Extreme-scale Computational Fluid Dynamics with AMR on GPUs

Joshua Hoke Davis†, Justin Shafner*, Daniel Nichols†, Nathan Grube*, Pino Martin*, Abhinav Bhatele†

†Department of Computer Science, University of Maryland, College Park, MD 20742 USA
*Department of Aerospace Engineering, University of Maryland, College Park, MD 20742 USA

***Abstract*—Accurate modeling of turbulent hypersonic flows has tremendous scientific and commercial value, and applies to atmospheric flight, supersonic combustion, materials discovery and climate prediction. In this paper, we describe our experiences in extending the capabilities of and modernizing CRoCCo, an MPI-based, CPU-only compressible computational fluid dynamics code. We extend CRoCCo to support block-structured adaptive mesh refinement using a highly-scalable AMR library, AMReX, and add support for a fully curvilinear solver. We also port the computational kernels in CRoCCo to GPUs to enable scaling on modern exascale systems. We present our techniques for overcoming performance challenges and evaluate the updated code, CRoCCo v2.0, on the Summit system, demonstrating a 5× to 24× speedup over the CPU-only version.**

***Index Terms*—hypersonics, computational fluid dynamics, adaptive mesh refinement, GPU computing**

I. Summary

A. CRoCCo Application

The potential for hypersonic flight and commercial space access to enhance life on earth is immeasurable – the growth of new markets, the discovery of new materials, and our ability to monitor climate change are just a few of many possibilities that hypersonic and space flight will bring. Resolving all important spatial and time scales along with highly nonlinear multi-physics interactions, especially at the full vehicle level, goes well beyond the current state-of-the-art.

CRoCCo is a hypersonic flow simulation code with robust shock capturing and high-bandwidth-resolving efficiency that has been fully validated for unsteady, highly-turbulent, high-enthalpy, chemically-reacting hypersonic flows on structured grids. In this poster, we describe the pre-existing structure of CRoCCo, our development process in extending its capabilities to support AMR for a fully curvilinear solver, and our experiences preparing it for acceleration on current and future accelerator-based systems. CRoCCo has been updated to support adaptive mesh refinement using AMReX [1] for a fully curvilinear solver. In addition, we have converted the FORTRAN kernels to C++ to exploit the GPU-offload capabilities in AMReX.

CRoCCo solves the conservative form of the equations governing fluid motion, namely the conservation of the species mass, momentum, and total energy equations. The convective fluxes are solved using a finite-difference, weighted essentially non-oscillatory (WENO) method that has been bandwidth-optimized (WENO-SYMBO), as described by Martín et al. [2]. In WENO-SYMBO, the flux at an interface in the ith-direction, $f_{i+1/2}$, is reconstructed with multiple stencils around the interface, using four grid points in each direction.

B. Curvilinear Support and GPU Porting Method

Adaptive mesh refinement (AMR) allows for computational domains to be locally and dynamically coarsened or refined during a simulation. Block-structured AMR was adapted into CRoCCo using the AMReX library [1] which allowed for near-seamless integration of FORTRAN numerical kernels into a Cartesian AMR framework. CRoCCo's core numerics kernels were originally written and optimized for CPU-only systems in FORTRAN. We decided to convert these FORTRAN kernels to C++ in order to maximize compatibility with our existing AMReX-based C++ code.

Curvilinear grids as they are necessary for gathering HiFi datasets of hypersonic flows around complex geometries. However, most AMR libraries (including AMReX) natively support Cartesian grids and cylindrical coordinate transforms only, so we undertook a large development effort to adapt the AMReX framework itself to handle the demanding needs of a fully curvilinear solver, summarized in three points:

1) Grid metrics: we store the curvilinear grid in an AMReX::MultiFab, rather than performing the expensive on-the-fly computation of first- and second-order grid metrics.
2) Interpolation: we replace the default AMReX trilinear interpolator with a custom interpolator accounting for non-uniform spacing of grid points.
3) Regridding: we store the entire grid in memory to avoid an expensive I/O operation when creating a new fine or coarse patch during regrid.

C. Benchmarking CRoCCo-AMR

This paper uses the Double Mach Reflection (DMR) test case, a Mach 10, inviscid flow problem that has been studied extensively in the literature [3]–[5]. The true experimental setup of the DMR problem is a planar Mach 10 shock incident on a 30° compression ramp. Throughout development we relied on regular validation runs to ensure code changes did not impact correctness on the DMR case. The correctness of CRoCCo was thoroughly tested and the the absolute differences of the L2-norm of significant variables on the DMR test case are within acceptable limits for this problem. While

the DMR problem does not require a curvilinear solver to resolve, all curvilinear features of our solver are exercised in resolving the problem in this work. Thus, the problem is useful for performance evaluation as it is easier to set up than a more complex curvilinear grid, and straightforward to validate against, given its extensive use in the literature.

Our results collection was conducted on two platforms, OLCF Summit for GPU runs and LLNL Quartz for CPU runs. Summit nodes have six NVIDIA V100s, hosted on two 22-core IBM POWER9 CPUs, connected with a non-blocking fat tree topology. Quartz nodes have Intel Xeon E5-2695 CPUs with 36 cores per node, connected with a fat tree network topology.

In a strong scaling performance test on 1.27×10^9 grid points, shown in Figure 1, we observe moderate speedup of the AMR-enabled version of the code over the non-AMR version ($4.7\times$ to $1.2\times$), and high speedup of the GPU version over the CPU + AMR version ($24.5\times$ to $5.1\times$). We also note that the performance of the GPU version of `CRoCCo` stops improving with additional nodes around 128 nodes in strong scaling, while the CPU version scales well up to 1024.

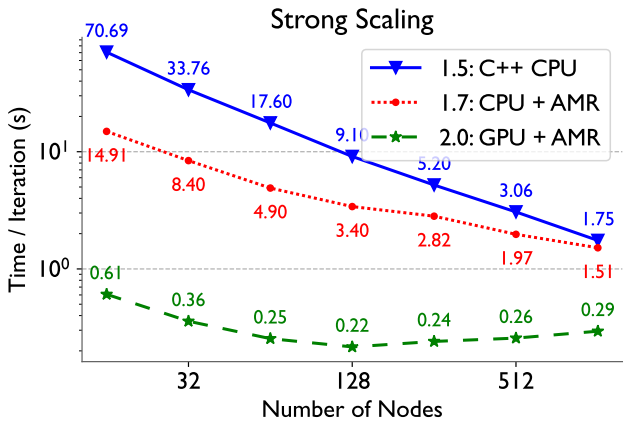

Fig. 1. Strong Scaling `CRoCCo` Comparison

D. Performance Profiling

Figure 2 shows the breakdown of time spent in the execution of our GPU-enabled `CRoCCo` 2.0 on 4, 16, 36, 64, and 100 Summit nodes, for 20 simulation iterations. The problem sizes for these experiments follows a weak scaling scheme, with 4.1×10^7 grid points per node. After porting to GPU, `CRoCCo` is strongly communication-bound, as demonstrated by the increasing portion of time spent in the `FillPatch` routine, showing roughly a 60% increase in time spent in `FillPatch` from 4 to 100 nodes. In `FillPatch`, `CRoCCo` exchanges ghost points and carries out a `ParallelCopy` in our custom curvilinear interpolation routine to read the grid.

Due to global communication in the form of an `amrex::ParallelCopy` in the `FillPatch` routine, our communication costs increase with the number of nodes, which explains why our scaling benefit ends earlier in the GPU version of the code. The end of good scaling behavior at 128 nodes corresponds to when the problem size per GPU becomes too small for the benefit of additional GPUs to outweigh the communication costs of additional nodes.

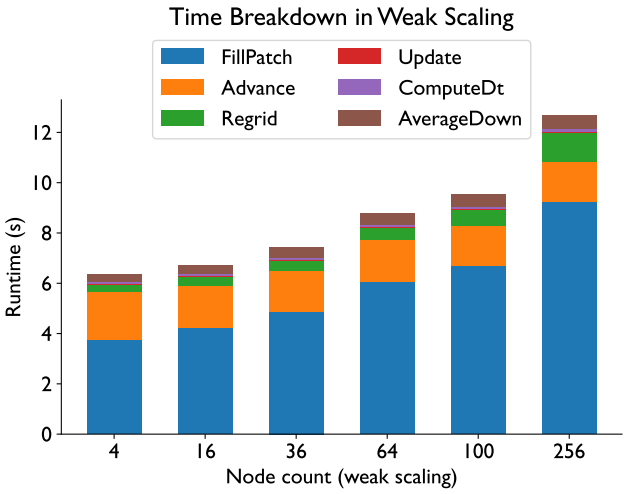

Fig. 2. Decomposition of `CRoCCo` runtime for five weak scaling cases on Summit

E. Conclusion and Future Work

We have described our approach and experiences in porting `CRoCCo` to extreme-scale GPU platforms with the addition of AMR capabilities. We have also described our modifications to the AMReX framework to enable curvilinear simulation, a capability previously unsupported by the AMReX framework. We advise future developers working on similar engineering efforts to expect degraded scaling performance of a GPU port due to the relative increase in communication overhead compared to computation. We also suggest minimizing parallel copy operations in AMReX and considering alternative approaches like storing needed data in memory.

Future efforts in improving `CRoCCo` 2.0 will include further optimizing our GPU kernels to improve the FLOP/s attained, optimizing communication to reduce the impact of the communication bottleneck exposed by our accelerated kernels, and extending the scientific capabilities of `CRoCCo` to take advantage of our much improved time-to-solution.

Acknowledgment

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1840340. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] W. Zhang, A. Myers, K. Gott, A. Almgren, and J. Bell, "AMReX: Block-structured adaptive mesh refinement for multiphysics applications," *The International Journal of High Performance Computing Applications*, vol. 35, no. 6, pp. 508–526, 2021. [Online]. Available: https://doi.org/10.1177/10943420211022811

[2] M. Martin, E. Taylor, M. Wu, and V. Weirs, "Bandwidth-optimized weno scheme for the direct numerical simulation of compressible turbulence," *J. Comp. Phys.*, vol. 220, pp. 270–289, 2006.
[3] F. Kemm, "On the proper setup of the double Mach reflection as a test case for the resolution of gas dynamics codes," *Computers & Fluids*, vol. 132, pp. 72–75, 2016.
[4] U. Vevek, B. Zang, and T. H. New, "On alternative setups of the double Mach reflection problem," *Journal of Scientific Computing*, vol. 78, no. 2, pp. 1291–1303, 2019.
[5] G. Ben-Dor, "Analytical solution of double-Mach reflection," *AIAA Journal*, vol. 18, no. 9, pp. 1036–1043, 1980.