UCX protocols v2 update

Yossi Itigin | UCF Workshop 2022

Proto v2 basics

- Goals
  - Separate protocol definition from selection logic
  - Build protocols independently from common infrastructure
  - Common debug/tracing utilities
  - Account for datatype, topology, and operation flags during selection
  - Allow protocol restart when endpoint configuration changed

- Protocol building blocks:
  - **ucp_proto_t** – Static protocol definition, including protocol name, init, progress, and abort callbacks
  - **ucp_proto_select_param_t** – Operation parameters that influence protocol selection, except for data size. Usually there is a small number of possible combinations in one application.
  - **ucp_proto_init_params_t** – Common parameters that specify how to initialize a protocol. This structure is extended by specific protocols, for example **ucp_proto_rndv_ctrl_init_params_t**
  - **ucp_proto_caps_t** – Output of protocol initialization per specific selection parameters, that include supported range and performance estimation in that range (time as function of message size). Used by the selection logic.
  - **proto_single_*** – Family of internal APIs and structures for protocols that sends only one message per UCP request.
  - **proto_multi_*** - Family of internal APIs and structures for protocols that send multiple messages per UCP request, including fragmentation and and utilizing multiple lanes.
  - **proto_select_*** - Functions and structures that drive the common selection logic. Typically, not used directly by protocols.

Proto v2 rendezvous

- Selection process is similar to “minimax” algorithm:
  - Performance estimation of a rendezvous protocol is based on “playing out” the best-protocol selection of the remote peer
  - All rendezvous protocols start with “rndv/rts”
  - “rndv/rts” is derived from rndv_ctrl protocol – which generates the remote selection parameters for RNDV_RECV operation.
  - RNDV_RECV operation can select from rndv/get/zcopy, rndv/get/mtype, rndv/rtr, etc.
  - “rndv/rtr” protocol is also derived from rndv_ctrl – for RNDV_SEND operation. It can select from rndv/put/zcopy, rndv/am/bcopy, rndv/put/mtype, etc.
  - Eventually, the best rendezvous protocol is selected, and compared with other options such as eager/bcopy, eager/zcopy.
- When RTS or RTR message arrives, a rendezvous send request is initialized with the remote buffer information, and the selection process happens again. It may reuse cached decisions from the initial selection process.
- Pipeline protocols have extra level of selection
  - “rndv/send/ppln” and “rndv/recv/ppln” are selecting pipeline-capable protocols: rndv/put/mtype, rndv/get/mtype, rndv/rtr/mtype.
  - Break the top-level send request to smaller fragment-requests.

Proto v2 latest additions

Optional subtitle

- Active messages with new protocols
- Rendezvous protocol to use transport-optimal alignment
- Hardware tag matching: eager merged, rendezvous in review
- Moved memory registration cache to UCP layer
- Error handling flows: protocol abort and reset
- Debug info: Performance tree, protocol select info
- Improved PCIe topology detection
- Support more lanes per UCP endpoint – to enable GPU-aware selection
- GPU memory 2-stage pipeline – in review

Plans for 2023

- Performance tuning
  - Blocking/nonblocking send (NBR)
  - Optimal number of fragments based on message size
  - More accurate performance estimation
- Enable proto v2 by default, and remove proto v1 code
- Zero-copy rendezvous with IOV datatype
- Register a memory region only after it was used few times
- Fallback to bcopy if failed to register memory
- GPU memory pipeline

Active messages

- Full compatibility with v1
- Separate small message protocols with reply-ep, to reduce runtime branches
  - Eager short with/without reply-ep
  - Eager bcopy/zcopy single with/without reply-ep
  - Eager bcopy/zcopy multi-fragment
  - Rendezvous
- Support UCP_AM_SEND_FLAG_COPY_HEADER flag – in review

Registration cache in UCP

- Will replace UCT-level local registration caches
  - Currently still exist in IB and KNEM because proto v1 does not use the UCP registration cache
- Lower zero-copy overhead
  - Avoid rwlock and rely on UCP context lock
  - Inline rcache functions into UCP layer
- Query dmabuf fd only during cache miss
- Can track memory region usage before paying the cost of registration

Error handling / config change flows

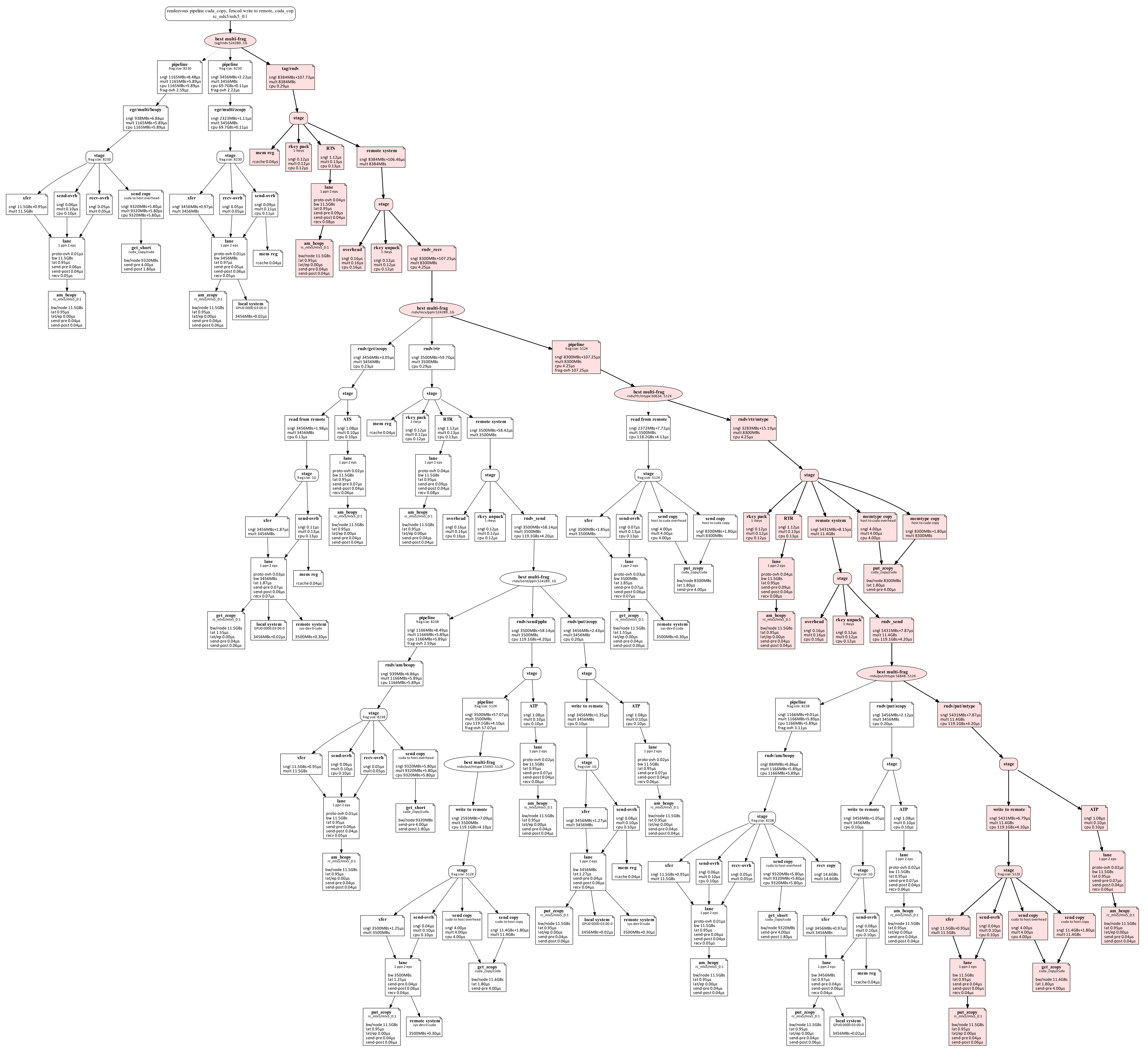
- Each protocol defines “abort” and “reset” callbacks
  - With proto v1, a single function “ucp_request_send_state_ff” handles all protocols
- “**abort**” completes the send request with error when there is a network error on the endpoint, or a UCT operation failed as part of the protocol progress.
- “**reset**” cleans up send request resources allocated by the protocol and brings the request back to its initial state, before the protocol started to progress. Used to re-select a protocol when endpoint configuration changes.
- WIP
  - Invalidate remote memory keys when rendezvous protocol is aborted.
  - Increase CI coverage of protocol abort and reset flows.

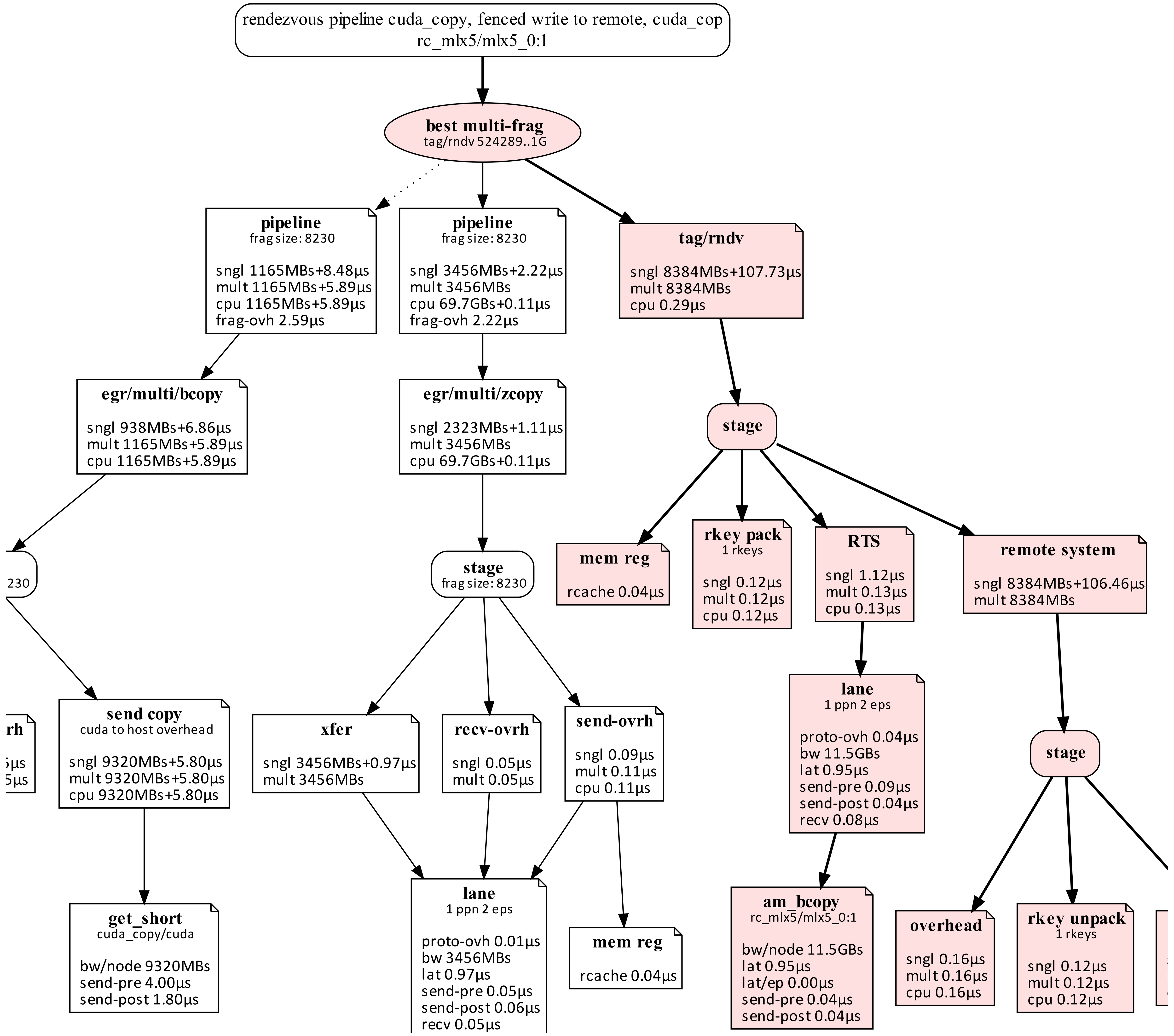
Protocol selection info

```
$ mpirun -n 2 -mca pml_ucx_multi_send_nb 1 --map-by node \
      -x UCX_NET_DEVICES=mlx5_0:1 -x UCX_PROTO_ENABLE=y -x UCX_PROTO_INFO=y \
      osu_bw -m 128:1048576 -W 256 D D

...

# OSU MPI-CUDA Bandwidth Test v5.8
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)

...

[1663677747.261504] [vulcan02:8949 :0]   +----------------+-------------------------------------------------------------------------+
[1663677747.261515] [vulcan02:8949 :0]   | mpi inter-node | rendezvous data fetch(multi) into cuda/GPU0 from cuda/dev[0]            |
[1663677747.261517] [vulcan02:8949 :0]   +----------------+-------------------------------------------------------+-----------------+
[1663677747.261519] [vulcan02:8949 :0]   |              0 | no data fetch                                         |                 |
[1663677747.261521] [vulcan02:8949 :0]   |          1..64 | (?) zero-copy fenced write to remote                  | rc_mlx5/mlx5_0:1 |
[1663677747.261523] [vulcan02:8949 :0]   |      65..16727 | zero-copy read from remote                            | rc_mlx5/mlx5_0:1 |
[1663677747.261530] [vulcan02:8949 :0]   |    16728..512K | (?) cuda_copy, fenced write to remote, cuda_copy      | rc_mlx5/mlx5_0:1 |
[1663677747.261532] [vulcan02:8949 :0]   |    524289..inf | pipeline cuda_copy, fenced write to remote, cuda_copy | rc_mlx5/mlx5_0:1 |
[1663677747.261535] [vulcan02:8949 :0]   +----------------+-------------------------------------------------------+-----------------+

…
```

Protocol selection detail

```
$ mpirun -n 2 -mca pml_ucx_multi_send_nb 1 --map-by node \
      -x UCX_NET_DEVICES=mlx5_0:1 -x UCX_PROTO_ENABLE=y -x UCX_PROTO_INFO_DIR=<dirpath> \
      osu_bw -m 128:1048576 -W 256 D D
```





https://gist.githubusercontent.com/yosefe/e30911168ef6c683d9495c70ad502ca9/raw/f3dbdaaf42af6a14d8032b7436a78a2c9e60b0e1/proto_select_demo1.svg

PCIe topology detection

- Use sysfs directory structure to determine system distance
- Classify maximal common ancestor:
  - System root (inter-socket)
  - PCIe root (same root complex) – bandwidth is inverse proportional to max amount of different path components
- Added “ucx_info –T“ to show system topology:

```
[user@swx-dgx01 ucx]$ ucx_info -T
#
# System topology:
#
```

MB/s	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	mlx5_0	mlx5_1	mlx5_2	mlx5_3
GPU0	-	inf	2400.0	2400.0	220.0	220.0	220.0	220.0	inf	2400.0	220.0	220.0
GPU1	inf	-	2400.0	2400.0	220.0	220.0	220.0	220.0	inf	2400.0	220.0	220.0
GPU2	2400.0	2400.0	-	inf	220.0	220.0	220.0	220.0	2400.0	inf	220.0	220.0
GPU3	2400.0	2400.0	inf	-	220.0	220.0	220.0	220.0	2400.0	inf	220.0	220.0
GPU4	220.0	220.0	220.0	220.0	-	inf	2400.0	2400.0	220.0	220.0	inf	2400.0
GPU5	220.0	220.0	220.0	220.0	inf	-	2400.0	2400.0	220.0	220.0	inf	2400.0
GPU6	220.0	220.0	220.0	220.0	2400.0	2400.0	-	inf	220.0	220.0	2400.0	inf
GPU7	220.0	220.0	220.0	220.0	2400.0	2400.0	inf	-	220.0	220.0	2400.0	inf
mlx5_0	inf	inf	2400.0	2400.0	220.0	220.0	220.0	220.0	-	2400.0	220.0	220.0
mlx5_1	2400.0	2400.0	inf	inf	220.0	220.0	220.0	220.0	2400.0	-	220.0	220.0
mlx5_2	220.0	220.0	220.0	220.0	inf	inf	2400.0	2400.0	220.0	220.0	-	2400.0
mlx5_3	220.0	220.0	220.0	220.0	2400.0	2400.0	inf	inf	220.0	220.0	2400.0	-

More lanes per UCP endpoint

- Until recently, a single UCP endpoint was limited to 6 lanes (devices/transports)
- Systems with multiple NICs and CPUs (DGX):
  - Currently we select best NIC only if GPU is known up-front
  - We want to select best NIC at runtime, according to GPU pointer and topology distance
- Extended UCP endpoint to support “fast-path” and “slow-path” lanes
  - Systems with small amount of NICs will not have memory overhead increase
  - Allocate slow-path lanes array dynamically according to actual number
- Up to 8 lanes - merged, up to 16 lanes - in review

NVIDIA®