On the Potential of Compression Hiding in MPI Applications

Yicheng Li, Michael Jantz

EuroMPI 2025

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Distributed Computing: Message Passing

- Distributed computing: Essential for solving complex computational problems across various domains of science
- Message Passing Interface (MPI): A standard or *de facto* for efficient data exchange in distributed computing
- Open MPI: An advanced open-source implementation of MPI
  - https://github.com/open-mpi/ompi

Using Compression in MPI Communication

- A major performance factor in distributed applications is the amount of data communicated among processes
- Network bandwidth is limited so it is important to minimize cost spent on communication
- Compression decreases the amount of data transmitted over the network
  - Lower bandwidth consumption
  - Enhanced communication speed
- Investigate hiding compression cost by overlapping compression with other operations (computation, communication)
  - At communication time, detect if the compressed copy is still valid

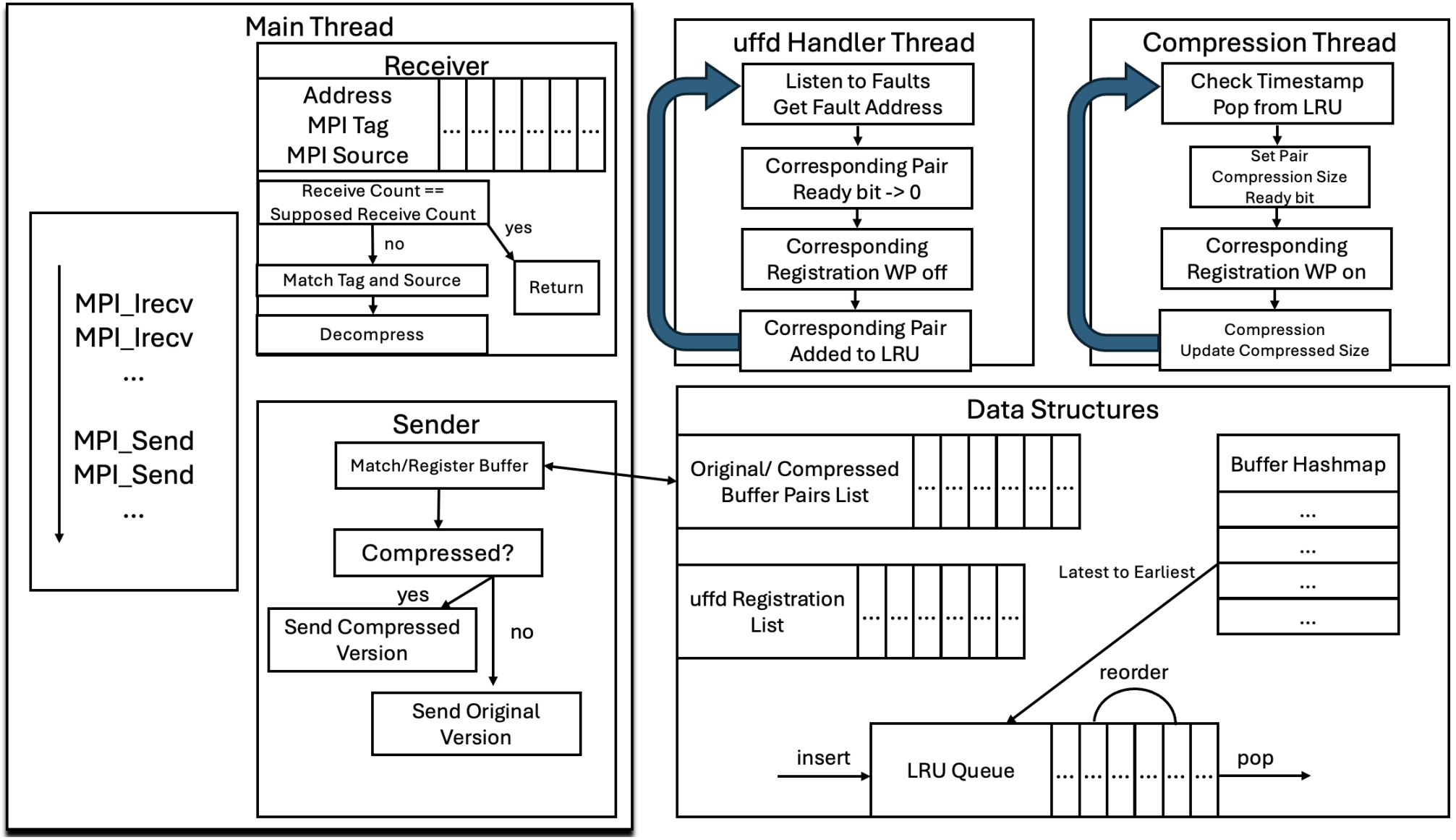
Our Approach: userfaultfd

- What is userfaultfd?
  - Linux kernel feature for handling faults in user-space processes
  - Offers applications coarse to fine grain control over memory management
- How does it work?
  - Register memory regions via userfaultfd system call
  - When faults happen in the registered region, kernel will halt
  - A separate process will be listening to the events and will handle fault
  - Kernel resumes after fault is handled

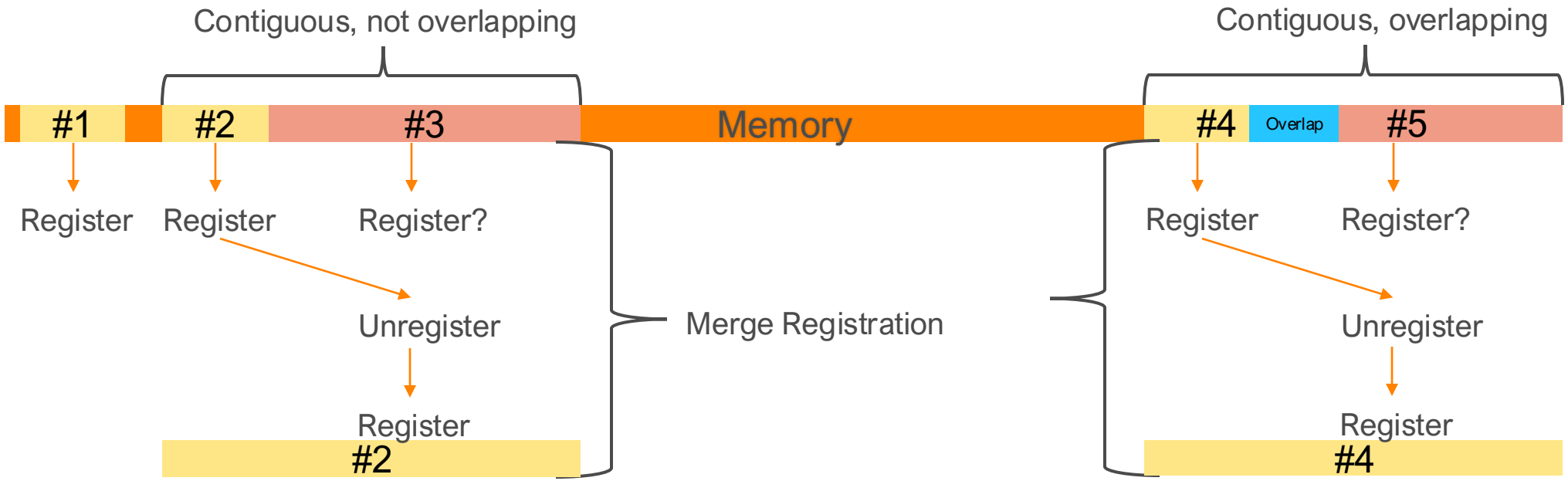
Our Approach: userfaultfd

- Application thread
  - MPI register communication buffers
  - Check whether a compressed counterpart is ready
- Compression thread
  - Check if there are buffers that needed to be compressed from the compression queue
  - Put write protect on the memory region
  - Compress
- Write handler thread
  - Listen to write fault
  - Take write protect off registered memory region
  - Push corresponding communication buffers onto compression queue

Design

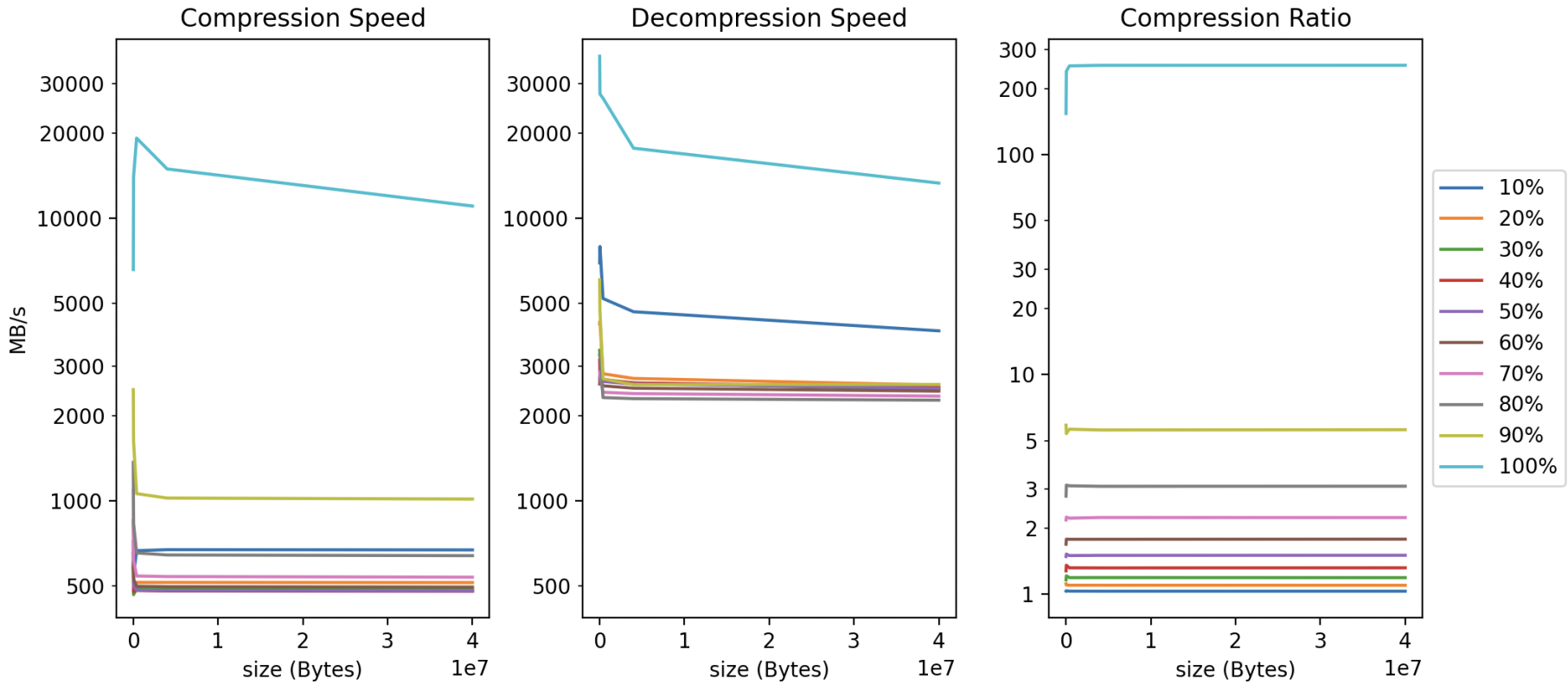


uffd Registration Strategy



LZ4 Compression Algorithm

- LZ4 is a lossless compression algorithm known for its exceptionally fast compression and decompression speeds
- LZ4 HC is the high-compression variant of LZ4, it trades off compression speed for compression ratio. But decompression speed remains the same

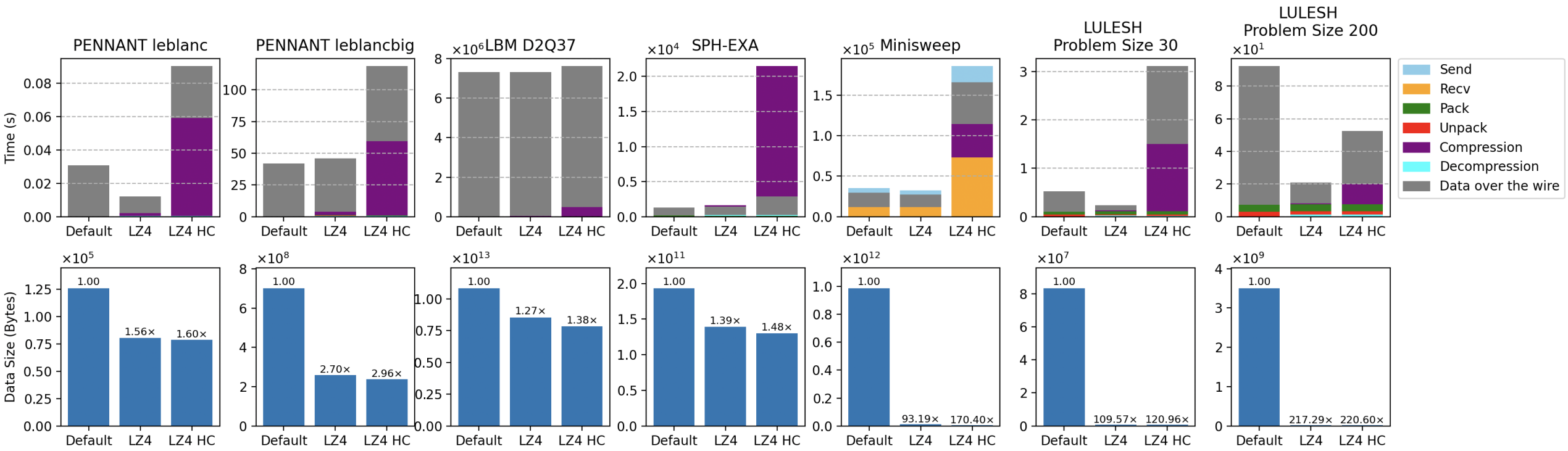


Benchmarks and Applications

Benchmark	Description
PENNANT	A mini-app from CORAL-2 modeling unstructured mesh physics, focusing on Lagrangian and radiation hydrodynamics. It serves as a compact proxy for large-scale multi-physics codes.
LBM	Simulates fluid dynamics using the Lattice Boltzmann Method. It models mesoscopic flow behavior and emphasizes parallel scalability and memory bandwidth.
SPH-EXA	Implements the Smoothed Particle Hydrodynamics method for fluid and solid dynamics. Designed to explore portability and scalability across HPC platforms.
Minisweep	Models sweep-based transport used in neutron and radiative transfer simulations. Captures communication-heavy patterns with directional dependencies.
LULESH	A proxy for shock hydrodynamics on unstructured meshes, modeling core compute patterns of typical hydrodynamics codes for performance benchmarking.

Benchmark	Test Command
PENNANT leblanc	mpirun -np 32 –bind-to hwthread –map-by hwthread ./build/pennant ./test/leblanc/leblanc.pnt
PENNANT leblancbig	mpirun -np 32 –bind-to hwthread –map-by hwthread ./build/pennant ./test/leblancbig/leblancbig.pnt
LBM	runhpc –config=config.cfg –action=ref 505.lbm_t
SPH-EXA	runhpc –config=config.cfg –action=ref 532.sph_exa_t
Minisweep	runhpc –config=config.cfg –action=ref 521.miniswp_t
LULESH	mpirun -np 8 –bind-to hwthread –map-by hwthread ./lulesh2.0 -s ${Problem Size} -i 50

Benchmarks and Applications Initial Performance with On-the-fly Compression



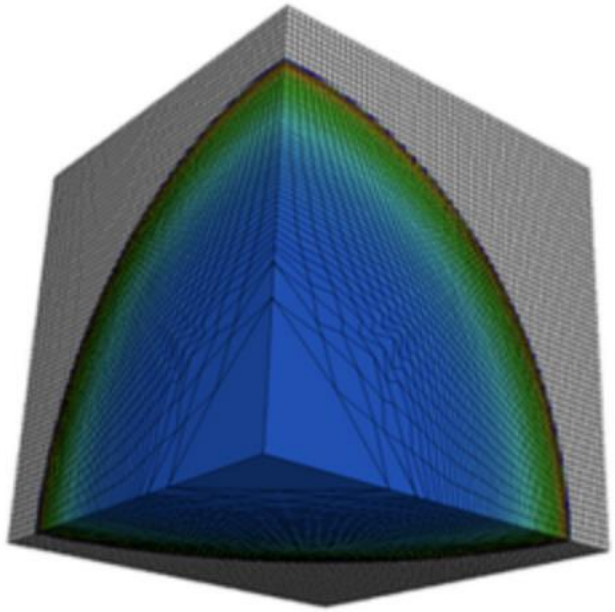
- On-the-fly LZ4 and LZ4 HC compression applied to the partial or whole blocking point-to-point communication
- Time measured to the end of communication for partially non-blocking communication or measured separately (send and receive) if it is blocking

Application: LULESH

- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics *https://asc.llnl.gov/codes/proxy-apps/lulesh*
- LULESH approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collective of volumetric elements defined by a mesh

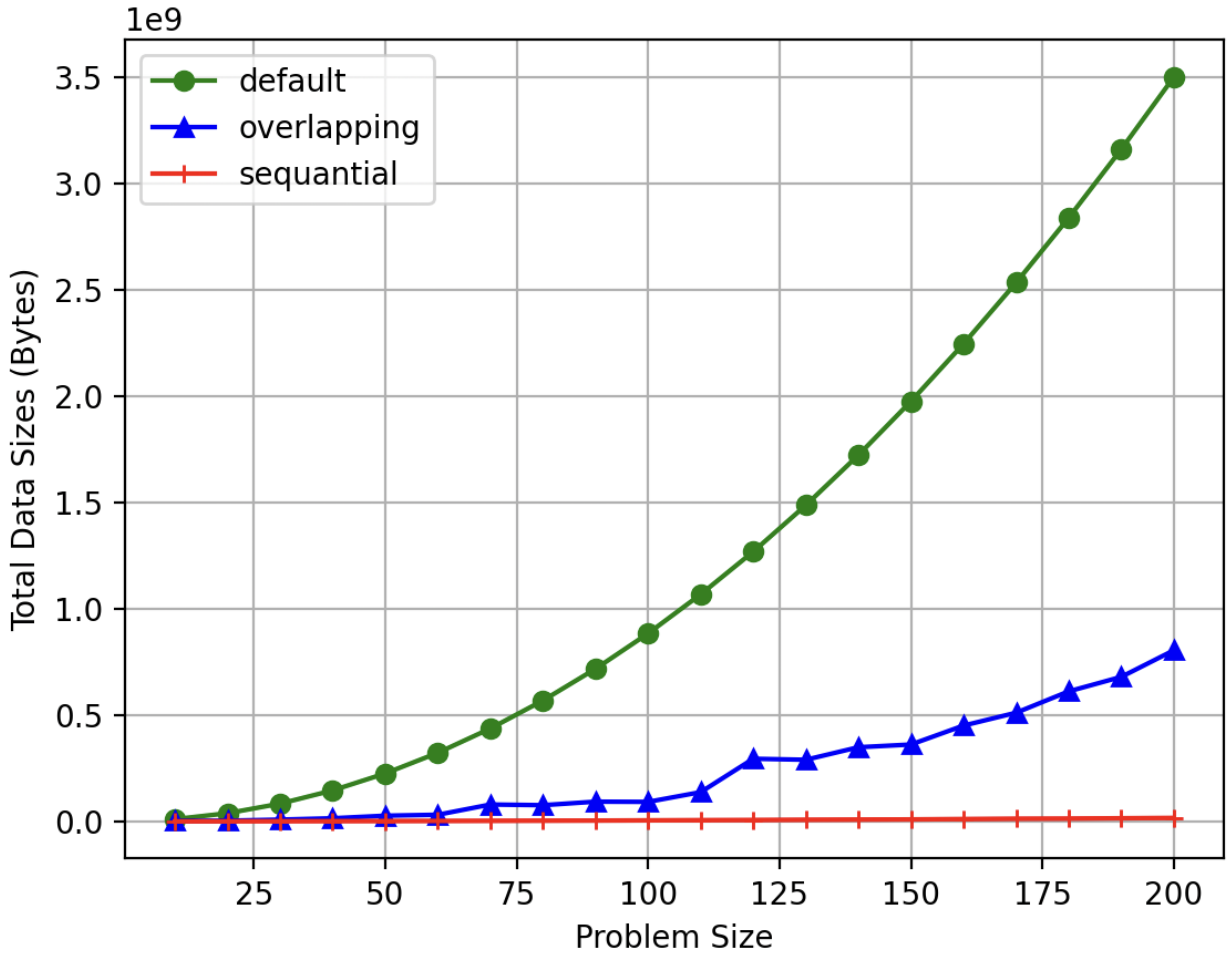
What we really cared about

- LULESH is MPI-enabled
- LULESH uses MPI communication (point-to-point)
- LULESH packs and communicates contiguous memory region
- LULESH reuses buffer for communicaton across iterations
- LULESH must be run with n^3 processes
- Varies problem size through command line

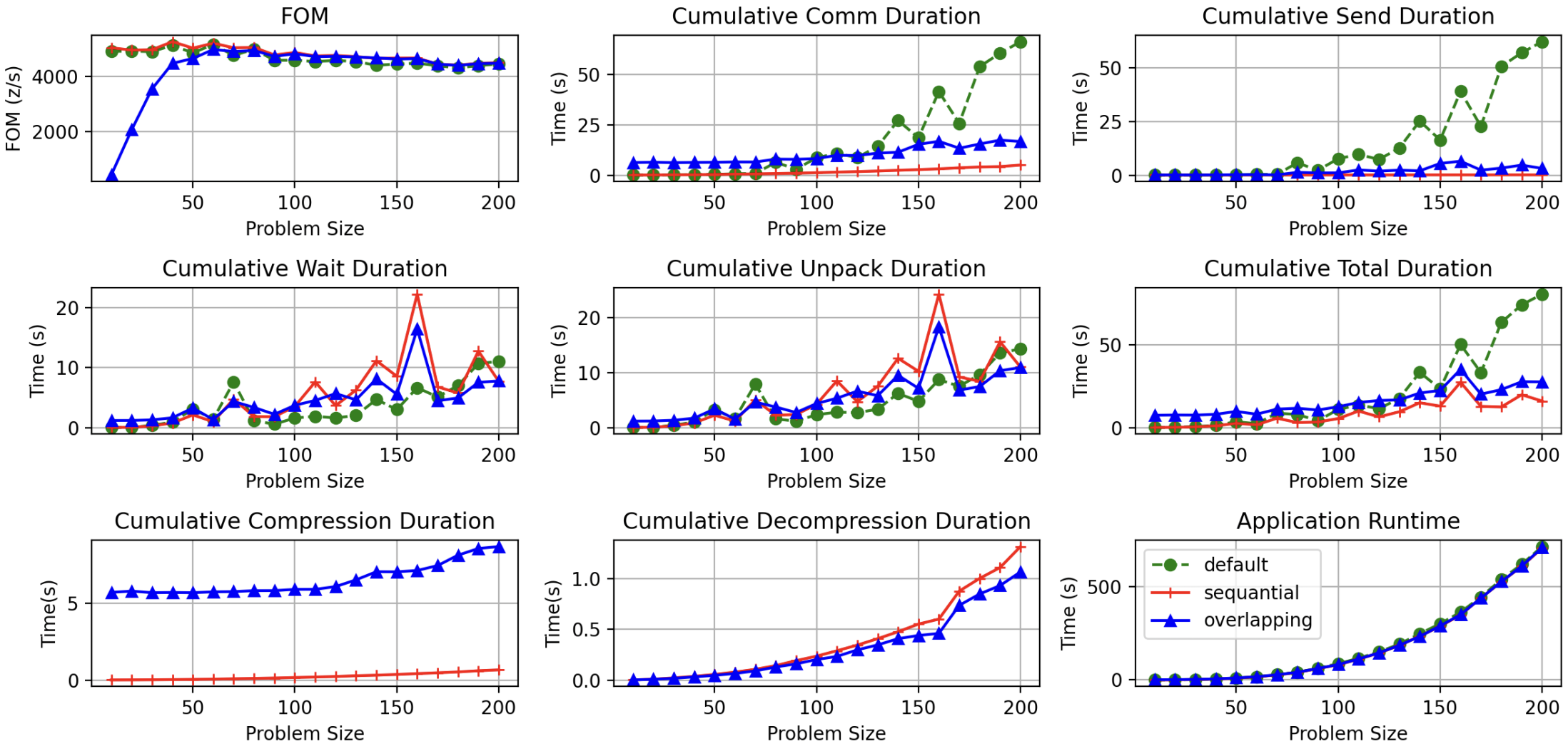


Message Sizes Exchanged in LULESH

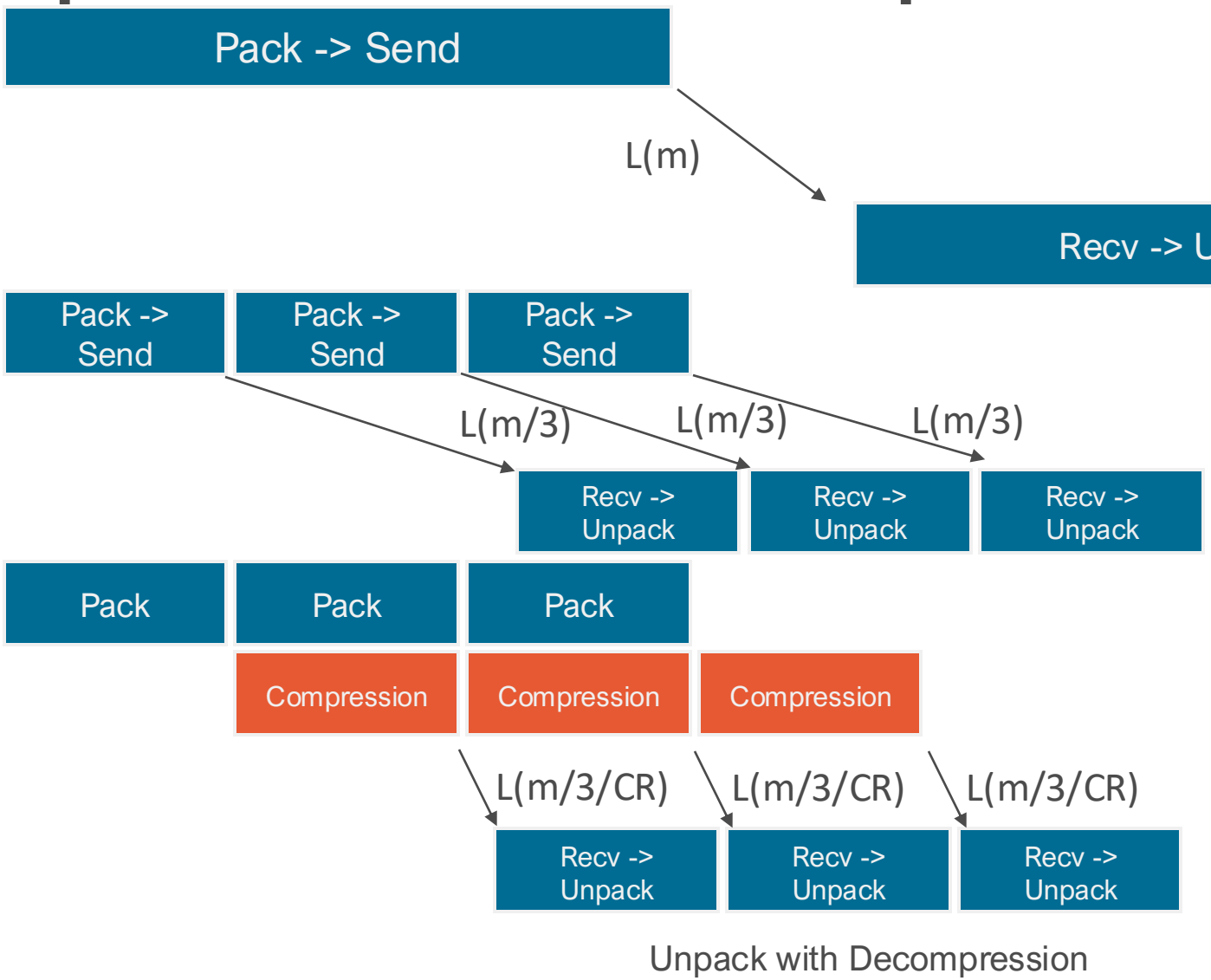
- Varied problem size in LULESH
- Iteration count for each run kept at 50
- LULESH's data has a extremely high chance of long sequence of identical data, thus, high compression ratio
- uffd version has hash collision, causing a few buffers not to be compressed for all iterations



LULESH Performance



Optimal Scenario for Compression Hiding (CH) Framework



- Choice of compress algorithm could depend on the overhead of either pack or network latency or network speed
- Cost of compression will be hide behind either pack or communication

CR=Compression Ratio

Compression Hiding (CH) Framwork Conclusions and Future Work

- Compression overhead may mitigate the benefit received from reduced message size
  - Introduced CH framework
  - Has the potential to hide compression overhead by using free computing resources
- Current framework does not compress every buffer
  - Need better hash strategy to avoid collision in compression queue
- One write invalidates every buffer in the registration
  - uffd registration merging -> fine-grained uffd registration (page based)
- Find balance among compression algorithm overhead, compression ratio, pack overhead and communication overhead
- Challenge to find right application with characteristics
- Or modify the current communication pattern

Questions?