Lightweight Communication Interface:

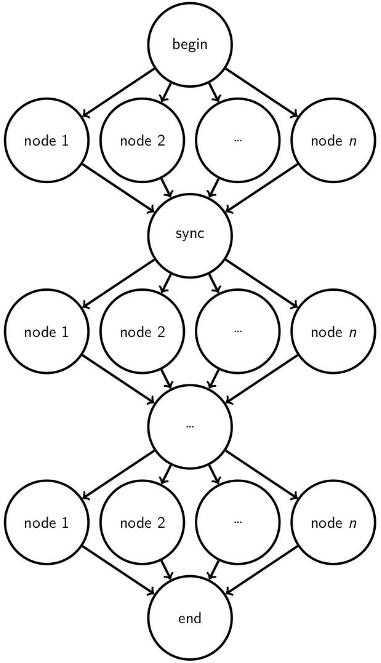
Efficient Message Passing Support for Irregular, Multithreaded Communication.

Jiakun Yan, Omri Mor, Hoang-Vu Dang, Marc Snir

University of Illinois Urbana-Champaign

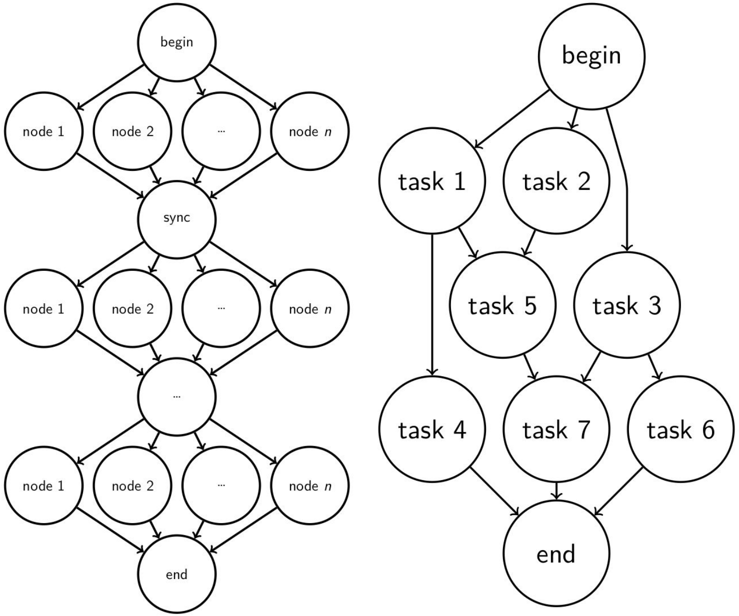
Traditional Parallel Programming

- Dominated by Message Passing Interface (MPI).
- Typically, a MPI application uses…
  - Bulk-Synchronous Programming (BSP).
    - Global synchronous steps.
    - Local computation -> communication -> local computation…
  - MPI everywhere.
    - One MPI process per CPU core.
    - MPI is known for its poor multithreaded performance.



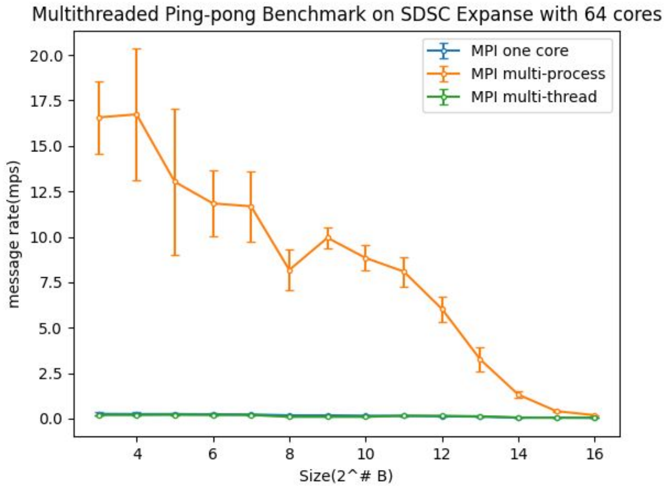
New Architecture calls for New Programming Model

- Modern parallel architecture
  - Increased intra-node parallelism.
  - Increased heterogeneity.
  - Powerful interconnect.
- Task-based programming model
  - Programmers decompose their program into **tasks** along with their **dependencies**.
  - The runtime will handle the mapping, scheduling and data movement.
  - E.g. HPX, Legion, PaRSEC.
  - Communication layer: MPI/GASNet.



New Communication Pattern

- New parallel architecture + new programming model -> new communication pattern.
  - Multithreaded.
  - Irregular destinations.
  - Small messages.
- Traditional communication libraries are not efficient enough.



Lightweight Communication Interface (LCI)

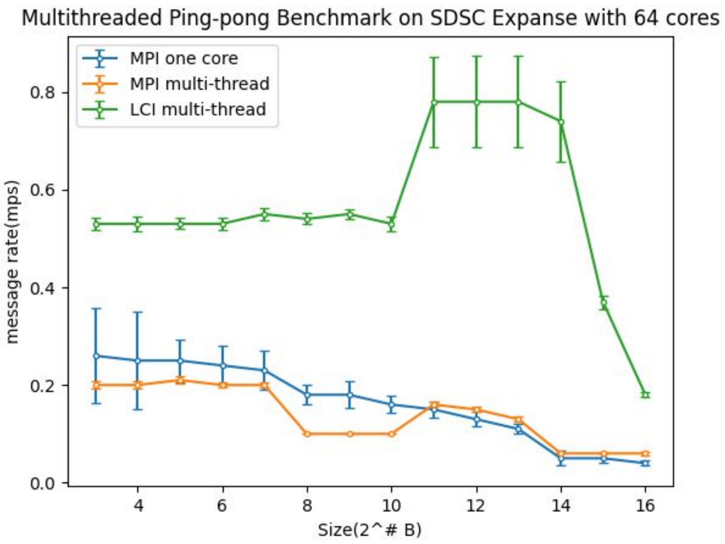
- Designed with task-based runtime as the target clients.
  - Should also apply to other irregular applications such as graph analysis/sparse linear algebra.
- A low-level communication library.
  - Intended user: high-level library developers.
  - Like UCX/Libfabric/GASNet, as opposed to MPI.
- Major features:
  - Flexible communication primitives and signaling mechanisms.
  - Better multithreaded performance.
  - Explicit user control of communication behaviors and resources.

Flexible communication primitives and signaling mechanisms

- How much data to send
  - Short (inline), medium (eager), long (zero-copy)
  - iovec (one medium + multiple long)
- Who provides the source/target buffers
  - Two-sided send/recv
  - One-sided put/get
- What signaling mechanisms to use
  - Synchronizer (MPI_Request)
  - Completion queue
  - Active message handler
  - No signaling
- Whether the source/target buffers are user-provided or LCI-provided
  - Using LCI-provided buffers can potentially save one memory copy
- For long messages, whether the source/target buffers are registered.
- How to match the send and recv
  - tag only/rank+tag

Better multithreaded performance

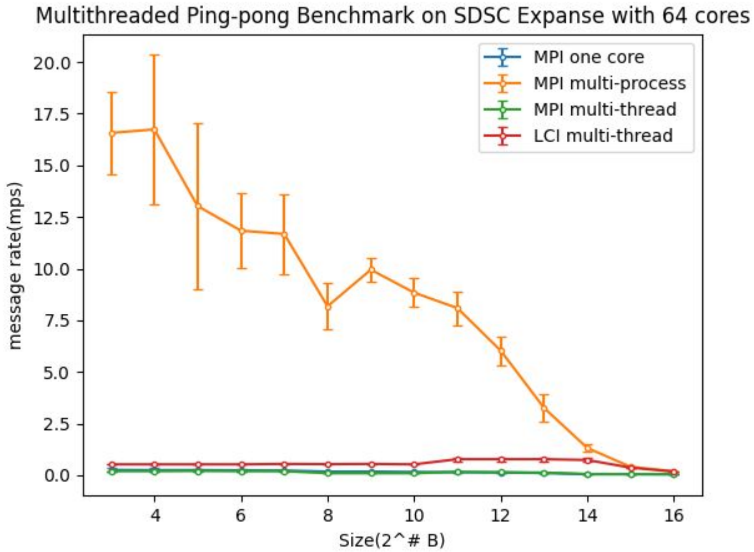
- At the LCI level[1],
  - No coarse-grained mutex locks.
  - Replace the centralized MPI matching queue with hashtable.
    - Give up the MPI ordering semantics.
      - Message received can be out-of-order.
    - Give up MPI_ANY_SOURCE and MPI_ANY_TAG.
      - But you can achieve almost the same thing with tag-only matching.



[1] Hoang-Vu Dang, Marc Snir, William Gropp: Towards millions of communicating threads. EuroMPI 2016: 1-14

Better multithreaded performance

- At the LCI level,
  - No coarse-grained mutex locks.
  - Replace the centralized MPI matching queue with hashtable.
- At the lower level, allocate multiple hardware contexts
  - ibverbs queue pairs, libfabric endpoints, UCX workers…
  - But unlike MPI endpoint proposal, we can still maintain one rank per node.
  - Working in progress.



Explicit user control of communication behaviors and resources

- Give users as much control as possible:
    - How many resources to allocate: matching table number and size, completion queue number and size…
    - Which threads sharing which resources (hardware context, matching table, completion queue…)
- Always propagate back pressure to users:
    - MPI send always succeeds, but LCI send can return LCI_ERR_RETRY.
- LCI_progress(): explicit making progress on background works.
    - Unlike MPI, while background progressing happens as a side-effect of MPI function calls.
    - The recommended way is to use one (or more) dedicated progressing threads.
    - LCI_progress() on specific hardware context and send/recv.

LCI: Current Status

- Actively evolving.
- Current backends:
  - libibverbs (for Infiniband)
  - libfabric (for Cray interconnect)
- Existing clients and collaborators:
  - Gluon, D-Galois, D-Ligra[1]: graph analytics.
  - PaRSEC[2] (working in progress): task-based programming model with explicit task dependency graph
  - HPX[3] (working in progress): task-based programming model with implicit task dependency graph

[1] Dathathri, Roshan, et al. "Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics." *Proceedings of the 39th ACM SIGPLAN conference on programming language design and implementation*. 2018.
[2] Bosilca, George, et al. "Parsec: Exploiting heterogeneity to enhance scalability." *Computing in Science & Engineering* 15.6 (2013): 36-45.
[3] Kaiser, Hartmut, et al. "Hpx: A task based programming model in a global address space." *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. 2014.

HPX+LCI: Current Status

- HPX: task-based programming model with implicit task dependency graph.
    - Users invoke tasks like sequential code, and the runtime analyzes data usage and builds task dependency graph.
- Its communication layer implements a “parcelport” interface.
    - A parcel consists of a small buffer (control data + small arguments) and optionally a few large zero-copy buffers.
    - Current implementations: MPI, libfabric (working in progress)
- We are developing a LCI parcelport for HPX.
    - Use one-sided put, iovec, completion queue, dedicated progress thread.
        - We can transfer a parcel with only one LCI function call.
    - Minimum number of messages and memory copies.
    - No mutex lock.
    - Will use multiple hardware contexts.
- Working on performance evaluation.

Relevant Publications:

1. Hoang-Vu Dang, Marc Snir, William Gropp: Towards millions of communicating threads. EuroMPI 2016: 1-14
2. Hoang-Vu Dang, Roshan Dathathri, Gurbinder Gill, Alex Brooks, Nikoli Dryden, Andrew Lenharth, Loc Hoang, Keshav Pingali, Marc Snir: A Lightweight Communication Runtime for Distributed Graph Analytics. IPDPS 2018: 980-989
3. Hoang-Vu Dang, Marc Snir: FULT: Fast User-Level Thread Scheduling Using Bit-Vectors. ICPP 2018: 71:1-71:10
4. Roshan Dathathri, Gurbinder Gill, Loc Hoang, Vishwesh Jatala, Keshav Pingali, V. Krishna Nandivada, Hoang-Vu Dang, Marc Snir: Gluon-Async: A Bulk-Asynchronous System for Distributed and Heterogeneous Graph Analytics. PACT 2019: 15-28

Lightweight Communication Interface:

Efficient Message Passing support for irregular, multithreaded communication.

Github URL: https://github.com/uiuc-hpc/LC/tree/dev-v1.7

Q&A: Jiakun Yan (jiakuny3@illinois.edu)