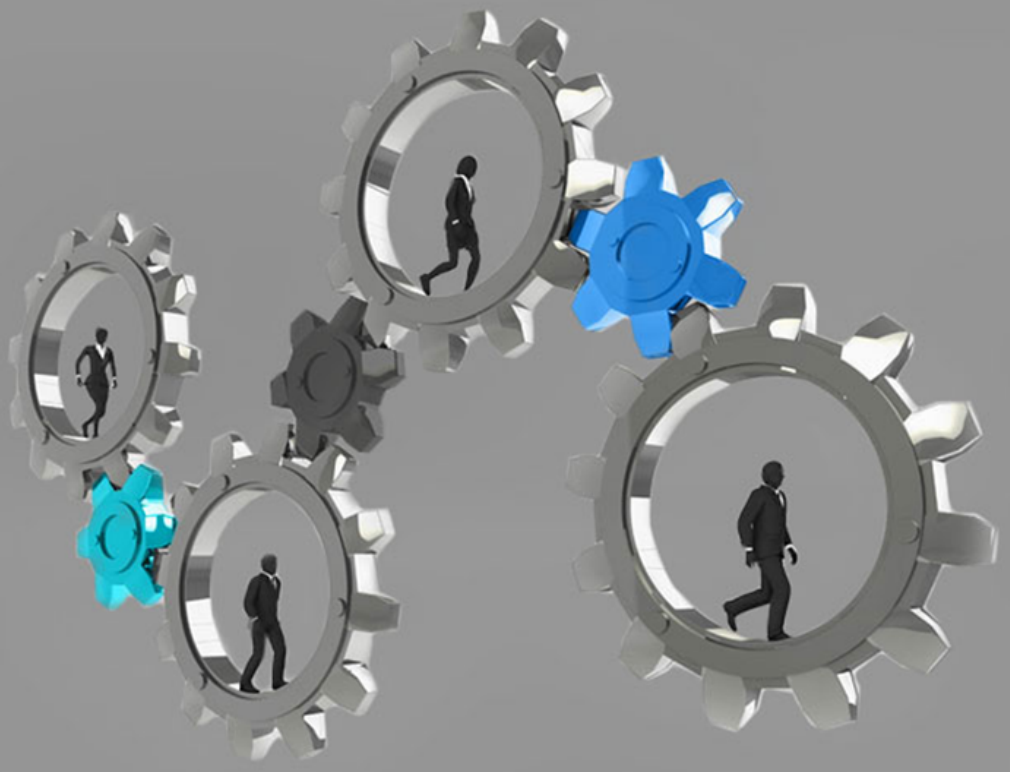
ENABLER OF CO-DESIGN

UCF



UCC Internal Abstractions

Schedules and Tasks

Manjunath Gorentla Venkata, UCF Collectives WG,

June 24th/July 8th, 2020

UCC WG (June 24th)

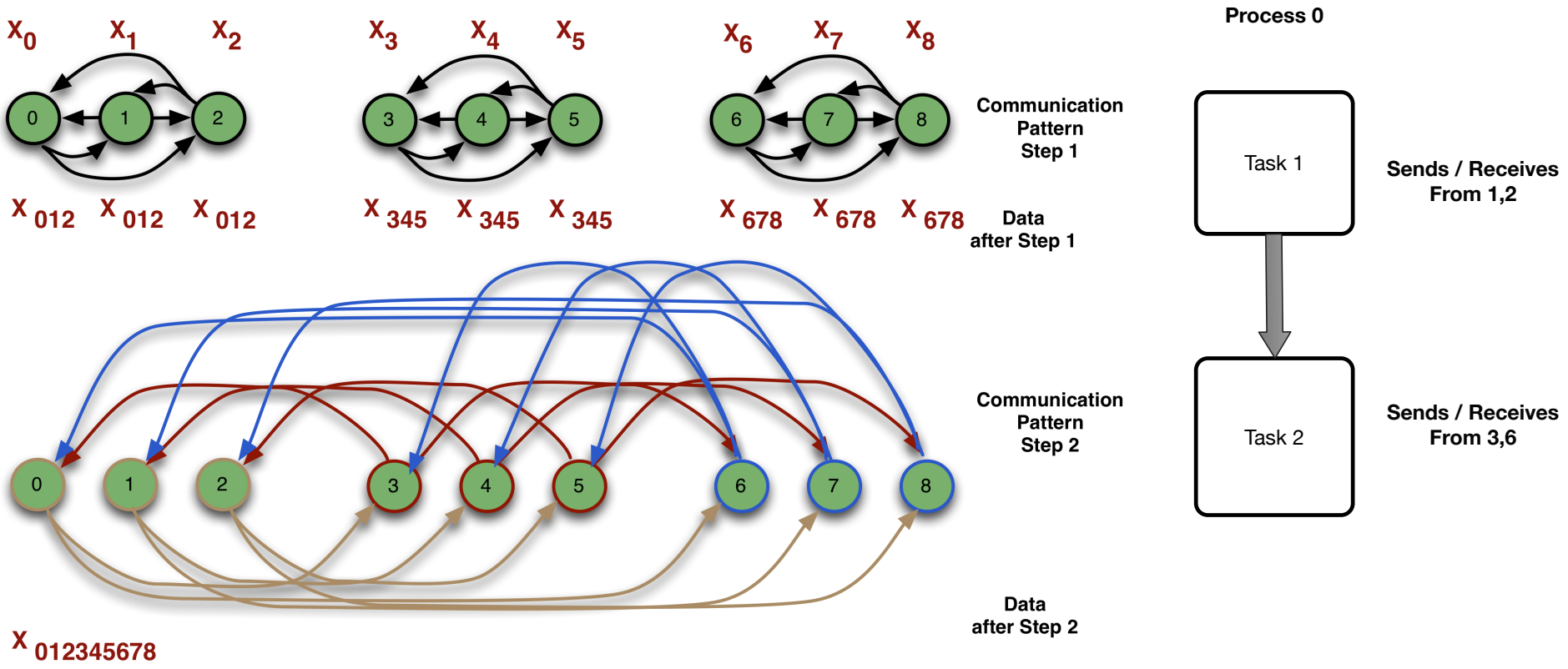
- Goals:
  - Common abstractions to express various collective implementation approaches
    - Examples : Hierarchical, Reactive or hybrid (hierarchical + reactive)
  - Flesh out the details of the abstraction

How to express the collective ?

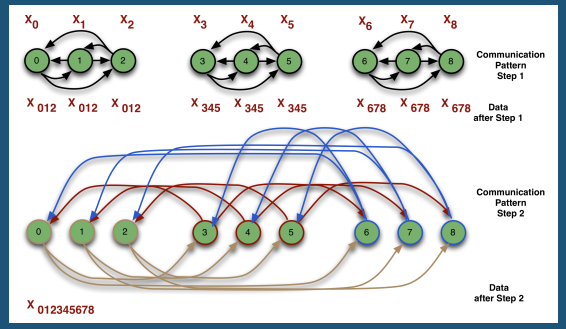
- Abstractions
  - Schedule:
    - Expresses complete collective operation (Allreduce, Reduce, Alltoall) for a single process/rank/thread
    - Includes tasks that needs to be executed for a collective operation for a single process/rank/thread
  - Tasks:
    - A set of communication operations abstracting a single step in the algorithm
    - A task includes
      - A set of send and receive operations to complete a step (in single hierarchy / flat hierarchy implementation)
      - A collective operation (in hierarchical based implementation)
    - Let's discuss with an example, what is a schedule and tasks
  - Agree on the main abstractions required
  - Names can be changed

Example: Allreduce using Recursing K-ing pattern (No hierarchy)

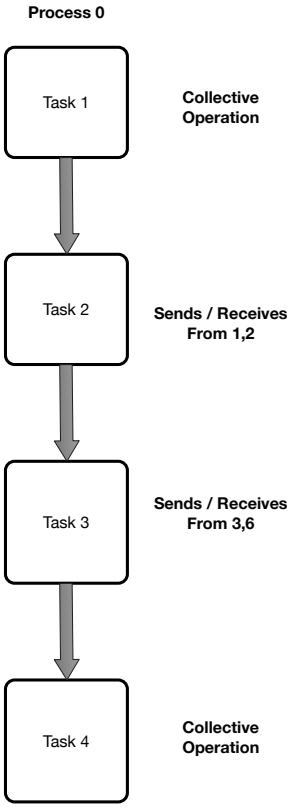
- Intentionally choosing a complex pattern



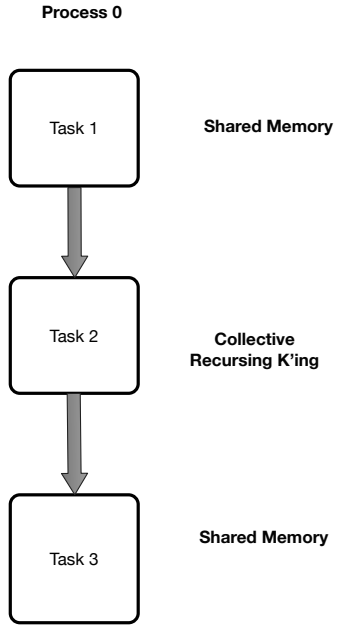
Extend Schedules and Tasks to Hierarchies



Collective operation (SHARP / Shared Memory)

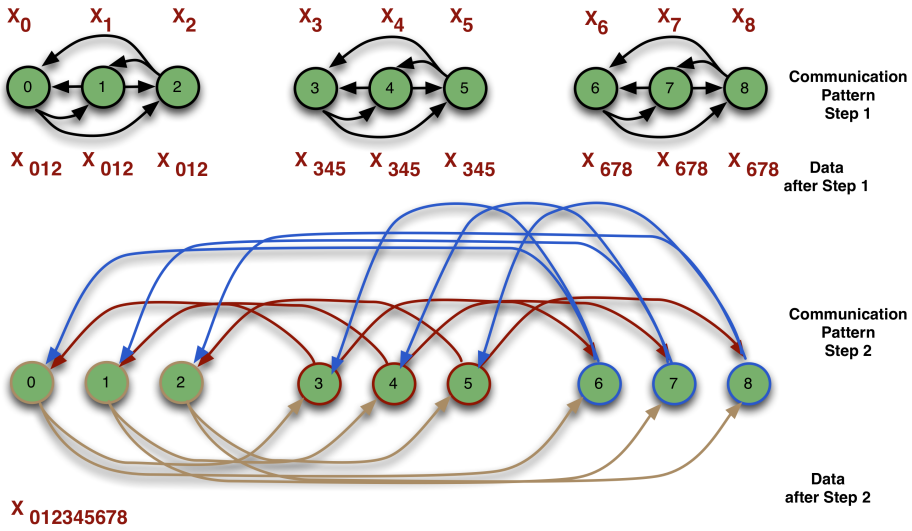


Approach 1



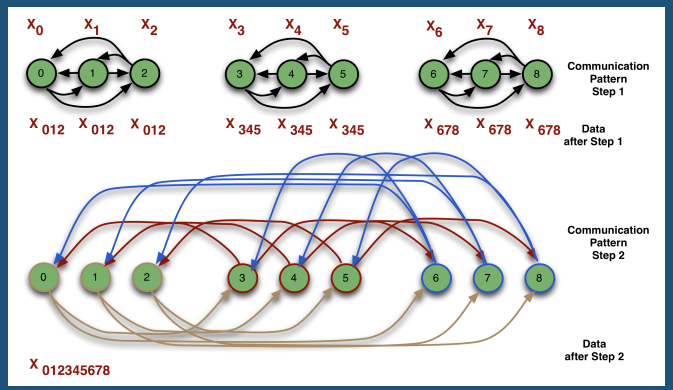
Approach 2

Example: Allreduce using Recursing K-ing pattern (No hierarchy)



- Schedule
  - Number of steps (1 and 2) in the algorithm
  - User provided info
    - Team information
    - Collective Input / Output buffers
    - Operations
  - Fragmentation information
  - What is missing ?
- Task
  - Number of sends and receives at each step (2 sends and 2 receives)
  - Communication info
    - Buffer information for send and receive
  - Network transport information
  - Preprocessing function
  - Postprocessing function
  - How to find the next task to trigger and how to trigger?
- Other optimizations
  - How to differentiate between static/dynamic information ?
- Book keeping information not captured

Example: Allreduce using Recursing K-ing pattern (With hierarchy)



Collective operation (SHARP / Shared Memory)

- Schedule
  - Number of steps (1 and 2) in the algorithm
  - User provided info
    - Team information
    - Collective Input / Output buffers
    - Operations
  - Fragmentation information
  - Relationship between the steps
  - What is missing ?
- Task
  - Number of sends and receives at each step (2 sends and 2 receives)
  - Communication info
    - Buffer information for send and receive
    - Collective primitive information
  - Network transport information
  - Preprocessing function
  - Postprocessing function
  - How to find the next task to trigger and how to trigger?
- Other optimizations
- Book keeping information not captured

Summarize : Differences between two approaches

- Schedule
  - Relationship between the steps
    - Hierarchical
      - May need to express different ordering (sequential, no-ordering)
    - Reactive
      - Might not need this information as the ordering is captured by the callbacks

- Tasks
  - How to trigger the next task ?
    - Hierarchical
      - Progress – Progresses the sends and receives
      - Trigger next task – by the progress function
    - Reactive
      - Progress – Progresses the sends and receives
      - Trigger next task – by a callback
  - Need to express collective, p2p, or both
    - Hierarchical – p2p, collective
    - Reactive – p2p

Data Structures and APIs

- Schedule
  - Number of steps
  - User info (Teams, collective operations)
  - Tasks (Array, list)
- Task
  - Triggering info
    - Completion callback
    - Threshold info (Number of receives, completion of sends, and next task)
    - ???
  - Communication info
    - P2p info (Buffer info)
    - Collective info
      - Collective function, Progress function (?)
  - Fragmentation information
  - Network information
  - Preprocessing function
  - Postprocessing function

APIs

- Create and Destroy (Schedule, Task)
- Progress Tasks

Datatypes in UCC: Approach

- Option 1
  - Basic types + Stride / IOV (similar to UCX)
  - For other datatypes, it is user's responsibility (rely on UDT – Pavan's WG)
- Option 2
  - Basic types + Stride/IOV + Pair types
  - For other datatypes, it is user's responsibility
- Option 3
  - Define all types required for MPI (OpenSHMEM is a small subset of MPI)
    - Basic types, Pair types, Stride, IOV, Subarrays …

Collective Buffer Arguments

```
typedef struct ucc_coll_buffer_info {
    uint64_t        mask;
    void            *src_buffer;
    uint32_t        *scounts;
    uint32_t        *src_displacements;
    void            *dst_buffer;
    uint32_t        *dst_counts;
    uint32_t        *dst_displacements;
    ucc_datatype_t  src_datatype;
    ucc_datatype_t  dst_datatype;
    uint64_t        flags;
} ucc_coll_buffer_info_t
```

Counts and Displacement

- Counts and Displacement as uint32_t
  - Works for MPI 3.1 and OpenSHMEM 1.5

- Problem:

  We need to support both “small” and “large” counts for collectives (moves large amount of data) and displacements to reflect 64-bit address
  - MPI 4.0 is introducing large count/displacement (passed first vote in June) ☺
    - This introduces new collectives MPI_Bcast_l, MPI_Scatterv_l
    - https://github.com/mpi-forum/mpi-issues/issues/137
  - Need to support both uint32_t and uint64_t (or higher) for MPI 4.0

- Solution:
  - Option 1: Add UCC_Count and UCC_Aint datatypes and use ”hint” flags
  - Option 2: Add uint32_t and uint64_t and optionally select using field mask
  - Option 3: Add uint64_t and cast for each collective.

Counts and Displacement: Solution Options

Option 1: Add UCC_Count and UCC_Aint datatypes and use ”hint” flags

Option 2: Add uint32_t and uint64_t and optionally select using field mask

Option 3: Add uint64_t and cast for each collective.

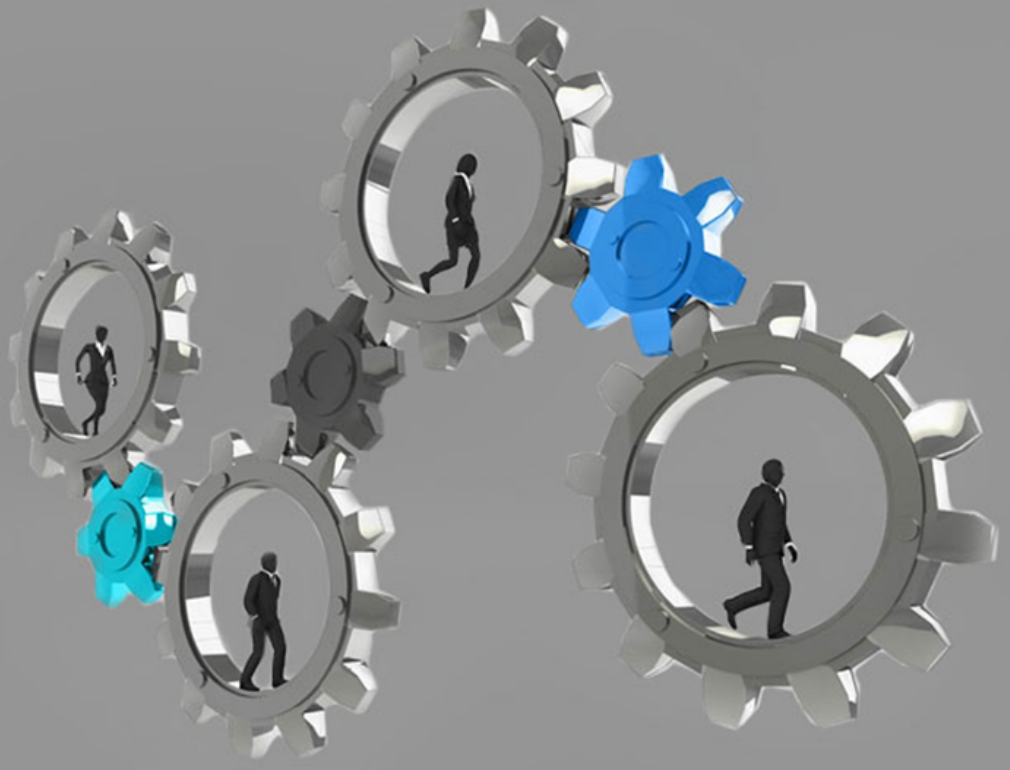
```
typedef struct ucc_coll_buffer_info {
    uint64_t        mask;
    void            *src_buffer;
    ucc_count_t     *scounts;
    ucc_aint_t      *src_displacements;
    void            *dst_buffer;
    ucc_count_t     *dst_counts;
    ucc_aint_t      *dst_displacements;
    ucc_datatype_t  src_datatype;
    ucc_datatype_t  dst_datatype;
    uint64_t        flags;
} ucc_coll_buffer_info_t

typedef struct ucc_coll_buffer_info {
    uint64_t        mask;
    void            *src_buffer;
    uint32_t        *scounts;
    uint32_t        *src_displacements;
    uint64_t        *scounts_l;
    uint64_t        *src_displacements_l;
    void            *dst_buffer;
    uint32_t        *dst_counts;
    uint32_t        *dst_displacements;
    uint32_t       *dst_counts_l;
    uint32_t        *dst_displacements_l;
    ucc_datatype_t  src_datatype;
    ucc_datatype_t  dst_datatype;
    uint64_t        flags;
} ucc_coll_buffer_info_t

typedef struct ucc_coll_buffer_info {
    uint64_t        mask;
    void            *src_buffer;
    uint64_t        *scounts_l;
    uint64_t        *src_displacements_l;
    void            *dst_buffer;
    uint32_t        *dst_counts_l;
    uint32_t        *dst_displacements_l;
    ucc_datatype_t  src_datatype;
    ucc_datatype_t  dst_datatype;
    uint64_t        flags;
} ucc_coll_buffer_info_t
```

ENABLER OF CO-DESIGN

UCF



Thank You

The UCF Consortium is a collaboration between industry, laboratories, and academia to create production grade communication frameworks and open standards for data centric and high-performance applications.