
Evolving HPCToolkit

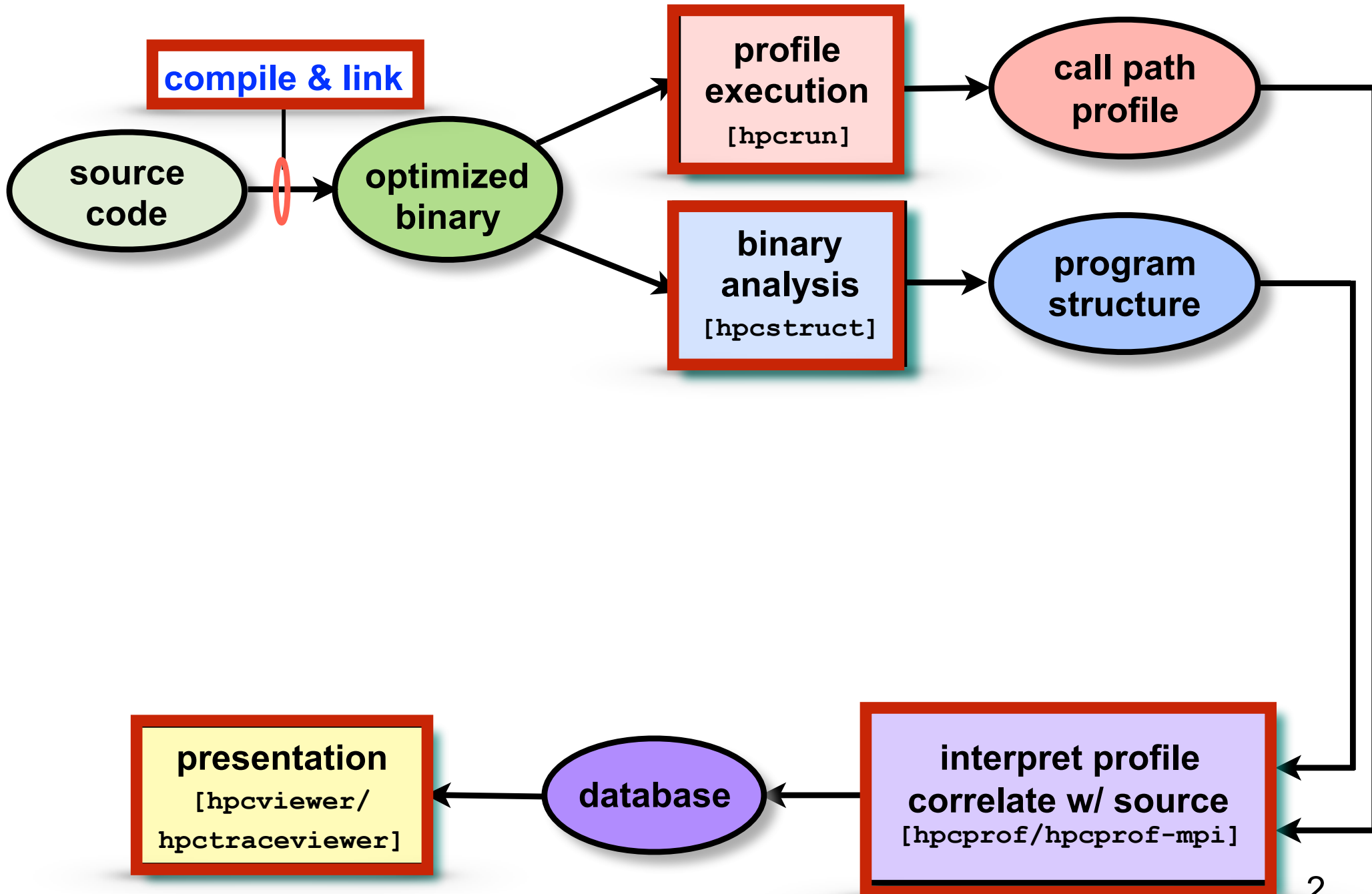
John Mellor-Crummey
Department of Computer Science
Rice University



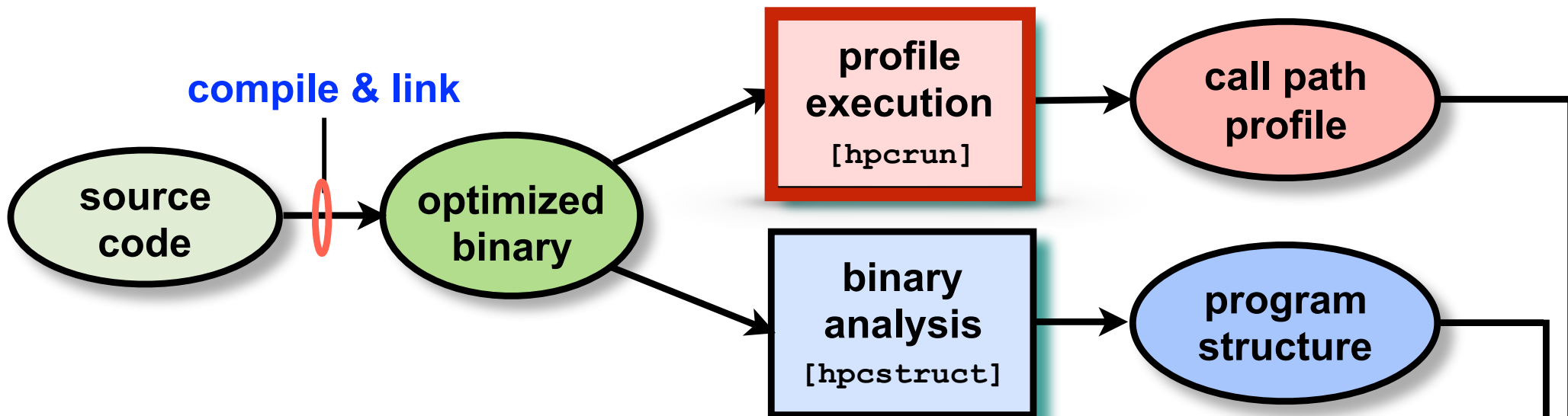
<http://hpctoolkit.org>



HPCToolkit Workflow



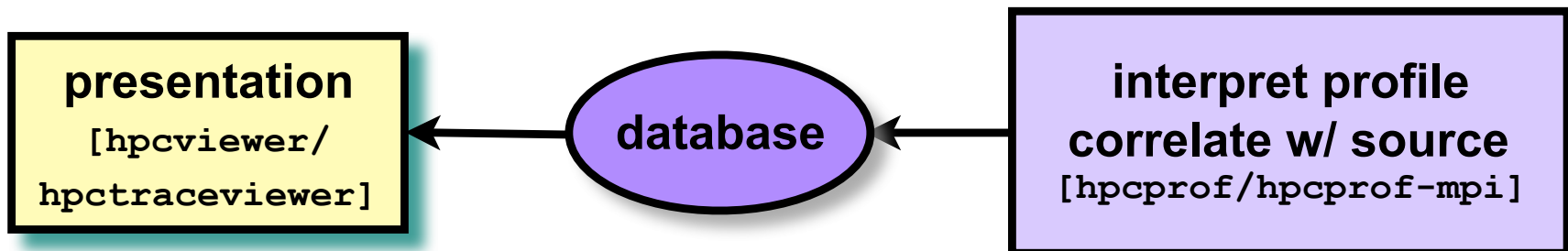
HPCToolkit Workflow



Ongoing work

- **Improving measurement**
- **Improving attribution to source**
- **Accelerating analysis with multithreaded parallelism**

Next Steps



Call Path Profiling of Optimized Code

- **Optimized code presents challenges for stack unwinding**
 - optimized code often lacks frame pointers
 - routines may have multiple epilogues, multiple frame sizes
 - code may be partially stripped: no info about function bounds
- **HPCToolkit's approach for nearly a decade**
 - use binary analysis to compute unwinding recipes for intervals
 - often, no compiler information to assist unwinding is available
 - cache unwind recipes for reuse at runtime (more about this later)

Nathan R. Tallent, John Mellor-Crummey, and Michael W. Fagan. Binary analysis for measurement and attribution of program performance. Proceedings of ACM *PLDI*. ACM, New York, NY, USA, 2009, 441–452. Distinguished Paper. ([doi:10.1145/1542476.1542526](https://doi.org/10.1145/1542476.1542526))

Challenges for Unwinding

- **Binary analysis of optimized multithreaded applications has become increasingly difficult**
 - **previously: procedures were typically contiguous**
 - **today: procedures are often discontinuous**

Code generated by
Intel's OpenMP compiler

```
void f(...) {  
    ...  
    #pragma omp parallel  
    {  
        ...  
    }  
    ...  
}
```

New Unwinding Approach in HPCToolkit

- **Use libunwind to unwind procedure frames where compiler-provided information is available**
- **Use binary analysis for procedure frames where no unwinding information is available**
- **Transition seamlessly between the two approaches**
- **Status:**
 - first implementation for x86_64 completed on Friday
 - under evaluation

Surprises

- libunwind sometimes unwound incorrectly from signal contexts [our fixes are now in libunwind git]
- On Power, register frame procedures are not only at call chain leaves [unwind fixes in an hpctoolkit branch]

Caching Unwind Recipes in HPCToolkit

Concurrent Skip Lists

- **Two-level data structure: concurrent skip list of binary trees**
 - maintain a concurrent skip list of procedure intervals
 - [proc begin, proc end)
 - associate an immutable balanced binary tree of unwind recipes with each procedure interval
- **Synchronization needs**
 - scalable reader/writer locks [Brandenburg & Anderson; RTS '10]
 - read lock: find, insert
 - write lock: delete
 - MCS queuing locks [Mellor-Crummey & Scott; ACM TOCS '91]
 - lock skip-list predecessors to coordinate concurrent inserts

Validating Fast Synchronization

- **Used C++ weak atomics in MCS locks and phase-fair reader/writer synchronization**
 - **against Herb Sutter's advice**
 - **C++ and Beyond 2012: atomic<> Weapons (bit.ly/atomic_weapons)**
 - **as Herb predicted: we got it wrong!**
 - **Wrote small benchmarks that exercised our synchronization**
 - **Identified bugs with CDS checker - model checker for C11 and C++11 Atomics**
 - **http://plrg.eecs.uci.edu/software_page/42-2/**
- Brian Norris and Brian Demsky. CDSchecker: checking concurrent data structures written with C/C++ atomics. Proceedings of the 2013 ACM SIGPLAN OOPSLA. 2013. ACM, New York, NY, USA, 131-150. (doi: [10.1145/2509136.2509514](https://doi.org/10.1145/2509136.2509514))
- **Fixed them**
 - **Validated the use of C11 atomics by our primitives**

**We recommend CDS checker
to others facing similar issues**

Understanding Kernel Activity and Blocking

- Some programs spend a lot of time in the kernel or blocked
- Understanding their performance requires measurement of kernel activity and blocking

```
...
#define N 1000000000
#define REPS 32

int main()
{
    omp_set_num_threads(2);
    int pagesize = sysconf(_SC_PAGESIZE);
    printf("page size = %d\n", pagesize);
    int nextpage = pagesize/sizeof(int);
    for (int i=0; i < REPS; i++) {
        int *v = malloc(sizeof(int) * N);
        #pragma omp parallel
        {
            #pragma omp for
                for(int j = 0; j < N; j += nextpage) {
                    v[j] = 5;
                }
        }
        free(v);
    }
    return 0;
}
```

Measuring Kernel Activity and Blocking

- **Problem**
 - Linux timers and PAPI are inadequate
 - neither measure nor precisely attribute kernel activity
- **Approach**
 - layer HPCToolkit directly on top of Linux perf_events
 - also sample kernel activity: perf_events collect kernel call stack
 - use sampling in conjunction with Linux CONTEXT_SWITCH events to measure and attribute blocking

VTune Result

The screenshot displays the Intel VTune Amplifier XE 2016 interface. The main window shows a 'Call Stack' view of CPU Time. The 'page_fault' function is highlighted with a purple box. A call stack window on the right shows the path: `test2!_start` -> `main` -> `main_omp_fn` -> `page_fault`. A purple text box on the right contains the text: "performance problem appears to be page faults".

Function Stack	Effective Time by Utilization	Spin Time	Ove.. Time	Effective Time by
Total	100.0%	0.0%	0.0%	0s
_start	55.7%	0.0%	0.0%	0s
_libc_start_main	55.7%	0.0%	0.0%	0s
main	55.7%	0.0%	0.0%	0s
main_omp_fn	43.5%	0.0%	0.0%	0.963s
page_fault	39.5%	0.0%	0.0%	9.707s
apic_timer_in	0.1%	0.0%	0.0%	0.020s
retint_carefu	0.0%	0.0%	0.0%	0s
munmap	12.1%	0.0%	0.0%	2.978s
func@0xf120	0.1%	0.0%	0.0%	0.030s
printf	0.0%	0.0%	0.0%	0s
_clone	44.3%	0.0%	0.0%	0s
[Unknown stack fra	0.0%	0.0%	0.0%	0s
kretprobe_trampoline	0.0%	0.0%	0.0%	0s

Selected 1 row(s): 55.7% 0.0% 0.0%

Understanding Kernel Activity with HPCToolkit

The screenshot displays the HPCToolkit interface. The top pane shows the source code for `test2.c`, with line 24 (`v[j] = 5;`) highlighted. A purple callout box on the right explains that the real problem is zero-filling pages returned to and reacquired from the OS. The bottom pane shows the performance metrics table, with a purple box highlighting the `clear_huge_page` and `clear_user_page` entries.

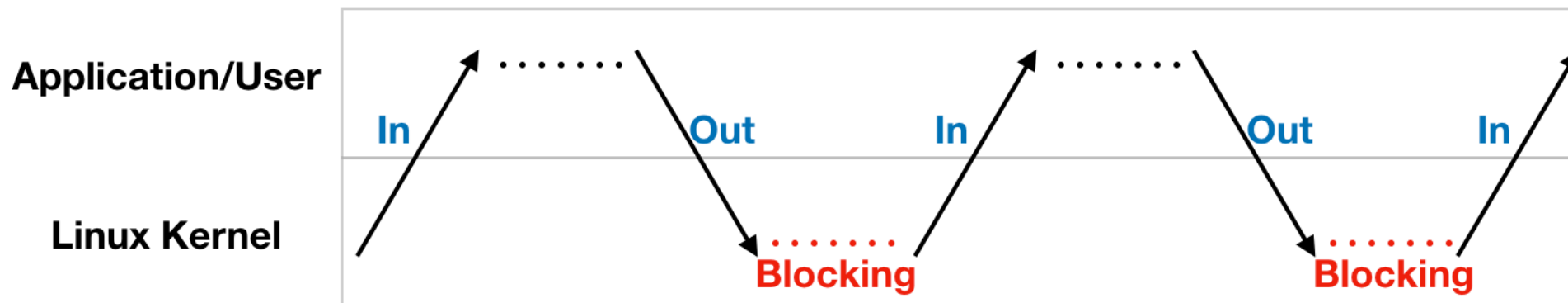
```
7 #define N 1000000000
8 #define REPS 32
9
10
11 int main()
12 {
13     omp_set_num_threads(2);
14     int pagesize = sysconf(_SC_PAGESIZE);
15     printf("page size = %d\n", pagesize);
16     int nextpage = pagesize/sizeof(int);
17
18     for (int i=0; i < REPS; i++) {
19         int *v = malloc(sizeof(int) * N);
20 #pragma omp parallel
21     {
22 #pragma omp for
23         for(int j = 0; j < N; j += nextpage) {
24             v[j] = 5;
25         }
26     }
27     free(v);
28 }
29 return 0;
30 }
```

the real problem:
zero-filling pages
returned to and
reacquired from the OS

Scope	PERF_COUNT_HW_CPU_CYCLES:Sum (I)	PERF_COUNT_HW_CPU_CYCLES:Sum (E)
Experiment Aggregate Metrics	1.65e+10 100 %	1.65e+10 100 %
<program root>	8.35e+09 50.7%	
497: main	8.35e+09 50.7%	
loop at test2.c: 18	8.35e+09 50.7%	
20: main_omp_fn.0	7.62e+09 46.3%	4.00e+06 0.0%
loop at test2.c: 24	7.62e+09 46.3%	4.00e+06 0.0%
24: handle_page_fault	7.55e+09 45.9%	
do_page_fault	7.55e+09 45.9%	
handle_mm_fault	7.52e+09 45.7%	
do_huge_pmd_anonymous_page	7.49e+09 45.5%	
<unknown procedure>	7.37e+09 44.7%	
clear_huge_page	7.37e+09 44.7%	
clear_user_page	7.32e+09 44.5%	7.24e+09 44.0%

Kernel Blocking

Original idea: Measure kernel **blocking** time

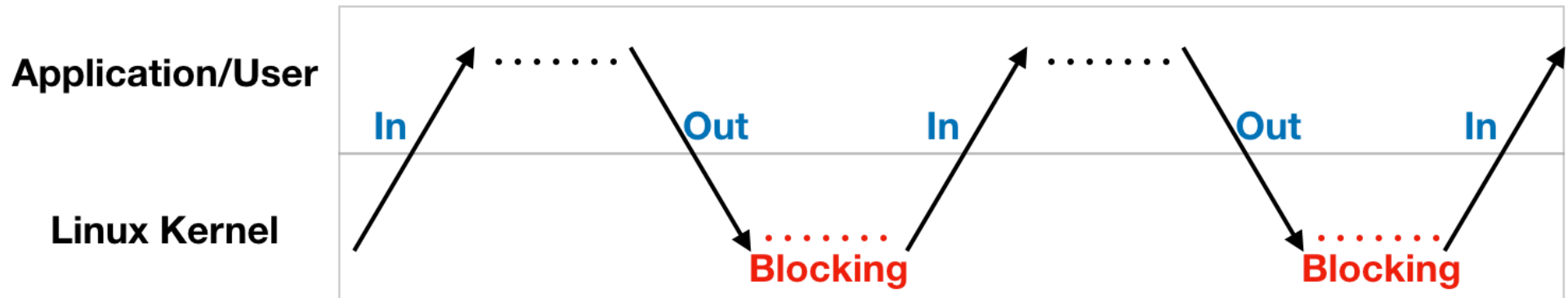


Surprise

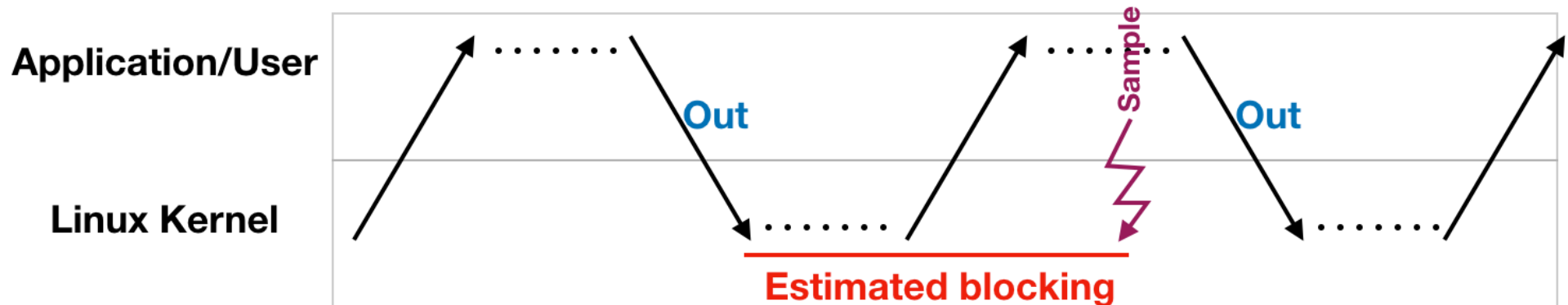
- Third-party monitoring: SWITCH_OUT & SWITCH_IN
- First party monitoring: SWITCH_OUT only
- IBM Linux team working to upstream a fix

Kernel Blocking

Original idea: Measure kernel **blocking** time



Approximation: Estimate kernel blocking time



Measuring Kernel Blocking

hpcviewer: test2

main.c test2.c pthread.c

```

10
11 int main()
12 {
13     omp_set_num_threads(2);
14     int pagesize = sysconf(_SC_PAGESIZE);
15     printf("page size = %d\n", pagesize);
16     int nextpage = pagesize/sizeof(int);
17
18     for (int i=0; i < REPS; i++) {
19         int *v = malloc(sizeof(int) * N);
20 #pragma omp parallel
21     {
22 #pragma omp for
23         for(int j = 0; j < N; j += nextpage) {
24             v[j] = 5;
25         }
26     }
27     free(v);
28 }

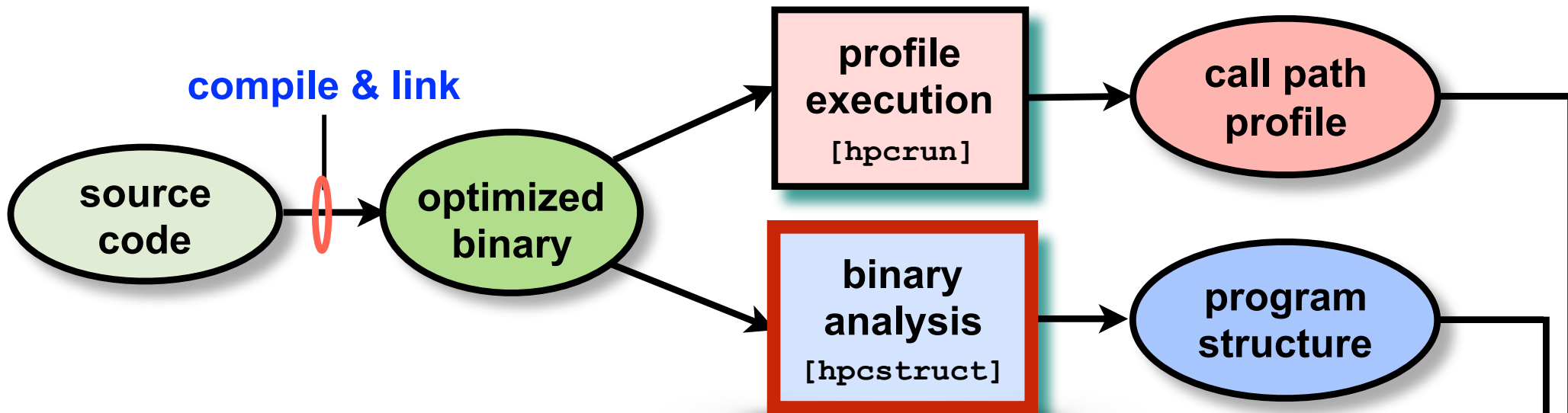
```

Calling Context View Callers View Flat View

Scope

Scope	CYCLES:Sum (I)	CYCLES:Sum (E)	KERNEL_BLOCKING	KERNEL_BLOCKING:SI
Experiment Aggregate Metrics	1.11e+11 100 %	1.11e+11 100 %	4.77e+09 100 %	4.77e+09 100 %
<thread root>	5.55e+10 50.1%		4.16e+09 87.1%	
↳ 943: <unknown procedure>	5.55e+10 50.1%	7.86e+05 0.0%	4.16e+09 87.1%	
<program root>	5.52e+10 49.9%		6.15e+08 12.9%	
↳ 500: main	5.52e+10 49.9%		6.15e+08 12.9%	
↳ loop at test2.c: 18	5.52e+10 49.9%		6.15e+08 12.9%	
↳ 27: __GI__libc_free	2.74e+07 0.0%		6.00e+08 12.6%	
↳ __munmap	2.74e+07 0.0%		6.00e+08 12.6%	
↳ entry_SYSCALL_64_fastpath	2.74e+07 0.0%		6.00e+08 12.6%	
↳ sys_munmap	2.74e+07 0.0%		6.00e+08 12.6%	
↳ do_munmap	2.74e+07 0.0%		6.00e+08 12.6%	
↳ unmap_region	2.74e+07 0.0%		6.00e+08 12.6%	
↳ unmap_vmas	2.74e+07 0.0%		6.00e+08 12.6%	
↳ unmap_single_vma	2.74e+07 0.0%		6.00e+08 12.6%	
↳ unmap_page_range	2.57e+07 0.0%		6.00e+08 12.6%	
↳ _cond_resched			6.00e+08 12.6%	
↳ fuse_ctl_cleanup [fuse]			6.00e+08 12.6%	6.00e+08 12.6%
<unknown file>: 0			6.00e+08 12.6%	6.00e+08 12.6%

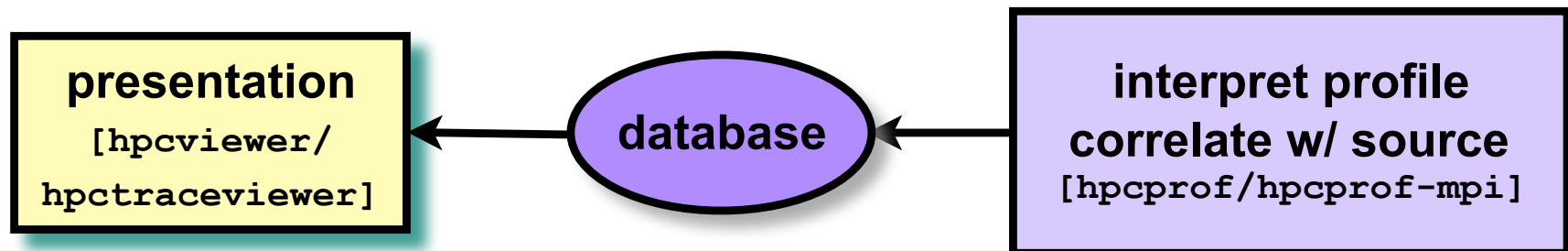
HPCToolkit Workflow



Ongoing work

- **Improving measurement**
- **Improving attribution to source**
- **Accelerating analysis with multithreaded parallelism**

Next Steps



Binary Analysis with hpcstruct

```

1185 /* compute the hourglass modes */
1186
1187 RAJA::forall<elem_exec_policy>(*domElemList, [&] (int i2) {
1188 #ifndef OMP_HACK
1189     Real_t hgfx[8], hgyf[8], hgfy[8];
1190 #endif
1191     Real_t coefficient;
1192
1193     Real_t hourgam0[4], hourgam1[4], hourgam2[4], hourgam3[4];
1194     Real_t houraam4[4], houraam5[4], houraam6[4], houraam7[4];

```

- function calls
- inlined functions
- inlined RAJA templates
- loops
- outlined OMP loop
- lambda function

Calling Context View | Callers View | Flat View

↑ ↓ f(x) CSV A+ A- |||

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)	OMP_IDL
Experiment Aggregate Metrics	2.26e+08 100 %	2.26e+08 100 %	2.43e-
<program root>	1.45e+08 63.9%		2.24e-
497: main	1.45e+08 63.9%	6.01e+03 0.0%	2.24e-
loop at luleshRAJA-parallel.cxx: 3526	1.44e+08 63.8%		2.23e-
3528: [] LagrangeLeapFrog(Domain*)	1.44e+08 63.8%		2.23e-
2715: [] LagrangeNodal(Domain*)	8.70e+07 38.5%		6.11e-
1554: [] CalcForceForNodes(Domain*)	8.30e+07 36.7%		4.02e-
1469: CalcVolumeForceForElems(Domain*)	8.25e+07 36.5%		3.32e-
1454: [] CalcHourglassControlForElems(Domain*, double*, double)	5.15e+07 22.8%		1.16e-
1399: [] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*)	3.10e+07 13.7%		8.97e-
1187: [] void RAJA::forall<RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>>(*domElemList, [&] (int i2) {	2.43e+07 10.8%		1.94e-
405: [] void RAJA::forall<RAJA::omp_parallel_for_exec, CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*)>(*domElemList, [&] (int i2) {	2.43e+07 10.8%		1.94e-
loop at forall_seq_any.hxx: 498	2.43e+07 10.8%		1.94e-
505: [] void RAJA::forall<CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*)>(*domElemList, [&] (int i2) {	2.43e+07 10.8%	1.00e+03 0.0%	1.94e-
89: outline forall_omp_any.hxx:89 (0x423620)	2.42e+07 10.7%	3.91e+04 0.0%	1.87e-
loop at forall_omp_any.hxx: 90	2.42e+07 10.7%	3.41e+04 0.0%	1.81e-
91: [] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*)	2.42e+07 10.7%	9.84e+06 4.3%	1.81e-
1300: [] CalcElemFBHourglassForce(double*, double*, double*, double*)	1.11e+07 4.9%	1.11e+07 4.9%	7.96e-
1260: [] CBRT(double)	3.27e+06 1.4%	2.00e+05 0.1%	2.23e-

Binary Analysis of GPU Code

- **Challenge: NVIDIA is very closed about their code**
 - has not shared any CUBIN documentation even through NDA
- **Awkward approach: reverse engineer CUBIN binaries**
- **Findings**
 - each GPU function is in its own text segment
 - all text segments begin at offset 0
 - result: all functions begin at 0 and overlap
- **Goal**
 - use Dyninst to analyze CUBINs in hpcstruct
- **Challenge**
 - Dyninst SymtabAPI and ParseAPI are not equipped to analyze overlapping functions and regions
- **Approach**
 - memory map CUBIN load module
 - relocate text segments, symbols, and line map in hpcstruct prior to analysis using Dyninst inside

Binary Analysis of CUBINs: Preliminary Results

```
32 __device__ __forceinline__ float g(const float a, const float b)
33 {
34     return a+b;
35 }
36
37 __device__ __forceinline__ float f(const float a, const float b)
38 {
39     return g(a,b);
40 }
41
42 /**
43  * CUDA Kernel Device code
44  *
45  * Computes the vector addition of A and B into C. The 3 vectors have the same
46  * number of elements numElements.
47  */
48 __global__ void
49 vectorAdd(const float *A, const float *B, float *C, int numElements)
50 {
51     int i = blockDim.x * blockIdx.x + threadIdx.x;
52
53     if (i < numElements)
54     {
55         C[i] = f(A[i],B[i]);
56     }
57 }
```

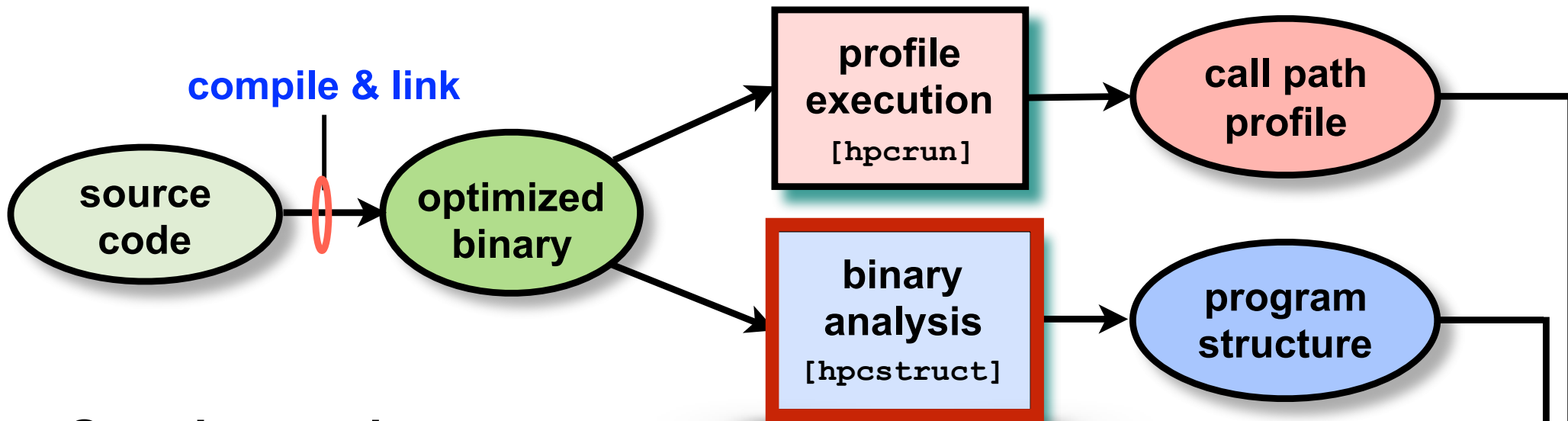
Limitation: CUBINs currently only have inlining information for unoptimized code

Next step: full analysis of heterogeneous binaries

— host binary with GPU load modules embedded as segments

```
<P i="32" n="vectorAdd(float const*, float const*, float*, int)" ln="_Z9vectorAddPKfS0_Pfi" l="0" v="{[0xc1e0-0xc1e1]}">
  <S i="33" l="0" v="{[0xc1e0-0xc60]}" />
  <S i="34" l="0" v="{[0xc1e0-0xc318]}" />
  <A i="35" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="&lt;inline&gt;" v="{}">
    <S i="36" l="39" v="{[0xc5e8-0xc5f0]}" />
    <S i="37" l="51" v="{[0xc318-0xc370]}" />
    <S i="38" l="53" v="{[0xc370-0xc398]}" />
    <S i="39" l="55" v="{[0xc398-0xc5e8] [0xc618-0xc710]}" />
    <S i="40" l="57" v="{[0xc710-0xc760]}" />
  </A>
  <A i="41" l="55" f="&lt;unknown&gt;" n="" v="{}">
    <A i="42" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="f(float, float)" v="{}">
      <S i="43" l="39" v="{[0xc608-0xc618]}" />
      <A i="44" l="39" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="" v="{}">
        <A i="45" l="34" f="/home/johnmc/llnl-visit/cuda-test/vectorAdd/vectorAdd.cu" n="g(float, float)" v="{}">
          <S i="46" l="34" v="{[0xc5f0-0xc608]}" />
        </A>
      </A>
    </A>
  </A>
</P>
```

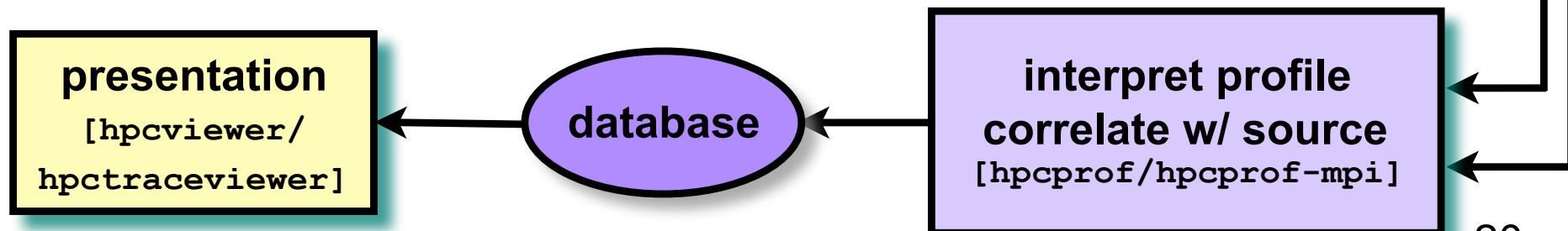
HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps



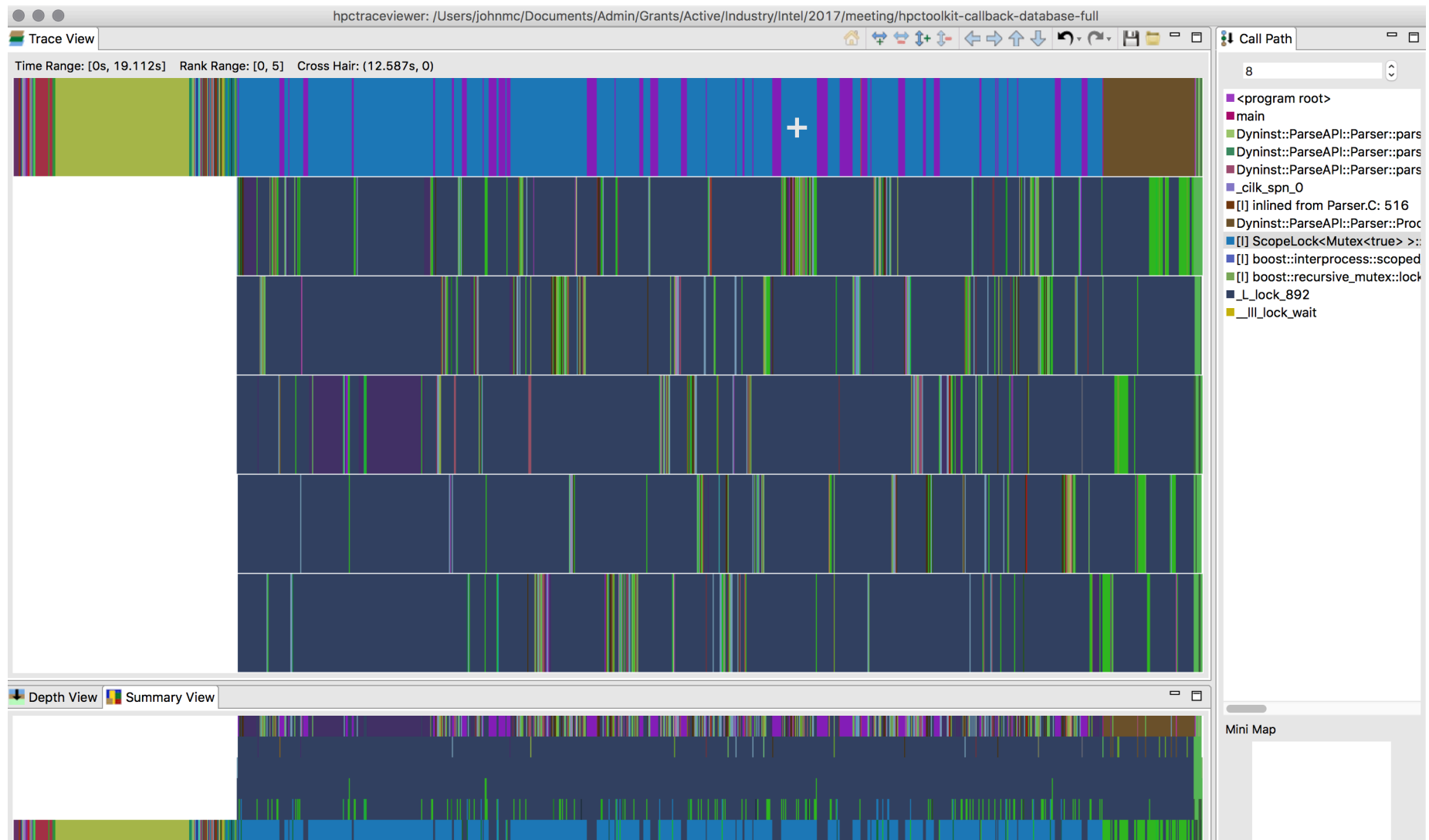
Parallel Binary Analysis: Why?

- **Static binaries on DOE Cray systems are big**
- **Binary analysis of large application binaries is too slow**
 - **NWchem binary from Cray platform at NERSC (Edison)**
157M (104M text)
 - **serial hpcstruct based on Dyninst v9.3.2**
Intel Westmere @ 2.8GHz: 10 minutes
KNL @ 1.4GHz: 28 minutes
- **Tests user patience and is an impediment to tool use**

Parallelizing hpcstruct: Two Approaches

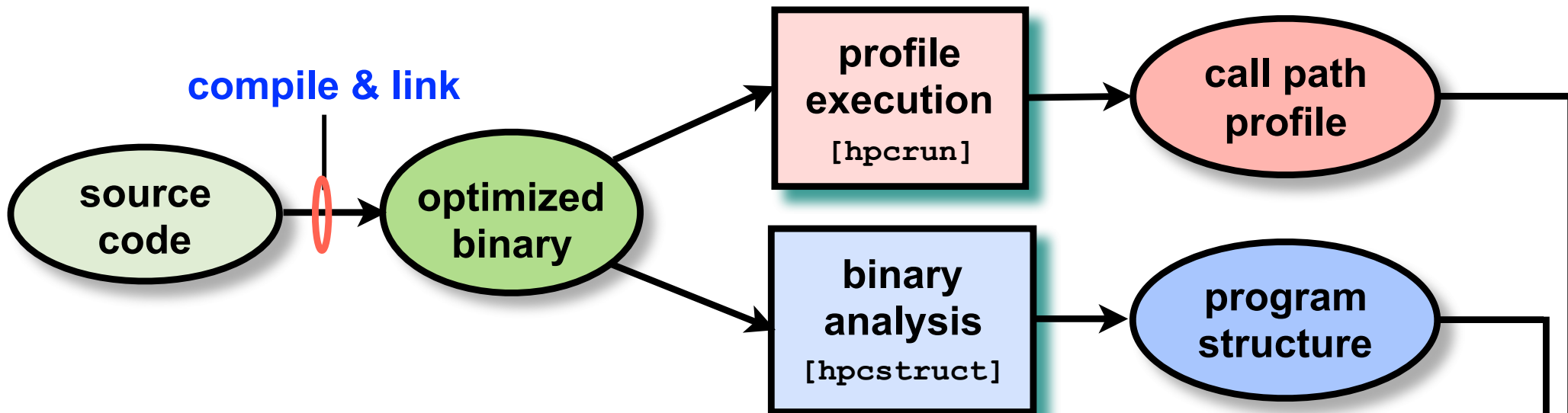
- **Light**
 - approach
 - parse the binary with Dyninst's ParseAPI, SymtabAPI
 - parallelize hpcstruct's binary analysis, which runs atop Dyninst APIs
- **Full**
 - approach
 - parallelize parsing of the binary with Dyninst
 - Dyninst supports a callback when a procedure parse is finalized
 - register callback to perform hpcstruct analysis at that time
 - potential benefits
 - opportunity for speedup as much as number of procedures

Parallel Binary Parsing with Dyninst



Added parallelism using CilkPlus constructs

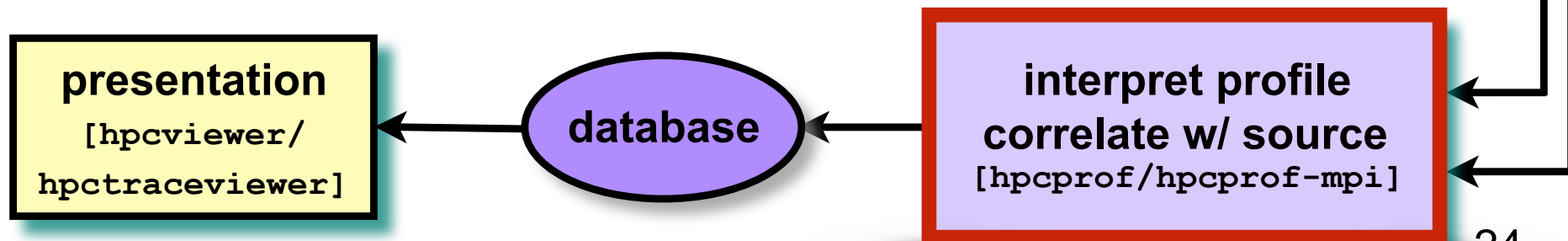
HPCToolkit Workflow



Ongoing work

- **Improving measurement**
- **Improving attribution to source**
- **Accelerating analysis with multithreaded parallelism**

Next Steps

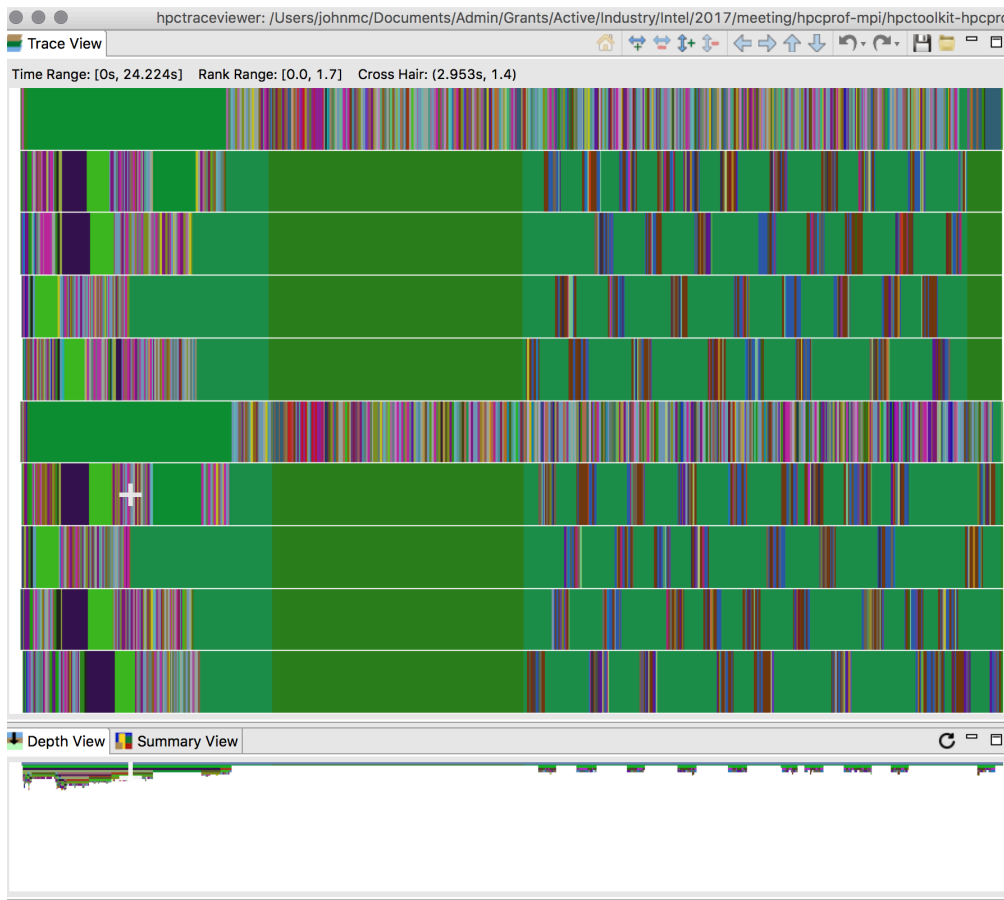


Accelerating Data Analysis

- **Problem**
 - need massive parallelism to analyze large-scale measurements
 - MPI-everywhere is not the best way to use Xeon Phi
- **Approach**
 - add thread-level parallelism to hpcprof-mpi
 - threads collaboratively process multiple performance data files

hpcprof-mpi with Thread-level Parallelism

- **Add thread-level parallelism with OpenMP**
 - program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus



MPI thread (OpenMP master)

OpenMP worker threads

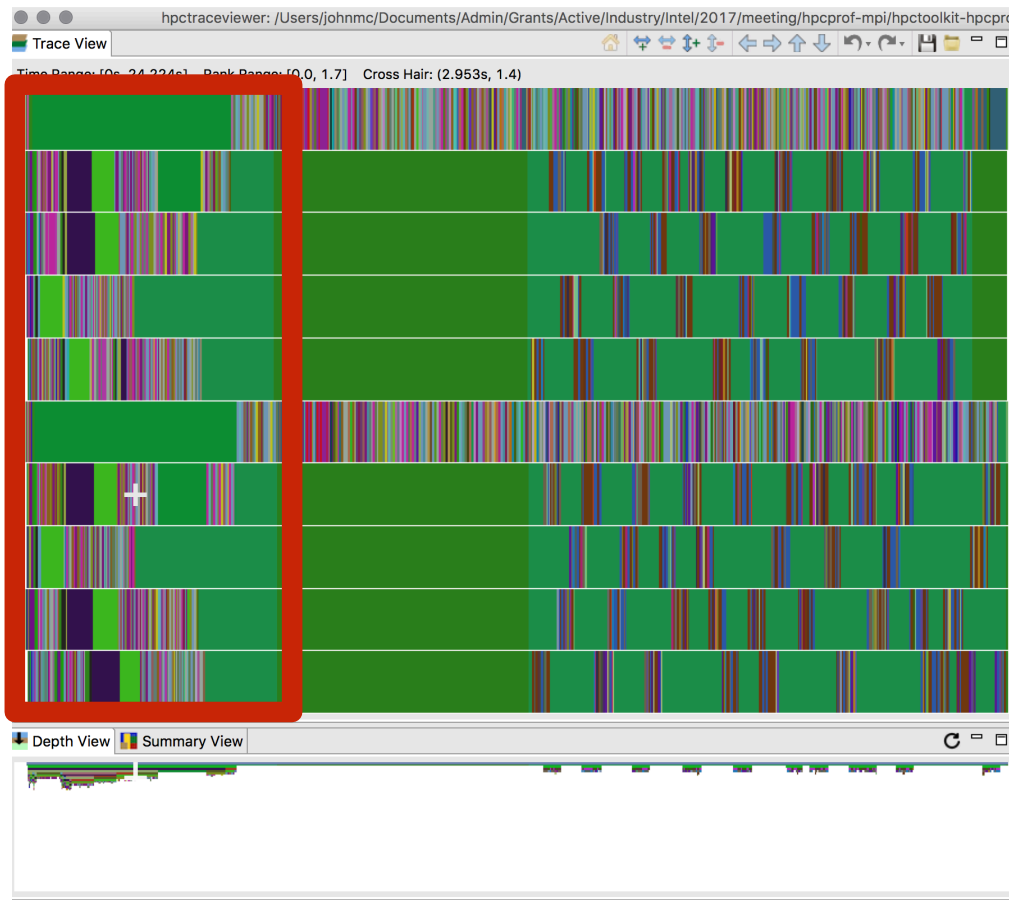
MPI thread (OpenMP master)

OpenMP worker threads

hpcprof-mpi with Thread-level Parallelism

- Add thread-level parallelism with OpenMP
 - program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus

merge profiles using a parallel reduction tree

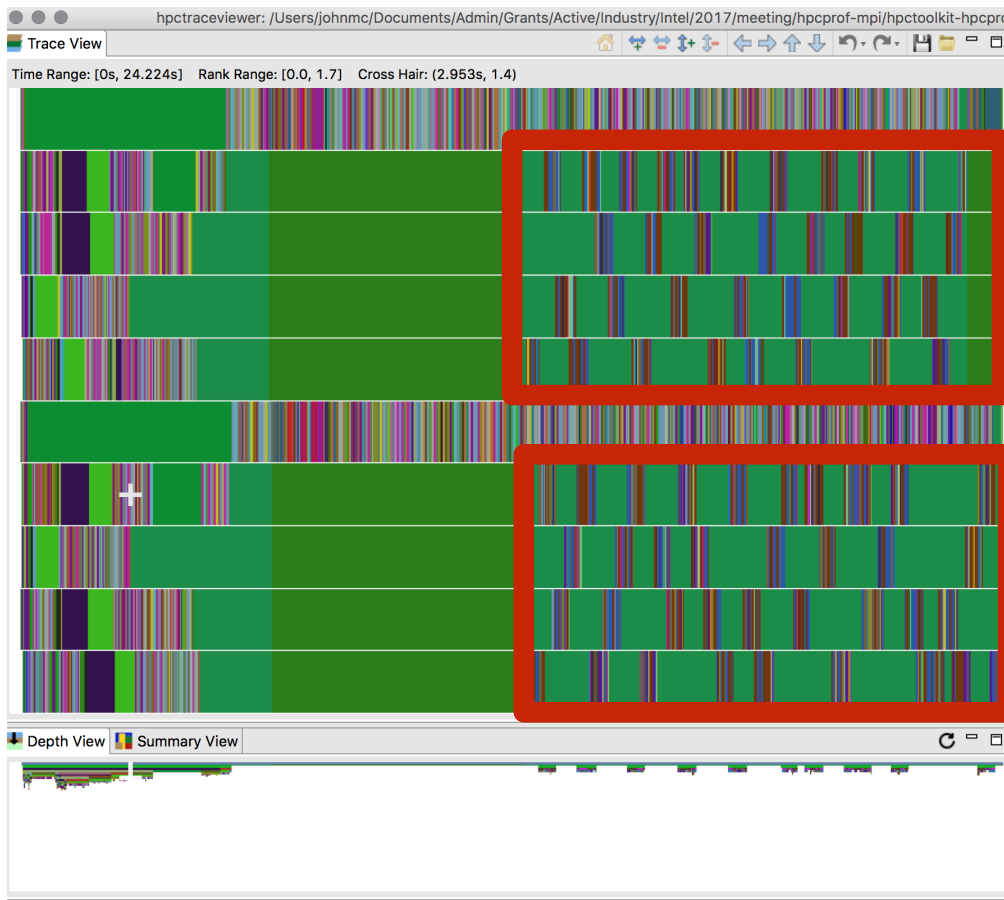


```
if (upper == lower) { // singleton profile
    Prof::CallPath::Profile* p = readSingle(profileFiles, groupMap, lower, rFlags);
    return p;
} else { // multiple profiles
    Prof::CallPath::Profile* left = 0;
    Prof::CallPath::Profile* right = 0;
    uint mid = lower + (upper - lower)/2;
    #pragma omp task shared(left, profileFiles, groupMap) \
    firstprivate(mergeTy, rFlags, mrgFlags, lower, mid)
    {
        left = readSet(profileFiles, groupMap, mergeTy, rFlags, mrgFlags, lower, mid);
    }
    right = readSet(profileFiles, groupMap, mergeTy, rFlags, mrgFlags, mid + 1, upper);
}
#pragma omp taskwait
left->merge(*right, mergeTy, mrgFlags);
```

```
Prof::CallPath::Profile* prof = 0;
#pragma omp parallel shared(prof)
{
    #pragma omp master
    {
        prof = readSet(profileFiles, groupMap, mergeTy,
            rFlags, mrgFlags, 0, profileFiles.size() - 1);
    }
}
#endif
```

hpcprof-mpi with Thread-level Parallelism

- Add thread-level parallelism with OpenMP
 - program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus



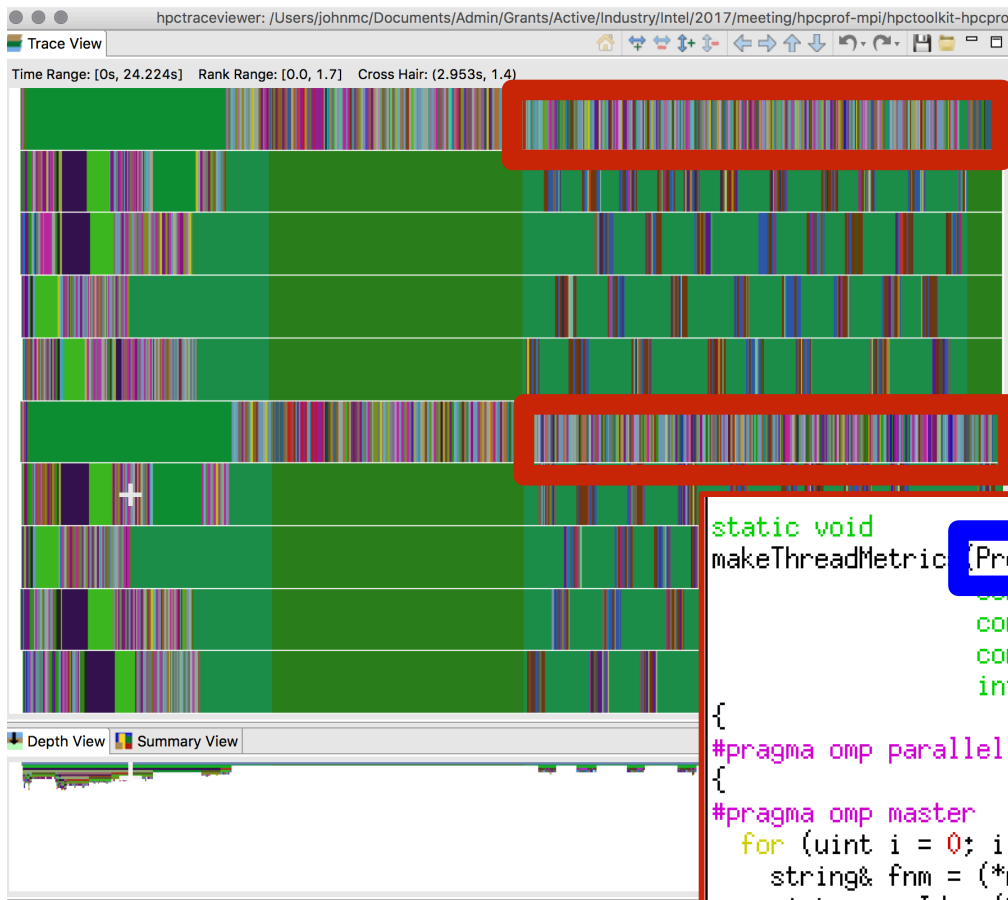
update traces asynchronously

```
void
Profile::merge_fixTrace(const CCT::MergeEffectList* mrgEffects)
{
    string *file = new string(m_traceFileName);
    #pragma omp task firstprivate(file)
    {
        doFixTrace(*file, mrgEffects);
        delete mrgEffects;
        delete file;
    }
}
```

```
static void
makeThreadMetrics(Prof::CallPath::Profile& profGbl,
                  const Analysis::Args& args,
                  const Analysis::Util::NormalizeProfileArgs_t& nArgs,
                  const vector<uint>& groupIdToGroupSizeMap,
                  int myRank, int numRanks, int rootRank)
{
    #pragma omp parallel
    {
        #pragma omp master
        for (uint i = 0; i < nArgs.paths->size(); ++i) {
            string& fnm = (*nArgs.paths)[i];
            uint groupId = (*nArgs.groupMap)[i];
            makeThreadMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax, myRank);
        }
    }
}
```

hpcprof-mpi with Thread-level Parallelism

- Add thread-level parallelism with OpenMP
 - program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus

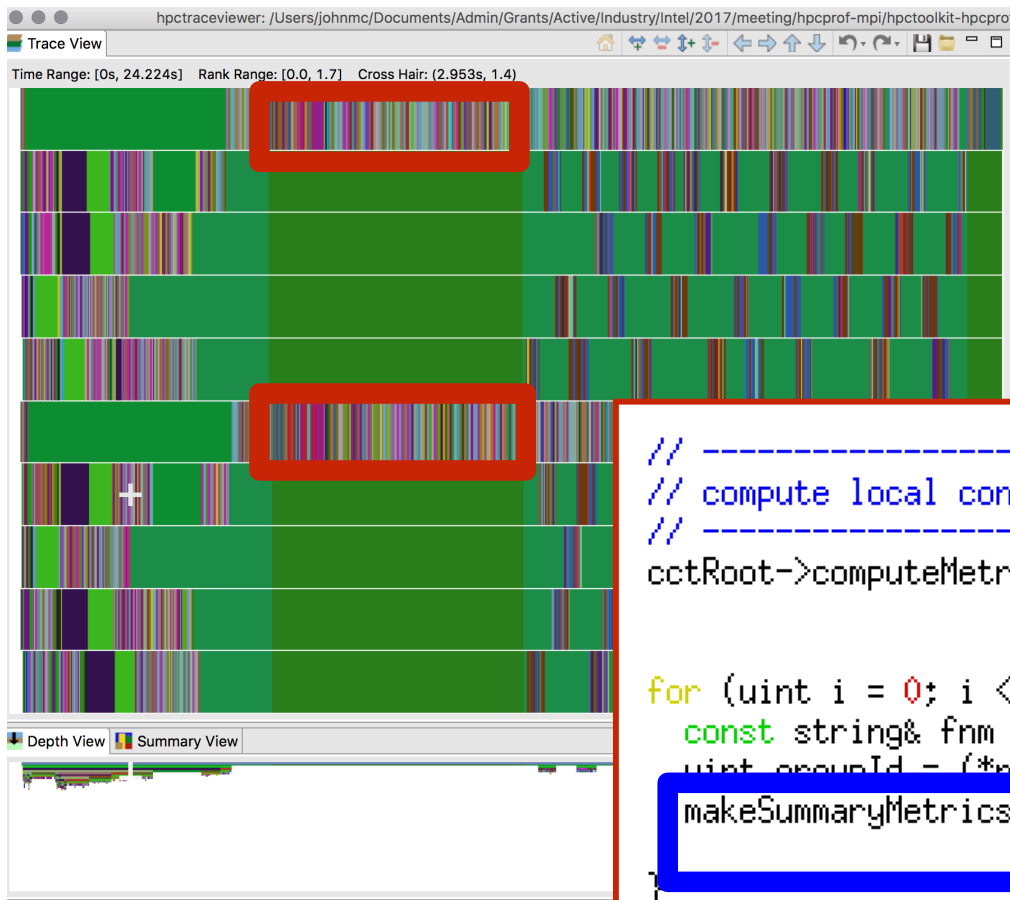


compute thread metrics locally
using a global variable

```
static void  
makeThreadMetric(Prof::CallPath::Profile& profGbl,  
                const Analysis::Util::NormalizeProfileArgs_t& nArgs,  
                const vector<uint>& groupIdToGroupSizeMap,  
                int myRank, int numRanks, int rootRank)  
{  
    #pragma omp parallel  
    {  
        #pragma omp master  
        for (uint i = 0; i < nArgs.paths->size(); ++i) {  
            string& fnm = (*nArgs.paths)[i];  
            makeThreadMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax, myRank);  
        }  
    }  
}
```

hpcprof-mpi with Thread-level Parallelism

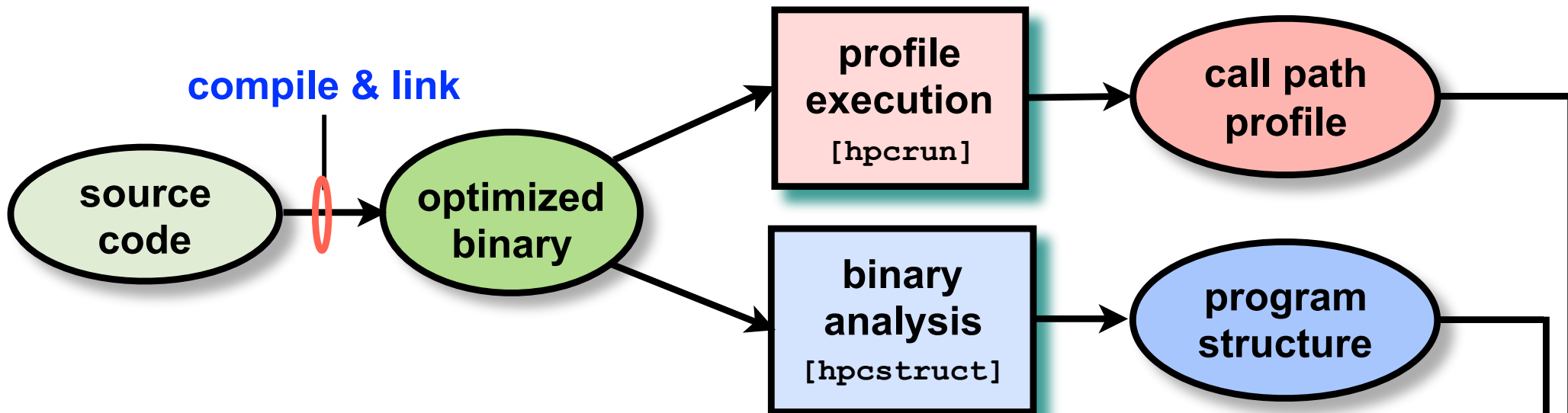
- Add thread-level parallelism with OpenMP
 - program structure where the opportunity for an asynchronous task appears deep on call chains is not well suited for CilkPlus



accumulate metric values locally
into a global variable

```
// -----  
// compute local contribution summary metrics (accumulate function)  
// -----  
cctRoot->computeMetricsIncr(mMgrGbl, mDrvdBeg, mDrvdEnd,  
                           Prof::Metric::AExprIncr::FnInit);  
  
for (uint i = 0; i < nArgs.paths->size(); ++i) {  
    const string& fnm = (*nArgs.paths)[i];  
    uint groupId = (*nArgs.groupMap)[i];  
    makeSummaryMetrics_Lcl(profGbl, fnm, args, groupId, nArgs.groupMax,  
                          groupIdToGroupMetricsMap, myRank);  
}
```

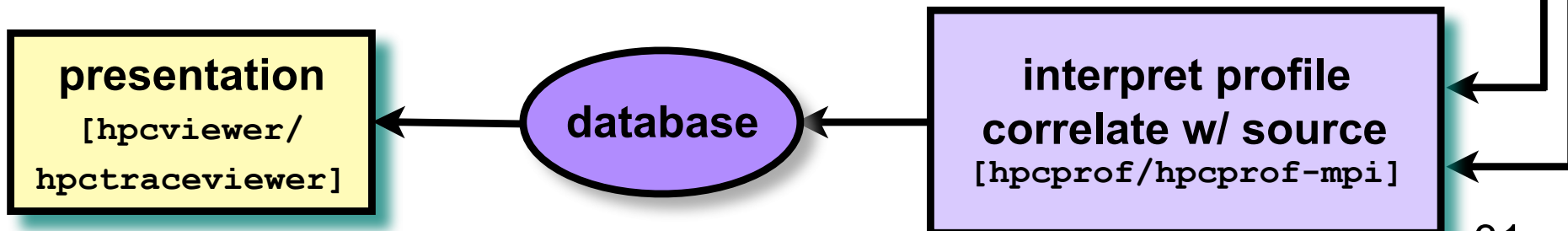
HPCToolkit Workflow



Ongoing work

- Improving measurement
- Improving attribution to source
- Accelerating analysis with multithreaded parallelism

Next Steps



Next Steps

- Integrate hpcstruct and perf_events improvements into trunk
- Data-centric measurement with perf_events
- Continue work with Wisconsin on parallelization of hpcstruct
- Work with OpenMP community to finalize OMPT and OpenMP 5
 - test and validate new LLVM OMPT host-side implementation
 - integrate OMPT support for libomptarget into LLVM trunk
- Finish OpenMP 5 and CUDA support in HPCToolkit
- Improve support for measurement and analysis at scale
 - reduce file counts
 - improve multithreaded parallel analysis
- Explore GUI enhancements to improve developer workflows
- Add support for top-down models for architecture analysis