SCALI
Scalable Linux Systems

Lars Paul Huse (Scali AS):

Message passing programming with ScaMPI - Do and don'ts

E-mail: lph@scali.com
Web: http://www.scali.com
S-mail: Scali AS, Olaf Helsets vei 6,
Po.Box 70, Bogerud, N-0621 Oslo, Norway

Presentation outline

SCALI
Scalable Linux Systems

- Introduction to MPI
- ScaMPI features
- ScaMPI vs MPICH
- What can go wrong
- How to get performance
- Buffer allocation control
- Asynchronous programming
- Summing up

MPI Standard

SCALI
Scalable Linux Systems

- Detailed description of application programmers interface (API) for process communication. Programming language include Fortran and C/C++.
- MPI standard only gives advice / hints concerning implementation.
- Wide variety of basic communication modes (blocking / immediate / buffered / …)
- MPI data types may be simple, contiguous, vector, strided vector (non-contiguous), index or structuated.

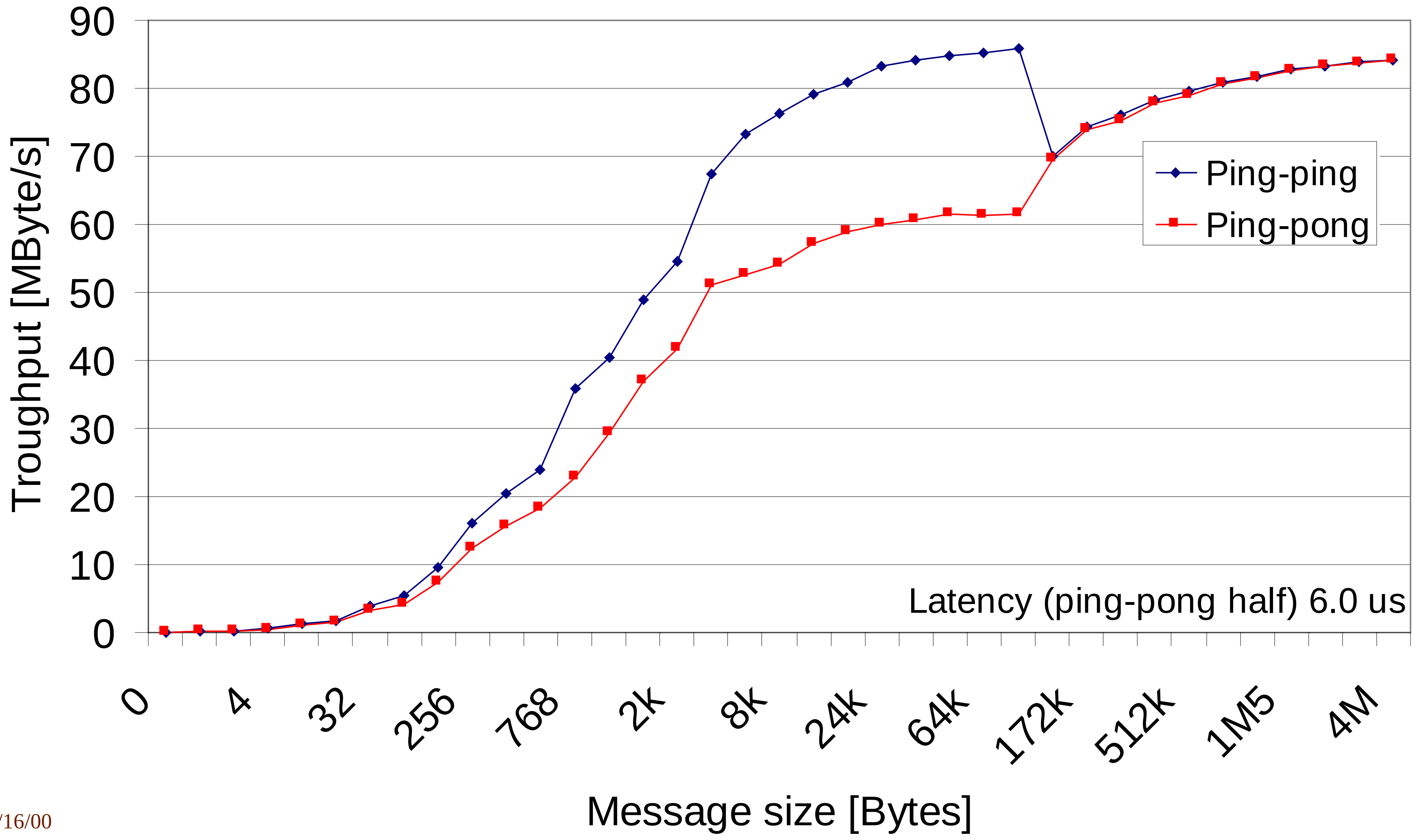
ScaMPI features

ScaMPI is Scali’s MPI (1.1 compliant) implementation over SCI / local shared memory (Write only protocol).

- **Three buffer types (in shared memory):**
  Transporter: Long transfers (rendezvous protocol).
  Eager: Limited number buffered messages.
  Channel: Self synchronized small messages.
- **Safe data transport guarantied by SCI and retransmitting undelivered data.** SCI guarantees correctness of data reaching its destination and that data will be delivered if possible, but not the ordering.

ScaMPI point-to-point performance

SCALI
Scalable Linux Systems



ScaMPI vs MPICH (Ver. 1.2.0)

ScaMPI	MPICH
is thread-safe	is not thread-safe
has one queue per active sender-receiver pair.	has a common queue for each process.
buffers data explicit (fixed buffer-space).	buffers data implicit (always when needed).
is linked as a static or dynamic library.	is linked as a static library only.
uses seamless SHM or network for communication.	communication medium has to be specified at compile time.
uses a multi-program execution model.	uses a single-program execution model
ScaMPI handles passing of command line options.	Different mpi.h header file.

SSP 2.0: ScaMPI 1.9.1 (Scali Software Platform)

- SMP awareness for all collective operations
- Rewritten receive handling to use only a single thread
- Small messages use inlining for MPI_Sendrecv()
- Rewritten immediate handling to use less system resources
- Rewrite of yield handling
- Separate send queue giving the possibility to have more pending send operations
- New locking strategy for allocating shared memory
- SCI accesses from different threads / processes synchronized to improve performance
- ScaBench, Scalis test suite to benchmark MPI based work-station clusters is now added to ScaMPItst package.
- Bugfixes

Why does my application terminate abnormally?

- **The program terminates without an error message: core dump...**
  Investigate the core file, or rerun the program in a debugger (no core on Linux).
- **Are you reasonable certain that your algorithms are MPI safe?**
  Check if every send has a matching receive (MPI_Send $\rightarrow$ MPI_Ssend).
- **The program just hangs.**
  Buffer allocation problem ? Start the program with -init_comm_world specified.
  If the program has a large degree of asynchronisity increase the channel-size.
- **Alignment of double data types.** Reduce operators MINLOC & MAXLOC.
- **Odd behavior / namespace pollution**
  Functions and variables are listed in the mpi.h and mpif.h include file.
  Avoid using OS functions as application function names (send, open, yield)
- **SCI interconnect failures...**
  The program terminates with an ICMS_* message.

How do I optimize MPI performance?

- **Avoid sending messages.** SMP parallel by compiler automatic, threads, ...
- **Avoid sending short messages.**
  Group smaller messages to larger and send as one - ref. bandwidth curve.
- **Use collective operations when suitable, but avoid barriers.**
- **Avoid using MPI_Isend(), MPI_Irecv(), MPI_Bsend(), etc.**
  Using MPI_Sendrecv() transforms to MPI_Isend() and MPI_Recv() !
- **Avoid starving processes - introduce fairness in algorithms.**
- **Avoid creating more threads than available processors.**
- **Performance analysis e.g. VAMPIR profiling software.**
- **Communication / shared memory buffer adaption.**
- **Reorder network traffic to avoid resource conflicts.**
  Zero byte messages are low-cost, and may be used e.g. as ready-to-send and ready-to-receive flags.

How do I control SCI and local shared memory usage?

<u>Forcing the size parameters to mpimon is usually not required!</u>
Usually done to make MPI unsafe program run or improve performance.

- **The buffer space required by a communication channel is approximately:**

```
bufferspace = (2 * channel-size         * communicators)
                 + (transporter-size    * transporter-count)
                 + (eager-size          * eager-count)
                 + 512 (give-or-take-a-few-bytes...)
```

  Note that messages up to 560 bytes get in-lined in the channel buffer.

- **The pool-size is a limit for the total amount of shared memory.**
  Default pool-size is set to 32M inter and 4M intra node.

- **A program consists of P processes totally, typically P = nodes * P_intra**
  inter_partition = inter_pool_size / (P_intra*(P-P_intra))
  intra_partition = intra_pool_size / (P_intra * P_intra)

- **The automatic approach is to downsize all buffers associated with a communication channel until it fits in its part of the pool. The chunk size set the size of each individual allocated memory segment. The automatic chunk size is calculated to wrap a complete communication channel.**

Example: Running two processes per node with channel-size 256k - ScaMPI 1.9.1

SCALI
Scalable Linux Systems

- **mpimon -intra_channel_size 256k -intra_pool_size 4m <prog> -- <nodename> 2**
  --- mpimon --- intra_pool_size = 4194304 must be at least 4227072 bytes
  (2113536 bytes * 2 processes) for given set of parameters ---
- **Using the minimum pool size: mpimon -intra_channel_size 256k -intra_pool_size 4227072 <prog> -- <nodename> 2**
  Would start with the following parameters :

-intra_pool_size 4227072	-intra_chunk_size 1M
-intra_eager_count 2	-intra_eager_size 1K
-intra_transporter_count 4	-intra_transporter_size 256

- **A more natural choice of parameters may be: mpimon -intra_channel_size 256k -intra_pool_size 6m <prog> -- <nodename> 2**
  Would start with the following parameters :

-intra_pool_size 6M	-intra_chunk_size 1M
-intra_eager_count 4	-intra_eager_size 64K
-intra_transporter_count 4	-intra_transporter_size 32K

<u>Note:</u> channel_size 256k is an unusual high value.

Asynchronous programming

Improvements with 1.9.x:

- **“Unlimited” buffering of immediate send requests.**
- **Lazy immediate handling as default.** Can be forced to be handled by separate thread through mpimon switch.
- **If communication of medium sized messages are slow - try to set the -inter_eager_size 0**

- **Typical example:**

```
for(a_long_long_time) {
  <Prepare for exchange>
  for(all_cooperating_nodes) {
    MPI_Isend();
    MPI_Irecv();
  }

      <Serious work>

  for(all_cooperating_nodes) {
    MPI_Waitall();
  }
}
```

Benchmarking

"How to fool the masses when giving performance results on parallel computers", David H. Bailey, Supercomputer, Vol. 8, No. 5, September 1991, pp. 4-7]

- **Improving performance for short runs / The first iteration is slow.** Use “mpimon -init_comm_world” to allocate communication buffers between all pairs of processes. A MPI_Barrier() on MPI_COMM_WORLD may be used before the timing loop to eliminate initial time skew.
- **Caching the application program and/or data files locally in node memory.** If the application / data files are stored on a network server it has to be fetched over the SAN the first time it is accessed. When the application is rerun with the same input-files within a short timeframe, they may be locally cashed. Alternative is to copy the application and input files to /tmp on all nodes...
- **Memory consumption may increase after warm-up** Group operations (MPI_Comm_{create,dup,splitt}()) may involve creating new communication (channel) buffers.

Summing up

Scalable Linux Systems

- **ScaMPI is not MPICH**
- **ScaMPI delivers good scalable performance**
- **Forcing SHM/SCI buffer sizes generally not acquired !**
  May improve performance or make MPI unsafe program run.
- **ScaMPI FAQ: http://www.scali.com/support**