Intel MPI Library

Evan Cohn
Intel

Agenda

Intel MPI Library overview

- Features
- Value Proposition
- Architecture
- Devices and Fabrics
- Cluster Tools Support

Lab Session

Features

MPI-2 specification conformance
- Standardized job startup mechanism (mpiexec)
- Process spawning/attachment (tcp device only)
- One-sided communication
- Extended collective operations
- File I/O

Multiple communication fabrics selectable at runtime

Additional thread-safe libraries at level MPI_THREAD_MULTIPLE

Dynamic or static linking, plus debugging and tracing libraries

Features

IA-32 and Intel® 64 platforms
Intel® compilers 10.1 or later, and GNU* compilers 3.3 or later
Red Hat Enterprise Linux* 4 and 5,
SuSE SLES10 and 11
C, C++, Fortran-77, and Fortran-90 language bindings
Dynamic or static linking, plus debugging and tracing libraries
Affordable SDK package
Freely downloadable and royalty-free RTO package

Current Limitations

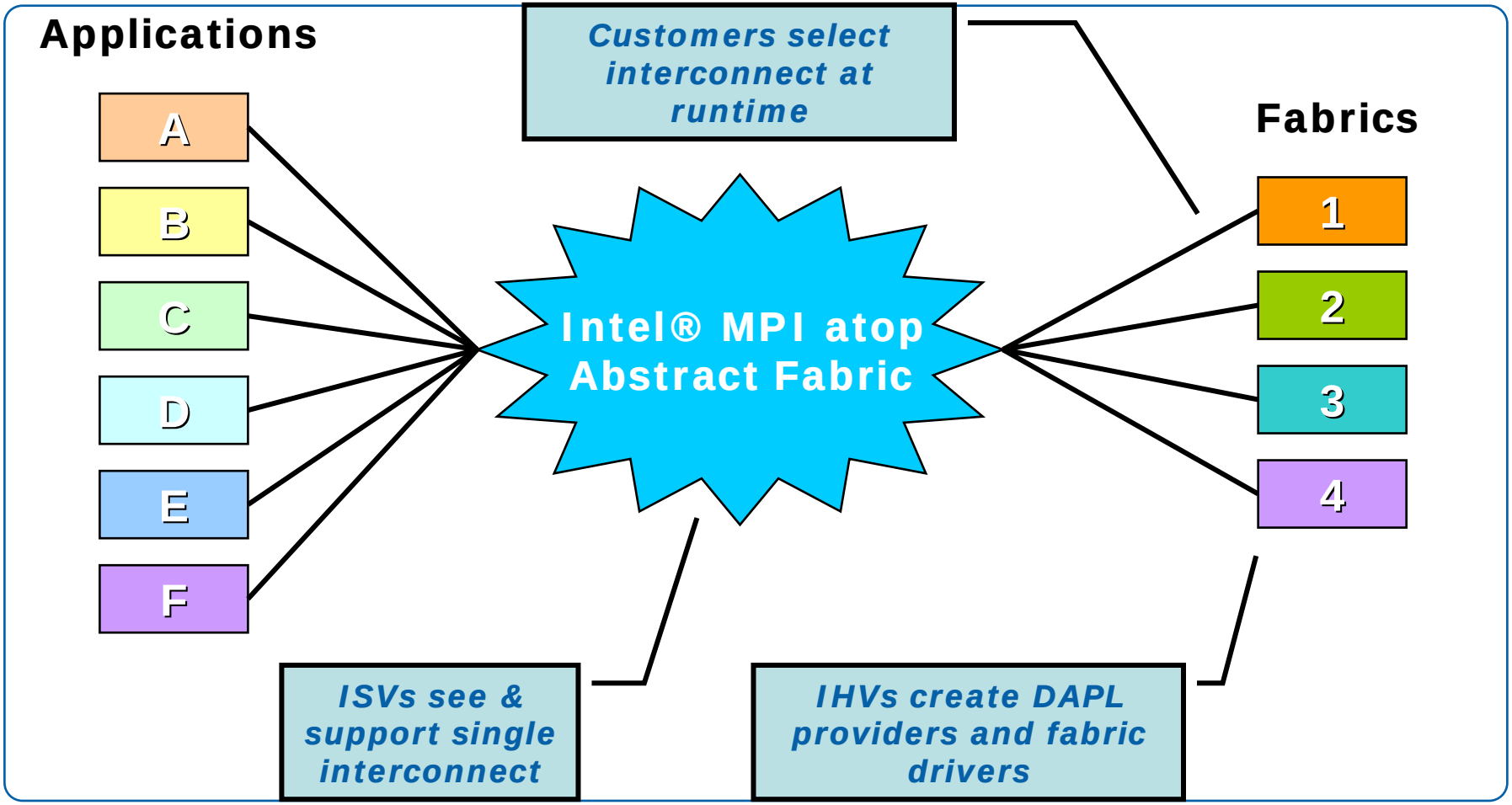
No heterogeneous clusters

No channel bonding

No checkpoint/restart

Experimental fault tolerance

Value Proposition



MPI Devices

Selectable at runtime

- shm (shared memory only)
- tcp (sockets only)
- shm:tcp (shared memory + sockets)
- dapl (DAPL only)
- shm:dapl (shared memory + DAPL, use I_MPI_DAPL_UD=on for connectionless mode)
- shm:tmi (shared memory + tag matching capability of Qlogic and Myricom hardware)
- shm:ofa (shared memory + OFED* verbs, supports multi-rail)

Fault tolerance at startup:

- tcp device used as fallback at initialization step
- Disable fallback for best application performance

Intel MPI Library 4.0

Streamlined product setup

- Installation under root or ordinary user ID
- mpivars.(c)sh scripts for easy path setting

Simplified process management

- mpiexec -perhost and -nolocal options
- mpirun script that automates MPD startup and cleanup
- System-, user-, and session-specific configuration files

Intel MPI Library 4.0 (*contd.*)

Environment variables for runtime control over

- Process pinning
- Optimized collective operations
- Device-specific protocol thresholds
- Collective algorithm thresholds
- Enhanced memory registration cache
- Many others …

Intel MPI Library 4.0 (*contd.*)

Increased interoperability

- Support for DAPL* v1.2 and DAPL* v2.0 compliant providers
- Message queue browsing with TotalView* and DDT* debuggers
- Internal MPI library state tracing with Intel® Trace Collector

Intel MPI Library 4.0 (*contd*.)

Enhanced support for operating systems and compilers

- RedHat EL4, EL5
- SLES10, SLES11
- Fedora* 10, 11
- CAOS 1
- CentOS 5.3
- Microsoft Windows* XP , XP Professional x64, and Vista
- Microsoft Windows* CCS 2003 and Server 2003
- Microsoft Windows* HPC Server 2008 and Server 2008

Getting Started and Reference Manual documents

www.intel.com/go/mpi

Supporting Cluster Tools

Intel® Cluster Toolkit Compiler Edition 4.0

- Intel MPI Library 4.0
- Intel Math Kernel Library 10.2
- Intel Trace Analyzer and Collector 8.0
- Intel MPI Benchmarks 3.2.1

Integration with the job schedulers

- Altair* PBS Pro* 9.2 and higher
- LSF* 6.1 and higher
- Torque* 1.2.0 and higher
- ParallelNavi* NQS* for Linux* OS V2.0L10 and higher
- ParallelNavi for Linux* OS Advanced Edition V1.0L10A and higher
- NetBatch* v6.x and higher
- SLURM* 1.2.21 and higher
- Sun* Grid Engine* 6.1 and higher

Agenda

Intel MPI Library Overview

Lab session

- Download and Installation
- Environment and Commands
- Multiple Purpose Daemons
- Compiling and Running Simple Programs
- Process Placement and Device Selection
- Performance Measurements
- Interaction with other tools

Environment

Important: Get the PATH and LD_LIBRARY_PATH right!
(in particular on x86_64)

Have Python* v2.2 or higher in your PATH

Adjust the environment by sourcing files:

- C-shell: *source /opt/intel/impi/4.0/bin/mpivars.csh*
- Bourne shell: *. /opt/intel/impi/4.0/bin/mpivars.sh*
- Will add Intel MPI Library's *.../bin* and *.../lib* to the paths

Which version on x86_64: 64 bit or 32 bit?

- Source from *.../bin64/* or *.../bin/*
- Will add *.../lib64* or *.../lib* to the LD_LIBRARY_PATH
- CAUTION: Fabric drivers may be only available for 64 bit!

Compile and Link Commands

Using Intel compilers

- mpiicc, mpiicpc, mpiifort, ...

Using Gnu compilers (same underlying Intel MPI Library)

- mpicc, mpicxx, mpif77, ...

Ease of use

- Commands find Intel® MPI include files automatically
- Commands link Intel MPI Library libraries automatically

Commands use compilers from PATH (or selected through options); compilers not hard-wired!

Execution Commands

Model: Running an Intel® MPI program requires a Multiple Purpose Daemon (MPD) per node for the start-up

Starting the "ring" of MPD daemons

- mpdboot [–d] [–v] –n #nodes –f hostfile [–r ssh]
- Flags: –d = debug, –v = verbose, –r = toggle between ssh or rsh

Checking the MPD daemons

- mpdtrace

Running an MPI program

- mpiexec [– l] –n #processes executable
- Flag: –l = prefix output with process sequence number (rank)

Stopping the MPD daemons

- mpdallexit

Execution Commands (*contd.*)

All-inclusive

- mpirun –f hostfile –n #processes executable
- Includes start and stop of an MPD ring
  - Convenient
  - May be too “slow” for interactive work
  - May be good for jobs in batch system
    “In-session” mode: mpirun acquires the list of nodes from the batch system

Which Intel MPI Library version?

- mpiexec –V
- cat $MPI_HOME/4.0.0.025/mpisupport.txt
- rpm –qa | grep intel-mpi (only if MPI installed by root using rpm)

Multiple Purpose Daemons

Used to start and stop Intel® MPI programs

Typical use of daemons in interactive mode:

- “Start once, use many times”
- MPD daemons may run until next re-boot!

Running Intel MPI Library jobs in a Batch System:

- Start (and stop) one MPD ring per batch job
- Overlapping MPD rings from different jobs will kill each other
- Use mpirun, which creates and destroys the MPD ring for you

Multiple Purpose Daemons (*contd.*)

Benefit of MPD daemons:

- Faster start-up of Intel® MPI programs
- Ctrl-C works! All processes get the signal
  Experience: no zombies left on [remote] nodes
- No more hung jobs!
  Job will be terminated according to environment variable MPIEXEC_TIMEOUT at launch

Process Placement

Simple process placement (consecutive assignment of MPI ranks to round robin selection of nodes, see mpdtrace output)

- mpiexec [–perhost #ppn] –n #procs executable
- Place #ppn processes per node until the total number #procs of processes is reached

Exact process placement using Argument Sets:

- mpiexec –n #p1 –host node1 exe1 : –n #p2 –host node2 exe2

- Argument Set (separated by ":") is valid for the specified node:
- Place #p1 processes of exe1 on node1
  Place #p2 processes of exe2 on node2, …
  (usually: exe1 = exe2 = …)
- "exe" may actually be "executable exeparams"

Understand default process placement

Observe default placement

$ *mpiexec –n 4 ./testc*

Hello world: rank 0 of 4 running on node1

Hello world: rank 1 of 4 running on node1

Hello world: rank 2 of 4 running on node1

Hello world: rank 3 of 4 running on node1

Use group round robin placement

Place 2 consecutive ranks on every node using the –perhost option

```
$ mpiexec –perhost 2 –n 4 ./testc
Hello world: rank 0 of 4 running on node1
Hello world: rank 1 of 4 running on node1
Hello world: rank 2 of 4 running on node2
Hello world: rank 3 of 4 running on node2
```

Alternative flag names:

-ppn	processes per node, equivalent to -perhost
-rr	round robin placement (-ppn 1)
-grr	group round robin placement (-ppn)

Exact process placement (Argument Sets)

Place several instances of *testc* exactly on the desired nodes, for example

$ *mpiexec –host node1 –n 1 ./testc : –host node2 –n 2 ./testc*

Hello world: rank 0 of 3 running on node1

Hello world: rank 1 of 3 running on node2

Hello world: rank 2 of 3 running on node2

Note that you can start different executables on different nodes

Process Placement (*contd.*)

Exact process placement with a config file

- One argument set per line in a file (without ":")
- Handy: comment unused lines with "#"
- Example config file:

```
–n #p1 –host node1 exe1
–n #p2 –host node2 exe2
#–n #p3 –host dead_node3 exe3
–n #p4 –host node4 exe4
```

- mpiexec –configfile theconfigfile
- No other *mpiexec* flags on the command line!

Skip Head Node

Avoid running an MPI process on the head node

$ *mpiexec –nolocal –n 2 ./testc*

Hello world: rank 0 of 2 running on node2

Hello world: rank 1 of 2 running on node2

This option is good for Rocks* and OSCAR* controlled clusters that do have a head node

Process Placement (*contd.*)

Hitherto: argument sets contain local options

Place global options for all nodes before all local options in the first argument set

- Example: Set environment variable *I_MPI_DEBUG*

```
mpiexec –genv I_MPI_DEBUG 2 –n #p1 –host node1 exe1 : \
                                –n #p2 –host node2 exe2
```

- Analogue in config file

```
$ cat config
  -genv I_MPI_DEBUG 2
  -host node1 -n 2 ./testc
  -host node2 -n 1 ./testc
```

Intel MPI Library Device: Selection

Environment variable *I_MPI_FABRICS* selects the interconnect device at runtime

I_MPI_FABRICS values:

- *shm* (shared memory only)
- dapl (DAPL fabrics)
- tcp (sockets)
- tmi
- ofa

shm:dapl fabrics is default

Recommendations

- Put *I_MPI_FABRICS* on the *mpiexec* command line
- Add *I_MPI_DEBUG* > 1 to get informed
- mpiexec –genv I_MPI_FABRICS tcp –genv I_MPI_DEBUG 2 –n 2 exe

Select Different Devices

Check selected device

$ *mpiexec –genv I_MPI_DEBUG 2 –n 2 –host node1 ./testc*

….. will use default device shm:dapl (RDMA-enabled device + shared memory)

Change selected device

$ *mpiexec –genv I_MPI_DEBUG 2 –genv I_MPI_FABRICS shm *
–n 2 –host node1 ./testc

….. will use device shm (shared memory only)

Select Different Devices (cont.)

Change selected device in configuration file

Enter environment options at the beginning of first line

```
$ cat > config
–genv I_MPI_DEBUG 2 –genv I_MPI_FABRICS shm:tcp
–host node1 –n 1 ./testc
–host node2 –n 1 ./testc
<ctrl>-D
```

Measuring Performance with IMB

IMB = Intel MPI Library Benchmarks
(improved former PMB = Pallas MPI Benchmarks)

IMB-MPI1 measures performance of different communication patterns:
PingPong, *SendRecv*, *Allreduce*, *Alltoall*, ...

IMB-EXT and IMB-IO measure MPI-2 features (one-sided communication: MPI_Put/Get etc.)

Download it from www.intel.com/software/imb

BKM:Always run full IMB before running an application. Will reveal interconnect issues!

Measuring Performance with IMB (*contd.*)

Making IMB-MPI1: Edit the Makefile and sub-makefile

- Remove or comment *MPI_HOME* and *MPI_INCLUDE* lines (not necessary for Linux)
- Set *CC=mpiicc* or *CC=mpicc*
  (Will be found in *PATH* which was adjusted using *mpivars*.
  Make will create IMB-MPI1 by default—not anymore, use
  `gmake -f make_ict`

Running full IMB-MPI1 [or specific benchmarks]

- mpiexec –n #procs ./IMB-MPI1 [pingpong alltoall …]

Other IMB-MPI1 Options

- –multi 1: Span several process groups (n/2 PingPong pairs)
- –npmin #p: Minimum group contains #p processes

BKM: Read the IMB documentation

Hands-on: Running IMB (cont.)

Run IMB-MPI1 *PingPong* on 1 and 2 nodes for different Intel MPI Library devices

- Compare the bandwidth (Mbytes/sec for large #bytes) e.g. for tcp and dapl
- Compare the latencies (time[usec] for 0 or 1 byte messages) e.g. for tcp and shm:tcp

Run IMB-MPI1 *PingPong* on all nodes with IMB flag *–multi 1*

- mpiexec … ./IMB-MPI1 pingpong –multi 1

Debugging

Compile and link application with *–g*

Application run will output detailed Intel MPI Library information (if I_MPI_DEBUG > 0)

Support of Intel, GNU*, TotalView*, and DDT* debuggers

- mpiexec –idb …
- mpiexec –gdb …
- mpiexec –tv …
- Startup under DDT is managed by itself

Will run all (remote) processes under TotalView*

Performance Tuning: Basics

Check that Intel MPI Library selects correct communication fabric(s)

*I_MPI_DEBUG = 2 (*higher settings will output more and more data)

Play with the fabrics selection if necessary

I_MPI_FABRICS = <intra-node fabric>:<inter-node fabric>

<intra-node fabric> = {shm, dapl, tcp, tmi, ofa}

<inter-node fabric> = {dapl, tcp, tmi, ofa}

where:

shm – shared memory only

dapl – DAPL-capable network

tcp – sockets only

tmi – TMI-capable network

ofa – OFA-capable network

Tuning: Process Placement and Pinning

Use proper process placement, e.g., for the said processors:

I_MPI_PERHOST = allcores

or

mpiexec –perhost <number of processes per node> …

Note: Intel MPI Library will use allcores by default

Use proper process pinning, e.g., for the said processors:

I_MPI_PIN = on

I_MPI_PIN_MODE = lib

I_MPI_PIN_PROCESSOR_LIST = allcores

I_MPI_PIN_PROCESSOR_LIST = 0,2

Note: Intel MPI Library will use this setting by default.

(You can learn more about CPU configuration using the cpuinfo utility)

Performance Tuning: Options

Tune for the given fabric, process number, layout, and pinning:
Select optimized collective algorithms

I_MPI_ADJUST_<collective op> = <algorthmNo>...

for Allgather,...,Bcast,...,Reduce,...

Tune pt2pt communication algorithms, for example:

I_MPI_DYNAMIC_CONNECTION = off (for small jobs)
I_MPI_DAPL_SCALABLE_PROGRESS = on (for large jobs)
I_MPI_EAGER_THRESHOLD = <number of bytes>
I_MPI_SHM_BYPASS = on
I_MPI_SHM_CACHE_BYPASS = on
I_MPI_WAIT = on (for oversubscribed ssm runs)

Tuning Reference in the Reference Manual is your friend!

Performance Tuning: mpitune

Use automatic Intel MPI Library tuning facility to tune Intel MPI Library for your cluster (done once, may take a long time)

mpitune ... (See mpitune –h for options)

Creates options settings which are used with the –tune flag

mpiexec –tune …

Or tune your own application:

mpitune --application \"mpiexec -n 32 ./myprog arg1 arg2 \" -of ./myprog.conf

Use the optimal recorded values for your application

mpiexec -tune ./myprog.conf -n 32 ./myprog

Summary

Intel MPI Library is EASY … If you know it!

Please read the documentation:

- Getting Started
- Reference Manual
- Release Notes
- Intel MPI Library Knowledge Base (see www.intel.com/go/mpi)


intel®
Leap ahead™