University of Michigan XFlow Group
Correspondence: Steve Kast, *kastsm@umich.edu* PI: Krzysztof Fidkowski

C1.2 Ringleb Flow

1. Code description

XFlow is a high-order discontinuous Galerkin (DG) finite element solver written in ANSI C, intended to be run on Linux-type platforms. Relevant supported equation sets include compressible Euler, Navier-Stokes, and RANS with the Spalart-Allmaras model. High-order is achieved compactly within elements using various high-order bases on triangles, tetrahedra, quadrilaterals, and hexahedra. Parallel runs are supported using domain partitioning and MPI communication. Visual post-processing is performed with an in-house plotter. Output-based adaptivity is available using discrete adjoints.

2. Case summary

The default implicit Newton solver was used for all runs in this case. The residual was converged to an absolute L_1 norm below 10^{-10} using a conservative state vector of $\mathcal{O}(1)$ freestream density, velocity, and pressure, and gas constant $R = 1.0$. Uniform refinement and adaptive runs were performed on the *flux* supercomputing cluster at the University of Michigan. One Taubench unit on this machine corresponds to 9.08 seconds of compute time. The number of cores ranged from 1 to 3.

3. Meshes

The meshes were generated using a Matlab script with Q=4 and consist of quadrilateral elements. They are the same meshes as on the workshop site. The inviscid wall boundary condition led to numerical difficulties for some interpolation orders, so we have instead used the exact (analytical) states as boundary conditions along the walls.

4. Results

The following figures and tables show the requested data for this case. Uniform refinement results are shown first, followed by adaptive runs.

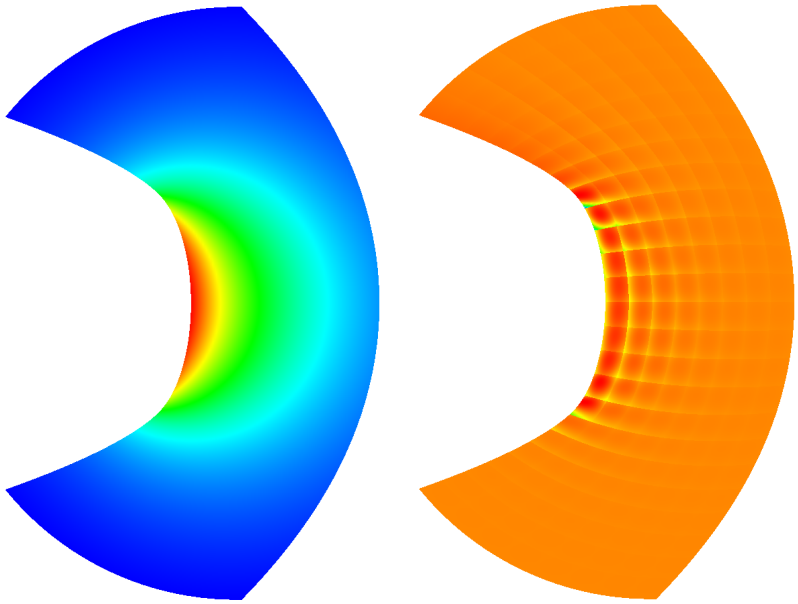
Uniform Refinement

Representative solution contours are shown in Figure 1, and entropy error convergence for various p- and h-refinements is provided in Figure 2. In addition, convergence for the errors in the conserved quantities is shown in Figure 3. The behavior of the entropy and conserved-variable errors is similar overall. Sample entropy error convergence rates and work units are given in Table 1, and expected rates are achieved for all orders.

Adaptive Refinement

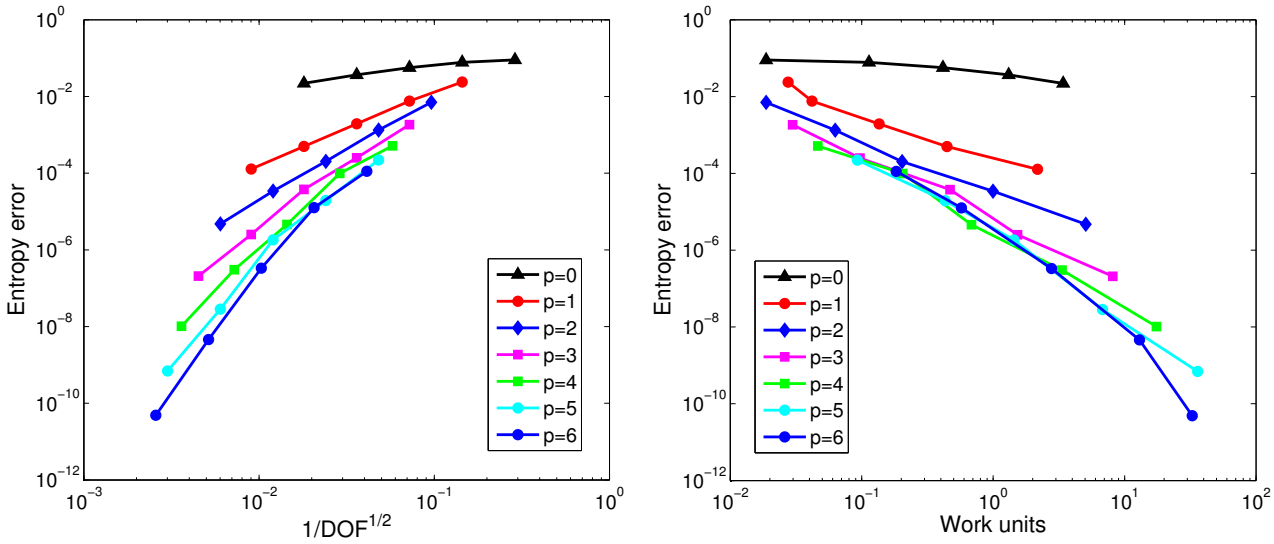
Next, we perform adaptive mesh refinement, with several indicators used for comparison. These indicators consist of (i) the entropy variables (which serve as an adjoint to an entropy-flux output [1]), (ii) the unweighted residuals, (iii) the adjoint for the entropy error, and (iv) uniform p- and h-refinements. Figure 4 shows the convergence for each of these methods in terms of both mesh size and work units.

In terms of mesh size, we see that the entropy variables, entropy error adjoint, and residual perform the best. Uniform p-refinement also does well (given the smooth nature of the problem), while uniform h-refinement is the least efficient. In terms of work units, the residual adaptation, entropy variables,



(a) mach number (b) entropy

Figure 1: Mach number and entropy contours. The entropy generation is spurious, and occurs near regions of high curvature.

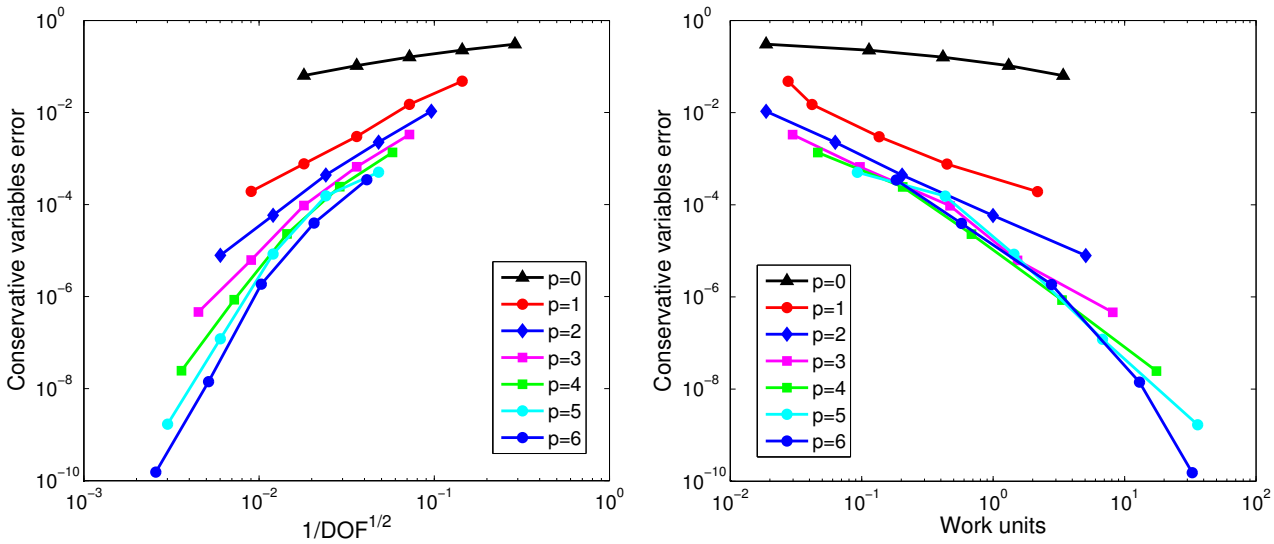


(a) (b)

Figure 2: Entropy error convergence as a function of mesh size and work units for uniform refinement.

and uniform p-refinement show similar efficiency, with the adjoint trailing slightly behind and uniform h-adaptation lagging significantly. Adapted meshes for each method are shown in Figure 5. The adjoint, residual, and entropy variables adapt similar regions, focusing refinement near the regions of large curvature.

Finally, note that a direct comparison between the adaptive work units and the uniform-refinement work units (shown in Figure 2) should not be made. This is because each run in the uniform-refinement



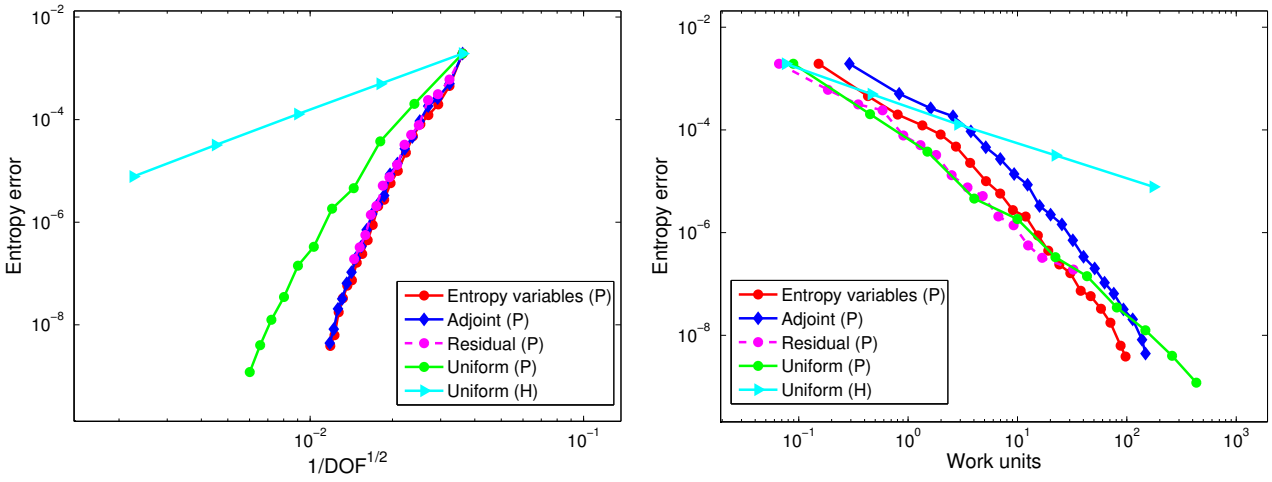
(a) (b)

Figure 3: Conservative-variable error convergence as a function of mesh size and work units. To compute the error, the squared errors in the individual state components are summed at each point (as part of an L2 norm), and this sum is then integrated over the domain.

Table 1: Entropy error convergence for the uniform refinement runs.

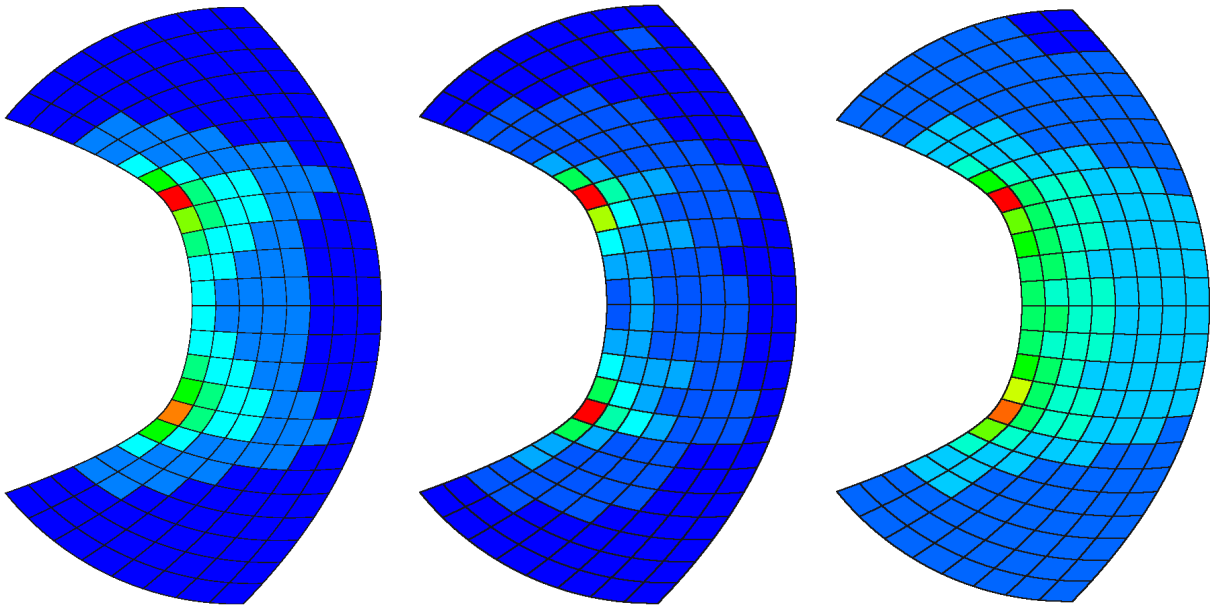
	Ref=0	Ref=1	Ref=2	Ref=3	Ref=4
$p = 0$	8.9978e-2	7.7872e-2	5.6572e-2	3.6795e-2	2.2045e-2
Rate	-	0.2	0.5	0.6	0.7
Work	1.8722e-2	1.1344e-1	4.1520e-1	1.3128	3.4064
$p = 1$	2.3855e-2	7.6064e-3	1.9380e-3	4.9872e-4	1.2827e-4
Rate	-	1.6	2.0	2.0	2.0
Work	2.7533e-2	4.1850e-2	1.3546e-1	4.4493e-1	2.1740
$p = 2$	7.0529e-3	1.3295e-3	2.0239e-4	3.4282e-5	4.7445e-6
Rate	-	2.4	2.7	2.6	2.9
Work	1.8722e-2	6.2775e-2	2.0264e-1	9.9670e-1	5.0672
$p = 3$	1.8459e-3	2.5038e-4	3.7781e-5	2.5221e-6	2.0796e-7
Rate	-	2.9	2.7	3.9	3.6
Work	2.9736e-2	9.6916e-2	4.6916e-1	1.5308	8.1200
$p = 4$	5.1607e-4	9.8888e-5	4.5996e-6	3.0381e-7	1.0220e-8
Rate	-	2.4	4.4	3.9	4.9
Work	4.6256e-2	2.0485e-1	6.8282e-1	3.3370	1.7483e+1
$p = 5$	2.2170e-4	1.9471e-5	1.8366e-6	2.8503e-8	6.9519e-10
Rate	-	3.5	3.4	6.0	5.4
Work	9.2511e-2	4.3172e-1	1.4394	6.7974	3.5980e+1
$p = 6$	1.1249e-4	1.2642e-5	3.3273e-7	4.6005e-9	4.8770e-11
Rate	-	3.2	5.2	6.2	6.6
Work	1.8282e-1	5.7379e-1	2.7786	1.2982e+1	3.2751e+1

sequence is initialized from the exact solution, whereas only the first adaptive run is initialized with the exact states. This makes the runs in the uniform-refinement sequence appear cheaper than the adaptive runs.



(a) (b)

Figure 4: Entropy error convergence for the adaptive runs. Note that an "H" or "P" in the legend indicates the type of adaptation performed. For H adaptation, an order of $p = 1$ was used.



(a) entropy variables (b) residual (c) adjoint

Figure 5: Final meshes for the various adaptive methods (blue is low order, red is high order). All methods target the regions of large curvature, where entropy is generated.

References

[1] K. J. Fidkowski, P. L. Roe, An entropy adjoint approach to mesh refinement, SIAM Journal on Scientific Computing 32 (3) (2010) 1261–1287.