Lecture 23b:

Implementing Parallel Runtimes, Part 2

Parallel Computer Architecture and Programming
CMU 15-418/15-618, Spring 2018

Objectives

- What are the costs of using parallelism APIs?
- How do the runtimes operate?

Basis of Lecture

- This lecture is based on runtime and source code analysis of Intel's open source parallel runtimes
  - OpenMP – https://www.openmprtl.org/
  - Cilk – https://bitbucket.org/intelcilkruntime/intel-cilk-runtime

- And using the LLVM compiler
  - OpenMP – part of LLVM as of 3.8
  - Cilk - http://cilkplus.github.io/

OpenMP and Cilk

- What do these have in common?
  - pthreads

- What benefit does abstraction versus implementation provide?

Simple OpenMP Loop Compiled

- **What is this code doing?**
- **What do the OpenMP semantics specify?**
- **How might you accomplish this?**

```
extern float foo( void );
int main (int argc, char** argv) {
   int i;
   float r = 0.0;
   #pragma omp parallel for schedule(dynamic) reduction(+:r)
   for ( i = 0; i < 10; i ++ ) {
      r += foo();
   }
   return 0;
}
```

Example from OpenMP runtime documentation

Simple OpenMP Loop Compiled

```
extern float foo( void );
int main (int argc, char** argv) {
        static int zero = 0;
        auto int gtid;
        auto float r = 0.0;
        __kmpc_begin( & loc3, 0 );
        gtid = __kmpc_global thread num( & loc3 );
        __kmpc_fork call( &loc7, 1, main_7_parallel_3, &r );
        __kmpc_end( & loc0 );
        return 0;
}
```

Call a function in parallel with the argument(s)

Example from OpenMP runtime documentation

Simple OpenMP Loop Compiled

- OpenMP "microtask"
  - Each thread runs the task
- Initializes local iteration bounds and reduction
- Each iteration receives a chunk and operates locally
- After finishing all chunks, combine into global reduction

```
struct main_10_reduction_t_5 { float r_10_rpr; };

void main_7_parallel_3( int *gtid, int *btid, float *r_7_shp ) {
        auto int i_7_pr;
        auto int lower, upper, liter, incr;
        auto struct main_10_reduction_t_5 reduce;
        reduce.r_10_rpr = 0.F;
        liter = 0;
        __kmpc_dispatch_init_4( & loc7,*gtid, 35, 0, 9, 1, 1 );
        while ( __kmpc_dispatch_next_4( & loc7, *gtid, &liter,
          &lower, &upper, &incr) ) {
                for( i_7_pr = lower; upper >= i_7_pr; i_7_pr ++ )
                        reduce.r_10_rpr += foo();
        }
        switch( __kmpc_reduce_nowait( & loc10, *gtid, 1, 4,
          &reduce, main_10_reduce_5, &lck ) ) {
        case 1:
                *r_7_shp += reduce.r_10_rpr;
                __kmpc_end_reduce_nowait( & loc10, *gtid, &lck);
        break;
        case 2:
                __kmpc_atomic_float4_add( & loc10, *gtid,
                  r_7_shp, reduce.r_10_rpr );
        break;
        default:;
        }
}
```

Example from OpenMP runtime documentation

Simple OpenMP Loop Compiled

- **All code combined**

```
extern float foo( void );
int main (int argc, char** argv) {
        static int zero = 0;
        auto int gtid;
        auto float r = 0.0;
        __kmpc_begin( & loc3, 0 );
        gtid = __kmpc_global thread num( & loc3 );
        __kmpc_fork call( &loc7, 1, main_7_parallel_3, &r );
        __kmpc_end( & loc0 );
        return 0;
}

struct main_10_reduction_t_5 { float r_10_rpr; };
static kmp_critical_name lck = { 0 };
static ident_t loc10;

void main_10_reduce_5( struct main_10_reduction_t_5 *reduce_lhs,
struct main_10_reduction_t_5 *reduce_rhs )
{
        reduce_lhs->r_10_rpr += reduce_rhs->r_10_rpr;
}
```

```
void main_7_parallel_3( int *gtid, int *btid, float *r_7_shp ) {
        auto int i_7_pr;
        auto int lower, upper, liter, incr;
        auto struct main_10_reduction_t_5 reduce;
        reduce.r_10_rpr = 0.F;
        liter = 0;
        __kmpc_dispatch_init_4( & loc7,*gtid, 35, 0, 9, 1, 1 );
        while ( __kmpc_dispatch_next_4( & loc7, *gtid, &liter,
           &lower, &upper, &incr) ) {
                for( i_7_pr = lower; upper >= i_7_pr; i_7_pr ++ )
                        reduce.r_10_rpr += foo();
        }
        switch( __kmpc_reduce_nowait( & loc10, *gtid, 1, 4,
           &reduce, main_10_reduce_5, &lck ) ) {
        case 1:
                *r_7_shp += reduce.r_10_rpr;
                __kmpc_end_reduce_nowait( & loc10, *gtid, &lck);
        break;
        case 2:
                __kmpc_atomic_float4_add( & loc10, *gtid, r_7_shp,
                   reduce.r_10_rpr );
        break;
        default:;
        }
}
```

Example from OpenMP runtime documentation

Fork Call

- "Forks" execution and calls a specified routine (microtask)

- Determine how many threads to allocate to the parallel region
- Setup task structures
- Release allocated threads from their idle loop

Iteration Mechanisms

- Static, compile time iterations
  - __kmp_for_static_init
  - Compute one set of iteration bounds
- Everything else
  - __kmp_dispatch_next
  - Compute the next set of iteration bounds

OMP Barriers

- Two phase -> gather and release
  - Gather non-master threads pass, master waits
  - Release is opposite

- Barrier can be:
  - Linear
  - Tree
  - Hypercube
  - Hierarchical

OMP Atomic

- **Can the compiler do this in a read-modify-write (RMW) op?**

- **Otherwise, create a compare-and-swap loop**

```
T* val;
T update;
#pragma omp atomic
   *val += update;
```

If T is int, this is "lock add …".
If T is float, this is "lock cmpxchg …"

Why?

OMP Tasks

- #pragma omp task depend (inout:x) …

- Create microtasks for each task
  - Track dependencies by a list of address / length tuples

Cilk

- **Covered in Lecture 5**
- **We discussed the what and why, now the how**

Simple Cilk Program Compiled

- What is this code doing?
- What do the Cilk semantics specify?
- Which is the child? Which is the continuation?

```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}
```

How to create a continuation?

- Continuation needs all of the state to continue
  - Register values, stack, etc.

- What function allows code to jump to a prior point of execution?

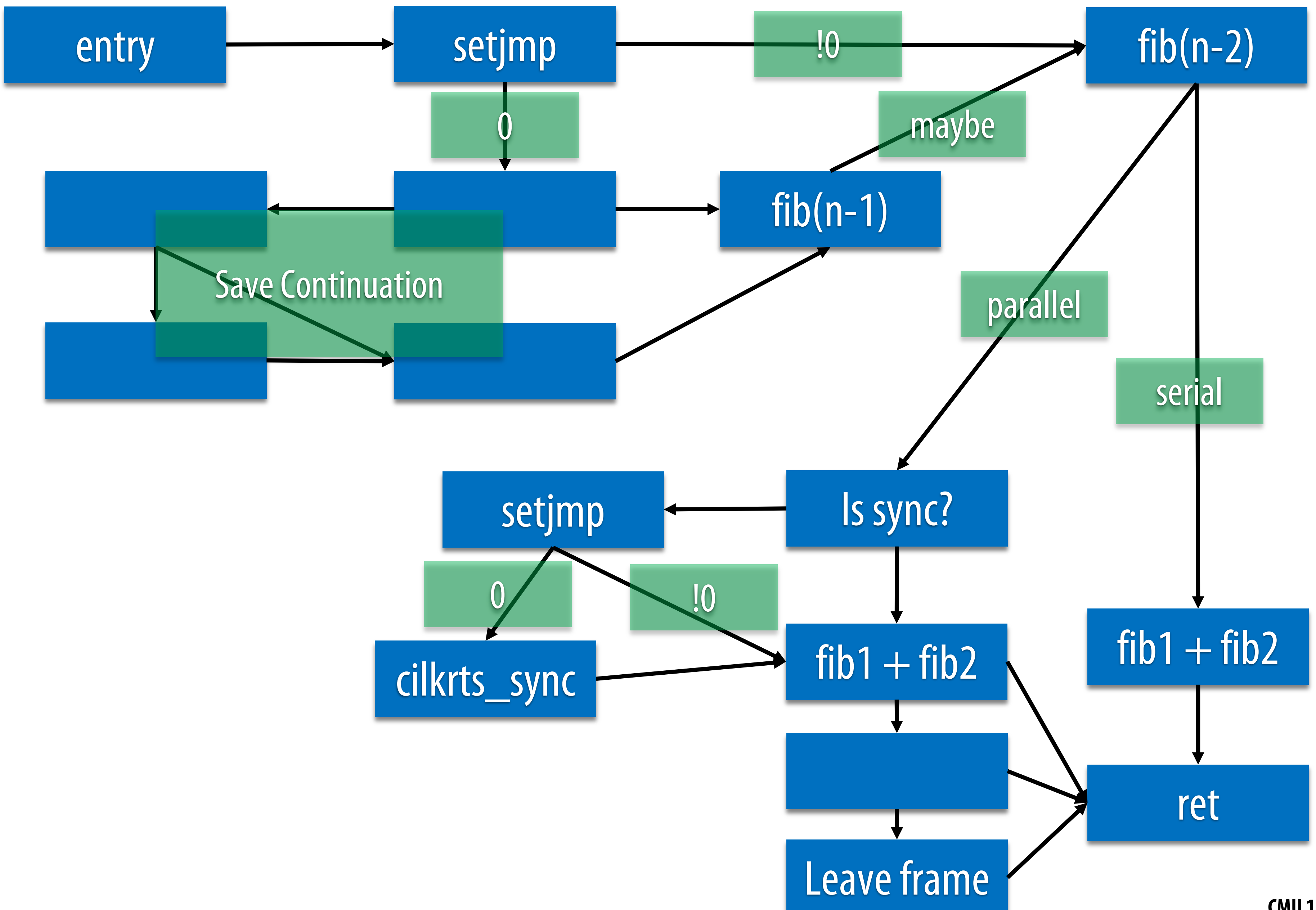
- Setjmp(jmp_buf env)
  - Save stack context
  - Return via longjmp(env, val)
  - Setjmp returns 0 if saving, val if returning via longjmp

Basic Block

- Unit of Code Analysis

- Sequence of instructions
  - Execution can only enter at the first instruction
    - Cannot jump into the middle
  - Execution can only exit at the last instruction
    - Branch or Function Call
    - Or the start of another basic block (fall through)

Simple Cilk Program Revisited



Cilk Workers

- While there may be work
  - Try to get the next item from our queue
  - Else try to get work from a random queue
  - If there is no work found, wait on semaphore

- If work item is found
  - Resume with the continuation's stack

Thread Local Storage

- Linux supports thread local storage
  - New: C11 - _Thread_local keyword
    - one global instance of the variable per thread
  - Compiler places values into .tbss
  - OS provides each thread with this space

- Since Cilk and OpenMP are using pthreads
  - These values are in the layer below them

DEMO