HPCToolkit: New Capabilities, Ongoing Work, & Challenges Ahead

John Mellor-Crummey, Nathan Tallent, Xu Liu

Laksono Adhianto, Michael Fagan, Mark Krentel

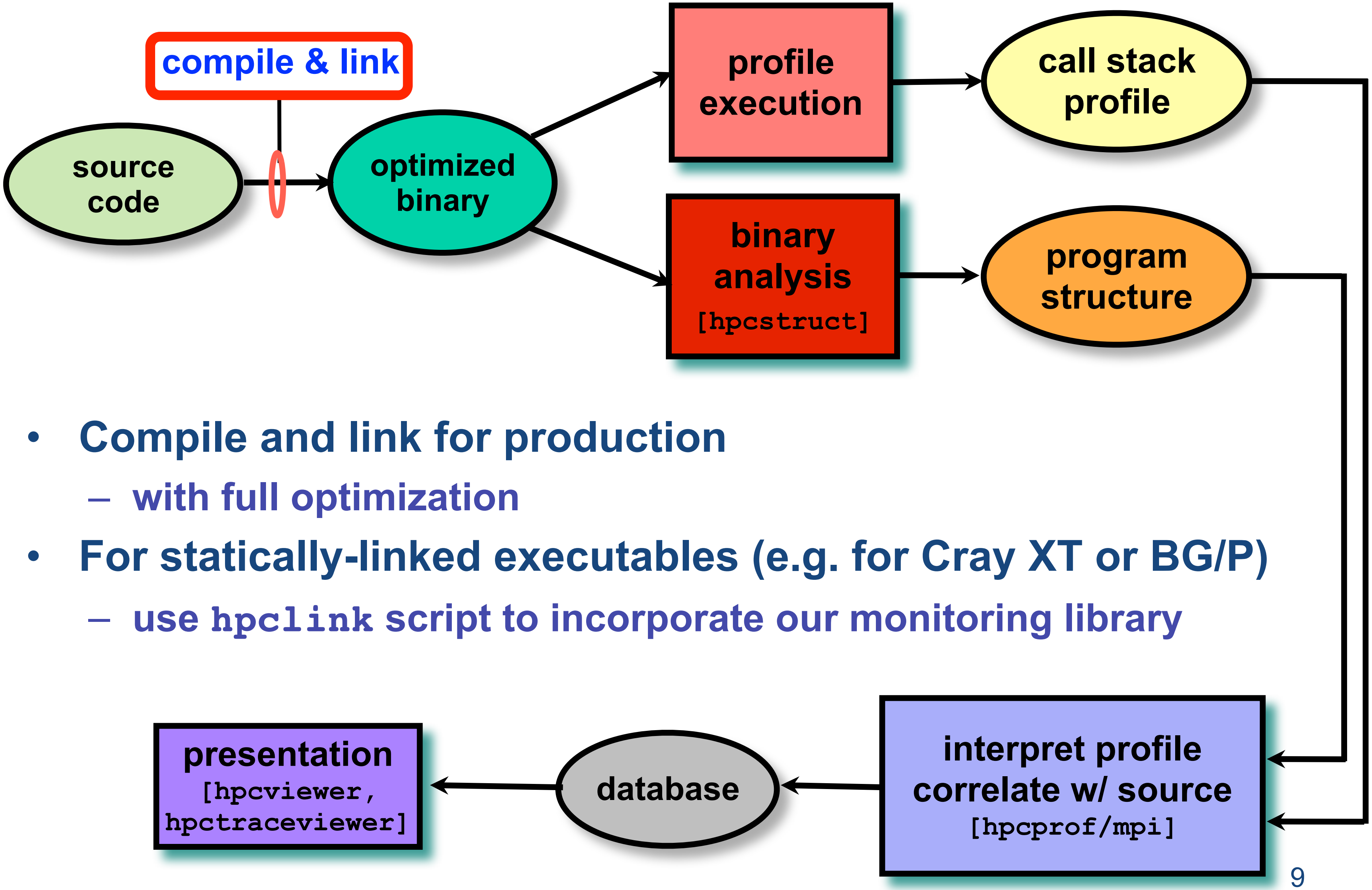
Dept. of Computer Science, Rice University

CScADS Performance Tools for Extreme Scale Computing • August 2011

RICE

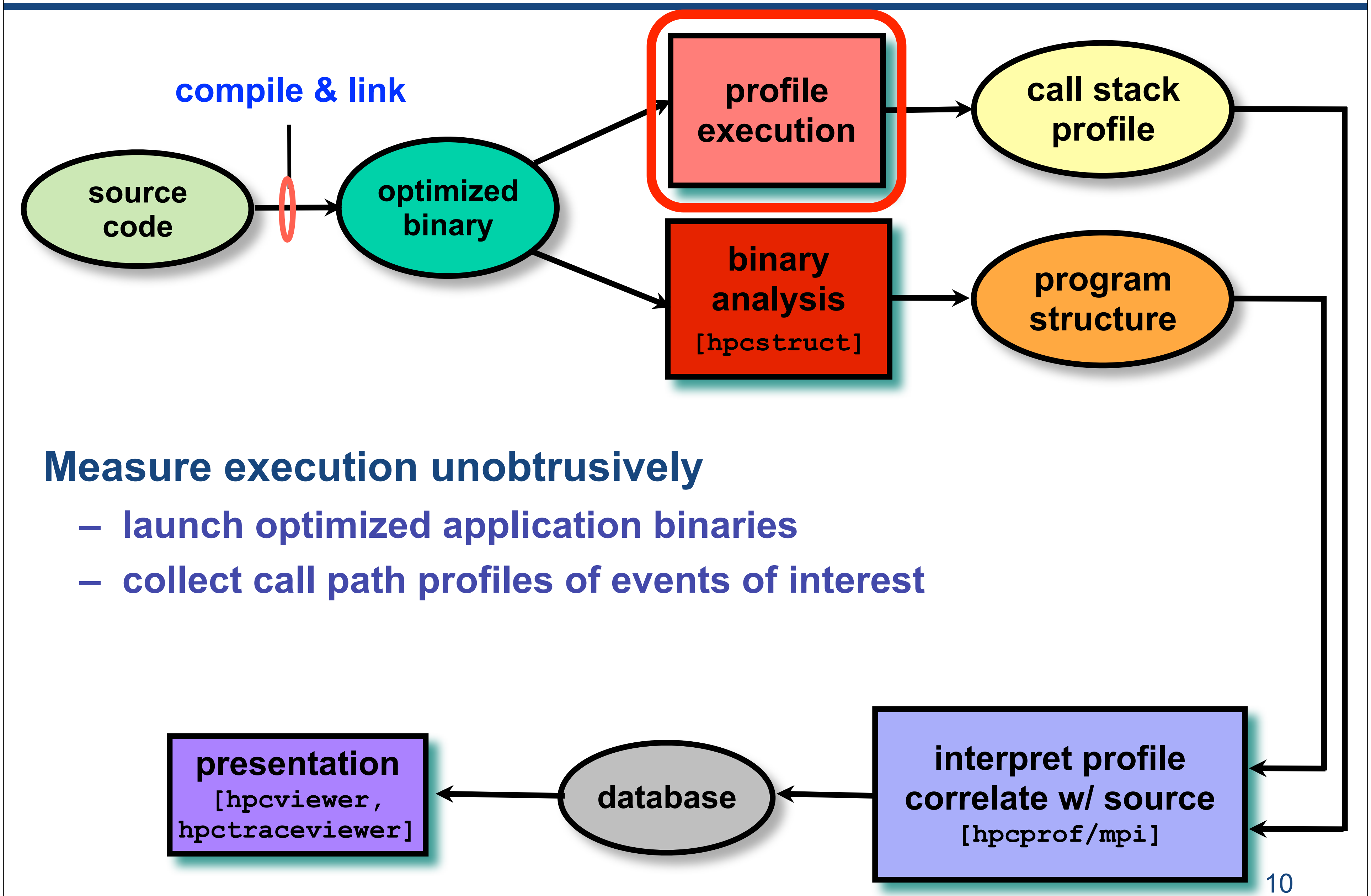
hpctoolkit.org

HPCToolkit Performance Tools



- Compile and link for production
  - with full optimization
- For statically-linked executables (e.g. for Cray XT or BG/P)
  - use `hpclink` script to incorporate our monitoring library

HPCToolkit Performance Tools



compile & link

source code

optimized binary

profile execution

binary analysis `[hpcstruct]`

call stack profile

program structure

interpret profile correlate w/ source `[hpcprof/mpi]`

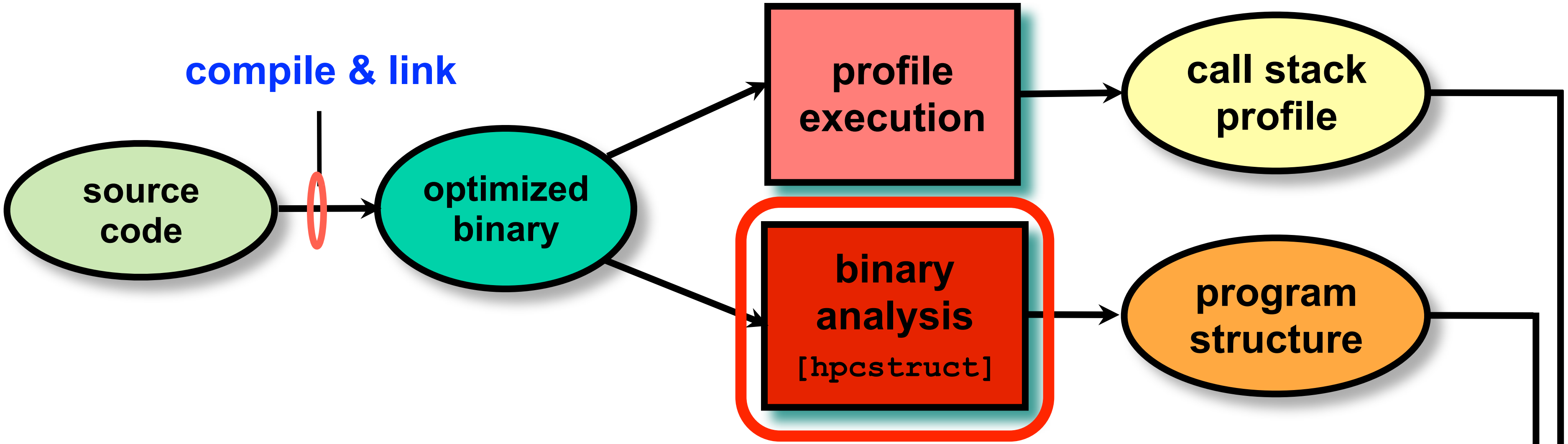
database

presentation `[hpcviewer, hpctraceviewer]`

Measure execution unobtrusively

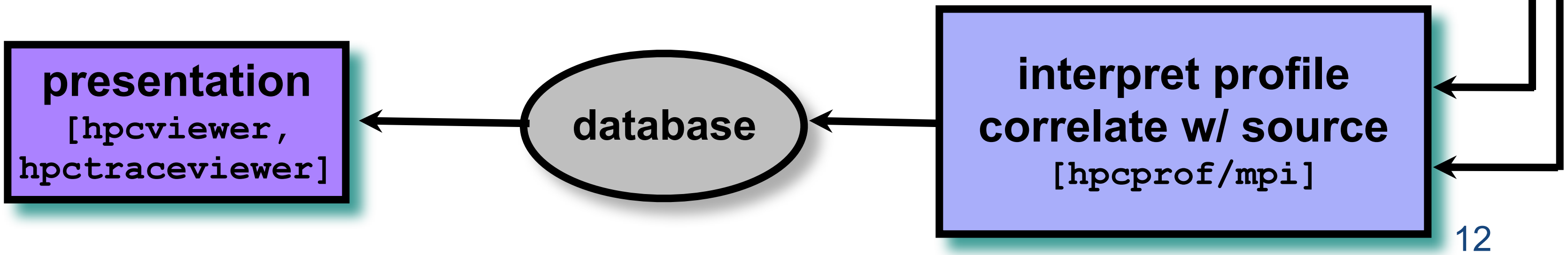
- launch optimized application binaries
- collect call path profiles of events of interest

HPCToolkit Performance Tools

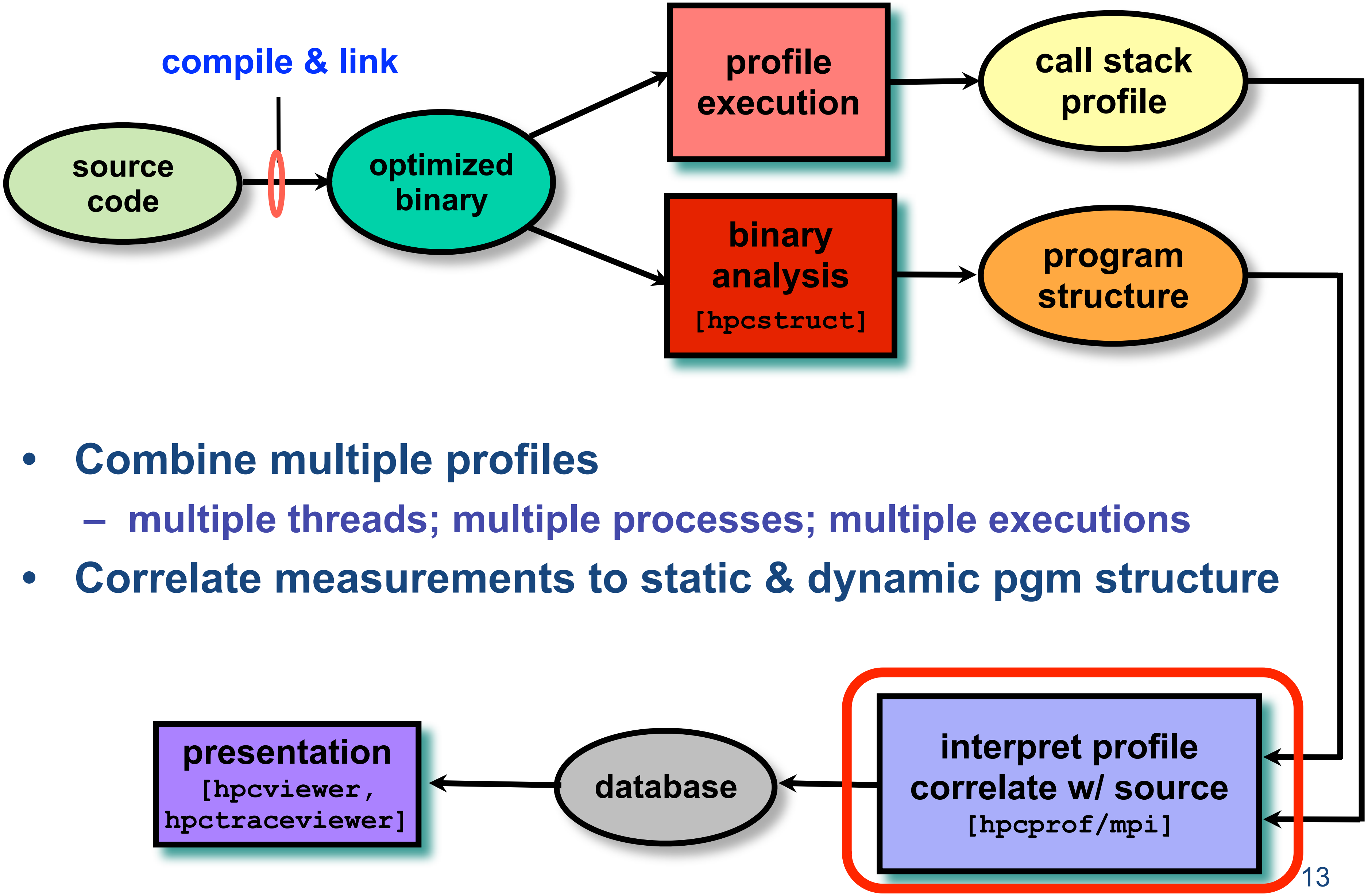


Analyze binary to recover program structure

- analyze machine code, line map, and debugging information
- extract loop nesting information and identify inlined procedures
- map transformed loops and procedures back to source

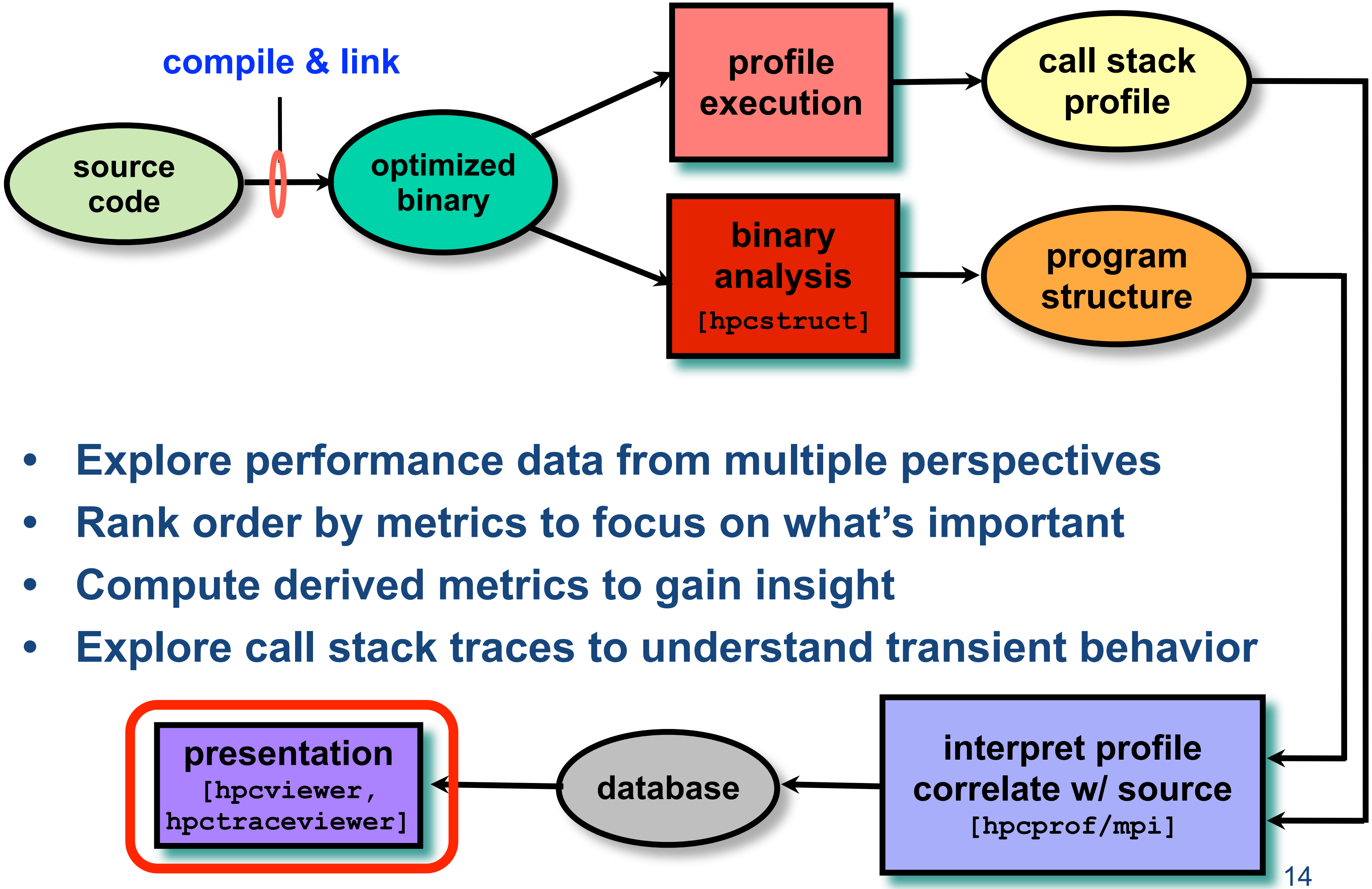


HPCToolkit Performance Tools



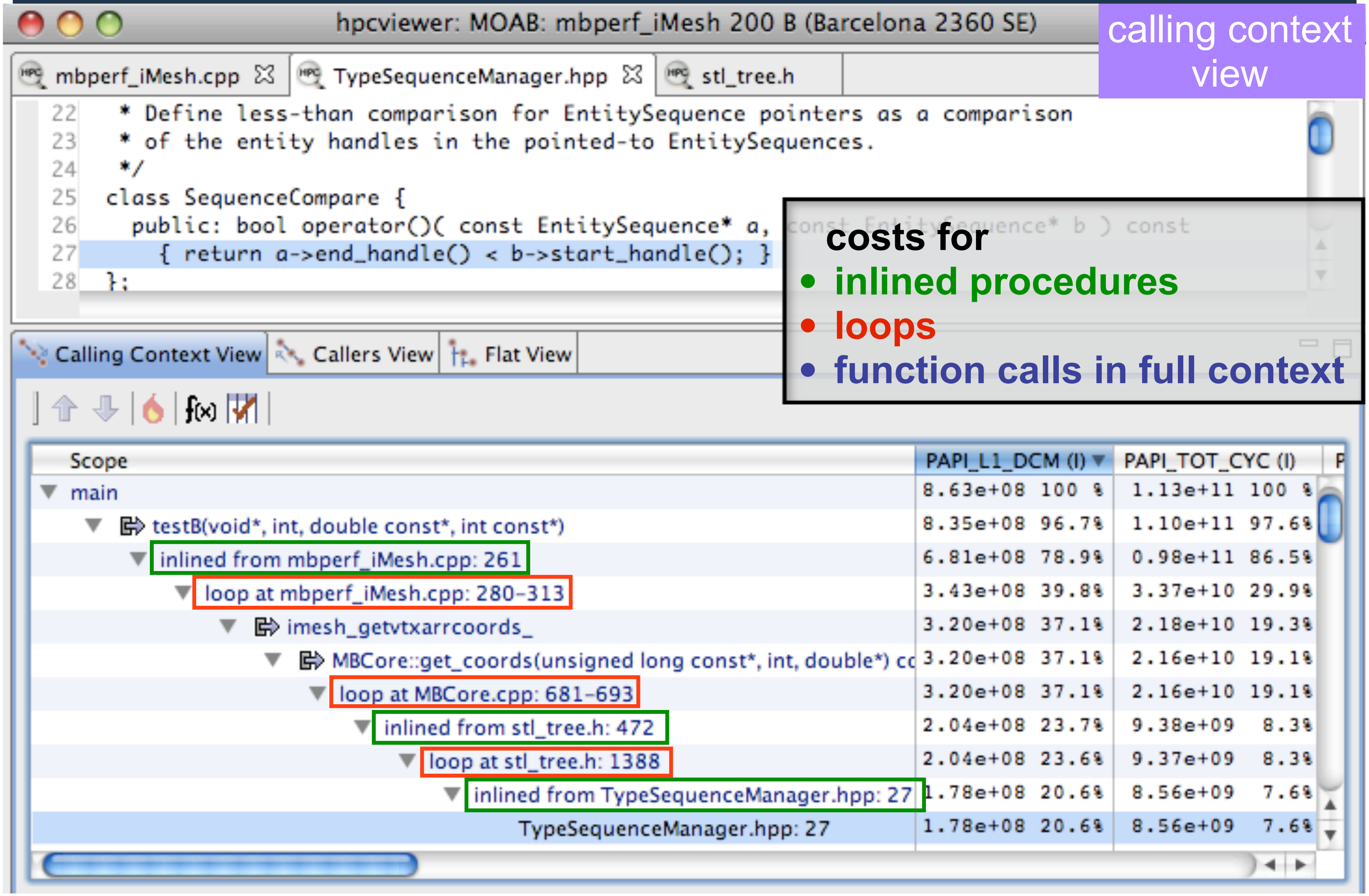
- Combine multiple profiles
  - multiple threads; multiple processes; multiple executions
- Correlate measurements to static & dynamic pgm structure

HPCToolkit Performance Tools



- Explore performance data from multiple perspectives
- Rank order by metrics to focus on what’s important
- Compute derived metrics to gain insight
- Explore call stack traces to understand transient behavior

Attribution to Static + Dynamic 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, const EntitySequence* b ) const
    { 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%	0.98e+11 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*) cc	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%

calling context view

costs for

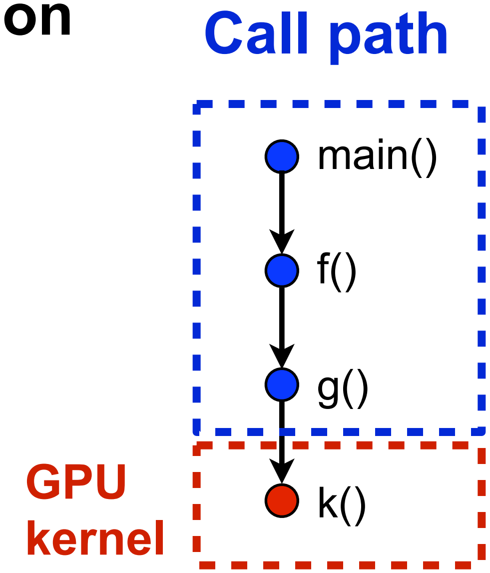
- **inlined procedures**
- **loops**
- **function calls in full context**

Outline

- GPU profiling
- Detecting memory leaks
- Call path tracing
- More work on scaling
- Applying HPCToolkit to FY2011 “Joule Metric” applications
- Data-centric measurement and analysis
- Static analysis of memory access patterns
- Some challenges ahead

Call Path Profiling for GPU-based Systems

- **Why call path profiling? Flat context often isn't enough**
  - same operations used differently in multiple places
- **Many apps experimenting w/ GPU acceleration**
  - call path of GPU kernel is separated in space
    - host stack + GPU kernel
  - call path of GPU kernel is separated in time
    - kernels may be executed asynchronously
  - GPUs contain interesting hardware performance counters

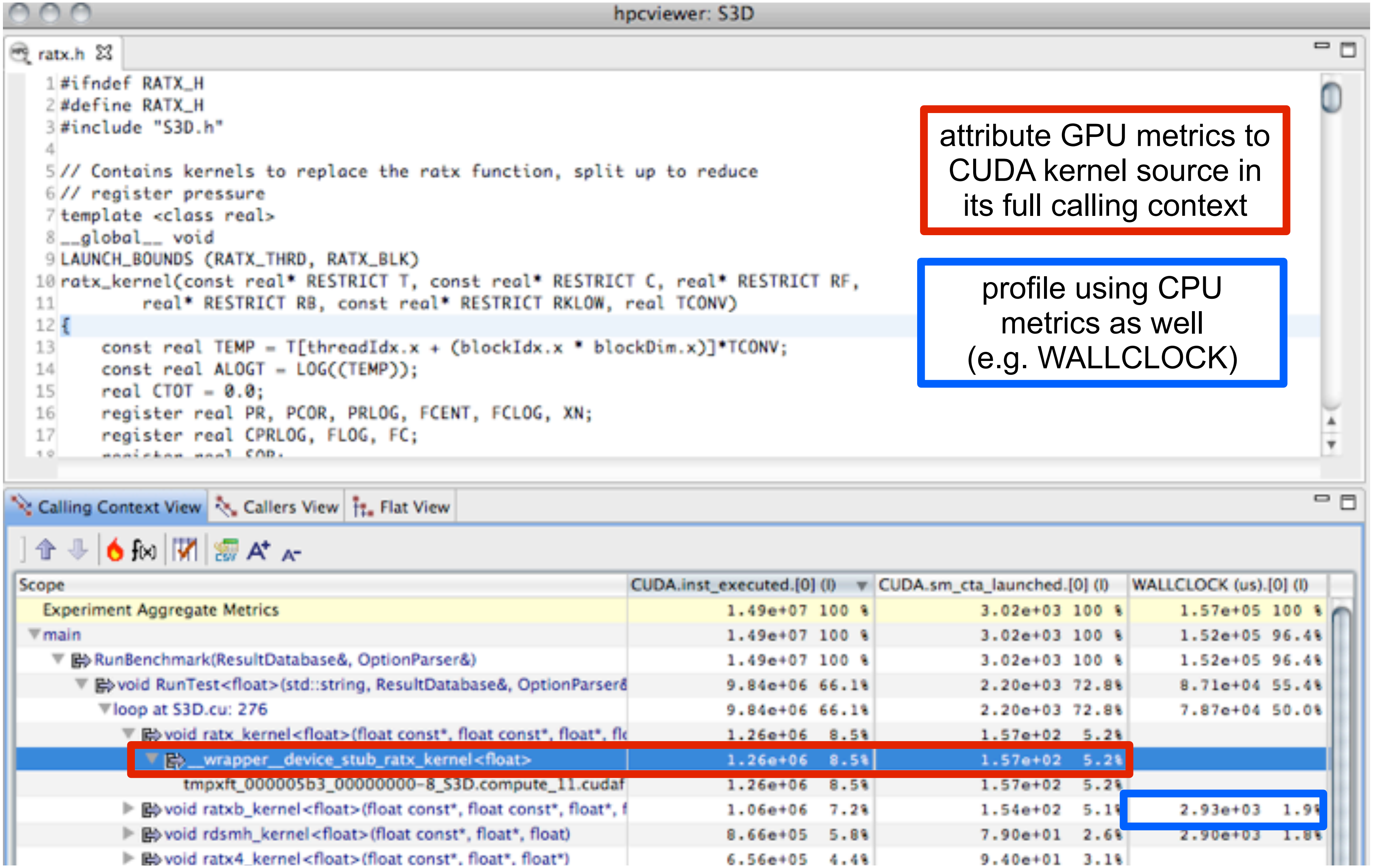


Adapt HPToolkit profiling to CUDA-accelerated executions

Prototype of GPU-Enabled Profiler

- **Use PAPI + NVIDIA's CUPTI profiling interface**
- **On entering a CUDA "kernel launch"**
  - cudaThreadSynchronize() // wait for GPU to finish
  - start GPU performance counters
- **On exiting a CUDA "kernel launch"**
  - cudaThreadSynchronize() // wait for GPU to finish
  - stop GPU performance counters
  - gather calling context of kernel (synchronously)
  - associate GPU performance with kernel (in context)
- **Limitations**
  - counters are not kernel-specific (hardware limitation)
    - must either serialize kernels or work with throughput metrics
  - cudaThreadSynchronize() on entry/exit
    - destroys CPU/GPU overlap
    - shouldn't affect GPU measurements of individual kernels
  - kernel is finest granularity of GPU counter metrics
    - no line-level attribution within GPU code

GPU-Aware Call Path Profiles



```
#ifndef RATX_H
#define RATX_H
#include "S3D.h"

// Contains kernels to replace the ratx function, split up to reduce
// register pressure
template <class real>
__global__ void
LAUNCH_BOUNDS (RATX_THRD, RATX_BLK)
ratx_kernel(const real* RESTRICT T, const real* RESTRICT C, real* RESTRICT RF,
        real* RESTRICT RB, const real* RESTRICT RKLOW, real TCONV)
{
    const real TEMP = T[threadIdx.x + (blockIdx.x * blockDim.x)]*TCONV;
    const real ALOGT = LOG((TEMP));
    real CTOT = 0.0;
    register real PR, PCOR, PRLOG, FCENT, FCLOG, XN;
    register real CPRLOG, FLOG, FC;
```

Scope	CUDA.inst_executed.[0] (I)	CUDA.sm_cta_launched.[0] (I)	WALLCLOCK (us).[0] (I)
Experiment Aggregate Metrics	1.49e+07 100 %	3.02e+03 100 %	1.57e+05 100 %
main	1.49e+07 100 %	3.02e+03 100 %	1.52e+05 96.4%
RunBenchmark(ResultDatabase&, OptionParser&)	1.49e+07 100 %	3.02e+03 100 %	1.52e+05 96.4%
void RunTest<float>(std::string, ResultDatabase&, OptionParser&	9.84e+06 66.1%	2.20e+03 72.8%	8.71e+04 55.4%
loop at S3D.cu: 276	9.84e+06 66.1%	2.20e+03 72.8%	7.87e+04 50.0%
void ratx_kernel<float>(float const*, float const*, float*, flo	1.26e+06 8.5%	1.57e+02 5.2%	
__wrapper__device_stub_ratx_kernel<float>	1.26e+06 8.5%	1.57e+02 5.2%	
tmpxft_000005b3_00000000-8_S3D.compute_11.cudaf	1.26e+06 8.5%	1.57e+02 5.2%	
void ratxb_kernel<float>(float const*, float const*, float*, f	1.06e+06 7.2%	1.54e+02 5.1%	2.93e+03 1.9%
void rdsmh_kernel<float>(float const*, float*, float)	8.66e+05 5.8%	7.90e+01 2.6%	2.90e+03 1.8%
void ratx4_kernel<float>(float const*, float*, float*)	6.56e+05 4.4%	9.40e+01 3.1%	

attribute GPU metrics to CUDA kernel source in its full calling context

profile using CPU metrics as well (e.g. WALLCLOCK)

GPU Profiling Support: What Next?

- Look at overall CPU and GPU utilization
- Quantify overlap of
  - CPU execution
  - data movement to accelerator
  - GPU execution
- Look at gap between potential vs. realized performance
  - compute derived metrics to understand GPU performance
    - degree of multithreaded parallelism utilized
    - fraction of compute capability utilized (instructions per cycle)
    - fraction of available memory bandwidth consumed
    - fraction of memory accesses that hit in cache
    - balance of reads and writes across cache and memory slices
    - fraction of divergent branches
    - ...

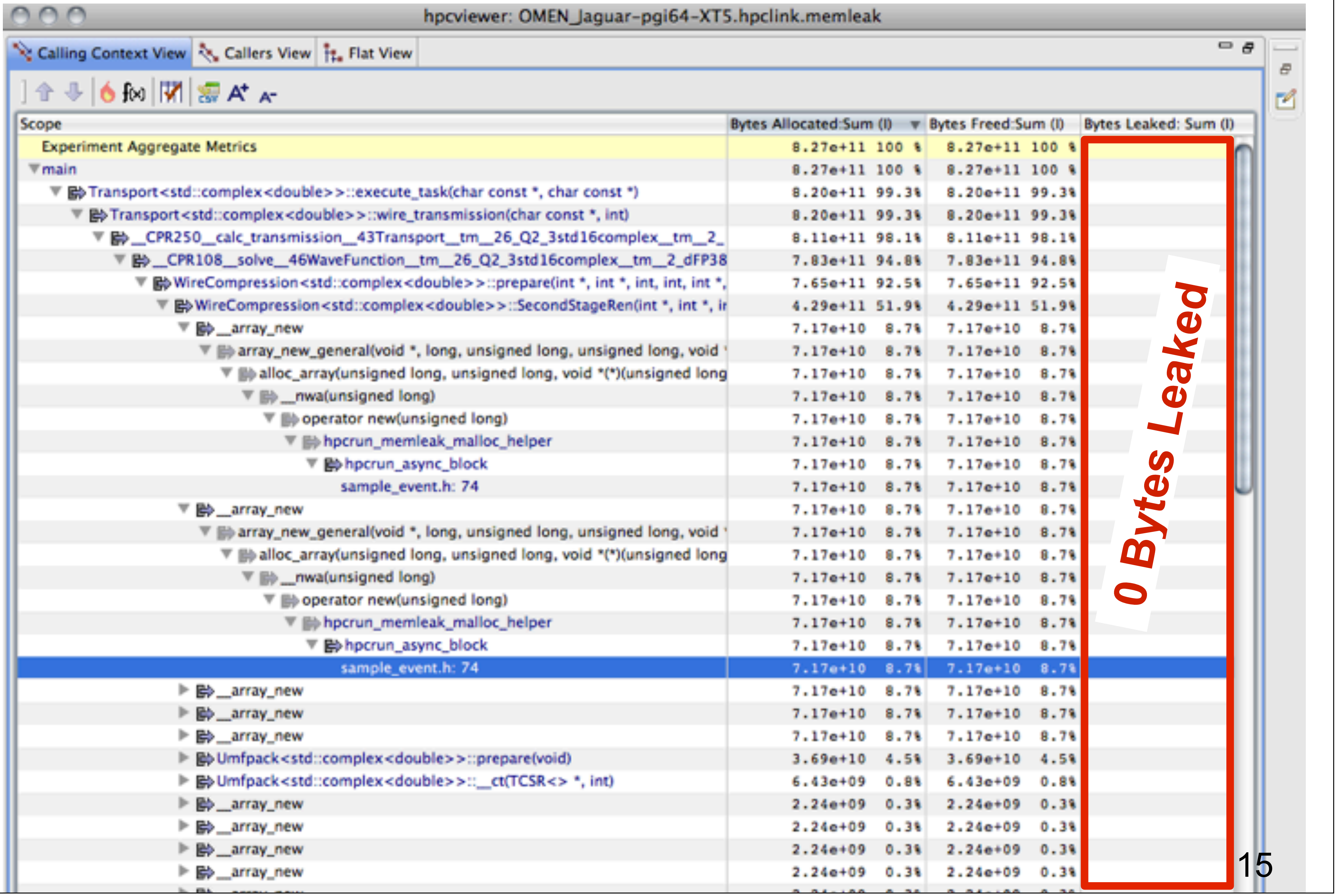
Outline

- GPU profiling
- Detecting memory leaks
- Call path tracing
- More work on scaling
- Applying HPCToolkit to FY2011 “Joule Metric” applications
- Data-centric measurement and analysis
- Static analysis of memory access patterns
- Some challenges ahead

Correctness Tool: Memory Leak Detector

- **Intercept malloc() and free() (and variants)**
  - malloc: gather calling context (synchronously)
  - free: note that the corresponding allocation point is freed
- **Storing metadata: in-band vs. out-of-band**
  - associate malloc calling context with allocated block
  - out-of-band: process-wide splay tree (with locks)
    - advantage: easy to implement
    - disadvantage: overhead
  - in-band: add header or footer to memory block [our approach]
    - prefer headers: constant time lookup, no synchronization
    - use footers as needed
      - advantage: avoids disturbing specified memory block alignment
      - disadvantage: synchronized lookup
- **Can trade monitoring overhead for incompleteness**
  - monitor every *n*th malloc; monitor all frees
- **Detail: getcontext() is surprisingly expensive; write our own**

Confirming OMEN Has No Leaks



Scope	Bytes Allocated:Sum (I)	Bytes Freed:Sum (I)	Bytes Leaked: Sum (I)
Experiment Aggregate Metrics	8.27e+11 100 %	8.27e+11 100 %	
main	8.27e+11 100 %	8.27e+11 100 %	
Transport<std::complex<double>>::execute_task(char const *, char const *)	8.20e+11 99.3%	8.20e+11 99.3%	
Transport<std::complex<double>>::wire_transmission(char const *, int)	8.20e+11 99.3%	8.20e+11 99.3%	
__CPR250__calc_transmission__43Transport__tm__26_Q2_3std16complex__tm__2_	8.11e+11 98.1%	8.11e+11 98.1%	
__CPR108__solve__46WaveFunction__tm__26_Q2_3std16complex__tm__2_dFP38	7.83e+11 94.8%	7.83e+11 94.8%	
WireCompression<std::complex<double>>::prepare(int *, int *, int, int, int *,	7.65e+11 92.5%	7.65e+11 92.5%	
WireCompression<std::complex<double>>::SecondStageRen(int *, int *, ir	4.29e+11 51.9%	4.29e+11 51.9%	
__array_new	7.17e+10 8.7%	7.17e+10 8.7%	
array_new_general(void *, long, unsigned long, unsigned long, void	7.17e+10 8.7%	7.17e+10 8.7%	
alloc_array(unsigned long, unsigned long, void *(*)(unsigned long	7.17e+10 8.7%	7.17e+10 8.7%	
__nwa(unsigned long)	7.17e+10 8.7%	7.17e+10 8.7%	
operator new(unsigned long)	7.17e+10 8.7%	7.17e+10 8.7%	
hpcrun_memleak_malloc_helper	7.17e+10 8.7%	7.17e+10 8.7%	
hpcrun_async_block	7.17e+10 8.7%	7.17e+10 8.7%	
sample_event.h: 74	7.17e+10 8.7%	7.17e+10 8.7%	
__array_new	7.17e+10 8.7%	7.17e+10 8.7%	
array_new_general(void *, long, unsigned long, unsigned long, void	7.17e+10 8.7%	7.17e+10 8.7%	
alloc_array(unsigned long, unsigned long, void *(*)(unsigned long	7.17e+10 8.7%	7.17e+10 8.7%	
__nwa(unsigned long)	7.17e+10 8.7%	7.17e+10 8.7%	
operator new(unsigned long)	7.17e+10 8.7%	7.17e+10 8.7%	
hpcrun_memleak_malloc_helper	7.17e+10 8.7%	7.17e+10 8.7%	
hpcrun_async_block	7.17e+10 8.7%	7.17e+10 8.7%	
sample_event.h: 74	7.17e+10 8.7%	7.17e+10 8.7%	
__array_new	7.17e+10 8.7%	7.17e+10 8.7%	
__array_new	7.17e+10 8.7%	7.17e+10 8.7%	
__array_new	7.17e+10 8.7%	7.17e+10 8.7%	
Umfpack<std::complex<double>>::prepare(void)	3.69e+10 4.5%	3.69e+10 4.5%	
Umfpack<std::complex<double>>::__ct(TCSR<> *, int)	6.43e+09 0.8%	6.43e+09 0.8%	
__array_new	2.24e+09 0.3%	2.24e+09 0.3%	
__array_new	2.24e+09 0.3%	2.24e+09 0.3%	
__array_new	2.24e+09 0.3%	2.24e+09 0.3%	
__array_new	2.24e+09 0.3%	2.24e+09 0.3%	

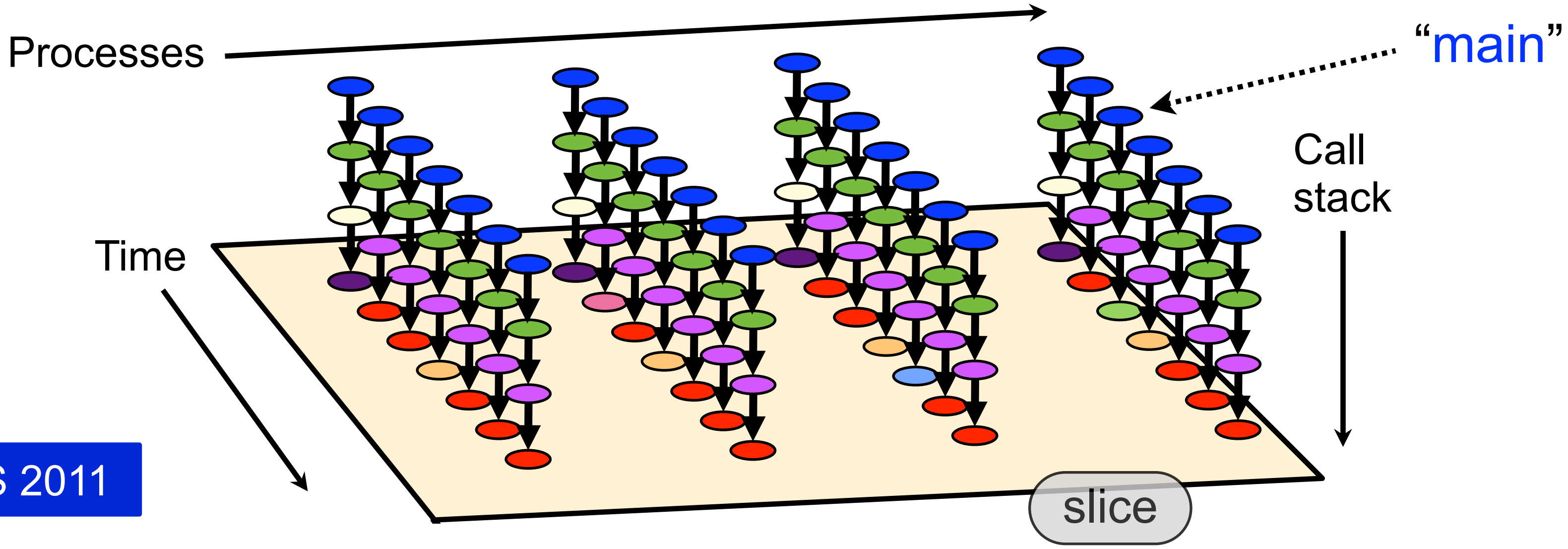
0 Bytes Leaked

Outline

- GPU profiling
- Detecting memory leaks
- Call path tracing
- More work on scaling
- Applying HPCToolkit to FY2011 “Joule Metric” applications
- Data-centric measurement and analysis
- Static analysis of memory access patterns
- Some challenges ahead

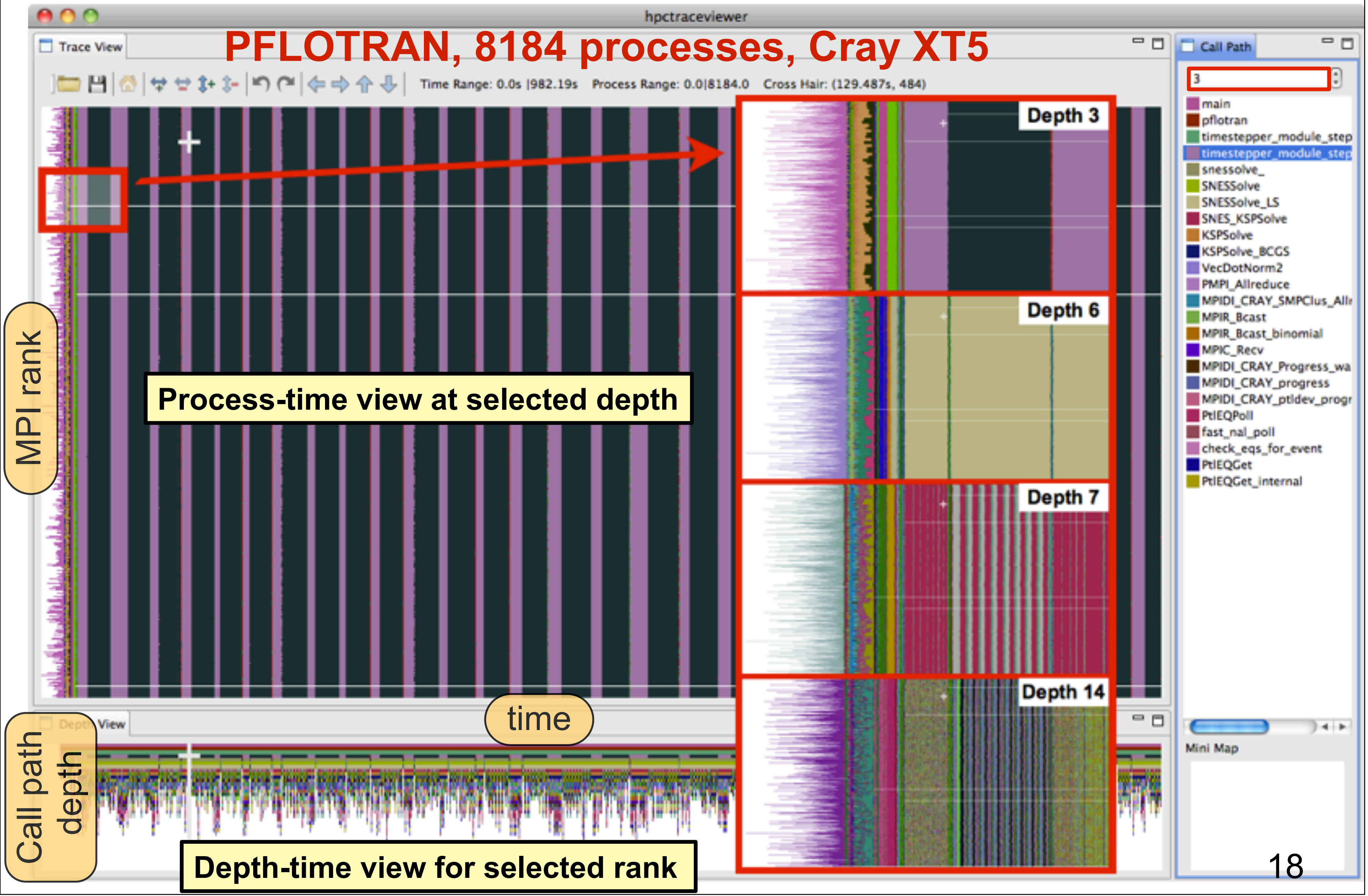
Understanding Temporal Behavior

- **Time-dependent behavior is often invisible in profiles**
  - — **but tracing is difficult to scale to long or large executions**
- **What can we do? Trace call path samples:**
  - — **on each sample, record call path of each thread**
  - — **organize the samples for each thread along a time line**
  - — **view how the execution hierarchically evolves**
    - **assign each procedure a color; view a depth slice of an execution**
  - — **use <u>sampling</u> to scalably render large-scale traces**



ICS 2011

Exposing Temporal Call Path Patterns



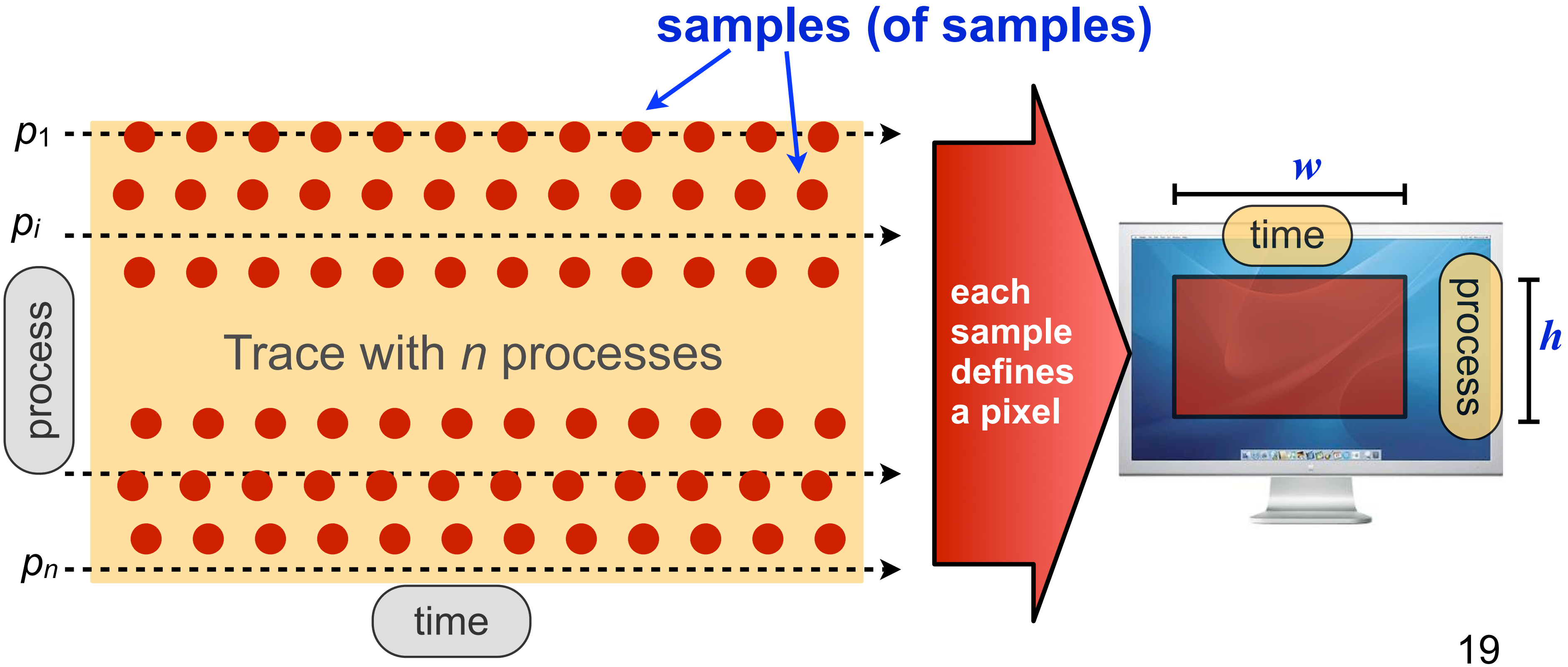
PFLOTRAN, 8184 processes, Cray XT5

Process-time view at selected depth

Depth-time view for selected rank

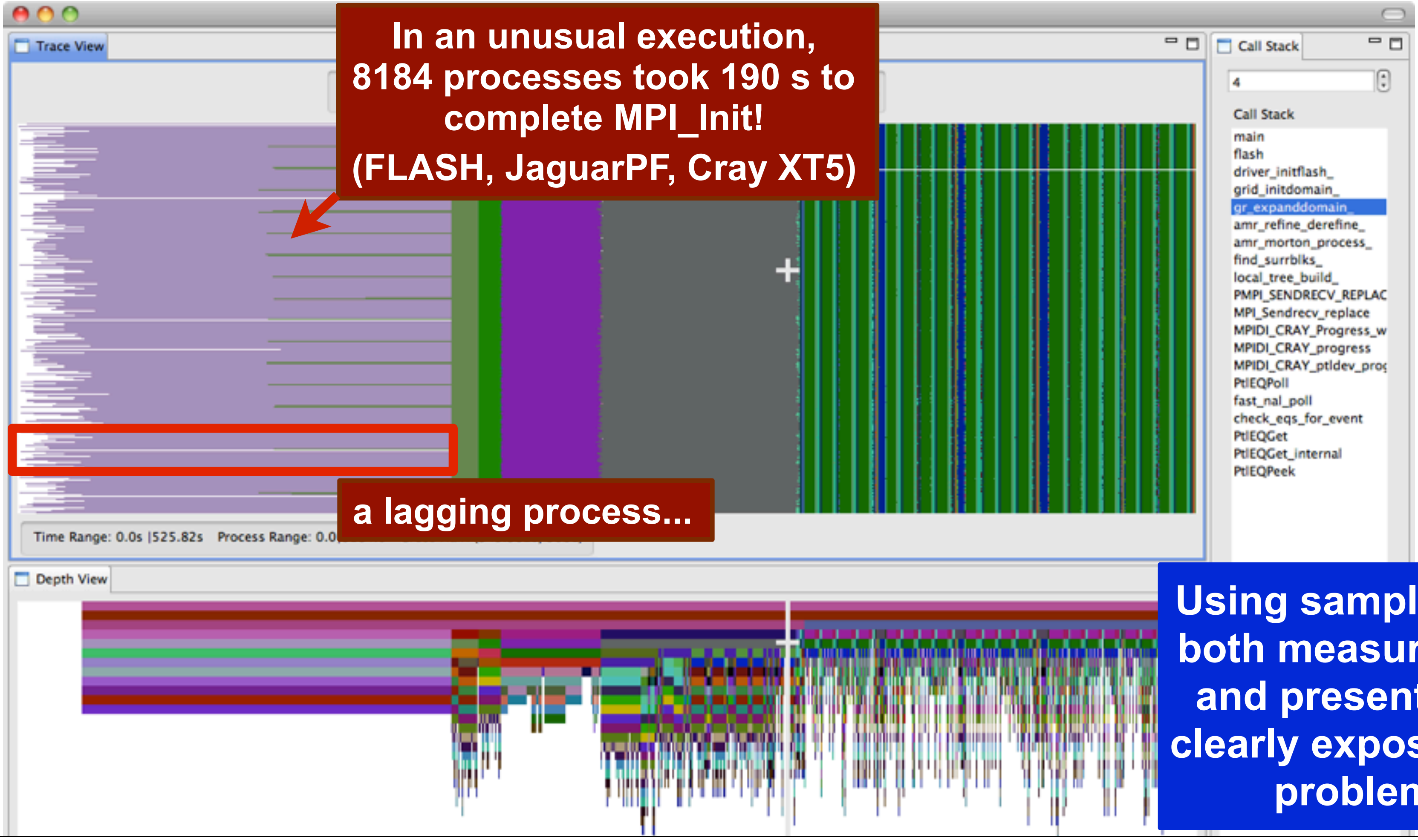
Presenting Large Traces on Small Displays

- **How to render an arbitrary portion of an arbitrarily large trace?**
  - — **we have a display window of dimensions $h \times w$**
  - — **typically many more processes (or threads) than h**
  - — **typically many more samples (trace records) than w**
- **Solution: sample the samples!**



Will Sampling Miss Something Important?

- **Sampling may miss the precise cause of an anomaly...**
  - **but, important anomalies will have (local/non-local) effects**
- **Sampling exposes effects of the important anomalies**



Call Stack

```
main
flash
driver_initflash_
grid_initdomain_
gr_expanddomain_
amr_refine_derefine_
amr_morton_process_
find_surrblks_
local_tree_build_
PMPI_SENDRECV_REPLAC
MPI_Sendrecv_replace
MPIDI_CRAY_Progress_w
MPIDI_CRAY_progress
MPIDI_CRAY_ptldev_proc
PtlEQPoll
fast_nal_poll
check_eqs_for_event
PtlEQGet
PtlEQGet_internal
PtlEQPeek
```

In an unusual execution, 8184 processes took 190 s to complete MPI_Init!

(FLASH, JaguarPF, Cray XT5)

a lagging process...

Using sampling for both measurement and presentation clearly exposed the problem.

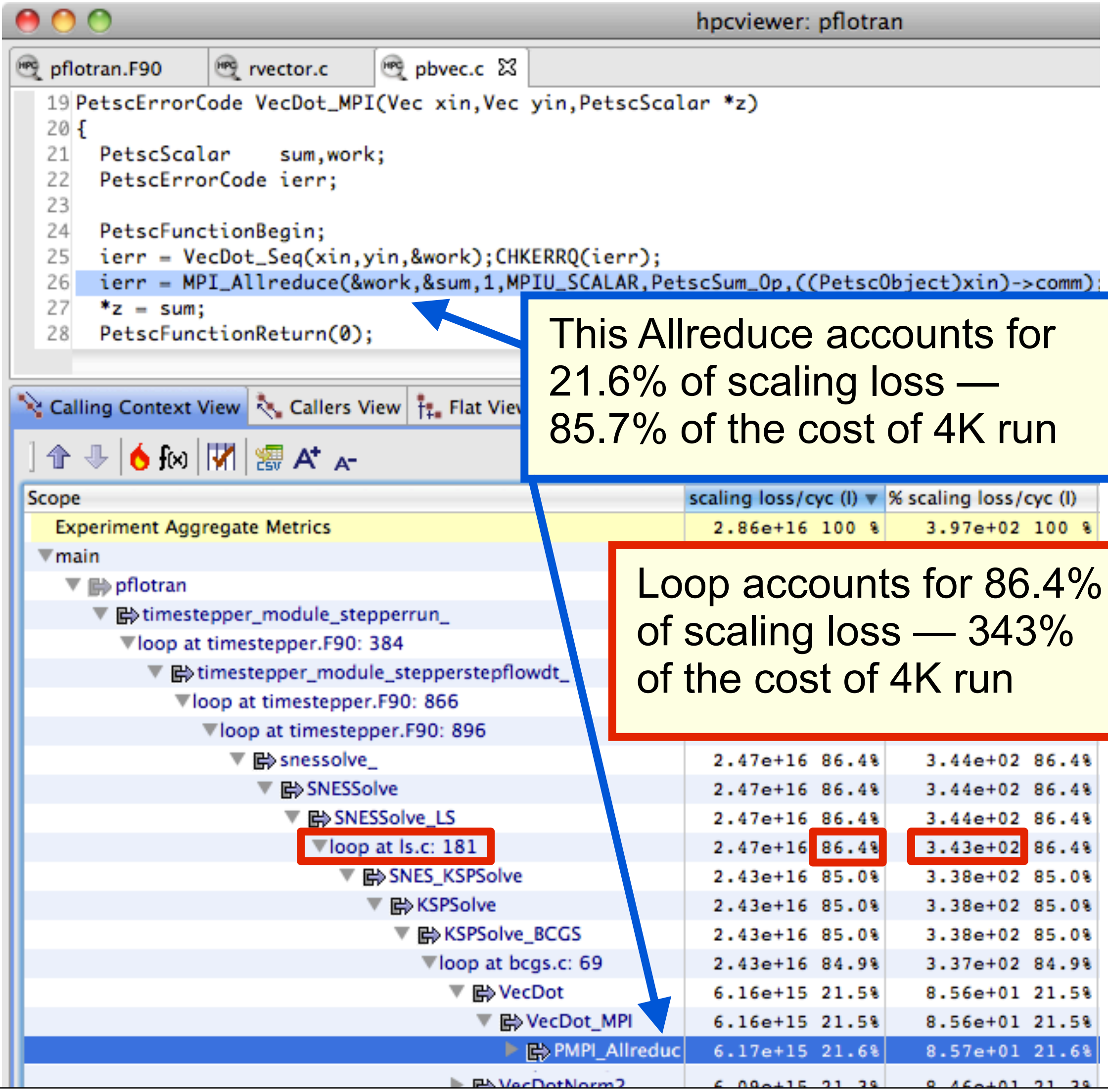
Outline

- GPU profiling
- Detecting memory leaks
- Call path tracing
- More work on scaling
- Applying HPCToolkit to FY2011 “Joule Metric” applications
- Data-centric measurement and analysis
- Static analysis of memory access patterns
- Some challenges ahead

Enabling Larger-Scale Measurements

- **Sample processes within SPMD applications**
  - **record data on a process with probability *p***
  - **simplification of Gamblin et al., IPDPS ’08**
  - **effective**

Pinpoint strong scaling bottlenecks in PFLOTRAN, 4K — 32K cores, running on a Cray XT5 (JaguarPF).



```
PetscErrorCode VecDot_MPI(Vec xin,Vec yin,PetscScalar *z)
{
  PetscScalar    sum,work;
  PetscErrorCode ierr;

  PetscFunctionBegin;
  ierr = VecDot_Seq(xin,yin,&work);CHKERRQ(ierr);
  ierr = MPI_Allreduce(&work,&sum,1,MPIU_SCALAR,PetscSum_Op,((PetscObject)xin)->comm);
  *z = sum;
  PetscFunctionReturn(0);
```

Scope	scaling loss/cyc (I) ▼	% scaling loss/cyc (I)
Experiment Aggregate Metrics	2.86e+16 100 %	3.97e+02 100 %
▼main		
▼ pflotran		
▼ timestepper_module_stepperrun_		
▼loop at timestepper.F90: 384		
▼ timestepper_module_stepperstepflowdt_		
▼loop at timestepper.F90: 866		
▼loop at timestepper.F90: 896		
▼ snessolve_	2.47e+16 86.4%	3.44e+02 86.4%
▼ SNESSolve	2.47e+16 86.4%	3.44e+02 86.4%
▼ SNESSolve_LS	2.47e+16 86.4%	3.44e+02 86.4%
▼loop at ls.c: 181	2.47e+16 86.4%	3.43e+02 86.4%
▼ SNES_KSPSolve	2.43e+16 85.0%	3.38e+02 85.0%
▼ KSPSolve	2.43e+16 85.0%	3.38e+02 85.0%
▼ KSPSolve_BCGS	2.43e+16 85.0%	3.38e+02 85.0%
▼loop at bcgs.c: 69	2.43e+16 84.9%	3.37e+02 84.9%
▼ VecDot	6.16e+15 21.5%	8.56e+01 21.5%
▼ VecDot_MPI	6.16e+15 21.5%	8.56e+01 21.5%
▶ PMPI_Allreduc	6.17e+15 21.6%	8.57e+01 21.6%

This Allreduce accounts for 21.6% of scaling loss — 85.7% of the cost of 4K run

Loop accounts for 86.4% of scaling loss — 343% of the cost of 4K run

Real Tools Must Address the ‘D’ in R&D

- First version of tracer used one trace file per process
- Problem: File systems don’t handle 1000s of files per directory
  - FSs optimize for data integrity rather than for fast file lookup
    - typical: store files in order of creation and use linear search instead of data structure optimized for lookup
- Bleeding-edge version of tracer
  - fast and scalable trace record lookup
    - merge all trace files into one file
      - index + trace files
  - resolve several inefficiencies
    - e.g.: eliminate unnecessary duplication of call path data
    - one can only expect so much from high school seniors
- TODO: use SionLib or PLFS to write profile and trace data

Refining Analysis and Presentation

CCT = Calling Context Tree

- **Current scalable database requires O(1 CCT) space**
  - — non-distributed data structure → per-process requirement
- **Many opportunities for refining database**
  - — never, ever use XML
    - replacing with Google Protocol Buffers
    - expect 1–2 orders of magnitude in space savings
  - — use appropriate (sub) data structures
    - use dense vectors for dense data (e.g., inclusive metric values)
    - use sparse vectors for sparse data (e.g., exclusive metric values)
  - — post-process data to accelerate performance of user interface
    - scatter plots: better to have per-thread metric values for CCT node instead of all CCT-node metric values for a thread
  - — incrementally prune irrelevant profile data
    - reduce the high-water space requirement for building a CCT
  - — possibly another order of magnitude (on top of XML change)

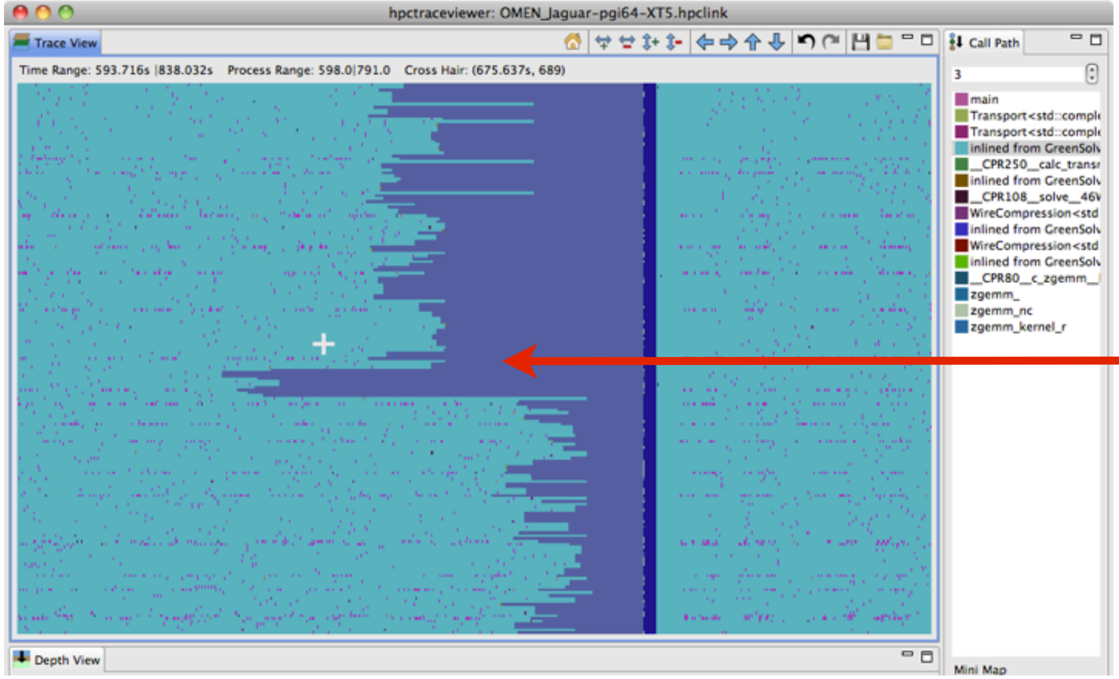
Outline

- **GPU profiling**
- **Detecting memory leaks**
- **Call path tracing**
- **More work on scaling**
- **Applying HPCToolkit to FY2011 “Joule Metric” applications**
- **Data-centric measurement and analysis**
- **Static analysis of memory access patterns**
- **Some challenges ahead**

Work on FY2011 ‘Joule Metric’ Codes

K. Roche: Given OMEN, a highly tuned app, can you find anything?

- **Identified load imbalance:**
  - using tracing of call path samples
    - found early/late arrivers at an MPI_Allreduce
  - using differential profiling & load imbalance analysis
    - compare early/late arrivers
    - confirm that exposed idleness is fully offset by FP computation
  - simple case of load imbalance
    - mismatch between input data and # of processors

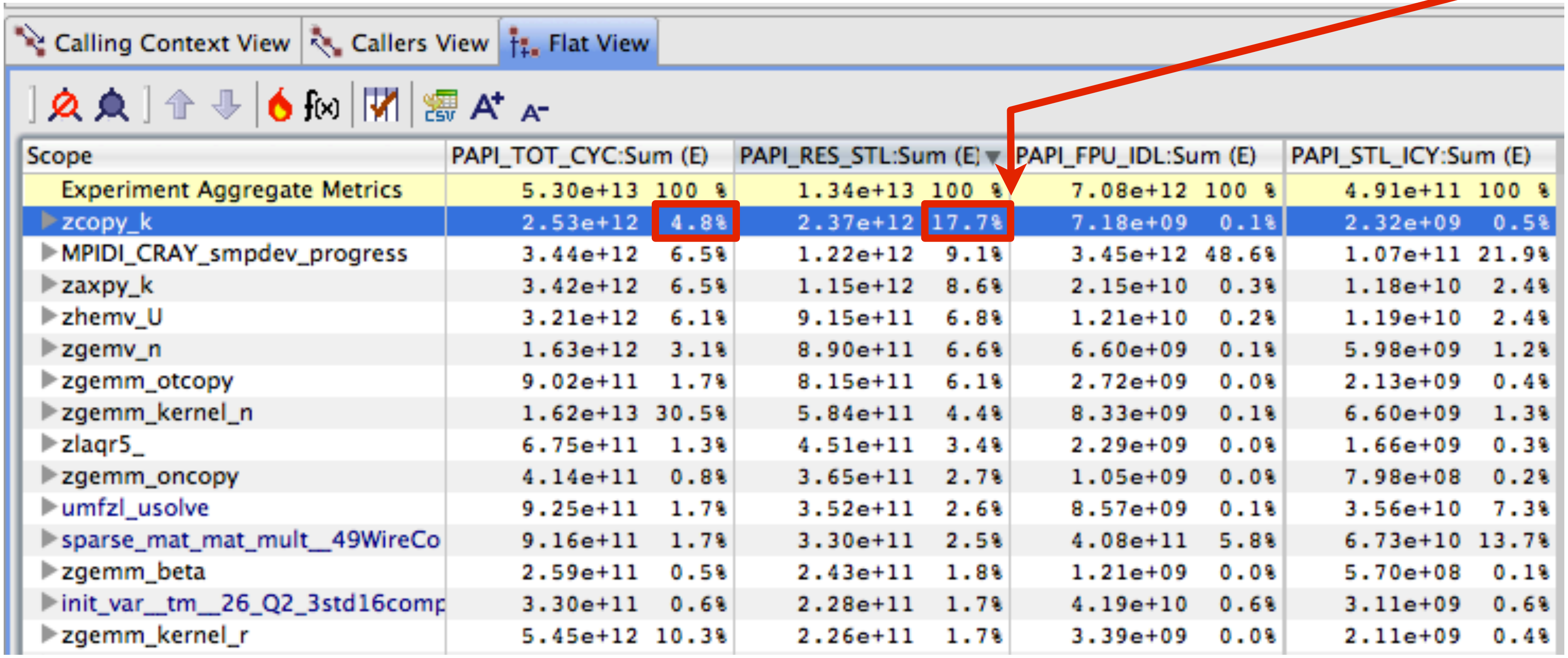


hpctraceviewer: OMEN_Jaguar-pgi64-XT5.hpclink

Time Range: 593.716s |838.032s   Process Range: 598.0|791.0   Cross Hair: (675.637s, 689)

Work on FY2011 ‘Joule Metric’ Codes

- **Improved performance of array copies:**
  - — most inefficiency was in Goto BLAS *x*copy wrappers
    - 5% of execution time; 18% of resource stalls
    - xcopy: assembly — no unwind information!
  - — specialized calls to *x*copy to use memcopy when possible
    - Goto BLAS copy didn’t exploit memory parallelism, prefetching
  - — improved cost of copies by 25% (1.3% overall)



Scope	PAPI_TOT_CYC:Sum (E)		PAPI_RES_STL:Sum (E)		PAPI_FPU_IDL:Sum (E)		PAPI_STL_ICY:Sum (E)	
Experiment Aggregate Metrics	5.30e+13	100 %	1.34e+13	100 %	7.08e+12	100 %	4.91e+11	100 %
zcopy_k	2.53e+12	4.8%	2.37e+12	17.7%	7.18e+09	0.1%	2.32e+09	0.5%
MPIDI_CRAY_smpdev_progress	3.44e+12	6.5%	1.22e+12	9.1%	3.45e+12	48.6%	1.07e+11	21.9%
zaxpy_k	3.42e+12	6.5%	1.15e+12	8.6%	2.15e+10	0.3%	1.18e+10	2.4%
zhemv_U	3.21e+12	6.1%	9.15e+11	6.8%	1.21e+10	0.2%	1.19e+10	2.4%
zgemv_n	1.63e+12	3.1%	8.90e+11	6.6%	6.60e+09	0.1%	5.98e+09	1.2%
zgemm_otcopy	9.02e+11	1.7%	8.15e+11	6.1%	2.72e+09	0.0%	2.13e+09	0.4%
zgemm_kernel_n	1.62e+13	30.5%	5.84e+11	4.4%	8.33e+09	0.1%	6.60e+09	1.3%
zlaqr5_	6.75e+11	1.3%	4.51e+11	3.4%	2.29e+09	0.0%	1.66e+09	0.3%
zgemm_oncopy	4.14e+11	0.8%	3.65e+11	2.7%	1.05e+09	0.0%	7.98e+08	0.2%
umfzl_usolve	9.25e+11	1.7%	3.52e+11	2.6%	8.57e+09	0.1%	3.56e+10	7.3%
sparse_mat_mat_mult__49WireCo	9.16e+11	1.7%	3.30e+11	2.5%	4.08e+11	5.8%	6.73e+10	13.7%
zgemm_beta	2.59e+11	0.5%	2.43e+11	1.8%	1.21e+09	0.0%	5.70e+08	0.1%
init_var__tm__26_Q2_3std16comp	3.30e+11	0.6%	2.28e+11	1.7%	4.19e+10	0.6%	3.11e+09	0.6%
zgemm_kernel_r	5.45e+12	10.3%	2.26e+11	1.7%	3.39e+09	0.0%	2.11e+09	0.4%

Outline

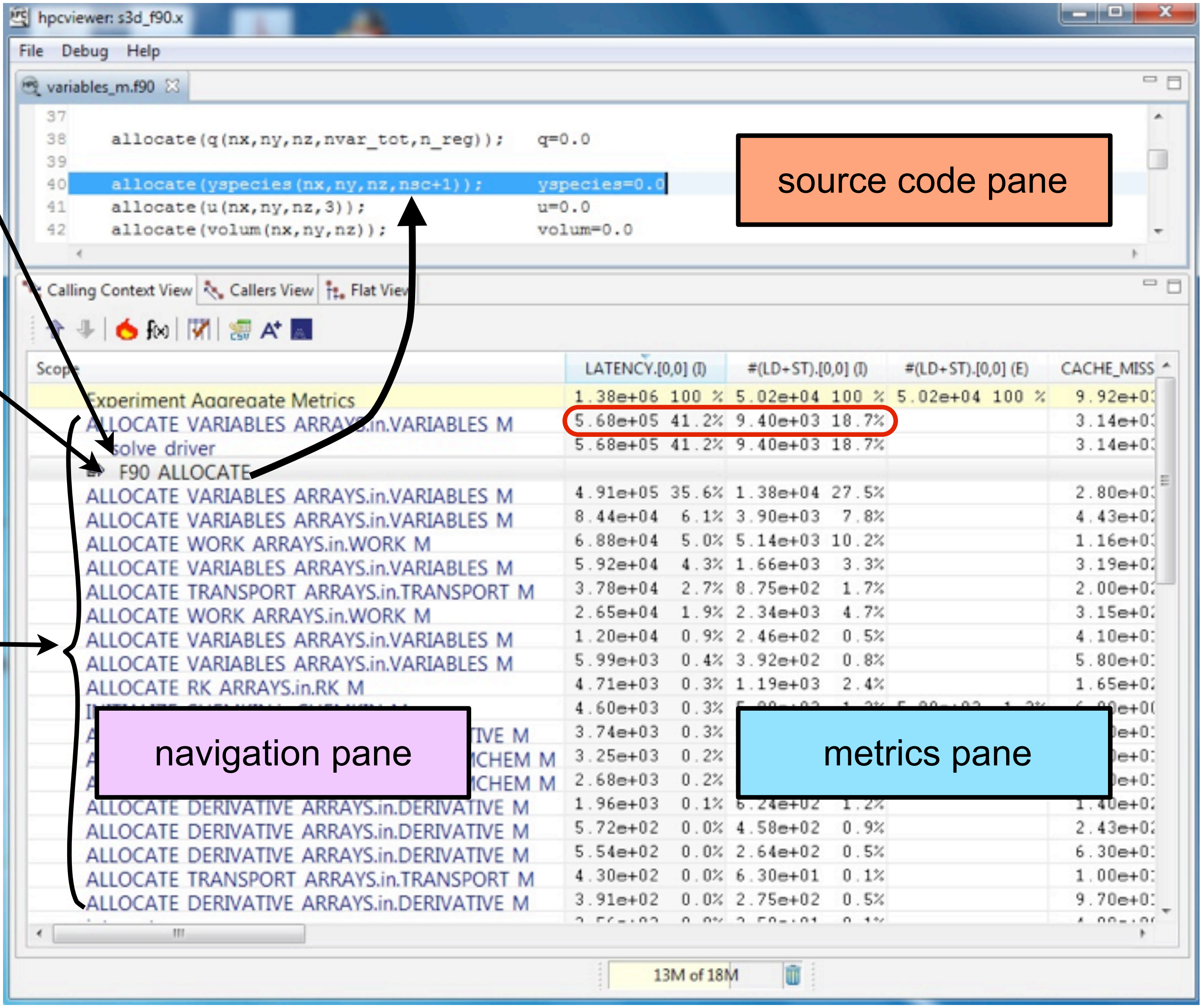
- GPU profiling
- Detecting memory leaks
- Call path tracing
- More work on scaling
- Applying HPCToolkit to FY2011 “Joule Metric” applications
- Data-centric measurement and analysis
- Static analysis of memory access patterns
- Some challenges ahead

Data-centric View

Root of all uses

Allocation point

Data structures



source code pane

```
      allocate(q(nx,ny,nz,nvar_tot,n_reg));    q=0.0

      allocate(yspecies(nx,ny,nz,nsc+1));      yspecies=0.0
      allocate(u(nx,ny,nz,3));                 u=0.0
      allocate(volum(nx,ny,nz));               volum=0.0
```

Scope	LATENCY.[0,0] (I)	#(LD+ST).[0,0] (I)	#(LD+ST).[0,0] (E)	CACHE_MISS
Experiment Aggregate Metrics	1.38e+06 100 %	5.02e+04 100 %	5.02e+04 100 %	9.92e+0
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	5.68e+05 41.2%	9.40e+03 18.7%		3.14e+0
solve driver	5.68e+05 41.2%	9.40e+03 18.7%		3.14e+0
F90 ALLOCATE				
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	4.91e+05 35.6%	1.38e+04 27.5%		2.80e+0
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	8.44e+04 6.1%	3.90e+03 7.8%		4.43e+0
ALLOCATE WORK ARRAYS.in.WORK M	6.88e+04 5.0%	5.14e+03 10.2%		1.16e+0
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	5.92e+04 4.3%	1.66e+03 3.3%		3.19e+0
ALLOCATE TRANSPORT ARRAYS.in.TRANSPORT M	3.78e+04 2.7%	8.75e+02 1.7%		2.00e+0
ALLOCATE WORK ARRAYS.in.WORK M	2.65e+04 1.9%	2.34e+03 4.7%		3.15e+0
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	1.20e+04 0.9%	2.46e+02 0.5%		4.10e+0
ALLOCATE VARIABLES ARRAYS.in.VARIABLES M	5.99e+03 0.4%	3.92e+02 0.8%		5.80e+0
ALLOCATE RK ARRAYS.in.RK M	4.71e+03 0.3%	1.19e+03 2.4%		1.65e+0
[illegible]	4.60e+03 0.3%	[illegible]	[illegible]	[illegible]
[illegible]TIVE M	3.74e+03 0.3%	[illegible]	[illegible]	[illegible]
[illegible]CHEM M	3.25e+03 0.2%	[illegible]	[illegible]	[illegible]
[illegible]CHEM M	2.68e+03 0.2%	[illegible]	[illegible]	[illegible]
ALLOCATE DERIVATIVE ARRAYS.in.DERIVATIVE M	1.96e+03 0.1%	6.24e+02 1.2%		1.40e+0
ALLOCATE DERIVATIVE ARRAYS.in.DERIVATIVE M	5.72e+02 0.0%	4.58e+02 0.9%		2.43e+0
ALLOCATE DERIVATIVE ARRAYS.in.DERIVATIVE M	5.54e+02 0.0%	2.64e+02 0.5%		6.30e+0
ALLOCATE TRANSPORT ARRAYS.in.TRANSPORT M	4.30e+02 0.0%	6.30e+01 0.1%		1.00e+0
ALLOCATE DERIVATIVE ARRAYS.in.DERIVATIVE M	3.91e+02 0.0%	2.75e+02 0.5%		9.70e+0

navigation pane

metrics pane

Linux Kernel Support for AMD's IBS

- **Why perfmon2 & libpfm3 vs. perf_events?**
  - perfmon2 supports per-thread mode for IBS
  - HPCToolkit monitors threads separately
- **Problem in perfmon2 driver for Linux 2.6.30**
  - runaway kernel process (kondemand/12)
    - causes system crash
  - occurs very few times when run sequentially
  - always occurs when monitoring parallel programs
- **Patches (from Oprofile kernel and already known workarounds)**
  - erratum 420: set IbsOpMaxCnt & IbsOpEn bits in two steps
  - UBTS 227027: enable/disable LBR
  - UBTS 299030: Read IP immediately after setting the IBS OP

No errors, no crashes

Outline

- **GPU profiling**
- **Detecting memory leaks**
- **Call path tracing**
- **More work on scaling**
- **Applying HPCToolkit to FY2011 “Joule Metric” applications**
- **Data-centric measurement and analysis**
- **Static analysis of memory access patterns**
- **Some challenges ahead**

Memory Access Patterns Matter

- Understand cache usage
  - non-unit stride → poor spatial locality
  - data access = non-unit stride% + unit stride%
    - combine data-centric analysis: whether to transpose an array’s layout
- Understand memory access patterns at the loop level
  - e.g. memory footprint of an access
  - recording all memory accesses has large overhead
    - instrument all loads/stores and collect them in a buffer
    - compute the reuse distance for each element in the buffer
    - more than 100x slow down
  - use combination of static analysis + dynamic information instead
    - use stride analysis to reduce instrumentation necessary

Goal and Approach

- Goal: feed pattern information to HPCToolkit (future work)
  - data-centric measurement and analysis
  - memory footprint analysis
- Approach: analyze memory access stride in loops
  - perform static analysis of an application binary
    - only analyze indexed accesses
  - use Dyninst
    - parseAPI & instructionAPI
      - extract loop information and memory access instructions
    - dataflowAPI
      - perform data flow analysis using program slicing

Precursor to Stride Analysis: Loop Analysis

- **parseAPI: analyze the control flow, build CFG**
- **Return all basic blocks in the loop exclusively or inclusively**
- **Find loop headers**

Stride Analysis Algorithm

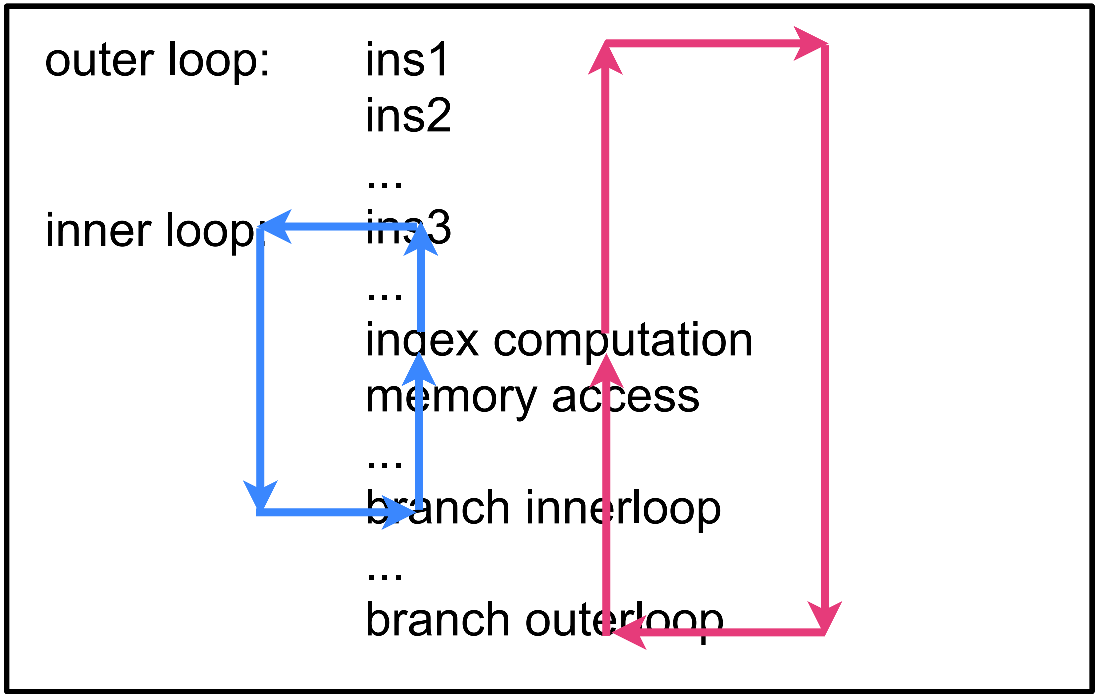
- Analyze each memory access instruction in the loop
  - filter out the scalar instructions
    - access the memory using the unchanged register as the index (bp)
  - get the multiplier for indexed operations: (%rbp,%rax,4) is 4
- Find the instruction, compute the index
  - backward slice from the memory access instruction in the loop
    - backward slice on the index register (rax is definitely the index)
      - movss 0x602080(,%rax,4),%xmm0
      - movss (%rbp,%rax,4),%xmm0
      - movss (%rax),%xmm0
    - backward slice on both index registers (rax, rbx are possible indexes)
      - movss (%rbx,%rax,1),%xmm0
  - symbolic evaluation
    - compute the symbolic expression using slicing
    - find how the index register changes in the loop body
    - return an AST

Simplify Results of Stride Analysis

- Raw AST data using ROSE symbols
  - <extract:32>(<add>(<extMSB>(<V([S [_Z19initialize_matricesv,-24,0]]:80487c8)>,<33:32>,),<add>(<extMSB>(<4:32>,<33:32>,),<0:1>,),),<0:33>,<32:33>,)
- Simply the AST
  - remove unnecessary operators
  - handle the value from the memory
    - index
      - if the value comes from an LHS of an instruction in the loop
    - constant
      - if the value comes from an LHS of an instruction outside the loop
      - if the value is not an LHS → it does not change in the loop
    - indirect
      - if the value is from an unknown location which is an indirect reference
  - simplified version of above expression: (index+(0x4+0x0))

Eliminate Extraneous Details

- **Inner loop details are irrelevant when analyzing outer loop**
  - **slicing in the inner loop generates extraneous detail**
  - **for example: ((constant*0x3e8)+((index+(0x1+0x0))+0x0))**

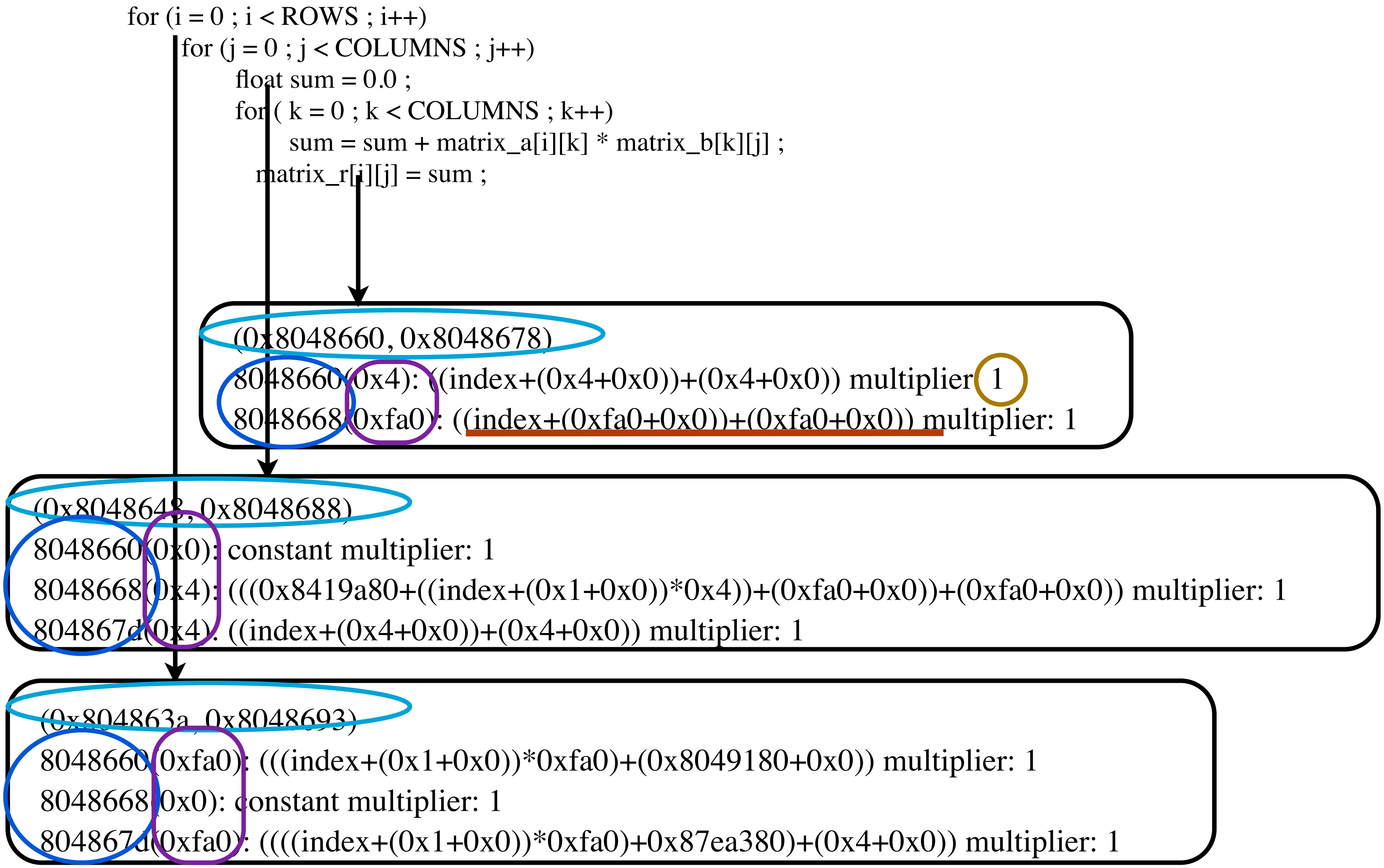


```
outer loop:     ins1
                ins2
                ...
inner loop:     ins3
                ...
                index computation
                memory access
                ...
                branch innerloop
                ...
                branch outerloop
```

- **Prune the AST to eliminate the extraneous detail**
  - **keep a sub-AST only if it is related to the index**

Preliminary Experimental Results

- **Kernel of matrix multiplication**



```
for (i = 0 ; i < ROWS ; i++)
        for (j = 0 ; j < COLUMNS ; j++)
                float sum = 0.0 ;
                for ( k = 0 ; k < COLUMNS ; k++)
                        sum = sum + matrix_a[i][k] * matrix_b[k][j] ;
                   matrix_r[i][j] = sum ;
```

```
(0x8048660, 0x8048678)
8048660(0x4): ((index+(0x4+0x0))+(0x4+0x0)) multiplier: 1
8048668(0xfa0): ((index+(0xfa0+0x0))+(0xfa0+0x0)) multiplier: 1
```

```
(0x8048648, 0x8048688)
8048660(0x0): constant multiplier: 1
8048668(0x4): (((0x8419a80+((index+(0x1+0x0))*0x4))+(0xfa0+0x0))+(0xfa0+0x0)) multiplier: 1
804867d(0x4): ((index+(0x4+0x0))+(0x4+0x0)) multiplier: 1
```

```
(0x804863a, 0x8048693)
8048660(0xfa0): (((index+(0x1+0x0))*0xfa0)+(0x8049180+0x0)) multiplier: 1
8048668(0x0): constant multiplier: 1
804867d(0xfa0): ((((index+(0x1+0x0))*0xfa0)+0x87ea380)+(0x4+0x0)) multiplier: 1
```

Outline

- **GPU profiling**
- **Detecting memory leaks**
- **Call path tracing**
- **More work on scaling**
- **Applying HPCToolkit to FY2011 “Joule Metric” applications**
- **Data-centric measurement and analysis**
- **Static analysis of memory access patterns**
- **Some challenges ahead**

Some Challenges Ahead

- Scale measurement and analysis to > 1M cores
- Handle requirements for asymmetric measurement
- Understand usage of shared resources
  - examples
    - shared cores (SMT)
    - shared cache
    - memory bandwidth
    - network
  - quantify utilization
  - quantify impact of contention
    - aggregate
    - over time
  - attribute metrics to code
- Complete analysis of hybrid programs
- From metrics to bottleneck diagnosis
  - work with PerfExpert team at UT and Texas State