Visualizing OpenMP Execution using OMPT

Yonghong Yan, Philip Conrad, Yudong Sun

Department of Computer Science and Engineering, University of South Carolina

OpenMP Tool Interface

- OpenMP Instrumentation and Tracing API
  - For building OpenMP performance profiling and analysis tools
    - First party tools
- Built into the OpenMP runtime
  - No user instrumentation or recompilation/relinking needed
- Detailed inspection of program events/states via callbacks.
  - Thread states
  - Mutex behavior
  - Parallel behavior

Some Events and Callback

```
ompt_callback_thread_begin
ompt_callback_thread_end
ompt_callback_parallel_begin
ompt_callback_parallel_end
ompt_callback_task_create
ompt_callback_task_schedule
ompt_callback_implicit_task
ompt_callback_target
ompt_callback_target_data_op
ompt_callback_target_submit
ompt_callback_control_tool
ompt_callback_device_initialize
ompt_callback_device_finalize
ompt_callback_device_load
ompt_callback_device_unload
ompt_callback_sync_region_wait
```

```
ompt_callback_mutex_released
ompt_callback_task_dependences
ompt_callback_task_dependence
ompt_callback_work
ompt_callback_master
ompt_callback_target_map
ompt_callback_sync_region
ompt_callback_reduction
ompt_callback_lock_init
ompt_callback_lock_destroy
ompt_callback_mutex_acquire
ompt_callback_mutex_acquired
ompt_callback_nest_lock
ompt_callback_flush
ompt_callback_cancel
ompt_callback_dispatch
...
```

An Example

```
33  void init(REAL *A, int N) {
34      int i;
35  #pragma omp parallel for shared(A, N) private(i)
36      for (i = 0; i < N; i++) {
37          A[i] = (double) drand48();
38      }
39  }
```

```
103 void axpy_omp_parallel_for(int N, REAL *Y, REAL *X, REAL a) {
104     int i;
105     #pragma omp parallel shared(N, X, Y, a) private(i)
106     {
107     #pragma omp for schedule(dynamic,64)
108     for (i = 0; i < N; ++i) {
109         Y[i] += a * X[i];
110         }
111
112     #pragma omp barrier
113
114     #pragma omp master
115         printf("num_threads: %d\n", omp_get_num_threads());
116    }
117 }
```

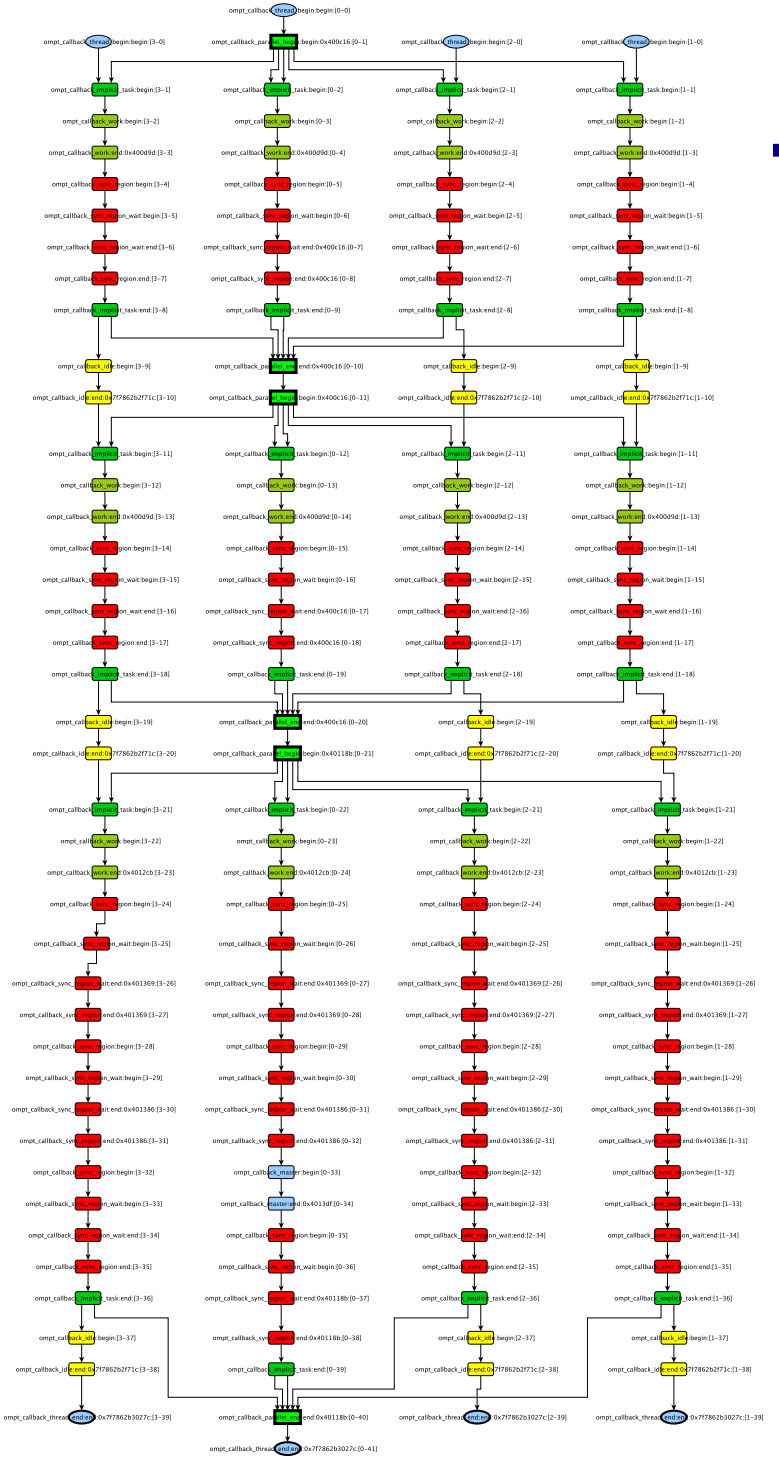
```
71      init(X, N);
72      init(Y_base, N);
73      axpy_omp_parallel_for(N, Y_base, X, a);
```

OMPT Events Tracing

```
33  void init(REAL *A, int N) {
34      int i;
35  #pragma omp parallel for shared(A, N) private(i)
36      for (i = 0; i < N; i++) {
37          A[i] = (double) drand48();
38      }
39  }
```

```
103 void axpy_omp_parallel_for(int N, REAL *Y, REAL *X, REAL a) {
104     int i;
105     #pragma omp parallel shared(N, X, Y, a) private(i)
106     {
107     #pragma omp for schedule(dynamic,64)
108     for (i = 0; i < N; ++i) {
109         Y[i] += a * X[i];
110         }
111
112     #pragma omp barrier
113
114     #pragma omp master
115         printf("num_threads: %d\n", omp_get_num_threads());
116    }
117 }
```

```
71      init(X, N);
72      init(Y_base, N);
73      axpy_omp_parallel_for(N, Y_base, X, a);
```

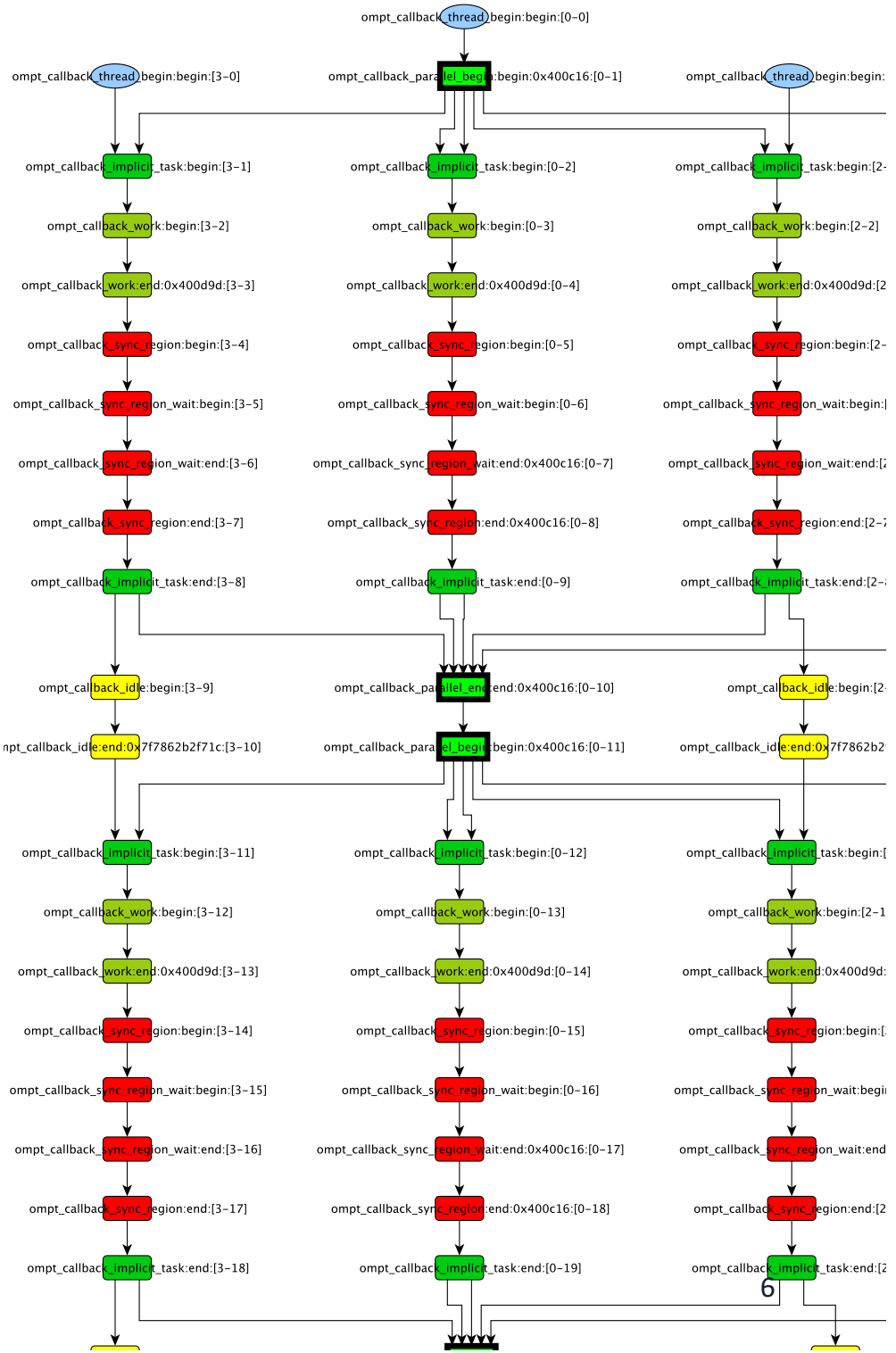


OMPT Events Tracing

```
33  void init(REAL *A, int N) {
34      int i;
35  #pragma omp parallel for shared(A, N) private(i)
36      for (i = 0; i < N; i++) {
37          A[i] = (double) drand48();
38      }
39  }
```

```
103 void axpy_omp_parallel_for(int N, REAL *Y, REAL *X, REAL a) {
104     int i;
105     #pragma omp parallel shared(N, X, Y, a) private(i)
106     {
107     #pragma omp for schedule(dynamic,64)
108     for (i = 0; i < N; ++i) {
109         Y[i] += a * X[i];
110         }
111
112     #pragma omp barrier
113
114     #pragma omp master
115         printf("num_threads: %d\n", omp_get_num_threads());
116    }
117 }
```

```
71      init(X, N);
72      init(Y_base, N);
73      axpy_omp_parallel_for(N, Y_base, X, a);
```

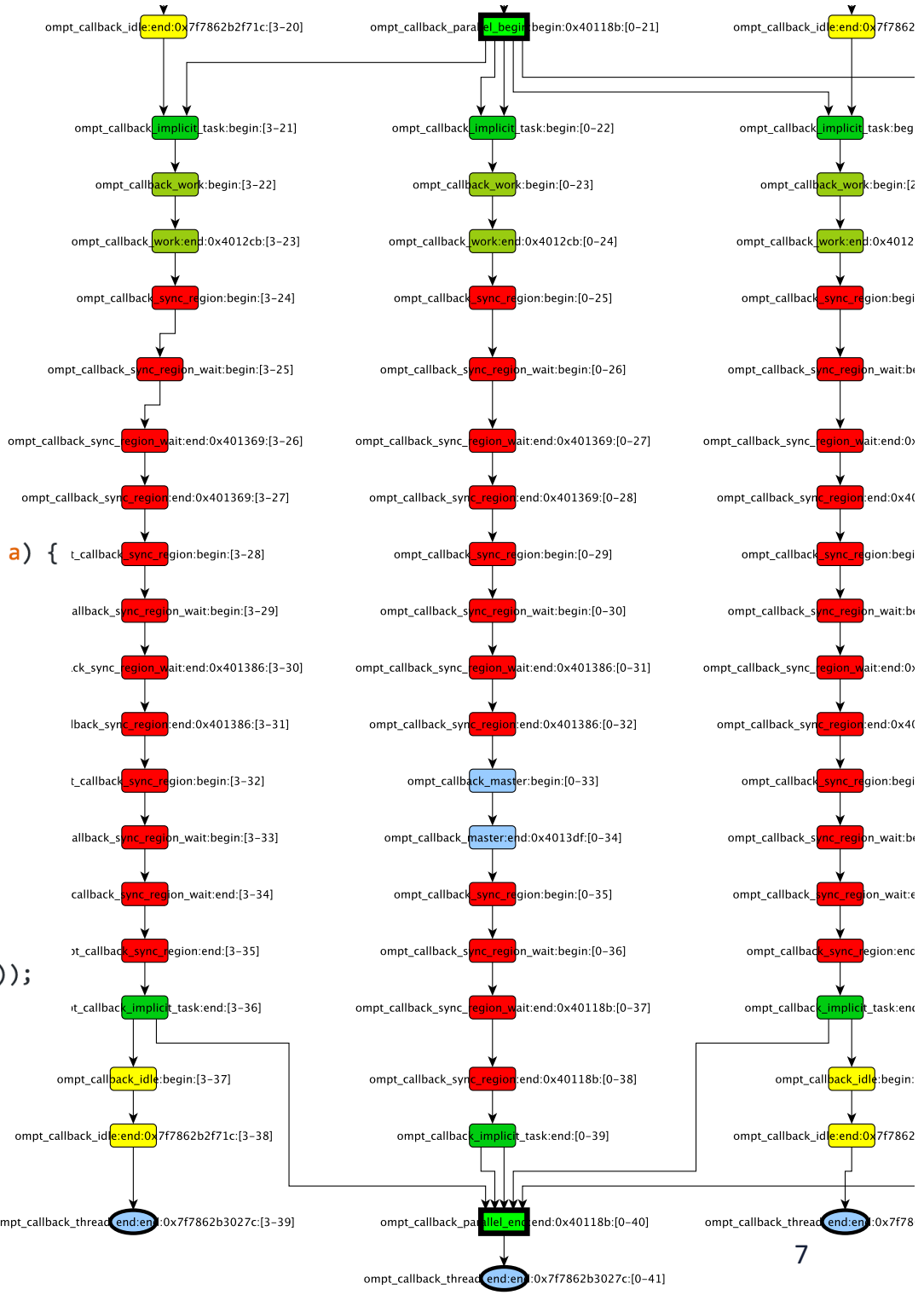


OMPT Events Tracing

```
33  void init(REAL *A, int N) {
34      int i;
35  #pragma omp parallel for shared(A, N) private(i)
36      for (i = 0; i < N; i++) {
37          A[i] = (double) drand48();
38      }
39  }
```

```
103 void axpy_omp_parallel_for(int N, REAL *Y, REAL *X, REAL a) {
104     int i;
105     #pragma omp parallel shared(N, X, Y, a) private(i)
106     {
107     #pragma omp for schedule(dynamic,64)
108     for (i = 0; i < N; ++i) {
109         Y[i] += a * X[i];
110         }
111
112     #pragma omp barrier
113
114     #pragma omp master
115         printf("num_threads: %d\n", omp_get_num_threads());
116    }
117 }
```

```
71      init(X, N);
72      init(Y_base, N);
73      axpy_omp_parallel_for(N, Y_base, X, a);
```



Tracing

- Each thread has its own tracing buffer for storing tracing events
  - Consecutively added when an event happen by the trace callback
- Lexgion: OpenMP Lexical Code Region
  - Lexical scope provided by codeptr_ra
  - Each thread has its own lexgion stack
    - Each lexgion has a link-list of events for the region
- Parallel region
  - Master threads has buffer to store the implicit-task-begin events records of worker threads
- Added some performance profiling features
  - Timestamp, PAPI read, and power read
- GraphML output