Profiling with HPCToolkit

Mark W. Krentel
Department of Computer Science
Rice University
krentel@rice.edu

http://hpctoolkit.org

HPCToolkit Basic Features

- Run application natively (optimized) and every 100-200 times per second, interrupt program, unwind back to main(), record call stack, and combine these into a calling context tree (CCT).
- Combine sampling data with a static analysis of the program structure for loops and inline functions (hpcstruct).
- Present top-down, bottom-up and flat views of calling context tree (CCT) and time-sequence trace view. Metrics are displayed per source line in the context of their call path.
- Can sample on POSIX timers and Hardware Performance Counters (Perfmon or PAPI events): cycles, flops, cache misses, etc.
- Note: always include -g in compile flags (plus optimization) for attribution to source lines.

HPCToolkit Advanced Features

- Finely-tuned unwinder to handle multi-lingual, optimized code, no frame pointers, broken return pointers, stack trolling, etc.
- Derived metrics -- compute flops per cycle, or flops per memory reads, etc. and attribute to lines in source code.
- Compute strong and weak scaling loss, for example:

  strong: 8 * (time at 8K cores) - (time at 1K cores)

  weak: (time at 8K cores and 8x size) - (time at 1K cores)
- Load imbalance -- display distribution and variance in metrics across processes and threads.
- Blame shifting -- when thread is idle or waiting on a lock, blame the working threads or holder of lock.
- Inline sequences — show full inline sequence for C++ templates.

New Features

- Spack — now build hpctoolkit and prereqs with spack and install with spack modules.
- hpcstruct — now supports openmp threads.
  - — hpcstruct -j num …
- Kernel Blocktime — use Perf Events to count time spent blocked inside kernel, eg, I/O, barriers, locks, etc.
  - — hpcrun -e CYCLES -e BLOCKTIME …

Ongoing and Future Work

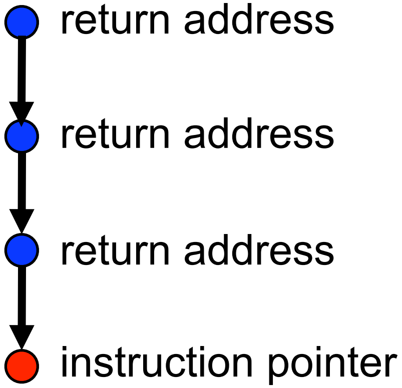
- **OpenMP parallel regions (in progress) — splice thread call paths onto master thread and identify work and idle (requires libomp replacement library), part of OpenMP 5.**
- **GPU (in progress) — count time in GPU, attach to call path where launched.**
- **Scaling (future) — better support for exascale size processes, threads and data.**

Call Path Profiling

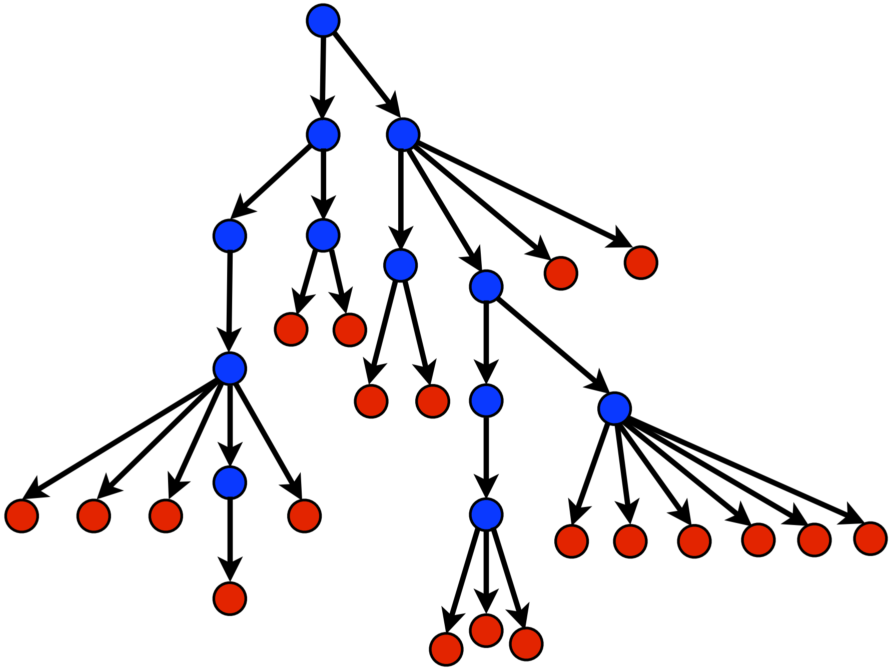
Measure and attribute costs in context

- sample timer or hardware counter overflows
- gather calling context using stack unwinding

Call path sample



Calling context tree



**Overhead proportional to sampling frequency...
...not call frequency**

Where to find HPCToolkit

- Home site: user’s manual, build instructions, links to source code, download viewers.

  http://hpctoolkit.org/

- On theta, available as module hpctoolkit and hpcviewer.

  module load hpctoolkit/2020.03.01

  module load hpcviewer/2020.02

- Source code on GitHub

  https://github.com/hpctoolkit

  git clone https://github.com/hpctoolkit/hpctoolkit

  spack build instructions: spack/README.spack

- Send questions to:

  hpctoolkit-forum at mailman.rice.edu

HPCToolkit Quickstart

- Unload Darshan module, edit Makefile, add hpclink to front of final link line.

```
hpclink cc file.o …
```

- Run job with HPCRUN environment variables (separated by spaces).

```
export HPCRUN_EVENT_LIST=“event@period  …”
export HPCRUN_TRACE=1
```

- Run hpcstruct on program binary (for loops and inline).

```
hpcstruct -j num program
```

- Run hpcprof to produce database.

```
hpcprof  -S program.hpcstruct  -I /path/to/source/tree/+  \
   hpctoolkit-measurements-directory
```

- View results with hpcviewer and hpctraceviewer.
Note: viewers on MacOS require Java 8 (not 9).

Running on Theta

- Load hpctoolkit module, unload darshan.

```
module load hpctoolkit/2020.03.01
module unload darshan
```

- On KNL, set sampling period to limit interrupts to about 100 per second. For example,

```
REALTIME@10000              (micro-seconds)
PAPI_TOT_CYC@14000000       (cycles)
CYCLES@f100                 (frequency per second)
```

- For large node counts (more than 50-100 nodes), reduce the process count for profiling with the following (or some other fraction).

```
export HPCRUN_PROCESS_FRACTION=0.1
```

Using OpenMP Tools Library

- Use hpctoolkit ompt module.

  module load hpctoolkit/2020.04.ompt

- Compile with -fopenmp, but on hpclink link line, replace -fopenmp with libomp.a from LLVM runtime. Supports GNU, Intel and Clang. On theta,

  /projects/Tools/hpctoolkit/pkgs-theta/llvm-openmp/lib/libomp.a

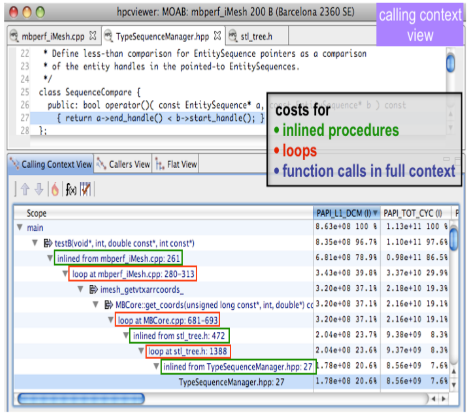
- Add event OMP_IDLE (no number) plus time-based event: REALTIME, PAPI_TOT_CYC or CYCLES.

- Workarounds on theta to turn off thread affinity.

  aprun —cc none …

  export KMP_AFFINITY=none

HPCToolkit Capabilities at a Glance



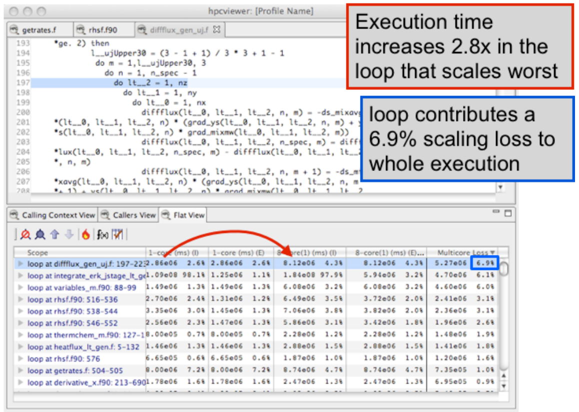
calling context view

costs for
- inlined procedures
- loops
- function calls in full context

```
* Define less-than comparison for EntitySequence pointers as a comparison
* of the entity handles in the pointed-to EntitySequences.
*/
class SequenceCompare {
  public: bool operator()( const EntitySequence* a,
    { return a->end_handle() < b->start_handle(); }
};
```

Scope	PAPI_L1_DCM (I)	PAPI_TOT_CYC (I)
main	8.63e+08 100 %	1.13e+11 100 %
testB(void*, int, double const*, int const*)	8.35e+08 96.7%	1.10e+11 97.6%
inlined from mbperf_iMesh.cpp: 261	6.81e+08 78.9%	9.98e+10 86.5%
loop at mbperf_iMesh.cpp: 280-313	3.43e+08 39.8%	3.37e+10 29.9%
imesh_getvtxarrcoords_	3.20e+08 37.1%	2.18e+10 19.3%
MBCore::get_coords(unsigned long const*, int, double*) const	3.20e+08 37.1%	2.16e+10 19.1%
loop at MBCore.cpp: 681-693	3.20e+08 37.1%	2.16e+10 19.1%
inlined from stl_tree.h: 472	2.04e+08 23.7%	9.38e+09 8.3%
loop at stl_tree.h: 1388	2.04e+08 23.6%	9.37e+09 8.3%
inlined from TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%
TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%

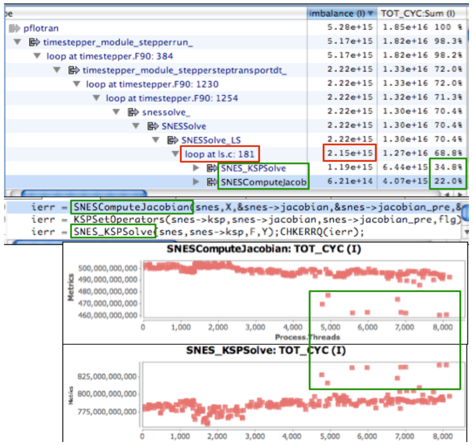
Attribute Costs to Code



Execution time increases 2.8x in the loop that scales worst

loop contributes a 6.9% scaling loss to whole execution

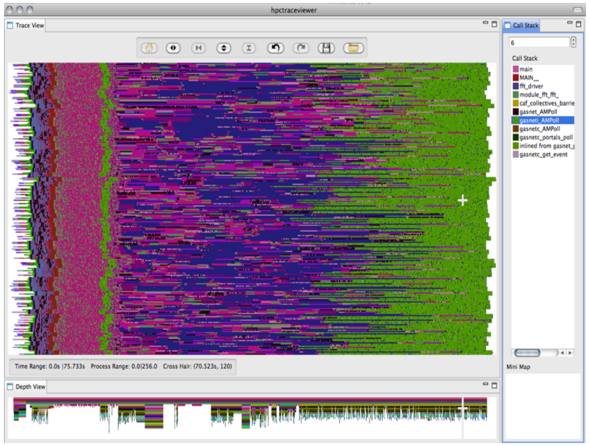
Pinpoint & Quantify Scaling Bottlenecks



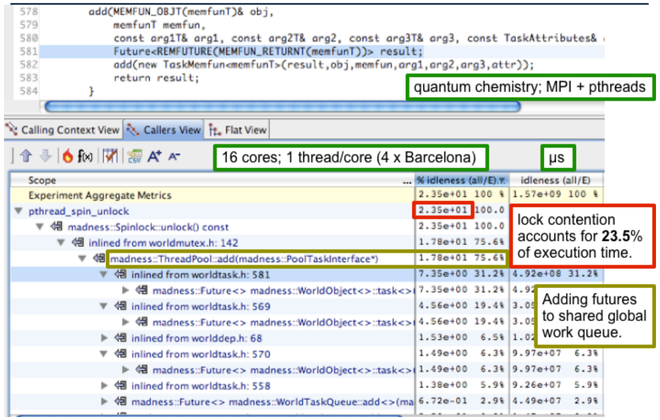
SNESComputeJacobian: TOT_CYC (I)

SNES_KSPSolve: TOT_CYC (I)

Assess Imbalance and Variability



Analyze Behavior over Time



```
add(MEMFUN_OBJT(memfunT)& obj,
    memfunT memfun,
    const arg1T& arg1, const arg2T& arg2, const arg3T& arg3, const TaskAttributes&
    Future<REMFUTURE(MEMFUN_RETURN(memfunT))> result;
    add(new TaskMemfun<memfunT>(result,obj,memfun,arg1,arg2,arg3,attr));
    return result;
}
```

quantum chemistry; MPI + pthreads

16 cores; 1 thread/core (4 x Barcelona)

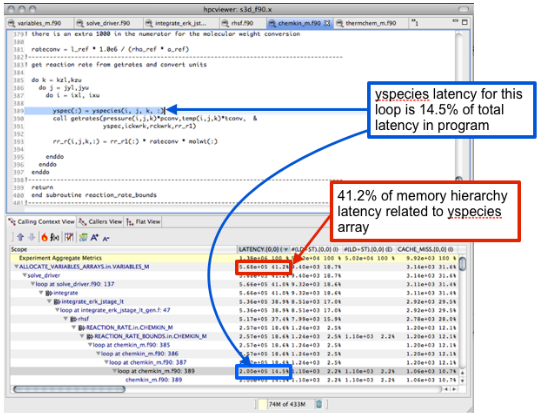
µs

Scope	% idleness (all/E)	idleness (all/E)
Experiment Aggregate Metrics	2.35e+01 100 %	1.57e+09 100 %
pthread_spin_unlock	2.35e+01 100.0	
madness::Spinlock::unlock() const	2.35e+01 100.0	
inlined from worldmutex.h: 142	1.78e+01 75.6%	
madness::ThreadPool::add(madness::PoolTaskInterface*)	1.78e+01 75.6%	
inlined from worldtask.h: 581	7.35e+00 31.2%	4.92e+08 31.2%
madness::Future<> madness::WorldObject<>::task<>	7.35e+00 31.2%	4.92
inlined from worldtask.h: 569	4.56e+00 19.4%	3.05
madness::Future<> madness::WorldObject<>::task<>	4.56e+00 19.4%	3.05
inlined from worlddep.h: 68	1.53e+00 6.5%	1.02
inlined from worldtask.h: 570	1.49e+00 6.3%	9.97e+07 6.3%
madness::Future<> madness::WorldObject<>::task<>	1.49e+00 6.3%	9.97e+07 6.3%
inlined from worldtask.h: 558	1.38e+00 5.9%	9.26e+07 5.9%
madness::Future<> madness::WorldTaskQueue::add<>(ma	6.72e-01 2.9%	4.49e+07 2.9%

lock contention accounts for 23.5% of execution time.

Adding futures to shared global work queue.

Shift Blame from Symptoms to Causes



yspecies latency for this loop is 14.5% of total latency in program

41.2% of memory hierarchy latency related to yspecies array

Associate Costs with Data

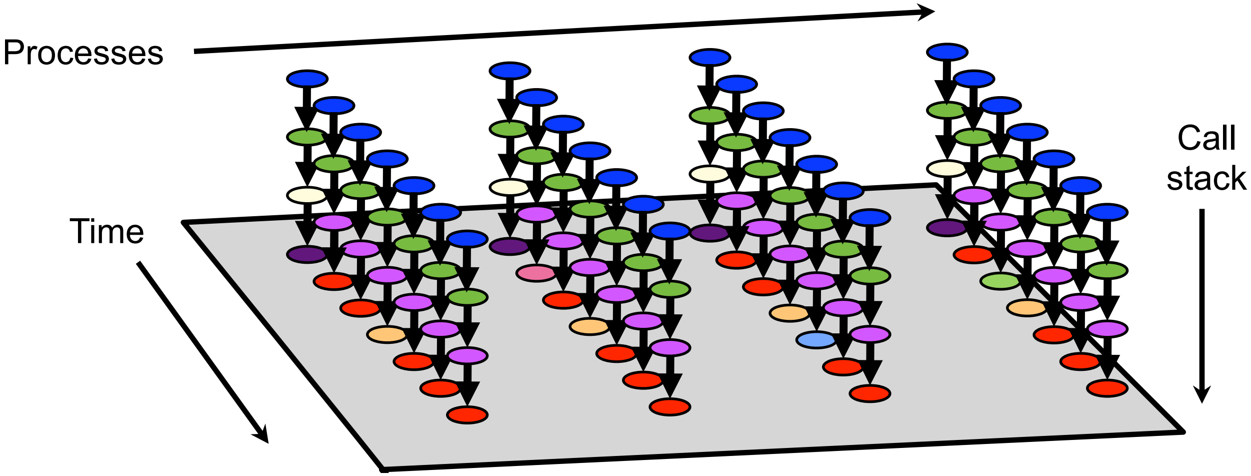
RICE

hpctoolkit.org

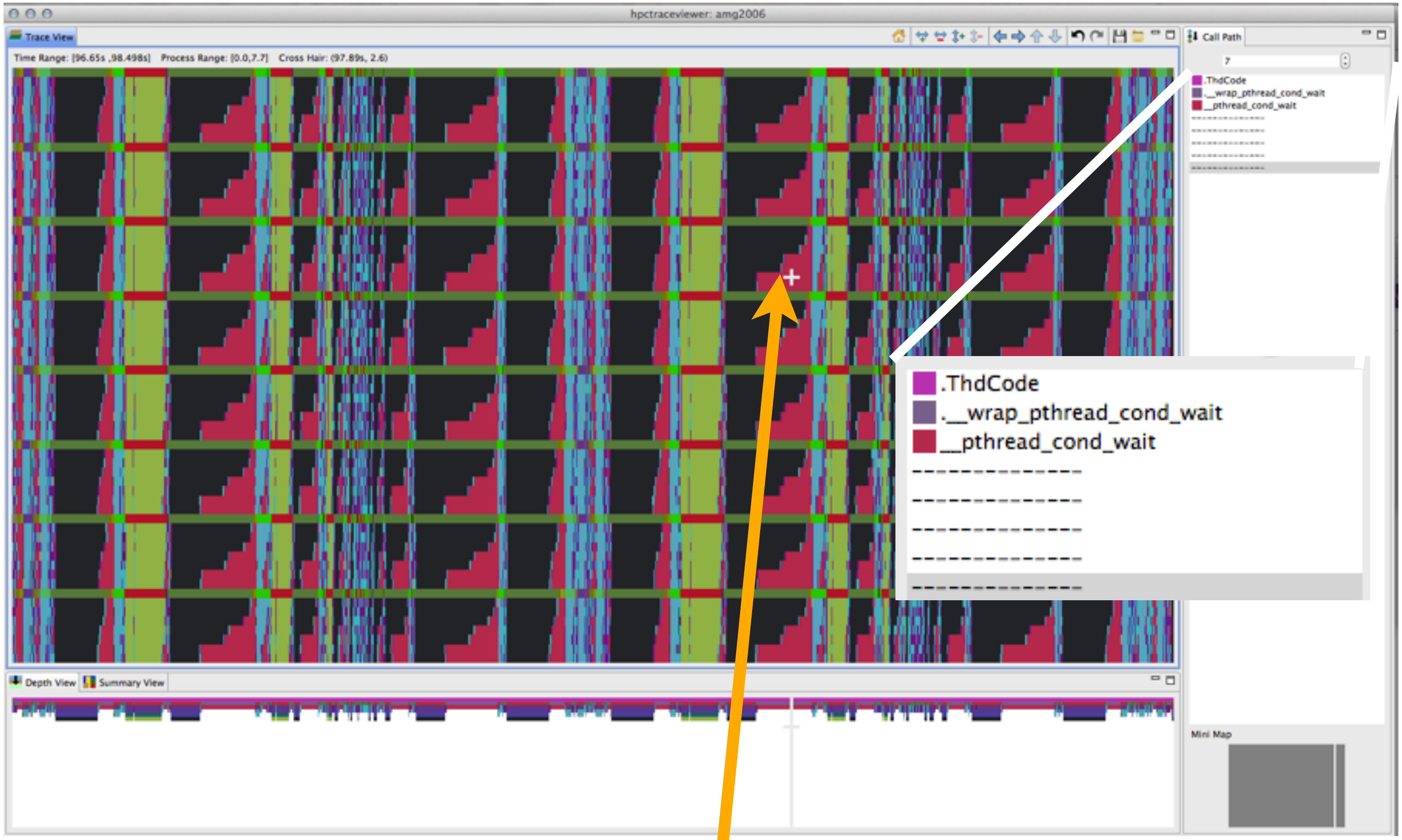
Understanding Temporal Behavior

- **Profiling compresses out the temporal dimension**
  - —temporal patterns, e.g. serialization, are invisible in profiles
- **What can we do? Trace call path samples**
  - —sketch:
    - N times per second, take a call path sample of each thread
    - organize the samples for each thread along a time line
    - view how the execution evolves left to right
    - what do we view?

      assign each procedure a color; view a depth slice of an execution



AMG2006: 8PE x 8 OMP Threads

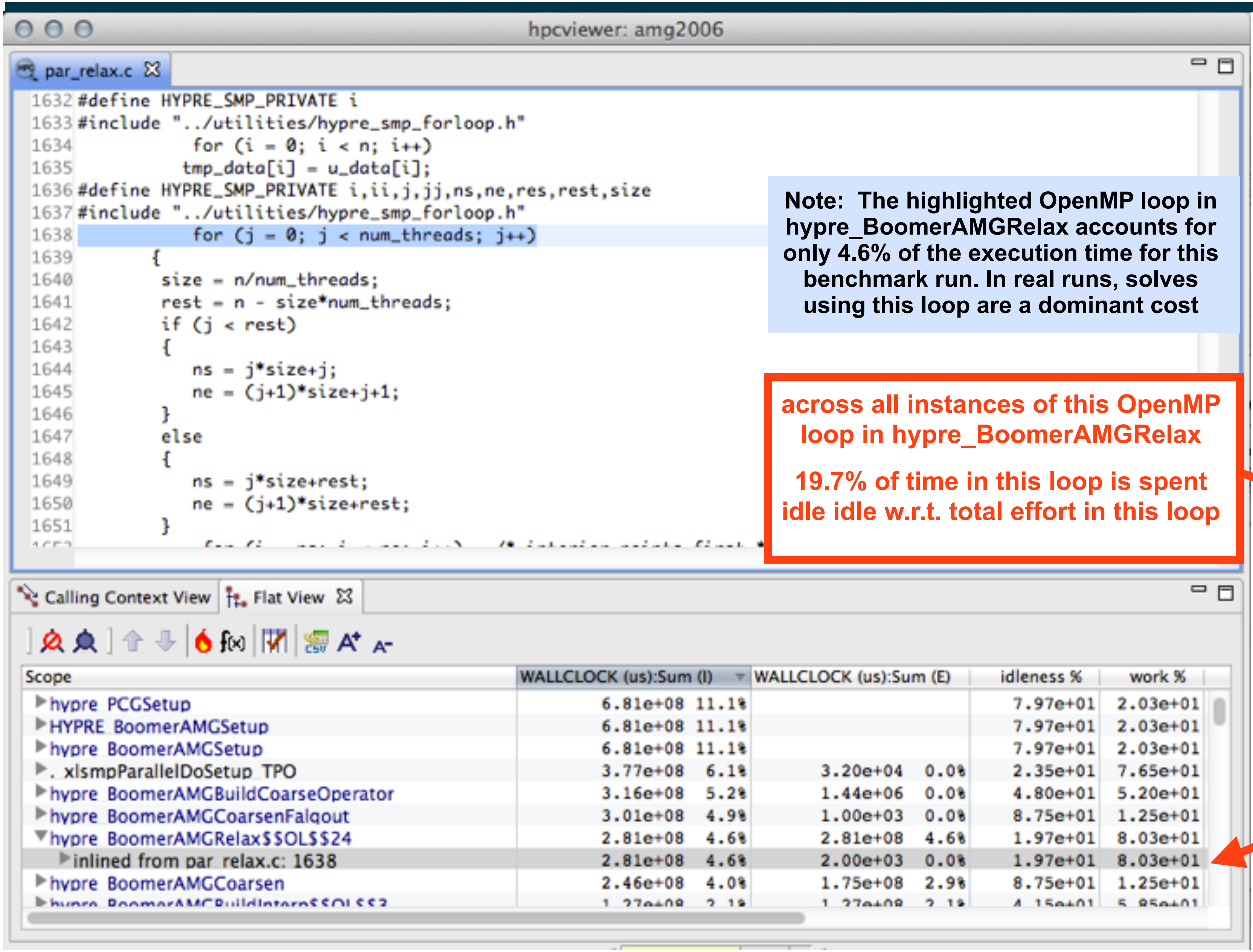


Time Range: [96.65s ,98.498s] Process Range: [0.0,7.7] Cross Hair: (97.89s, 2.6)

🟪 .ThdCode
🟪 .__wrap_pthread_cond_wait
🟥 __pthread_cond_wait

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

Code-centric view: `hypre_BoomerAMGRelax`



```
1632 #define HYPRE_SMP_PRIVATE i
1633 #include "../utilities/hypre_smp_forloop.h"
1634            for (i = 0; i < n; i++)
1635           tmp_data[i] = u_data[i];
1636 #define HYPRE_SMP_PRIVATE i,ii,j,jj,ns,ne,res,rest,size
1637 #include "../utilities/hypre_smp_forloop.h"
1638            for (j = 0; j < num_threads; j++)
1639        {
1640         size = n/num_threads;
1641         rest = n - size*num_threads;
1642         if (j < rest)
1643         {
1644            ns = j*size+j;
1645            ne = (j+1)*size+j+1;
1646         }
1647         else
1648         {
1649            ns = j*size+rest;
1650            ne = (j+1)*size+rest;
1651         }
1652            for (i = ns; i < ne; i++)   /* interior points first *
```

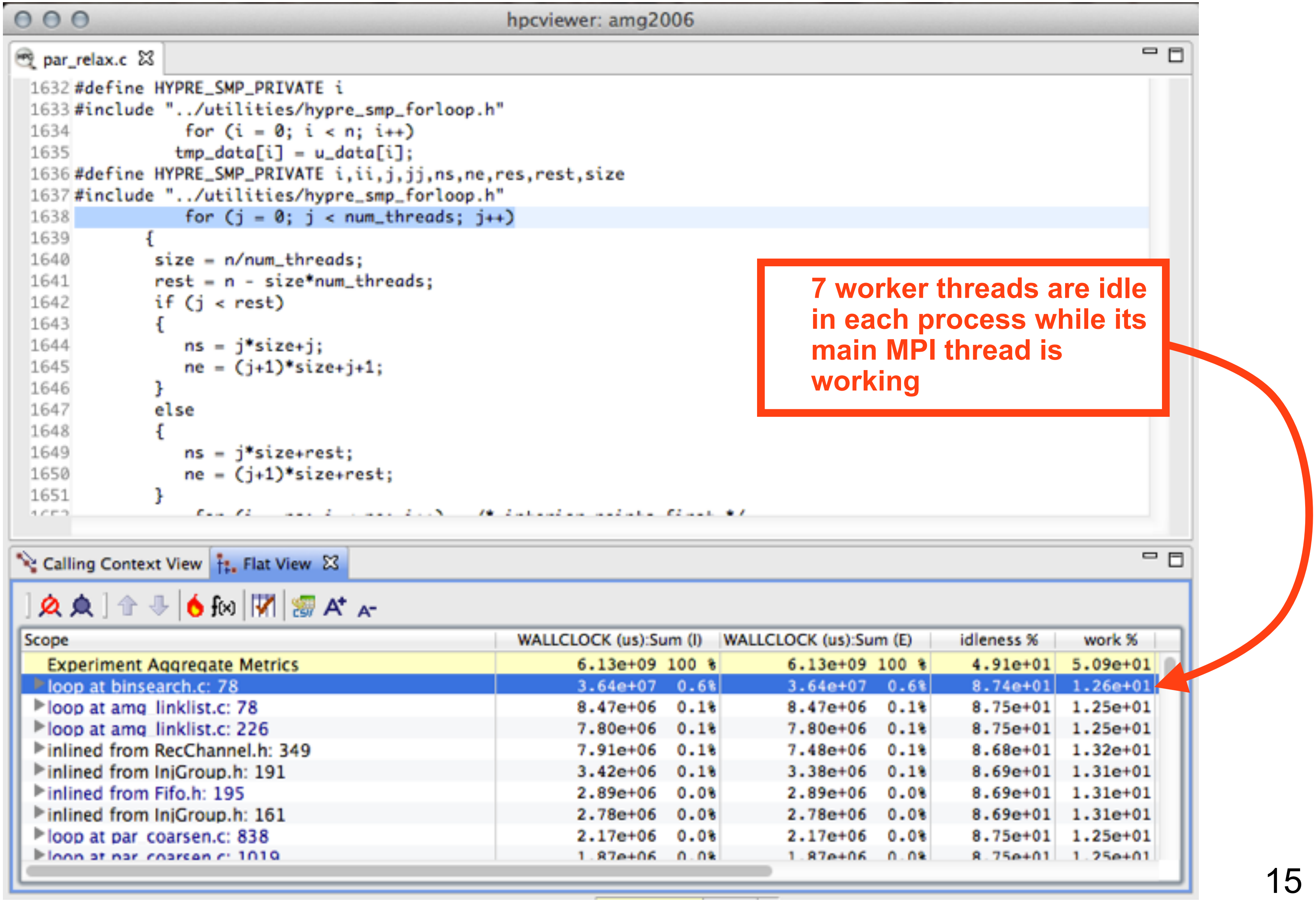
Scope	WALLCLOCK (us):Sum (I)		WALLCLOCK (us):Sum (E)		idleness %	work %
▶hypre_PCGSetup	6.81e+08	11.1%			7.97e+01	2.03e+01
▶HYPRE_BoomerAMGSetup	6.81e+08	11.1%			7.97e+01	2.03e+01
▶hypre_BoomerAMGSetup	6.81e+08	11.1%			7.97e+01	2.03e+01
▶._xlsmpParallelDoSetup_TPO	3.77e+08	6.1%	3.20e+04	0.0%	2.35e+01	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

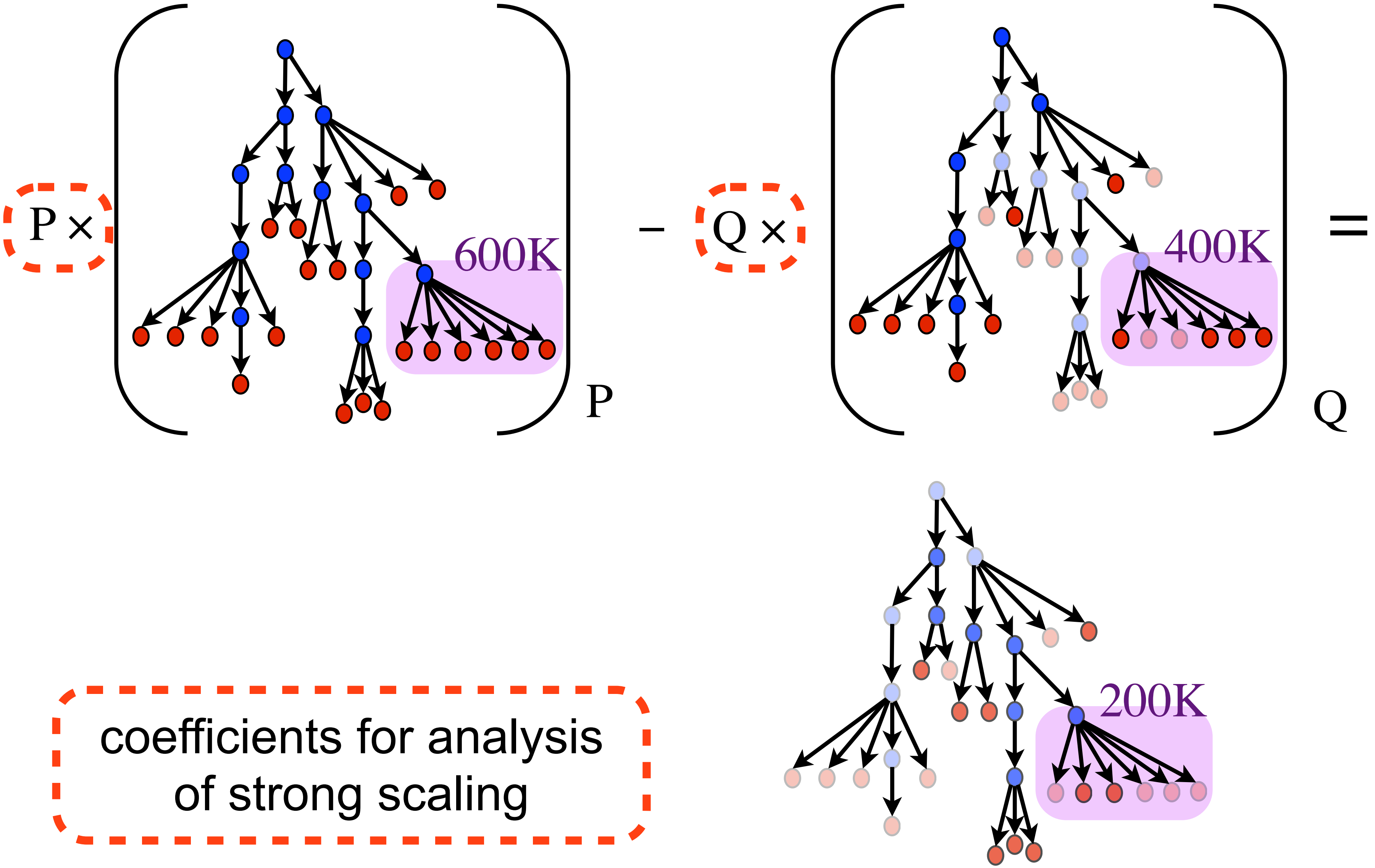


```
#define HYPRE_SMP_PRIVATE i
#include "../utilities/hypre_smp_forloop.h"
            for (i = 0; i < n; i++)
           tmp_data[i] = u_data[i];
#define HYPRE_SMP_PRIVATE i,ii,j,jj,ns,ne,res,rest,size
#include "../utilities/hypre_smp_forloop.h"
            for (j = 0; j < num_threads; j++)
      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

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

Pinpointing and Quantifying Scalability Bottlenecks



coefficients for analysis
of strong scaling