



Interoperability Across GPU Languages and Platforms

Priscilla Kelly^{1,2}

Robert Robey² (Advisor)

¹San Diego State University, San Diego, CA 92182

²Los Alamos National Laboratory, Los Alamos, NM 87545



Abstract

As researchers look towards exascale computing for large scientific applications, the portability and composability of GPU parallelism becomes a vital endeavor. CUDA and OpenCL are specialized, low-level programming of GPU hardware, while OpenACC and OpenMP4 are higher-level languages tailored for portability, as details are left to the compiler. How these methodologies perform together is the focus of this research. We assess current interoperability and the performance of a large-scale grid running the Game of Life on GPUs.

Background

Modern scientific applications are composed of multi-physics packages and libraries that were developed independently and can use different programming languages and/or models. Joining these packages can lead performance issues on top of barriers to interoperability. When utilizing GPUs, interoperability is particularly problematic due to how GPU languages interact with the hardware.

Currently there are two methods of programming GPUs in scientific applications:

Low(System)-Level

- The management of memory on/off the device and division of work on GPU must be done by the user.
- **Pros:** Kernels can be specialized to particular GPU features.
- **Cons:** Code becomes difficult to maintain across platforms.
- Examples: CUDA and OpenCL

High-Level

- The user introduces directives around portions of code the compiler needs to parallelize on the GPU.
- **Pros:** User friendly and designed for portability
- **Cons:** Compiler isn't always reliable.
- Examples: OpenACC and OpenMP4.X

Conway's Game Of Life

Conway's Game Of Life^{1,2} uses an 8-point stencil, which is a common motif among physics algorithms, to generate the next iteration. To test interoperability among GPU languages, packages exchange buffers on the GPU and apply the rules of the game.

Game of Life Rules:

1. Under-population: Any live cell with less than 2 live neighbors dies
2. Longevity: Any live cell with 2 or 3 live neighbors lives on
3. Over-population: Any live cell with >3 live neighbors dies
4. Reproduction: Any dead cell with 3 live neighbors becomes alive

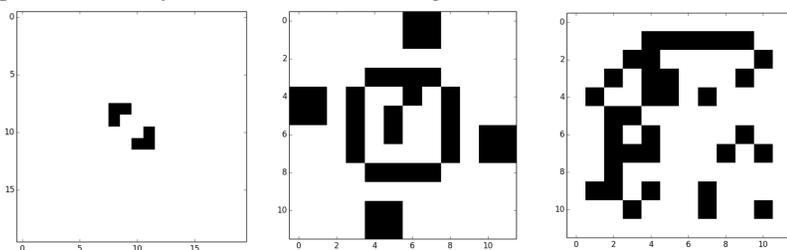


Figure 1: Possible starting patterns for the Game of Life

A random seed, such as Fig. 1, was used to initialize the experiments as the grid scaled from 10^2 to 10^6 elements.

Interoperability Results

Interoperability between GPU languages is defined as the ability for each language to affect data on the GPU in an alternating pattern only by the use of its buffer. As shown in Figure 2, grids are allocated on the GPU and the package will apply the rules for 10 generations.

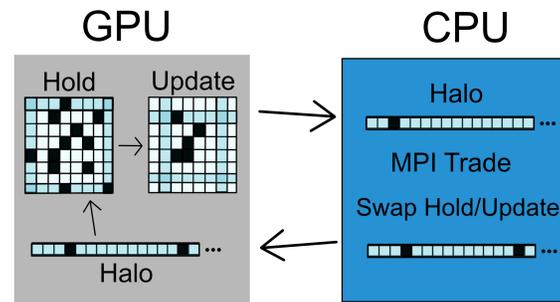


Figure 2: Each loop in the figure is what the package will enact on the CPU and GPU

As Nvidia created CUDA and helped develop OpenACC, these two languages are interoperable using a PGI compiler version greater than 12.6. To test this, two packages were written in C, one in CUDA and OpenACC, which apply the rules. Interoperability is also valid for Fortran with CUDA+OpenACC³.

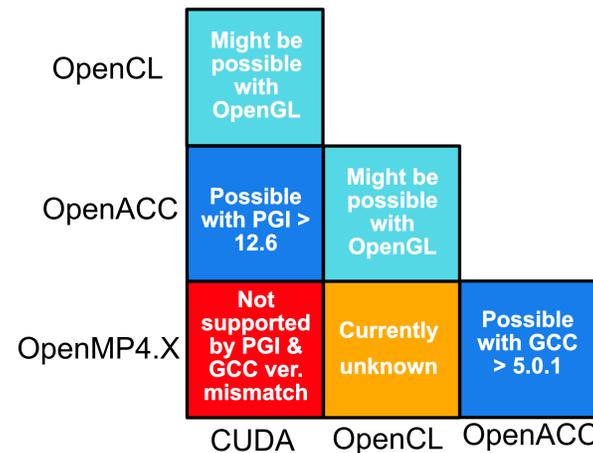


Figure 3: This is our current interoperability among the four major GPU languages

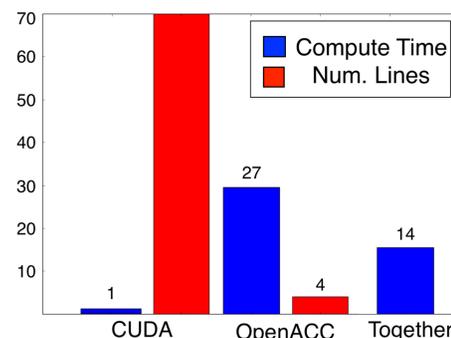


Figure 4: Lines of serial code versus total run time for 6.4×10^5 elements

Productivity vs. Performance

Figure 4 shows a comparison of development productivity to performance. A fully integrated CUDA version of the test code is 27 times faster than pure OpenACC, but requires 70 lines of code – 18 of which are just to generate pointers.

This only applies to the serial case. For the parallel case, there is additional code required to put and pull halo values for MPI.

Performance Results

The timings were run on two types of GPUs: Nvidia's Tesla K40m (on Intel Xeon E5-2660 @ 2.20 GHz processor) and Tesla M2090 (Intel Xeon E5-2670 @ 2.60 GHz). The figures below show that CUDA+OpenACC, when CUDA and OpenACC alternate iterations on the GPU, there is no added expense to runtime.

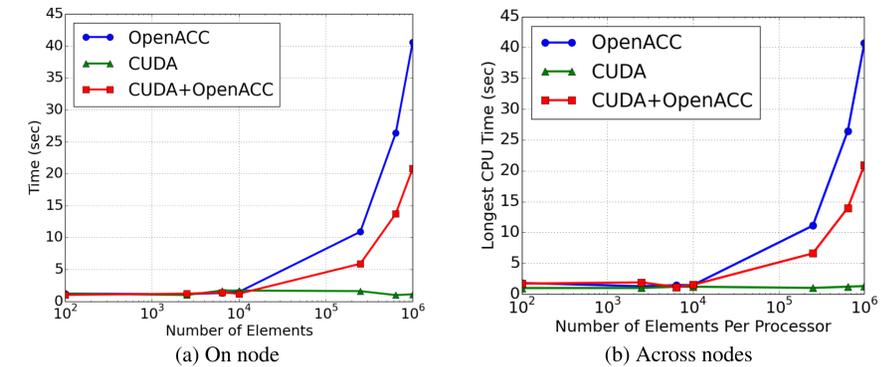


Figure 5: These are the runtime vs. number of elements results for Tesla K40m

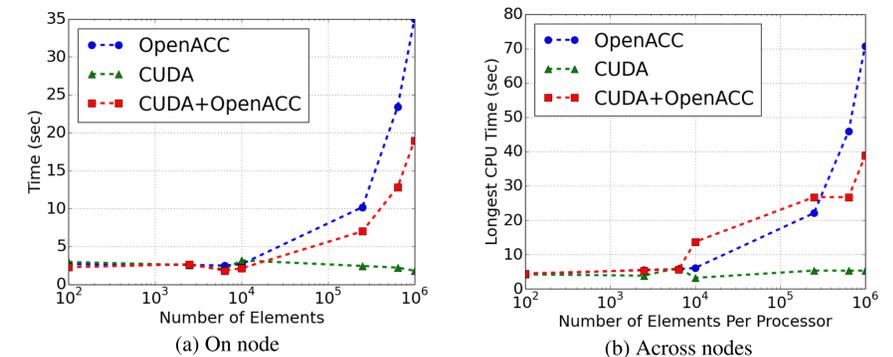


Figure 6: These are the runtime vs. number of elements results for Tesla M2090

Conclusion

With large multi-physics codes, inter-language operability between physics packages, such as ocean and atmospheric models, is necessary to model real-world problems. While the conclusions presented here show that there is a significant benefit of moving to a lower-level language such as CUDA, there is some benefit to including packages where only OpenACC present.

Future Directions: Incorporating OpenGL to pass buffers between CUDA and OpenCL. Use Unified Memory Access (UMA) for multiple MPI ranks to interact with the GPU, and among GPUs, for better performance.

References

- ¹ <https://www.olcf.ornl.gov/tutorials/>
- ² Larkin, Jeff, OpenACC Interoperability Tricks, *PGI Insider* (2013)
- ³ Gardner, Martin, *Scientific American* 223, 120–123. (1970)

Acknowledgments

The authors thank Jeff Larkin, Software Engineer for NVIDIA, for his insight. The authors also thank LANL Institutional Computing Resources to access to their clusters. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1321850. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.