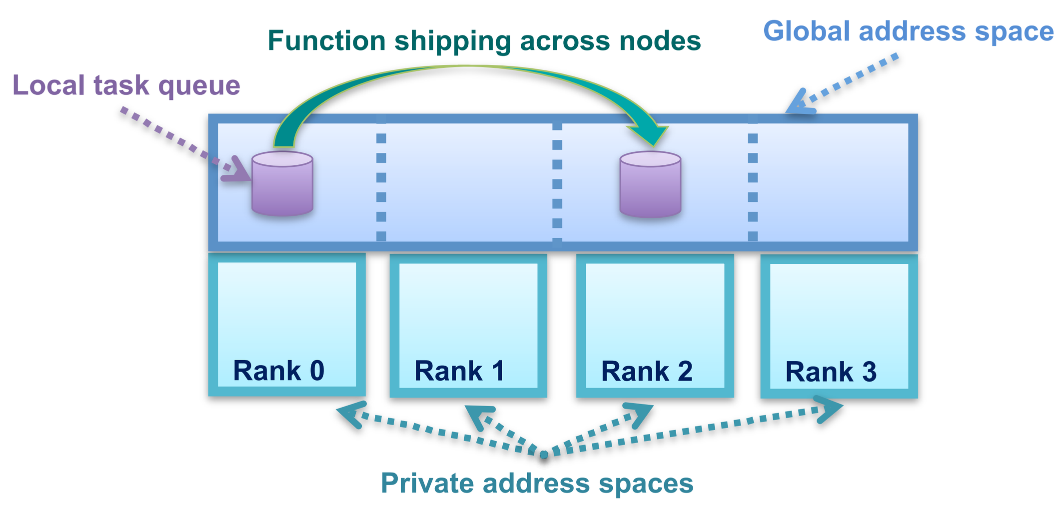
UPC++ and GASNet: PGAS Support for Exascale Apps and Runtimes (WBS 2.3.1.14)

Paul H. Hargrove (PI)

With team members: Dan Bonachea, Max Grossman, Amir Kamil, Colin A. MacLean, Daniel Waters

UPC++ at Lawrence Berkeley National Lab (upcxx.lbl.gov)

- UPC++ is a C++11 PGAS library
  - Lightweight, asynchronous, one-sided communication (RMA)
  - Asynchronous remote procedure call (RPC)
  - Data transfers may be non-contiguous
  - Futures manage asynchrony, enable communication overlap
  - Collectives, teams, remote atomic updates
  - Provides building blocks to construct irregular data structures
- Latest software release: March 2021
  - Runs on systems from laptops to supercomputers
- Easy on-ramp and integration
  - Enables incremental development
  - Selectively replace performance-critical sections with UPC++
  - Interoperable with MPI, OpenMP, CUDA, etc.



Integration efforts with ExaBiome (WBS 2.2.4.04)

- MetaHipMer 1 (MHM1) – a UPC / UPC++ hybrid code
  - In 2019, the k-mer counting step rewritten from MPI to UPC++
- MetaHipMer 2 (MHM2) – a pure UPC++ code
  - In 2020, the previous UPC stages of MHM1 rewritten in UPC++
- UPC++'s RPC is a better fit to the problem than previous alternatives
- Each rewrite reduced code size by roughly ½
- Comparable genome assembly results
- Lower memory requirements and up to 6x better performance

Integration efforts with ExaGraph (WBS 2.2.6.07)

- With PNNL team, have developed two UPC++ versions of a graph matching problem from their IPDPS'19 paper
  - RMA version uses Puts to communicate among processes
  - RPC version uses asynchronous remote procedure calls to execute logic on remote parts of the graph
- Initial results on NERSC Cori Haswell (3.6B-edge Friendster):
  - Both UPC++ versions competitive with (or better than) best MPI versions up to at least 4,096 processes

Integration efforts with NWChemEX (WBS 2.2.1.02)

- Ported TAMM code base from Global Arrays/MPI to UPC++
  - TAMM offers distributed in-memory data store and compute for NWChemEx
  - Achieved comparable performance to hardened GA code (+10-15%) after a few months of work by 1 SDE
- Continuing to identify UPC++ enabled optimization opportunities
  - Multi-threaded remote accumulators via RPC
  - Finer grain asynchrony and completion control

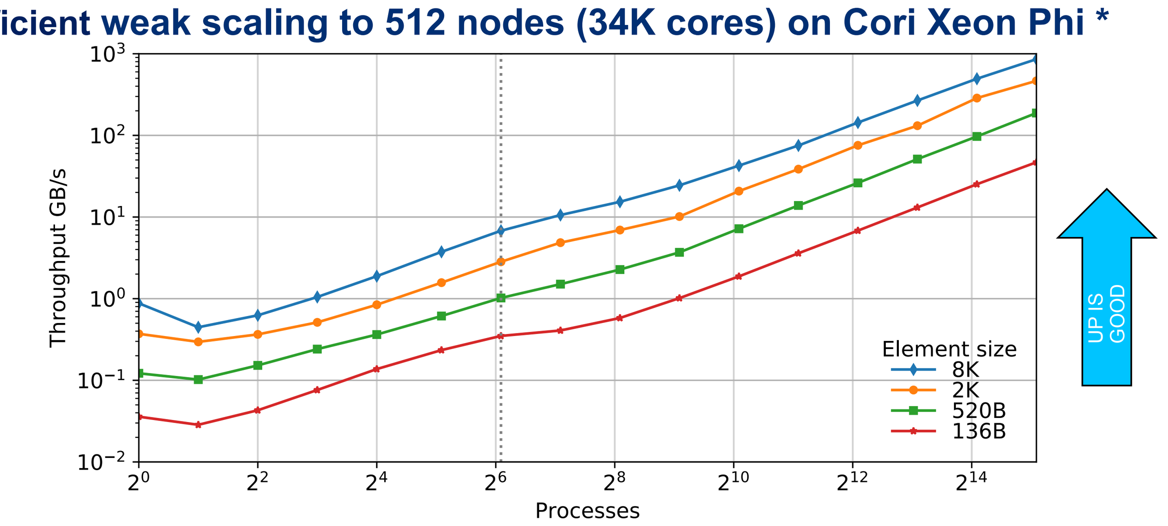
Case 1: Easy Distributed Hash-Table via Function Shipping and Futures

- **Distributed hash-table design is based on function shipping**
  - RPC inserts the key metadata at the target
  - Once the RPC completes, an attached callback issues a one-sided RMA Put (rput) to store the value data

```
// C++ global variables correspond to rank-local state
std::unordered_map<uint64_t, global_ptr<char>> local_map;
// insert a key-value pair and return a future
future<> dht_insert(uint64_t key, char *val, size_t sz) {
 future<global_ptr<char>> fut =
     rpc(key % rank_n(),                    // RPC obtains location for the data
            [key,sz]() -> global_ptr<char> {    // lambda invoked by RPC
              global_ptr<char> gptr = new_array<char>(sz);
              local_map[key] = gptr;                // insert in local map
              return gptr;
            });
 return fut.then(                          // callback executes when RPC completes
        [val,sz](global_ptr<char> loc) -> future<> {
            return rput(val, loc, sz); });    // RMA Put the value payload
}
```

- **Benefits:**
  - Use of **RPC** simplifies distributed data-structure design
    - Argument passing, remote queue management and progress engine are factored out of the application code
  - Asynchronous execution enables overlap

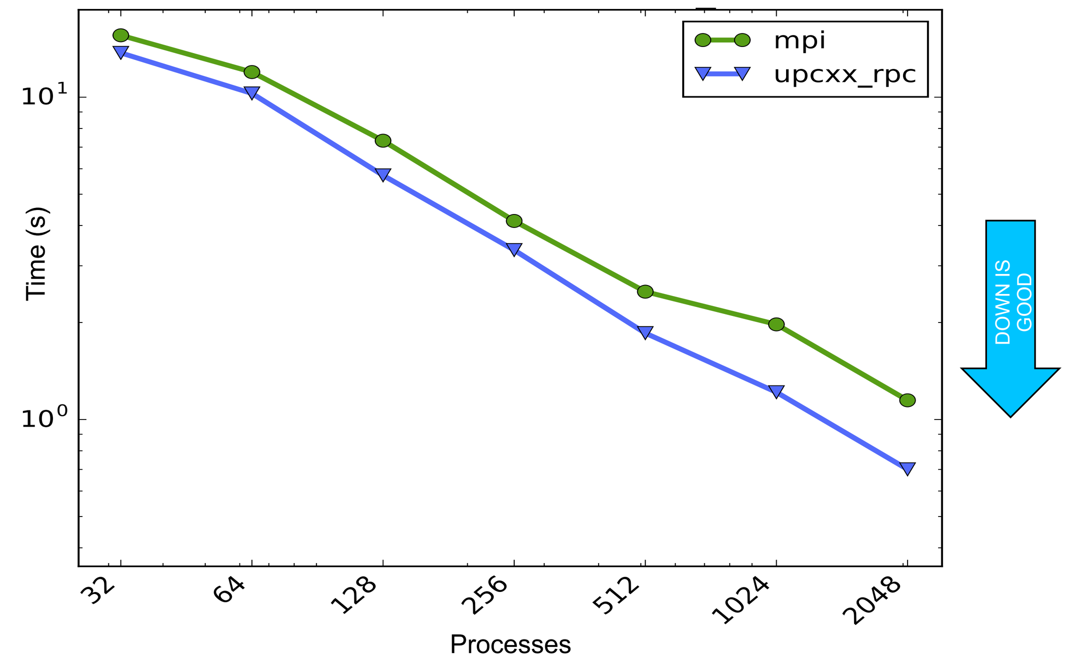
Efficient weak scaling to 512 nodes (34K cores) on Cori Xeon Phi *



Case 2: Asynchronous Sparse Matrix Solvers

- **A time consuming operation in multifrontal sparse solvers:**
  - ***Extend-add:*** update a distributed sparse matrix, scattering the packed data source
- **Challenge:**
  - This operation has low computational intensity and exhibits irregular communication patterns
- **Solution**:
  - UPC++ **function shipping** via RPC enables efficient communication and asynchrony, increasing overlap and improving performance of ***Extend-add***
- **Impact:**
  - UPC++ enhances overlap in ***Extend-add***, yielding up to a 1.63x speedup over MPI collective and 3.11x over MPI message-passing implementations. The green line in the figure corresponds to the fastest of these two variants.

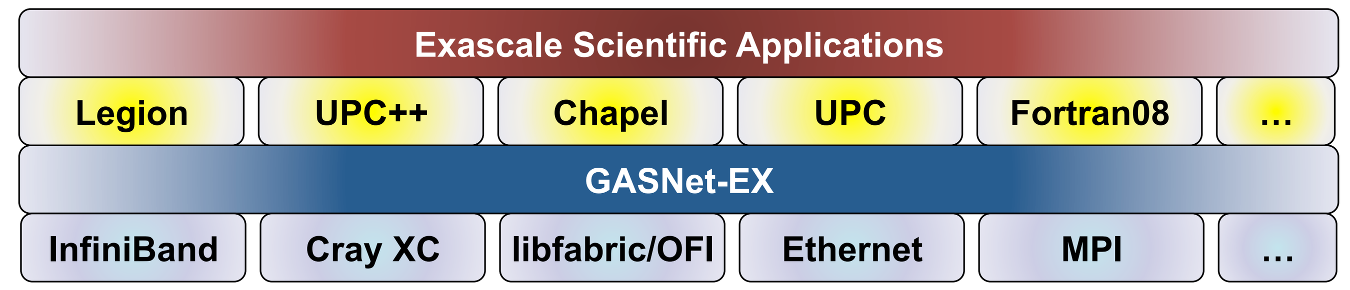
Strong scaling comparison of the UPC++ implementation of Extend-add using RPC and an MPI variant for the audikw_1 matrix on NERSC Cori Xeon Phi (using 64 cores/node) *



*** For more details see IPDPS'19. https://doi.org/10.25344/S4V88H**

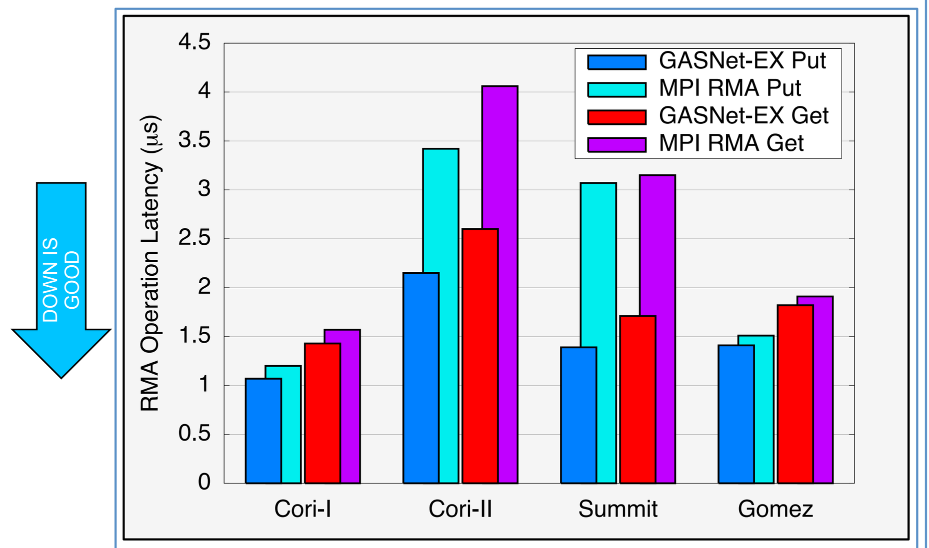
GASNet-EX at Lawrence Berkeley National Lab (gasnet.lbl.gov)

- GASNet-EX: communications middleware to support exascale clients
  - One-sided communication – Remote Memory Access (RMA)
  - Active Messages (AMs) - remote procedure call
  - Implemented over native APIs of all networks of interest to DOE
- GASNet-EX is an evolution of GASNet-1 for exascale
  - Retains GASNet-1's wide portability (laptops to supercomputers)
  - Provides backwards compatibility for the dozens of GASNet-1 clients, including multiple UPC and CAF/Fortran08 compilers
  - Focus remains on one-sided RMA and Active Messages
  - Reduces CPU and memory overheads
  - Improves many-core and multi-threading support
- Major enhancements relative to GASNet-1:
  - "Immediate mode" injection to avoid stalls due to back-pressure
  - Explicit handling of local completion (source buffer lifetime)
  - New and revised AM interfaces, including
    - E.g. "NPAM" to reduce buffer copies between layers
  - Vector-Index-Strided for non-contiguous point-to-point RMA
  - Remote Atomics, implemented with NIC offload where available
  - Subset teams and non-blocking collectives
  - RMA directly to/from device memory on supported hardware
    - E.g. GPUs on OLCF's Summit

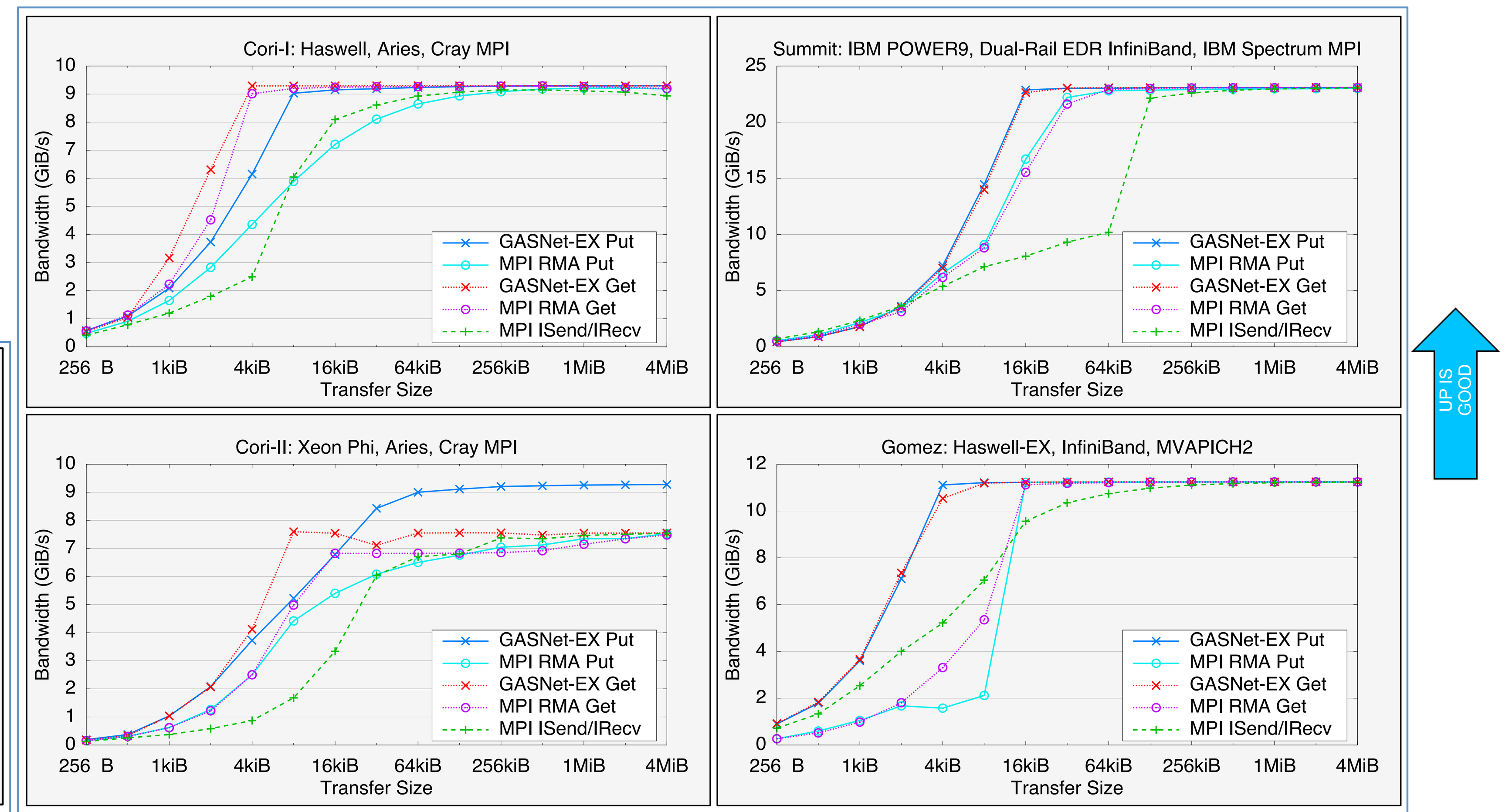


GASNet-EX RMA Performance versus MPI RMA and Isend/Irecv

- Three different MPI implementations
- Two distinct network hardware types
- On four systems the performance of GASNet-EX matches or exceeds that of MPI RMA and message-passing:
  - 8-byte Put latency 6% to 55% better
  - 8-byte Get latency 5% to 45% better
  - Better flood bandwidth efficiency, typically saturating at ½ or ¼ the transfer size



8-Byte RMA Operation Latency (one-at-a-time)



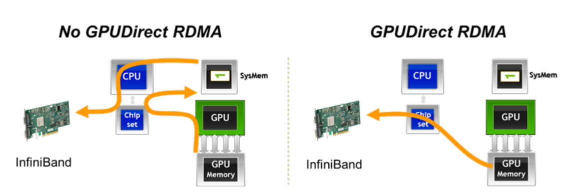
Uni-directional Flood Bandwidth (many-at-a-time)

GASNet-EX results from v2018.9.0
MPI results from Intel MPI Benchmarks v2018.1

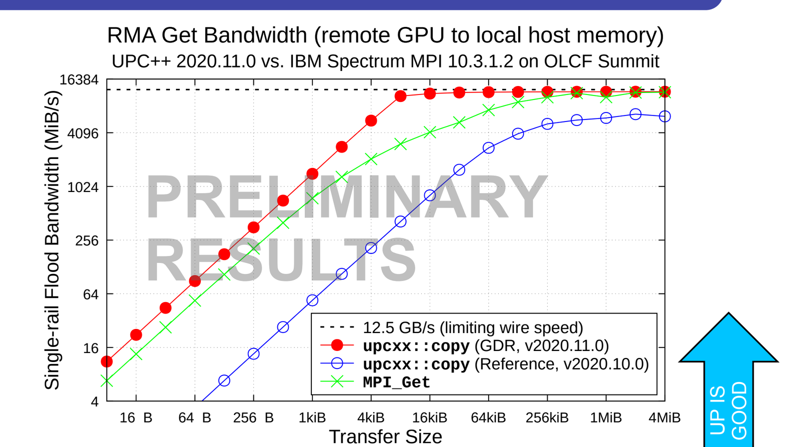
For more details see Languages and Compilers for Parallel Computing (LCPC'18). **https://doi.org/10.25344/S4QP4W**

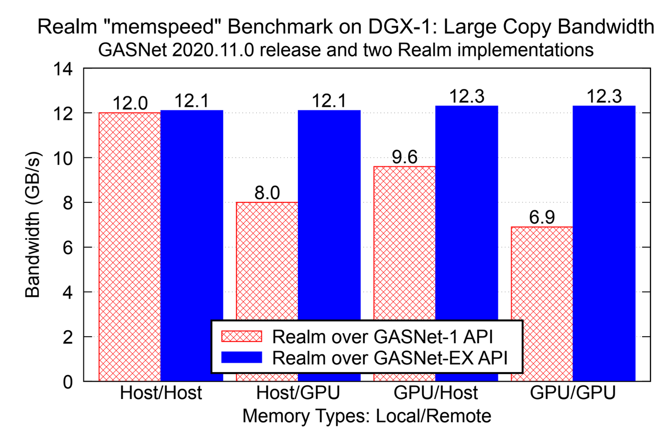
Support for GPUDirect RDMA (GDR) – UPC++ and Legion/Realm Benchmarks

- GASNex-EX supports GPUDirect RDMA (GDR) since 2020.11.0
  - Removes host CPU and memory bottlenecks from one-sided transfers to/from GPU memory (see diagram ➔)
  - Currently supports Nvidia GPUs + Mellanox InfiniBand
    - Other accelerators and networks are subject of future work



- Preliminary comparisons of UPC++ to MPI-3 RMA in GDR-enabled IBM MPI show UPC++ saturating more quickly to the peak (top-right plot)
- Realm is the low-level runtime for the Legion Programming System (WBS 2.3.1.08)
  - Communications services originally implemented over GASNet-1
  - GASNet-1 backend still works using legacy API support in current GASNet-EX
- Realm introduced a new GASNet-EX communications backend (Dec 2020)
  - Embraces capabilities specific to GASNet-EX
  - Leverages Immediate, NPAM, and local completion events for AM
  - Most notable new capability is GDR support
- Some performance benefits of using GASNet-EX's GDR support in Realm:
  - Large GPU memory xfers: same bandwidth as host memory (bottom-right plot)
  - Small GPU memory xfers: 2.2x to 3.0x latency improvement





This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.