

Lessons from MPI for Petaflops

William Gropp

Argonne National Laboratory

<http://www.mcs.anl.gov/~gropp>



Overview

- MPI as a demanding application
- Implementing MPI
 - Progress
 - Collective Operations
 - Performance

MPI as an Application: Background

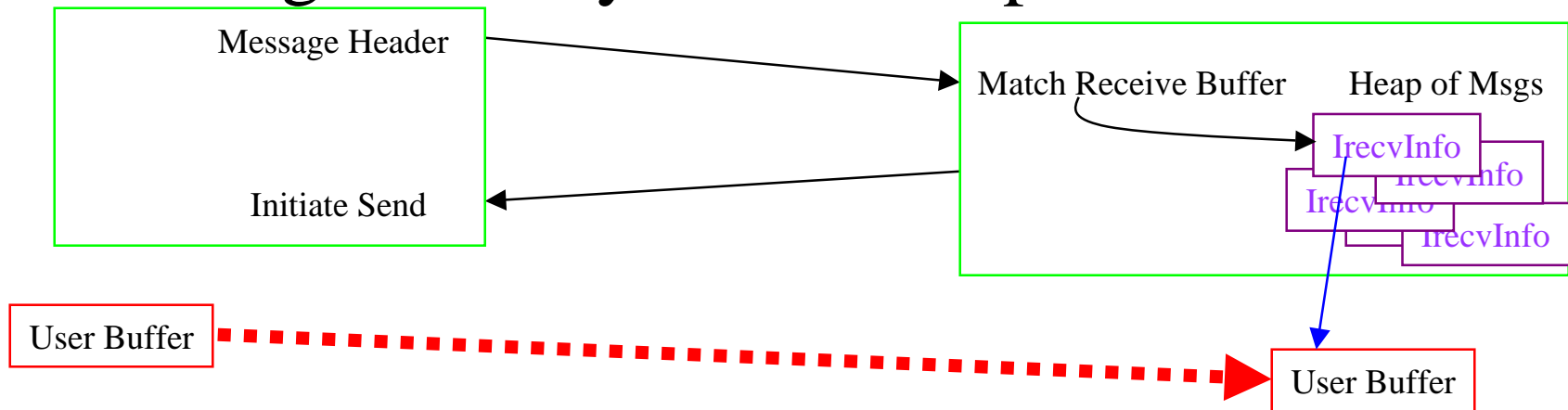
- Process model (separate address space)
 - Global name spaces are bad for programmers but good for compilers
 - Processes provide an effective model of memory locality
 - Think of threads with no shared named variables
 - Process creation also collective (scalable)
 - An MPI “process” does *not* need to be an OS process
- Ignore simple SIMD cases
 - Communication patterns unknown at compile-time
- MPI supports modular and hierarchical programming

Implementing MPI

- Implementing MPI has stressed most platforms
- Progress
 - guarantee that one process is not blocked by another
- Collective operations
 - important for many applications but require generality
- Performance
 - MPI was designed for single-copy semantics. Why is double-copy common?

Progress: Message Delivery

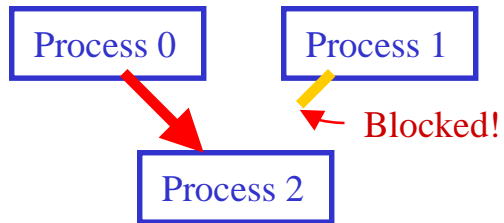
- Message delivery involves 2 parties:



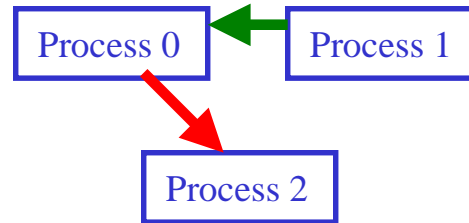
- Data transfer may also involve handshake
 - In fact, must to avoid 3-party interference...

Example of 3-party Interference

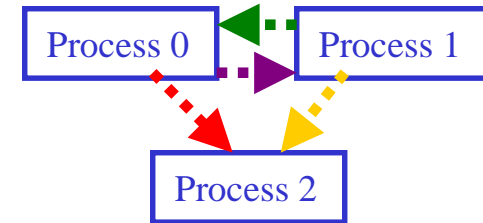
Large Block Transfers



Order Transfers
(requires oracle)

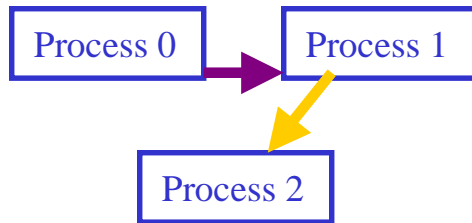


Interleaved Transfers

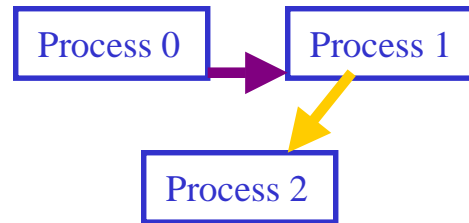


Example of 3-party Interference

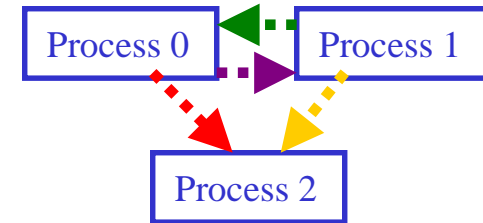
Large Block Transfers



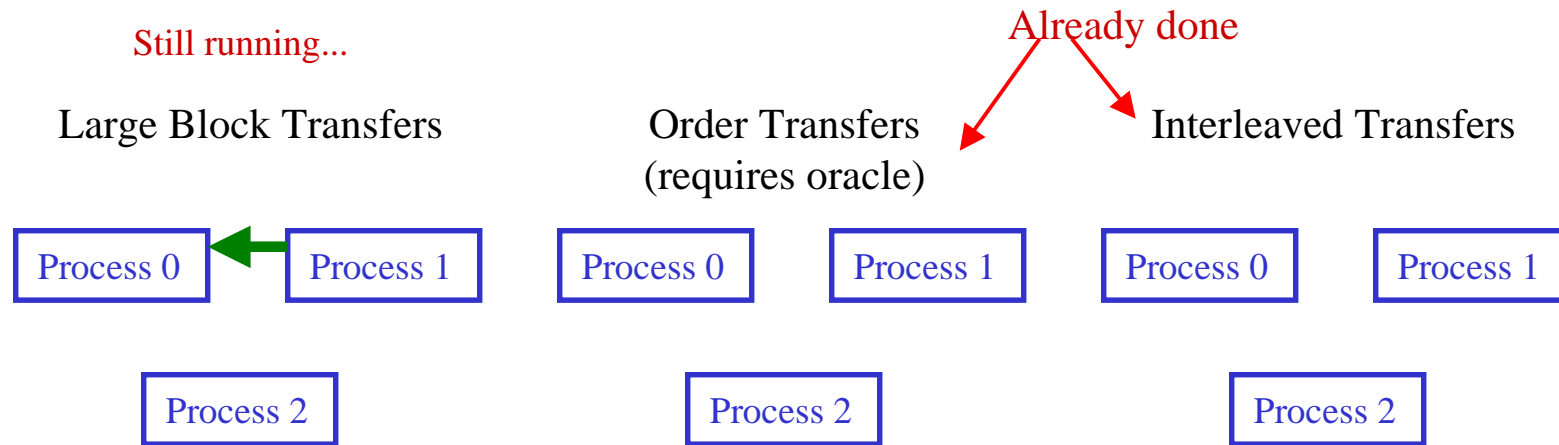
Order Transfers
(requires oracle)



Interleaved Transfers



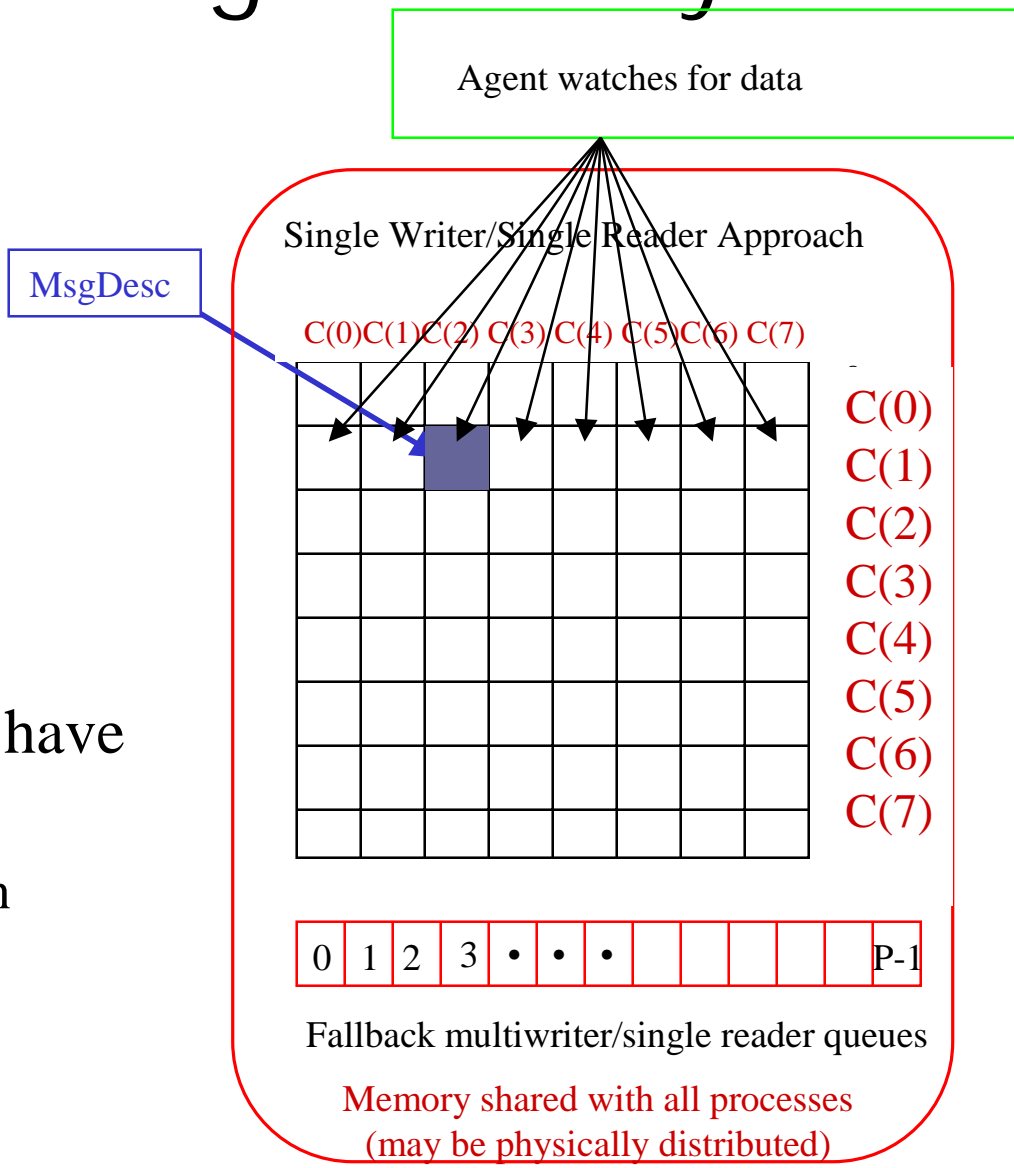
Example of 3-party Interference



- Contention for end-point resources can reduce performance
- Current solutions require active involvement of CPU in managing transfers ...

Sample Message Delivery

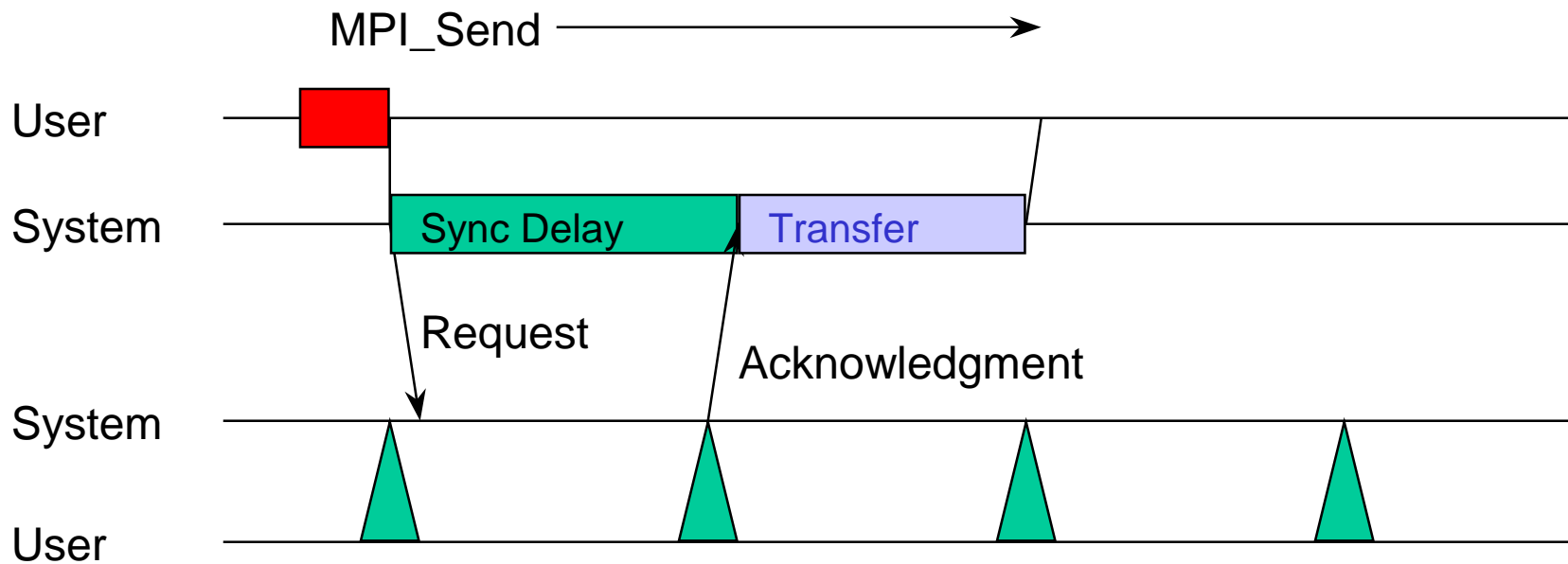
- Deposit header or data packet into incoming queue (messages ordered)
- Fast and Fair
 - but not scalable
- Scalable computations have limited connectivity
 - adapt to communication pattern
- Agent needs to run ...



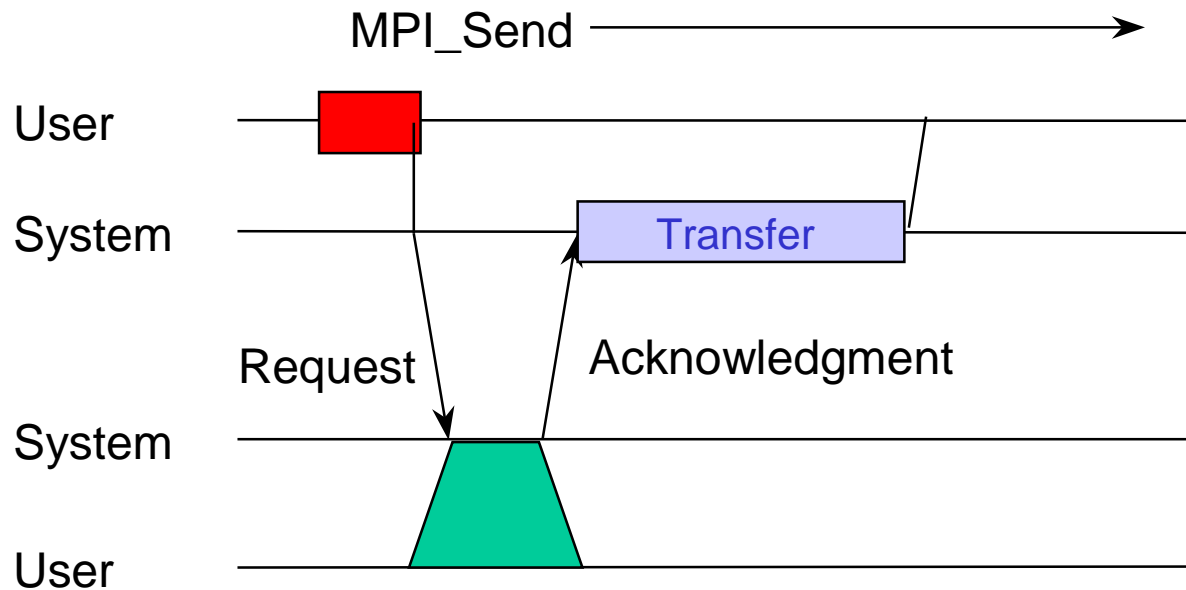
Ensuring Progress

- Autonomous engine
 - What (many) users expect (everything happens as soon as possible)
 - Not what really happens
- Polling
 - Progress only during MPI calls
- Interrupt
 - Service thread scheduled when data arrives

Polling Mode MPI



Interrupt Mode MPI



- Cost of interrupt is (usually) higher than polling

Example of the Effect of Polling

- IBM SP2 MPI_Allreduce times for each mode

Processors	Polling	Interrupt
2	105	260
4	179	490
8	259	746
16	357	1020
32	468	1302
64	670	1614

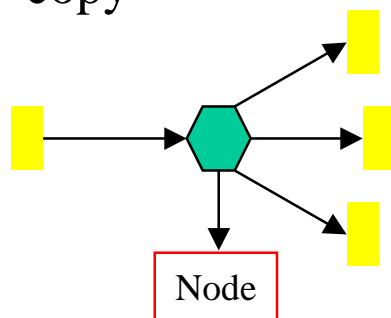
- Times in μ secs. Similar effects on other operations.
- **BUT** some programs (with extensive computing and irregular communication) can show better performance with interrupt mode
- Absolute times enormous ...

Collective Operations

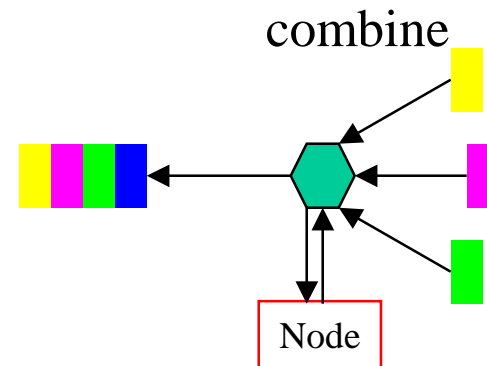
- Must be over arbitrary collections of processes, not just “everyone”
 - Hierarchical algorithms
- Many are short (single word, e.g., reduction)
 - Typical reduction on 64 processes takes 100’s of microseconds = tens of thousands of FLOPS
- Many are synchronizing (like Allreduce)
 - asynchronous *system* events can add huge penalties
 - e.g., IBM has MP_PRIORITY to tune (but can cause deadlocks)

Support for Fast Collective Operations

- Collective groups in MPI are relatively static
 - Setup does not need to be cheap to be useful
- Simple communication building blocks
 - copy



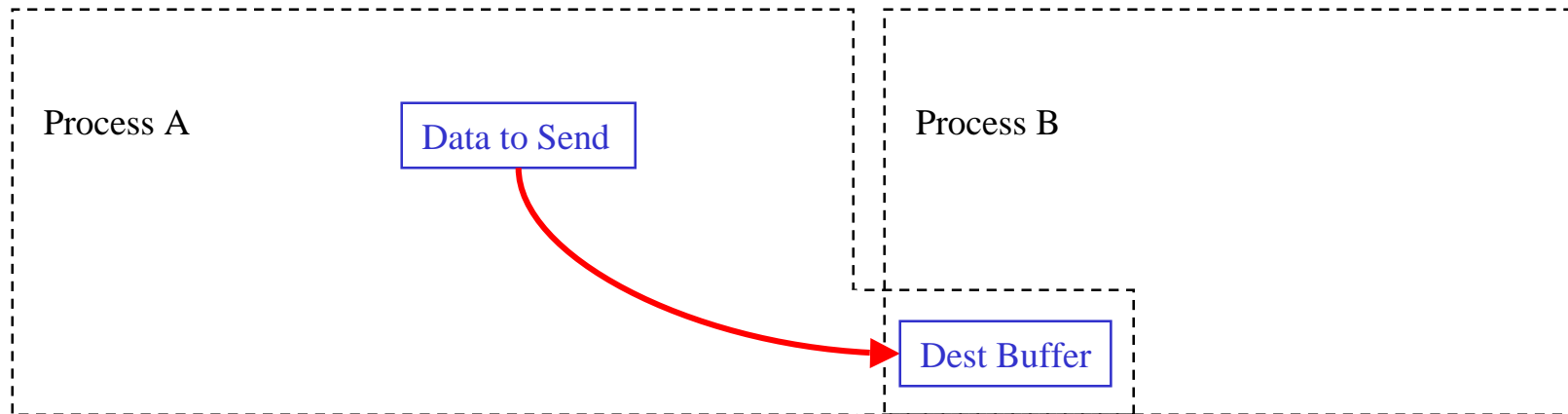
and



- Combine is a more general form of join (fast barrier)
- Build scan and reduce from combine
- Find a balance between complexity in the network and performance of collective operations

Performance

- Single copy between processes requires that the address spaces be “shared” in some sense



- Easy (almost) to do on T3x
- One SMP vendor did this for a while and then gave up
- VIA allows this, but has implementation limits on the amount of memory that can be used
- Other factors
 - Handling cache coherence, even locally, contributes to latency

Wish List for MPI

- Any fast way to maintain progress
 - Threads (without sacrificing performance)
 - Lock-free data structures need powerful atomic ops (compare&swap, multiword LL/SC, ...)
- Fast, collective data motion, including copy and join
 - Can build scatter, gather, broadcast, and allreduce and scan (using redundant computation)
- Single-copy support
 - Collections of processes sharing (some) address space
 - Remote memory operations a good model