

# Profiling with HPCToolkit

Mark W. Krentel

Department of Computer Science

Rice University

[krentel@rice.edu](mailto: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.**
- **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 Wallclock (itimer), POSIX timers and Hardware Performance Counters (Perf Events and 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:  $8 * (\text{time at 8K cores}) - (\text{time at 1K cores})$**   
**weak:  $(\text{time at 8K cores and 8x size}) - (\text{time at 1K cores})$**
- **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.
- **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.**



# Where to find HPCToolkit

---

- **Home site: user's manual, build instructions, links to source code, download viewers.**  
<http://hpctoolkit.org/>
- **On theta, add to PATH:**  
</projects/Tools/hpctoolkit/pkg-theta/hpctoolkit/bin/>
- **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](mailto:hpctoolkit-forum@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.**  
`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 -l /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

---

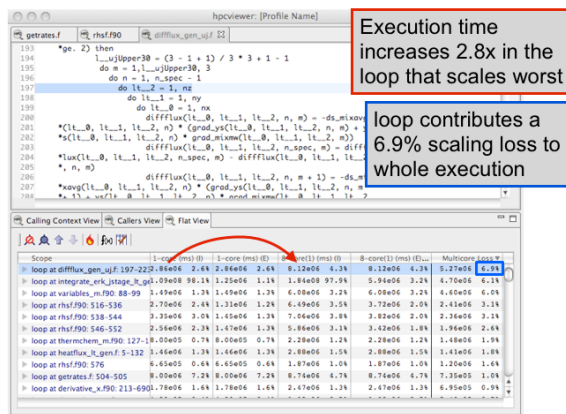
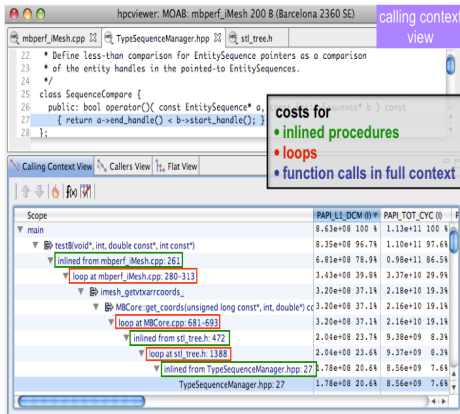
- **Add to PATH:**  
`/projects/Tools/hpctoolkit/pkgs-theta/hpctoolkit/bin/`
- **On KNL, set sampling period to limit interrupts to about 100 per second. For example,**  
`REALTIME@10000`  
`PAPI_TOT_CYC@14000000`  
`CYCLES@f100`
- **For large node counts (more than 50-100 nodes), reduce the process count for profiling with the following (or some other fraction).**  
`export HPCRUN_PROCESS_FRACTION=0.1`

# Using OpenMP Tools Library

---

- Use `hpclink` from `hpctoolkit-ompt`. On theta,  
`/projects/Tools/hpctoolkit/pkgs-theta/hpctoolkit-ompt/bin/  
hpclink`
- Compile with `-fopenmp`, but on `hpclink` link line, replace `-fopenmp` with `libomp.a` from LLVM runtime. 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`

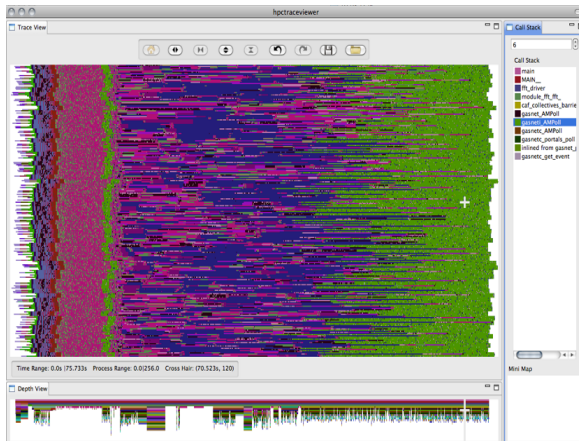
# HPCToolkit Capabilities at a Glance



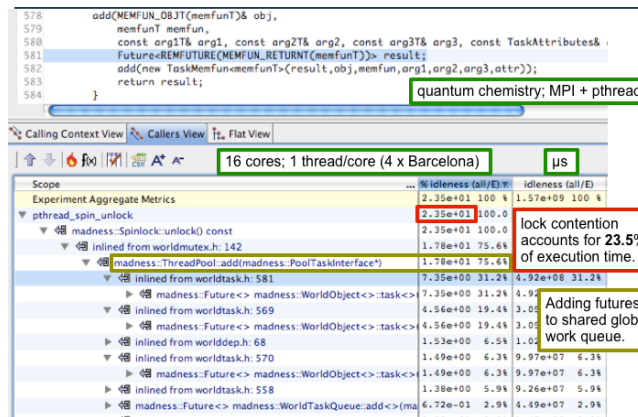
Attribute Costs to Code

Pinpoint & Quantify Scaling Bottlenecks

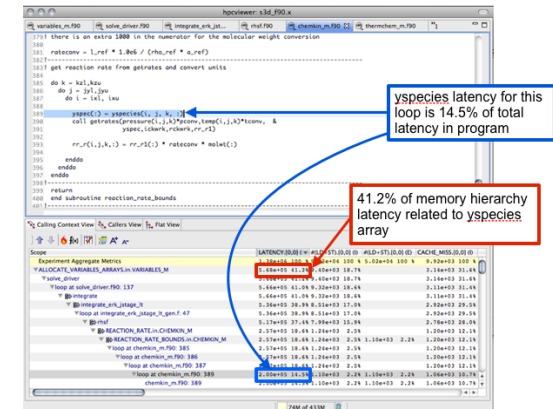
Assess Imbalance and Variability



Analyze Behavior over Time



Shift Blame from Symptoms to Causes

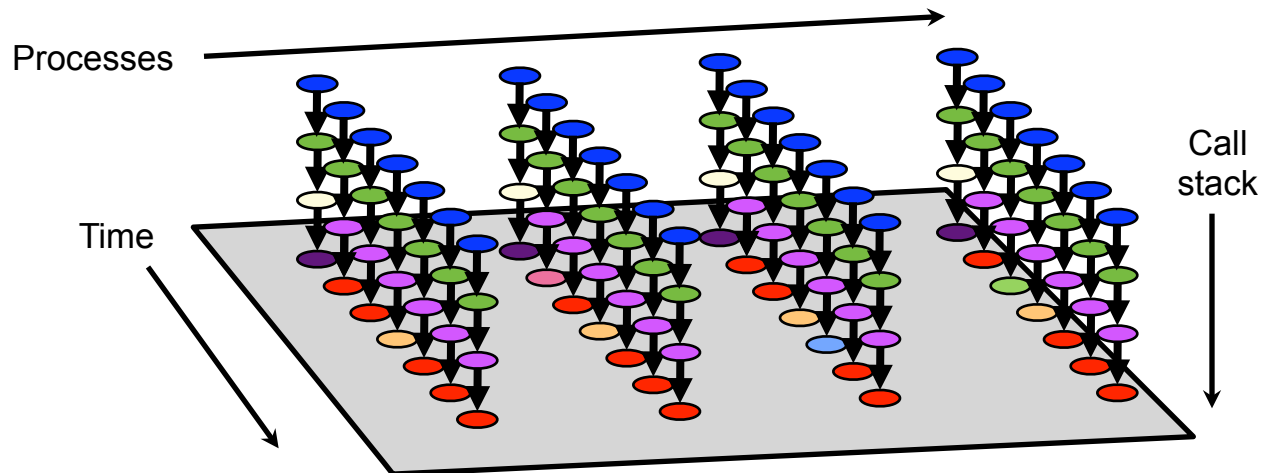


Associate Costs with Data

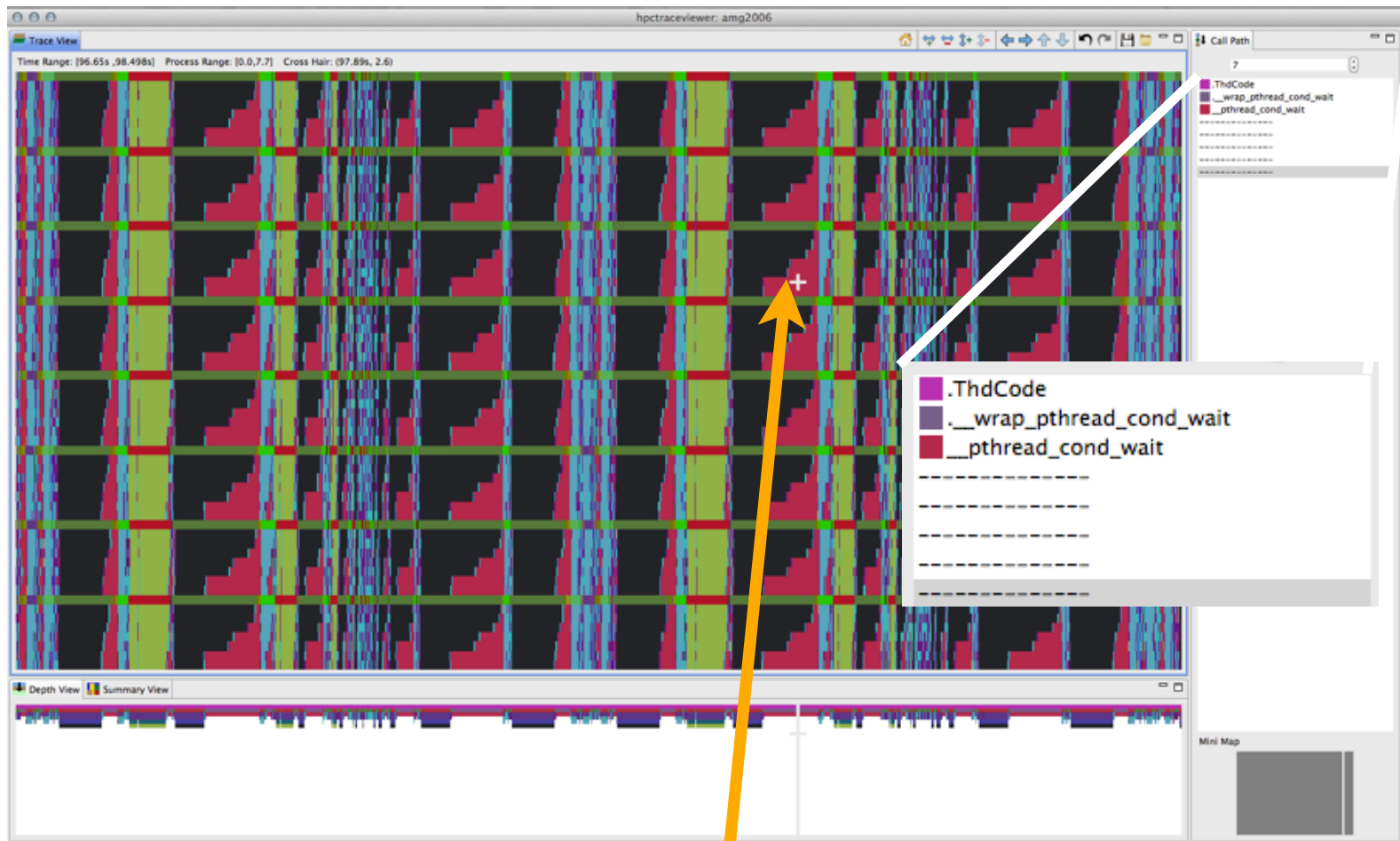


# Understanding Temporal Behavior

- 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



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

The screenshot shows the hpcviewer interface for a benchmark run named 'amg2006'. The top pane displays the source code for 'par\_relax.c', with an OpenMP loop highlighted in blue. A blue callout box notes that this loop accounts for only 4.6% of the execution time. The bottom pane shows a performance table with columns for Scope, WALLCLOCK (us):Sum (I), WALLCLOCK (us):Sum (E), idleness %, and work %. A red callout box highlights that 19.7% of the time in this loop is spent idle, with an arrow pointing to the 'idleness %' column for the 'inlined from par\_relax.c: 1638' entry.

```
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     }
```

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 w.r.t. total effort in this loop

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 BoomerAMGCoarsenFalqout	3.01e+08 4.9%	1.00e+03 0.0%	8.75e+01	1.25e+01
▼ hypre BoomerAMGRelax\$SOLS\$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 BoomerAMGBuildIterate\$SOLS\$22	1.27e+08 2.1%	1.27e+08 2.1%	4.15e+01	5.85e+01

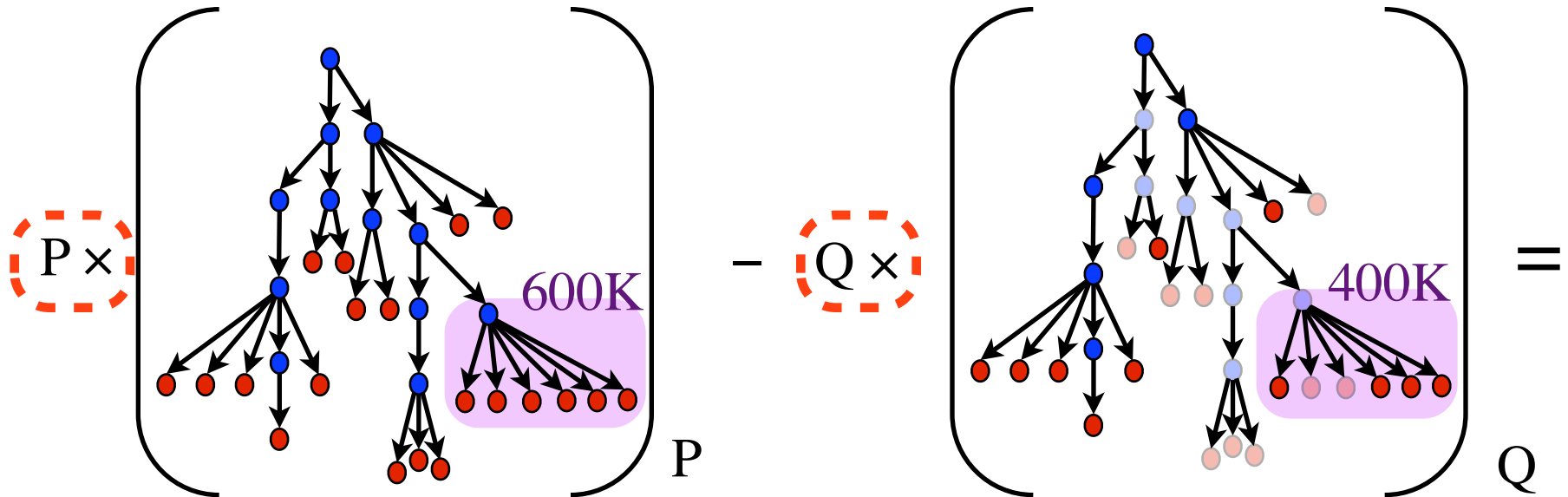
# Serial Code in AMG2006 8 PE, 8 Threads

The screenshot shows the hpcviewer interface for the 'amg2006' application. The top pane displays the source code for 'par\_relax.c'. The code includes a loop over 'num\_threads' (lines 1638-1651) which is highlighted in blue. A red box with white text is overlaid on the right side of the code, stating: "7 worker threads are idle in each process while its main MPI thread is working". An arrow points from this box to the 'loop at binsearch.c: 78' row in the performance table below.

```
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     }
```

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

# Pinpointing and Quantifying Scalability Bottlenecks



coefficients for analysis of strong scaling

