
OpenMP Tools API (OMPT) and HPCToolkit

John Mellor-Crummey

**Department of Computer Science
Rice University**

johnmc@rice.edu

OpenMP Tools Subcommittee

- **Executive lead**
 - Martin Schulz - LLNL
- **Technical leads**
 - Alexandre Eichenberger - IBM
 - John Mellor-Crummey - Rice
- **Active subcommittee members**
 - Nawal Coptly - Oracle
 - Jim Cownie - Intel
 - Robert Dietrich - TU Dresden
 - Christian Iwainsky - TU Darmstadt
 - Xu Liu - Rice
 - Eugene Loh - Oracle
 - Daniel Lorenz - Juelich
 - Harald Servat - Barcelona Supercomputer Center

Motivation

- Large gap between OpenMP source and implementation

Calling Context View

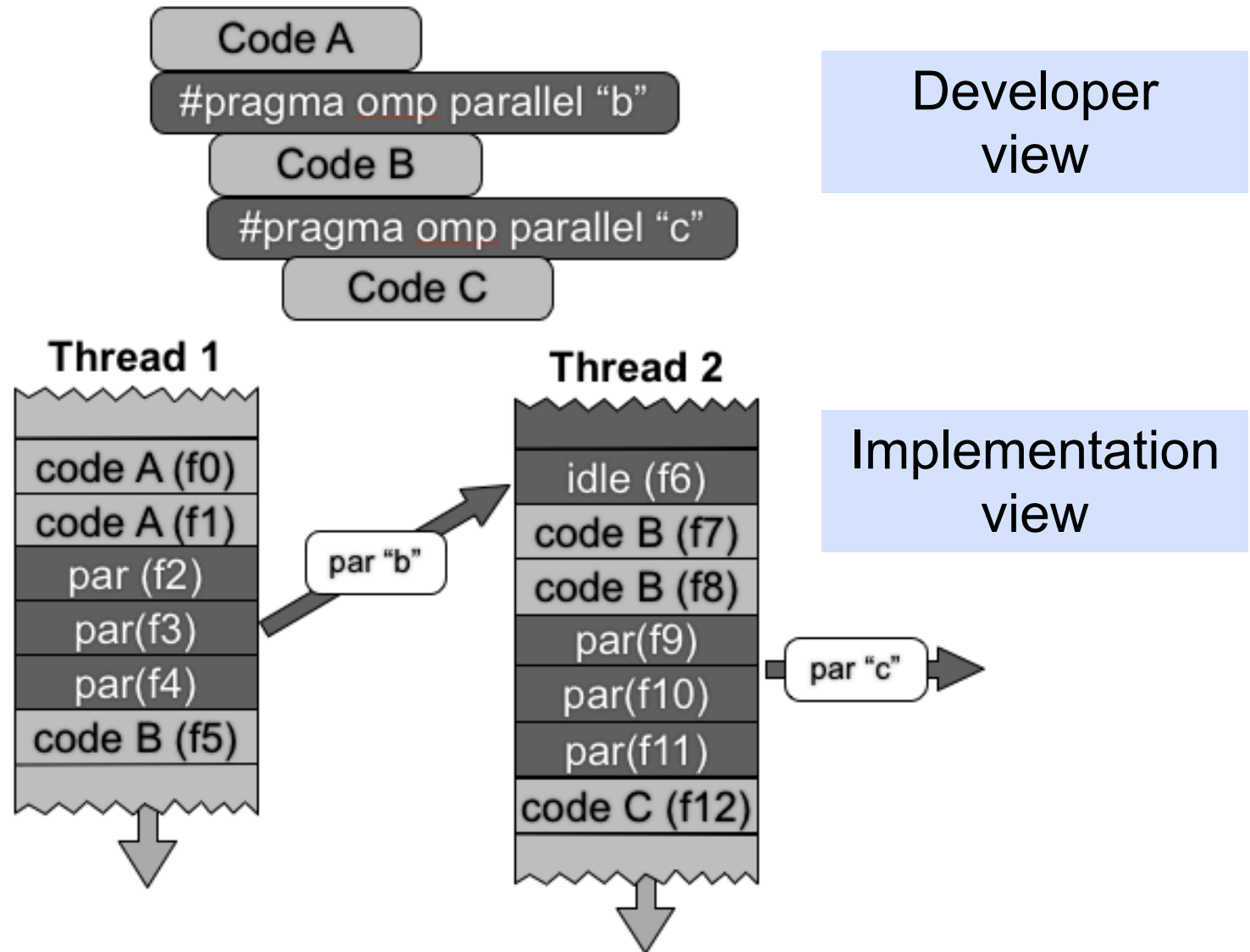
```
1287 /*****  
1288 /* compute the hourglass modes */  
1289  
1290  
1291 #pragma omp parallel for firstprivate(numElem, hourg)  
1292 for(Index_t i2=0; i2<numElem; ++i2){  
1293     Real_t *fx_local, *fy_local, *fz_local ;  
1294     Real_t hgfx[8], hgyf[8], hgfh[8] ;  
1295  
1296     Real_t coefficient;  
1297  
1298     ...  
1299 }
```

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)
Experiment Aggregate Metrics	6.32e+08 100 %	6.32e+08 100 %
monitor_begin_thread	6.06e+08 95.8%	
940: __kmp_launch_worker(void*)	5.80e+08 91.8%	
729: __kmp_launch_thread	5.80e+08 91.8%	1.51e+04 0.0%
6314: __kmp_invoke_task_func	3.38e+08 53.5%	
7586: kmp_invoke_pass_parms	3.38e+08 53.5%	
L_Z28CalcFBHourglassForceForElemsPdS_S_S_S_d_1291__par_loop0_2_276	6.48e+07 10.3%	4.14e+07 6.5%
L_Z22CalcKinematicsForElemsid_1931__par_loop0_2_855	5.36e+07 8.5%	1.72e+07 2.7%
L_Z28CalcHourglassControlForElemsPdd_1516__par_loop0_2_424	4.73e+07 7.5%	1.64e+07 2.6%
L_Z23IntegrateStressForElemsiPdS_S_S_864__par_loop0_2_125	4.34e+07 6.9%	8.66e+06 1.4%
L_Z31CalcMonotonicQGradientsForElemsv_2040__par_loop0_2_965	2.82e+07 4.5%	1.59e+07 2.5%
6333: __kmp_join_barrier(int)	1.63e+07 2.6%	2.50e+04 0.0%
6302: __kmp_clear_x87_fpu_status_word	2.00e+04 0.0%	2.00e+04 0.0%
kmp_runtime.c: 6236		
940: __kmp_launch_monitor(void*)	2.53e+07 4.0%	
monitor_main	2.63e+07 4.2%	
483: main	2.63e+07 4.2%	2.10e+05 0.0%
3187: LagrangeLeapFrog()	2.52e+07 4.0%	
3049: Domain::AllocateNodeElemIndexes()	4.66e+05 0.1%	2.15e+05 0.0%
2995: Domain::AllocateElemPersistent(unsigned long)	8.09e+04 0.0%	

Calling context for code in parallel regions and tasks executed by worker threads is not readily available

- Tools must bridge this gap to explain program performance

Calling Context is Distributed across Threads



Obstacles for Runtime-independent Tools

- **No standard API for OpenMP tools**
- **Principal prior efforts**
 - POMP - Mohr, Malony, Shende, Wolf
 - Collector API - Itzkowitz, Mazurov, Coptly, Lin
- **Differences in OpenMP implementations**
 - support for static linking
 - Intel: extra “monitor” thread
 - PGI: cactus stack
 - IBM: neither

Outline

- **OMPT - emerging performance tool API for OpenMP**
 - overview and goals
 - state tracking
 - event notification
 - API
- **Status and next steps**

OMPT Design Objectives

- **Enable tools to gather information and associate costs with application source and runtime system**
 - construct low-overhead tools based on asynchronous sampling
 - identify application stack frames vs. runtime frames
 - associate a thread's activity at any point with a descriptive state
 - parallel work, idle, lock wait, ...
- **Negligible overhead if OMPT interface is not in use**
- **Define support for trace-based performance tools**
- **Don't impose an unreasonable development burden**
 - runtime implementers
 - tool developers

OMPT Performance Tools API

Overview and Goals

- **Focus on minimal set of functionality**
 - provide essential support for sampling-based tools
 - only support tools attached at link-time or program launch
- **Minimize runtime cost**
 - reduce cost in runtime and tool where possible
 - encourage integration into optimized runtimes
 - make support for higher-overhead features optional
 - callbacks for blame shifting
 - callbacks for full-featured tracing tools

Major OMPT Functionality

- **State tracking**
 - have runtime track keep track of thread states (e.g., idle, parallel)
 - allow tools to query this state at any time (async signal safe)
 - provide (limited) persistent storage for tool data in runtime system
- **Call stack interpretation**
 - provide **hooks** that enable tools to **reconstruct application-level call stacks** from implementation-level information
- **Event notification**
 - provide **callbacks** for predefined events
 - few mandatory notifications
 - many optional ones

Runtime State Tracking

- OpenMP runtime keeps track of its own state
 - predefined states on next slide
- Query routine
 - `ompt_state_t ompt_get_state(ompt_wait_id_t *wait_id)`
 - async signal safe
- Wait IDs
 - only returned for states that signify waiting
 - identifies the cause for waiting (lock, critical region, ...)

OMPT Runtime States

- **Work**
 - serial, parallel
 - reduction
- **Idle**
- **Barrier**
 - implicit
 - explicit
 - either
- **Task wait**
 - task wait
 - task group wait
- **Mutual Exclusion**
 - wait lock
 - wait nest lock
 - wait critical
 - wait atomic
 - wait ordered
- **Overhead**
- **Miscellaneous**
 - first
 - undefined

OMPT Event Notifications

- **Mandatory events**
- **Blame-shifting events (optional)**
- **Trace events (optional)**

Mandatory Events

Essential support for any performance tool

- **Threads**
- **Parallel regions** create/exit event pairs
- **Tasks**
- **Runtime shutdown** singleton events
- **User-level control API**
—e.g., support tool start/stop

Blame-shifting Events (Optional)

Support designed for sampling-based performance tools

- **Idle**

- **Wait**

- barrier

- taskwait

- taskgroup wait

begin/end event pairs

- **Release**

- lock

- nest lock

- critical

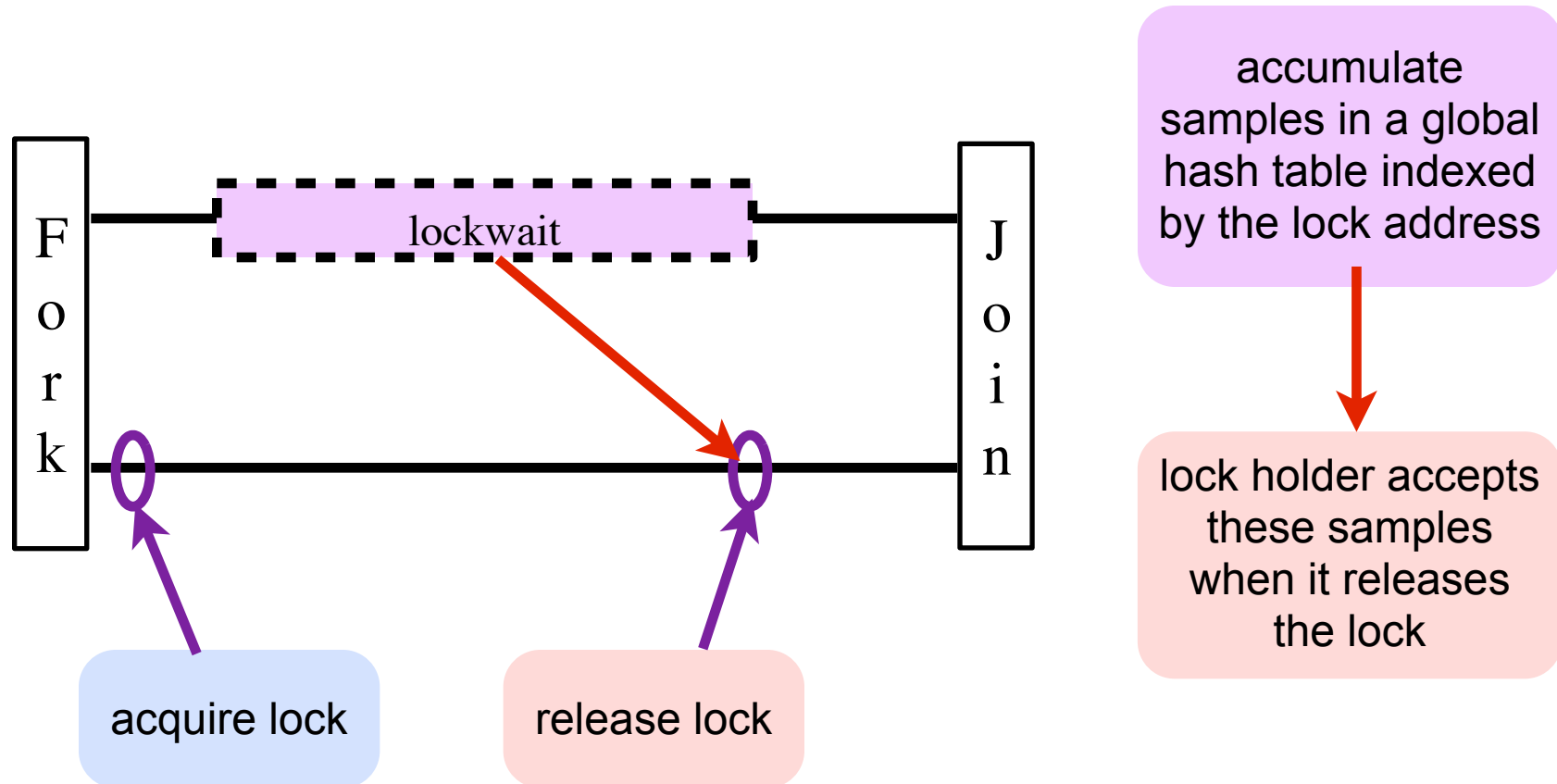
- atomic

- ordered section

singleton events

Directed Blame Shifting

- **Example:**
 - threads waiting at a lock are the symptom
 - the cause is the lock holder
- **Approach: blame lock waiting on lock holder**



Example: Directed Blame Shifting for

Blame a lock holder for delaying waiting threads

- Charge all samples that threads receive while awaiting a lock to the lock itself
- When releasing a lock, accept blame at the lock

all of the waiting occurs here (symptom)

almost all blame for the waiting is attributed here (cause)

```
locktest-2.c
1 #include <omp.h>
2 #include "fib.h"
3 void g() {
4     int i;
5     omp_lock_t l;
6     omp_init_lock(&l);
7     #pragma omp parallel
8     {
9         #pragma omp master
10        {
11            omp_set_lock(&l);
12            fib(40);
13            omp_unset_lock(&l);
14        }
15        #pragma omp for
16        for(i = 0; i < 100; i++) {
17            omp_set_lock(&l);
18            fib(10);
19            omp_unset_lock(&l);
20        }
21    }
22 }
23 void f() { g(); }
24 int main() { f(); return 0; }
```

Scope	MUTEX_WAIT:Sum (l)	MUTEX_BLAZE:Sum (l)
Experiment Aggregate Metrics	8.11e+07 100 %	7.93e+07 100 %
monitor_main	8.11e+07 100 %	7.93e+07 100 %
483: main	8.11e+07 100 %	7.93e+07 100 %
29: f	8.11e+07 100 %	7.93e+07 100 %
25: g	8.11e+07 100 %	7.93e+07 100 %
7: L_g_7__par_region0_2_90	8.11e+07 100 %	7.93e+07 100 %
17: kmpc set lock	8.11e+07 100 %	7.87e+07 99.2%
12: fib		
20: __kmpc_barrier		
locktest-2.c: 13		7.87e+07 99.2%

Trace Events (Optional)

<code>ompt_event_implicit_task_create</code>	<code>ompt_event_taskgroup_end</code>
<code>ompt_event_implicit_task_exit</code>	<code>ompt_event_release_nest_lock_prev</code>
<code>ompt_event_task_switch</code>	<code>ompt_event_wait_lock</code>
<code>ompt_event_loop_begin</code>	<code>ompt_event_wait_nest_lock</code>
<code>ompt_event_loop_end</code>	<code>ompt_event_wait_critical</code>
<code>ompt_event_section_begin</code>	<code>ompt_event_wait_atomic</code>
<code>ompt_event_section_end</code>	<code>ompt_event_wait_ordered</code>
<code>ompt_event_single_in_block_begin</code>	<code>ompt_event_acquired_lock</code>
<code>ompt_event_single_in_block_end</code>	<code>ompt_event_acquired_nest_lock_first</code>
<code>ompt_event_single_others_begin</code>	<code>ompt_event_acquired_nest_lock_next</code>
<code>ompt_event_single_others_end</code>	<code>ompt_event_acquired_critical</code>
<code>ompt_event_master_begin</code>	<code>ompt_event_acquired_atomic</code>
<code>ompt_event_master_end</code>	<code>ompt_event_acquired_ordered</code>
<code>ompt_event_barrier_begin</code>	<code>ompt_event_init_lock</code>
<code>ompt_event_barrier_end</code>	<code>ompt_event_init_nest_lock</code>
<code>ompt_event_taskwait_begin</code>	<code>ompt_event_destroy_lock</code>
<code>ompt_event_taskwait_end</code>	<code>ompt_event_destroy_nest_lock</code>
<code>ompt_event_taskgroup_begin</code>	<code>ompt_event_flush</code>

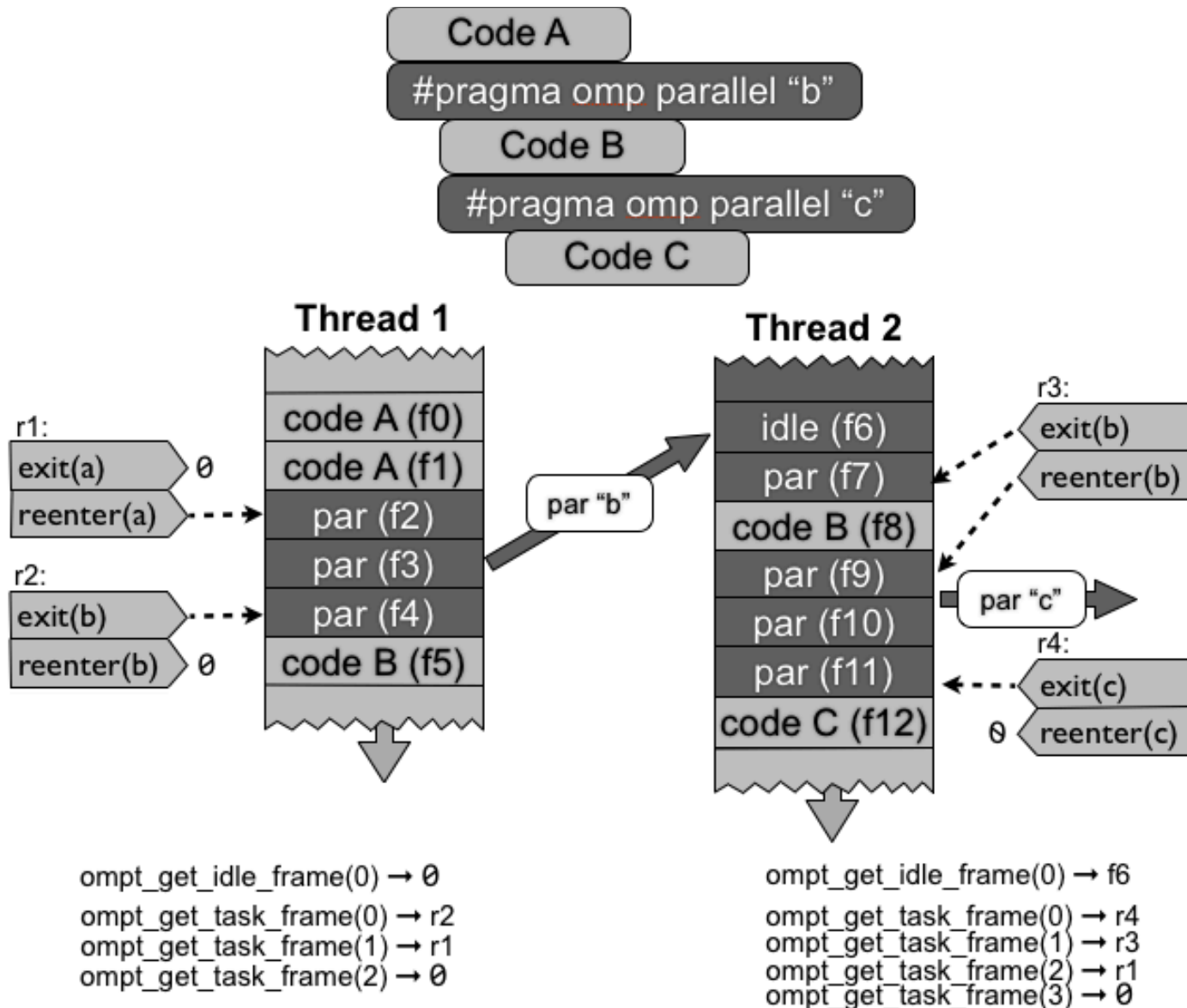
Parallel Region and Task IDs

- Unique id for each parallel region instance and task instance
- Ability to query parallel region and task IDs
 - `ompt_parallel_id_t ompt_get_parallel_id(int ancestor_level)`
 - `ompt_task_id_t ompt_get_task_id(int ancestor_level)`

 - current task: `ancestor_level = 0`
 - query IDs of ancestor task/region using higher ancestor levels

 - async signal safe

Call Stack Interpretation



Miscellaneous API Features

- **Tool-facing API functions**

- initialization**

- `int ompt_initialize(ompt_function_lookup_t ompt_fn_lookup, const char * version, int ompt_version)`
 - `int ompt_set_callback(ompt_event_t e, ompt_callback_t cb)`

- state enumeration**

- `int ompt_enumerate_state(int current_state, int *next_state, const char **next_state_name)`

- **Tool control**

- `void ompt_control(uint64_t command, uint64_t modifier)`

Outline

- **OMPT - emerging performance tool API for OpenMP**
 - overview and goals
 - state tracking
 - event notification
 - API
- **Status and next steps**

OMPT Status and Next Steps

- Subcommittee approved the document a few weeks ago
 - a few recent text changes based on late comments
- Available on the WWW for comment: <http://bit.ly/ompt-tr>
- Nov 7 ARB meeting
 - tools group approved as subcommittee of language committee
- Submit it to OpenMP language committee for official comment
 - turn it into an official OpenMP TR
- Runtime implementations
 - IBM implementation of OMPT interface for BG/Q and Power
 - Rice and Oregon prototype implementation OMPT in Intel runtime
- Tools
 - Rice University's HPCToolkit (development branch)
 - University of Oregon's Tau