IExchange:
Asynchronous Communication and Termination Detection
for Iterative Algorithms

Dmitriy Morozov
Lawrence Berkeley National Laboratory

Tom Peterka
Argonne National Laboratory

Hanqi Guo
Argonne National Laboratory

Mukund Raj
formerly Argonne National Laboratory

Jiayi Xu
The Ohio State University

Han-Wei Shen
The Ohio State University
Objective

- Asynchronous communication protocol for exchanging MPI messages in distributed iterative algorithms
- Efficient interleaving of communication with computation
- Asynchronous global termination interleaved with computation
- No assumptions on global work

Comparison of synchronous exchange (left) and asynchronous iexchange (right) Gantt charts show that the asynchronous method is several times faster because communication and computation are interleaved.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel particle tracing</td>
<td>Muller et al. 2013</td>
</tr>
<tr>
<td>Distributed union-find</td>
<td>Xu et al. 2021</td>
</tr>
<tr>
<td>Parallel Delaunay tessellation</td>
<td>Morozov et al. 2016</td>
</tr>
<tr>
<td>Distributed graph processing</td>
<td>Dathathri et al. 2019</td>
</tr>
<tr>
<td>Parallel merge tree</td>
<td>Nigmetov et al. 2019</td>
</tr>
</tbody>
</table>
### Parallel Programming Models

#### Bulk-Synchronous Parallel (BSP)
- **MPI**
- **DIY** (prior to this paper) [Morozov et al. 2016]

**Pros**
- Familiar, easy to understand
- Good for compute-intensive, load-balanced applications

**Cons**
- Excessive synchronization

#### Asynchronous Many-Task (AMT)
- **Charm++** [Kale et al. 1993]
- **Legion, Regent** [Bauer et al. 2012, Slaughter et al. 2015]

**Pros**
- Adapts to varying load balance

**Cons**
- New and unfamiliar
- Varying performance

### Relaxed BSP or Bulk Asynchronous Parallel (BASP)
- **DIY** (in this paper)
- **Gluon-Async** [Dathathri et al. 2019]

**Pros**
- Familiar, easy to program and understand
- Performs well for both regular and irregular algorithms

**Cons**
- Slightly higher overhead at small scale

---

**Data-parallel**
- **Strict BSP**
  - e.g., MPI

**Block-parallel**
- **Relaxed BSP**
  - DIY

**Task-parallel**
- **Asynchronous**
  - e.g., Legion
DIY Synchronous Exchange and Asynchronous IExchange Protocols

**Synchronous Exchange**

while (not globally done)
{
  user dequeues incoming messages
  user performs local work
  user enqueues outgoing messages
  user exchanges messages synchronously
  user determines global work (globally synchronous)
}

**Asynchronous IExchange**

while (not globally done)
{
  user dequeues incoming messages
  user performs local work
  user enqueues outgoing messages
  DIY exchanges messages asynchronously
  DIY determines global work asynchronously
}
Termination Detection (Quiescence)

Definition
All blocks (processes, workers) locally are done, and no work (messages) remains in the system.

No Assumptions on Global Amount of Work
We do not need to know global work in advance, and this can grow or shrink. Termination is cancellable.

Prior Approaches
• MPI one-sided global counter (not scalable)
• Scalable global counter [Sinha et al. 1993] (workers signal when they will not add more work, monotonic decreasing afterwards)
• Active Pebbles [Willcock et al. 2011] (limits on amount of new work)
• Gluon-Async [Dathathri et al. 2019] (only for graph analytics)
Conceptually: 3 Traversals of a Message Tree (Down-Up-Down)

After each iteration:
All blocks (processes, workers) send their local state up the tree.

When the aggregate state at the root is done:

Step 1:
Root sends all done message down the tree.

Step 2:
Blocks send either confirmation of being done or a cancel message up the tree.

Step 3:
Root sends terminate message down the tree.

Note: cancel message returns state back to normal processing of iterations
Implementation: 3-State Diagram with MPI Nonblocking Collectives

- Note: MPI_Isend used to send all messages guarantees that termination is not reached prematurely.
  - MPI_Isend: nonblocking but guaranteed that a matching receive has been posted on the receiving side before request tests true on the sending side
- Always returns to working after changing states (interleaving computation with termination detection).

key point: the events are ordered, so at no point both receiver and sender think they are done
while not globally done {
local particles ← dequeue incoming particles
for each local particle p {
while p ∈ local block bounds
  advect p with Runge-Kutta scheme
if p ∈ global domain bounds
  enqueue p to neighboring block
else
  retire p
}
}
Synthetic flow test generates load imbalance by modifying the velocity magnitude in the blocks that are positioned on the 3-d diagonal of the lattice of all the blocks.

Time to compute particle advection over a synthetic vector field of $512^3$ vectors that are unbalanced by a factor of 10:1. Particles are seeded at a rate of one per every 0.25 grid points in y and z dimensions in leftmost plane of the domain.
Streamlines traced in Nek5000 dataset indicate a high degree of turbulence in the flow field.

Time to compute particle advection of over Nek5000 dataset of 5123 vectors. Particles are seeded at a rate of one per every 4 grid points in each dimension.
while not globally done
{
    dequeue incoming vertices and labels of sets
    perform local serial union-find
    for each edge \((u,v)\)
    {
        if \(\text{process}(u) \neq \text{process}(v)\)
        {
            if set\((u)\) changed
                enqueue \([u, \text{set}(u)]\) to \(\text{process}(v)\)
            if set\((v)\) changed
                enqueue \([v, \text{set}(v)]\) to \(\text{process}(u)\)
        }
    }
}
Synthetic dataset consists of a regular grid of two spatial dimensions and 1 temporal dimension (3-d in total) with a variable number of critical points whose trajectories move in the space dimensions as a function of time. The trajectories of the critical points in space-time are the connected components (disjoint sets) in union-find.

Time to compute distributed union-find of connected components of critical points in a synthetic field of 1282 scalars over 128 time steps.
Microparticle tracking in exploding-wire experiments is one application of the distributed union-find algorithm.

Time to compute distributed union-find of connected components of critical points in high-speed images of 3,842 pixels over 4,745 time steps.
Conclusion

Summary
• Asynchronous communication protocol for exchanging MPI messages in iterative algorithms
• Efficiently interleaving of communication with computation
• Asynchronous global termination interleaved with computation
• No assumptions that global work decreases monotonically, allowing new work to be created

Advantages
• Ease of use
• Performance
• General applicability

Limitations
• Lack of strict synchronization must not alter final result
• Some minor performance overhead at small scale

Future Work
• Apply to other iterative algorithms, eg. merge trees and graphs
• Compare with other asynchronous programming models
Acknowledgments

Facilities
Argonne Leadership Computing Facility (ALCF)
National Energy Research Scientific Computing Center (NERSC)

Funding
DOE ASCR Research Programs
DOE SciDAC Programs