OpenMP Tools API (OMPT): Ready for Prime Time?

John Mellor-Crummey
Department of Computer Science
Rice University

OMPT: OpenMP Performance Tools API

- Goal: a standardized tool interface for OpenMP
  - prerequisite for portable tools for debugging and performance analysis
  - missing piece of the OpenMP language standard
- Design objectives
  - enable tools to measure and attribute costs to application source and runtime system
    - support low-overhead tools based on asynchronous sampling
    - attribute to user-level calling contexts
    - associate a thread's activity at any point with a descriptive state
  - minimize overhead if OMPT interface is not in use
    - features that may increase overhead are optional
  - define interface for trace-based performance tools
  - don't impose an unreasonable development burden
    - runtime implementers
    - tool developers

OMPT Chronology

- 2012
  - Began design at CScADS Performance Tools Workshop
- 2013
  - Intel released OpenMP runtime as open source
  - Began development of OMPT prototype in Intel OpenMP runtime
- 2014
  - Refined design & implementation based on experience with applications
  - OMPT Technical Report 2 accepted by OpenMP ARB
- 2015
  - Hardened OMPT implementation in Intel OpenMP runtime
    - support nested parallelism and tasks for both Intel and GNU APIs
  - Developed OMPT test suite
  - Contributed OMPT patches to LLVM OpenMP
  - Began design of OMPT extensions for accelerators

OMPT Support is Non-trivial

- OMPT assigns and maintains ids for both implicit and explicit tasks
  - compilers use the runtime differently
    - Intel compiler: runtime system always calls outlined parallel regions
    - GNU compiler: master calls outlined region between calls to the runtime
  - handling degenerate nested parallel regions is tricky
    - stack-allocate task state for degenerate regions for Intel compiler
    - heap-allocate task state for degenerate regions for GNU compiler
  - managing team reuse requires care
- Maintaining runtime state is also tricky
  - differentiate between
    - <u>idle</u> after arriving at a barrier ending a parallel region
    - <u>waiting at a barrier</u> in a parallel region
- More difficult for a third party developer after the fact!
- Implementation is not yet fully realized: more states, trace events

OMPT Test Suite

Goals

- Validate an implementation of OMPT in any OpenMP runtime
- Check correctness of OMPT independent of any tool
- Operate correctly with any OpenMP compiler
- Help resolve bugs experienced by OMPT tools being co-evolved

OMPT Test Suite Scope

- Regression tests
  - mandatory support
    - initialization
    - events
      - thread begin/end
      - parallel region begin/end
      - task begin/end
      - shutdown
      - user control
    - inquiry operations
      - get parallel region id
      - get task id - implicit and explicit tasks
      - get task frame
      - get state
  - blame shifting events
  - tracing events (largely unimplemented)
- Makefiles
  - LLVM runtime
    - Intel compilers: x86_64, mic
    - GNU compilers
  - IBM's runtime + XL compilers

Correctness criteria

- unique ids: threads, regions, tasks
- presence of required callbacks
- sequencing of event callbacks
- appropriate arguments to callbacks

if main is compiled with -openmp, Intel compiler initializes runtime immediately upon entering main

Intel runtime calls OpenMP shutdown after main exits!

testing some states, e.g., barrier, idle, lock wait is subtle

OpenMPToolsInterface Project

A shared repository for collaboration

- OMPT: OpenMP Tools API technical report
- OMPT Test Suite: regression tests for OMPT
- OMPD: OpenMP Debugging API technical report
- LLVM-openmp: LLVM runtime with experimental changes for OMPT

http://github.com/OpenMPToolsInterface

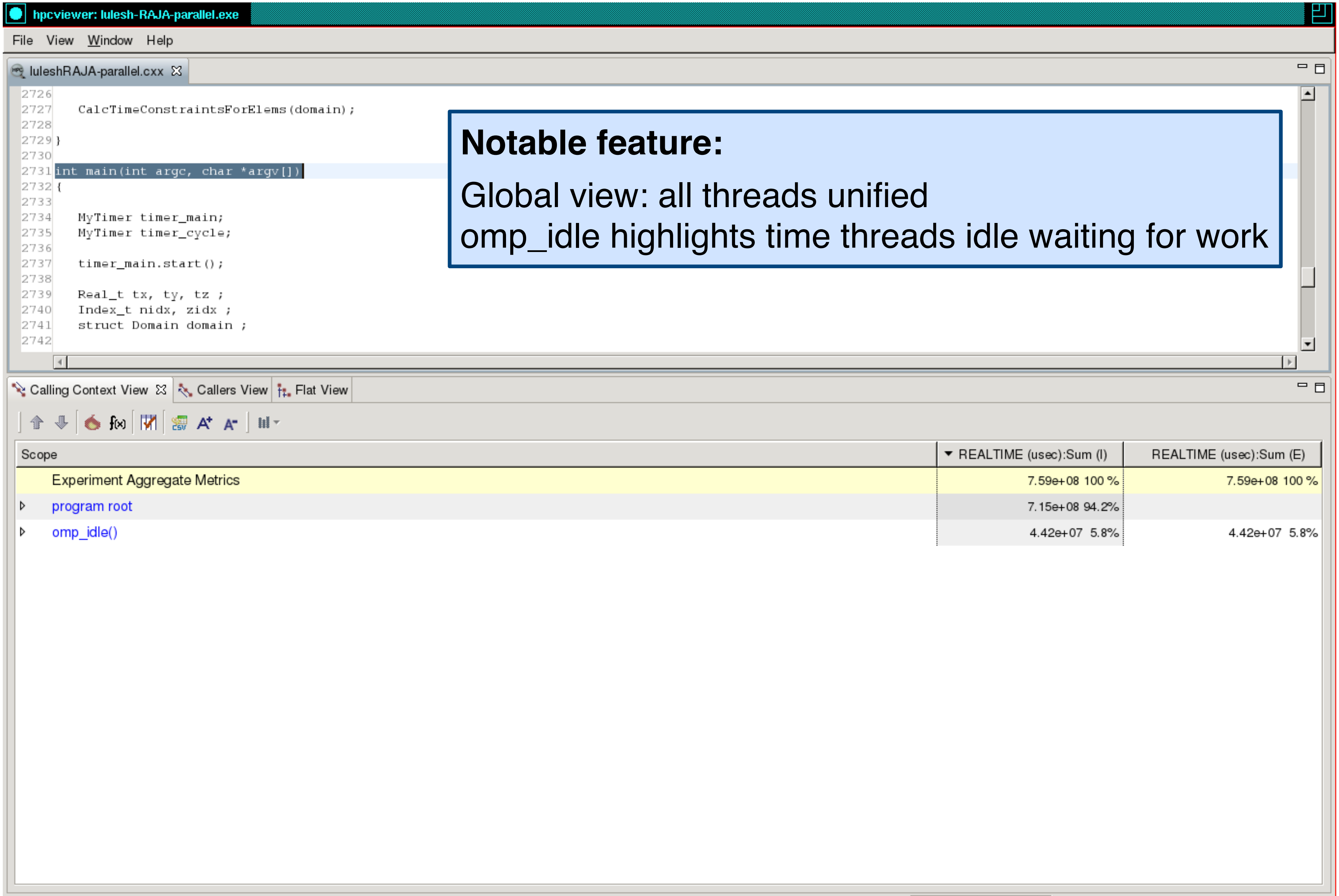
Case Study: LLNL's LULESH with RAJA

Livermore **U**nstructured **L**agrangian **E**xplicit **S**hock **H**ydrodynamics

- Compiled with high optimization
  - icpc -g -O3 -mavx -align -inline-max-total-size=20000 -inline-forceinline -ansi-alias -std=c++0x -openmp -debug inline-debug-info -parallel-source-info=2 -debug all -c -o luleshRAJA-parallel.o luleshRAJA-parallel.cxx -I. -I../../includes/ -DRAJA_PLATFORM_X86_AVX -DRAJA_COMPILER_ICC -DRAJA_USE_DOUBLE -DRAJA_USE_RESTRICT_PTR
  - icpc -g -O3 -mavx -align -inline-max-total-size=20000 -inline-forceinline -ansi-alias -std=c++0x -openmp -debug inline-debug-info -parallel-source-info=2 -debug all … -Wl,-rpath=/home/johnmc/pkgs/LLVM-openmp/lib /home/johnmc/pkgs/LLVM-openmp/lib/libiomp5.so -o lulesh-RAJA-parallel.exe
- Data collection:
  - hpcrun -e REALTIME@1000 -t ./lulesh-RAJA-parallel.exe
    - implicitly uses the OMPT performance tools interface, which is enabled in our OMPT-enhanced version of the Intel LLVM OpenMP runtime

2 18-core Haswell
72+1 threads

Case Study: LLNL's LULESH with RAJA

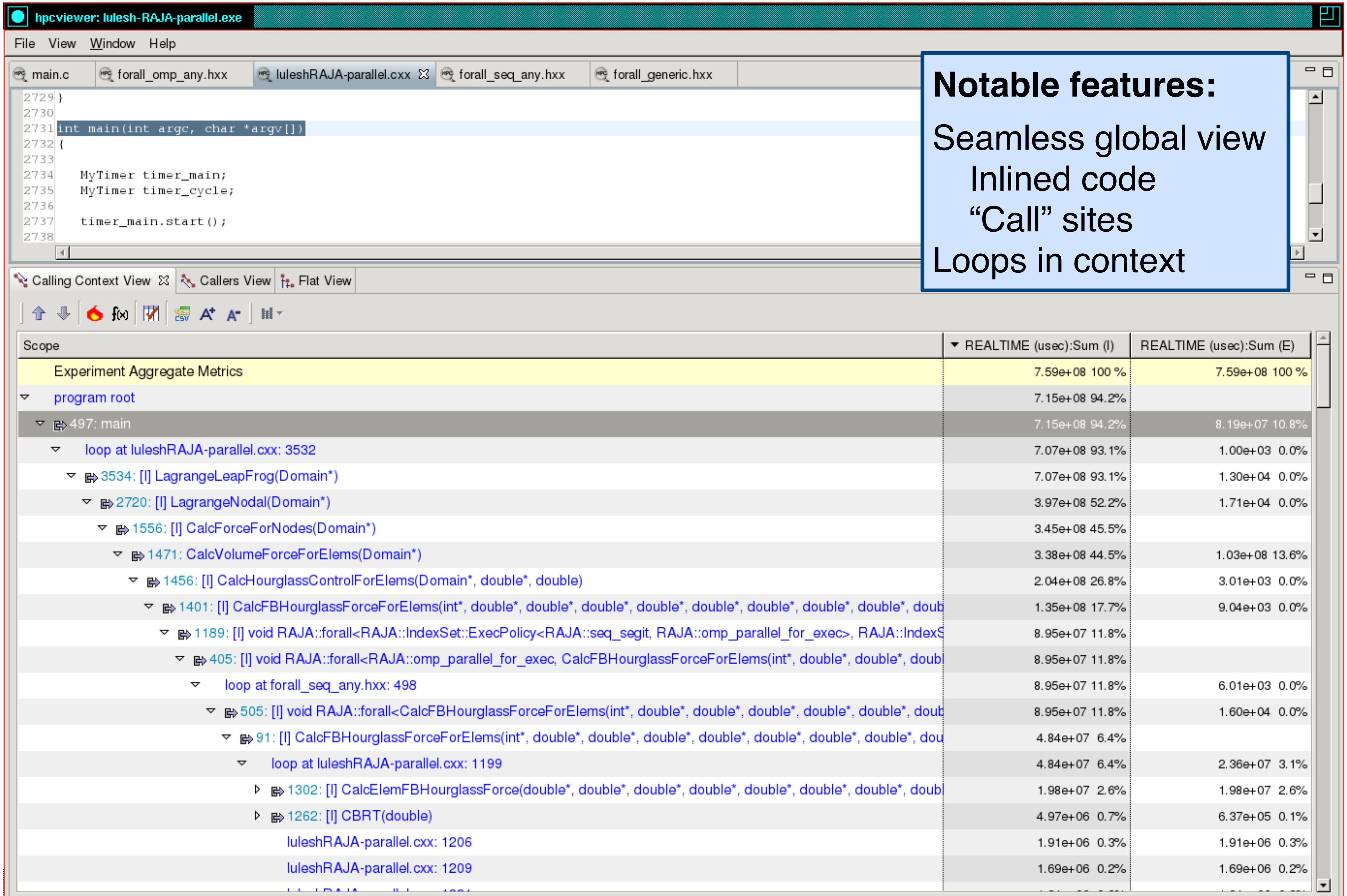


Notable feature:

Global view: all threads unified
omp_idle highlights time threads idle waiting for work

2 18-core Haswell
72+1 threads

Case Study: LLNL’s LULESH with RAJA



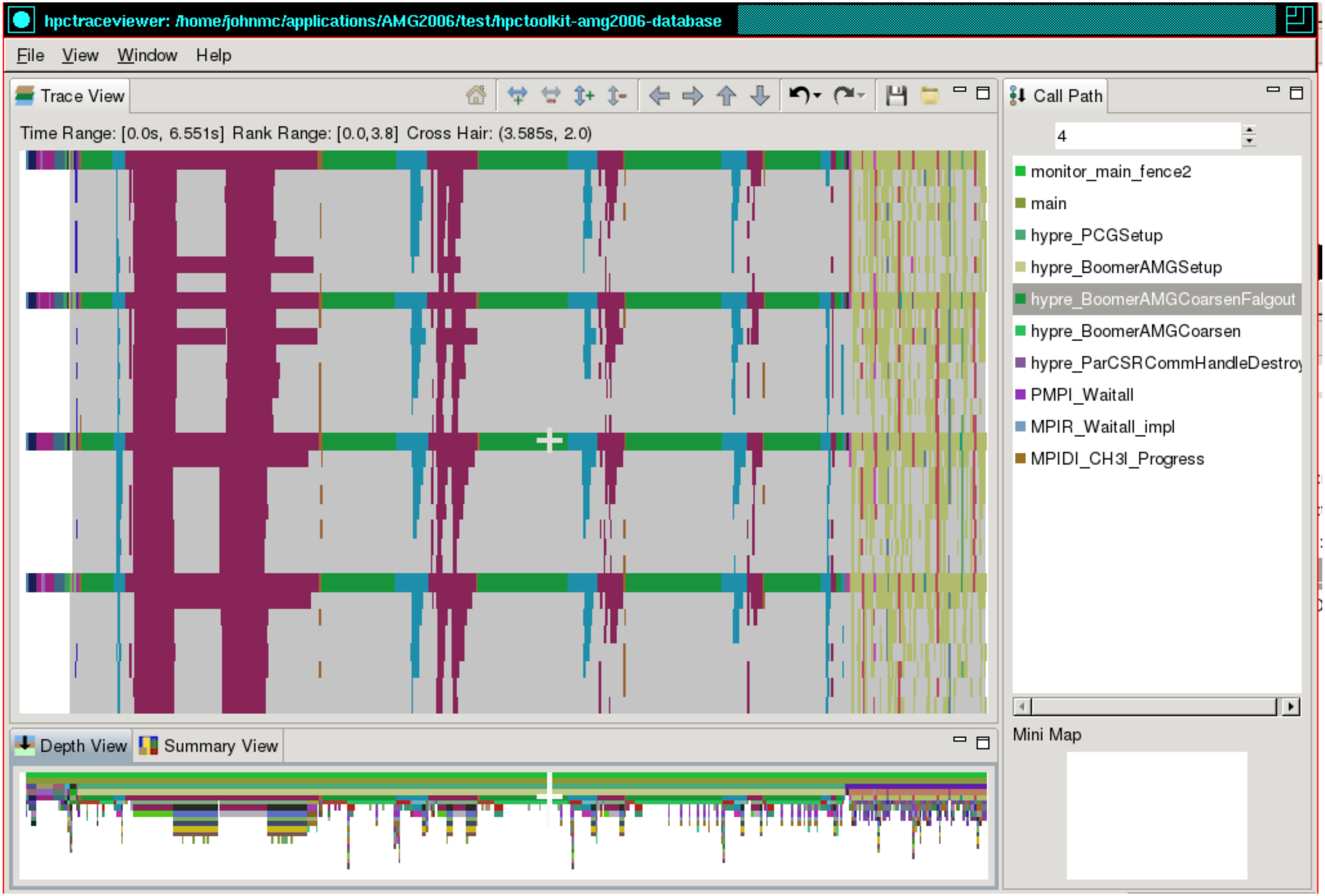
Notable features:

Seamless global view
- Inlined code
- “Call” sites

Loops in context

2 18-core Haswell
4 MPI ranks
6+3 threads per rank

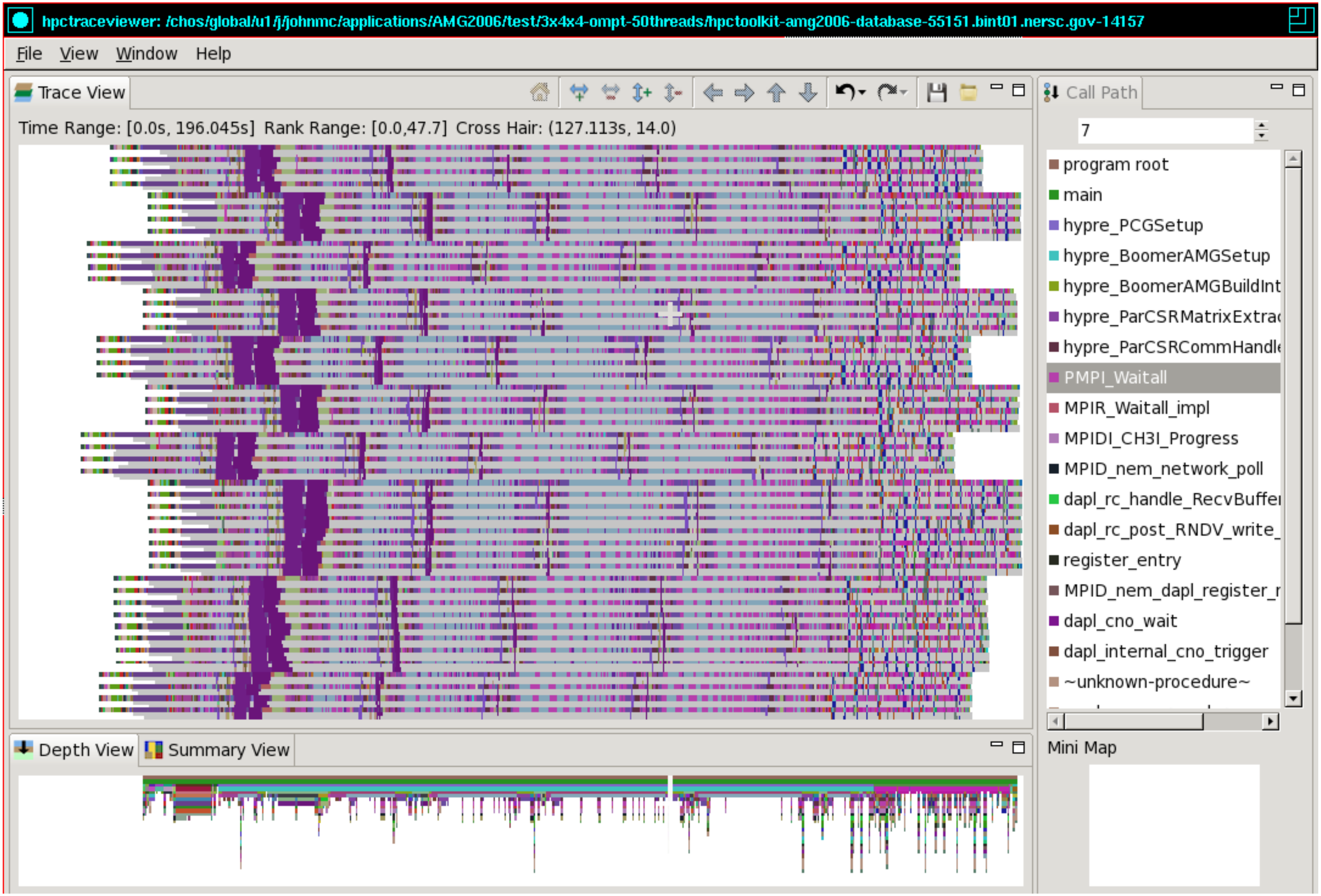
Case Study: AMG2006



12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

Case Study: AMG2006

Slice
Thread 0 from each MPI rank
First two OpenMP workers



Finishing OMPT

- Add support for task dependence tracking
  - callback event to inform tool of task dependences

- Add support for monitoring TARGET devices
  - callback events on the host
  - tracing on a device

TARGET Events on Host

- Mandatory Events
  - ompt_event_target_task_begin
  - ompt_event_target_task_end

- Optional events
  - ompt_event_target_data_begin
  - ompt_event_target_data_end

  - ompt_event_target_update_begin
  - ompt_event_target_update_end

TARGET Device Inquiry

```
OMPT_API int ompt_get_num_devices(void);

 OMPT_API int ompt_get_device_info(
       int device_id,
       const char **type,
       ompt_function_lookup_t *lookup
);
```

TARGET Device Inquiry

```
OMPT_API int ompt_get_num_devices(void);

 OMPT_API int ompt_get_device_info(
      int device_id,
      const char **type,
      ompt_function_lookup_t *lookup
);


OMPT_API int ompt_get_target_device_id(void);


OMPT_API ompt_target_device_time_t
   ompt_get_target_device_time(int device_id);
```

TARGET Device Tracing

```
OMPT_API int ompt_record_set(
  int device_id,
  ompt_bool enable,
  ompt_record_type_t rtype
);

OMPT_API int ompt_record_native_set(
  int device_id,
  ompt_bool enable,
  void *info,
  void **status
);

 typedef void (*ompt_buffer_request_callback_t) (
      int device_id,
      ompt_buffer_t **buffer,
      size_t *bytes
);

typedef void (*ompt_buffer_complete_callback_t) (
      int device_id,
      ompt_buffer_t *buffer,
      size_t bytes,
      ompt_buffer_cursor_t begin,
      ompt_buffer_cursor_t end
    );
```

```
 OMPT_API int ompt_recording_start (
      int device_id,
      ompt_buffer_request_callback_t request,
      ompt_buffer_complete_callback_t complete,
);

OMPT_API int ompt_recording_stop(
      int device_id
);
```

Processing Traces From TARGET Devices

OMPT Record Processing

```
OMPT_API int ompt_buffer_cursor_advance(
      ompt_buffer_t *buffer,
      ompt_buffer_cursor_t current,
      ompt_buffer_cursor_t *next
);

OMPT_API ompt_record_type_t
   ompt_record_get_type(
      ompt_buffer_t *buffer,
      ompt_buffer_cursor_t current
);

OMPT_API ompt_record_t *ompt_record_get(
      ompt_buffer_t *buffer,
      ompt_cursor_t current
);
```

Native Record Processing

```
OMPT_API void *ompt_record_native_get(
      ompt_buffer_t *buffer,
      ompt_cursor_t current
);

OMPT_API ompt_record_native_kind_t
  ompt_record_native_get_kind(
      void *native_record
);

OMPT_API const char*
  ompt_record_native_get_type(
      void *native_record
);

OMPT_API uint64_t ompt_record_native_get_time(
      void *native_record
);

OMPT_API int ompt_record_native_get_hwid(
      void *native_record
);
```

Next Steps

- Review proposed TARGET support
  - interact with OMPT TARGET monitoring, e.g., Xeon Phi
  - interacting with native TARGET monitoring, e.g., NVIDIA CUPTI
- Design libomptarget API to dovetail with OMPT
  - understand device HW/SW configuration
  - turn on monitoring
  - interpret performance data
- Prepare to wage a battle to have OMPT design incorporated as part of OpenMP standard