-grained task-parallel programs and run-time systems

Andi Drebes

Université Pierre et Marie Curie
Laboratoire d'Informatique de Paris VI
andi.drebes@lip6.fr

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

MULTIPROG'14, January 22nd, 2014

UPMC SORBONNE UNIVERSITÉS

LIP6

OpenStream

Context

Hardware and software environment

- Multi-core / many-core NUMA systems
- Task-parallel languages based on fine-grained tasks

Context

Hardware and software environment

- Multi-core / many-core NUMA systems
- Task-parallel languages based on fine-grained tasks

Efficient exploitation of the machine

- Enough parallelism to keep processors busy
- Data locality on NUMA systems
- Low management overhead
- Requires analysis of complex interactions:
  Application / Run-time / Machine

Context

Hardware and software environment

- Multi-core / many-core NUMA systems
- Task-parallel languages based on fine-grained tasks

Efficient exploitation of the machine

- Enough parallelism to keep processors busy
- Data locality on NUMA systems
- Low management overhead
- Requires analysis of complex interactions:
  Application / Run-time / Machine

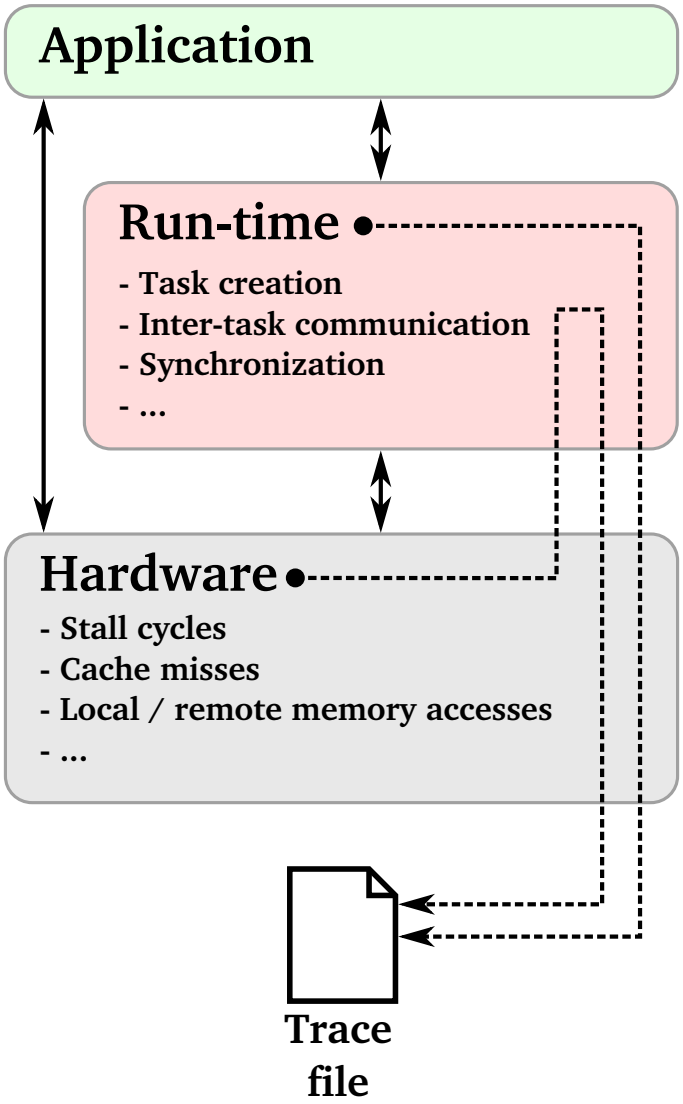
Tools for performance debugging

- Trace-based analysis

Outline

1. Trace-based analysis
2. Overview on Aftermath
3. Live demo
4. Use cases
5. Summary & Questions

Capturing event data



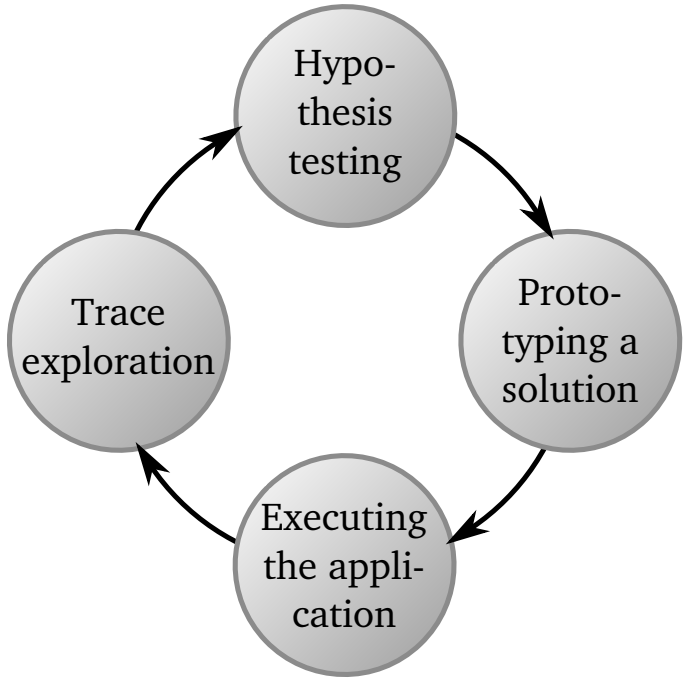
Trace data

- Captured during execution
- Event types
  - Discrete events
  - State changes
  - Counter values
- High-level events
- Low-level events
- Dumped to trace file

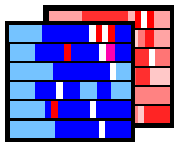
Off-line analysis

Off-line analysis scenarios

- Trace exploration
  - Optimization opportunities
  - Presence of a bottleneck?
  - No precise idea where to look
- Hypotheses testing
  - Precise idea of performance anomaly
  - Check if the hypothesis holds
- Cyclic pattern during application / run-time development and debugging
- Need for a tool supporting exploration *and* hypotheses testing



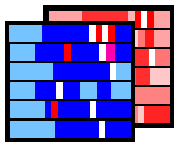
Requirements



Graphical representation

- Static and dynamic information
- Make strong correlations visible

Requirements



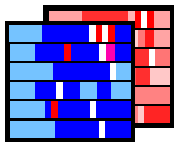
Graphical representation

- Static and dynamic information
- Make strong correlations visible

Filters

- Control over amount of detail
  - Task type, duration, processors, ...
- Ability to combine filters

Requirements

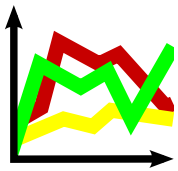


Graphical representation

- Static and dynamic information
- Make strong correlations visible

Filters

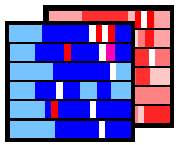
- Control over amount of detail
  - Task type, duration, processors, ...
- Ability to combine filters



Statistical counters

- Quantifying events
- Derived metrics

Requirements

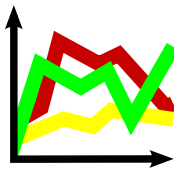


Graphical representation

- Static and dynamic information
- Make strong correlations visible

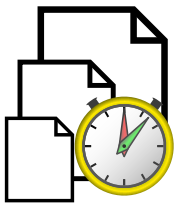
Filters

- Control over amount of detail
  - Task type, duration, processors, ...
- Ability to combine filters



Statistical counters

- Quantifying events
- Derived metrics



User interface

- Support for large trace files
- Reactive UI with immediate feedback

Why a new tool?

Existing approaches

- Trace analysis: Well-known technique, widely used in HPC
- Existing tools: Paraver, Vampir, Paraprof / TAU, ...
- Optimized for clusters / message passing
- Many good ideas in these tools, but none of them fully satisfied our requirements

Why a new tool?

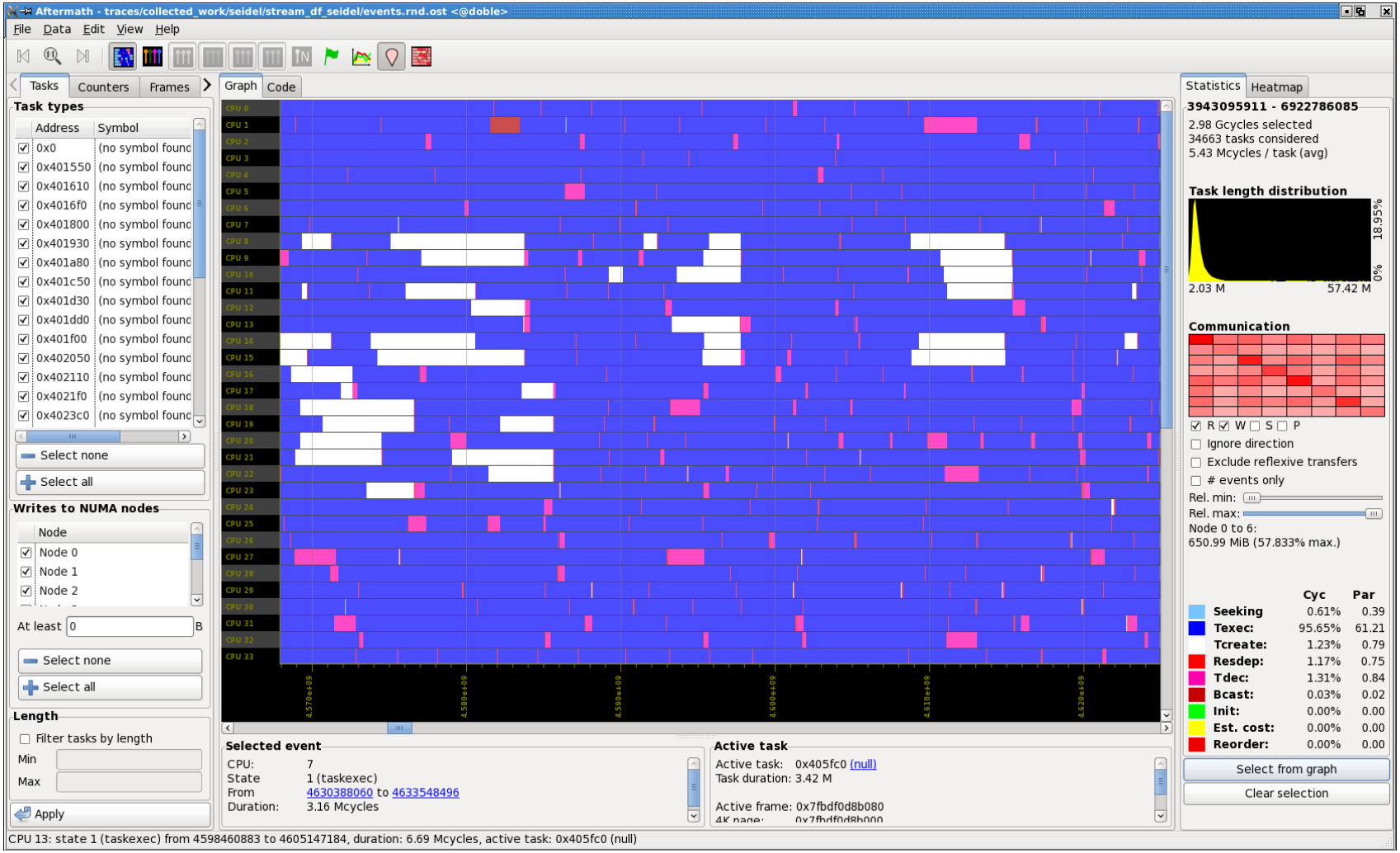
Existing approaches

- Trace analysis: Well-known technique, widely used in HPC
- Existing tools: Paraver, Vampir, Paraprof / TAU, ...
- Optimized for clusters / message passing
- Many good ideas in these tools, but none of them fully satisfied our requirements

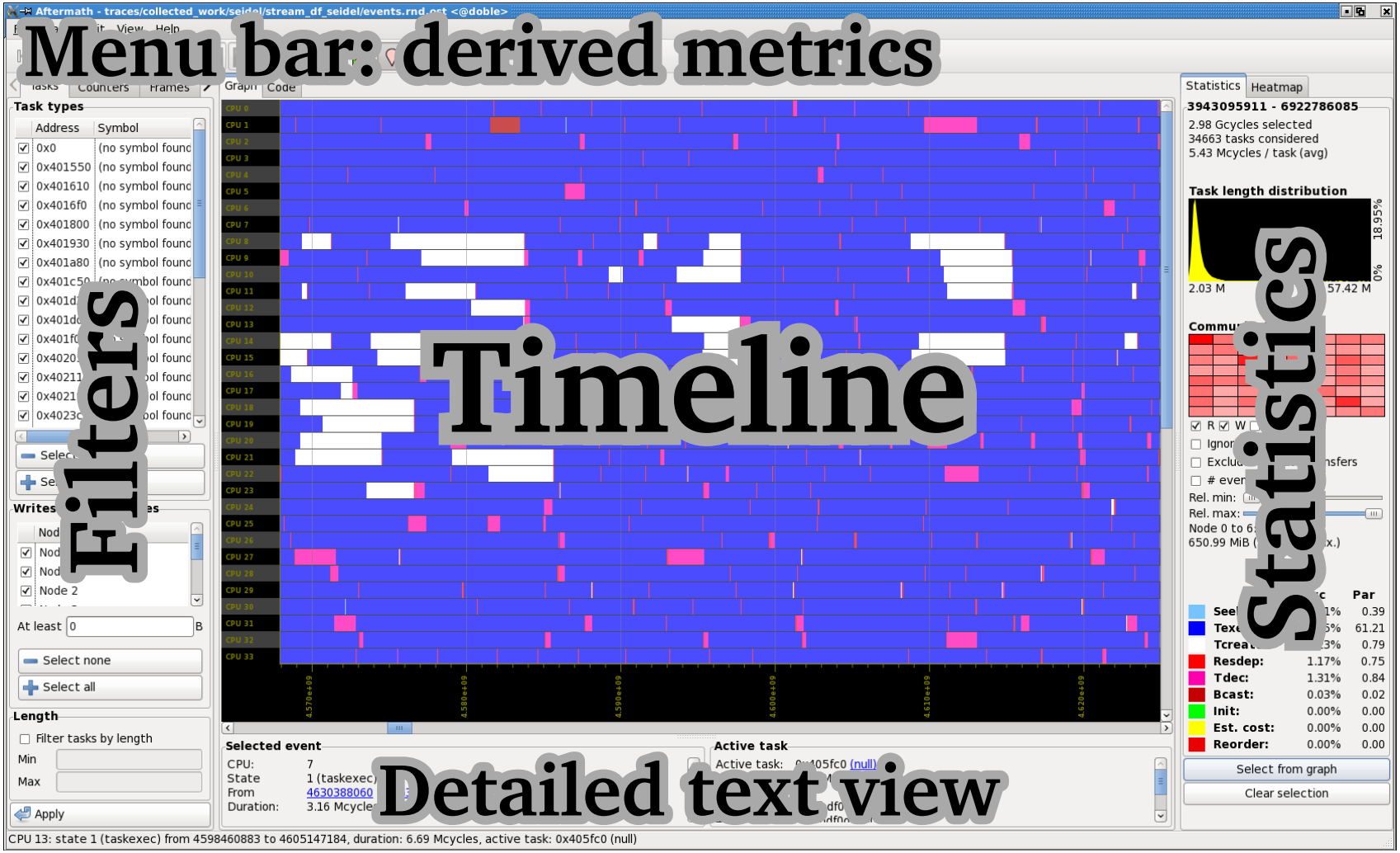
Additional requirements in the OpenStream project

- OpenStream: OpenMP extension, fine-grained data-flow tasks
- Native support for tasks
- Resource-model
  - NUMA analysis
  - Data-flow analysis
- Interactive exploration

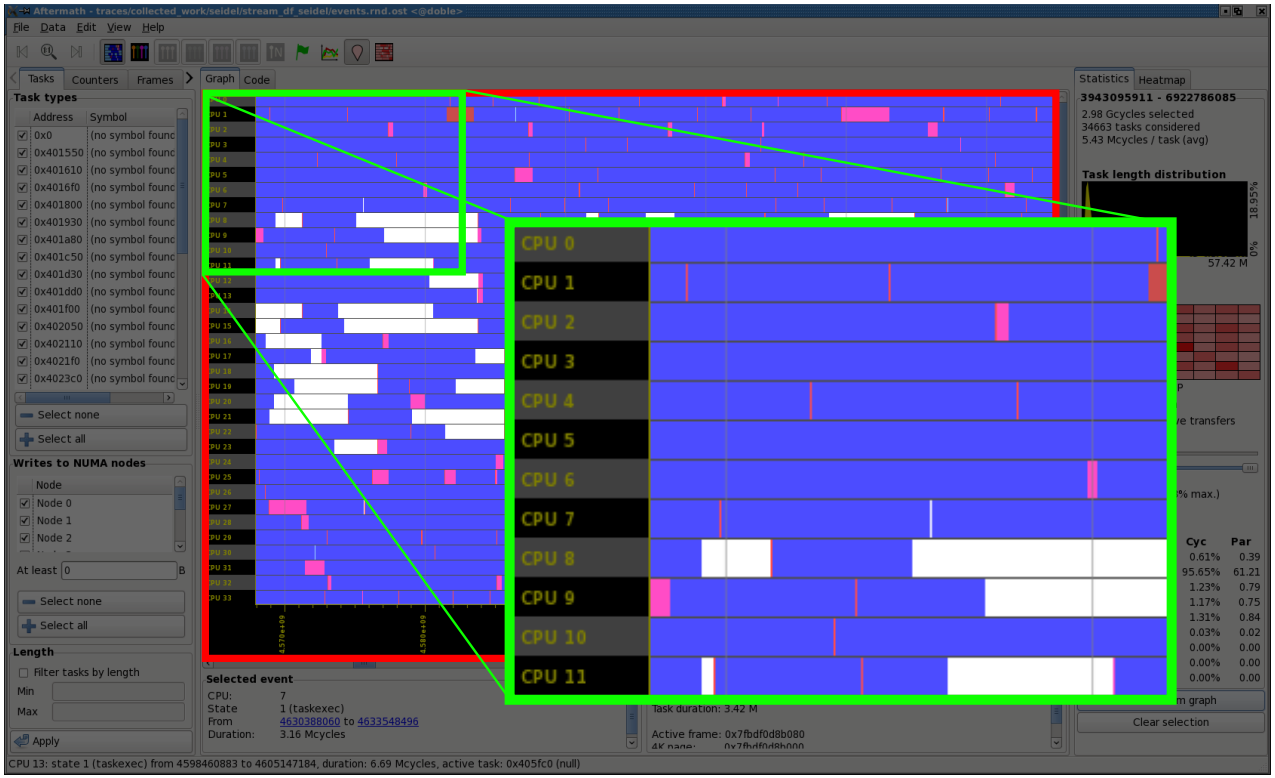
Aftermath: main user interface



Aftermath: main user interface



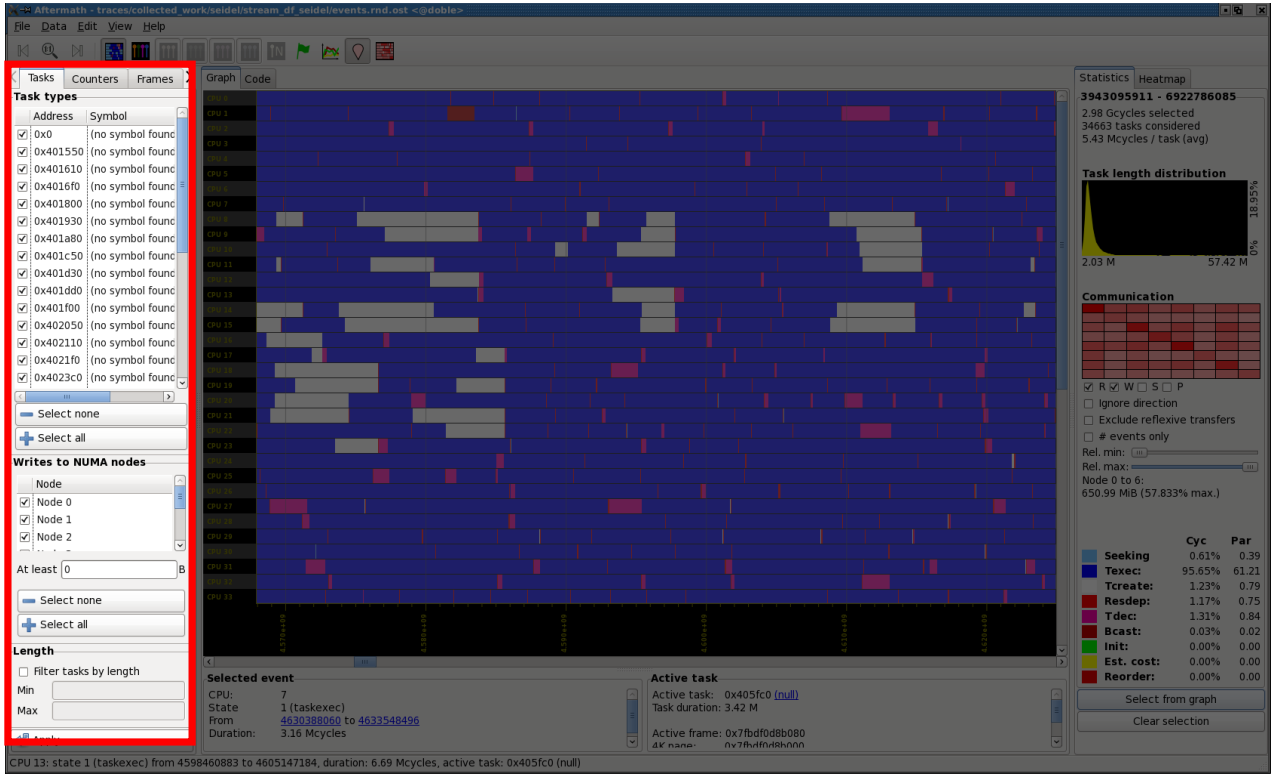
Aftermath: main user interface



Timeline component

- Processor activity over time
  - State changes
  - Communication
  - Hardware counters
- One “lane” per CPU
- Multiple views:
  States / Heatmap / NUMA

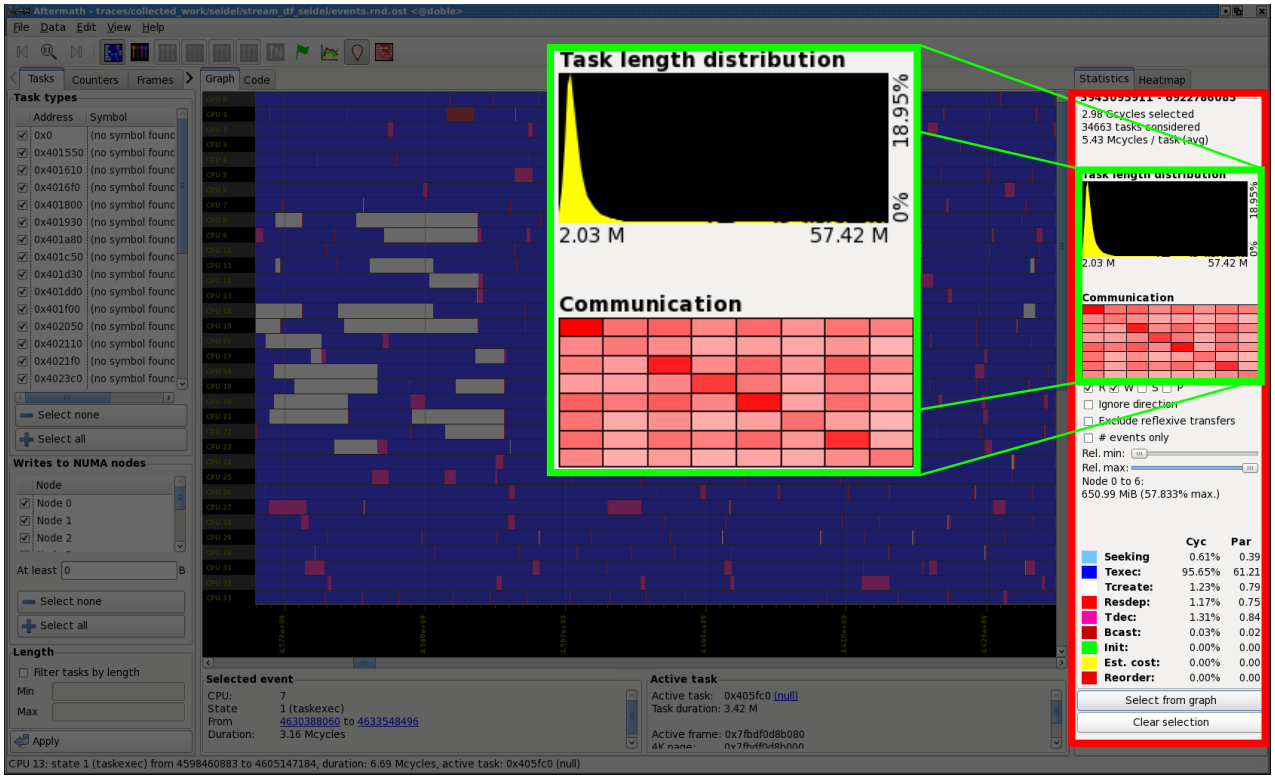
Aftermath: main user interface



Filters

- Task-related (type, length)
- Subset of CPUs
- Event types
- Communication (size, type)
- Data-flow
  - Specific data-flow buffers
  - Specific NUMA nodes

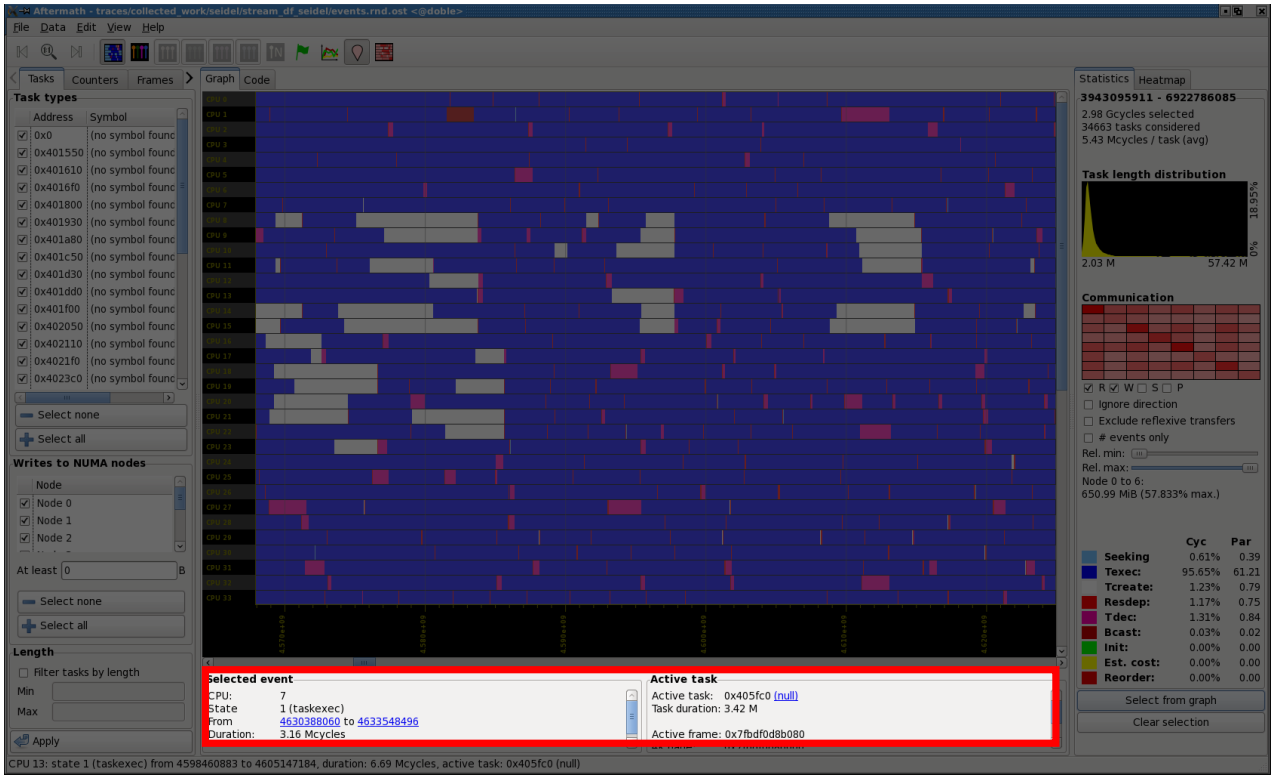
Aftermath: main user interface



Statistical graphs and counters

- Selectable area
- Number of tasks
- Average duration
- Time spent in different states
- Task duration histogram
- NUMA node comm. matrix

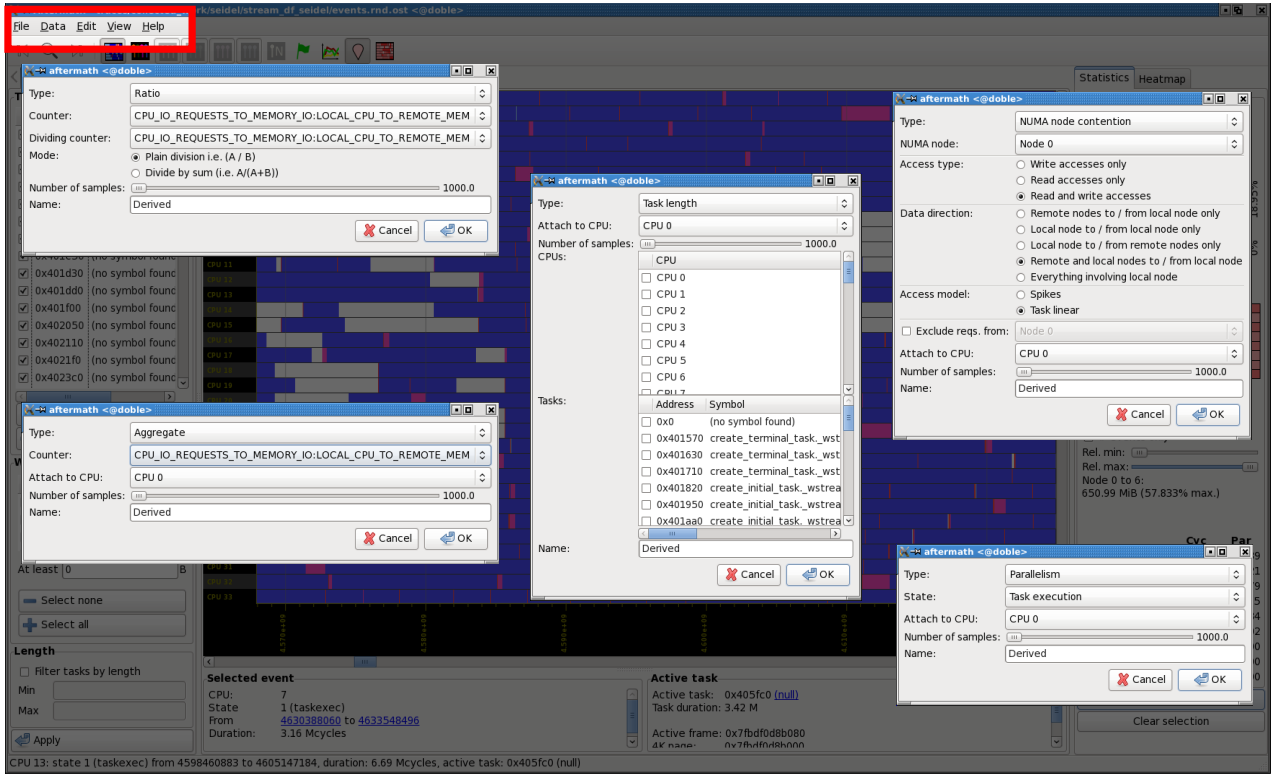
Aftermath: main user interface



Detailed textual information

- Selected task / state
- Task / state duration
- Task creation
- Data-flow information
  - Producers / Consumers
  - Reads / writes per NUMA node

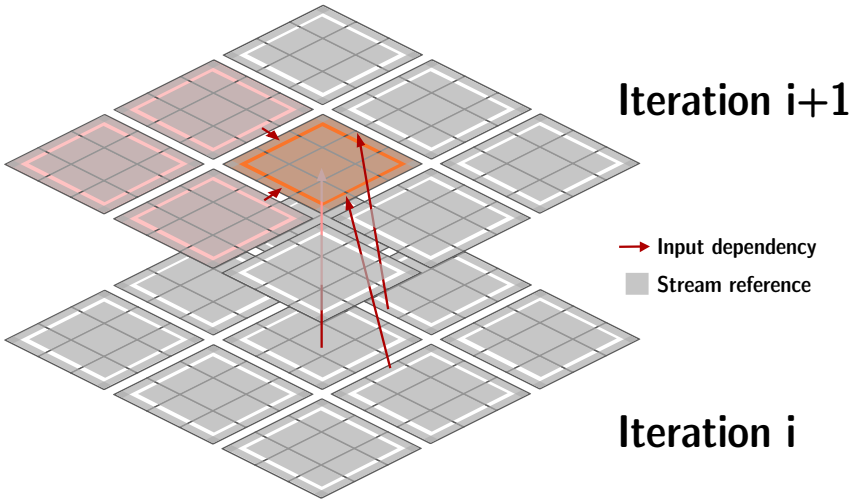
Aftermath: main user interface



Derived metrics

- Aggregation of CPUs
- Ratio of counters
- Parallelism on average
- NUMA node contention
- Average task duration

Example trace: Gauß-Seidel iterations



Benchmark from the OpenStream project

- Data-flow implementation of a 5-point stencil
- Main tasks + Auxiliary tasks (initialization, termination)
- Two traces: unoptimized / optimized task and data placement

Hardware environment

- 64 cores
- 8 NUMA domains

LIVE DEMO

Use cases

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

Summary

Aftermath at a glance

- Fast and responsive UI
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Multiple connected views
- Successfully used for performance debugging of the OpenStream run-time and applications

Summary

Aftermath at a glance

- Fast and responsive UI
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Multiple connected views
- Successfully used for performance debugging of the OpenStream run-time and applications

Source code and tutorial available at

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

- Free software license, small code base ($<$ 15k lines of C code)
- Few dependences: GTK+, Cairo, Pango, Glibc, ...

Summary

Aftermath at a glance

- Fast and responsive UI
- Various filters with immediate visual feedback
- Native support for dependent tasks
- Multiple connected views
- Successfully used for performance debugging of the OpenStream run-time and applications

Source code and tutorial available at

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

- Free software license, small code base ($<$ 15k lines of C code)
- Few dependences: GTK+, Cairo, Pango, Glibc, ...

Future Work

- More detailed resource model
- Trace compression
- Your idea here!