UCC Schedule and Tasks interface

Sergey Lebedev,
UCF Collective WG
July 15th, 2020

Goals

- Express collective operations in different team libraries using common abstraction
- United progress engine

Abstraction*

- Schedule
  - Expresses complete collective operation for a single rank
  - Includes tasks that need to be executed for a collective operation
- Task
  - A set of operation abstracting a single step in the algorithm
    - Send/Recv operations
    - Collective operations

* https://github.com/manjugv/ucc_wg_public/blob/master/slides/gorentla_ucc_schedule_tasks_datatypes_July8th.pdf

Event Manager

- Following “PubSub” pattern Event Manager is object that can generate event from predefined set.
- Each event manager object can subscribe to and handle events from other event managers.

```
typedef enum {
    UCC_EVENT_PROGRESS = 0,
    UCC_EVENT_COMPLETED,
    UCC_EVENT_LAST
} ucc_event_t;
```

```
typedef struct ucc_event_manager {
    ucc_coll_task_t  *listeners       [UCC_EVENT_LAST][MAX_LISTENERS];
    int               listeners_size [UCC_EVENT_LAST];
} ucc_event_manager_t;
```

UCC collective task

- UCC task derived from event manager that allows it to subscribe/handle events from other tasks
- Handlers table defines how particular task response to a particular event

```
typedef enum {
    UCC_TASK_STATE_NOT_READY,
    UCC_TASK_STATE_INPROGRESS,
    UCC_TASK_STATE_COMPLETED
} ucc_task_state_t;
```

```
typedef struct ucc_coll_task {
    ucc_event_manager_t        em;
    ucc_task_state_t           state;
    ucc_task_event_handler_p handlers[UCC_EVENT_LAST];
} ucc_coll_task_t;
```

UCC schedule

- UCC schedule is a special task
- schedule progresses all tasks that subscribed to the schedule
- schedule tracks completions from all subscribed tasks

```
typedef struct ucc_schedule {
    ucc_coll_task_t   super;
    ucc_coll_args_t   args;
    int               n_completed_tasks;
} ucc_schedule_t;
```

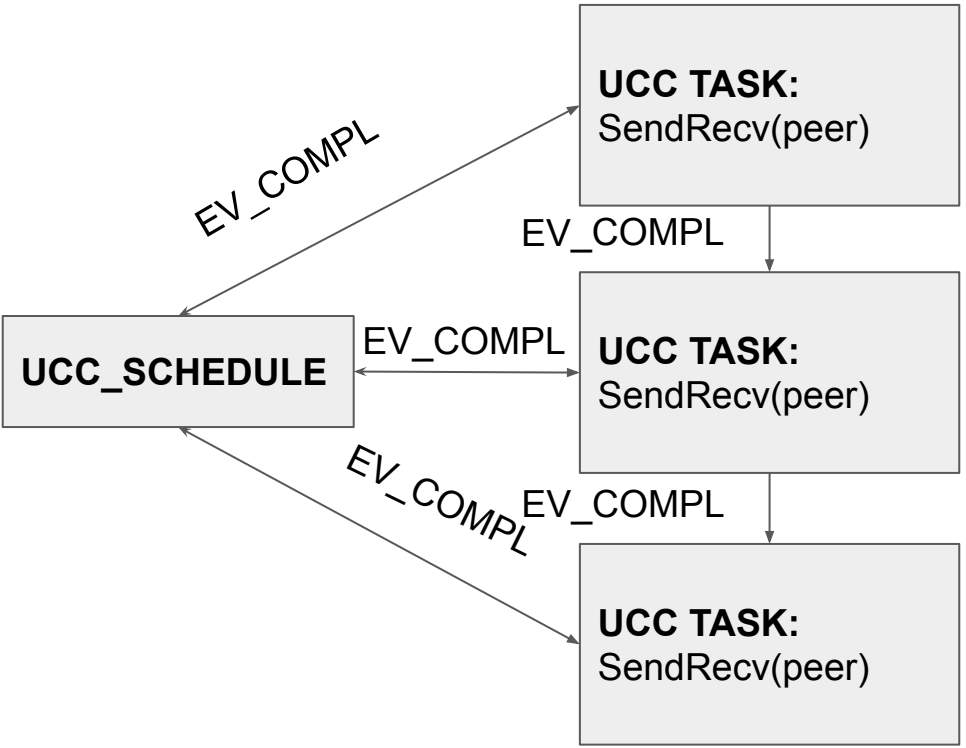
Interface for team library developer

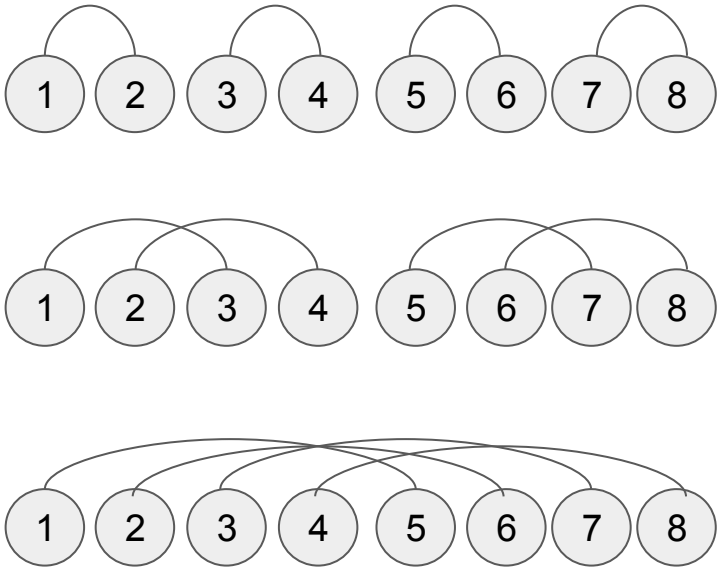
```
/* Event manager */
void ucc_event_manager_init(ucc_event_manager_t *em);
void ucc_event_manager_subscribe(ucc_event_manager_t *em, ucc_event_t event, ucc_coll_task_t *task);
void ucc_event_manager_notify(ucc_event_manager_t *em, ucc_event_t event);

/* Task */
void ucc_coll_task_init(ucc_coll_task_t *task);

/* Schedule */
void ucc_schedule_init(ucc_schedule_t *schedule);
void ucc_schedule_add_task(ucc_schedule_t *schedule, ucc_coll_task_t *task);
void ucc_schedule_start(ucc_schedule_t *schedule);
ucc_status_t ucc_schedule_progress(ucc_schedule_t *schedule);
```

Example: recursive doubling allreduce





Example: recursive doubling allreduce* (schedule)

```
ucc_schedule_init(&schedule.super);
schedule.src_buf = src_buf; schedule.dst_buf = dst_buf; /*set collective arguments for allreduce */
schedule.scratch = tmp_buf; schedule.count   = count;
for (i = 0; i < n_tasks; i++) {
   ucc_coll_task_init(&(tasks[i].super));
   tasks[i].stage = RD_STAGE_START;  tasks[i].schedule = &schedule.super; tasks[i].peer = rank ^ (1 << i); /* set task states */
   tasks[i].super.handlers[UCC_EVENT_PROGRESS]   = task_rd_progress_handler; /* set handlers for each event */
   tasks[i].super.handlers[UCC_EVENT_COMPLETED] = task_rd_completed_handler;
   if (i > 0) ucc_event_manager_subscribe(&tasks[i-1].super.em, UCC_EVENT_COMPLETED, &tasks[i].super); /* task dependencies */
   ucc_schedule_add_task(&schedule.super, &tasks[i].super);
}
ucc_schedule_start(&schedule.super);
do {
  status = ucc_schedule_progress(&schedule.super);
} while (status == UCC_INPROGRESS);
```

* https://gist.github.com/Sergei-Lebedev/1d736aaa7c2ff7d9d4a15e9e4887c863

Example: recursive doubling allreduce (progress)

```
void task_rd_progress_handler(ucc_coll_task_t *task) {
        ucc_coll_task_rd_t *self         = (ucc_coll_task_rd_t*)task;
        ucc_schedule_rd_t *schedule = (ucc_schedule_rd_t*)self->schedule;
        if (task->state == UCC_TASK_STATE_INPROGRESS) {
                int *dst_buf = schedule->dst_buf, *scratch = schedule->scratch;
                int peer     = self->peer;  count    = schedule->count;
                switch (self->stage) {
                case RD_STAGE_START:
                        MPI_Isend(dst_buf, count, MPI_INT, peer, 123, MPI_COMM_WORLD, &schedule->reqs[0]);
                        MPI_Irecv(scratch, count, MPI_INT, peer, 123, MPI_COMM_WORLD, &schedule->reqs[1]);
                        self->stage = RD_STAGE_PROGRESS;
                case RD_STAGE_PROGRESS:
                MPI_Testall(2, schedule->reqs, &flag, st);
                if (flag != 0) {
                    ucc_event_manager_notify(&task->em, UCC_EVENT_COMPLETED);
            task->state = UCC_TASK_STATE_COMPLETED;
                }
        }
}
```

Example: recursive doubling allreduce (completed)

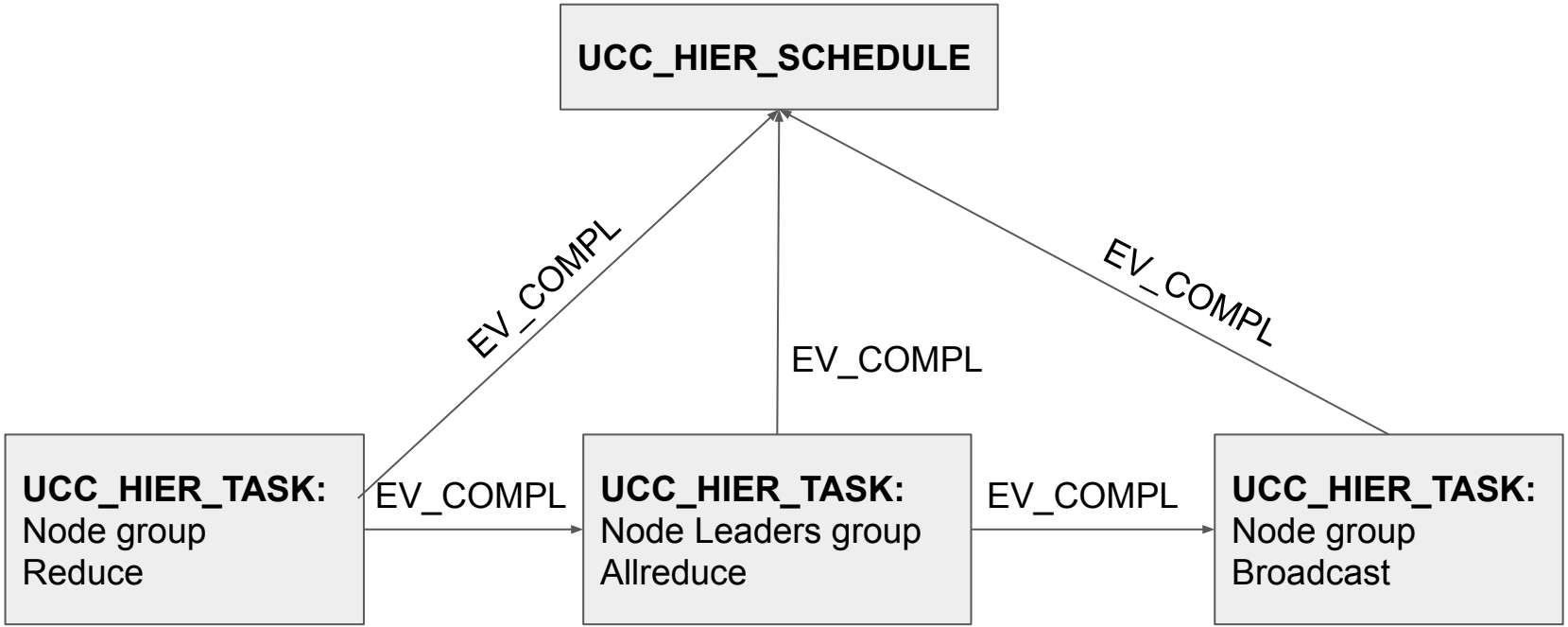
```
void task_rd_completed_handler(ucc_coll_task_t *task)
{
    task->state = UCC_TASK_STATE_INPROGRESS;
}
```

Example: hierarchical tasks

- In hierarchical team task represent collective operation over subgroup of ranks
- Each task starts/progresses/completes its collective using UCC interface

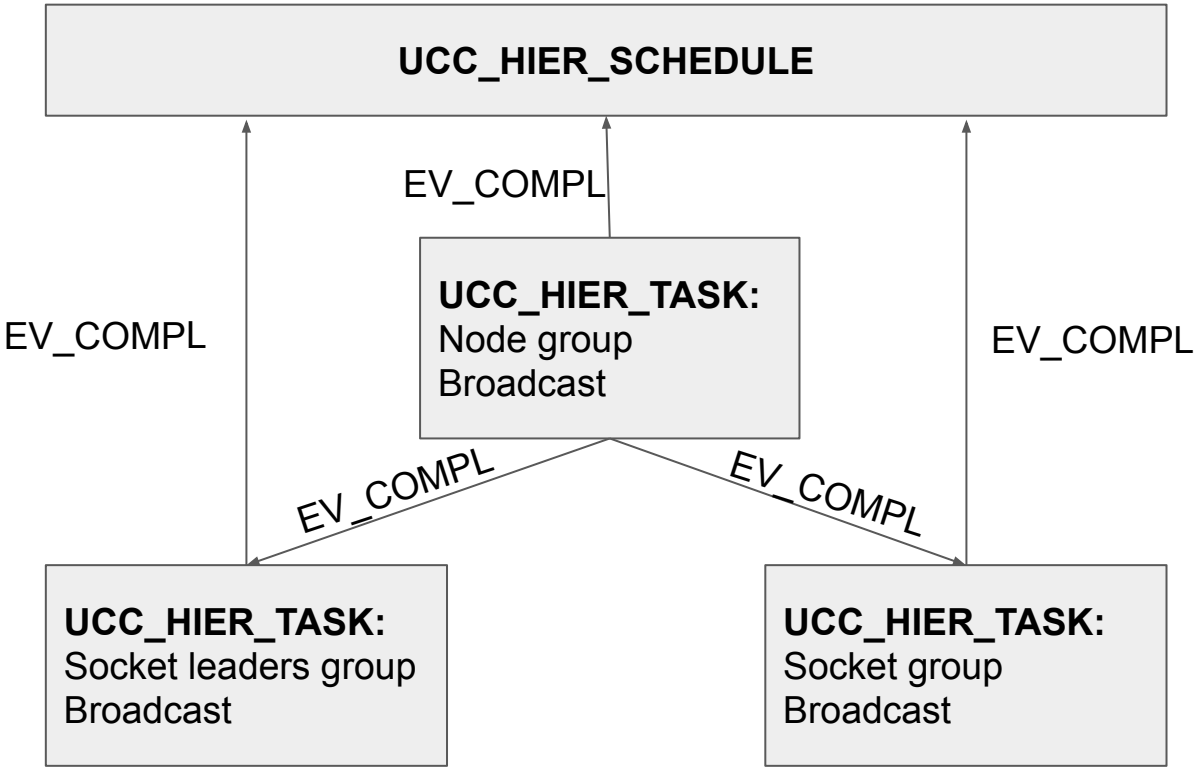
```
typedef struct xccl_hier_task {
    ucc_coll_task_t       super;
    xccl_hier_pair_t      *pair; /* subgroup and team */
    xccl_coll_req_h       req;
} xccl_hier_task_t;
```

Example: hierarchical allreduce



Example: hierarchical broadcast

Node leader non root rank



Open questions

- What events are missing
  - UCC_EVENT_FRAGMENT_COMPLETED
  - UCC_EVENT_CANCELED
- Unsubscribe from event
- Ability to subscribe/unsubscribe on the fly
- UCC context progress