Supplemental material: Mochi threading

Mochi Bootcamp
February 6, 2020

Argonne NATIONAL LABORATORY

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- *User-level threads (ULTs)* - an execution unit associated with a specific function
    - Scheduled on an ES, must yield to allow other ULTs execute on the ES
- *Pools* - set of schedulable work units for 1 or more ES
- *Scheduler* - Chooses what to execute from one or more associated pools

```
ABT_pool pool;
ABT_pool_create_basic(ABT_POOL_FIFO_WAIT, ABT_POOL_ACCESS_MPMC,
        ABT_TRUE, &pool);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- ➢ *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- ➢ *User-level threads (ULTs)* - an execution unit associated with a specific function
    - ○ Scheduled on an ES, must yield to allow other ULTs exec
- ➢ *Pools* - set of s ES
- r - Ch mor

Automatically free pool

First-in, first-out pool with ability to wait gracefully

Units in pool can be produced on any ES and consumed on any ES

```
BT_pool pool;
BT_pool_create_basic(ABT_POOL_FIFO_WAIT, ABT_POOL_ACCESS_MPMC,
    ABT_TRUE, &pool);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- ➢ *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- ➢ *User-level threads (ULTs)* - an execution unit associated with a specific function
  - ○ Scheduled on an ES, must yield to allow other ULTs execute on the ES
- ➢ *Pools* - set of schedulable work units for 1 or more ES
- ➢ *Scheduler* - Chooses what to execute from one or more associated pools

```
...
ABT_sched sched;
ABT_sched_create_basic(ABT_SCHED_BASIC_WAIT, 1, &pool,
        ABT_SCHED_CONFIG_NULL, &sched);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- ➢ *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- ➢ *User-level threads (ULTs)* - an execution unit associated with a specific function
  - ○ Scheduled on an ES, must yield to allow other ULTs execute on the ES
- ➢ *Pools* - set of schedulable work units for 1 or more ES
- ➢ *Scheduler* - Chooses v ssociate

Scheduler with ability to wait gracefully

1 or more pools to schedule work from

```
...
ABT_sched sched;
ABT_sched_create_basic(ABT_SCHED_BASIC_WAIT, 1, &pool,
        ABT_SCHED_CONFIG_NULL, &sched);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- *User-level threads (ULTs)* - an execution unit associated with a specific function
  - Scheduled on an ES, must yield to allow other ULTs execute on the ES
- *Pools* - set of schedulable work units for 1 or more ES
- *Scheduler* - Chooses what to execute from one or more associated pools

```
...
ABT_xstream xstream;
ABT_xstream_create(sched, &xstream);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- *User-level threads (ULTs)* - an execution unit associated with a specific function
    - Scheduled on an ES, must yield to allow other ULTs execute on the ES
- *Pools* - set of schedulable work units for 1 or more ES
- *Scheduler* - Chooses what to execute from one or more associated pools

```
...
ABT_thread thread;
ABT_thread_create(pool, func_ptr, func_arg, ABT_THREAD_ATTR_NULL, &thread);
```

Threading

Getting comfortable with Argobots threading is critical to achieving desired performance under high-concurrency

Keep in mind some key Argobots terminology:

- *Execution stream (ES)* - sequential execution streams, essentially an OS thread
- *User-level threads (ULTs)* - an execution unit associated with a specific function
    - Scheduled on an ES, must yield to allow other ULTs execute on the ES
- *Pools* - set of sch
- *Scheduler* - Choo ass

Associate the created ULT with this pool

Function pointer for thread handler, and user data pointer

```
...
ABT_thread thread;
ABT_thread_create(pool, func_ptr, func_arg, ABT_THREAD_ATTR_NULL, &thread);
```

Providing xstreams/pools for Margo

At Margo init time, we have the opportunity to specify a couple of threading options:

Using regular margo_init:

- *use_progress_thread* - boolean value, 1 to use dedicated progress, 0 to use calling thread
- *rpc_thread_count - number of ESs to allocate for RPC handlers, 0 to use calling thread, -1 to use progress thread*

```
margo_instance_id margo_init_opt(const char *addr_str, int mode,
    int use_progress_thread, int rpc_thread_count);
```

Providing xstreams/pools for Margo

At Margo init time, we have the opportunity to specify a couple of threading options:

Using regular margo_init_pool:

- *progress_pool* - ABT_pool to use for the progress thread
- *handler_pool* - ABT_pool to use for running RPC handlers

```
margo_instance_id margo_init_pool(ABT_pool progress_pool, ABT_pool handler_pool,
   hg_context_t *hg_context);
```

Note you need to also pass in an HG context, rather than an address string. This call is meant to provide caller most control over Margo init

Providing xstreams/pools for Margo

At Margo RPC registration time, we can override the default handler pool we have specified at init time:

- ➢ Last argument is an ABT_pool to use for executing handlers for the RPC type being registered

```
...
MARGO_REGISTER_PROVIDER(mid, "operation_name", void, void,
        operation_ult, provider_id, pool);
```