TEMPI: An Interposed MPI Library with a Canonical Representation of CUDA-aware Datatypes

Carl Pearson[1], Kun Wu[2], I-Hsin Chung[3], Jinjun Xiong[3], Wen-Mei Hwu[4]
[1]Sandia National Labs / [2]University of Illinois Electrical and Computer Engineering / [3]IBM T. J. Watson Research / [4]Nvidia Research

Introduction

TEMPI provides a transparent non-contiguous data-handling layer compatible with various MPIs.

MPI Datatypes are a powerful abstraction for allowing an MPI implementation to operate on non-contiguous data. CUDA-aware MPI implementations must also manage transfer of such data between the host system and GPU.

The non-unique and recursive nature of MPI datatypes mean that providing fast GPU handling is a challenge. The same non-contiguous pattern may be described in a variety of ways, all of which should be treated equivalently by an implementation. This work introduces a novel technique to do this for strided datatypes.

Methods for transferring non-contiguous data between the CPU and GPU depends on the properties of the data layout. This work shows that a simple performance model can accurately select the fastest method.

Unfortunately, the combination of MPI software and system hardware available may not provide sufficient performance. The contributions of this work are deployed on OLCF Summit through an interposer library which does not require privileged access to the system to use.

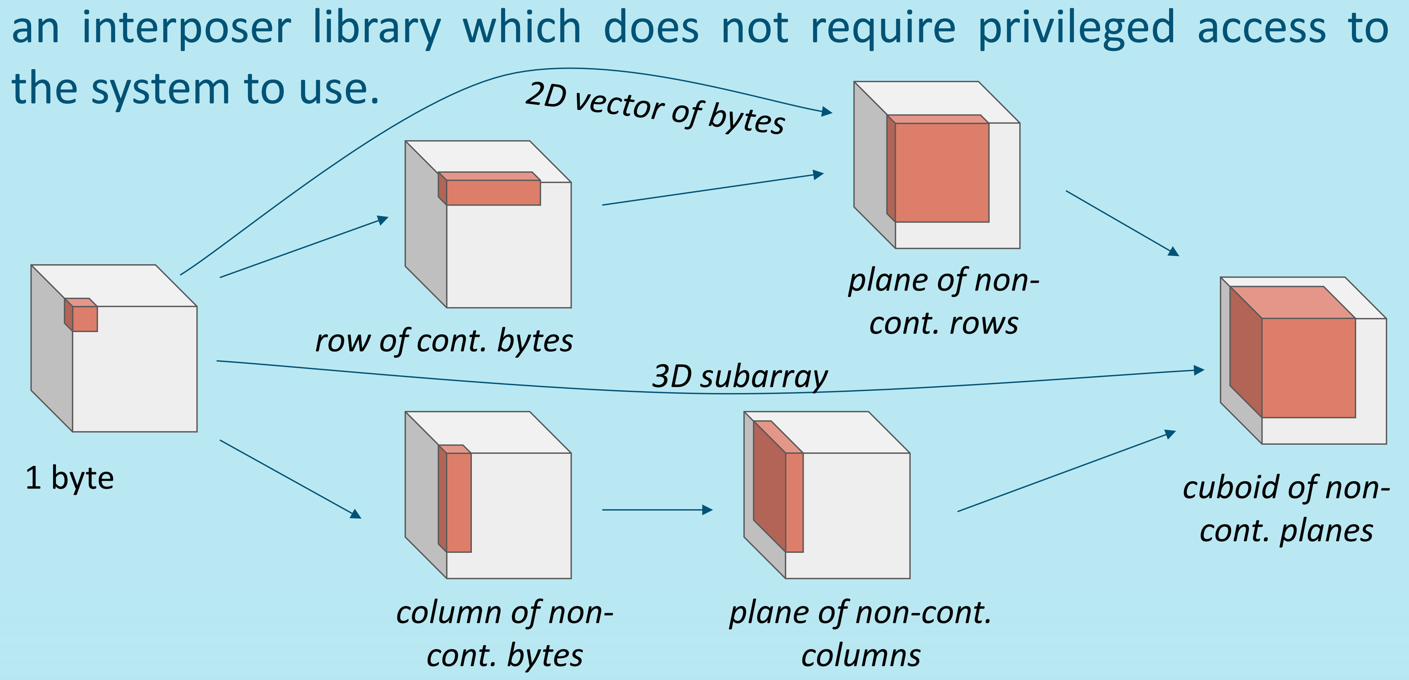

Figure 1: Multiple MPI datatype descriptions to arrive at a common non-contiguous region.

Translation and Canonicalization

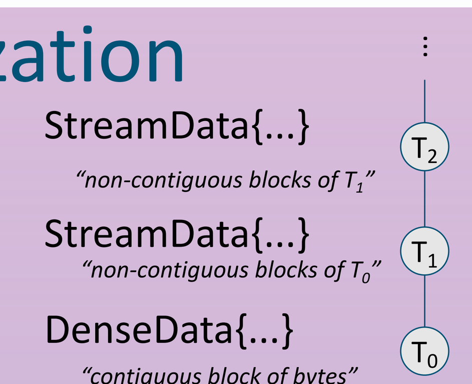

Figure 2: TEMPI's internal representation of the strided MPI datatype. A hierarchy of StreamData objects, each representing repeated copies of their children. The base of the IR is a DenseData representing a contiguous block of bytes.

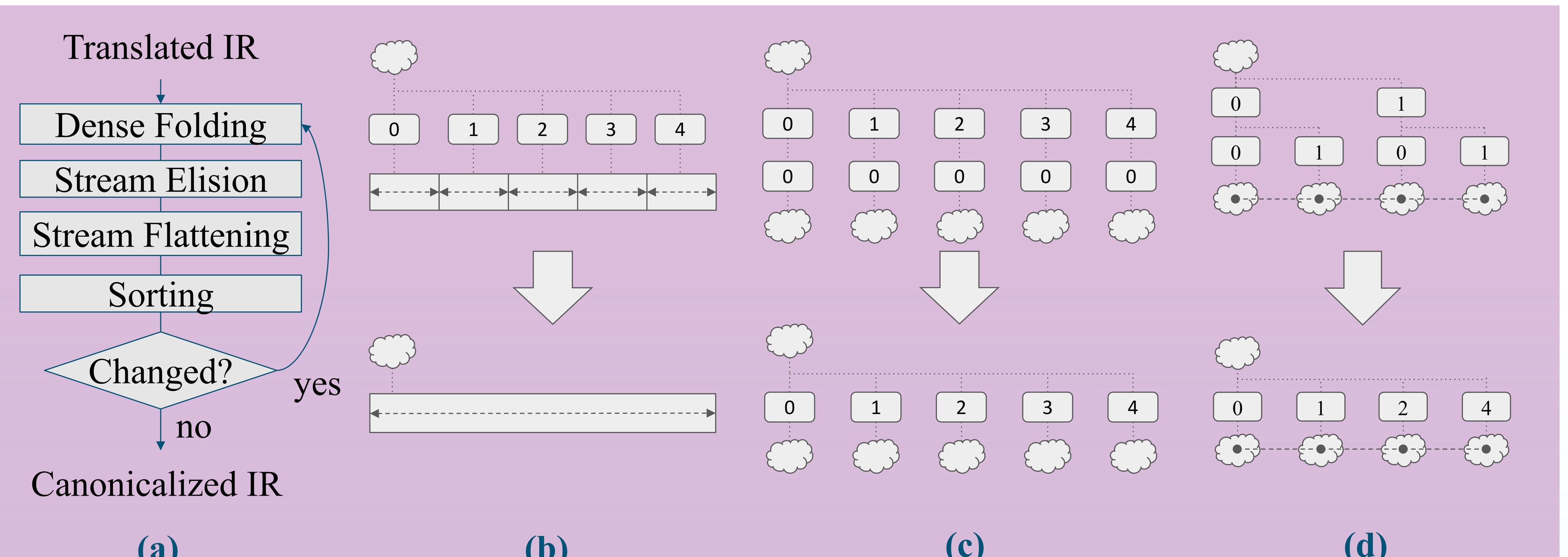

Figure 3: (a) The IR is canonicalized through repeated applications of four transformations. (b) Dense folding, where contiguous DenseData are merged into a larger DenseData, (c) stream elision, where single-element StreamData are removed, and (d) stream flattening transformation, where a hierarchy of StreamData is transformed into a single equivalent StreamData. Sorting (not shown) canonicalizes the order of the hierarchy.

When MPI_Type_commit is called, the MPI datatype is translated into an internal representation, a hierarchy of StreamData objects representing strided repetitions of their child elements (Fig. 2). Since equivalent MPI datatypes will yield different IR, the IR is canonicalized through a series of transformations. A data-packing kernel is selected based on the canonicalized IR. This kernel will be used to pack non-contiguous data in future MPI communication operations before providing the packed data to the underlying system MPI.

Automatic Data Transfer Selection

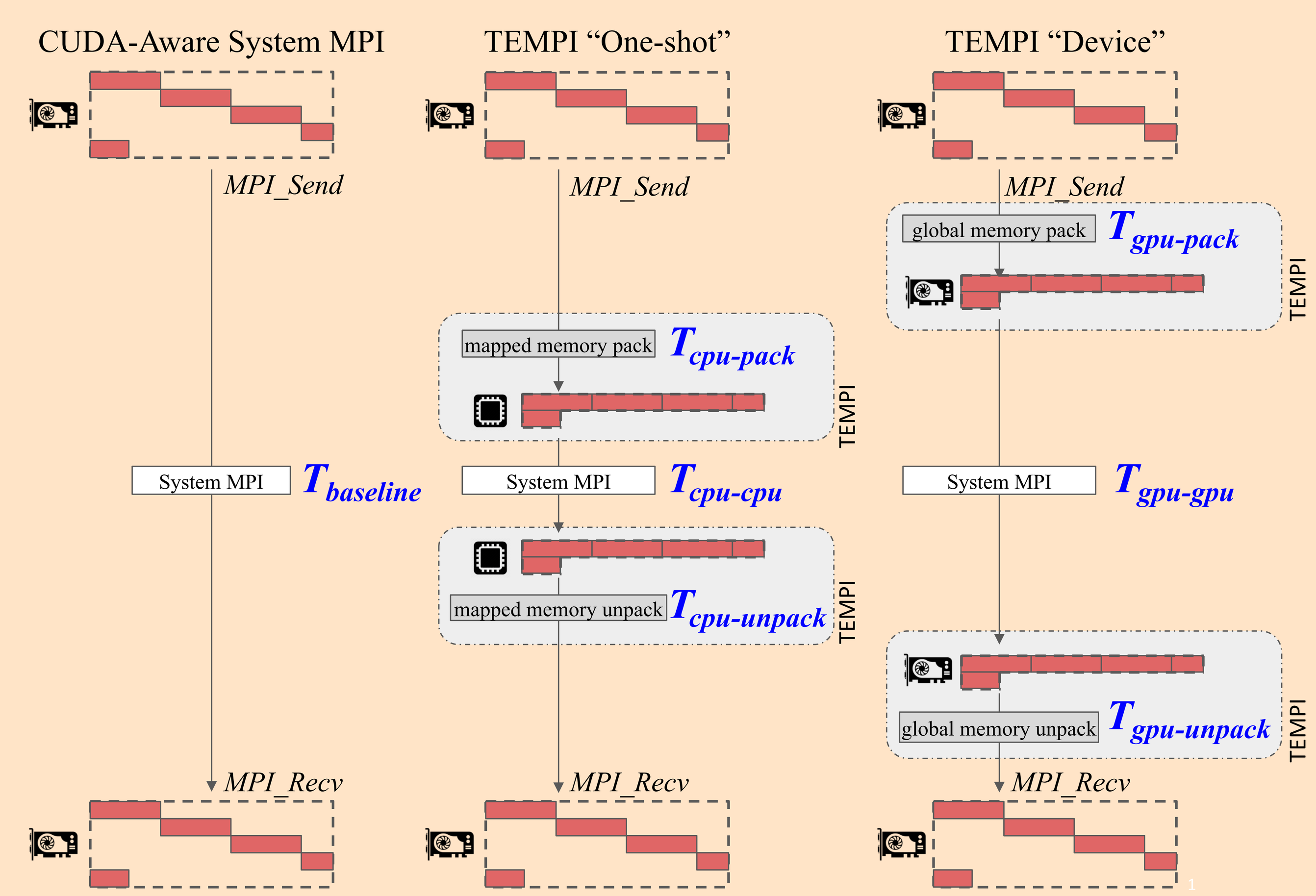

Figure 4: Diagram of the staged, one-shot, and device transfer methods annotated with contributions to the performance model.

Orthogonal to the datatype canonicalization is the strategy for moving the non-contiguous (unpacked) GPU data to the MPI implementation. In all cases, a GPU kernel is responsible for packing the data, but the packed buffer provided to MPI is different

The method used for each MPI communication is dynamically selected by evaluating the performance model (Fig. 5) The model parameters are measured on the target platform ahead of time (Fig. 6). The model is capable of selecting the correct implementation with negligible overhead (Fig. 7

$$T_{device} = T_{gpu-pack} + T_{gpu-gpu} + T_{gpu-unpack}$$
$$T_{oneshot} = T_{host-pack} + T_{cpu-cpu} + T_{host-unpack}$$
$$T_{staged} = T_{gpu-pack} + T_{d2h} + T_{cpu-cpu} + T_{h2d} + T_{gpu-unpack}$$

Figure 5: Performance model used to select the communication method at runtime.

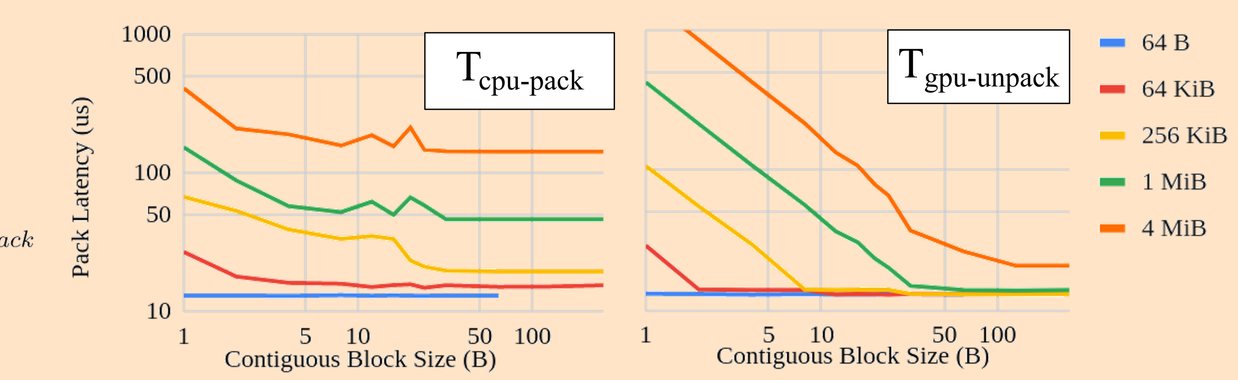

Figure 6: Example measured parameters used for runtime dynamic communication method selection.

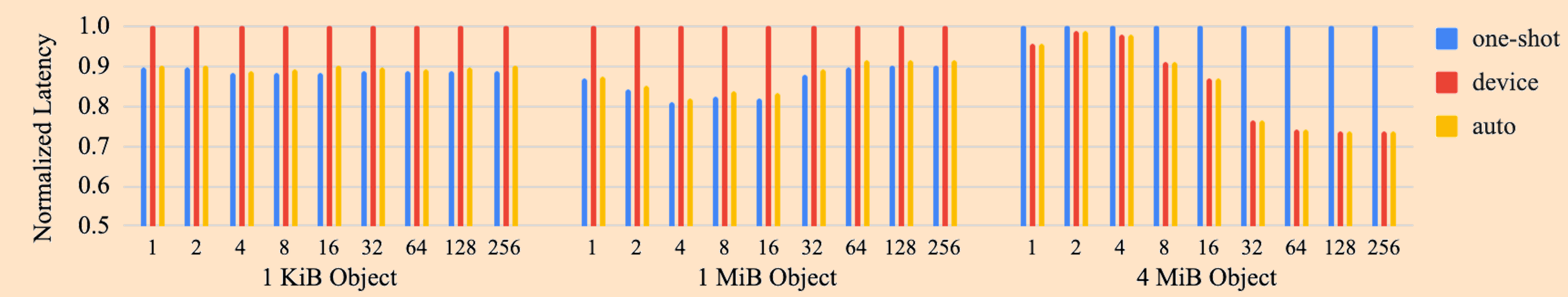

Figure 7: Elapsed MPI_Send time for the one-shot and device methods for a variety of objects and contiguous block sizes. Automatic selection (auto) always chooses the fastest method.

Halo Exchange Results

A 3D stencil halo exchange (double precision, eight quantities, radius 3) accelerated by ~1000x on Summit at 3072 ranks (512 nodes, 6 GPUs per node). The TEMPI library is loaded through the LD_PRELOAD mechanism over the system MPI implementation code.

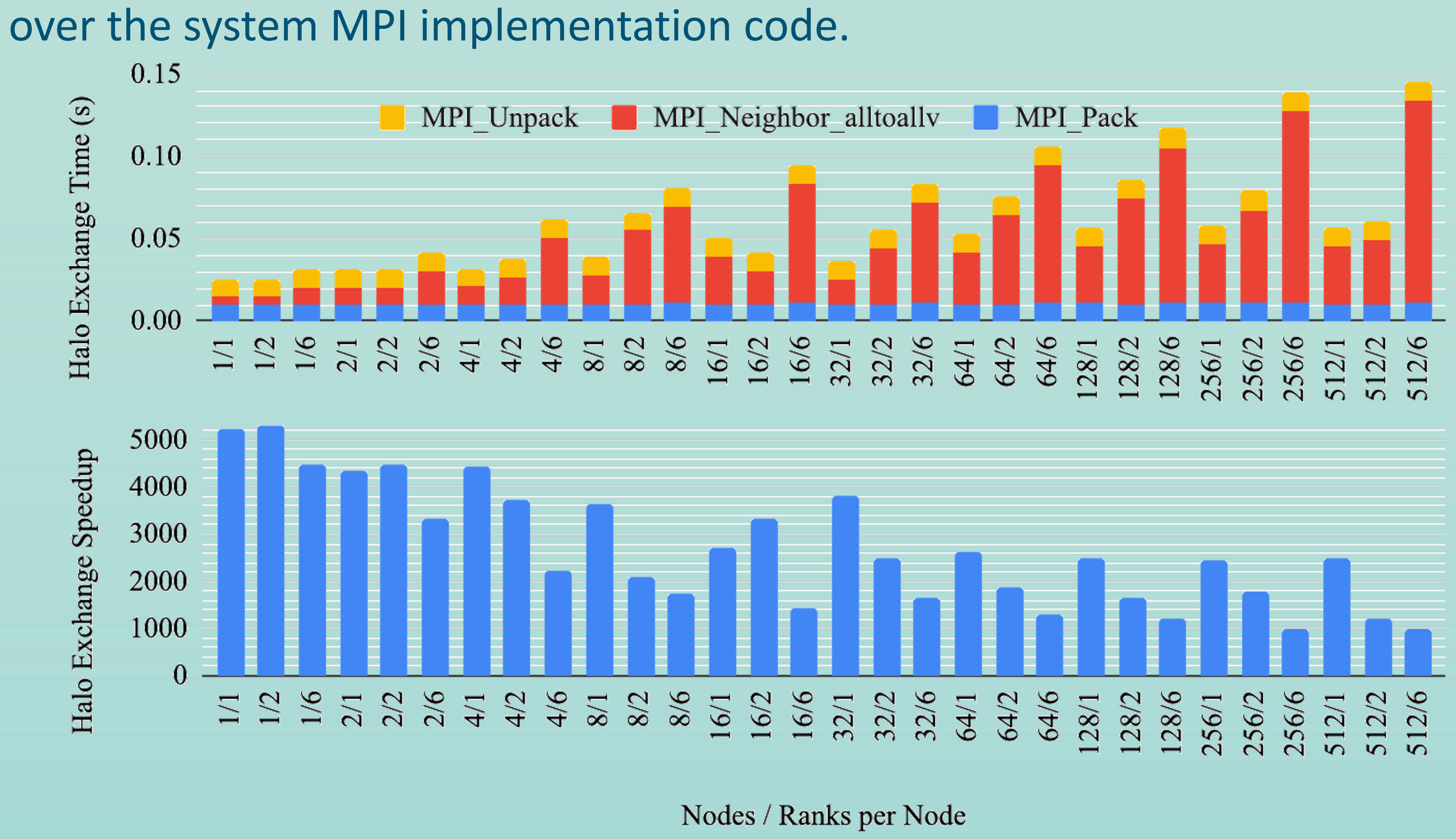

Figure 8: Halo exchange time and speedup on OLCF Summit.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology &Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND No. SAND2021-7110 C

This work is supported by IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizon Network. 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 Contract No. DE-AC05-00OR22725. This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant #1725729,as well as the University of Illinois at Urbana-Champaign

github.com/cwpearson/tempi

Work completed at University of Illinois prior to joining Sandia National Labs

center for cognitive computing systems research

Sandia National Laboratories