# OpenMP Tools API for Profiling

Xu Liu, John Mellor-Crummey and Mike Fagan
Department of Computer Science
Rice University

**`http://hpctoolkit.org`**

# Requirements for performance tools

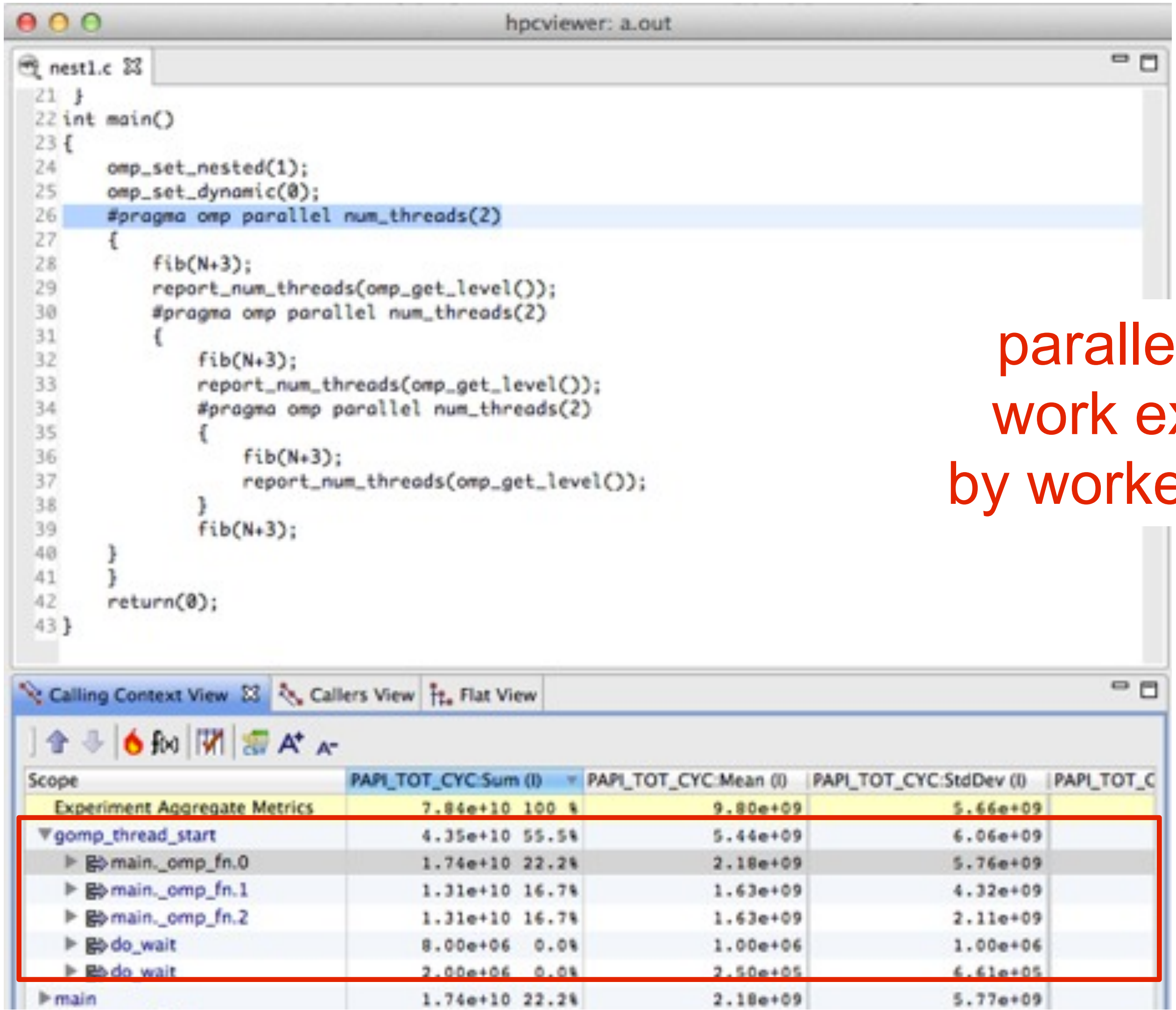
- Accurate measurement
  - low overhead: support sampling based data collection
  - attribute metrics to user-level global view
  - attribute performance losses to causes rather than symptoms
- Effective metrics: measure and attribute ...
  - idleness, work, and overhead
  - lock and critical section costs
    - quantify lock contention as well
- Complete support
  - work-sharing parallel regions
  - nested parallel regions
  - tasks
- Intuitive and insightful analysis
  - code centric: overhead & parallel efficiency of OpenMP constructs
  - time centric: how execution unfolds over time

# Approach

- Key features
  - support unified, user-level view of calling contexts across all threads
  - shift blame from symptoms to causes of performance losses
  - pinpoint lock and critical section contention
  - support both profiling and tracing
- Methods
  - lightweight instrumentation of OpenMP runtime system
  - efficient sampling-based measurement
  - post-mortem analysis

# Problem: separate views for different threads

Worker threads don’t know the full user-level context for work



```
}
int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        fib(N+3);
        report_num_threads(omp_get_level());
        #pragma omp parallel num_threads(2)
        {
            fib(N+3);
            report_num_threads(omp_get_level());
            #pragma omp parallel num_threads(2)
            {
                fib(N+3);
                report_num_threads(omp_get_level());
            }
            fib(N+3);
        }
    }
    return(0);
}
```

| Scope | PAPI_TOT_CYC:Sum (I) | PAPI_TOT_CYC:Mean (I) | PAPI_TOT_CYC:StdDev (I) | PAPI_TOT_C |
|---|---|---|---|---|
| Experiment Aggregate Metrics | 7.84e+10 100 % | 9.80e+09 | 5.66e+09 | |
| gomp_thread_start | 4.35e+10 55.5% | 5.44e+09 | 6.06e+09 | |
| main._omp_fn.0 | 1.74e+10 22.2% | 2.18e+09 | 5.76e+09 | |
| main._omp_fn.1 | 1.31e+10 16.7% | 1.63e+09 | 4.32e+09 | |
| main._omp_fn.2 | 1.31e+10 16.7% | 1.63e+09 | 2.11e+09 | |
| do_wait | 8.00e+06 0.0% | 1.00e+06 | 1.00e+06 | |
| do_wait | 2.00e+06 0.0% | 2.50e+05 | 6.61e+05 | |
| main | 1.74e+10 22.2% | 2.18e+09 | 5.77e+09 | |

parallel region
work executed
by worker threads

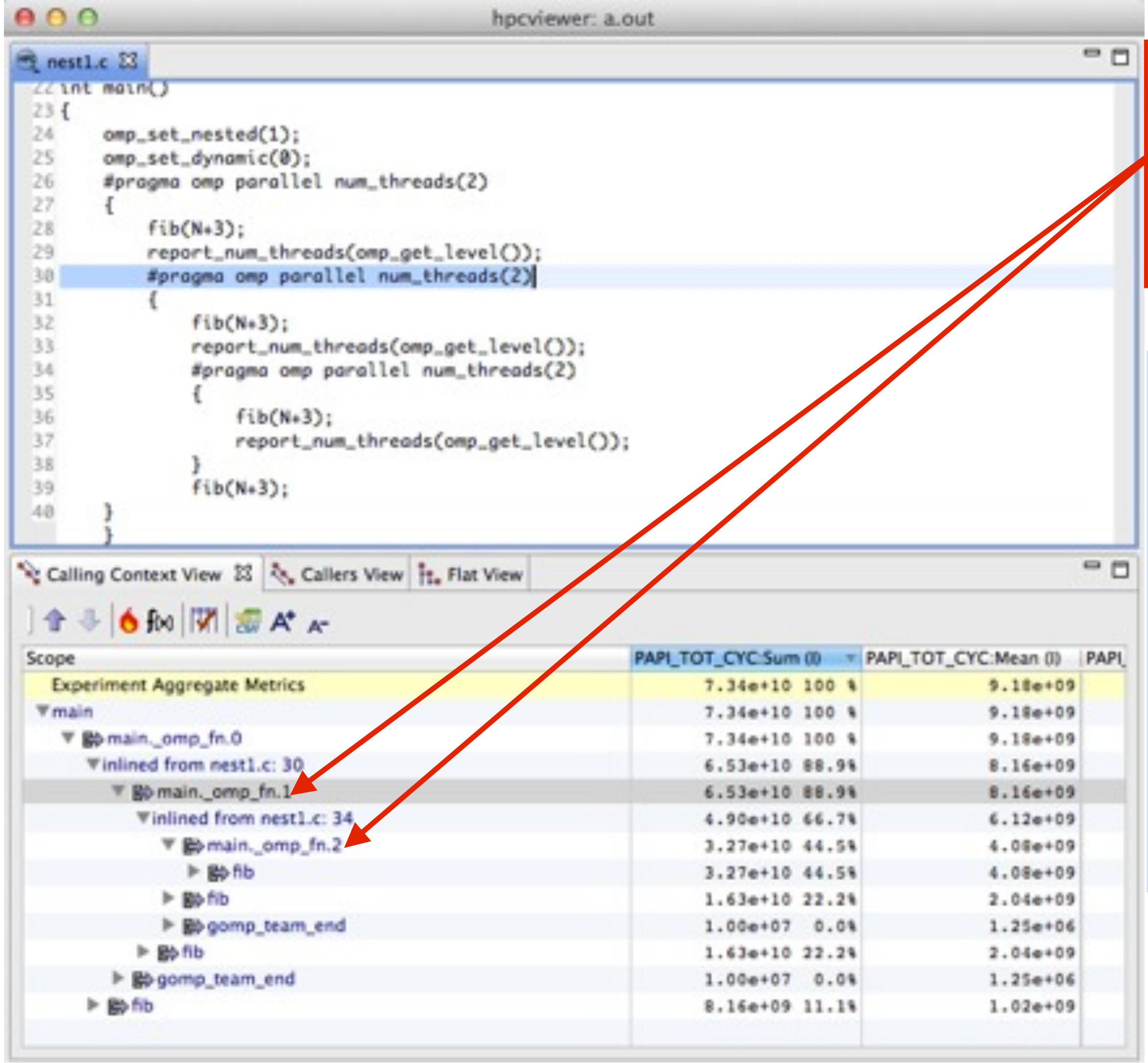
# Solution: efficient deferred context construction

- OpenMP runtime system
  - supply a tool callback interface
    - parallel region begin: void region_entry_callback(void)
    - parallel region end: void region_exit_callback(void)
  - assign a unique region ID for every instance of a parallel region
    - use atomic increment to generate an ID for each new region instance upon entry
  - implement query API
    - get parallel region ID: uint64_t omp_get_region_id()
    - whether the frame should be elided or replaced
      - elided: e.g., GOMP_thread_start, GOMP_team_start
      - replace with <parallel region> in the call stack
  - make callbacks upon parallel region entry/exit
    - if (region_entry_callback) (*region_entry_callback)()
    - if (region_exit_callback) (*region_exit_callback)()
    - <u>minimal cost if callback pointers not provided by a tool</u> (see slide 26)

# Solution: efficient deferred context construction

- Tool support
  - mechanisms
    - register callbacks with the OpenMP tools API
      - enter/exit a parallel region
    - maintain a global map: region ID → region info
      - key: region ID
      - value: region info
        - number of samples in the region
        - the calling context of the region
  - mechanisms in use
    - master thread callback at region entry
      - create a new entry in the map
    - worker threads at a sample event
      - unwind to a root in TBD set indexed by a region ID
      - update number of samples for the region ID in the map
      - if the calling context for region ID is available in the map, resolve it for the work
    - master thread callback at region exit
      - iff number of samples for the region ID > 0
        - unwind the stack to determine the calling context for the region
        - insert the full context of the region to the map

# Results of deferred context construction



```
22 int main()
23 {
24     omp_set_nested(1);
25     omp_set_dynamic(0);
26     #pragma omp parallel num_threads(2)
27     {
28         fib(N+3);
29         report_num_threads(omp_get_level());
30         #pragma omp parallel num_threads(2)
31         {
32             fib(N+3);
33             report_num_threads(omp_get_level());
34             #pragma omp parallel num_threads(2)
35             {
36                 fib(N+3);
37                 report_num_threads(omp_get_level());
38             }
39             fib(N+3);
40     }
41 }
```

| Scope | PAPI_TOT_CYC:Sum (I) | PAPI_TOT_CYC:Mean (I) | PAPI_ |
|---|---|---|---|
| Experiment Aggregate Metrics | 7.34e+10 100 % | 9.18e+09 | |
| ▼ main | 7.34e+10 100 % | 9.18e+09 | |
| ▼ main._omp_fn.0 | 7.34e+10 100 % | 9.18e+09 | |
| ▼ inlined from nest1.c: 30 | 6.53e+10 88.9% | 8.16e+09 | |
| ▼ main._omp_fn.1 | 6.53e+10 88.9% | 8.16e+09 | |
| ▼ inlined from nest1.c: 34 | 4.90e+10 66.7% | 6.12e+09 | |
| ▼ main._omp_fn.2 | 3.27e+10 44.5% | 4.08e+09 | |
| ▶ fib | 3.27e+10 44.5% | 4.08e+09 | |
| ▶ fib | 1.63e+10 22.2% | 2.04e+09 | |
| ▶ gomp_team_end | 1.00e+07 0.0% | 1.25e+06 | |
| ▶ fib | 1.63e+10 22.2% | 2.04e+09 | |
| ▶ gomp_team_end | 1.00e+07 0.0% | 1.25e+06 | |
| ▶ fib | 8.16e+09 11.1% | 1.02e+09 | |

main._omp_fn.*: outlined functions that correspond to <parallel region>

parallel regions are identified with full calling context through the deferred context creation mechanism that involves unwinding at region end if samples were taken in the region by any worker thread

# Nested regions

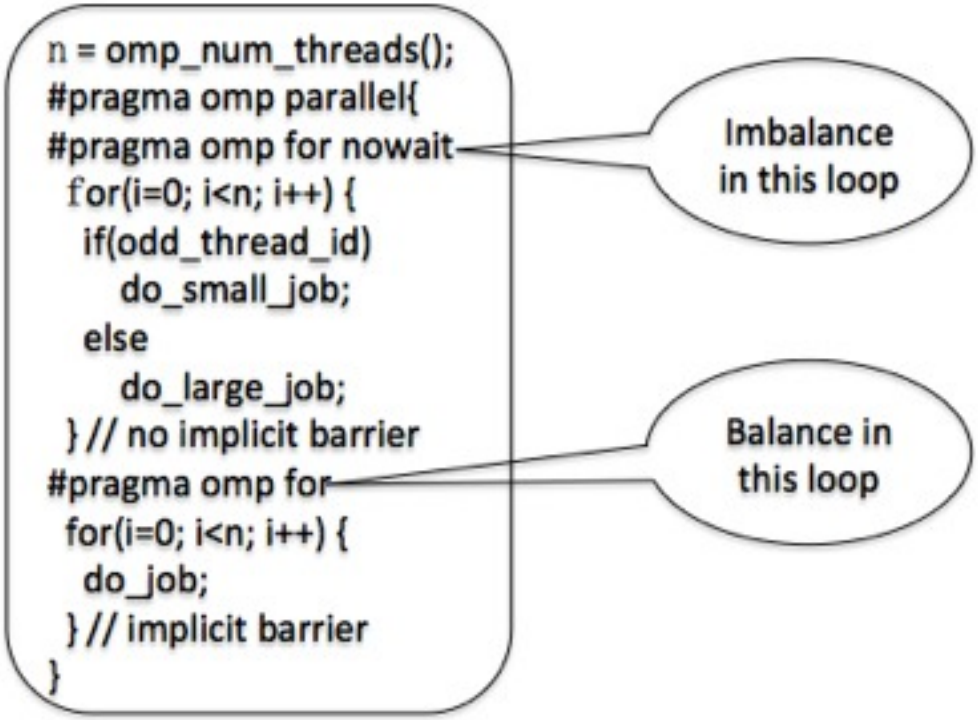
- OpenMP runtime system
  - add an additional query API for use by a tool
    - get parent parallel region ID: uint64_t omp_get_parent_region_id()
- Tool
  - uses the same map discussed before
  - thread actions
    - master thread
      - do the same operations described previously
    - worker threads
      - record the outer-most region ID
      - unwind itself to the root with outer-most region ID in the TBD set
    - sub-master threads
      - partially resolve the context of parallel regions
      - add the partially resolved context to its TBD set until resolved
  - at process termination, process writes out the performance data after all trees in TBD set are fully resolved

# Tasks

- Resolve task context to its execution point
  - openMP runtime system
    - no special support needed
  - tool
    - use deferred context construction for parallel regions
    - no special handling for tasks
- Resolve task context to its creation point (costly, but available if desired)
  - openMP runtime system
    - allocate an 8-byte slot in a task structure for tool use (to record its creation context)
    - add a callback when creating a new untied task
      - passes the address of the 8-byte slot to the tool
    - add a query API to identify when a procedure frame is the root of a untied task instance
  - tool
    - register the callback
    - unwind the call stack at task creation callback and return a pointer to a calling context
    - fills in the 8-byte slot in the task structure with a pointer to the task creation context
    - when executing an untied task is interrupted at a sample event
      - unwind the call stack to the task’s root frame
      - concatenate with the task creation context as the prefix

# Blame shifting: from symptoms to causes

- Goals
  - quantify insufficient parallelism
  - quantify excessive parallelism (too fine granularity)
  - attribute performance losses to causes rather than symptoms

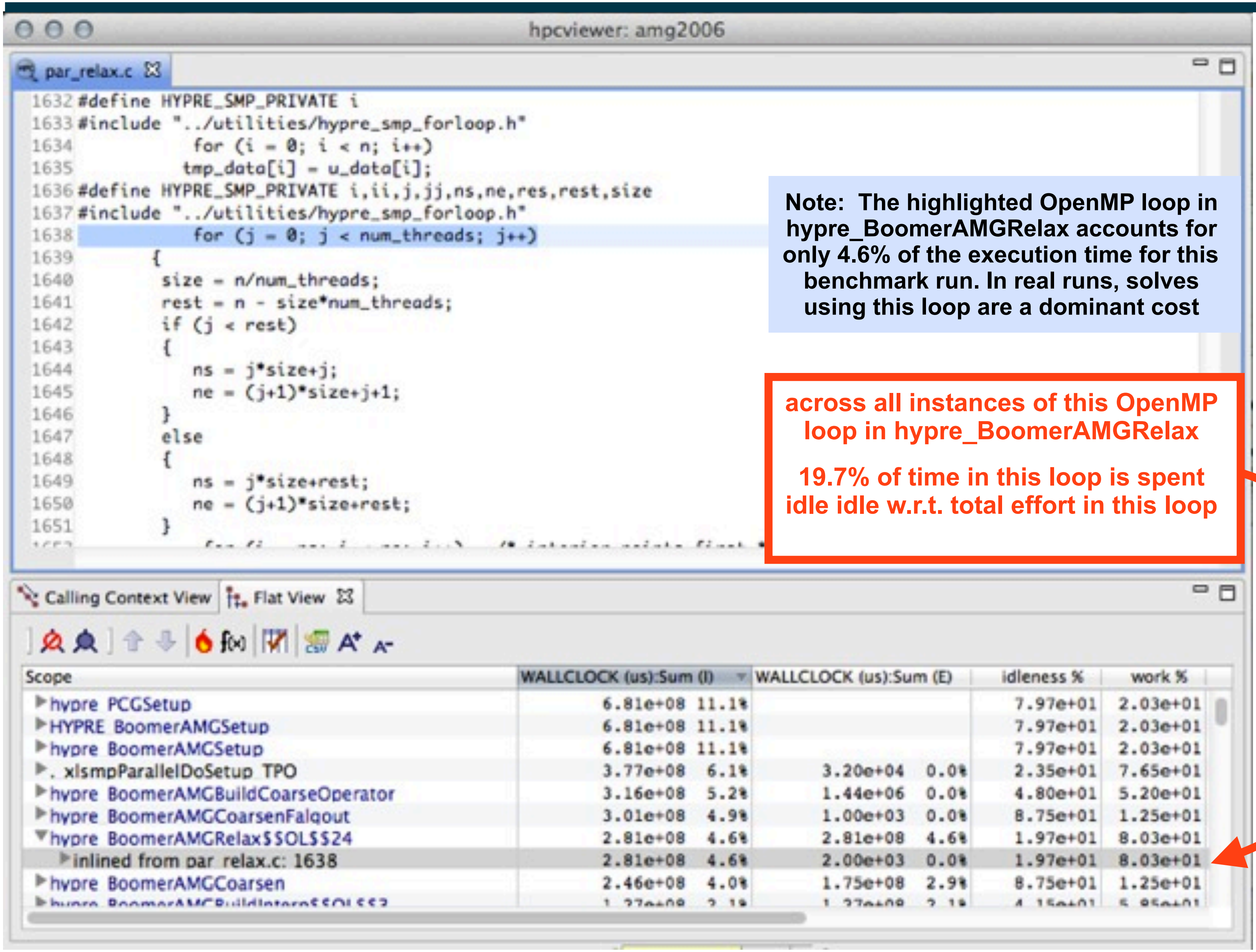


```
n = omp_num_threads();
#pragma omp parallel{
#pragma omp for nowait
  for(i=0; i<n; i++) {
   if(odd_thread_id)
      do_small_job;
   else
      do_large_job;
  } // no implicit barrier
#pragma omp for
  for(i=0; i<n; i++) {
   do_job;
  } // implicit barrier
}
```

# Blame shifting support

- Approach
  - create derived metrics
    - idleness: time threads are idle waiting for work
    - work: time threads execute user code
    - overhead: time threads execute code in the OpenMP runtime system
  - blame idleness and overhead to working threads
    - overhead blamed directly to an executing thread
    - shift blame for idleness to code that is being executed while other threads are idle
- Implementation
  - openMP runtime system
    - make callbacks when at thread state transitions
      - thread transitions idle ↔ working
      - thread creation/exit
        - these callbacks identify to the tool which threads belong to OpenMP
  - tool
    - maintains two global counters
      - number of threads that are created (or dedicated HW resources that are reserved)
      - number of threads that are working
    - idleness is the difference between the two counters
    - at a sample event
      - if the thread is actively working
        - attribute a sample of work to the present context
        - attribute a fractional sample of idleness to the present context of the active worker
          fractional sample = # idle threads / # active workers
      - else, ignore the sample event

# Code-centric view: `hypre_BoomerAMGRelax`



```
#define HYPRE_SMP_PRIVATE i
#include "../utilities/hypre_smp_forloop.h"
                for (i = 0; i < n; i++)
              tmp_data[i] = u_data[i];
#define HYPRE_SMP_PRIVATE i,ii,j,jj,ns,ne,res,rest,size
#include "../utilities/hypre_smp_forloop.h"
                for (j = 0; j < num_threads; j++)
           {
             size = n/num_threads;
             rest = n - size*num_threads;
             if (j < rest)
             {
                ns = j*size+j;
                ne = (j+1)*size+j+1;
             }
             else
             {
                ns = j*size+rest;
                ne = (j+1)*size+rest;
             }
```

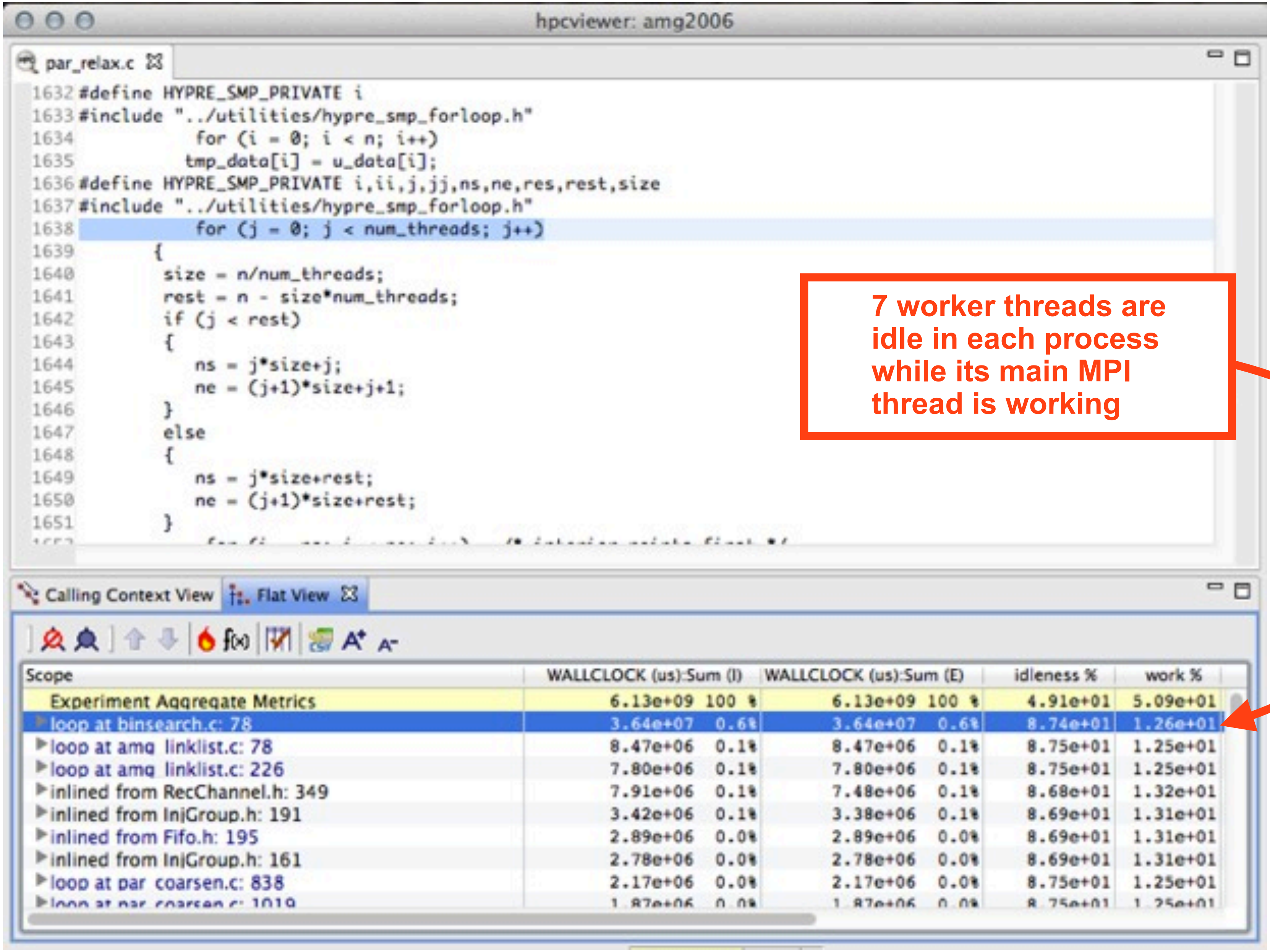
| Scope | WALLCLOCK (us):Sum (I) | | WALLCLOCK (us):Sum (E) | | idleness % | work % |
|---|---|---|---|---|---|---|
| hypre_PCGSetup | 6.81e+08 | 11.1% | | | 7.97e+01 | 2.03e+01 |
| HYPRE_BoomerAMGSetup | 6.81e+08 | 11.1% | | | 7.97e+01 | 2.03e+01 |
| hypre_BoomerAMGSetup | 6.81e+08 | 11.1% | | | 7.97e+01 | 2.03e+01 |
| ._xlsmpParallelDoSetup_TPO | 3.77e+08 | 6.1% | 3.20e+04 | 0.0% | 2.35e+01 | 7.65e+01 |
| hypre_BoomerAMGBuildCoarseOperator | 3.16e+08 | 5.2% | 1.44e+06 | 0.0% | 4.80e+01 | 5.20e+01 |
| hypre_BoomerAMGCoarsenFalgout | 3.01e+08 | 4.9% | 1.00e+03 | 0.0% | 8.75e+01 | 1.25e+01 |
| hypre_BoomerAMGRelax$$OL$$24 | 2.81e+08 | 4.6% | 2.81e+08 | 4.6% | 1.97e+01 | 8.03e+01 |
| inlined from par_relax.c: 1638 | 2.81e+08 | 4.6% | 2.00e+03 | 0.0% | 1.97e+01 | 8.03e+01 |
| hypre_BoomerAMGCoarsen | 2.46e+08 | 4.0% | 1.75e+08 | 2.9% | 8.75e+01 | 1.25e+01 |
| hypre_BoomerAMGBuildInterp$$OL$$3 | 1.27e+08 | 2.1% | 1.27e+08 | 2.1% | 4.15e+01 | 5.85e+01 |

Note: The highlighted OpenMP loop in hypre_BoomerAMGRelax accounts for only 4.6% of the execution time for this benchmark run. In real runs, solves using this loop are a dominant cost

across all instances of this OpenMP loop in hypre_BoomerAMGRelax

19.7% of time in this loop is spent idle idle w.r.t. total effort in this loop

# Serial Code in AMG2006 8 PE, 8 Threads



```
#define HYPRE_SMP_PRIVATE i
#include "../utilities/hypre_smp_forloop.h"
            for (i = 0; i < n; i++)
           tmp_data[i] = u_data[i];
#define HYPRE_SMP_PRIVATE i,ii,j,jj,ns,ne,res,rest,size
#include "../utilities/hypre_smp_forloop.h"
            for (j = 0; j < num_threads; j++)
        {
         size = n/num_threads;
         rest = n - size*num_threads;
         if (j < rest)
         {
            ns = j*size+j;
            ne = (j+1)*size+j+1;
         }
         else
         {
            ns = j*size+rest;
            ne = (j+1)*size+rest;
         }
```

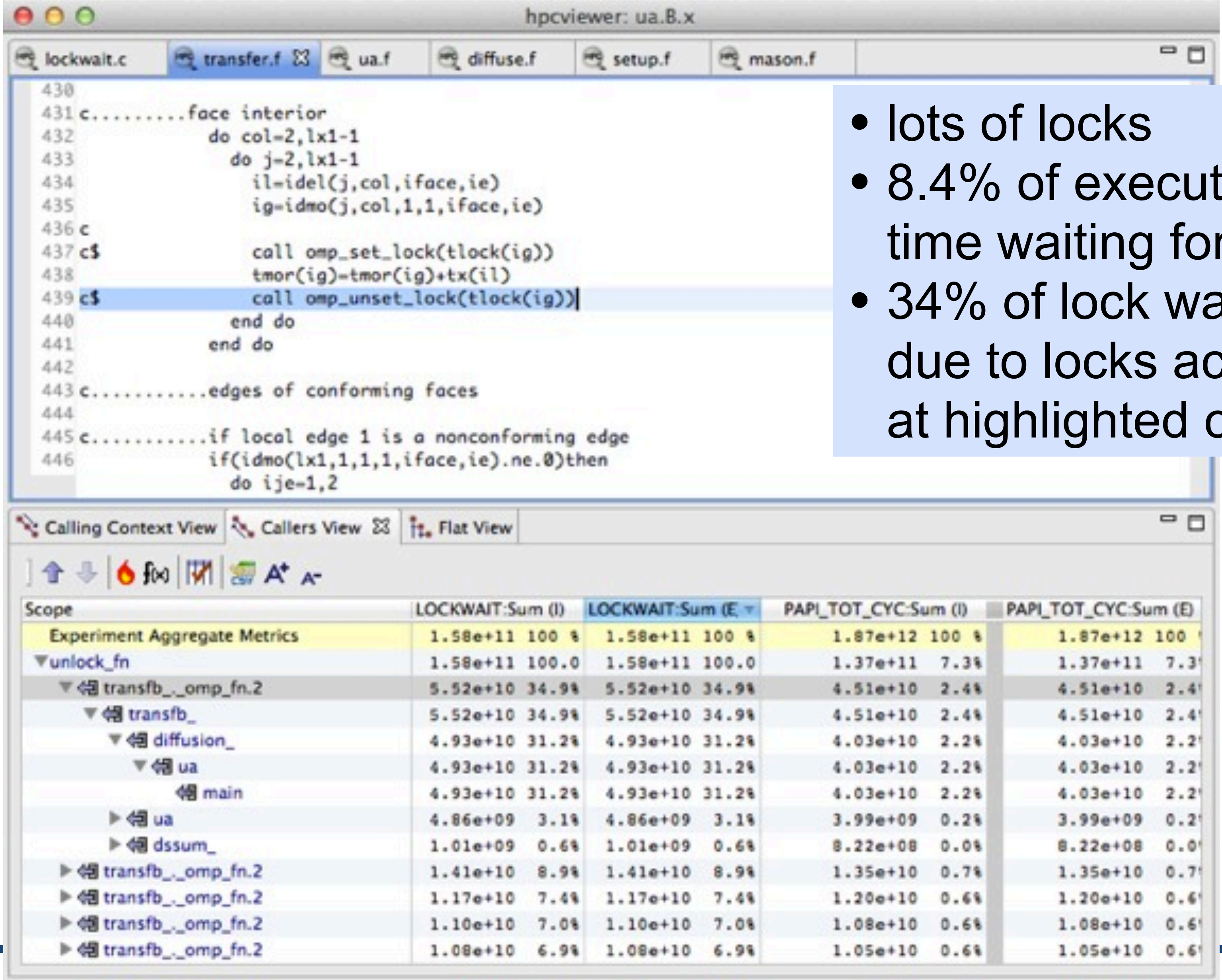
| Scope | WALLCLOCK (us):Sum (I) | | WALLCLOCK (us):Sum (E) | | idleness % | work % |
|---|---|---|---|---|---|---|
| Experiment Aggregate Metrics | 6.13e+09 | 100 % | 6.13e+09 | 100 % | 4.91e+01 | 5.09e+01 |
| loop at binsearch.c: 78 | 3.64e+07 | 0.6% | 3.64e+07 | 0.6% | 8.74e+01 | 1.26e+01 |
| loop at amg_linklist.c: 78 | 8.47e+06 | 0.1% | 8.47e+06 | 0.1% | 8.75e+01 | 1.25e+01 |
| loop at amg_linklist.c: 226 | 7.80e+06 | 0.1% | 7.80e+06 | 0.1% | 8.75e+01 | 1.25e+01 |
| inlined from RecChannel.h: 349 | 7.91e+06 | 0.1% | 7.48e+06 | 0.1% | 8.68e+01 | 1.32e+01 |
| inlined from InjGroup.h: 191 | 3.42e+06 | 0.1% | 3.38e+06 | 0.1% | 8.69e+01 | 1.31e+01 |
| inlined from Fifo.h: 195 | 2.89e+06 | 0.0% | 2.89e+06 | 0.0% | 8.69e+01 | 1.31e+01 |
| inlined from InjGroup.h: 161 | 2.78e+06 | 0.0% | 2.78e+06 | 0.0% | 8.69e+01 | 1.31e+01 |
| loop at par_coarsen.c: 838 | 2.17e+06 | 0.0% | 2.17e+06 | 0.0% | 8.75e+01 | 1.25e+01 |
| loop at par_coarsen.c: 1019 | 1.87e+06 | 0.0% | 1.87e+06 | 0.0% | 8.75e+01 | 1.25e+01 |

**7 worker threads are idle in each process while its main MPI thread is working**

# Locks and Critical Sections (CS)

- Issues
  - code with many locks or CS; high acquisition rates; substantial time waiting for access
  - code that is waiting may be different from the code holding a lock or critical section
- Solution
  - quantitatively shift blame to lock holder for the lock waiting time of other threads
- Implementation
  - openMP runtime system
    - add an interface for switching to a lock implementation supplied by a tool when a thread fails to acquire a lock
      - if (lock_wait_callback) (*lock_wait_callback)(&lock)
        - address of lock needed by tool to blame waiting on the particular lock
      - if (unlock_callback) (*unlock_callback)(&lock)
        else normal_openmp_unlock()
  - tool
    - register customized spin lock routine
      - 32 bit representation consistent with pthreads
        - 1 lowest bit for the lock
        - 30 bits for samples and 1 highest bit for overflow mark
    - record the lock ID which the thread is spin waiting for
      - charge the sample to the lock: atomic add to the lock
    - charge samples attributed to the lock while it was held to the lock holder at the lock release point
      - use an atomic swap

# Example: blame shifting for locks

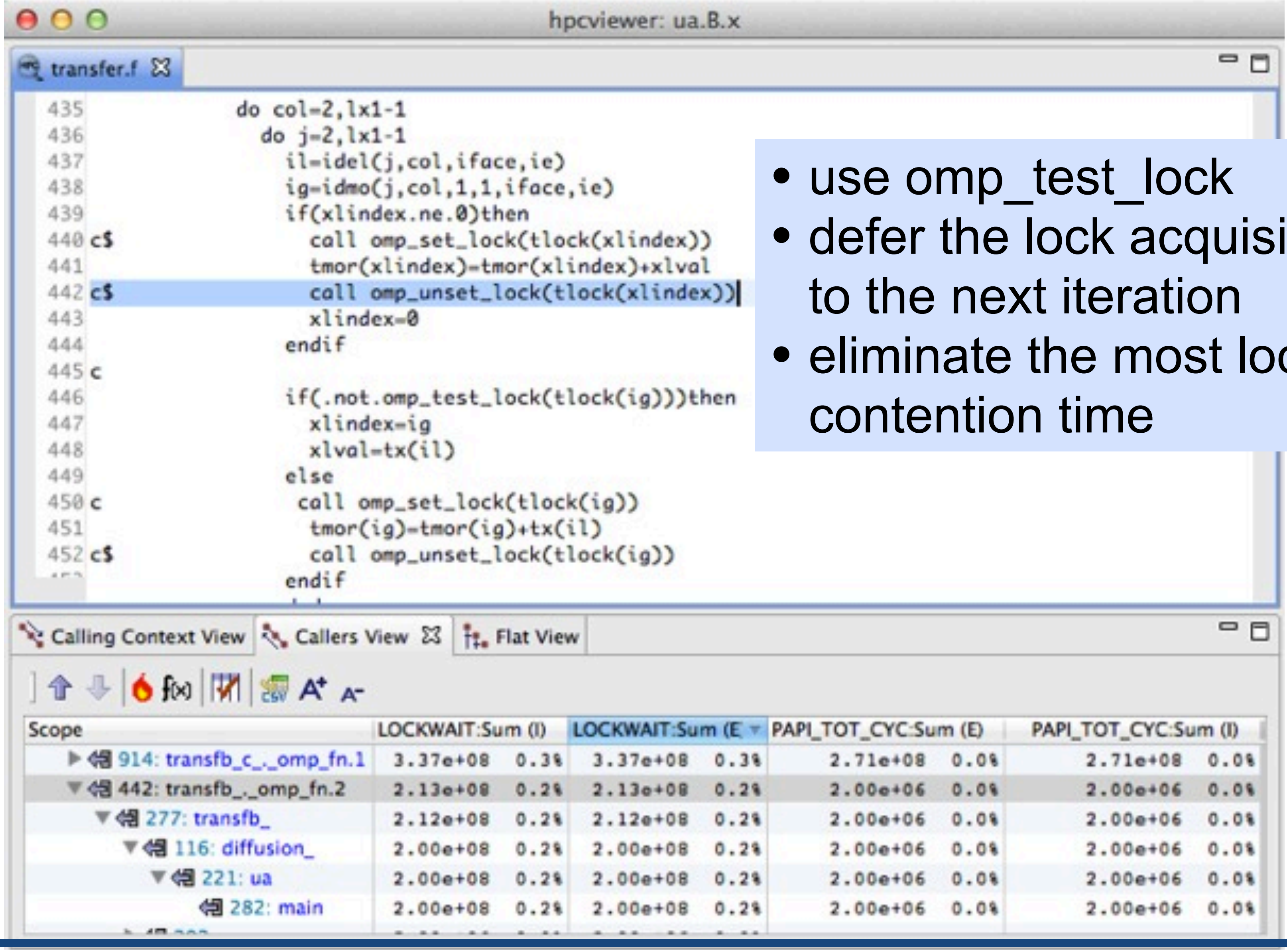


```

c.........face interior
            do col=2,lx1-1
              do j=2,lx1-1
                il=idel(j,col,iface,ie)
                ig=idmo(j,col,1,1,iface,ie)
c
c$              call omp_set_lock(tlock(ig))
                tmor(ig)=tmor(ig)+tx(il)
c$              call omp_unset_lock(tlock(ig))
              end do
            end do

c...........edges of conforming faces

c...........if local edge 1 is a nonconforming edge
            if(idmo(lx1,1,1,1,iface,ie).ne.0)then
              do ije=1,2
```

| Scope | LOCKWAIT:Sum (I) | LOCKWAIT:Sum (E | PAPI_TOT_CYC:Sum (I) | PAPI_TOT_CYC:Sum (E) |
|---|---|---|---|---|
| Experiment Aggregate Metrics | 1.58e+11 100 % | 1.58e+11 100 % | 1.87e+12 100 % | 1.87e+12 100 |
| ▼unlock_fn | 1.58e+11 100.0 | 1.58e+11 100.0 | 1.37e+11 7.3% | 1.37e+11 7.3 |
| ▼ transfb_._omp_fn.2 | 5.52e+10 34.9% | 5.52e+10 34.9% | 4.51e+10 2.4% | 4.51e+10 2.4 |
| ▼ transfb_ | 5.52e+10 34.9% | 5.52e+10 34.9% | 4.51e+10 2.4% | 4.51e+10 2.4 |
| ▼ diffusion_ | 4.93e+10 31.2% | 4.93e+10 31.2% | 4.03e+10 2.2% | 4.03e+10 2.2 |
| ▼ ua | 4.93e+10 31.2% | 4.93e+10 31.2% | 4.03e+10 2.2% | 4.03e+10 2.2 |
| main | 4.93e+10 31.2% | 4.93e+10 31.2% | 4.03e+10 2.2% | 4.03e+10 2.2 |
| ▶ ua | 4.86e+09 3.1% | 4.86e+09 3.1% | 3.99e+09 0.2% | 3.99e+09 0.2 |
| ▶ dssum_ | 1.01e+09 0.6% | 1.01e+09 0.6% | 8.22e+08 0.0% | 8.22e+08 0.0 |
| ▶ transfb_._omp_fn.2 | 1.41e+10 8.9% | 1.41e+10 8.9% | 1.35e+10 0.7% | 1.35e+10 0.7 |
| ▶ transfb_._omp_fn.2 | 1.17e+10 7.4% | 1.17e+10 7.4% | 1.20e+10 0.6% | 1.20e+10 0.6 |
| ▶ transfb_._omp_fn.2 | 1.10e+10 7.0% | 1.10e+10 7.0% | 1.08e+10 0.6% | 1.08e+10 0.6 |
| ▶ transfb_._omp_fn.2 | 1.08e+10 6.9% | 1.08e+10 6.9% | 1.05e+10 0.6% | 1.05e+10 0.6 |

- lots of locks
- 8.4% of execution time waiting for locks
- 34% of lock waiting due to locks acquired at highlighted call site

# Blame shifting for locks, optimization



```
            do col=2,lx1-1
              do j=2,lx1-1
                il=idel(j,col,iface,ie)
                ig=idmo(j,col,1,1,iface,ie)
                if(xlindex.ne.0)then
c$                call omp_set_lock(tlock(xlindex))
                  tmor(xlindex)=tmor(xlindex)+xlval
c$                call omp_unset_lock(tlock(xlindex))
                  xlindex=0
                endif
c
                if(.not.omp_test_lock(tlock(ig)))then
                  xlindex=ig
                  xlval=tx(il)
                else
c                call omp_set_lock(tlock(ig))
                  tmor(ig)=tmor(ig)+tx(il)
c$                call omp_unset_lock(tlock(ig))
                endif
```

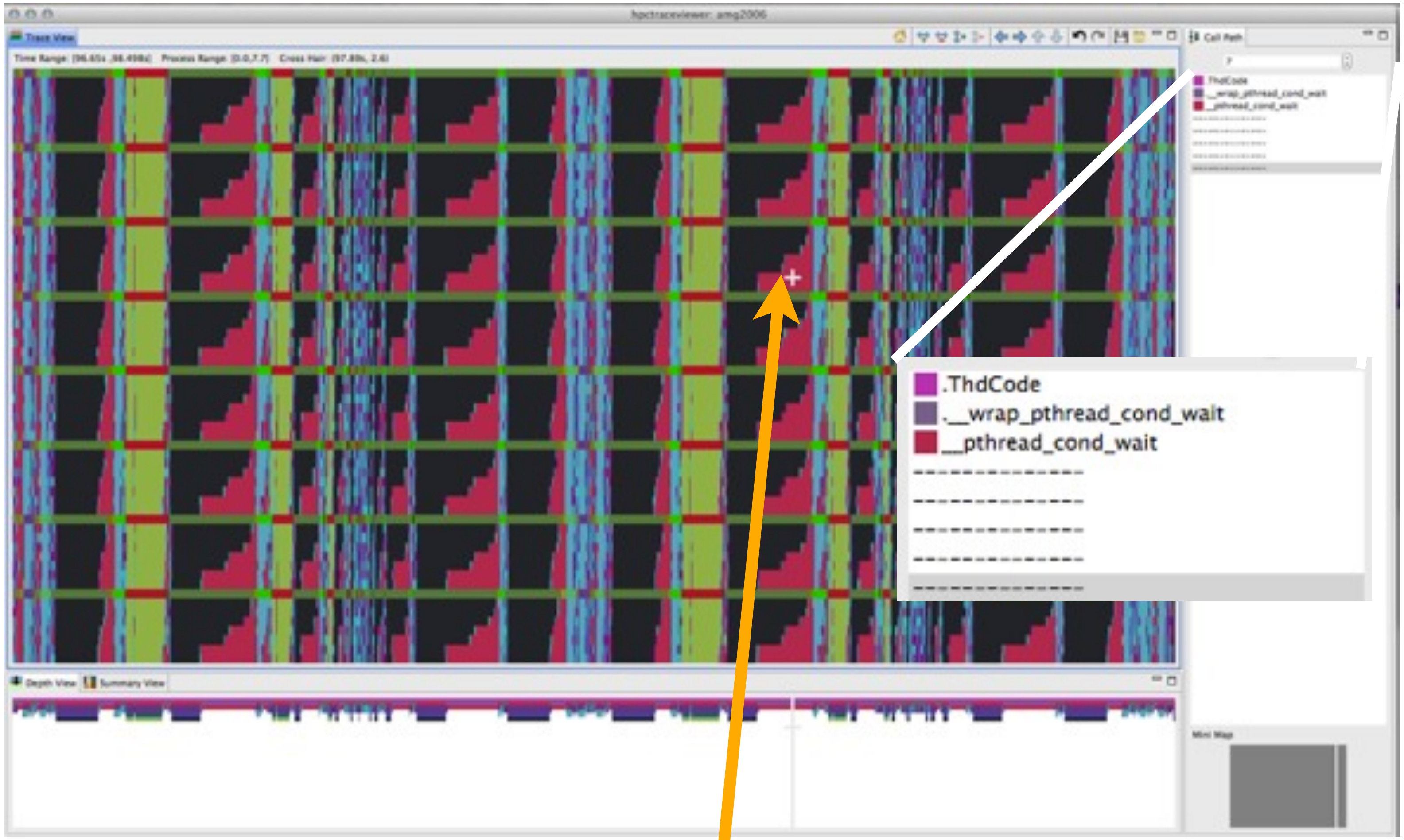
| Scope | LOCKWAIT:Sum (I) | LOCKWAIT:Sum (E | PAPI_TOT_CYC:Sum (E) | PAPI_TOT_CYC:Sum (I) |
|---|---|---|---|---|
| ▶ 914: transfb_c_._omp_fn.1 | 3.37e+08 0.3% | 3.37e+08 0.3% | 2.71e+08 0.0% | 2.71e+08 0.0% |
| ▼ 442: transfb_._omp_fn.2 | 2.13e+08 0.2% | 2.13e+08 0.2% | 2.00e+06 0.0% | 2.00e+06 0.0% |
| ▼ 277: transfb_ | 2.12e+08 0.2% | 2.12e+08 0.2% | 2.00e+06 0.0% | 2.00e+06 0.0% |
| ▼ 116: diffusion_ | 2.00e+08 0.2% | 2.00e+08 0.2% | 2.00e+06 0.0% | 2.00e+06 0.0% |
| ▼ 221: ua | 2.00e+08 0.2% | 2.00e+08 0.2% | 2.00e+06 0.0% | 2.00e+06 0.0% |
| 282: main | 2.00e+08 0.2% | 2.00e+08 0.2% | 2.00e+06 0.0% | 2.00e+06 0.0% |

- use omp_test_lock
- defer the lock acquisition to the next iteration
- eliminate the most lock contention time

# Tracing

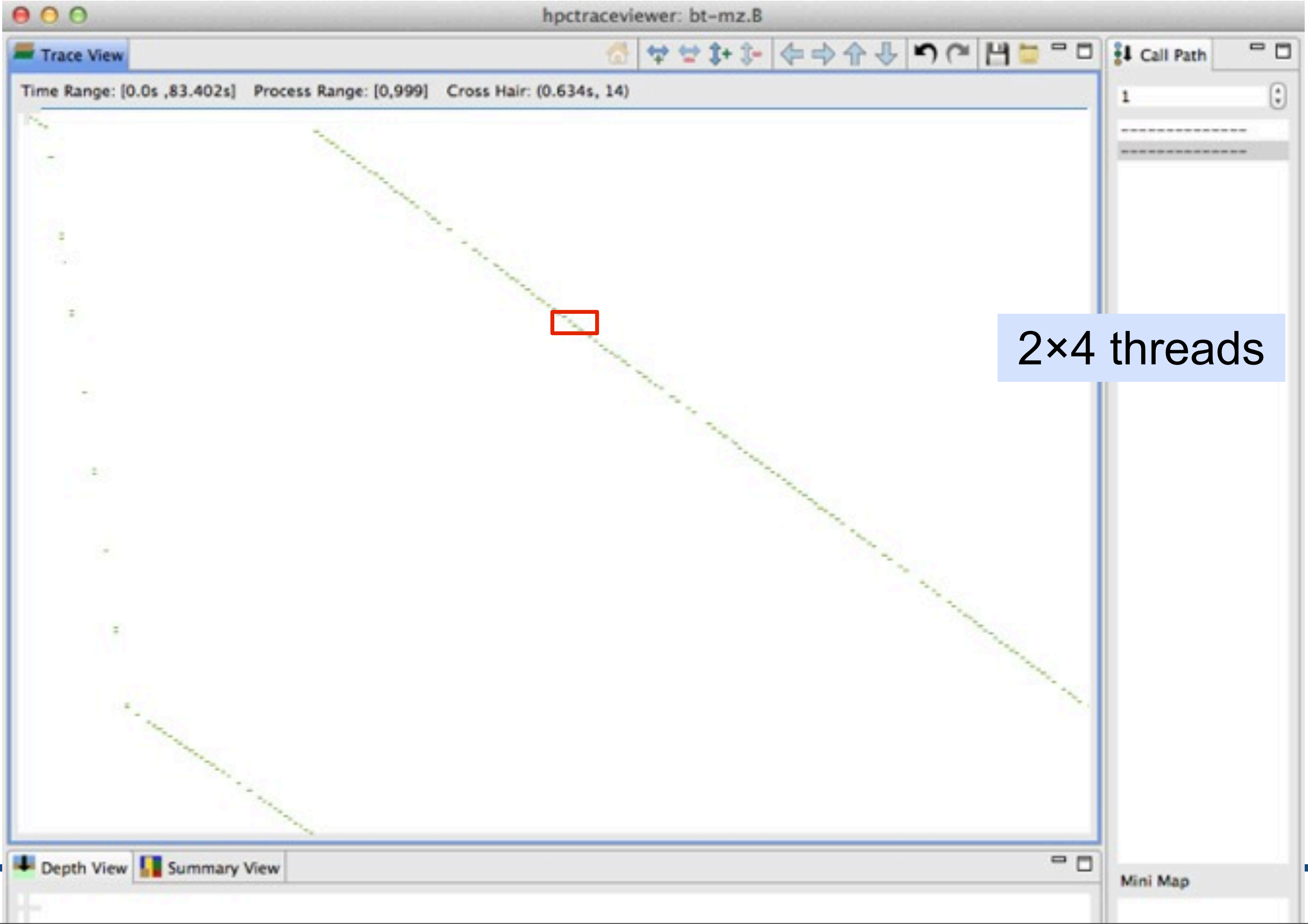
- Tracing
  - captures information about execution dynamics
  - trace visualization offers intuition into dynamic interplay between work, idleness, and overhead unfold during execution
- Issues
  - potentially high overhead
  - threads are frequently created/exit because no thread pool is used
- Solution
  - sampling-based tracing
  - no additional OpenMP runtime support beyond that for assembling user-level contexts
  - reuse the timeline of one thread and show the logical view

# Example: AMG 2006 (solver phase) trace



OpenMP loop in `hypre_BoomerAMGRelax` using static scheduling has load imbalance; threads idle for a significant fraction of their time

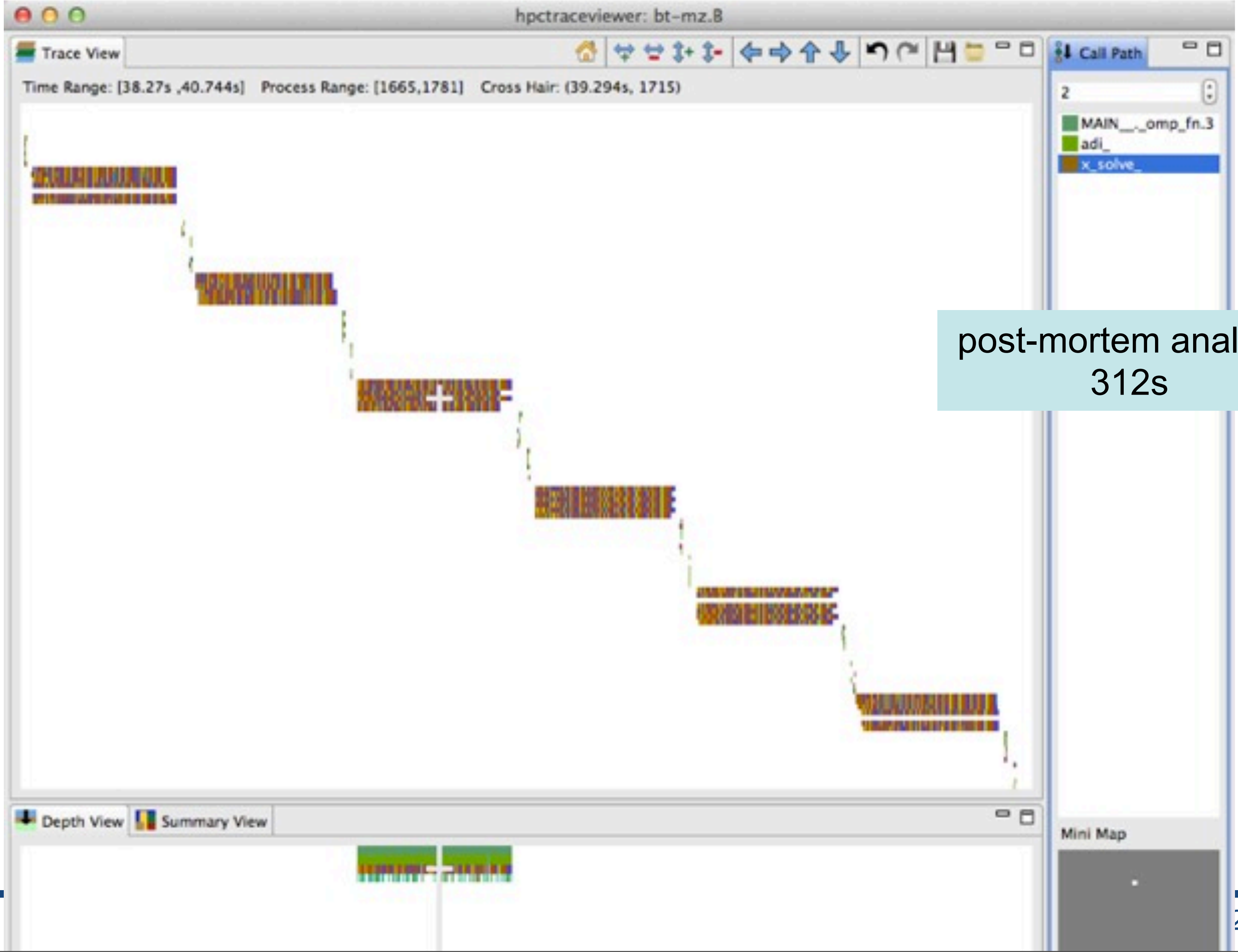
# BT-MZ nested parallelism tracing



Time Range: [0.0s ,83.402s]  Process Range: [0,999]  Cross Hair: (0.634s, 14)

2×4 threads

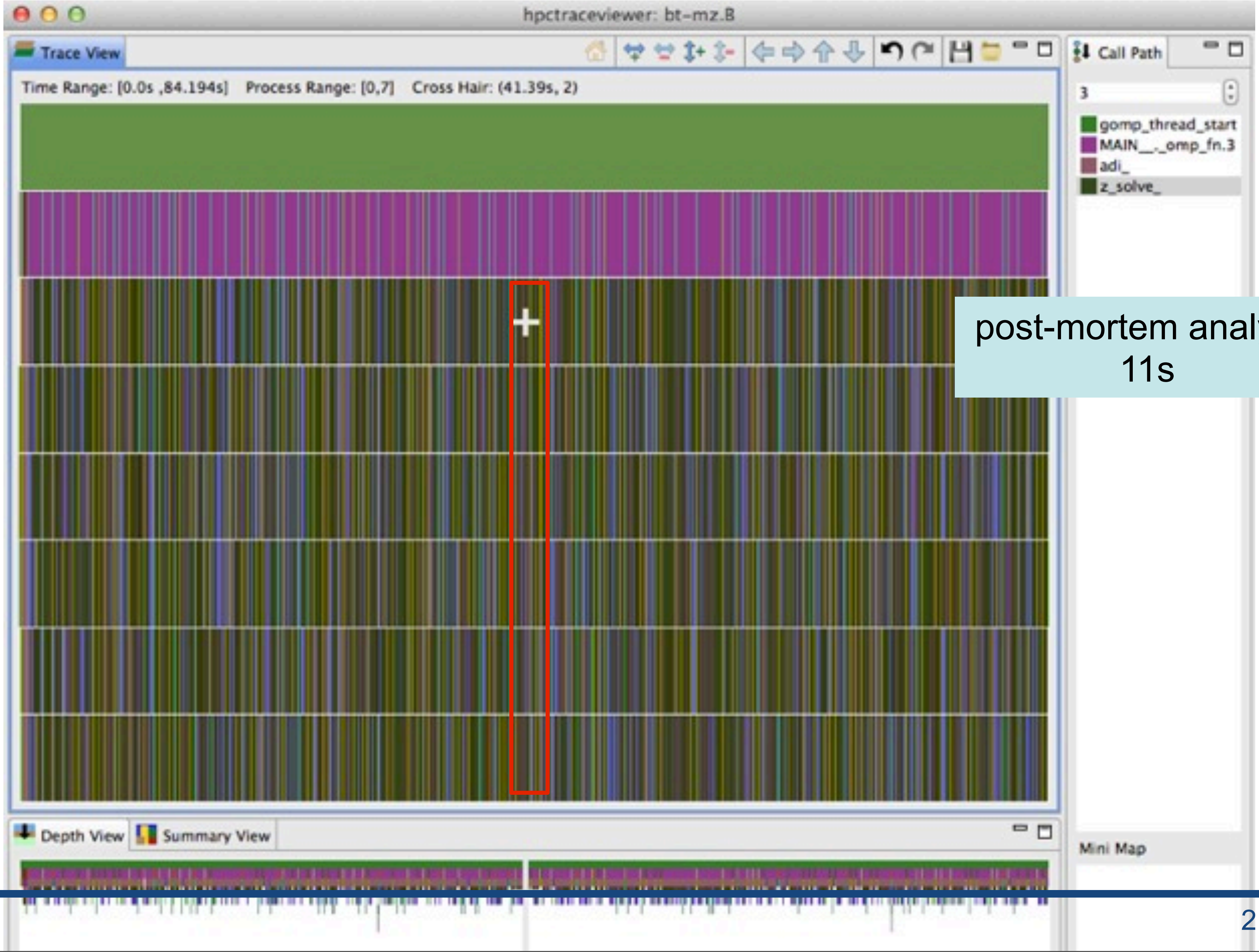
# BT-MZ nested parallelism tracing



Time Range: [38.27s ,40.744s]   Process Range: [1665,1781]   Cross Hair: (39.294s, 1715)

post-mortem analysis
312s

# BT-MZ logical trace view



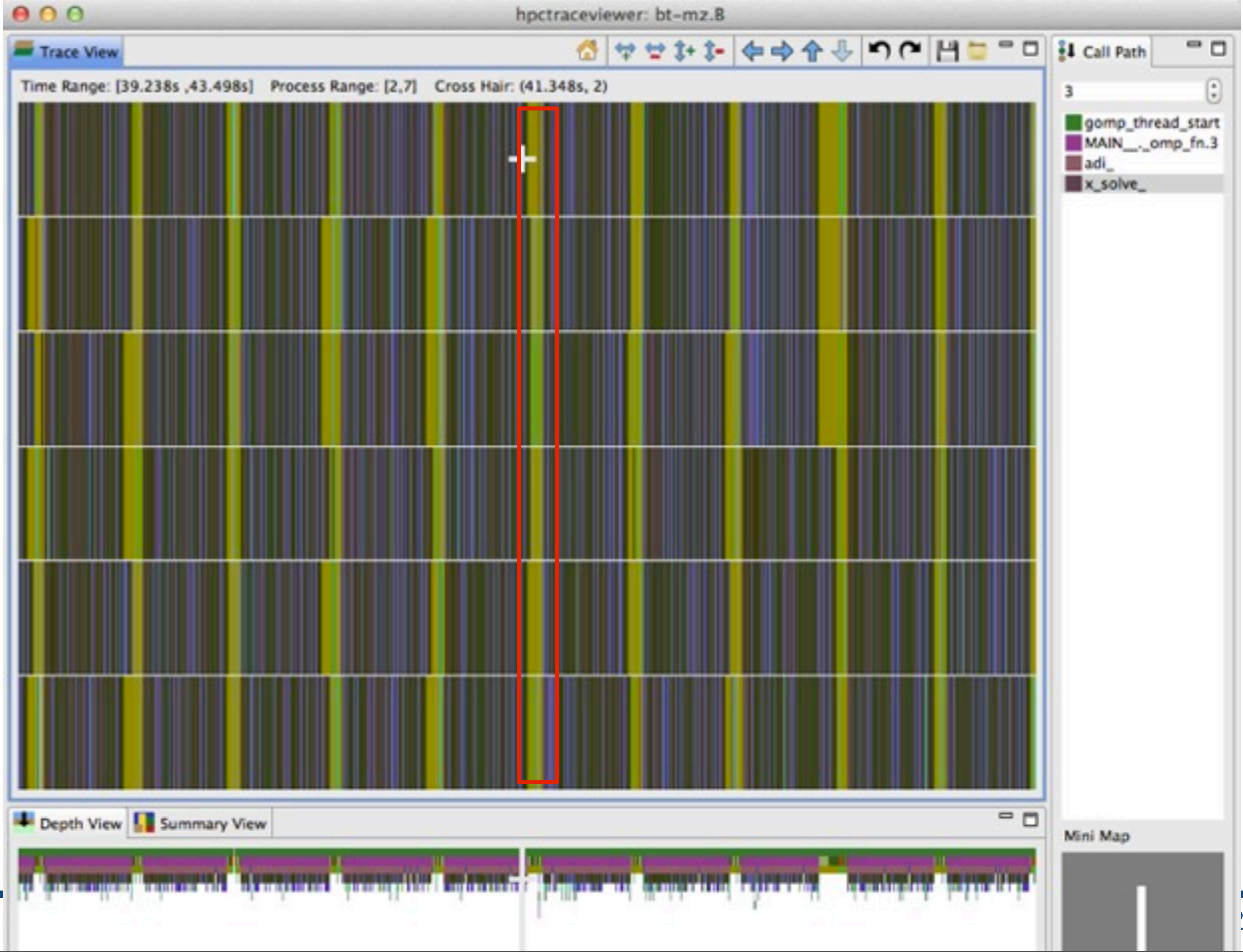
Time Range: [0.0s ,84.194s]   Process Range: [0,7]   Cross Hair: (41.39s, 2)

3

- 🟩 gomp_thread_start
- 🟪 MAIN__._omp_fn.3
- 🟫 adi_
- 🟩 z_solve_

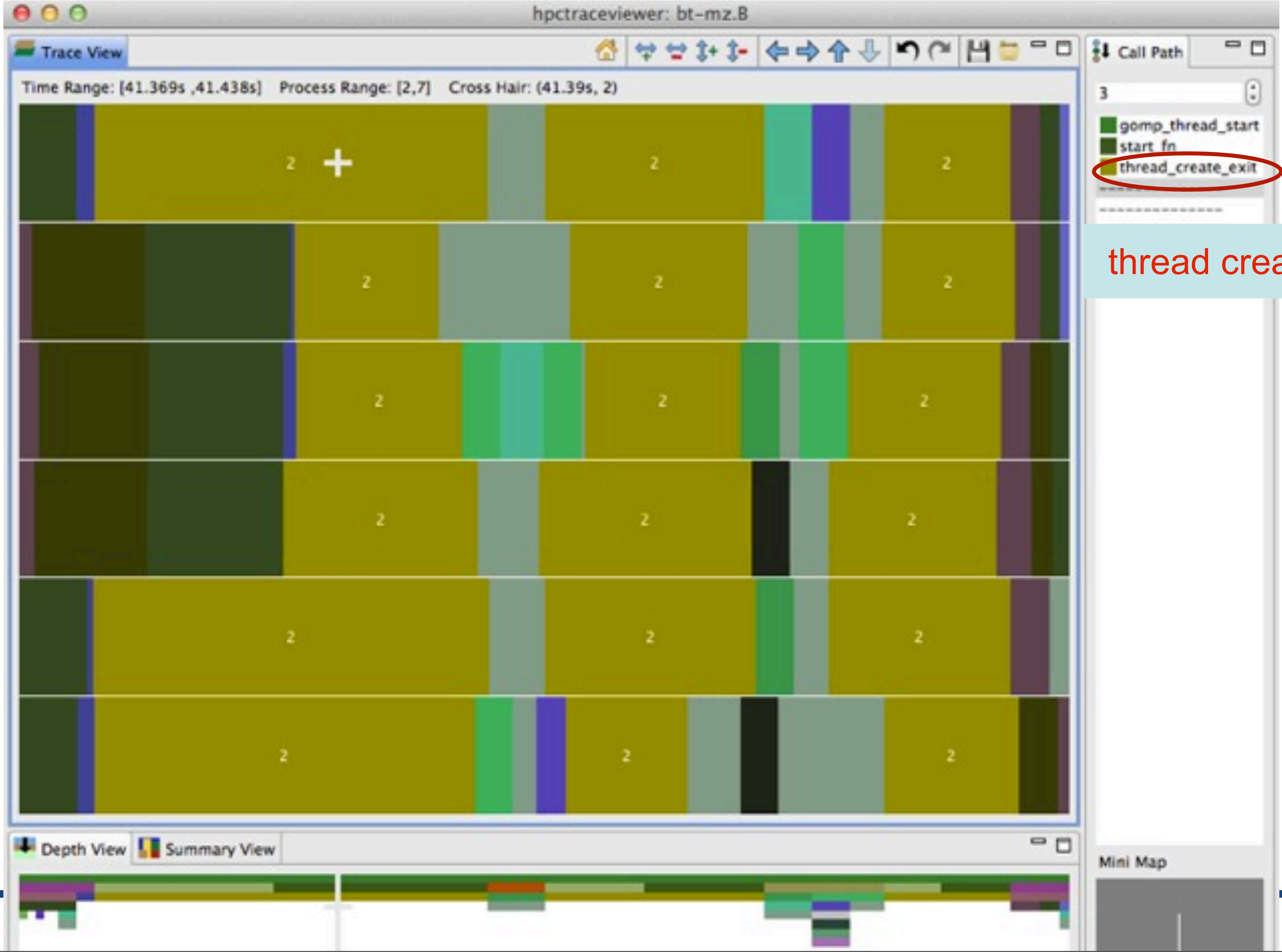
post-mortem analysis
11s

# BT-MZ logical trace view



Time Range: [39.238s ,43.498s] Process Range: [2,7] Cross Hair: (41.348s, 2)

# BT-MZ logical trace view



Time Range: [41.369s ,41.438s]  Process Range: [2,7]  Cross Hair: (41.39s, 2)

3

- 🟩 gomp_thread_start
- 🟩 start_fn
- 🟨 thread_create_exit

thread creation

# OpenMP runtime support for our tool API

## Fully worked example for GOMP (GNU OpenMP)

- Summary
  - changed 5 files
  - added less than 50 lines of code
- Principal changes
  - assign a region ID atomically as each parallel region is created
  - call to enter/exit callbacks at parallel region enter/exit
  - call to idle/work callbacks as threads enter/leave the barrier
  - call to start/end callbacks as threads start/end
  - add a pointer in the task structure to record the task creation context
  - call to task creation callbacks when an untied task is created
  - call to a lock_wait_callback callback when a lock acquire fails
  - call to a unlock_callback to release a lock
- Source and diffs available upon request

# Performance evaluation of tools API

- Three case studies
  - LULESH
    - a real application from LLNL
    - uses work-sharing parallel regions without nesting and tasking
    - 8 threads
  - BT-MZ.B
    - BT in multi-zone NPB with workload B
    - uses nested parallel regions without tasking
    - 8 threads: 2 for outer region and 4 for inner region
  - HEALTH
    - a benchmark in Barcelona tasking benchmarks
    - uses tasking: more than 17 million tasks
    - 8 threads, using medium input

# Profiling and tracing overhead

| applications | unmodified GOMP | modified GOMP w/o callbacks | modified GOMP w/ perf. measurement | | |
|---|---|---|---|---|---|
| | | | sampling | sampling +idleness | sampling +idleness +tracing |
| LULESH | 82.67s | 82.74s | 84.66s | 84.80s | 85.59s |
| BT-MZ.B | 60.87s | 61s | 72.81s<br>high overhead of PAPI profile initialization for thousands of dynamic threads | 72.60s | 83.22s |
| HEALTH | 72.78s | 71.56s | 73.18s<br>Associating tasks with execution context has low overhead; task creation context costs 424.58s | 73.56s | 72.07s |

lock contention
73.60s

Table measurements: average of three runs

Virtually no overhead if API not in use

# Summary

- Simple mechanisms in OpenMP runtime can support effective tools
  - slide 24 outlines suggested OpenMP runtime tools API mechanisms
  - almost no runtime overhead if suggested tools API is unused
  - suitable for use in a default high-performance runtime version
- We believe that any OpenMP tool API should include our suggested features
  - low to no overhead if unused (see slide 26)
  - low implementation cost (see slide 24)
- Other tool groups might want more extensive API features to support detailed tracing, e.g. POMP
  - if these cause significant overhead, we would prefer them to be supported in a separate “debugging” version of the runtime