Outline

1. UCG status (and how did we get to it)

2. UCG in the software stack

3. UCG & UCC – where is this going?

4. Huawei’s roadmap for collective operations
A Brief history of UCG (G for Groups) and UCC

- [Aug. 2018] UCG Started (just me, soon after joining Huawei)
- [Dec. 2018] UCG API submitted for upstreaming (#3091 #3602)
- [Nov. 2019] Talks about upstreaming UCG (#4545) – didn’t work out...
- [Dec. 2019] UCF formed the UCC working-group
- [Sep. 2020] Huawei Cloud officially publishes its "High-Performance Communication Library"
- [Sep. 2020] UCC’s external API finalized (#1)
- [Dec. 2020] ... this update.

*Any UCX version?! Yes. See next slide.
Current Status

Right now - there are too many versions!
1. my personal UCG repo (deprecated, finally 😊)
2. xUCG (OpenUCX github)
   - This is my “master” branch – should fit ANY* UCX version, not 100% stable...
3. Huawei’s xUCG (Huawei’s github)
   - Recently created by the team in China – assumes UCX v1.6, very stable
4. Huawei’s internal UCG (Steady release schedule - within Hyper-MPI)
   - Includes some proprietary extensions, but mostly just experimental code

*Any UCX version?! Yes. See next slide.
UCG Today

Supports:
• a range of collectives (Bcast, Reduce, Allreduce, Barrier, Scatter, Gather, Allgather),
• any datatype Open-MPI does,
• any transports UCT does,
• any protocol UCP does, (not even close... but we’re working on it!)
• various hardware platforms.

Also: tested up to 256 (x86) nodes and up to 256 (ARM) cores on a host.
Outline

1. UCG status (and how did we get to it)

2. UCG in the software stack

3. UCG & UCC – where is this going?

4. Huawei’s roadmap for collective operations
UCG in the software stack

APP: OpenFOAM, WRF, ......, NAMD

OpenMPI layer
- MPI P2P: OB1, UCX
- MPI Collective: Tuned, UCX
- ORTE, OPAL

OpenUCX
- UCP: Tag-matched send/recv, Remote memory get/put
- Active Messages
- UCT: Transport: sm, rc, ud, rc/ud for Kunpeng, dc, knem
- UCG: Topology-ware algorithms, SHM-based algorithms
- Plan&Phase, trigger/exec
- UCS

OFED
- Verbs

HW
- Driver: librdmacm, libibverbs, Lib for kunpeng
- NIC (IB/RoCE)
Making UCG work with ANY UCX version

**Challenge**: keep using UCX internal APIs (UCS / UCT) as it evolves

**Incident #1** - Change-ID 8da6a5be2e:
- typedef void (*uct_completion_callback_t)(uct_completion_t *self,
  - ucs_status_t status);
+ typedef void (*uct_completion_callback_t)(uct_completion_t *self);

**Incident #2** - Change-ID fca960826a:
- #define UCS_CONFIG_REGISTER_TABLE_ENTRY(_entry) \
  + #define UCS_CONFIG_REGISTER_TABLE_ENTRY(_entry, _list) \
  ...
Making UCG work with ANY UCX version

**Challenge**: keep using UCX internal APIs (UCS / UCT) as it evolves
*Not as hard as it sounds, actually.

**Incident #1 - Change-ID 8da6a5be2e:**
```c
#include "uct/api/uct.h"

uct_completion_callback_t func = NULL;
func(NULL, UCS_OK);
```

**Incident #2 - Change-ID fca960826a:**
```c
#include "ucs/config/parser.h"
#undef UCS_CONFIG_REGISTER_TABLE_ENTRY
#define UCS_CONFIG_REGISTER_TABLE_ENTRY(a, b)
[UCS_CONFIG_REGISTER_TABLE_ENTRY(NULL, NULL,
```
APIs

• “Northbound”:
  • Generic (src/ucg/api/ucg.h): ucg_group_create, ucg_collective_create
  • MPI-specific (src/ucg/api/ucg_mpi.h): MPI_Reduce(),
  • Versioning (src/ucg/api/ucg_version.h): similar to ucp_version.h

• “Southbound”:
  • Plan Components (src/ucg/api/ucg_plan_components.h):
    - for accommodating multiple implementations (similar to OMPI’s MCA COLL)
  • Expected components: “builtin” (day-1), hicoll, NCCL?
### Northbound API, side-by-side (Blocking)

<table>
<thead>
<tr>
<th>MPI Application (or MPI-specific API)</th>
<th>UCX Groups (UCG) Equivalent (Generic API)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPI_Bcast</strong> (bufA, ..., rowcomm)</td>
<td>paramsA.buf = bufA;</td>
</tr>
<tr>
<td></td>
<td>paramsA.type.modifiers = BCAST_MODIFIERS;</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>ucg_collective_create(rowGroup, &amp;paramsA, &amp;collA);</td>
</tr>
<tr>
<td></td>
<td>ucg_collective_start_nb(collA, &amp;reqA);</td>
</tr>
<tr>
<td></td>
<td>MPI_Wait(reqA, &amp;status);</td>
</tr>
</tbody>
</table>
Northbound API, side-by-side (Non-blocking)

<table>
<thead>
<tr>
<th>MPI Application (or MPI-specific API)</th>
<th>UCX Groups (UCG) Equivalent (Generic API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Ibcast_init(..., &amp;req);</td>
<td>ucg_collective_create(Group, ..., &amp;coll);</td>
</tr>
<tr>
<td>for (i=0; i&lt;MAXITER; i++) {</td>
<td>for (i=0; i&lt;MAXITER; i++) {</td>
</tr>
<tr>
<td>compute(buf);</td>
<td>compute(buf);</td>
</tr>
<tr>
<td>MPI_Start(req);</td>
<td>ucg_collective_start_nb(coll, &amp;req);</td>
</tr>
<tr>
<td>MPI_Wait(req, &amp;status);</td>
<td>MPI_Wait(req, &amp;status);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>MPI_Request_free(req);</td>
<td>ucg_request_free(req);</td>
</tr>
</tbody>
</table>
Decoupling Planning and Execution – Recursive Doubling

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Ranks</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Send a message and also wait for one</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Send a message and also wait for one</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>Send a message and also wait for one</td>
</tr>
</tbody>
</table>
Decoupling Planning and Execution – Tree-based

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Ranks</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,3</td>
<td>Wait for a message</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Send a message</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Wait for a message</td>
</tr>
<tr>
<td>4</td>
<td>2,3</td>
<td>Send a message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Ranks</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,3</td>
<td>Action: Wait for a message from each node</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Incoming buffer:</strong> temporary buffer of 24 bytes which starts with the user’s input</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Incoming length:</strong> 16 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Incoming offset:</strong> 8 bytes</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Action: Send a message to each node</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Outgoing buffer:</strong> the temporary buffer from the previous step</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Outgoing length:</strong> 24 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Outgoing offset:</strong> 0 bytes</td>
</tr>
</tbody>
</table>

8-byte call to MPI_Gather()

*Is tree the best choice for MPI_Gather? Not always...
Decoupling Planning and Execution – Tree-based

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Ranks</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,3</td>
<td>Wait for a message</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Send a message</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Wait for a message</td>
</tr>
<tr>
<td>4</td>
<td>2,3</td>
<td>Send a message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Ranks</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| 3       | 0     | **Action**: Wait for a message from each node  
Incoming buffer: destination buffer  
Incoming length: 8 bytes  
Incoming offset: 0 bytes |
| 4       | 2,3   | **Action**: Send a message to each node  
Outgoing buffer: destination buffer  
Outgoing length: 8 bytes  
Outgoing offset: 0 bytes |

8-byte call to MPI_Bcast()
Outline

1. UCG status (and how did we get to it)

2. UCG in the software stack

3. UCG & UCC – where is this going?

4. Huawei’s roadmap for collective operations
My point of view on UCC

UCC is coming about, slowly – it’ll be a while before we can use it, and...

- UCC looks to me like a big performance risk!
- UCC adds levels of abstraction, possibly returning issues from Open-MPI
- Example #1: UCC is only limited to UCX API and cannot access internals.
- Example #2: progress is not aware whether SHARP is used at the moment.
- Development is "breadth-first" instead of "depth-first"
- Preliminary performance results are not expected any time soon.
- UCC is practically just two participants: Mellanox and Huawei.
- Missing representation from other vendors and potential users.
My point of view on UCC

UCC is coming about, slowly – it’ll be a while before we can use it, and...

UCC looks to me like a big performance risk!

- UCC adds levels of abstraction, possibly returning issues from Open-MPI
  - Example #1: UCC is limited to UCP API... how/when to use atomics for reduction?
  - Example #2: progress is not aware whether SHARP is used at the moment.

Development is "breadth-first" instead of "depth-first"

Preliminary performance results are not expected any time soon.

UCC is practically just two participants: Mellanox and Huawei.

Missing representation from other vendors and potential users.
My point of view on UCC

UCC is coming about, slowly – it’ll be a while before we can use it, and...

UCC looks to me like a big performance risk!

- UCC adds levels of abstraction, possibly returning issues from Open-MPI
  - Example #1: UCC is limited to UCP API... how/when to use atomics for reduction?
  - Example #2: progress is not aware whether SHARP is used at the moment.

- Development is “breadth-first” instead of “depth-first”
  - Preliminary performance results are not expected any time soon.

- UCC is practically just two participants: Mellanox and Huawei.
  - Missing representation from other vendors and potential users.
Challenges, looking forward

1. Must show the approach is beneficial - before investing much effort
   - I don’t think xCCL’s or xUCG’s individual performance is proof enough...
Challenges, looking forward

1. Must show the approach is beneficial - before investing much effort
   - I don’t think xCCL’s or xUCG’s individual performance is proof enough...

2. Community involvement:
   • More contributing parties
   • More representation for use-cases
   • More exposure, feedback, stakeholders...
Challenges, looking forward

1. Must show the approach is beneficial - before investing much effort
   - I don’t think xCCL’s or xUCG’s individual performance is proof enough...

2. Community involvement:
   • More contributing parties
   • More representation for use-cases
   • More exposure, feedback, stakeholders...

3. Delivering relevant, high-quality software.
UCG

Not going anywhere, anytime soon... no open-source alternative (yet)

3 possible paths:
1. Merge into UCC
2. Merge into Huawei’s UCX package
3. Merge into upstream UCX
Outline

1. UCG status (and how did we get to it)
2. UCG in the software stack
3. UCG & UCC – where is this going?
4. Huawei’s roadmap for collective operations
Development Roadmap

• Hardware-specific capabilities
  • Kunpeng CPU
  • Storage product-lines
  • Atlas AI accelerator product-lines

• Integration with related HPC Software
  • HPC Storage and parallel filesystems, e.g. MPI I/O
  • Batch scheduler, e.g. SLURM (for job information)
  • Builders, e.g. Spack (provide collectives for other packages)

• Custom acceleration hints (through MPI?)
  • Persistent operations
  • Read-only / write-only buffers