Contech Part 3

Outline

- Introduction
- Contech’s Task Graph Representation
- Parallel Program Instrumentation
- (Break)
- **Analysis and Usage of a Contech Task Graph**
- Hands-on Exercises

Rich Analysis Support

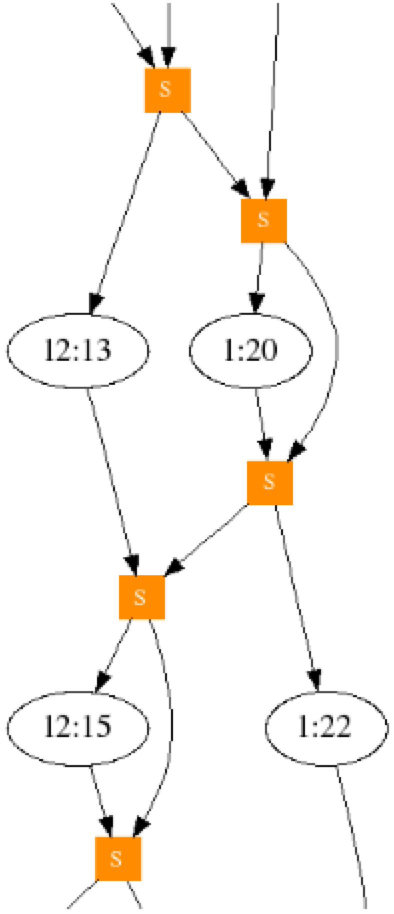
- Compiler-based framework to generate task graphs
- Example analysis (i.e., backends) of Task Graphs
  - Data Race Detection
  - Cache modeling
  - Lock contention

Data Races in Task Graph

- Model memory space and use edges to find happens-before

- Critical Sections
  - Syncs around task 1:20
  - Syncs around task 12:15

- 1:20 -> 12:25 is not a race
- 12:13 -> 1:22 is a race



Data Race Backend

- Similar to Helgrind, Eraser, FastTrack, IFRit, LightRace, PACER, et cetera

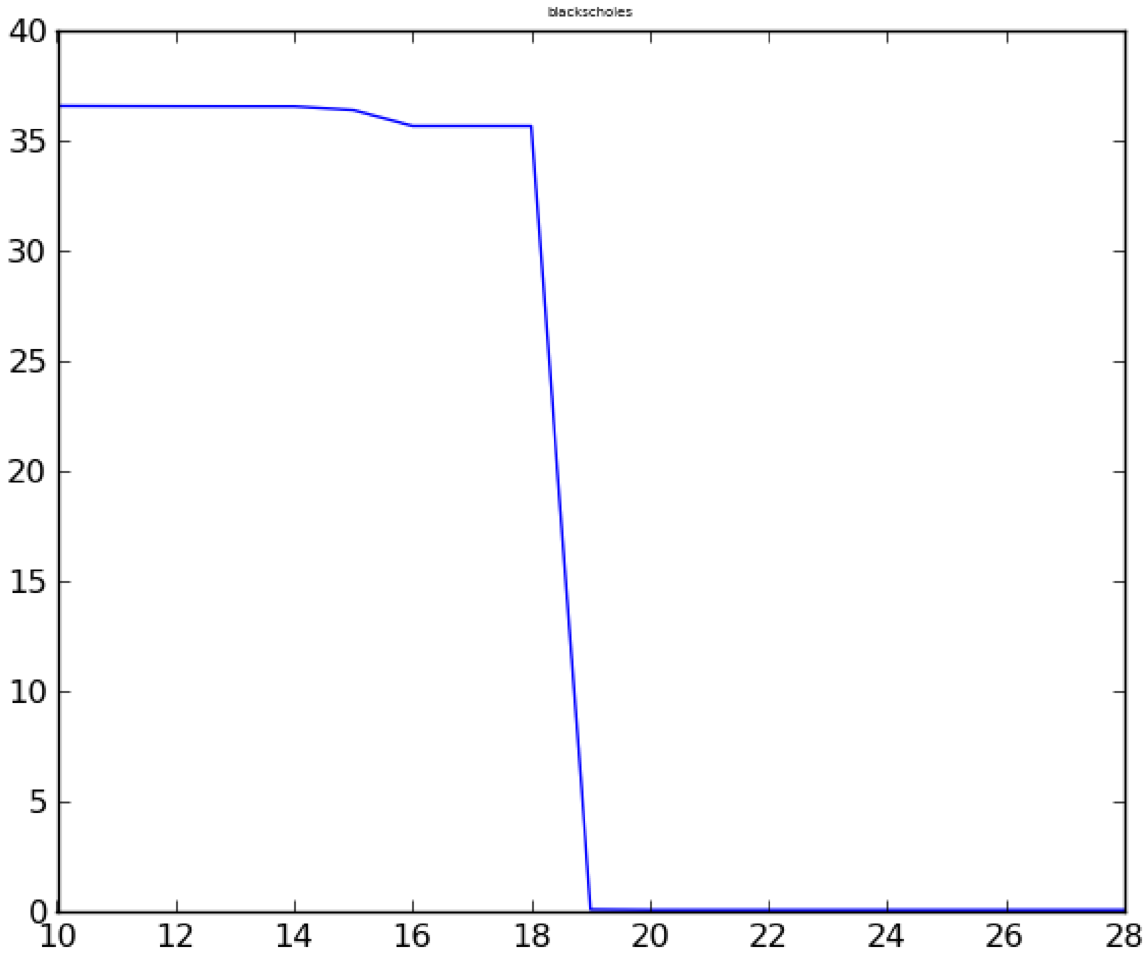
- Bodytrack
  - 91037 races observed
  - Number of BB's containing races: 16
    - Conflicting access address: 7fff936f3be0(Idx:0) in (Context:Task) -- (0:32) and (11:0)

    - 16107, mainPthreads(std::string, int, int, int, int, int, bool), main.bc:43
    - 9941, threads::thread_entry(void*), Thread.bc:32

Cache Modeling

- Supply a cache simulator with a sequence of read and write addresses
  - Change size, associativity, replacement
- Task graph also has basic blocks and memory allocations

Blackscholes Cache Misses



Miss Rate

LOG2 Cache Size

Blackscholes Detailed

- 256KB shared cache (35% miss rate)
  - Basic block 26 – 98.5% of all misses
    - bs_thread(void*) @ blackscholes.m4.bc:376
    - Each thread has a block start < i < end to process
    - `price = BlkSchlsEqEuroNoDiv(` **`sptprice[i], strike[i], rate[i], volatility[i], otime[i]`**`, otype[i], 0);`

  - Allocation at block 67 of 327936B (99.8% of misses)

```
buffer = (fptype *) malloc(5 * numOptions * sizeof(fptype) + PAD);
sptprice = (fptype *) (((unsigned long long)buffer + PAD) &
  ~(LINESIZE - 1));
strike = sptprice + numOptions;
rate = strike + numOptions;
volatility = rate + numOptions;
otime = volatility + numOptions;
```

Lock Contention

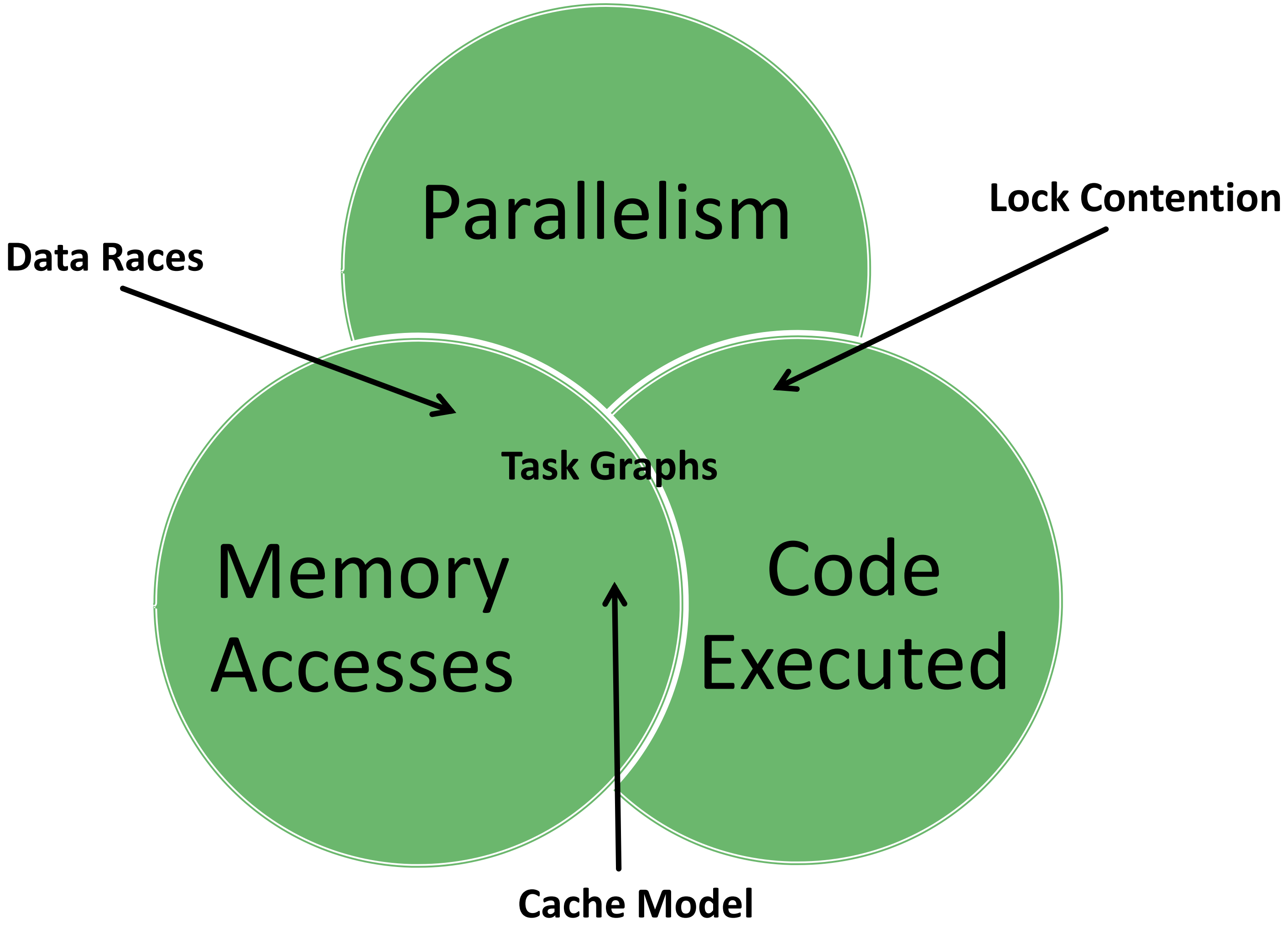
- Track the synchronization in a program
  - When do timestamps overlap for [Release] -> [Acquire]
  - What program points generate the most contention?

Fluidanimate Lock Contention (16 threads)

- Almost 6 million lock acquires
  - Less than 600 are contended
- Contention is doubled on the second of a pair of locks

Contented Acquires	Function Name	File and Line Number
97	ComputeDensitiesMT(int)	pthreads.bc:732
224	ComputeDensitiesMT(int)	pthreads.bc:741
88	ComputeForcesMT(int)	pthreads.bc:834
173	ComputeForcesMT(int)	pthreads.bc:843

Analysis Features



Using Task Graphs

- C++11-based API for analysis
- Three major classes
  - Task Graph – Contains everything
  - Task Graph Info – Debugging-like information
  - Task – Actual contents

Task Graph Class

- Instantiates from a task graph file
  - Reads in the Task Graph Info
  - Parses the "table of contents"
    - Provides the location of every task in the file
    - Provides a breadth-first traversal of the graph
  - Sequential and random access to tasks

Task Graph Info Class

- Static Information about the Task Graph's program
  - Map of basic block ID to information about that block
    - Filename, line number
    - Parent function
    - Count of IR operations, memory operations, etc
  - (Future work) Type information, Function types, etc

Task Class

- All of the data associated with this node in the graph
  - Identifiers
  - Task predecessors and successors
  - Type (i.e., partition)
  - Timestamps
  - Basic block and memory actions

Two types

- Identifiers
  - TaskId = ContextId | SeqId
  - Task relations are expressed using IDs, not pointers
- Actions
  - Basic block – ID
  - Memory operation – Reads and Writes
  - Memory action – Ops + malloc, free, bulk accesses (memcpy)

Cache Simulator Revisited

- Cache simulator takes a trace of memory accesses
  - Iterate through the tasks to generate the sequence of accesses

```
auto memOps = currentTask->getMemOps();

for (auto itMemOp = memOps.begin(),
          etMemOp = memOps.end();
         itMemOp != etMemOp; ++itMemOp)
{
  auto MemoryAction ma = *itMemOp;

  char numOfBytes = (0x1 << ma.pow_size);
  uint64_t address = ma.addr;

  // invoke cache simulator
```

Backend Composition

- Simple backends can extend the Backend class
  - Iterates through the tasks, passing each to the backend
    - void updateBackend(contech::Task*);
  - When all tasks have been parsed, output the analysis to a file
    - void completeBackend(FILE*, contech::TaskGraphInfo*);

```
BackendMemUse* bmu = new BackendMemUse();
SimpleBackendWrapper* sbw = new SimpleBackendWrapper(argv[1], bmu);

sbw->runBackend();
sbw->completeRun(stdout);

delete sbw;
delete bmu;
```

Locks, Mallocs, and other Non-Access Addresses

- How to store an address for non-loads and stores
  - Locks are identified by address
  - Malloc returns an address
- Sync tasks contain a single memory operation
- Mallocs are followed by a memory operations
  - Action type malloc contains the return address
  - Action type size contains the size in the address field

Hands-on Work