Tuning MPI Programs for Peak Performance

William Gropp
Ewing Lusk
Argonne National Laboratory

1

Overview

- Background and Models
- Measurement techniques
- Tuning for performance

- Online examples available at http://www.mcs.anl.gov/mpi/tutorials/perf
- ftp://ftp.mcs.anl.gov/mpi/misc/mpiexmpl.tar.gz contains source code and run scripts that allows you to evaluate your own MPI implementation
- Examples based on vendor implementations available in October, 1996

2

Background and Models

- Goal : Develop a suite of *a small number of* techniques that are generally applicable
- Present many examples of how these apply

- Develop a Computational Model
- Describe Sample Problems
- Performance Model

3

Goals of the Tutorial

- Understand the effect of different MPI functions that accomplish the same communication
- Understand how MPI implementations work
- Know how to discover and understand performance problems
- Be able to predict performance
- Understand the consequences of different algorithms
- Know how to use different MPI functions to solve performance problems

4

Background

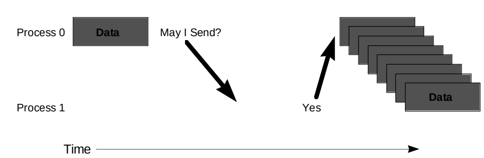
- Quick review of MPI Message passing
- Abstract model of MPI implementations

  Understanding the way in which the different MPI operations are implemented is critical in tuning for performance.

5

What is message passing?

- Data transfer plus synchronization



- Requires cooperation of sender and receiver

6

Quick review of MPI Message passing

- Basic terms
  - » nonblocking - Operation does not wait for completion
  - » synchronous - Completion of send *requires* initiation (but not completion) of receive
  - » ready - *Correct* send requires a matching receive
  - » asynchronous - communication and computation take place simultaneously, ***not*** an MPI concept (implementations *may* use asynchronous methods)

7

Basic Send/Receive modes

- MPI_Send
  - » Sends data. May wait for matching receive. Depends on implementation, message size, and possibly history of computation
- MPI_Recv
  - » Receives data
- MPI_Ssend
  - » Waits for matching receive
- MPI_Rsend
  - » Expects matching receive to be posted

8

Nonblocking Modes

- MPI_Isend
  - » Does not complete until send buffer available
- MPI_Irsend
  - » Expects matching receive to be posted when called
- MPI_Issend
  - » Does not complete until buffer available *and* matching receive posted
- MPI_Irecv
  - » Does not complete until receive buffer available (e.g., message received)

9

Completion

- MPI_Test
  - » Nonblocking test for the completion of a nonblocking operation
- MPI_Wait
  - » Blocking test
- MPI_Testall, MPI_Waitall
  - » For all in a collection of requests
- MPI_Testany, MPI_Waitany
- MPI_Testsome, MPI_Waitsome

10

Persistent Communications

- MPI_Send_init
  - » Creates a request (like an MPI_Isend) but doesn't start it
- MPI_Start
  - » Actually begin an operation
- MPI_Startall
  - » Start all in a collection
- Also MPI_Recv_init, MPI_Rsend_init, MPI_Ssend_init, MPI_Bsend_init

11

Testing for Messages

- MPI_Probe
  - » Blocking test for a message in a specific communicator
- MPI_Iprobe
  - » Nonblocking test
- No way to test in all/any communicator

12

Buffered Communications

- MPI_Bsend
  - » May use user-defined buffer
- MPI_Buffer_attach
  - » Defines buffer for all buffered sends
- MPI_Buffer_detach
  - » Completes all pending buffered sends and releases buffer
- MPI_Ibsend
  - » Nonblocking version of MPI_Bsend

13

Abstract Model of MPI Implementation

- The MPI Automaton
  - » Mechanism that implements MPI
  - » Handles the coordination with the network
  - » Polling, interrupt, shared processor implementations
- MPI Implementation
  - » The protocols used to deliver messages

14

The MPI Automaton

- Some mechanism must manage the coordination.
  - » Polling - User's process checks for MPI messages; low overhead
  - » Interrupt - Processes respond "immediately" to messages; higher overhead but more responsive
  - » Shared processor - Like constant polling
- Combinations possible
  - » Polling with regular timer interrupts
  - » Threads

15

Message protocols

- Message consists of "envelope" and data
  - » Envelope contains tag, communicator, length, source information, plus impl. private data
- Short
  - » Message data (message for short) sent with envelope
- Eager
  - » Message sent assuming destination can store
- Rendezvous
  - » Message not sent until destination oks

16

Special Protocols for DSM

- Message passing a good way to use distributed shared memory (DSM) machines because it provides a way to express memory locality.
- Put
  - » Sender puts to destination memory (user or MPI buffer). Like Eager
- Get
  - » Receiver gets data from sender or MPI buffer. Like Rendezvous.
- Short, long, rendezvous versions of these

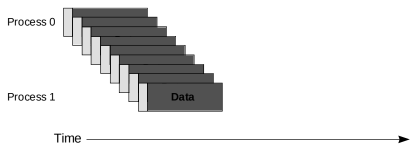
17

Message Protocol Details

- Eager not Rsend, rendezvous not Ssend resp., but related
- User versus system buffer space
- Packetization
- Collective operations
- Datatypes, particularly non-contiguous
  - » Handling of important special cases
    - – Constant stride
    - – Contiguous structures

18

Eager Protocol



- Data delivered to process 1
  - » No matching receive may exist; process 1 must then buffer and copy.

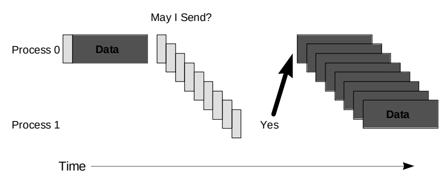
Eager Features

- Reduces synchronization delays
- Simplifies programming (just MPI_Send)
- Requires significant buffering
- May require active involvement of CPU to drain network at receiver's end
- May introduce additional copy (buffer to final destination)

How Scaleable is Eager Delivery?

- Buffering must be reserved for arbitrary senders
- User-model mismatch (often expect buffering allocated entirely to "used" connections).
- Common approach in implementations is to provide same buffering for all members of MPI_COMM_WORLD; this is optimizing for non-scaleable computations

Rendezvous Protocol



- Envelope delivered
- Data delivered when user-buffer available
  - » Only buffering of envelopes required

Rendezvous Features

- Robust and safe
  - » (except for limit on the number of envelopes…)
- May remove copy (user to user direct)
- More complex programming (waits/tests)
- May introduce synchronization delays (waiting for receiver to ok send)

Short Protocol

- Data is part of the envelope
- Otherwise like eager protocol
- May be performance optimization in interconnection system for short messages, particularly for networks that send fixed-length packets

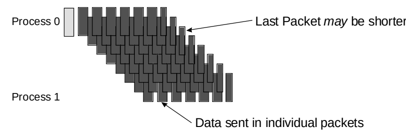
User and System Buffering

- Where is data stored (or staged) while being sent?
  - » User's memory
    - – Allocated on the fly
    - – Preallocated
  - » System memory
    - – May be limited
    - – Special memory may be faster

25

Packetization

- Some networks send data in discrete chuncks called *packets*



26

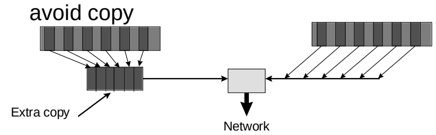
Collective operations

- May be implemented with MPI point-to-point
- Implementations can optimize for small (latency), large (bandwidth), or both.
- Generality of some MPI collective operations can limit performance
  - » Routines must assume datatypes are general and non-contiguous
  - » Time/memory tradeoffs (for internal temporary buffers)

27

Non-contiguous Datatypes

- Provided to *allow* MPI implementations to avoid copy



  - » Not widely implemented *yet*
- Handling of important special cases
  - » Constant stride
  - » Contiguous structures

28

Why Bsend

- Buffer space not infinite
- Careful management of buffering required for *correctness* and *performance*
- MPI provides user control over buffering modes
- Allows implementations freedom to pick specific protocols
  - » User's give up some control for more convenient general-purpose send (MPI_Send)

29

BREAK

Performance Model

- Basic Model
  - » Needed to evaluate approaches
  - » Must be simple
- Related effects
  - » Understand deviations from the model

31

Main components

- Latency and Bandwidth
- Synchronization delays
- Load balancing

32

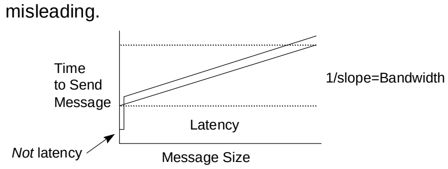
Latency and Bandwidth

- Simplest model s + rn
- s includes both hardware (gate delays) and software (context switch, setup)
- r includes both hardware (raw bandwidth of interconnection and memory system) and software (packetization, copies between user and system)
- head-to-head and pingpong values may differ

33

Interpreting Latency and Bandwidth

- Bandwidth is the inverse of the slope of the line
  time = latency + (1/rate) size_of_message
- Latency is sometimes described as "time to send a message of zero bytes". This is true *only* for the simple model. The number quoted is sometimes misleading.



34

Including contention

- Lack of contention greatest limitation of latency/bandwidth model
- Hyperbolic model of Stoica, Sultan, and Keyes provides a way to estimate effects of contention for different communication patterns see ftp://ftp.icase.edu/pub/techreports/96/96-34.ps.Z

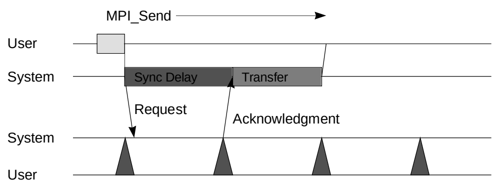
35

Synchronization Delays

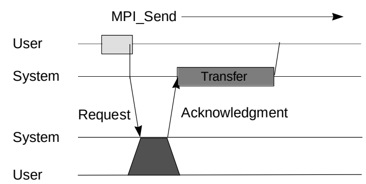
- Message passing is a cooperative method - if the partner doesn't react quickly, a delay results
- There is a performance tradeoff caused by reacting quickly - it requires devoting resources to checking for things to do

36

Polling Mode MPI



Interrupt Mode MPI



- Cost of interrupt higher than polling (usually)

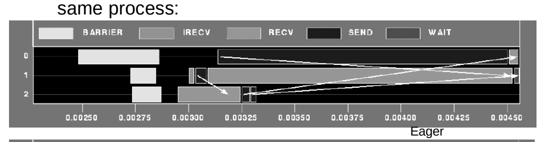
Example of the effect of Polling

- IBM SP2 MPI_Allreduce times for each mode

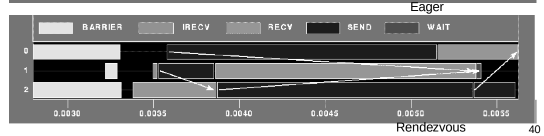
- Times in usecs. Similar effects on other operations.
- ***BUT*** some programs (with extensive computing) can show better performance with interrupt mode

More on Synchronization Delays

- 3 processors sending data, with one sending a short message and another sending a long message to the same process:



Eager



Rendezvous

Load Balancing

- Compute total work, divide by P (T_p)
- Compute maximum for any process (T_max)
- Difference is load imbalance; Efficiency is bounded by T_p/T_max.

Related effects

- Contention
- Memory Copies
- Packet sizes and stepping

Contention

- Point-to-point analysis ignores fact that communications links (usually) shared
- Easiest model is to equally share bandwidth (if K can shared at one time, give each 1/K of the bandwidth).
- "Topology doesn't matter anymore" is *not* true, but there is less you can do about it (just like cache memory)

43

Effect of contention

- IBM SP2 has a multistage switch. This test shows the point-to-point bandwidth with half the nodes sending and half receiving

44

Memory copies

- Memory copies are the primary source of performance problems
- Cost of non-contiguous datatypes
- Single processor memcpy is often much slower than the hardware.
  Measured memcpy performance:

45

Example: Performance Impact of Memory Copies

- Assume n bytes sent eagerly (and buffered)
  - » $s + rn + cn$
- Rendezvous, not buffered
  - » $s + s + (s + rn)$
- Rendezvous faster if $s < cn/2$
  - » Assumes no delays in responding to rendezvous control information

46

Example: Why MPI Datatypes

- Handling non-contiguous data
- Assume must pack/unpack on each end
  - » $cn + (s + rn) + cn$
- Can move directly
  - » $s + r' n$
  - » r' probably > r but < $(2c+r)$
- MPI implementation must copy data anyway (into network buffer or shared memory); having the datatype permits removing 2 copies

47

Performance of MPI Datatypes

- Test of 1000 element vector of doubles with stride of 24 doubles.
  - » MPI_Type_vector
  - » MPI_Type_struct (MPI_UB for stride)
  - » User packs and unpacks by hand
- Performance very dependent on implementation; should improve with time

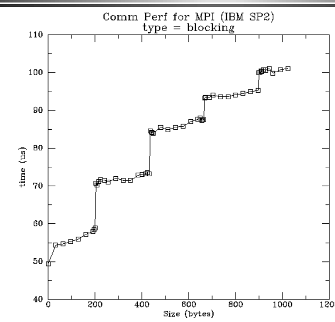
48

Packet sizes/stepping

- Data sent in fixed- or maximum-sized "packets"
- Introduces a ceil(n/packet_size) term

49

Example of Packetization



Packets contain 232 bytes of data. (first is 200 bytes, so MPI header is probably 32 bytes).

Data from mpptest, available at ftp://ftp.mcs.anl.gov/pub/mpi/misc/perftest.tar.gz

50

Measurement Techniques

- Using MPI_WTIME
- MPI oriented Tools
- Custom techniques
- Evaluating the data

51

Timing with WTIME/WTICK

- Overhead of timing
- Dealing with granularity of timer
  - » timers are *not* continuous - MPI_Wtick
- MPI_WTIME_IS_GLOBAL
  - » What to do if value is false
  - » NTP algorithms, simpler expedients
- Cache/setup effects
- Sample timing harness
  - » limitations of the harness

52

Sample Timing Harness

- Average times, make several trials

```
for (k<nloop) {
    t1 = MPI_Wtime();
    for (I<maxloop) {
        <operation to be timed>
        }
    time = MPI_Wtime() - t1;
    if (time < tfinal) tfinal = time;
    }
```

53

Pitfalls in timing

- Time too short:

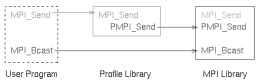
```
t = MPI_Wtime();
MPI_Send(…);
time = MPI_Wtime() - t;
```

- Underestimates by MPI_Wtick, over by cost of calling MPI_Wtime
- "Correcting" MPI_Wtime by subtracting average of MPI_Wtime calls overestimates MPI_Wtime
- Code not paged in (always run at least twice)
- Minimums not what users see
- Tests with 2 processors may not be representative
  - » T3D has processors in pairs, pingpong give 130 MB/sec for 2 but 75 MB/sec for 4

54

Using PMPI routines

- PMPI allows selective replacement of MPI routines at link time (no need to recompile)



- Some libraries already make use of PMPI
- Some MPI implementations have PMPI bugs
  - » IBM - PMPI_Wtime() returns 0
  - » Cray - PMPI in a separate library that some installations have not installed

55

Logging and visualization

- Upshot and MPE tools
- VT
- Pablo, Paragraph, and Paradyn
- Other vendor tools
- Validation by running with coarse-grain logging

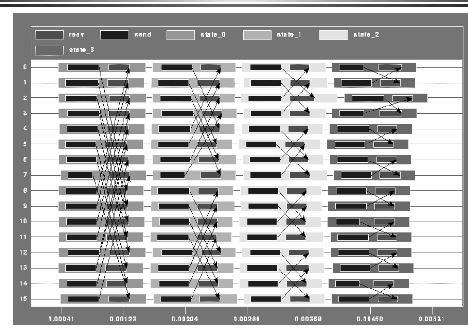
56

Upshot and MPE

- Automatic logging
  - » Uses PMPI interface and a special library
  - » mpicc -mpilog …
  - » mpirun -np 8 a.out ….
- User-directed logging
  - » MPE_Log_event calls inserted by user
  - » MPE_Describe_state defines user state
  - » states may be nested
- Works with MPICH and vendor MPIs

57

Sample Upshot Output



58

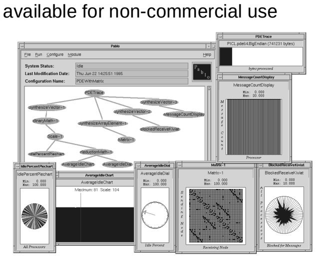
IBM VT

- Part of IBM SP2 software
- Automatic trace option
- http://www.rs6000/ibm.com/software/sp_products/pe.html

59

Pablo

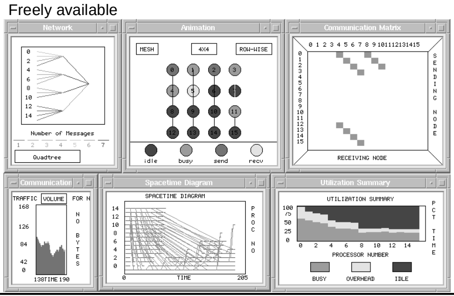
- http://www-pablo.cs.uiuc.edu/Projects/Pablo/pablo.html
- Tutorial at http://www.tc.cornell.edu/Edu/Tutor/Paplo
- Freely available for non-commercial use



60

Paragraph

- http://www.ncsa.uiuc.edu/Apps/MCS/ParaGraph/ParaGraph.html
- Freely available



61

Paradyn

- http://www.cs.wisc.edu/~paradyn
- Freely available in binary for non-commercial use

62

AIMS

- http://www.nas.nasa.gov/NAS/Tools/Projects/AIMS
- Available on request

63

Other visualization tools

- Many vendors developing tools
- Many research groups exploring new approaches
- IEEE Computer Nov 95 has a sampling of tools

64

Validating the logging

- Logging introduces some timing differences
- Can change the behavior of the computation (and not all can be filtered out, for example, if the presence of logging causes different protocols to be used)
- Compare coarser grain timings to check

65

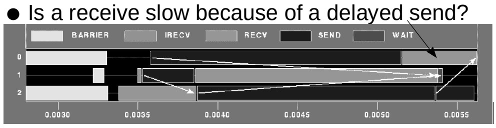
Deficiency analysis/filtering techniques

- Only interested in
  - » taking too long, and
  - » too slow (compared to model)
- Filter logs with thresholds for both; use this to further instrument

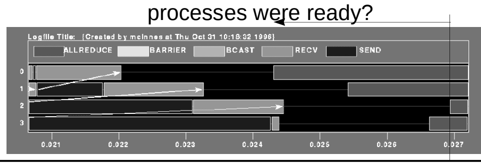
66

Correlating processes for synchronization delays

- Is a receive slow because of a delayed send?



- Is a collective operation slow because not all processes were ready?



67

LUNCH

Tuning for Performance

- Performance Techniques
- MPI-Specific tuning
- Pitfalls

69

Tuning for Performance (General Techniques)

- Aggregation
- Decomposition
- Load Balancing
- Changing the Algorithm

70

Aggregation to reduce latency

- Collect many small messages into a single large message
  - » Example: sending each component separately vs. sending them together; packing cost (usually) outweighed by latency.
    2(s+r) vs *(s + 2r + 2c)*
  - » Note that cache effects suggest keeping components together
- Use of collective when many copies bcast/gather

71

Aggregation in Collective operations

- Combining collective operations
  - » Much cheaper to do one 2-element allreduce than two 1-element allreduce
  - » Example - combine allreduce in algorithm with error or convergence values
- Bulk synchronous programming model http://www.bsp-worldwide.org

72

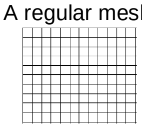
Decomposition

- Implementation of parallel data structures
- Regular meshes
  - » Different decompositions
  - » MPI Topologies
  - » Load balancing

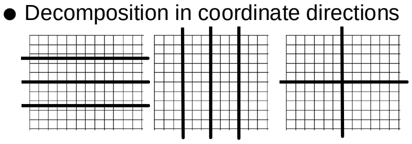
73

Regular data structures

- A regular mesh



- Decomposition in coordinate directions



74

Issues in choosing a decomposition

- 1 vs 2 vs 3d
- Minimize surface to volume (latency vs. bandwidth tradeoff)
- More complex decompositions (e.g.., hexagons in 2-d) possible, but *usually* not worth it
- Careful with relatively small problems - latency may dominate

75

More issues in decompositions

- Relationship to unstructured/irregular problems
  - » Strategies attempt to minimize surface to volume or related measure
- Interaction with algorithmic changes (ordering with/against flow direction)
- Dynamic meshes (refinement, reconfiguration)

76

MPI Support for regular decompositions

- Using topology routines
  - » cart create
  - » shift/coord
- Why you should use the topology routines
  - » Simple to use (why not?)
  - » Allow MPI implementation to provide low expected contention layout of processes (contention can matter)
  - » Remember, contention still matters; a good mapping can reduce contention effects

77

Performance Issues of Decompositions

- Interaction with load balancing (matrix factor example)
- Use of scaling behavior to identify problems
  - » fixed time suggests a poor decomposition (e.g.., slab in 3-d, line in 2-d)
- Topologies with corners
  - » 9-point, 2-d stencil trick
  - » Another form of aggregation

78

Example: Matrix factoring

- Two simple decompositions: by rows and by blocks
- In row form basic work unit is n words (a row); this limits work distribution
- In block form, basic work unit is 1 word; maximizes work distribution

79

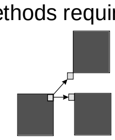
Scaling of decompositions

- 1 - D slices (plates)
  - »
  - » Communication constant independent of *p*
- 2 - D slices (bars)
  - »
- 3 - D slices (cubes)
  - »

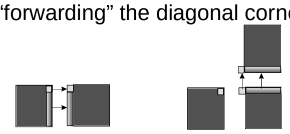
80

Sharing Data in Decompositions

- Many methods require shared edges and corners:



- Send in two phases, with second phase “forwarding” the diagonal corner



81

Irregular Data Structures

- Many of the same issues
  - » Aggregation
  - » Latency/Bandwidth tradeoffs
- Actual choice of decomposition complex
  - » Spectral bisection
  - » Coordinate-based
  - » Graph cutting

82

Changing the Algorithm

- Some algorithms are just not good candidates for parallelization
  - » If an approximation, another approximation may be a better choice (different physical model)
  - » If part of an iterative method, another iteration may be better (different numerical model)
  - » Need not be less efficient
    - – Hierarchical domain decomposition methods are an example

83

Issues in Changing the Algorithm

- Trading communication (slow) for computation (fast)
  - » Example: Solving a small linear system whose results are needed by all processes
    - – Parallel solution latency dominated (*ps* + …)
    - – Single solution uses gather/bcast, leaves processes idle
    - – All solve uses single gather (but has duplicate work)
- Multicoloring
  - » Divide computation into different “colors”, each of which is independent of the others
- Loop unrolling

84

Trading Communication for Computation

- For slowly converging algorithms, taking a number of steps and then checking convergence (rather than checking at each iteration) is another form of blocking
- Can trade bandwidth/computation for latency (unroll a compute loop once, do a single send of more data, do duplicate computation)

85

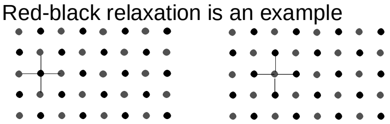
Analysis of Communication Tradeoff

- Per iteration costs
  » Tw - computation (work)
  » Tc - communication (latency bound)
  » n - steps per communication (when blocked)
  » m - total number of steps
- Optimal block size:
  » minimize overhead: *(m/n)Tc + n/2 Tw*
  »

86

Multicoloring

- Example of changing algorithm
- Red-black relaxation is an example



- Multicoloring a successful approach to Incomplete factorizations (ICC) http://www.mcs.anl.gov/Projects/blocksolve

87

Loop Unrolling

- Classic technique for improving performance

```
do I=1,10
  call f(I)
  exchange data for step I
```

changed to

```
do I=1,10,2
 call f(I)
 call f(I+1)
 exchange data for steps I and I+1
```

88

Aggregation in Loop Unrolling

- Blocking is another form of loop unrolling
  » Loop unrolling can be used to get longer messages
-
```
do timestep=1,n
    compute
    exchange
do timestep=1,n,2
    compute(timestep) (and some of timestep-1)
    compute(timestep+1)
    exchange 2 timesteps
```

89

Using Associativity

- Can allow reordering of communication
- May change numerical properties
- Example of Gramm-Schmidt orthogonalization (dot(w1,w2)=0):
  v = v - dot(v,w1) w1
  v = v - dot(v,w2) w2
  (two Allreduces) replaced with
  a = dot(v,w1); b = dot(v,w2)   (Allreduce of 2)
  v = v - a w1 - b w2
- This is ***NOT THE SAME*** in floating-point arithmetic

90

Load balancing

- Small amounts of work imbalance lead to large losses in performance
- First question is: is load balancing central to the algorithm or part of performance tuning?
  - » Central - master/slave models, multilevel work masters
  - » Tuning - decomposition tuning

Identifying Load Imbalances

- Identifying (distinguishing from latency/synchronization overhead)
- Poor load balance focuses attention on collective operations because the implicit synchronization of the collective operation "equalizes" the time taken on each process
  - » Can generate the appearance of good load balance if not timed correctly

Master/slave models

- Multilevel masters
  - » Work stealing (http://theory.lcs.mit.edu/~cilk)
- Using Ssend (or Issend) to manage buffers (lest you overwhelm buffering). Besides, its what is wanted
- Fairness in message-passing
  - » Ensure that no slave is starved of attention from master
  - » Simple codes can fail to express this

Implementing Fairness

- Waitsome
- Master's code is

```
for (i=0;i<n;i++)
     MPI_Irecv( …, &r[i]);
while (not done) {
  MPI_Waitsome( n, r, &nready, i_ready,
                statuses );
  … Process r[i_ready] and repost
}
```

- Can double buffer with Issend to control buffer use and allow slaves to overlap synchronization delays

Tuning Decompositions for Load Balance

- Static data decomposition
  - » Different boundary behavior means can't just count "mesh points"
  - » Example is 3 domains, 2-d, strips, need to make edges larger for
    - – boundary effects (computation)
    - – communication (total smaller)
  - » Rule of thumb for Matrix - equalize number of elements without breaking rows. Example of a good compromise between perfection and workability.

BREAK

MPI-Specific Tuning

- MPI Datatypes
- Different Send/Recv modes
- MPI Implementation parameters
- Miscellaneous Tricks

97

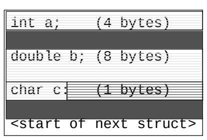
Constant stride datatypes

- MPI_Type_vector
  - » Good for fixed length operation (send 1 instance of a vector)
  - » Some implementations optimize (MPICH, Cray T3D)
- MPI_Type_struct with MPI_UB entry
  - » Good for variable length operation, but with fixed stride
  - » No implementation known to optimize (yet)
- User pack/unpack (with*out* MPI_Pack)

98

Contiguous Structures

- MPI specifies that “gaps” in a structure are *not* copied



- Can cause significant performance loss

99

Improving structure performance

- Use MPI_BYTE for homogeneous-only applications
- Add “fillers” for structure definition
  - » requires implementation to detect contiguous
  - » Note that this is trading extra data for better performance of MPI implementation
  - » MPI-2 proposals have tried to address this

100

Tuning for MPI protocols

- Aggressive Eager
- Rendezvous with sender push
- Rendezvous with receiver pull
- Rendezvous blocking (push or pull)

101

Aggressive Eager

- Performance problem: extra copies
- Possible deadlock for inadequate eager buffering

102

Tuning for Aggressive Eager

- Ensure that receives are posted before sends
- MPI_Issend can be used to express “wait until receive is posted”

103

Rendezvous with Sender Push

- Extra latency
- Possible delays while waiting for sender to begin

104

Rendezvous Blocking

- What happens once sender and receiver rendezvous?
  - » Sender (push) or receiver (pull) may complete operation
  - » May block other operations while completing
- Performance tradeoff
  - » If operation does *not* block (by checking for other requests), it adds latency or reduces bandwidth.
- Can reduce performance if a receiver, having acknowledged a send, must wait for the sender to complete a separate operation that it has started.

105

Tuning for Rendezvous with Sender Push

- Ensure receives posted before sends
  - » better, ensure receives occur before computation starts; may be better to do sends before receives
- Ensure that sends have time to start transfers
- Can use short control messages
- Beware of the cost of extra messages

106

Rendezvous with Receiver Pull

- Possible delays while waiting for receiver to begin

107

Tuning for Rendezvous with Receiver Pull

- Place MPI_Isends before receives
- Use short control messages to ensure matches
- Beware of the cost of extra messages

108

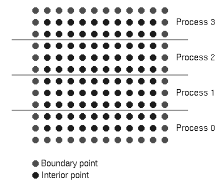
Sample Problems

- Multiparty data exchange
- Jacobi iteration in 2 dimensions
  - » Model for PDEs, Matrix-vector products
  - » Algorithms with surface/volume behavior
  - » Issues similar to unstructured grid problems (but harder to illustrate)
- Others at http://www.mcs.anl.gov/mpi/tutorials/perf

109

Jacobi Iteration

- Simple parallel data structure



- Processes exchange rows with neighbors

110

Background to tests

- Goals
  - » Identify better performing idioms for the same communication operation
  - » Understand these by understanding the underlying MPI process
  - » Provide a starting point for evaluating additional options (there are many ways to write even simple codes)

111

Different send/recv modes

- MPI provides many different ways to perform a send/recv
- Choose different ways to manage buffering (avoid copying) and synchronization
- Interaction with polling and interrupt modes

112

Some send/recv approaches

- Based on operation hypothesis. Most of these are for polling mode. Each of the following is a hypothesis that the experiments test
  - » Better to start receives first
  - » Ensure recvs posted before sends
  - » Ordered (no overlap)
  - » Nonblocking operations, overlap effective
  - » Use of Ssend, Rsend versions (EPCC/T3D can prefer Ssend over Send; uses Send for buffered send)
  - » Manually advance automaton
- Persistent operations

113

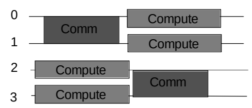
Scheduling Communications

- Is it better to use MPI_Waitall or to schedule/order the requests?
  - » Does the implementation complete a Waitall in any order or does it prefer requests as ordered in the array of requests?
- In principle, it should always be best to let MPI schedule the operations. In practice, it may be better to order either the short or long messages first, depending on how data is transferred.

114

Scheduling for contention

- Many programs alternate between communication and computation phases
- Contention can reduce effective bandwidth
- Consider restructuring program so that some nodes communicate while others compute:



115

Some Example Results

- Sumarize some different approaches
- More details at http://www.mcs.anl.gov/mpi/tutorial/perf/mpiexmpl/src3/runs.html

116

Send and Recv

- Simplest use of send and recv

- Very poor performance on SP2
  - » Rendezvous sequentializes sends/receives
- OK performance on TD3 (implementation tends to buffer operations)

117

Better to start receives first

- Irecv, Isend, Waitall

- Ok performance

118

Ensure recvs posted before sends

- Irecv, Sendrecv/Barrier, Rsend, Waitall

119

Receives posted before sends

- Best performer on SP2
- Fails to run on SGI (needs cancel) and T3D (core dumps)

120

Ordered (no overlap)

- Send, Recv or Recv, Send
- Sendrecv (exchange)
- Sendrecv (shift)

121

Shift with MPI_Sendrecv

- Performs reasonably well; simpler than many other approaches
- T3D performance is ok but other approaches are better

122

Use of Ssend versions

- Ssend allows send to wait until receive ready
  - » At least one implementation (T3D) gives better performance for Ssend than for Send

123

Nonblocking operations, overlap effective

- Isend, Irecv, Waitall
- A variant uses Waitsome with computation

124

Persistent operations

- Potential saving
  - » Allocation of MPI_Request
  - » Validating and storing arguments
- Variations of example
  - » sendinit, recvinit, startall, waitall
  - » startall(recvs), sendrecv/barrier, startall(rsends), waitall
- Vendor implementations are buggy

125

Manually advance automaton

- irecv, isend, iprobe in computation, waitall
- To test for messages:

```
MPI_Iprobe( MPI_ANY_SOURCE, 0,
  MPI_COMM_WORLD, &flag, &status )
```

126

Summary of Results

- Better to start sends before receives
  - » Most implementations use rendezvous protocols for long messages
- Synchronous sends better on T3D
  - » otherwise system buffers
- MPI_Rsend can offer some performance gain on SP2
  - » as long as receives can be guaranteed without extra messages

127

MPI Implementation parameters

- Each MPI implementation makes different choices about protocols, allocation of memory, etc
- Some of these parameters can be adjusted for each run

128

IBM SP

- MP_EAGER_LIMIT
  - » Note this is an example of non-scaleable implementation (4k for 16 or less, …)
- MP_BUFFER_MEM
- MP_CSS_INTERRUPT

129

MPICH

- short size
  - » Messages of this size and smaller are sent with the envelope. Configurable; between 1k and 16k
- eager size
  - » Messages of this size are sent eagerly. Starts at 128000 but can change based on amount of unreceived data
- Network socket sizes, window sizes
  - » TCP performance critically dependent on proper choice of network parameters.

130

T3D

- Cray native MPI
  - » http://www.epcc.ed.ac.uk/t3dmpi/Product
  - » MPI_BUFFER_MAX, MPI_SM_POOL, others
- MPICH

131

LAM

- LAM provides careful management of message resources, including envelopes
  - » MPI_GER (Guaranteed Envelope Resources)
- http://www.osc.edu/lam.html

132

MPI-2 techniques

- Nonblocking collective
  - » nonblocking reductions
  - » nonblocking barrier for termination
- One-sided
  - » put/get/barrier to reduce synchronization points
- Generalized requests (interrupt receive)
  - » MPI-2 will probably do this via MPI_RECV in a separate thread

134

Pitfalls I

- Replacing a good sequential algorithm with a terrible parallel one
  - » Classic example is using Jacobi iteration instead of hierarchically-preconditioned conjugate gradient
- Assuming floating point associative
  - » common mistake for all-reductions
  - » Lack of bitwise-identical results
    - – For example, don't assume that each process will compute exactly the same values given the same data unless you can guarantee the same order and intermediate forms (beware compilers!)

135

Pitfalls II

- Change numerical behavior for ill-posed or poorly conditioned problems (e.g., unmodified Gramm-Schmidt)
- Buffering limits
  - » Tuning for eager sends causes portability problems
  - » Fragility of application when presented with a larger dataset

136

Review of Techniques I

- Select algorithm for effective parallelism
  - » Don't sacrifice elapsed-time performance for artificial "efficiency" measurements
- Choose effective data decompositions
  - » Tune to problem sizes of interest
  - » Look carefully at parts of code that don't scale with *p*
- Model and measure
  - » Look for underperforming sections of code

137

Review of Techniques II

- Apply Aggregation to reduce latency
  - » Increase grain, combine elements
  - » Unroll loops, block operations
- Manage message flow to avoid handshake delays
  - » Use MPI message modes
  - » Handshake messages
  - » MPI-2 features (onesided, etc)
- Watch out for pitfalls (particularly floating point)

138

Review of Techniques III

- Match send/receive order to underlying implementation
- Aggressive eager
  - » receives before sends; MPI_Issend or MPI_Irsend
- Rendezvous with Sender Push
  - » receives before sends (sometimes…)
  - » consider short messages to ensure sends started
  - » faster barriers can also be used
- Rendezvous with Receiver Pull
  - » sends before receives

139