Goldilock Principle is Wrong

Marc Snir
MPI

• Old: Started 25 years ago
• Was designed as a low-level application programming interface
  – “MPI is more like an assembly language providing concepts about moving data between entities.”
• Has kept evolving 1 -2 -3
• Has been used both as an API and as an implementation layer for libraries and frameworks
Problem 1

- **MPI is large.**
  - MPI 3.1 standard is 836 pages long, (C++ 14 is 1354 pages long)
  - MPI software is ~1 MLOC
- Hard to continue evolution
  - Especially for features with global impact, such as fault tolerance
- Hard to implement all features efficiently
  - Plenty of features that are not supported well and are seldom used
    - Vicious cycle
Problem 2

- MPI has been (rightly) attacked for being too low-level (as an API)
  - But no good replacement has emerged...
- MPI should be attacked as being too high-level (as an implementation layer)

- *Time to bifurcate:*

Low-level communication interface
LLCI

- Can be much more efficient than full MPI
- Can be evolved more easily
- Can provide better semantic match to
  - the needs of various applications
  - the capabilities of modern NICs
Simple Example – One instruction Put

• Data is moved from contiguous send buffer to contiguous receive buffer
  – Same communication pattern reused
  – No synchronization involved
• descriptor = put_init(sendbuf, length, dest, recvbuf);
  – 10’s of instructions
• put(descriptor) -- one store to NIC!
  – Ideal match to a modern NIC
Match to Application

• Graph analytics library: fixed communication pattern but variable amount of data
  – Current implementation atop MPI
    • iprobe – find how much data sent
    • malloc
    • recv
  – Both application and MPI poll
  – Data is copied twice and memory is allocated twice
Let’s Deconstruct (Point-to-Point) Communication

• Send/put: Copies data from send buffer to receive buffer
  1. How are the two buffers matched?
  2. What synchronization marks communication completion?
Buffer Matching

• Sender knows where data goes
  – Most frequent case and easiest to support in HW
• Application does not care where data goes (Use queue)
  – Very useful for data analytics; expensive to “emulate” atop MPI
• Send and receive buffer are matched at each communication (send-receive)
  – Most complex and seldom needed
  – Made even more complex because of order requirements and don’t cares
Synchronization

- Not needed for each communication: Use fence construct
- Signal when communication completes
  - Most general: invoke method on sync object (active message)
  - Special cases: sync object is flag, counter,...
    - Should be supported by NIC! (T3D)
    - Sync objects should be understood by thread/task scheduler!
Performance of MPI—Simplified Matching & Integration with Scheduler

Latency 8 threads

N tasks, 14 cores
Graph Analytics MPI vs. LLCI

- Use queue for incoming data
Don’t develop “MPI NICs”

• Co-design:
  – What are the basic communication primitives needed to support algorithmic communication patterns?

• Low-level Communication Interface:
  – Simple interface that exposes these primitives

• API:
  – MPI, Legion... built atop LLCI