SAND2020-1959PE

Application Development and Readiness for Sierra Correctness, Deadlocks, Productivity

James Elliott

jjellio@sandia.gov

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
(SAND2018-12904 C)

Outline

Challenges when moving from traditional computing platforms to Sierra

- A Cautionary Tale of R&D and Preparing for Sierra
- Development
  - Memory types and what not to do
  - Understanding latencies
- Testing
  - Changes and additions to a multiplatform CI framework
- My code deadlocks… but only on machine _______ …

  , Or dealing with semantics that the MPI Standard refuses to acknowledge.

A Cautionary Tale of R&D and Preparing for Sierra

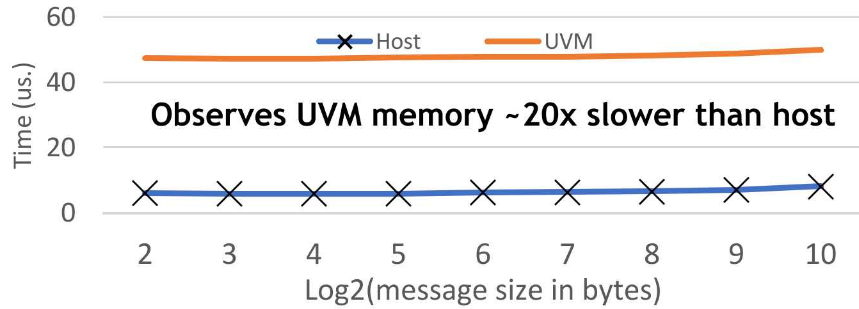
1

User notices app is slow in CG norms…
Allreduce is taking longer than **SpMV+Axpy combined!**

2

Runs on early access testbed

Runs with **MPI impl. A**

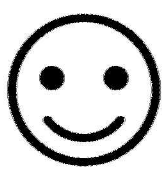


3

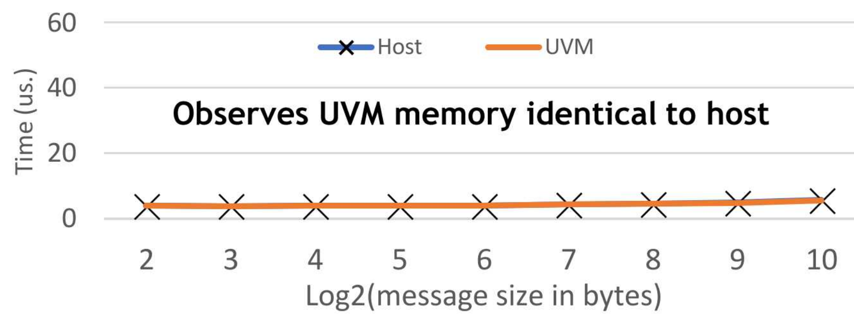
Developer eradicates use of UVM from all MPI calls, adding code to explicitly copy data to buffers of another type (E.g., host or device memory)
An expensive R&D effort!

4

A year or more passes and … Runs on new testbed with official vendor software



Runs with **MPI impl. B**



CCR
Center for Computing Research

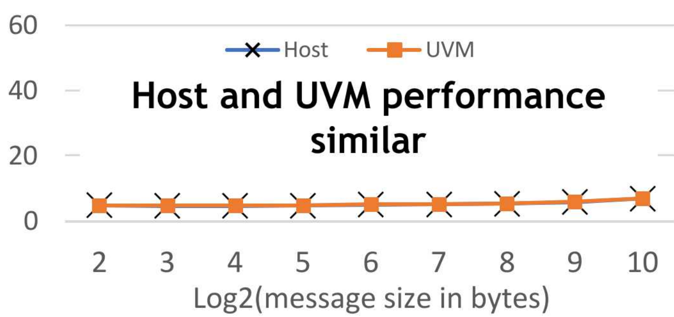
A Cautionary Tale of R&D and Preparing for Sierra

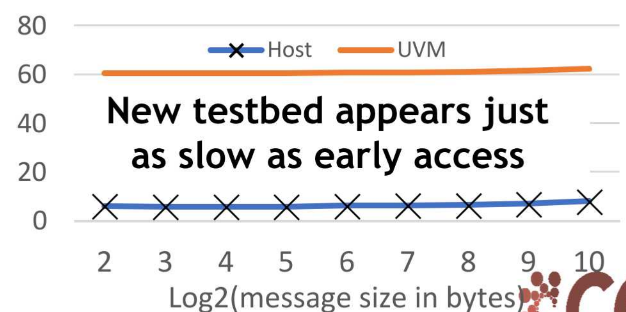
- Users frustrated that performance extremely different between testbeds.
  - Software stack should not be drastically different
  - Hardware is nearly identical
- Expert explores MPI performance on both machines
  - Device-aware MPI is available on both machines, but the **early access machine enables it by default, while the other disables it by default.**
  - Different vendors may set different defaults
  - Because an implementation supports device-aware MPI, does not mean it is necessarily enabled

Moral of the story:

Device-aware MPI likely **a runtime option.**
Users must now be aware of additional MPI settings that vary by vendor.

Expert user reruns, carefully ensuring each MPI impl. enables the same features





CCR
Center for Computing Research

A Cautionary Tale of R&D and Preparing for Sierra

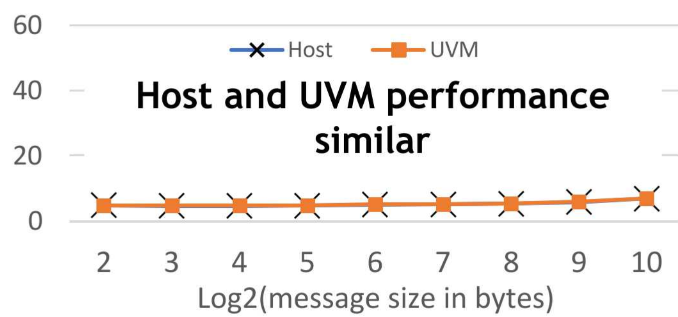
- Users frustrated that performance extremely different between testbeds.
  - Software stack should not be drastically different

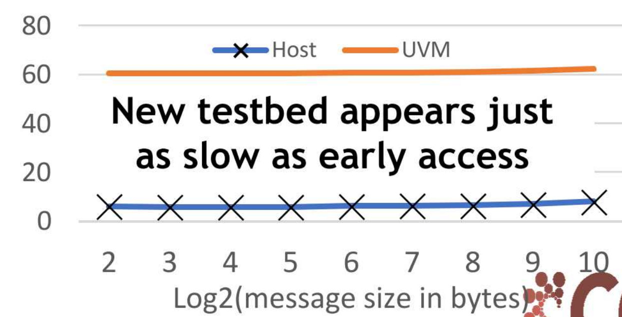
Focus of talk <u>not on performance with device-aware MPI</u>

This talk focuses on caveats and gotchas that can lead to headaches and developer frustration

that vary by vendor.

Expert user reruns, carefully ensuring each MPI impl. enables the same features





CCR
Center for Computing Research

MPI_Send(**void***, int, MPI_Datatype, int, int, MPI_Comm)

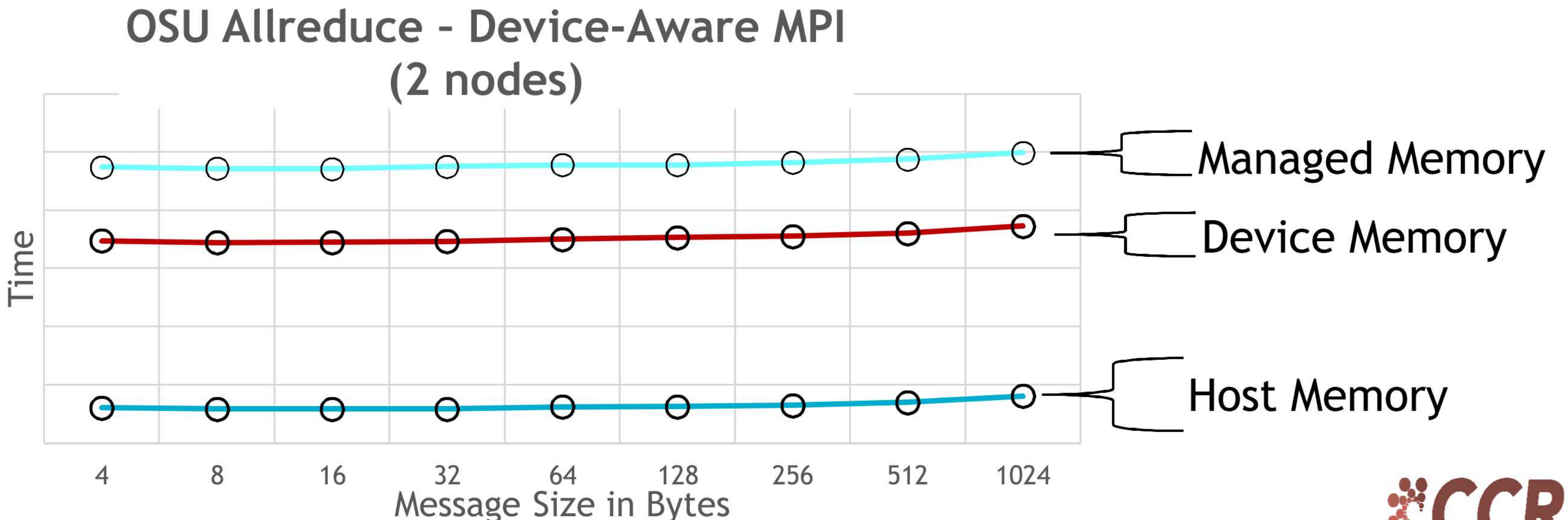
- MPI agnostic to memory location (device, host, or managed)
  - Users may see drastically different performance between different buffer locations
  - **Performance can change drastically as the software stack matures!**

DO: Design code so that memory type can change

DO: Understand MPI implementation **default behavior** and **runtime options**

DO: Expect performance to improve for different buffer types

DON'T: Early optimize based on MPI performance



Development

Latencies in GPU-centric apps

- GPUs offer extremely good data parallelism … but there is a cost to starting a GPU kernel
- Cost to synchronize / fence around a kernel launch ~ 30-40 us.
- If algorithm needs to perform communication between computation kernels
  - GPU kernel launch latency can inhibit performance.
  - Difficult to decide when to leave data on device or forego GPU use in favor of host execution
  - No clear answer to this… Kernel fusion can help, possibly cuda-graphs (but that functionality may not be portable)

Testing

Device aware-MPI introduces Compile and Runtime challenges

- Software (Trilinos) uses compile time checks to determine if MPI is device-aware (can device buffers be passed to MPI)
- Reality: Device-aware MPI is a runtime option for many MPI implementations.
- Tpetra (distributed linear algebra package) has different code paths for traditional and device-aware MPI
  - Algorithms need to pack, transfer, receive, and unpack data – entails copying device buffers to host if device-aware MPI is not supported (i.e., older platforms the library must support)
- Unit testing needs to exercise both code paths, which requires two passes through the test suite (toggling a runtime flag to MPI and changing Tpetra's expected behavior via an ENV variable)
  - Not extremely complicated to implement, but requires understanding how to enable/disable MPI features

CCR Center for Computing Research

My code deadlocks… but only on machine ________

Device-aware MPI semantics are fuzzy!

MPI Standard provides no mechanism for querying support for devices or buffer types

Example:

- App runs correctly on most machines, but deadlocks on one.
- MPI appears to be putting garbage into buffers
  - Valgrind, etc… report no issues with the code …

- Substantial R&D effort tries to debug the application and the machine
- Root cause: Managed Cuda memory (UVM) requires device-aware MPI or it silently produces bogus data.
- <u>No tools existed to report whether the app was using MPI appropriately</u>

My code deadlocks… but only on machine ________

- Developed MPI Profiler (PMPI based) that tests all buffers passed to all MPI functions and report buffer locations.

- Using tool,
  - Identified 6 call sites in one kernel of the app that passed UVM buffers to MPI.
  - Identified one callsite in a Trilinos package that inadvertently passed a UVM buffer to MPI
  - Different MPI implementations may have different semantics - **a headache**

- Corrected issue and code behaved as expected.

Moral of the story:

- Tools for asserting Device-MPI correct usage not existent.
- Understand the rules for your MPI implementation
- Understand how your app uses MPI
  (perhaps we can share our profiler)

CCR
Center for Computing Research

Conclusion

- Moving to Sierra has presented several challenges
  - Device-aware MPI presents introduces runtime options that change MPI performance and semantics
  - Device-aware MPI requires developers to be aware of the location of buffer types passed to MPI
  - Buffer location can impact performance, but vendors can optimize performance negating or reversing premature optimizations
  - Kernel launches / memory fencing can be expensive
  - Unit testing needs to exercise all code paths, which requires understanding how to enable/disable MPI features.
  - Not understanding device-aware semantics can lead to difficult debugging

Questions and Comments: James Elliott <jjellio@sandia.gov>

CCR Center for Computing Research