

# 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
    - idle after arriving at a barrier ending a parallel region
    - waiting at a barrier 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



# Case Study: LLNL's LULESH with RAJA

The screenshot displays the hpcviewer interface for the application 'lulesh-RAJA-parallel.exe'. The top pane shows the source code for 'luleshRAJA-parallel.cxx', with the `int main` function highlighted. A blue callout box points to the `omp_idle` function, noting its global view and highlighting of idle threads. The bottom pane shows the 'Calling Context View' with a table of performance metrics.

**Notable feature:**  
Global view: all threads unified  
`omp_idle` highlights time threads idle waiting for work

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)
Experiment Aggregate Metrics	7.59e+08 100 %	7.59e+08 100 %
▶ program root	7.15e+08 94.2%	
▶ omp_idle()	4.42e+07 5.8%	4.42e+07 5.8%

# Case Study: LLNL's LULESH with RAJA

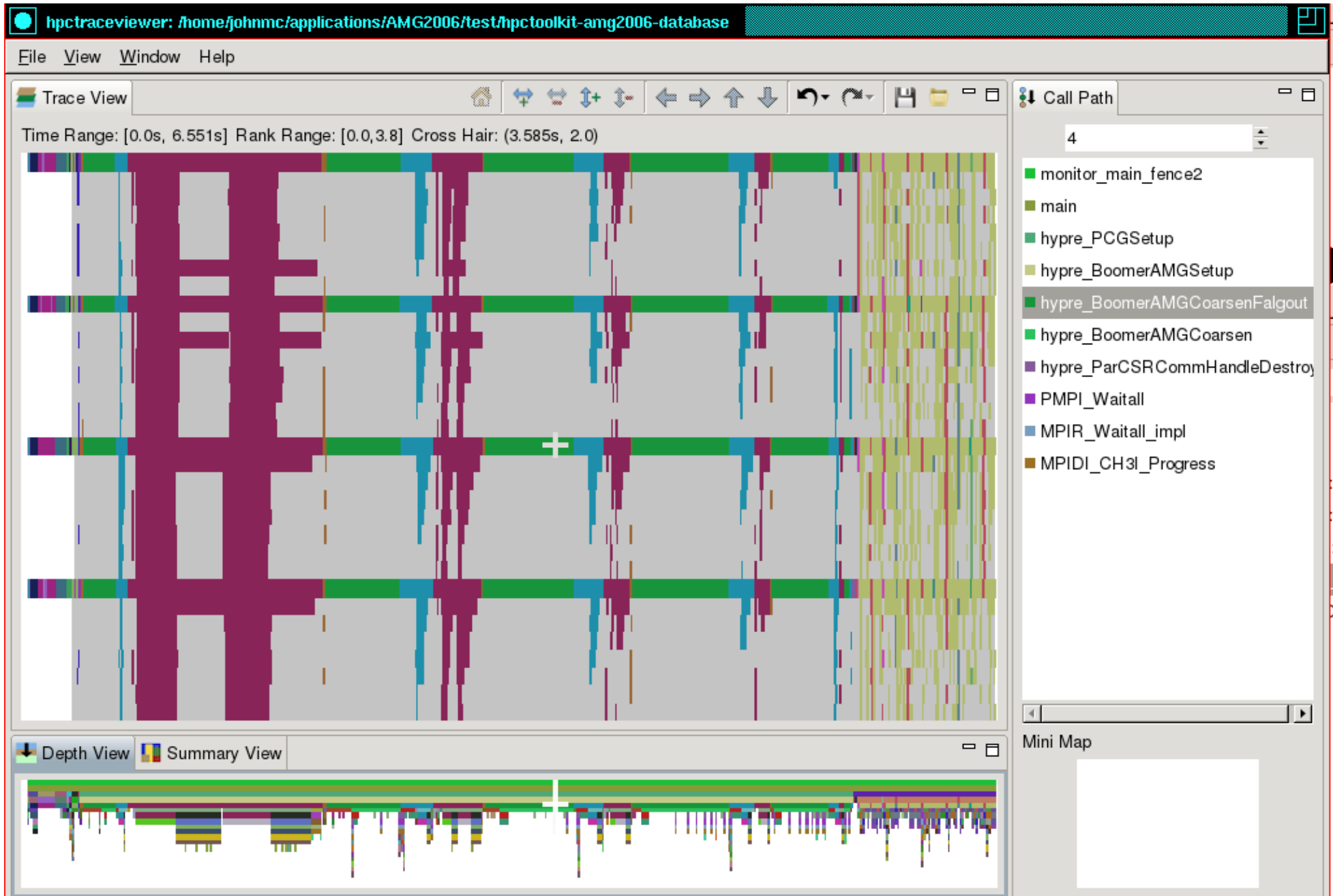
The screenshot shows the hpcviewer interface for the application 'lulesh-RAJA-parallel.exe'. The top pane displays source code from 'luleshRAJA-parallel.cxx', including the main function and timer declarations. The bottom pane shows a 'Calling Context View' with a tree structure of execution scopes and a corresponding performance table.

**Notable features:**  
 Seamless global view  
 Inlined code  
 "Call" sites  
 Loops in context

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)
Experiment Aggregate Metrics	7.59e+08 100 %	7.59e+08 100 %
program root	7.15e+08 94.2%	
497: main	7.15e+08 94.2%	8.19e+07 10.8%
loop at luleshRAJA-parallel.cxx: 3532	7.07e+08 93.1%	1.00e+03 0.0%
3534: [] LagrangeLeapFrog(Domain*)	7.07e+08 93.1%	1.30e+04 0.0%
2720: [] LagrangeNodal(Domain*)	3.97e+08 52.2%	1.71e+04 0.0%
1556: [] CalcForceForNodes(Domain*)	3.45e+08 45.5%	
1471: CalcVolumeForceForElems(Domain*)	3.38e+08 44.5%	1.03e+08 13.6%
1456: [] CalcHourglassControlForElems(Domain*, double*, double)	2.04e+08 26.8%	3.01e+03 0.0%
1401: [] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)	1.35e+08 17.7%	9.04e+03 0.0%
1189: [] void RAJA::forall<RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>, RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>>(RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>&, RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>&)	8.95e+07 11.8%	
405: [] void RAJA::forall<RAJA::omp_parallel_for_exec, CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)>(RAJA::omp_parallel_for_exec&, CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)&)	8.95e+07 11.8%	
loop at forall_seq_any.hxx: 498	8.95e+07 11.8%	6.01e+03 0.0%
505: [] void RAJA::forall<CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)>(CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)&)	8.95e+07 11.8%	1.60e+04 0.0%
91: [] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double)	4.84e+07 6.4%	
loop at luleshRAJA-parallel.cxx: 1199	4.84e+07 6.4%	2.36e+07 3.1%
1302: [] CalcElemFBHourglassForce(double*, double*, double*, double*, double*, double*, double*, double*, double)	1.98e+07 2.6%	1.98e+07 2.6%
1262: [] CBRT(double)	4.97e+06 0.7%	6.37e+05 0.1%
luleshRAJA-parallel.cxx: 1206	1.91e+06 0.3%	1.91e+06 0.3%
luleshRAJA-parallel.cxx: 1209	1.69e+06 0.2%	1.69e+06 0.2%

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

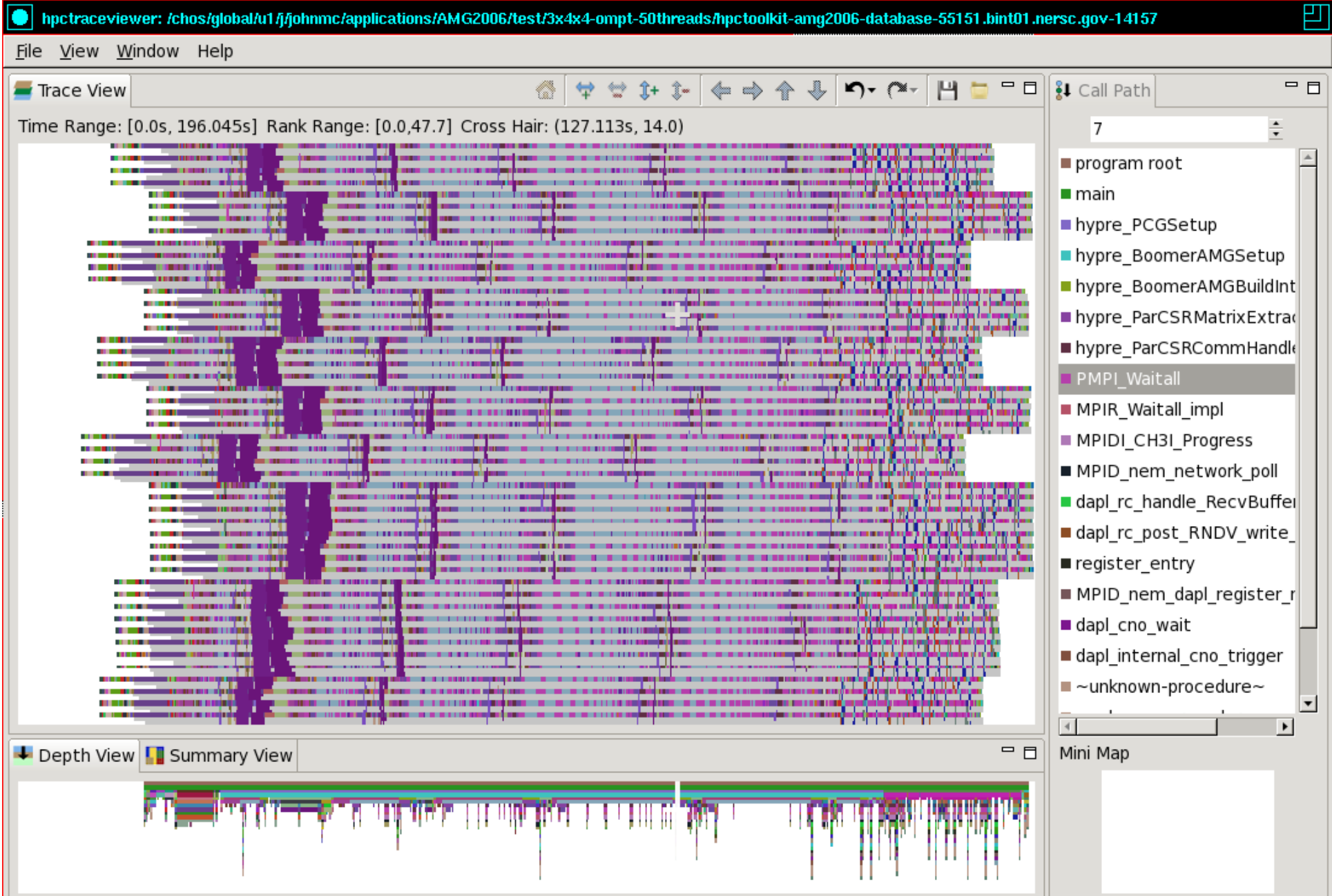
# Case Study: AMG2006



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