How to Succeed in MPI Without Really Trying

Dr. Jeff Squyres
jsquyres@cisco.com
December 9, 2010

Who am I?

• MPI Architect at Cisco Systems, Inc.
• Co-founder, Open MPI project
  ▪ http://www.open-mpi.org/
• Language Bindings
  Chapter author, MPI-2
• Secretary, MPI-3 Forum
  ▪ http://www.mpi-forum.org/

MPI Geek
Assumptions

- You generally know what MPI is
- Examples:
  - You’ve run an MPI application
  - You’ve written an application that uses MPI
  - You’ve ported an existing application to use MPI
  - You know how to spell MPI

How I think of MPI
MPI is hard

- Race conditions
- Deadlocks
- Performance bottlenecks
- High latency
- Low bandwidth

…oh my!

MPI is easy

- Network agnostic
- Multi-core aware
- Waayy easier than native network APIs
- Simple things are simple
- Continually evolving

…oh my!
Where do you land?

Happy MPI users

MPI = Fun!

MPI saved me 15% off my car insurance!

MPI: will you marry me?
MPI is ___

...a mindset
  - “Think in parallel”

...only a tool
  - It is not your work, your data, or your algorithms

...one of many tools
  - Is MPI the right tool for your application?

Become a happy MPI user

• There are many ways to do this
  - Sometimes it’s not about writing better code

• Think about all the parts involved
  - What is the problem you are trying to solve?
Way #1: Don’t reinvent the wheel

• “Not invented here” syndrome is dumb
  ▪ Did someone else MPI-ize (or otherwise parallelize) a tool that you can use?
  ▪ Visit the library, troll around on Google
  ▪ Become a happy MPI user by avoiding MPI
• Focus on your work
  ▪ Easier to think in serial rather than in parallel

#2: Get help

• Consult local MPI / parallel algorithm experts
  ▪ Parallel programming is hard
  ▪ There are people who can help
  ▪ “Cross functional collaboration” is sexy these days
• Consider: we readily ask for help from librarians, mechanics, doctors, …
  ▪ Why not ask a computer scientist?
Proof that networking is hard

• Open MPI SVN commit r22788
  ▪ Fixes a bug in TCP IPv4 and IPv6 handling
  ▪ Represents months of work
  ▪ 87 line commit message
  ▪ One character change in the code base: 0 to 1

• Users should not need to care about this junk
• This is why MPI (middleware) is good for you

#3: Focus on time to solution

• Which is most important?
  ▪ Individual application execution time
  ▪ Overall time to solution

• Put differently:
  ▪ Can you tolerate 5-10% performance loss if you finish the code a week earlier?

• “Premature optimization is the root of all evil”
  ▪ Donald Knuth (famous computer scientist)
#4: `printf()` is not enough

- For the love of all that is holy, please, *Please, PLEASE* use tools
  - Debuggers
  - Memory checkers
  - Profilers
  - Deadlock checkers
- Spending a day to learn how to use a tool can save you (at least) a week of debugging

Non-determinism = BAD

- Print statements (can) introduce non-determinism
  - Changes race condition timings
  - Makes Heisenbugs *incredibly* difficult to find
- Remember: print statements take up bandwidth and CPU resources
#5: Use the right equipment

- Find the right-sized parallel resource
  - Your laptop
  - Your desktop
  - One of Stanford’s clusters
  - Amazon EC2
  - NSF resource
- Look at your requirements

Application requirements

- The Big 4:
  - Memory
  - Disk (IO)
  - Network
  - Processor
- YMMV
  - ...but this is a good place to start
#6: Avoid common MPI mistakes

- Properly setup your parallel environment
  - Test those shell startup / module files
  - Make sure the basics work
  - Example: try “mpirun hostname” (with OMPI)
  - Example: try “mpirun ring”
  - …and so on
- Ensure PATH and LD_LIBRARY_PATH settings are right and consistent

#6: Avoid common MPI mistakes

- Don’t orphan MPI requests
  - Use MPI_TEST and MPI_WAIT
  - Don’t assume they have completed
- Great way to leak resources and memory

MPI_Isend(..., req)
...
MPI_Recv(...)
  // Ah ha! I know the send has completed.
  // But don’t forget to complete it anyway
MPI_Wait(req, ...)
#6: Avoid common MPI mistakes

- Avoid MPI_PROBE when possible
  - Usually forces an extra memory copy
- Try pre-posting MPI_IRECVs instead
  - No additional copy

```c
while (...) {
    MPI_Probe(...);
}

// Pre-post receives
MPI_Irecv(...)
while (...) {
    MPI_Test(...)
}
```

- Avoid mixing compiler suites when possible
  - Compile all middleware and your app with a single compiler suite
- Vendors do provide (some) inter-compiler compatibility
  - But it’s usually easier to avoid this
#6: Avoid common MPI mistakes

- Don’t assume MPI implementation quirks
  - Check what the MPI spec really says
- True quotes I’ve heard from users:
  - “MPI collectives always synchronize”
  - “MPI_BARRIER completes sends”
  - “MPI_RECV isn’t always necessary”

  (my personal favorite)

#6: Avoid common MPI mistakes

- Don’t blame MPI for application errors
- Your application is huge and complex
  - Try to replicate the problem in a small example
- Not to be a jerk, but it’s usually an application bug…
#6: Avoid common MPI mistakes

- Don’t (re-)use a buffer before it’s ready
  - Completion is not guaranteed until MPI_TEST or MPI_WAIT
  - Do not read / modify before completion!
    ```
    MPI_Isend(buf, ...);
    buf[3] = 10;
    // BAD!
    
    MPI_Irecv(buf, ...);
    MPI_Barrier(...);
    MPI_Wait(..., req);
    A = buf[3];
    // Bad!
    ```

- Do not mix MPI implementations
  - Compile with MPI ABC
  - Run with MPI XYZ

- Do not mix MPI implementation versions
  - Sometimes it may work
  - …sometimes it may not

- Be absolutely sure of your environment settings
#6: Avoid common MPI mistakes

- Avoid MPI\_ANY\_SOURCE when possible
  - Similar to MPI\_PROBE
- MPI\_ANY\_SOURCE can disable some internal optimizations
- Instead, pre-post receives
  - …when there are only a few possible peers
  - Ok to use MPI\_ANY\_SOURCE when many possible peers

- Double check for unintended serialization
  - Using non-blocking sends and receives
  - But there’s an accidental “domino effect”
  - Example: process X cannot continue until it receives from process (X-1)
- Message analyzer tools make this effect obvious
#6: Avoid common MPI mistakes

- Do not assume MPI\_SEND behavior
  - It may block
  - It may not block
- Completion ≠ the receiver has the data
  - Only means you can re-use the buffer
- Every implementation is different
  - *Never assume any of the above behaviors*

#7: Use MPI collectives

- MPI implementations use the fruits of 15+ years of collective algorithm research
  - [Almost] Always better than application-provided versions
- This was not always true
  - Collectives in mid-90’s were terrible
  - They’re much better now
  - Go audit your code
- Still an active research field
#7: Use MPI collectives

Application-provided broadcast

Implementation-provided broadcast

#8: Location, location, location!

- It’s all about the NUMA
  - Memory is distributed around the server
  - Avoid remote memory!
- Don’t forget the internal network (!)
  - It now matters 😞
#8: Location, location, location!

- Paired with MPI
  - For off-node communication / data access
  - (At least) 2 levels of networking
- “NUNA”
  - Non-uniform network access

- Design your algorithms to exploit data locality
  - Use what is local first
  - Use non-blocking for remote access
#8: Location, location, location!

- Shameless plug: Hardware Locality Toolkit (hwloc)
- Provides local topology via CLI and a C API

#9: Do not use MPI one-sided

- Some application (greatly) benefit from a one-sided communication model
  - MPI-2’s one-sided is terrible
  - It is being revamped in MPI-3
- Avoid MPI one-sided for now
#10: Talk to your vendors

- Server, network, MPI, ...
  - Tell us what’s good
  - Tell us what’s bad
  - Then tell us what’s good again 😊
- We don’t know you’re having a problem unless you tell us!
  - Particularly true for open source software

#11: Design for parallel

- Retro-fitting parallelism can be Bad
  - Can make spaghetti code
  - Can lead to questionable speedups
  - Think about parallelism from the very beginning
- Not everything parallelizes well
  - Might need a lot of synchronization
  - Might need to talk to all peers every iteration
  - Try solving the problem a different way
  - …or buy a Cray 😊
But… what about portability?

• I didn’t really mention “portability” at all
• Here’s the secret:
  ▪ If you do what I said, your code will be as portable as possible
  ▪ Write modular code for the non-portable sections

Moral(s) of the story

• It’s a complex world
  ▪ But MPI (and parallelism) can be your friend
• Focus on the problem you’re trying to solve
• Design for parallelism
  ▪ Design for change over time
Additional resources

• MPI Forum web site
  ▪ The *only* site for the official MPI standards

• NCSA MPI basic and intermediate tutorials
  ▪ Requires a free account

• “MPI Mechanic” magazine columns

Questions?