Nixing Scale Bugs in HPC

Saeed Taheri and Ganesh Gopalakrishnan (University of Utah, Salt Lake City, UT)
Sindhu Devale and Martin Burtscher (Texas State University, San Marcos, TX)
www.cs.utah.edu/fv/HybridDebugger

THE UNIVERSITY OF UTAH

TEXAS STATE UNIVERSITY
The rising STAR of Texas

XPS Exploratory Award CCF 1438963, 1439002

Objectives

Challenges of HPC Software Debugging Include

- Heterogeneity: hardware (CPUs, GPUs, …)
- Extreme scale: number of threads/cores
- Rapid evolution: new CPUs/GPUs/libraries
- Reality: High manual effort to annotate code
- Reality: No tools that collect enough debugging information per large-scale run with low overhead (Service Units or Core Hours get exhausted, precluding "second run")

Wish List

A tool for gathering information from HPC software that maintains a history of events and their causal relationships such that, upon failure detection, one can query and navigate the history to narrow down the likely cause of the bug.

Challenges in Realizing These Goals

- **Scalability**
  - Detecting memory corruption (e.g., out of bounds access) and data races requires heavy-weight instrumentation
- **Solution**
  - For now, focus on synchronization/control bugs (deadlocks, livelocks)
  - Higher degrees of scrutiny on relatively newer pieces of code

- **Handling Heterogeneity**
  - Tracking control-flow / happens-before across different types of execution hardware has not been addressed before

- **Solution**
  - Develop synchronization action collection methods across CPUs, GPUs, and code written under different concurrency models

Prior Work (exemplars)

Commercial Debuggers

- Very good at detailed trace collection
- Good at minutely examining execution state
- Poor at handling scale
- Little help toward identifying the root cause
- Little help bridging concurrency models

Research Lab Tools

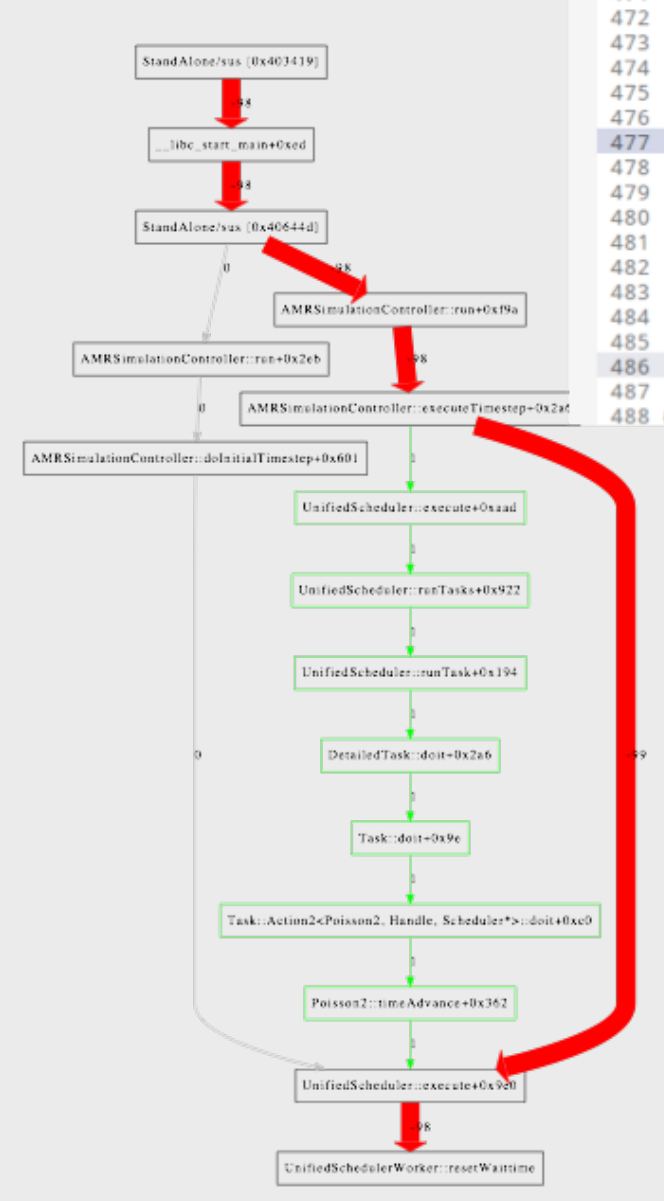
- Stack trace collection based
  - Often for MPI
- Progress tracking
  - Often based on loop progress order
- Example : STAT, AutomaDeD, Protometer (LLNL), Dynoptic (UW)
- Do not exploit behavioral differences

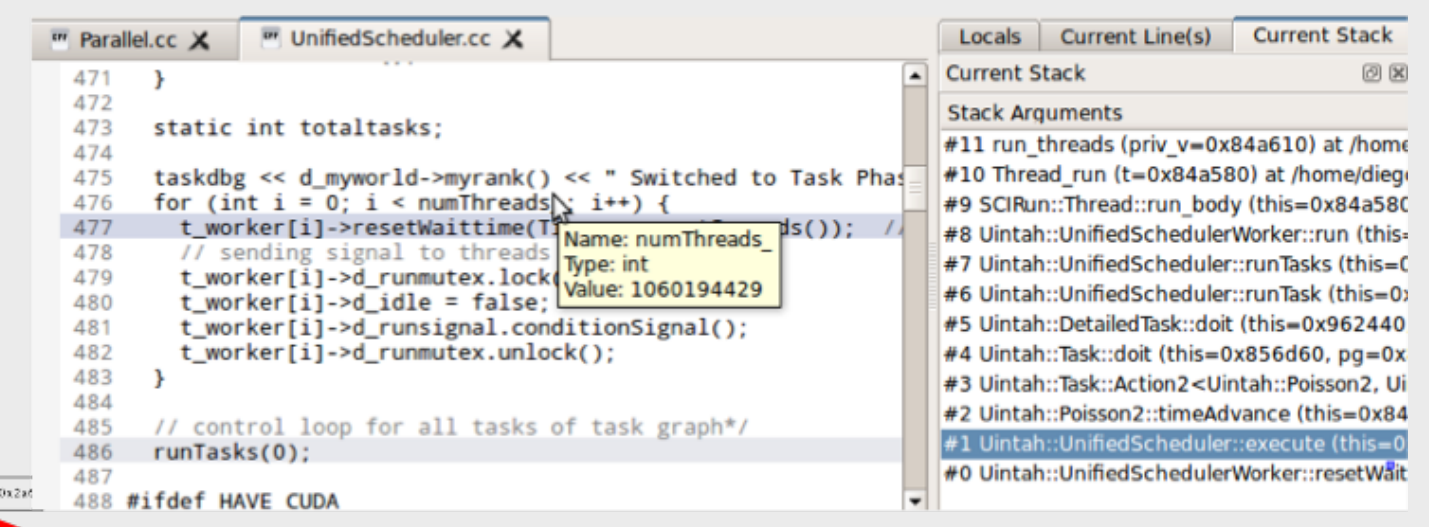
Use of Coalesced Stack Trace Graphs

- Proven useful in large code base
- Summarizes stack nests
- Does not handle heterogeneity
- While overhead is low, it was not focused toward synchronizations across multiple concurrency models

Example use of Coalesced Stack Trace Graphs in detecting uninitialized variables in Uintah code base (University of Utah; see LCPC 2014)

Nondeterminism due to Uninitialized Variable (Poisson2 Example)


StandAlone/sus [0x403419]
__libc_start_main+0xed
StandAlone/sus [0x40644d]
AMRSimulationController::run+0xf9a
AMRSimulationController::run+0x2eb
AMRSimulationController::executeTimestep+0x2a6
AMRSimulationController::doInitialTimestep+0x601
UnifiedScheduler::execute+0xaad
UnifiedScheduler::runTasks+0x922
UnifiedScheduler::runTask+0x194
DetailedTask::doit+0x2a6
Task::doit+0x9e
Task::Action2<Poisson2, Handle, Scheduler*>::doit+0xe0
Poisson2::timeAdvance+0x362
UnifiedScheduler::execute+0x9e0
UnifiedSchedulerWorker::resetWaittime
98
0
99



```
471   }
472
473   static int totaltasks;
474
475   taskdbg << d_myworld->myrank() << " Switched to Task Phas
476   for (int i = 0; i < numThreads_; i++) {
477     t_worker[i]->resetWaittime(Ti                  ds()); /
478     // sending signal to threads
479     t_worker[i]->d_runmutex.lock(
480     t_worker[i]->d_idle = false;
481     t_worker[i]->d_runsignal.conditionSignal();
482     t_worker[i]->d_runmutex.unlock();
483   }
484
485   // control loop for all tasks of task graph*/
486   runTasks(0);
487
488 #ifdef HAVE_CUDA
```

Name: numThreads_
Type: int
Value: 1060194429

Current Stack
Stack Arguments

```
#11 run_threads (priv_v=0x84a610) at /home
#10 Thread_run (t=0x84a580) at /home/dieg
#9 SCIRun::Thread::run_body (this=0x84a580
#8 Uintah::UnifiedSchedulerWorker::run (this=
#7 Uintah::UnifiedScheduler::runTasks (this=0
#6 Uintah::UnifiedScheduler::runTask (this=0x
#5 Uintah::DetailedTask::doit (this=0x962440
#4 Uintah::Task::doit (this=0x856d60, pg=0x
#3 Uintah::Task::Action2<Uintah::Poisson2, Ui
#2 Uintah::Poisson2::timeAdvance (this=0x84
#1 Uintah::UnifiedScheduler::execute (this=0
#0 Uintah::UnifiedSchedulerWorker::resetWait
```

Bug was located by comparing runs over two different periods of time.

Ongoing Work

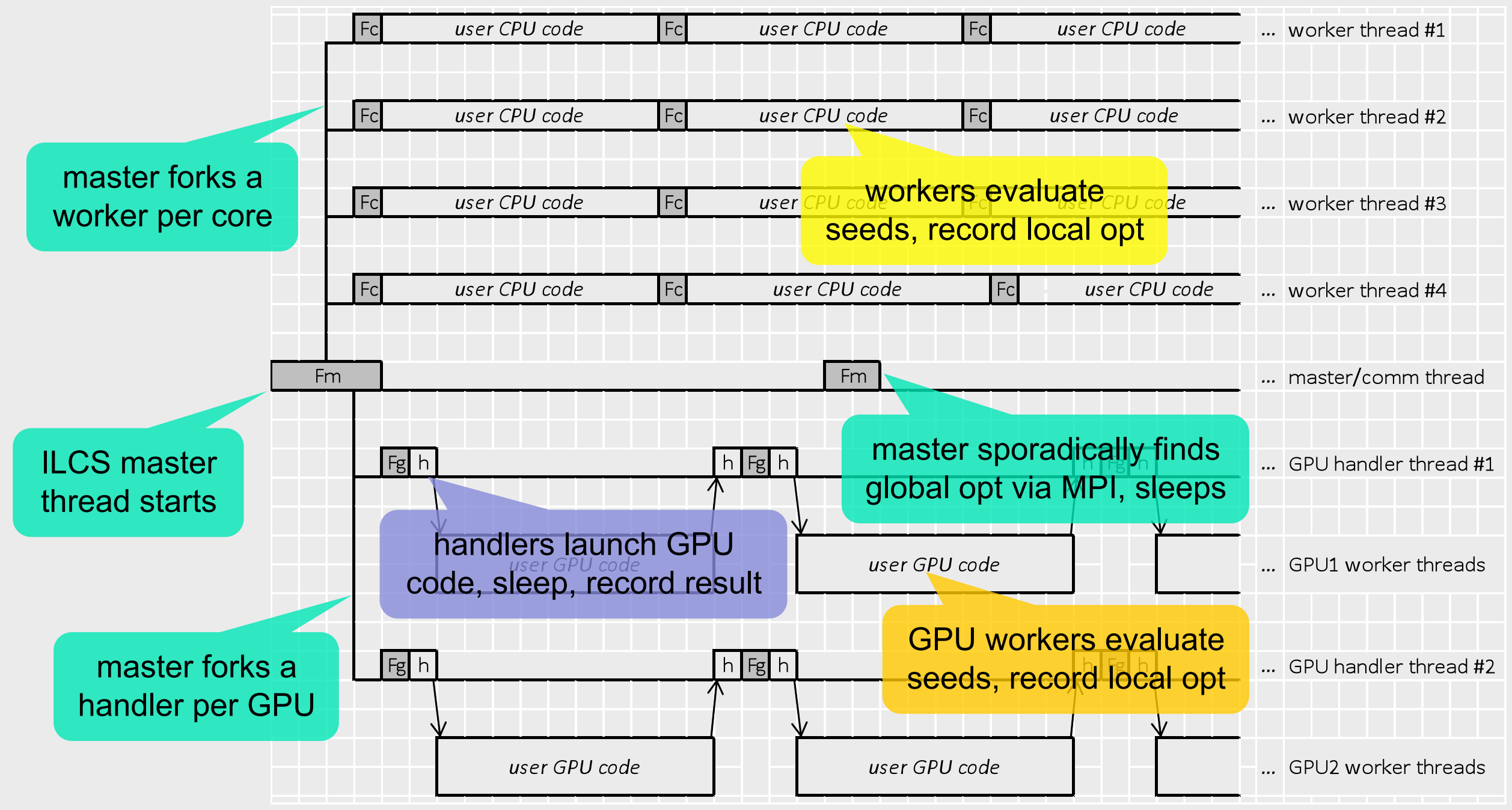
Case Study: ILCS

- A heterogeneous concurrent program called the Iterative Local Champion Search (ILCS) has been chosen

ILCS combines three flavors of concurrency

- MPI
- OpenMP
- GPU

Execution Flow of ILCS



master forks a worker per core

workers evaluate seeds, record local opt

ILCS master thread starts

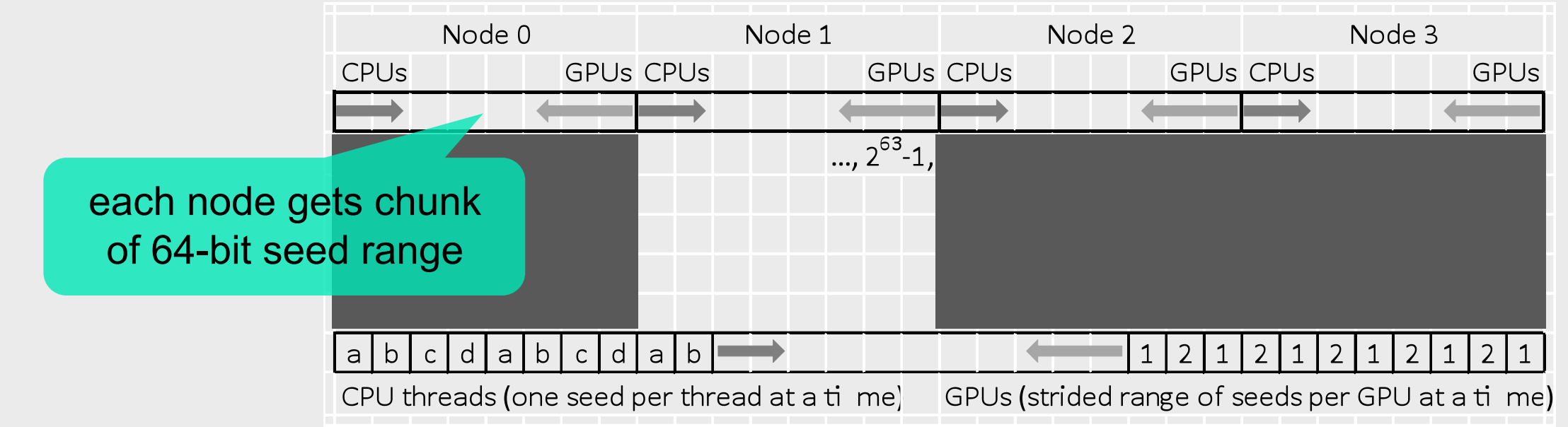
master sporadically finds global opt via MPI, sleeps

handlers launch GPU code, sleep, record result

master forks a handler per GPU

GPU workers evaluate seeds, record local opt

A Scalable Heterogeneous Parallelization Framework for Iterative Local Searches


Node 0
Node 1
Node 2
Node 3
CPUs
GPUs
…, 2^{63}-1,
a b c d a b c d a b
1 2 1 2 1 2 1 2 1 2 1
CPU threads (one seed per thread at a ti me)
GPUs (strided range of seeds per GPU at a ti me)

each node gets chunk of 64-bit seed range

Largest Scale Tested

system	compute nodes	total CPUs	total GPUs	total CPU cores	total GPU cores
Keeneland	128	256	384	2048	196,608
Ranger	2048	8192	0	32768	0
Stampede	1024	2048	0	16384	0

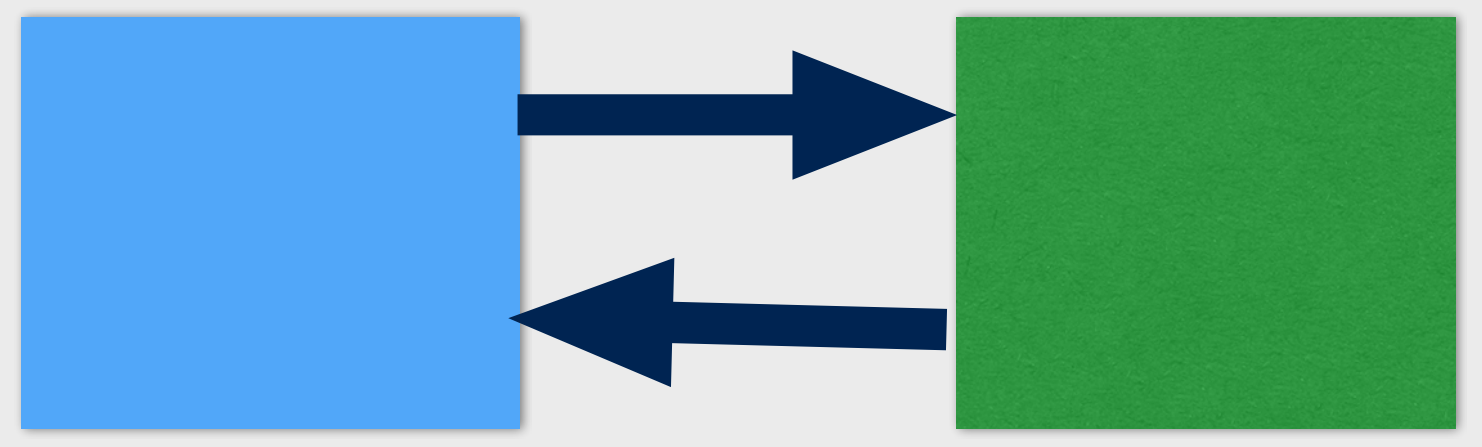
Other Case Studies Planned

- A Rigorous Global Optimizer for Floating-Point Precision Estimation
- Parallelized versions of a GPU Data Race Checker
- Using parallel execution frameworks to parallelize verification

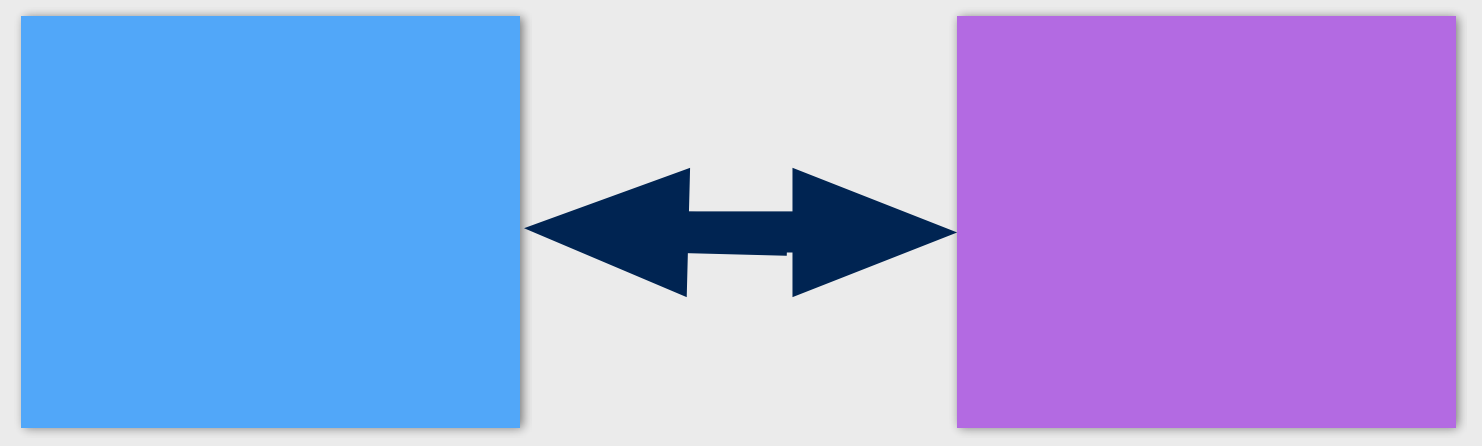
Merit, Impact, Milestones, Students

Work in progress

1) CPU ⟷ GPU control tracing methods



2) CPU ⟷ XeonPhi control tracing methods



3) Binary Instrumentation Methods for Heterogeneous Trace Collection

Intellectual Advances, Broader Impact

- Ways to track "happens before" at scale
- Ways to mine the tracked information for finding the root causes of bugs
- Enabling science at scale
  - Achieving Extreme Scale requires the use of powerful debugging tools
- An understanding of what information to collect, how collection scales, and how it facilitates debugging
- Effective utilization of HPC resources
- System heterogeneity is only bound to increase — impactful beyond HPC

Anticipated Tools and Utilities

- Control tracing utilities for heterogeneous concurrency
- A graphical tool for querying collected control dependencies for debugging

Student Training

- Saeed Taheri, PhD
- Sindhu Devale, MS
- Kurstie Lenear (BS, graduated)

Collaborations with the following PIs and researchers is gratefully acknowledged: Hassan Rabeti (ILCS), Zvonimir Rakamaric' (Debugging Methods). The CSTG example is from a paper authored with Berzins, Humphrey, de Oliviera, Meng, Rakamaric' and Gopalakrishnan. ILCS images are from a paper authored by Burtscher and Rabeti (IPDPS'13).