Endpoints Proposal Update

Jim Dinan
MPI Forum Hybrid Working Group
June, 2014
Outline

1. Big picture, performance perspective

2. New performance studies
   • Extended journal version of EuroMPI paper
   • Measure process/thread performance tradeoffs
   • Implementation paper submitted to SC
   • Results consistent with claims, will present in Japan

3. Endpoints interface review

4. Recent developments in the proposal
   • Primarily address gaps that arise when MPI processes share a virtual address space
   • Query function, communicator/group comparisons
Today: Threads/Processes Tradeoff

Threads/proc. are entangled, users must make tradeoff
- Benefits of threads to node-level performance/resources
- Versus benefits of processes to communication throughput
Future: MPI Endpoints

Enable threads to achieve process-like communication perf.
- Eliminate negative interference between threads
  - Both semantics (ordering) and mechanics (implementation issues)
- Enable threads to drive independent traffic injection/extraction points

Communication throughput

Reduce memory pressure
Improve compute perf.

More Processes
More Threads
Measure Process/Thread Tradeoffs

“Enabling Communication Concurrency through Flexible MPI Endpoints.” James Dînan, Ryan E. Grant, Pavan Balaji, David Goodell, Douglas Miller, Marc Snir, and Rajeev Thakur. Submitted to IJHPCA.

Look at communication performance trends in many-core system

Identify/measure opportunities for endpoints to improve performance

System setup:
• Intel® Xeon Phi™ 5110P “Knight’s Corner” Coprocessors
  – 60 cores @ 1.053 GHz, 8GB memory, 4-way hyperthreading
• Intel MPI Library v4.1 update 1, Intel C Compiler v13.1.2
  – Run in native mode – Phi cores do MPI processing
• Mellanox 4x QDR InfiniBand, max bandwidth 32 GB/sec

Intended to represent future systems where network is designed to support traffic from many cores
Impact of Increasing Num. Processes

Measure communication performance between two nodes

- OSU benchmark - N senders and N receivers per node
- Performance increases with more processes (P) driving communication

Processes represent “ideal” endpoints

- Private communication state and communication resources
- Represent performance upper bound
“I have N cores, how do I use them?”
- Threads address node level concerns (e.g. memory pressure)
- Processes provide better communication performance

Endpoints will enable threads to behave like processes
- Decouple threads in mechanics – private communication state
- Decouple threads in semantics – isolate in message ordering
Threads/Proc. Tradeoff (BW)

Thread/processes tradeoff impacts bandwidth
Saturate for 32KiB+ messages
More processes = better throughout for smaller messages
**MPI Endpoints**
Relax the 1-to-1 mapping of ranks to threads/processes

![Diagram of MPI Endpoints]

1. **Parent Comm**
2. **E.P. Comm**
3. **Parent MPI Process**
4. **Rank**
5. **P**
6. **T**

**Parent Comm**
- 0
  - 0
  - 1
  - 2

**Parent MPI Process**
- 1
  - 3
  - 4

**Parent MPI Process**
- 2
  - 5
  - 6

*Other brands and names are the property of their respective owners.*
MPI Endpoints Semantics

Parent Comm

E.P. Comm

Parent MPI Process

Rank

Parent MPI Process

Rank

Parent MPI Process

Rank

P

T

T

P

T

T

P

T

T

MPI Process: Set of resources supporting execution of MPI comm.s
- MPI rank and execution resources to drive it when needed
- Endpoints have MPI process semantics (e.g. progress, matching, ...)
  - Collectives are called concurrently on all endpoints (MPI processes)

Improve programmability of MPI + Threads
- Allow threads to be MPI processes, addressable through MPI
- Make number of VA spaces free parameter for apps

Enable threads to act like processes / have process-like performance
- Per-thread communication state/resources, process-like performance
**MPI Endpoints API**

MPI_Comm_create_endpoints

```c
(MPI_Comm parent_comm, int my_num_ep,
MPI_Info info, MPI_Comm &out_comm_handles[])
```

Creates new MPI ranks from existing ranks in parent comm.
- Each process in parent comm. requests a number of endpoints

Outputs handles correspond to different ranks in the same comm.
- Takes TLS out of the implementation and off the critical path

Can return MPI_ERR_ENDPOINTS if endpoints could not be created
```c
int main(int argc, char **argv) {
    int world_rank, tl;
    int max_threads = omp_get_max_threads();
    MPI_Comm ep_comm[max_threads];

    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &tl);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    #pragma omp parallel
    {
        int nt = omp_get_num_threads();
        int tn = omp_get_thread_num();
        int ep_rank;
        #pragma omp master
        {
            MPI_Comm_create_endpoints(MPI_COMM_WORLD, nt, MPI_INFO_NULL, ep_comm);
        }
        #pragma omp barrier
        MPI_Comm_rank(ep_comm[tn], &ep_rank);
        ... // Do work based on 'ep_rank'
        MPI_Allreduce(..., ep_comm[tn]);

        MPI_Comm_free(&ep_comm[tn]);
    }
    MPI_Finalize();
}
Recent Developments

1. MPI_Comm_free() semantics
2. Query function
3. Communicator comparison
4. Group comparison
MPI_Comm_free() with Endpoints

*Past*: Called in series on endpoints in a hosting MPI_COMM_WORLD process

- Enable endpoints communicators to be freed when using thread level < MPI_THREAD_MULTIPLE
- Changes MPI_Comm_free semantics, would have to always do a hierarchical free algorithm on endpoints even when using MPI_THREAD_MULTIPLE
- Breaks collective semantic that is expected by attribute callbacks

*New approach*: Leave this undefined for now, can define MPI_Comm_free_endpoints in future

- Would need to forbid collective attribute callbacks
Query Function: What to query?

Find out if processes share an address space
  • Libraries need to determine whether memory and other resources are private to an MPI rank
  • Existing issue when MPI processes impl. as threads
  • Also an issue with MPI endpoints
  • Pursuing as a separate ticket (#425)

Query number of endpoints:
  • Split with MPI_COMM_TYPE_ENDPOINTS
  • MPI_Comm_num_endpoints(MPI_Comm comm, int *num_ep, int *ep_id)

Query if any processes in comm. are in same VA space:
  • MPI_[Comm,Group,Win,File, ]_has_sharing(..., int *flag)
Communicator Comparison

When MPI processes share a VA space, it becomes possible for a process to see multiple handles to different ranks in the same communicator.

Currently comm. comparison can result in: MPI_IDENT, MPI_CONGRUENT, MPI_UNEQUAL

Past: Return MPI_IDENT, then check ranks to see if they differ.

New proposal: Return MPI_ALIASED to indicate same object, different ranks.
Group Comparison

out_comm_handles[0] corresponds to calling process’ rank in parent communicator

All other output ranks are new ranks that don’t appear in any other group

Needed for group operations to make sense
- Consider my_num_ep == 1 at all processes
- Comparison with parent group should be CONGRUENT
- MPI_Group_translate_ranks between output and parent communicators should yield same ranks
More Info

Endpoints:
- https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/380

Hybrid Working Group:
- https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Hybrid
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for
use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel
microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding
the specific instruction sets covered by this notice.

Notice revision #20110804