ADVANCED PARALLEL COMPUTING 2017
LECTURE 05 - SYNCHRONIZATION 2

Holger Fröning
holger.froening@ziti.uni-heidelberg.de
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg

Some material by Falsafi, Hardavellas, Nowatzyk of EPFL, Northwestern, CMU

BARRIERS

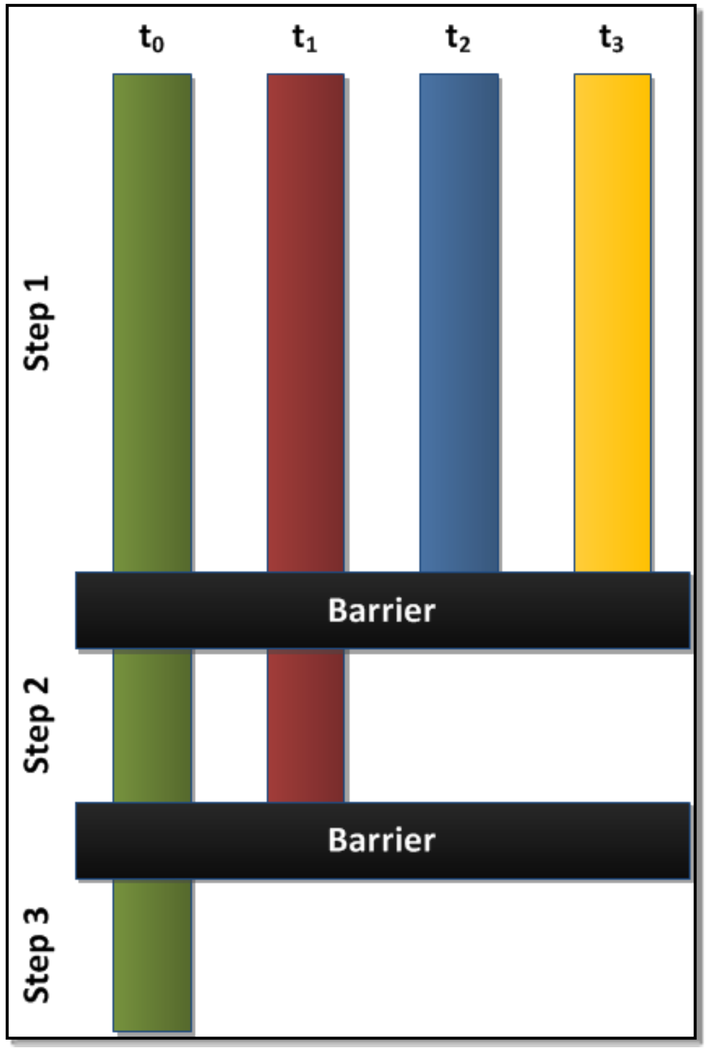
WHY DO WE NEED BARRIERS?

What is a barrier?

- Any participating thread has to wait until all threads have reached the barrier

Why do we need such coarse-grain (collective) synchronization?

- Any kind of lock-step computation
- Physics simulation computation
  - Divide up the computation of each time step into N independent pieces
  - Each time step: compute independently, synchronize
- Essential for BSP-like programming
- Stencil codes
- Six-step FFT method



Barrier-based Merge-Sort

BARRIERS VS. LOCKS

Lock

Threads are waiting because they have something to do

A lock master is usually not viable

- No thread/core may be free to take over this part
- Contention would quickly turn the master into a bottleneck

Barrier

Threads are rather waiting because they have finished their work and wait for their external dependencies to be resolved

A barrier master can be viable

- At least one thread is waiting anyway

GLOBAL SYNCHRONIZATION BARRIER

At a barrier, all threads wait until all other threads have reached it

- Strawman implementation (wrong)
- So what's wrong?

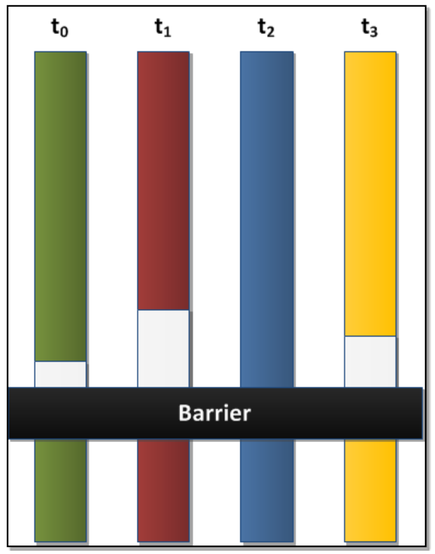
Race condition

- After resetting counter to P, before all threads are notified, another thread could again decrement counter

```
(shared) int count = P;

void strawman_barrier ()
  {
  if ( fetch_and_dec ( &count ) == 1 )
    count = P;
  else
    while ( count != P );
      //spin wait
  }
```

```
...
do_work_1 ();
strawman_barrier ();
do_work_2 ();
strawman_barrier ();
...
```



FUNCTIONALLY CORRECT BARRIER "SENSE-REVERSING"

```
(shared) int count = P;
(shared) bool sense = true;
(local) bool local_sense = true;

void sense_barrier ()
  {
  // each processor toggles its own sense
  local_sense = !local_sense;

  if ( fetch_and_dec ( &count ) == 1 )
    count = P;
    // last processor toggles global sense
    sense = local_sense;
  else
    while ( sense != local_sense ); //spin wait
  }
```

ISSUES WITH CENTRALIZED BARRIERS

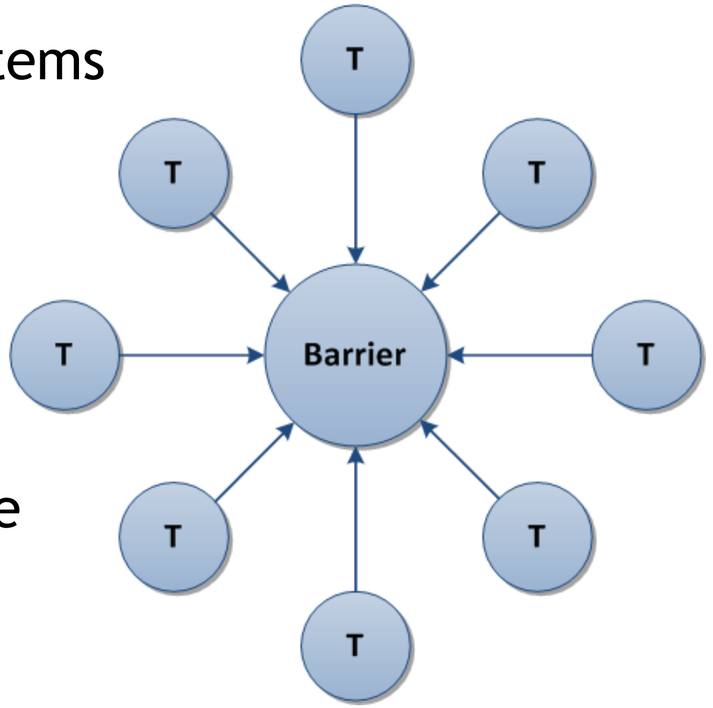
Single counter makes the sense-reversing barrier a centralized barrier

- Highly suitable for cache-coherent shared memory systems
- Cache coherence & polling on sense:
  uniform notification time (hopefully)

Problem with centralized barrier

- All processors must increment the counter
- Each RMW is a serialized coherence action, i.e. a cache miss
- O(n) threads arrive simultaneously, slow for lots of processors

=> Memory contention



COMBINING TREE BARRIER

Combining tree barrier: build a $\log_k(n)$ height tree of counters (one per cache block)

- k = radix
- Each thread coordinates with (k-1) other threads (i.e., by thread ID)
- Last of the k threads coordinates with next higher node in tree (no tournament - statically determined)
- As many coordination addresses are used, misses are not serialized (O (log n) in best case)

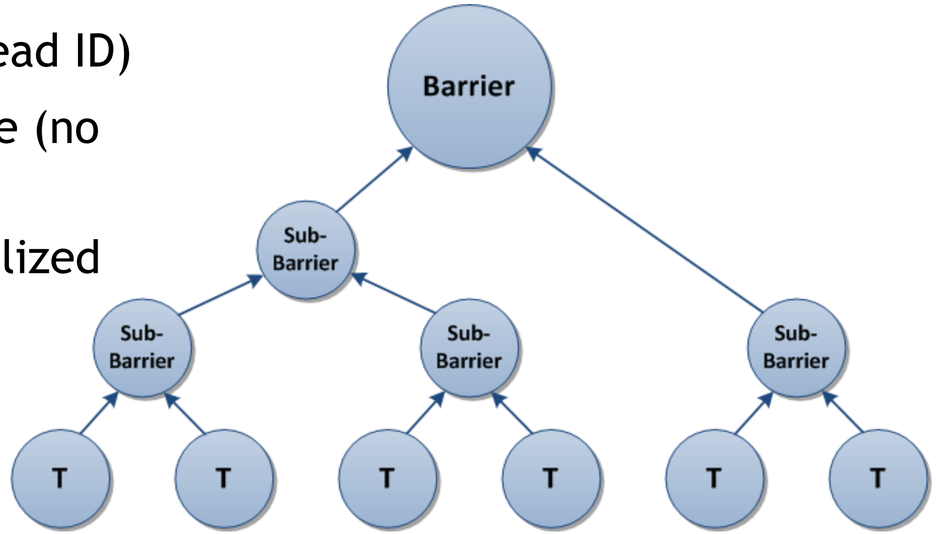
Spread memory accesses across multiple barriers

=> Reduce contention

- What is faster, decrement a single location or visiting a logarithmic number of barriers?

=> At the cost of increased latency

- Eventually



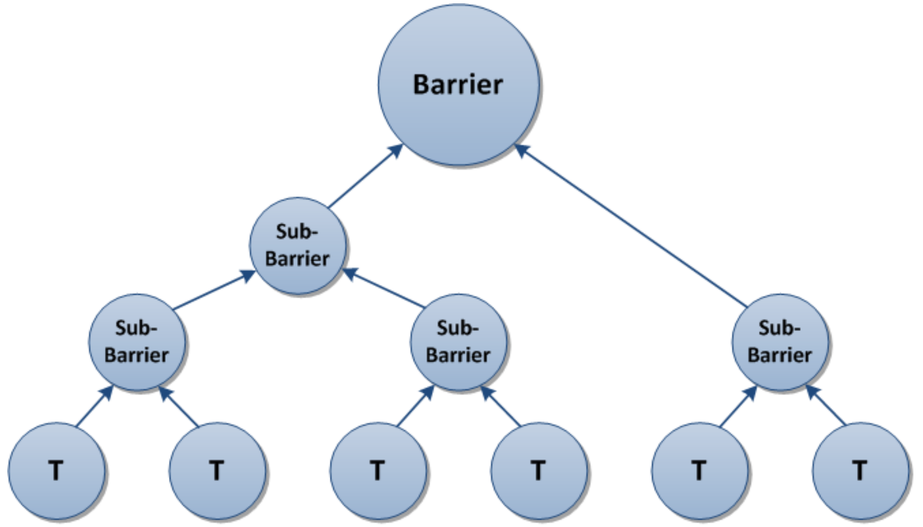
OTHER BARRIER IMPLEMENTATIONS - SW

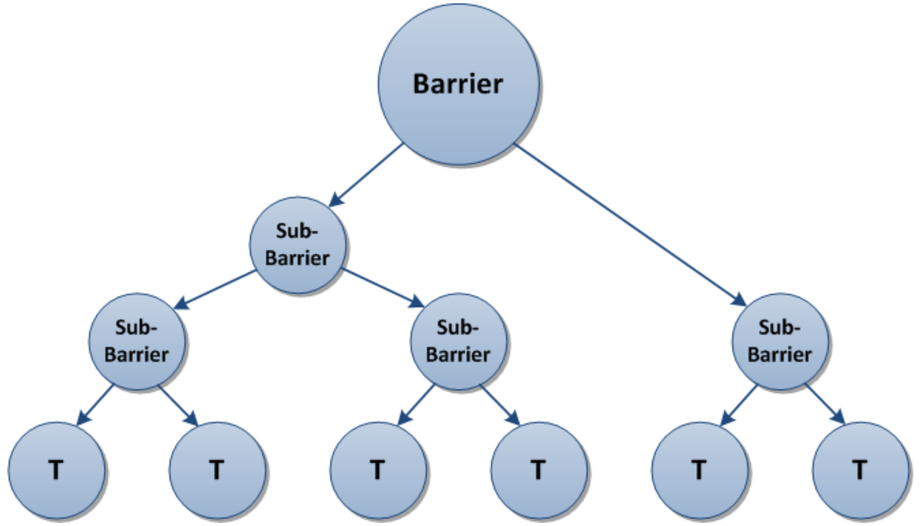
Separate barrier operation into two phases: arrival and release

Various optimizations

- Centralized or (software) combining tree
  - Different fan-outs, even different for arrival/release tree
- Dissemination for local spinning
- Tournament to avoid RMW

And these are only the software implementations





OTHER BARRIER IMPLEMENTATIONS - HW

In principle: AND trees (possibly even hard-wired)

Issues

- No one wants a second dedicated network
- Virtualization: support for more than one concurrent barrier

Proposals & Implementations

- Syncbits associated with each cache block
  - J. Goodman et al., “Efficient Synchronization Primitives for Large- Scale Cache-Coherent Shared-Memory Multiprocessors”, ASPLOS1989
- E-Registers & synchronization units (32, memory-mapped) (Cray T3E)
  - S.L. Scott, “Synchronization and Communication in the T3E Multiprocessor”, ASPLOS1996
- Starving cache line requests (postpone responses until barrier condition met)
  - J. Sampson et al., “Exploiting Fine-Grained Data Parallelism with Chip Multiprocessors and Fast Barriers”, MICRO2006
- GBarrier for many-core CMPs (G-Line based dedicated network)
  - J. L. Abellán, J. Fernández and M. E. Acacio, "Efficient Hardware Barrier Synchronization in Many-Core CMPs”, TPDS2012

BARRIER PERFORMANCE

Barrier performance on Symmetry (shared-memory bus-based 80386 system, cache-based)

Network transactions

Dissemination barrier: O (P log P), tournament & tree: O (P), centralized & arrival tree: O(P) if broadcast support, potentially unbounded otherwise

Critical path length

Assuming networks supports parallel transactions

Centralized: O (P), others: O (log P)

Space

Centralized: 1, tree: O (P), others: O (P log P)

Arrival tree

Tree barrier with a sense-reversing flag instead of a release tree

Lower slope than counter as N-1 writes are cheaper than N RMWs

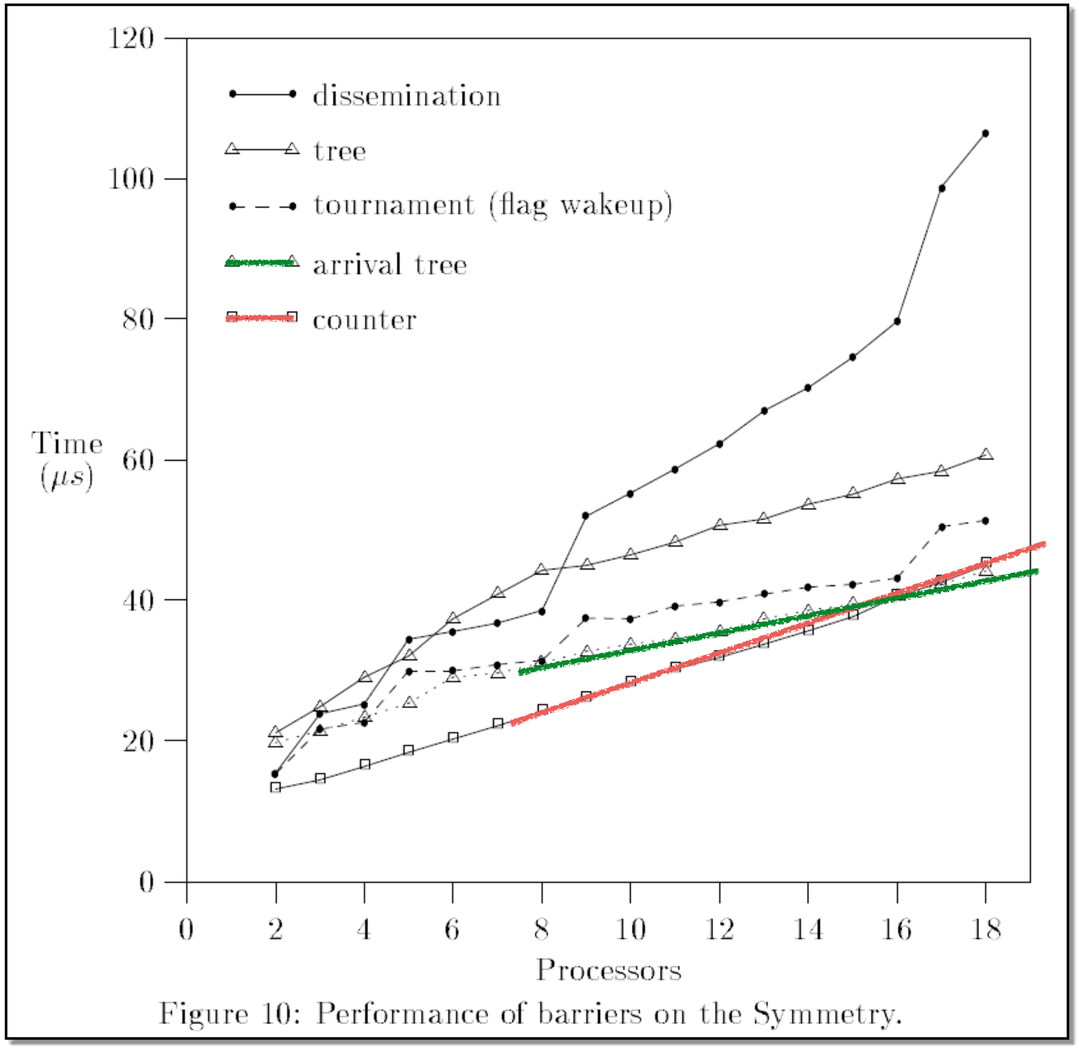


Figure 10: Performance of barriers on the Symmetry.

Mellor-Crummey, Scott, TOCS 1991

BARRIER PERFORMANCE

Barrier over distributed shared memory (MEMSCALE)

- No global coherence but local coherence domains
- Support for remote loads and stores
- No RMW

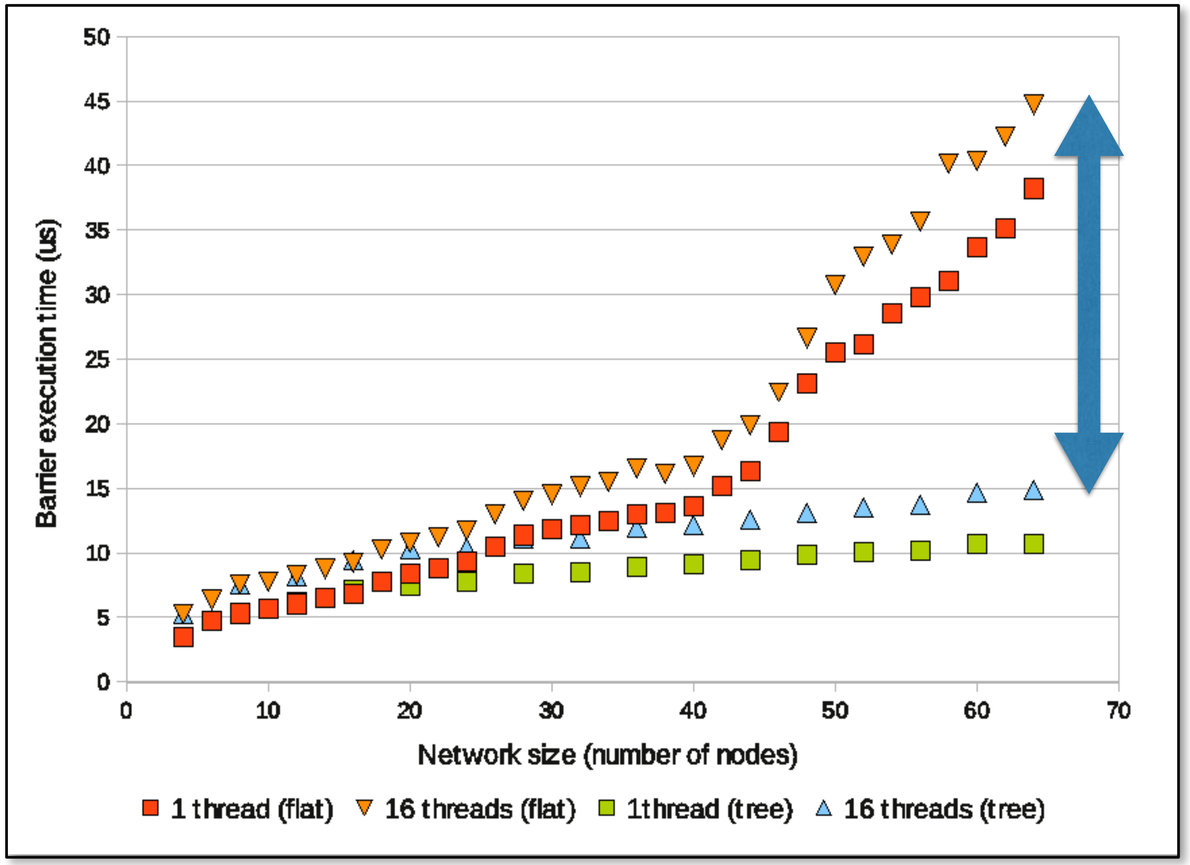
Tree-based: scalability, locality

Tournament: arrays of flags instead of RMWs

Dissemination: push instead of pull

=> 1k threads in 15usec

- 64P @IB: min. 20us



Holger Fröning, Alexander Giese, Hector Montaner, Federico Silla, Jose Duato, Highly Scalable Barriers for Future High-Performance Computing Clusters, 18th annual IEEE International Conference on High Performance Computing (HiPC 2011), 2011, Bangalore, India.

OTHER ASPECTS

Relaxations

- Do you really need to synchronize?
- Stencil codes with halo exchange: Probably at the cost of an increase in iterations
- Training of deep learning: asynchronous weight exchange

Overlap

- Hide barrier latency with communication
- Which model does that?

Jitter

- Large-scale SPMD programs might suffer from varying state
  => increased synchronization time

SUMMARY

Spin Lock

Avoid „Coherence Storms“

Make use of atomic operations if available

- Preferable Compare-and-Swap, or Fetch-and-Store
- Remember the Bakery algorithm as SW alternative

List-based queuing locks usually outperform any other

- In particular true for scalability

HW like QOLB: ROI too low

Barriers

Avoid „Coherence Storms“

Avoiding atomics can be beneficial

Use combining tree to minimize the critical path length, and dissemination to leverage locality effects

- Treat arrival and release structures differently

HW structures

- ROI too low for shared memory
- Potentially reasonable for message passing

PROJECT WORK PROPOSALS

CONSTRUCTION AREA

NVIDIA Fermi-class GPU

- 2 per server, 8 servers

NVIDIA Kepler-class GPU

- 16 per server (1x)
- 1 per server (8x)
- Embedded plattform (Tegra TK1, TX1)

NVIDIA Pascal-class GPU

- 2 per server (1x)

ARM Mali GPU

- Embedded plattform

ARM CPU

- Tegra TK1, TX1
- A couple more…

Micron PicoComputing System

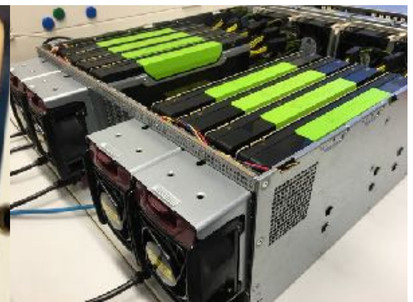
- OpenCL-capable FPGA+HMC

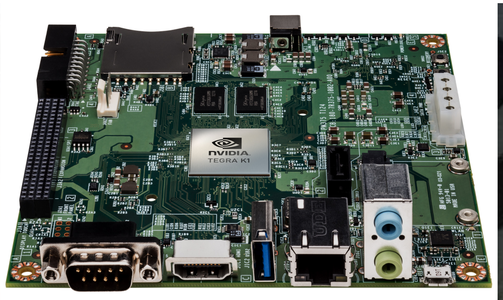
Intel X86

- Sandy Bridge (12 core, 8x)
- Ivy Bridge (16 core, 1x)

Intel MIC

- Possibly, but have to check...





ANALYZING DYNAMIC NUMA EFFECTS: LOAD VERSUS LATENCY

Static NUMA effects: memory access latency variations for an unloaded system

Dynamic NUMA effects: memory access latency variations for a loaded vs unloaded system

- GEMM as a simple load

Architectures

- Multi-GPU
- x86
- ARM

Goal: understand where this is an issue

TESTING GPU MEMORY MODELS

GPU memory models are weak and only informally defined

Approach

- Develop a set of tests (litmus tests)
- Evaluate different GPU architectures

Complex task, but plenty of related work

Goal: test memory models for fun (and profit)

IMPLEMENTING A SCALABLE DATA ANALYSIS FOR EXASCALE APPLICATIONS

Problem: we have more data than we can analyze in reasonable time

- Python and bash scripts
- Traces for MPI Exascale proxy applications: ~1/2TB analyzed, >25TB left

Open questions

- Do we need ordering?
- What is the impact of matching on overall performance?

Spark would be an interesting tool

Impact: huge

- We're already best paper finalist for ISC2017 paper (see website)

EXPLORING UNIFIED MEMORY: PERFORMANCE, ENERGY AND (MAYBE) SCHEDULING IMPLICATIONS

"Downgrade" some CUDA SDK apps to Unified Memory

- Replace cudaMalloc + cudaMemcpy with cudaMallocManaged

Explore performance and energy on NVIDIA Pascal GPUs

- Option: Interact with some multi-GPU guys

Design micro-benchmarks that highlight performance and fallacies

Optionally: prefetch instructions

Goal: understanding + micro-benchmarks

GLOBALLY VALID TIME MEASUREMENT IN CUDA KERNELS

Problem: CUDA GPUs have only a local time (one counter per SM)

- To understand scheduling effects better, a global time is required

Task: Design a method that synchronizes the different clock counters so that the notion of a global time can be established

Architecture

- NVIDIA GPU

TRACING SCHEDULING DECISIONS FOR CUDA APPLICATIONS

Problem: CUDA scheduling is mostly unknown

- Known: greedy-then-oldest, no preemption

Goal: more understanding

Approach: track CTA start/end time and physical SM ID

- Explore how scheduling can be influenced (streams, kernel sharding, kernel fission
- Explore co-scheduling of multiple kernels

Dependencies: previous

IMPACT OF MATRIX MULTIPLY OPTIMIZATIONS ON PORTABILITY

Conventional wisdom: optimizations hurt portability

Approach

- Implement different optimizations for an Open-CL matrix multiply
- Explore their impact on different processors

Architectures

- NVIDIA {Fermi, Kepler, Pascal}, embedded-class & server-class
- x86 CPU
- ARM CPU
- ARM GPU (Mali)

Possibly revert to CUDA

DISSECTING GPU MEMORY HIERARCHY

Recent work has shown that GPU's have huge TLB issues

http://dl.acm.org/citation.cfm?doid=3076113.3076115

In general, understanding GPUs in more detail is important

https://arxiv.org/pdf/1509.02308.pdf

http://www.comp.hkbu.edu.hk/~chxw/gpu_benchmark.html

Goal: Adapt existing benchmarks and possibly come up with new ones

Focus on TLB & address translation

OPTIMIZE LIST-BASED SEARCH FOR MPI MATCHING

MPI Matching is an important problem in HPC communication stacks

- Two lists: unexpected messages, and posted receives
- Ordering guarantees require sequential searching

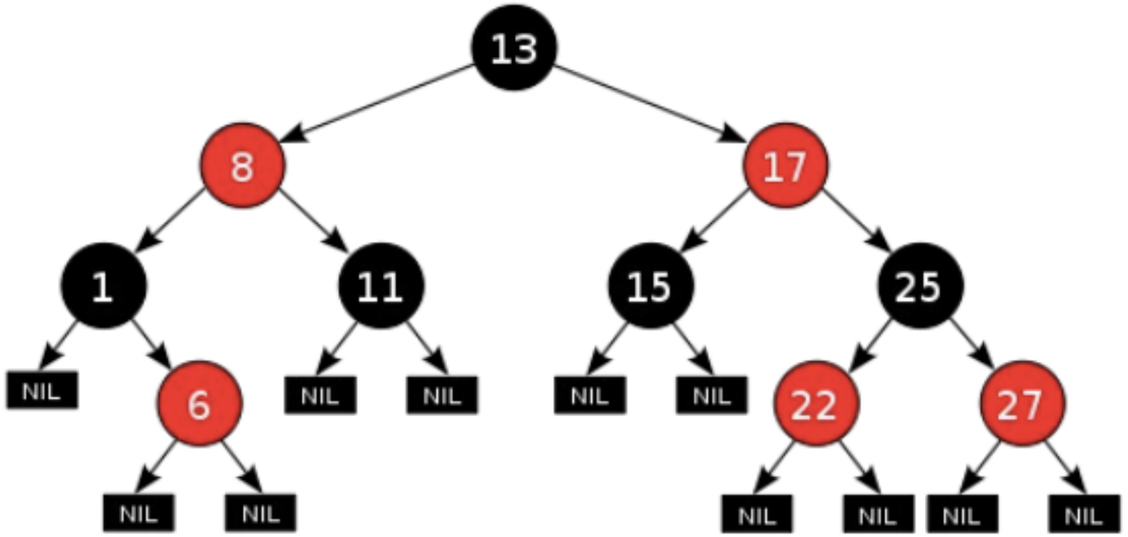
MPI implementations achieve about 25M matches per second, but vary significantly with list length

- Can we understand the different optimizations?
- Can we do better?

OPTIMIZING A RED-BLACK TREE USING LOCKING

Red-Black Tree (RBT) is a form of self-balancing binary search tree

Complex data structure to explore the impact of different locking schemes and methods



POCL

Portable Computing Language (pocl) aims to become a MIT-licensed open source implementation of the OpenCL standard which can be easily adapted for new targets and devices, both for homogeneous CPU and heterogeneous GPUs/accelerators.

https://github.com/pocl/pocl

Goal

- Explore portability
- Possibly using different optimizations of a matrix multiply

Architectures

- x86 CPU
- ARM CPU?
- NVIDIA GPU

MULTI-GPU CONVOLUTION BASED ON THE CUDA SDK VERSION

Addressing the permanent lack of multi-GPU benchmarks

- Convolution is an important operation for ML

Approach

- Start with the CUDA SDK single-GPU version
- Implement and optimize a multi-GPU version

Optionally: replace <convolution> with any other app

NOTES

Anyone eager to work with architecture simulators?

Make up your mind - we’re open for proposals!

Think about aligning this project assignment with future work

- Seminar talk, project work, MSc thesis