Performance Tool Integration in Programming Environments for GPU Acceleration: Experiences with TAU and HMPP

***Allen D. Malony*[1,2], Shangkar Mayanglambam[1]**

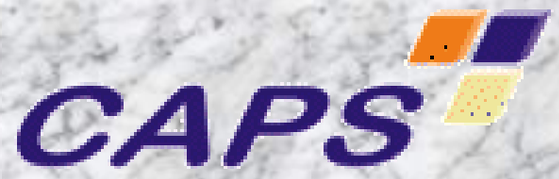
Sameer Shende[1,2], Matt Sottile[1]

[1]Computer and Information Science Department, University of Oregon

[2]ParaTools, Inc.

Laurent Morin, Stephane Bihan, Francois Bodin

CAPS Entreprise



Outline

- Motivation
- HMPP Workbench
- TAU and TAUcuda
- Instrumentation model
- Case studies
  - Game of Life
  - Double buffering optimization
- Future work

Motivation

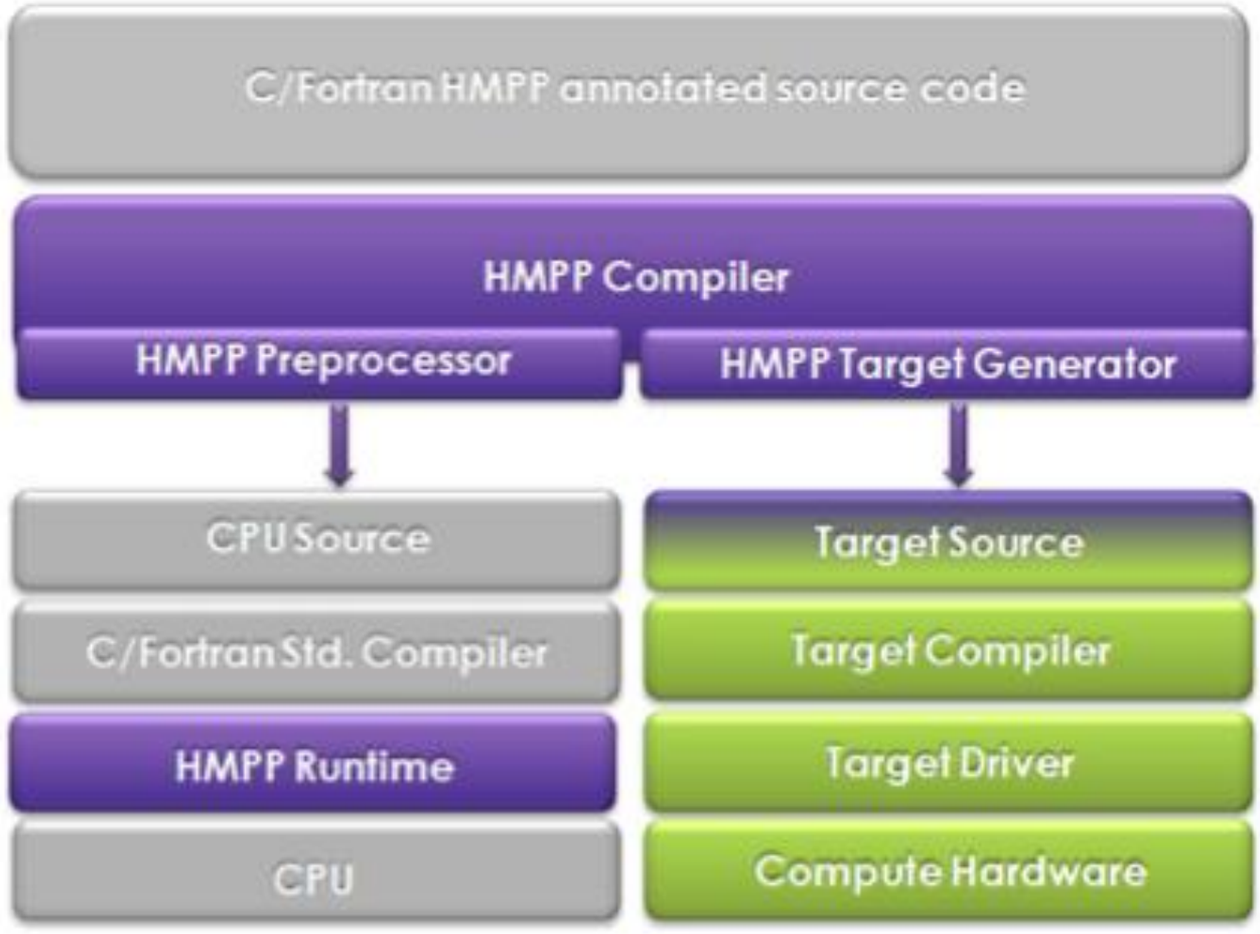
- Multi-core, heterogeneous (MCH) execution environments
  - Difficult to program with lower-level interfaces
  - Heterogeneous computation adds complexity (e.g., GPU)
  - Performance is harder measure, analyze, and understand
- Higher-level programming support
  - Facilitates MCH application development
  - Provides abstract computation / execution model
  - Distances developer from performance factors
- Integrate performance tools with MCH programming system
  - Performance analysis in context of high-level model
  - Automation of instrumentation and measurement
  - Enable performance feedback for optimization

Integration of HMPP and TAU / TAUcuda

- HMPP Workbench
  - High-level system for programming multi-core and GPU-accelerated systems
  - Portable and retargetable
- TAU Performance System®
  - Parallel performance instrumentation, measurement, and analysis system
  - Target scalable parallel systems
- TAUcuda
  - CUDA performance measurement
  - Integrated with TAU
- *HMPP–TAU* = HMPP + TAU + TAUcuda

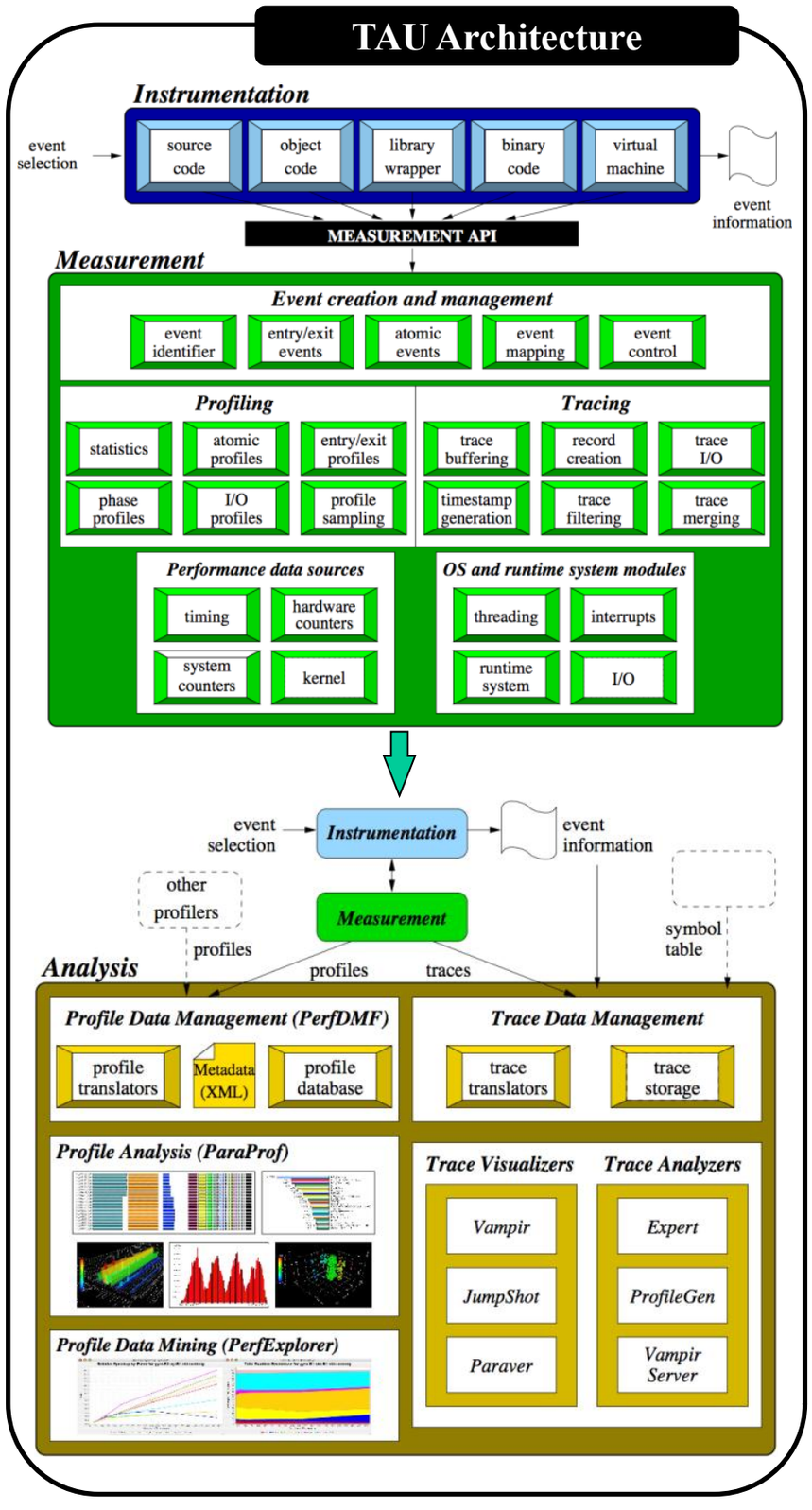
HMPP Workbench (www.caps-entreprise.com)

- C and Fortran GPU programming directives
  - Define and execute GPU-accelerated versions of functions
  - Implement efficient communication patterns
  - Build parallel hybrid applications with OpenMP and MPI
- Open hybrid compiling workbench
  - Automatically generated CUDA computations
  - Use standard compilers and hardware vendor tools
  - Drive the whole compilation workflow
- HMPP runtime library
  - Dispatch computations on available GPUs
  - Scale to multi-GPUs systems



TAU Performance System Project (tau.uoregon.edu)

- Tuning and Analysis Utilities
  - 16+ year project effort
- HPC performance system framework
  - Integrated, scalable, and flexible
  - Parallel programming paradigms
- Integrated performance toolkit
  - Instrumentation, measurement, analysis, and visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining

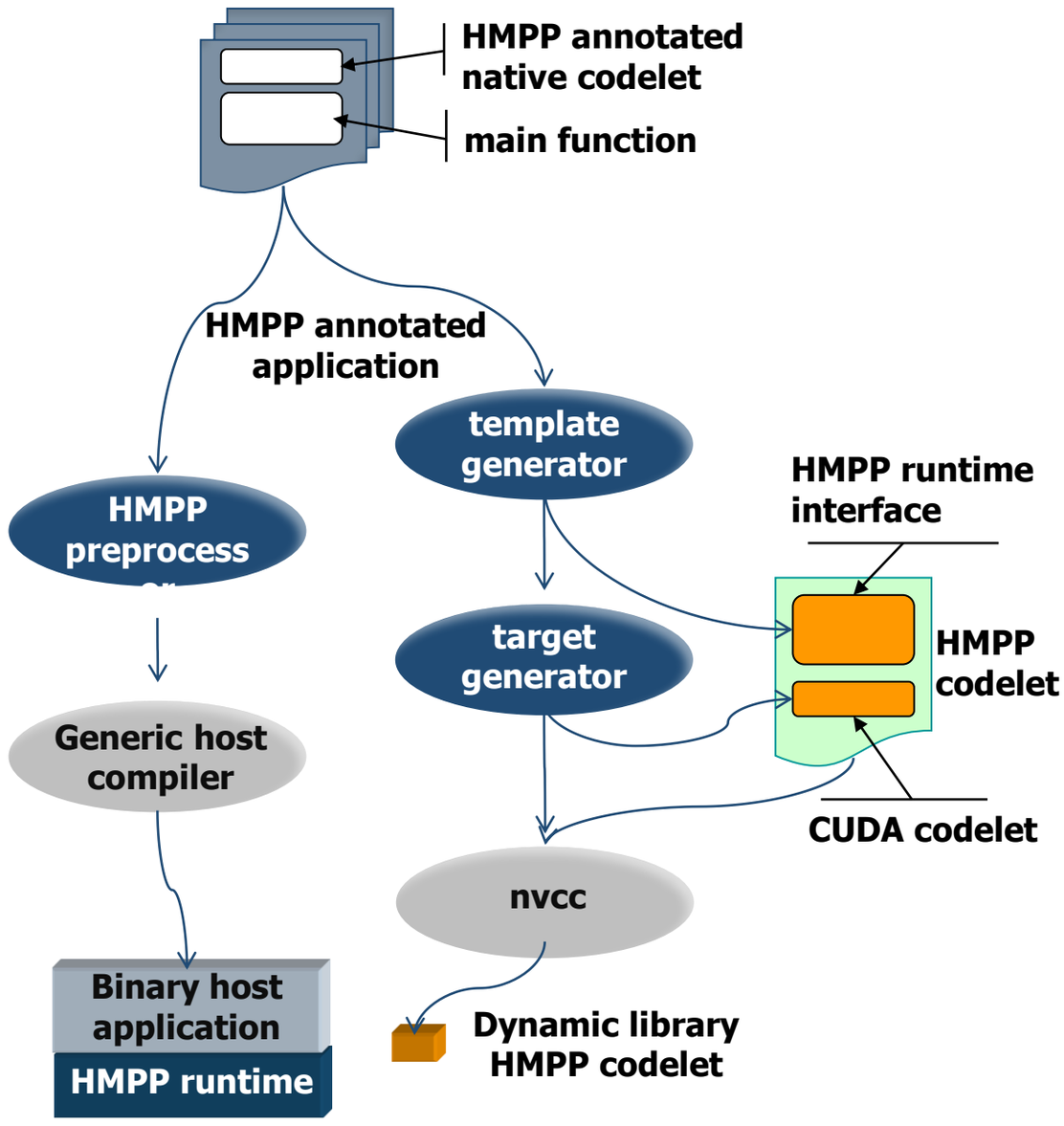


TAUcuda CUDA Performance Measurement

- CUDA kernel invocations are asynchronous
  - Difficult for CPU to measure concurrency (kernel begin/end)
- TAUcuda built on CUDA event interface (*cudaEventRecord*)
  - Allow "events" to be placed in streams and processed
    - events are timestamped
  - CUDA runtime reports GPU timing in event structure
  - Events are reported back to CPU when requested
    - use *begin* and *end* events to calculate intervals
  - CPU retrieves events in a non-blocking and blocking manner
- Associates *TAU event context* with CUDA events
- Can be used to capture CPU-side "waiting time"
- ParCo '09 paper on TAUcuda (*A-GPU2: GPU Programming*)

Integration Challenges

- Instrumentation model
  - HMPP generates code
    - host code and target code
  - Links with runtime library
  - Must insert instrumentation before/during code generation
  - Instrument runtime library
  - Events should support analysis
- Measurement APIs
  - Target TAU and TAUcuda
  - Make generic interface with weak binding
  - Need to be compatible with execution model (e.g., threading)



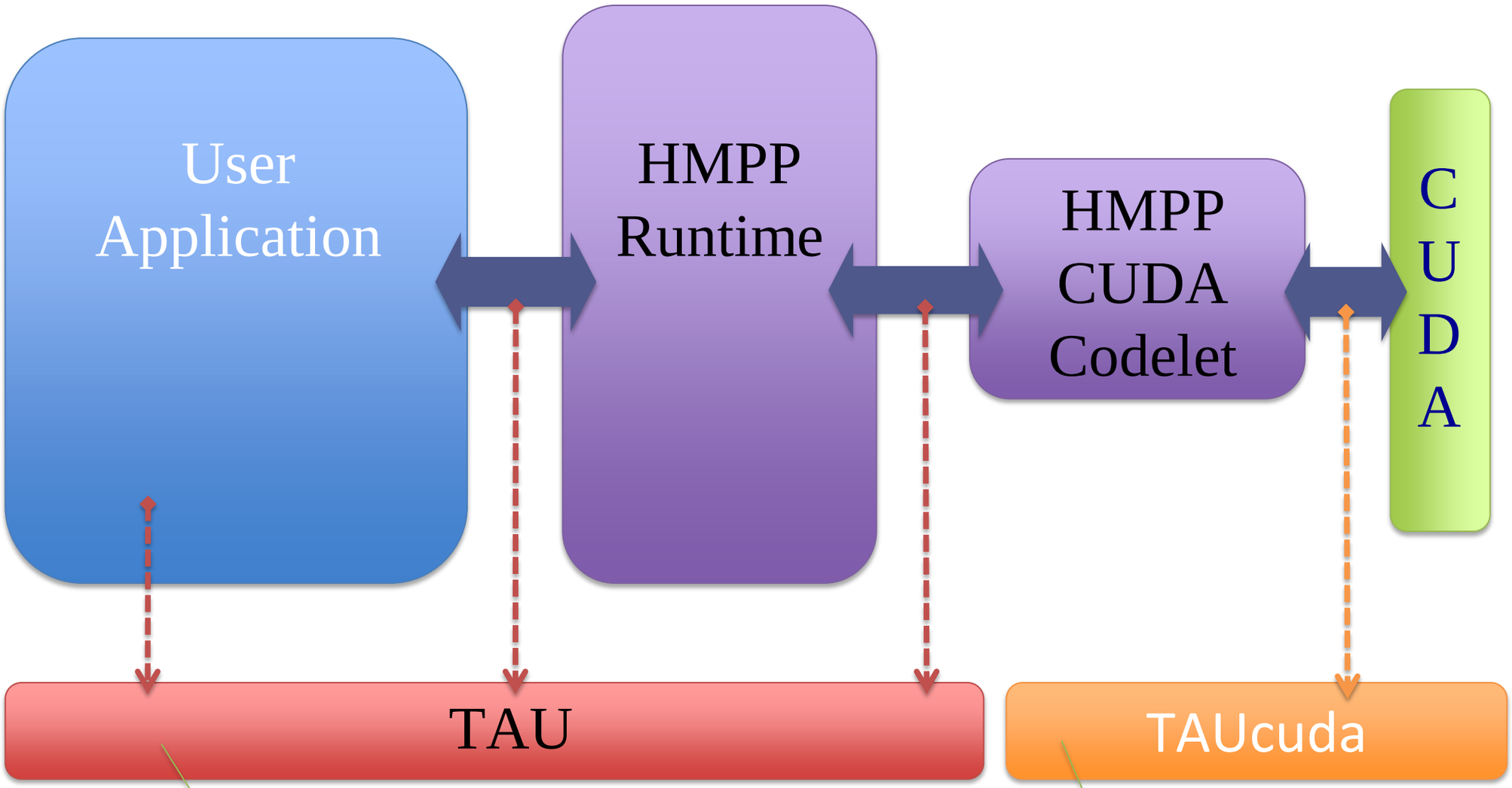
Integration Challenges (2)

- Analysis model
  - Portray meaningful and consistent performance views
    - support HMPP computation abstraction
  - Show all aspects of accelerator execution
    - data transfers
    - kernel operations
  - Present asynchronous and concurrent operation
  - Generate useful performance metrics

HMPP–TAU Instrumentation Workflow

- Goal
  - Automated, seamless instrumentation of HMPP applications
  - Instrumentation support for codelet target generation
- Workflow
  - Replace compiler with TAU compiler
    - allows instrumentation of application-level events
  - Indicate HMPP compiler (as TAU's compiler target)
    - TAUcuda instrumentation automatically generated
    - placed in codelet around CUDA kernel statements
- HMPP TAU management
  - Default support in the HMPP runtime
  - Activated using compiler flags

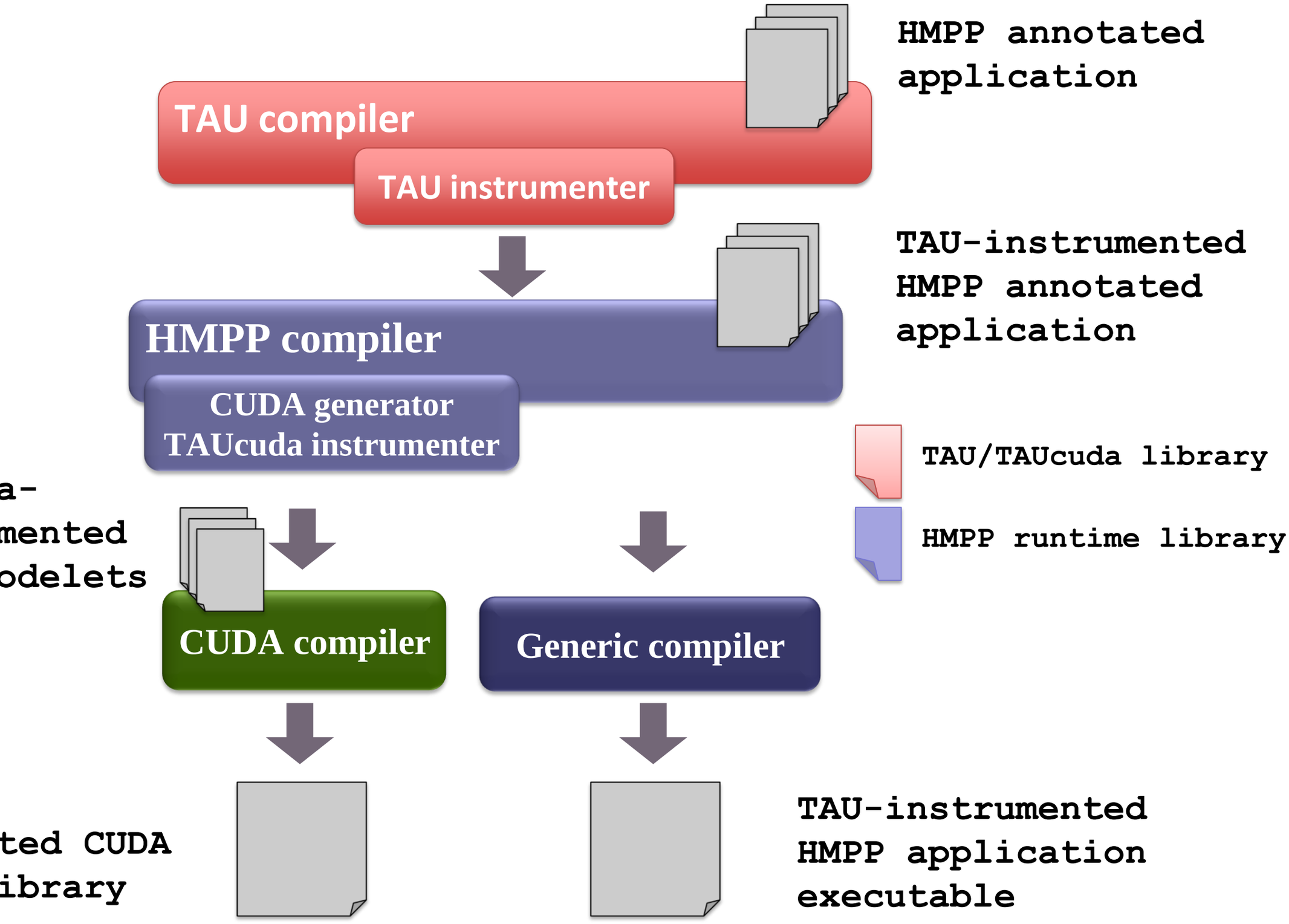
HMPP-TAU Event Instrumentation/Measurement



Measurement
- User events
- HMPP events
- Codelet events

Measurement
- CUDA stream events
- Waiting information

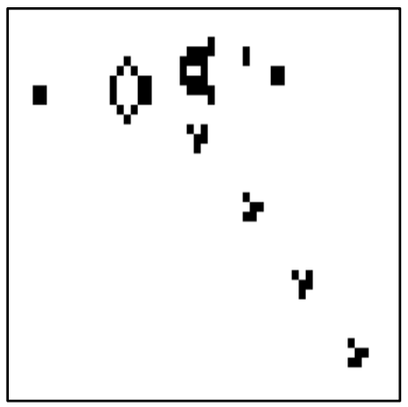
HMPP-TAU Compilation Workflow



Performance Measurement Results

- TAU measurements (Host CPU)
  - Performance profile
    - exclusive/inclusive, callpath
  - Performance trace
  - Timing and hardware counter data
  - HMPP CPU, CUDA data transfer, and runtime events
- TAUcuda measurements (GPU)
  - CUDA kernel performance profiles
    - exclusive/inclusive, nested
  - Kernel execution time data
  - CPU waiting time and “finalize” time
- Works with multiple CPUs and GPUs

Case Study: Conway's Game of Life (GoL)



- GoL is a cellular automata that computes on an orthogonal grid of cells each
- Cell has state
  - Life or death
- Next state (life) determined by state of nearest neighbors
- Simple HMPP program
  - One kernel
- Demonstrate and validate HMPP – TAU functionality
  - HMPP events
  - Codelet events
  - TAUcuda events
  - Effects of different problem sizes

GoL HMPP Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <strings.h>
#include "gol.h"

int main(int argc, char **argv) {
  world_t *w;
  int m,n,iters,seed;
  int last;
  int sz;
  char *wA, *wB;

  if (argc != 5) usage(argv[0]);
  m = atoi(argv[1]);
  n = atoi(argv[2]);
  iters = atoi(argv[3]);
  seed = atoi(argv[4]);
  srand( seed );
  w = newWorld(m,n);
  init(w);
  sz = m*n;
  wA = w->worldA;
  wB = w->worldB;

#pragma hmpp iterate callsite, args[0-1].size={sz}, &
#pragma hmpp iterate           args[4].size={1}
  iterate(wA,wB,m,n,&last,iters,0,m,0,n);

  destroyWorld(w);
  exit(EXIT_SUCCESS);
}
```

```
#pragma hmpp iterate codelet, args[0-1].io = inout, &
#pragma hmpp iterate          args[2-3].io = in,    &
#pragma hmpp iterate          args[4].io   = inout, &
#pragma hmpp iterate          args[5-9].io = in,    &
#pragma hmpp iterate          target=CUDA
void iterate(char *wA, char *wB, int n, int m, int *last,
             int iters, int m_lo, int m_hi, int n_lo, int n_hi) {
  int i,j,iter;
  char *cur, *prev, *tmp;

  cur = wB; prev = wA; *last = iters%2;
  for (iter=0;iter<iters;iter++) {

#pragma hmppcg parallel
    for (i=m_lo;i<m_hi;i++) {
      for (j=n_lo;j<n_hi;j++) {
        int sum =
          prev[MODIDX(j  ,i+1,n,m)] +
          prev[MODIDX(j+1,i+1,n,m)] +
          prev[MODIDX(j-1,i+1,n,m)] +
          prev[MODIDX(j  ,i-1,n,m)] +
          prev[MODIDX(j+1,i-1,n,m)] +
          prev[MODIDX(j-1,i-1,n,m)] +
          prev[MODIDX(j+1,i  ,n,m)] +
          prev[MODIDX(j-1,i  ,n,m)];
        if (prev[IDX(j,i,n)] == 0 && sum == 3)
          cur[IDX(j,i,n)] = 1;
        else {
          if (sum < 2 || sum > 3) cur[IDX(j,i,n)] = 0;
          else cur[IDX(j,i,n)] = prev[IDX(j,i,n)];
    }}}

#pragma hmppcg parallel
    for (i=m_lo;i<m_hi;i++) {
      for (j=n_lo;j<n_hi;j++) {
        prev[IDX(j,i,n)] = cur[IDX(j,i,n)];
}}}}
```

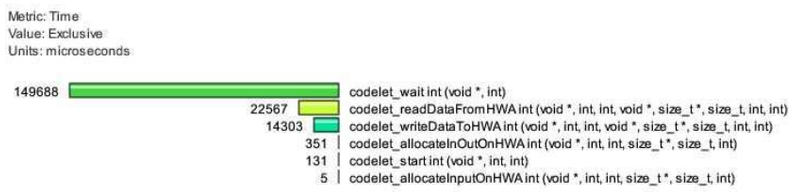
Loop 1

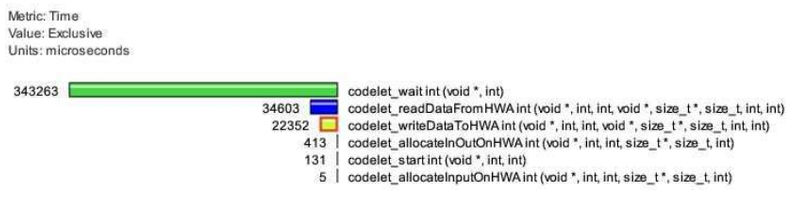
Loop 2

GoL Scaling Experiments

- TAU events: HMPP and codelet events
- Problem sizes: 1000x1000, …, 5000x5000

TAU Event	calls	Computation time(ms) for varying input matrix size				
		1000X1000	2000X2000	3000X3000	4000X4000	5000X5000
hmppStartCodelet	1	7931	25506	95711	152348	345741
hmppGetAvailableHWA	1	2765859	2702177	2705051	2769246	2714853
hmppReleaseDevice	1	50235	2732	2831	44236	2923
hmppWriteDataToHWA	10	1318	1596	2064	2694	3488
hmppAllocateInputOnHWA	7	1234	1235	1234	1241	1277
hmppReadDataFromHWA	3	1718	3261	5648	8894	12811
hmppAllocateInOutOnHWA	3	1218	1357	1256	1302	1277
hmppStartHMPP	1	9	11	11	12	11
codelet_wait	1	5566	22925	93184	149688	343263
codelet_readDataFromHWA	3	590	2021	4320	7522	11534
codelet_writeDataToHWA	10	121	371	808	1430	2235
codelet_allocateInOutOnHWA	3	81	88	99	117	138
codelet_start	1	104	127	134	131	131
codelet_allocateInputOnHWA	7	0	0	1	1	1

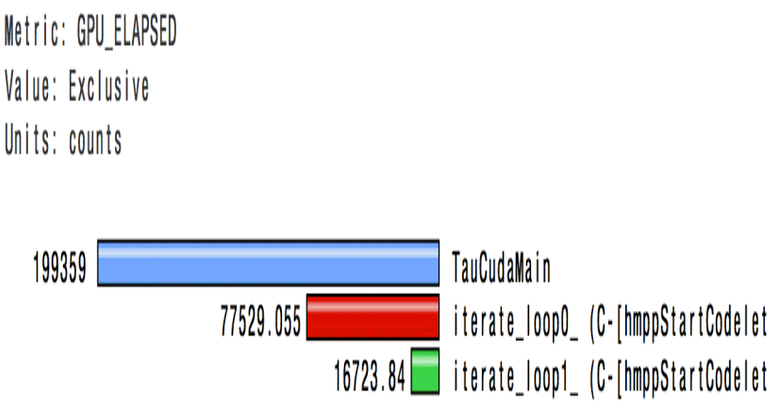


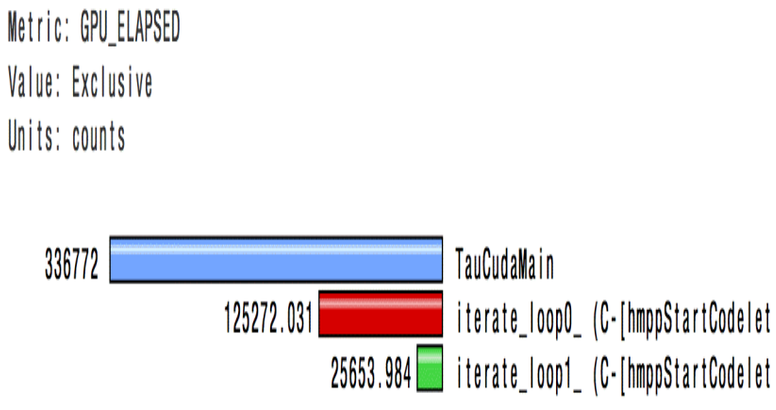


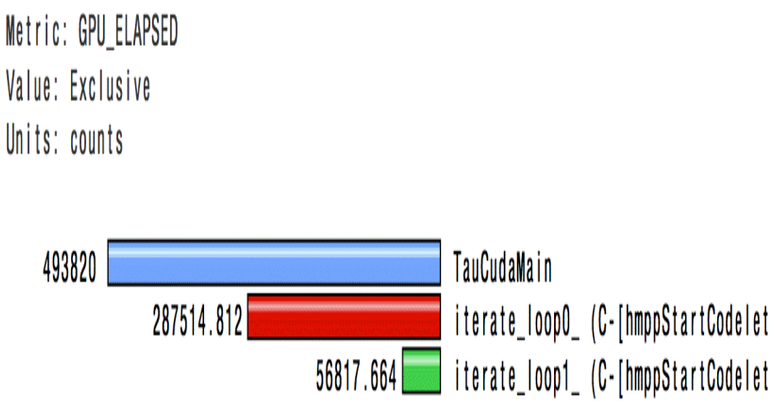
GoL Scaling Experiments (2)

- TAUcuda events
- TAU Context: hmppStartCodelet

Input Matrix Size	Accelerator	Inclusive Time	Exclusive Time	Wait Time	Finalize Time
1000X1000	iterate_loop0_	5389.0562	5389.0562	0.000	6702.9951
1000X1000	iterate_loop1_	1280.4160	1280.4160	0.000	6670.0059
2000X2000	iterate_loop0_	20043.4238	20043.4238	0.000	24202.0039
2000X2000	iterate_loop1_	4121.6001	4121.6001	0.000	24161.9941
3000X3000	iterate_loop0_	77529.0547	77529.0547	0.000	94290.0000
3000X3000	iterate_loop1_	16723.8398	16723.8398	0.000	94247.0000
4000X4000	iterate_loop0_	125272.0312	125272.0312	0.000	150963.9844
4000X4000	iterate_loop1_	25653.9844	25653.9844	0.000	150922.0156
5000X5000	iterate_loop0_	287514.8125	287514.8125	0.000	344370.0000
5000X5000	iterate_loop1_	56817.6641	56817.6641	0.000	344327.0000

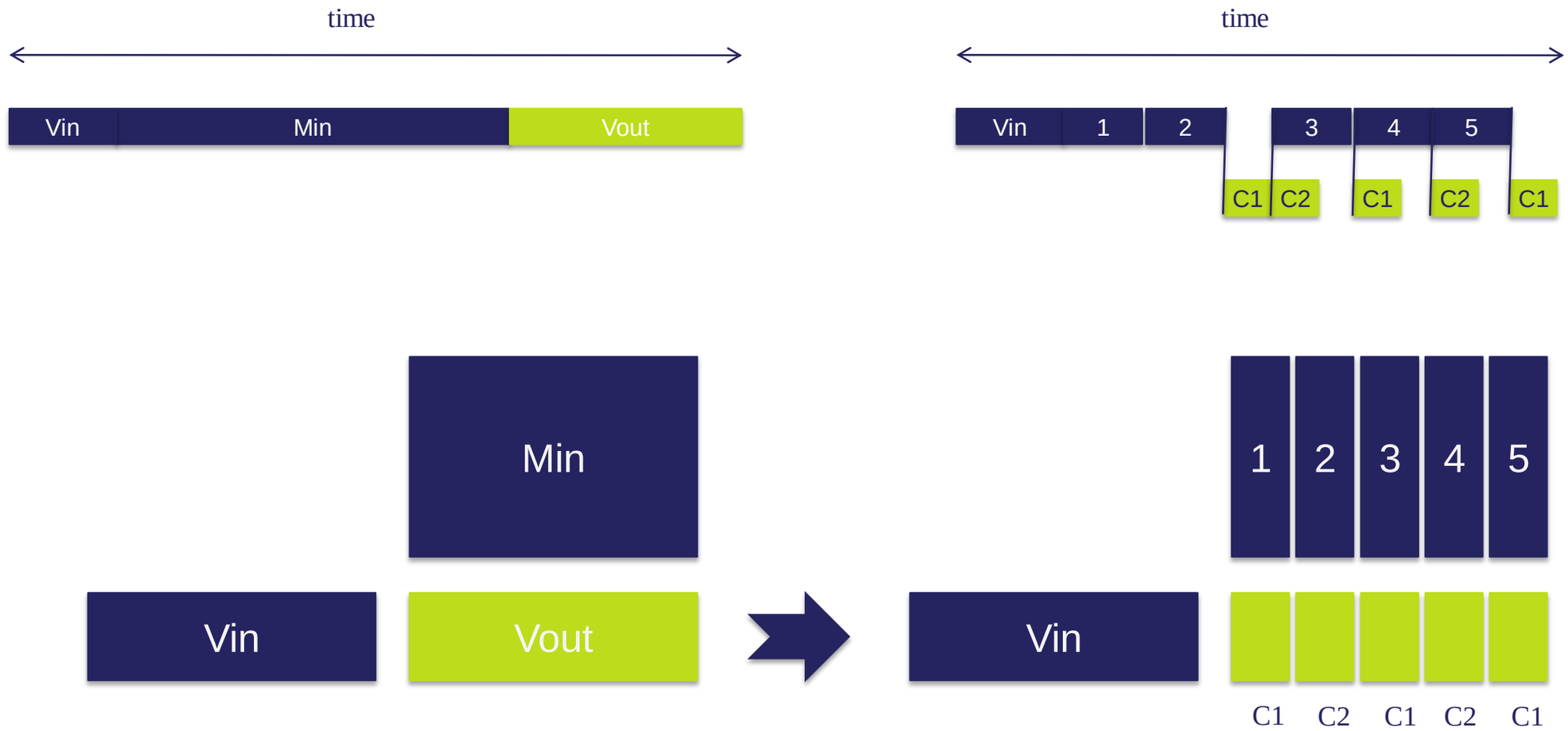




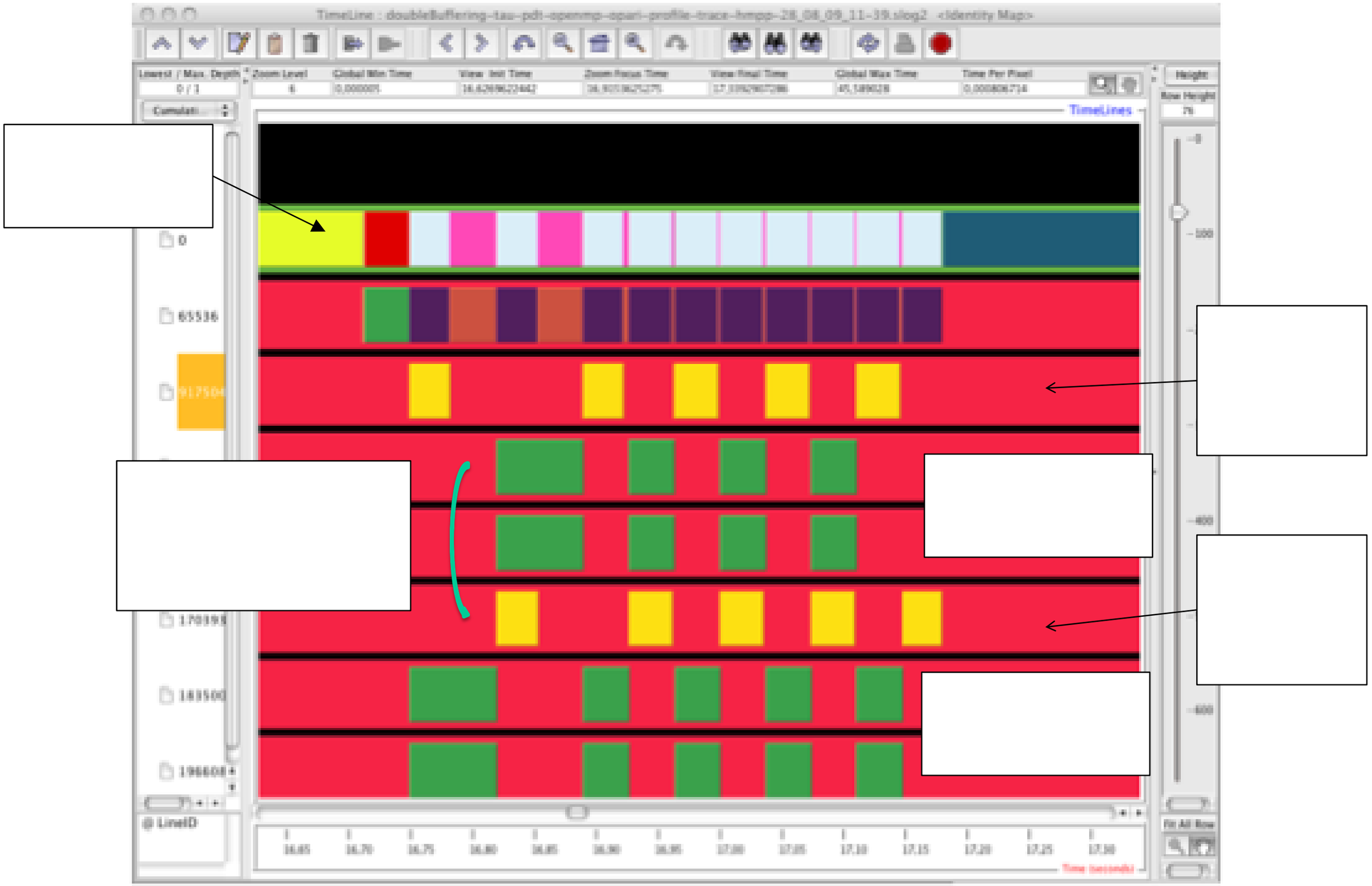


Case Study: Benefit of Data/Execution Overlap

- Matrix-vector multiplication
  - Single codelet sequentializes data transfer and execution
  - Two codelets allow overlap
- Demonstrate all levels of events and profiling+tracing



Data/Overlap Experiment (Execution Trace)



Data/Overlap Experiment (Execution Profile)

TAU: ParaProf: Standard Deviation Data - /amd/tilion/nfs/home/morinl/trav

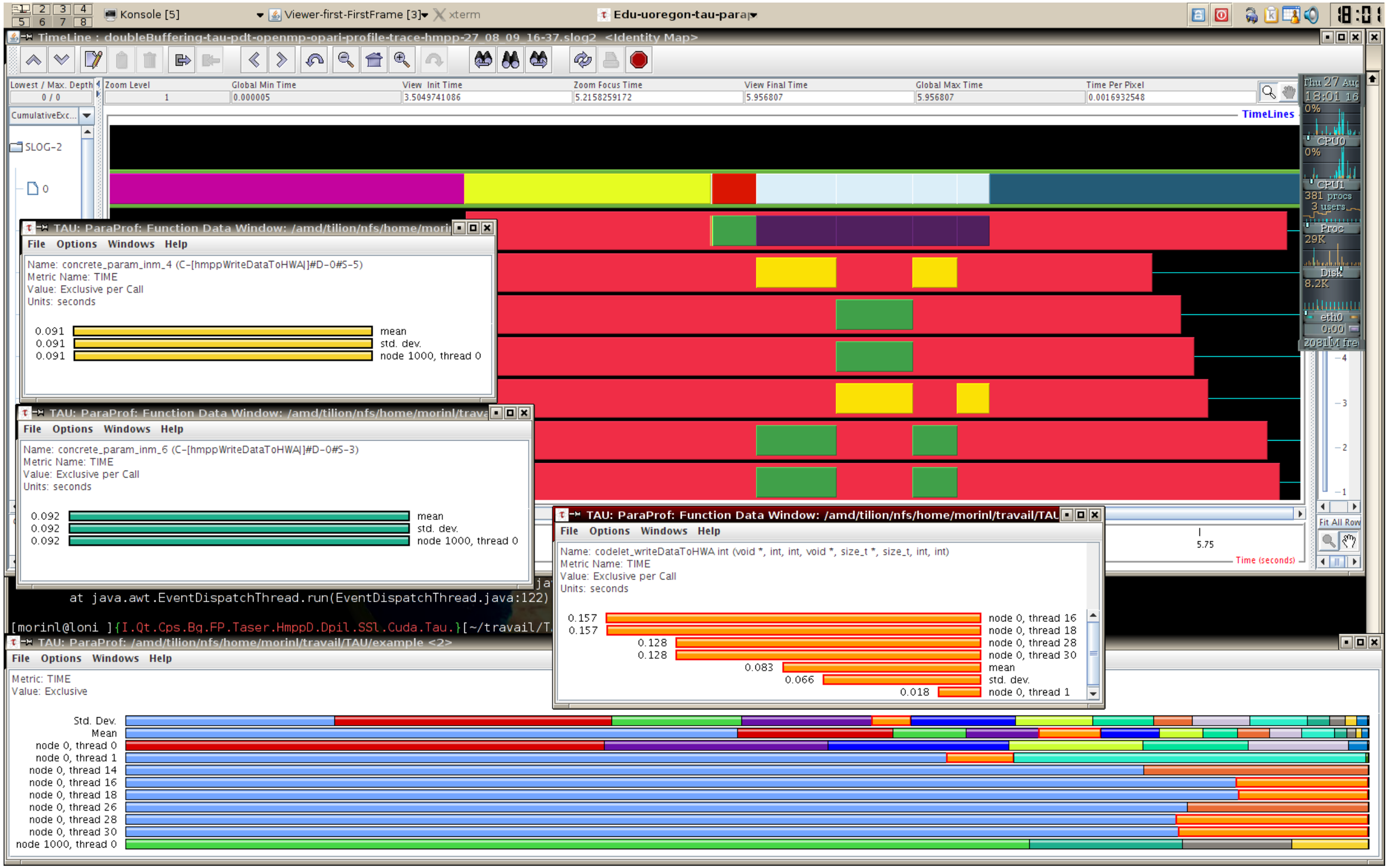
File Options Windows Help

Metric: TIME
Value: Exclusive
Units: seconds

0.721	void *thealloc(size_t, size_t) C [{test_doubleBuffuring
0.545	codelet_thread int (void *, int, int, size_t *, size_t, in
0.34	TauCudaMain
0.337	void initData(int, int, real *, real (*)[*], real *, real *)
0.273	void matvec(int, int, real *, real *, real (*)[*]) C [{test
0.201	hmppReleaseDevice int (int *)
0.159	hmppGetAvailableHWA int (int *, const char *, int *)
0.15	hmppWaitForCompletion int (int *, int *)
0.15	codelet_wait int (void *, int)
0.103	codelet_writeDataToHWA int (void *, int, int, void *,
0.1	codelet_start int (void *, int, int)
0.058	concrete_param_inm_6 (C-[hmppWriteDataToHWA|
0.041	matvec_simple_f2_loop1_ (C-[hmppStartCodelet|]#D
0.029	concrete_param_inm_4 (C-[hmppWriteDataToHWA|
0.029	hmppWriteDataToHWA int (int *, int *, int *, void *, i
0.001	hmppAllocateInputOnHWA int (int *, int *, int *, size_
0.001	codelet_allocateInputOnHWA int (void *, int, int, size
6.1E-4	int main(int, char **) C [{test_doubleBuffuring.chk.pc
1.4E-4	hmppReadDataFromHWA int (int *, int *, int *, void *
1.3E-4	codelet_readDataFromHWA int (void *, int, int, void *
8.2E-5	concrete_param_outv_5 (C-[hmppWriteDataToHWA|
5.8E-5	hmppAllocateInOutOnHWA int (int *, int *, int *, size_
5.4E-5	codelet_allocateInOutOnHWA int (void *, int, int, size
4.7E-5	hmppStartCodelet int (int *, int *, int *)
4.5E-5	concrete_param_outv_3 (C-[hmppWriteDataToHWA|
1.2E-5	void allocData(int, int, real **, real **, real **) C [{te
8.8E-6	hmppWaitForWriteTransfer int (int *, int *, int *)

TAUcuda events

Data/Execution Overlap (Full Environment)



Conclusions and Future Work

- MCH programming environments must integrate parallel performance technology to enable performance analysis
- Initial HMPP-TAU integration with TAUcuda (alpha version)
  - Two week intense effort to build prototype
  - Automatic instrumentation implemented in HMPP Workbench
  - Demonstrated benefits of performance tool integration
- Improve approach for TAU management of HMPP threads
- Better merge TAU and TAUcuda performance data
- Take advantage of other tools in TAU toolset
  - Performance database (PerfDMF), data mining (PerfExplorer)
- Extend to OpenCL codelets as TAUcuda makes available
- Real applications (e.g., seismic modeling, neuroinformatics)