MPI Communication Performance of a Shock Hydrodynamics Application

Ryan Goodner
Prof. Patrick G. Bridges
UNM Computer Science

CUP ECS
Center for Understandable, Performant Exascale Communication Systems
THE UNIVERSITY OF NEW MEXICO

Background

FIESTA (Fast InterfacES and Transport in the Atmosphere) is a UNM code by Brian Romero for modeling shock hydrodynamics. It is a **MPI** based, **GPU** accelerated code written in **C++** with **Kokkos**. The code was recently reviewed and revised by Prof. Bridges to investigate potential performance improvements related to MPI communications.

Code changes:

1. **Waitall** pattern was changed to require one instead of three waits per unit of computation.
2. Refactored original code in an object-oriented way to enable **three distinct MPI patterns** to be selected at run-time.

Three MPI Patterns:

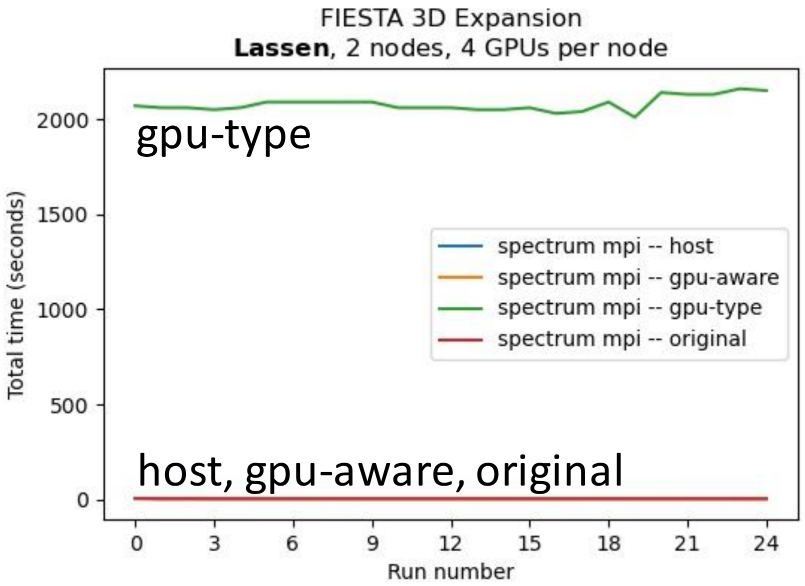
1. **Host**
   i. Pack data into contiguous memory on GPU
   ii. Copy data from GPU memory to CPU memory
   iii. Send
2. **GPU-Aware**
   i. Pack data into contiguous memory on GPU
   ii. GPU aware send
3. **GPU-Type**
   i. GPU aware send using MPI types

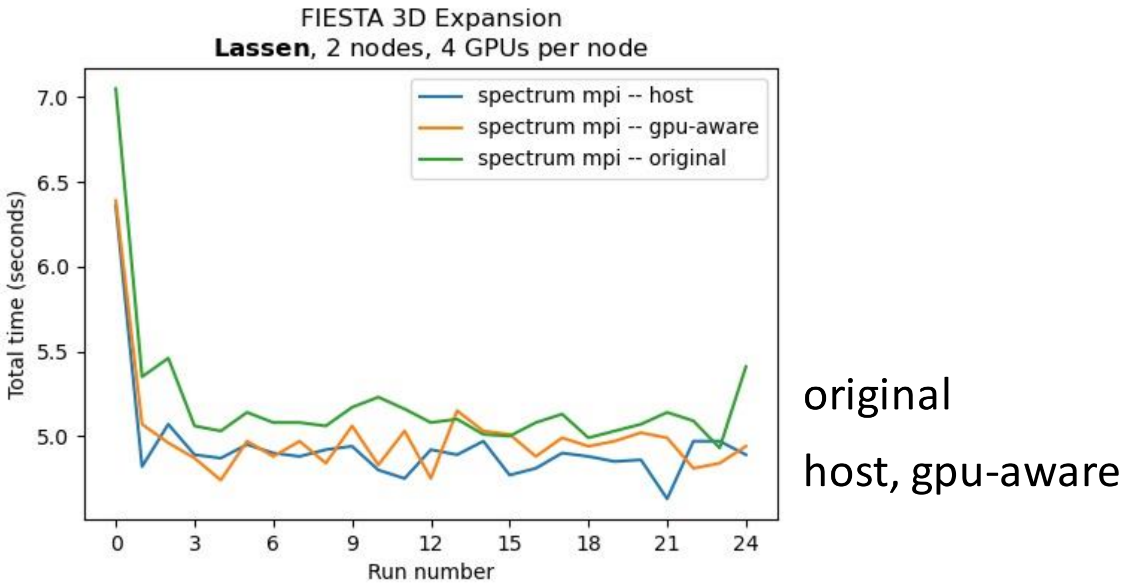
Which tools and techniques can be used to measure and observe MPI communication performance?

1. Measuring run times
2. Profiling (passive and instrumented)
3. Tracing

Measuring run times

Plotting of run times gathered via application output is a good first step to identify overall performance. These plots were created from the total run times reported in the application output.



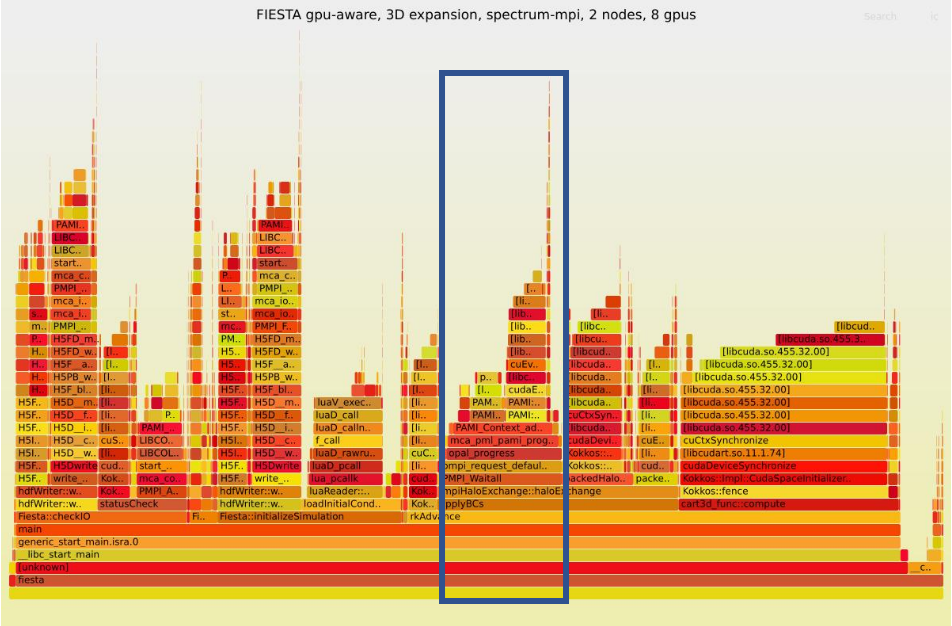


From these plots we find that there is a large performance penalty (~**400x**) to using the gpu-type pattern. We can also see that host and gpu-aware have similar performance, which are also both slightly faster than the original code.

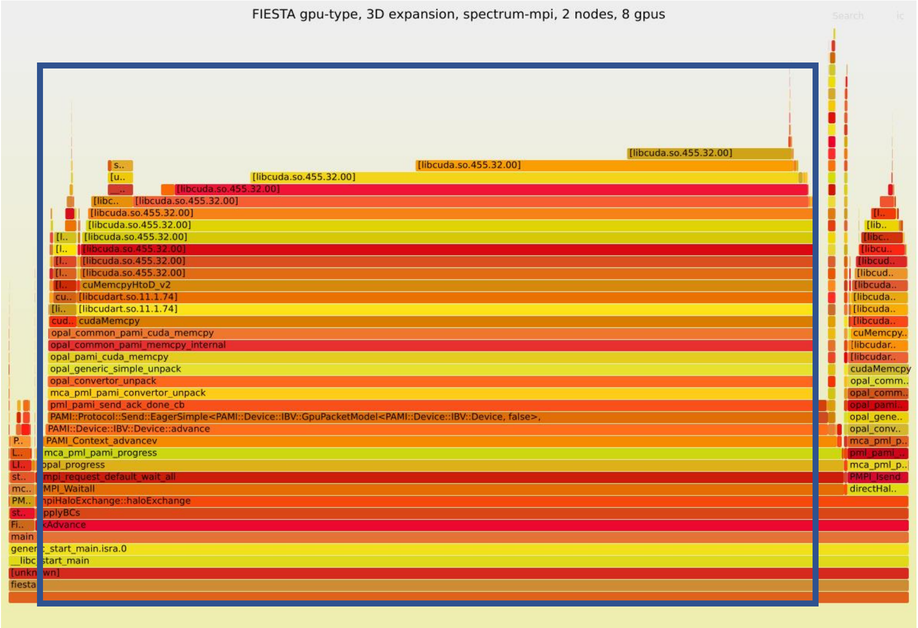
Profiling (passive)

Profiling is a good next step to identify where in the application behavior changes have occurred and identify potential targets for optimization.

gpu-aware



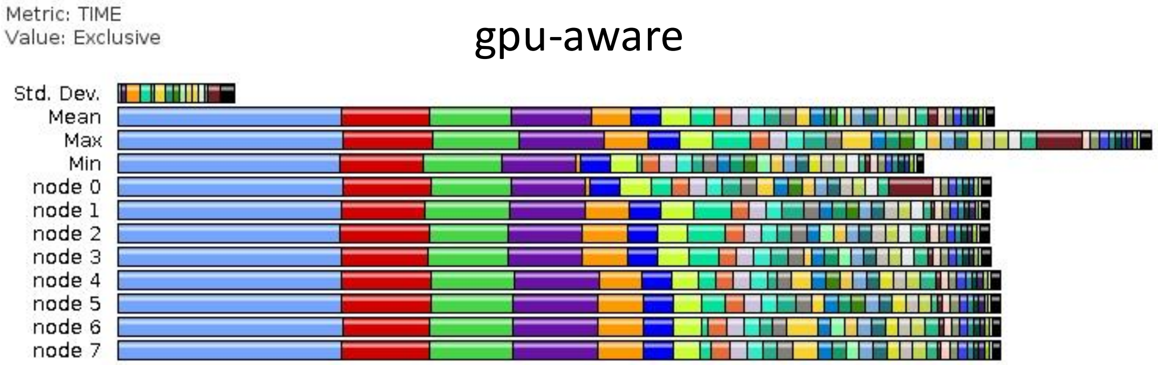
gpu-type

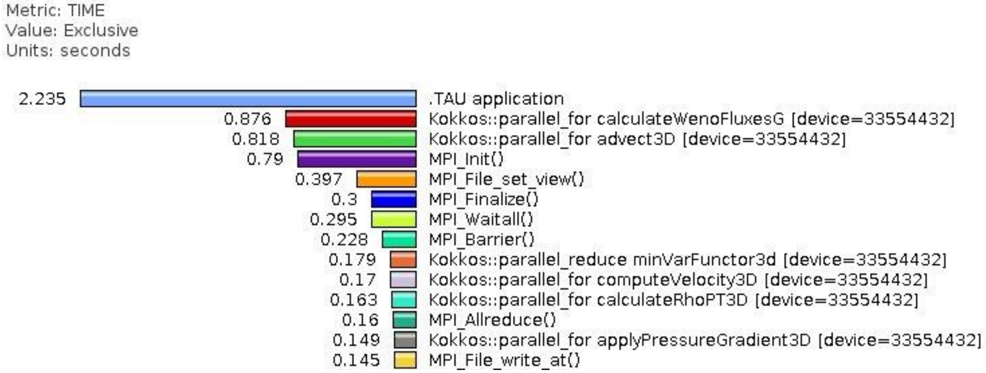


The perf events flame graphs show how the gpu-type code is dominated by **PMPI_Waitall** and **cudaMemcpy**, while the gpu-aware code spends comparatively little time in these same sections.

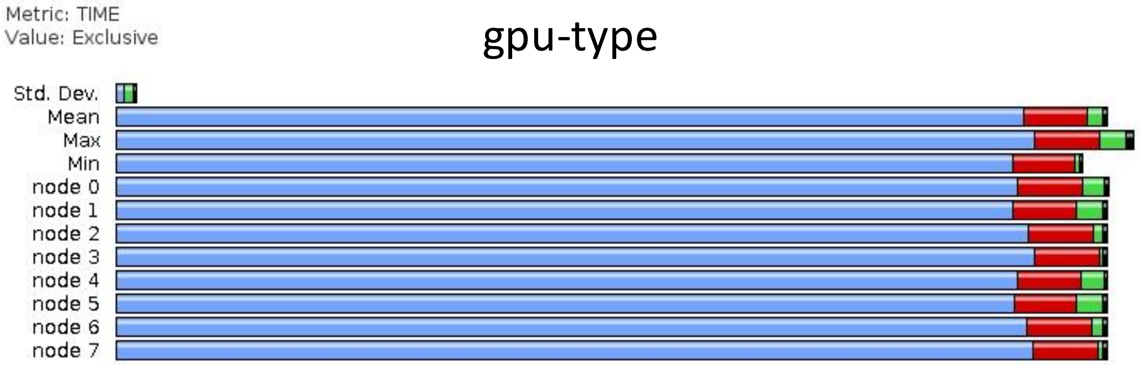
Profiling (instrumented)

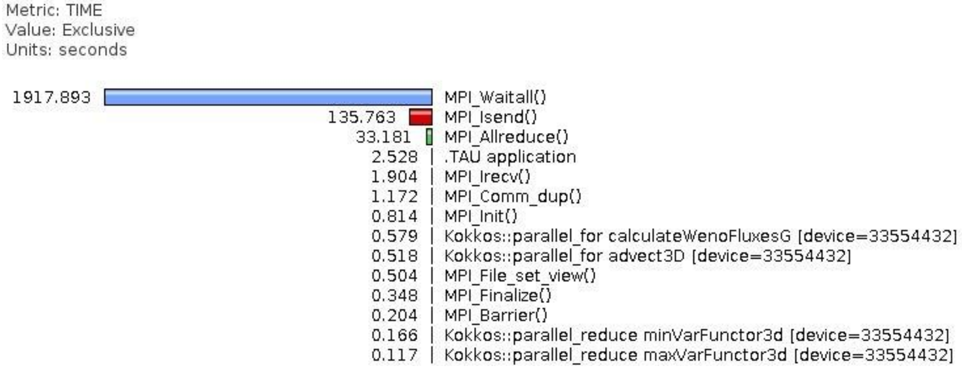
gpu-aware





gpu-type



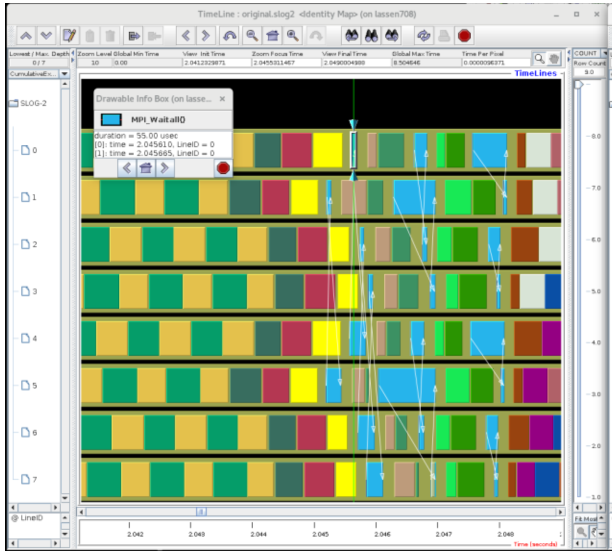


The TAU bar graphs show how the gpu-type code is dominated by **MPI_Waitall**, while the gpu-aware code spends a larger portion of it's time doing calculations.

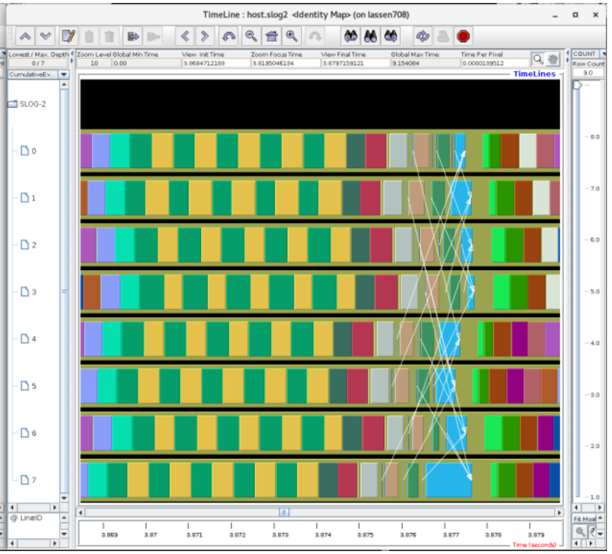
Tracing

MPI tracing is another method to dig deeper into specifics of communication behavior. Jumpshot is a powerful tool for visualizing MPI tracing data.

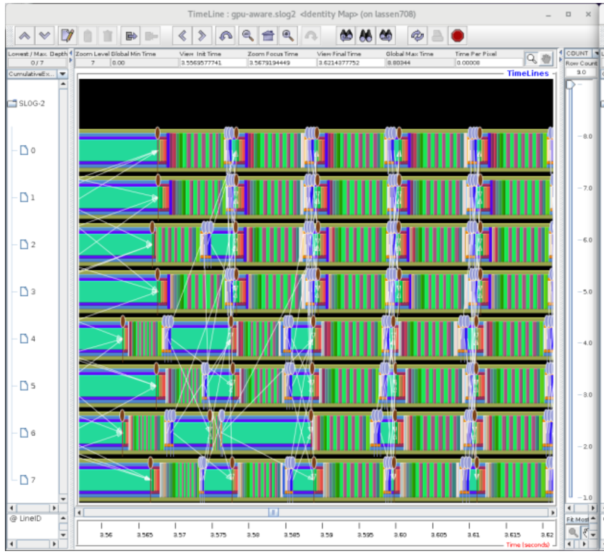
original



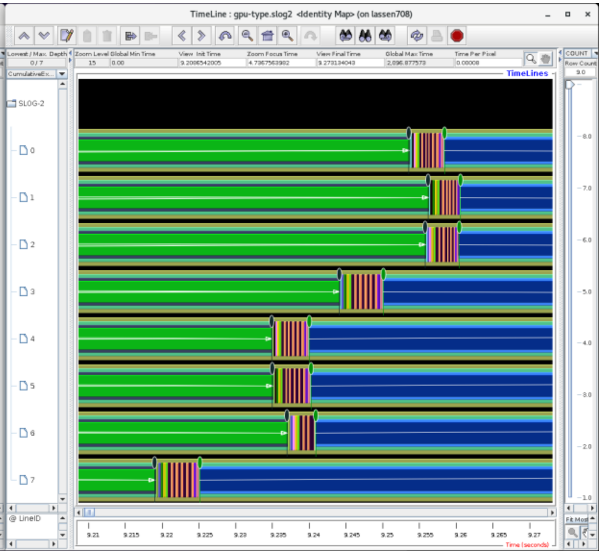
host



gpu-aware



gpu-type



The reduction from **three waits** per unit of computation to **one wait** can be easily seen here with Jumpshot

Using the same time scale shows that gpu-aware is completing computation faster than gpu-type b/c gpu-type spends much more time in **MPI_Waitall** and **MPI_Isend**.

References & Acknowledgements

References

- Using LC's Sierra Systems, https://hpc.llnl.gov/training/tutorials/using-lcs-sierra-system
- Flame Graphs and Linux Perf Events, https://www.brendangregg.com/flamegraphs.htm
- TAU, https://www.cs.uoregon.edu/research/tau/home.php
- Jumpshot, https://www.mcs.anl.gov/research/projects/perfvis/software/viewers/jumpshot-4/usersguide.html

Acknowledgements

- Prof. Amanda Bienz – For helping me learn how to use HPC clusters and debugging builds
- Brian Romero – Author of FIESTA
- This work was [partially] supported by the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program (PSAAP-III), Award #DE-NA0003966