11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

A GPU Accelerated Discontinuous Galerkin Approach to Conservative Level Sets

Zechariah J. Jibben[*1†], **Marcus Herrmann**[2†]

[1] zjibben@asu.edu [2] mherrma1@asu.edu
[†] Arizona State University, P.O. Box 876106 Tempe, AZ 85287-6106, USA, multiphase.asu.edu

Key words: GPU, Multiphase, Discontinuous Galerkin, Level Set

Abstract. We present a GPU-accelerated, arbitrary-order, nearly quadrature-free, Runge-Kutta (RK) discontinuous Galerkin (DG) approach to interface capturing for atomizing multiphase flows via the conservative level set (CLS) method [16, 17]. An arbitrary-order DG numerical method is utilized for both advection and reinitialization, further developing the ideas of Czajkowski and Desjardins [3] by implementing a quadrature-free approach, allowing for arbitrary polynomial degree, and treating the normal function in a DG sense. For effective use of processing power, the method is executed with the dual narrow band overset mesh approach of the refined level set grid method [11]. Computation is performed in parallel on either CPU or GPU architectures to make the method feasible at high order. Finally, by using normalized Legendre polynomial basis functions, we are able to pre-compute volume and surface integrals analytically. The resulting sparse integral arrays are stored in the compressed row storage format to take full advantage of parallelism on the GPU, where performance relies heavily on well-managed memory operations.

The accuracy, consistency, and convergence of the resulting method is demonstrated using the method of manufactured solutions (MMS). Using MMS, we demonstrate $k+1$ order spatial convergence for k^{th} order normalized Legendre polynomial basis functions on both advection and reinitialization. MMS is also used to demonstrate the benefits of GPU hardware, where advection is found to provide a speedup factor >45x comparing a 2.0GHz Intel Xeon E5-2620 in serial against a NVIDIA Tesla K20c, with speedup increasing with polynomial degree. Arbitrarily high convergence rates combined with speedup factors that increase with polynomial degree motivate the development and use of a GPU accelerated, arbitrary-order DG method.

1 Introduction

The level set method is a popular approach to follow the motion of interfaces in numerical simulations [22] and has been widely used in simulations of multiphase flows involving phase interfaces. While exhibiting some advantages over alternative numerical approaches

to capture the interface, level set methods have the distinct disadvantage that they are not locally volume conserving for divergence free velocity fields. That is, there is no built in discrete constraint that conserves the volume enclosed by the iso-surface of the level set scalar that defines the position of the interface. Numerous numerical methods have been devised to overcome this issue by coupling the level set method to other, better volume conserving interface capturing or tracking methods, see for example [9, 23].

The approach proposed by Olsson and Kreiss [16] and Olsson et al. [17], on the other hand, reformulates the level set scalar as a conserved quantity itself by using the divergence free velocity constraint of low Mach number flows. As such, the level set scalar, in essence, becomes a smeared out Heaviside function. While this Conservative Level Set (CLS) method strictly speaking still does not guarantee discrete conservation of the level set iso-scalar enclosed volume if the thickness of the smeared out Heaviside function is non-zero, the method exhibits drastically improved volume conservation properties compared to other popular level set methods that are based, for example, on a distance function formulation. The CLS method has, for example, been successfully applied to atomizing flows by Desjardins and Pitsch [5, 6] and Desjardins et al. [7]. The discrete volume conservation quality of the CLS method is directly linked to the imposed thickness of the smeared out Heaviside function. Numerical methods that are able to solve the linear level set advection equation with minimum numerical dissipation and dispersion for a nearly discontinuous solution variable are thus ideal candidates for the CLS method. Discontinuous Galerkin methods potentially fall into this category. They have the added benefit that they are easy to parallelize and thus applicable to many modern massively parallel supercomputer platforms.

In this paper, we present a Runge-Kutta Discontinous Galerkin method in quadrature-free form for solving the advection and reinitialization equations of the CLS method. We follow to a significant extent the work done by Czajkowski and Desjardins [3], however, we introduce some key modifications and developments that allow the method to be formally of order $k+1$ for k-th order Legendre polynomial basis functions. Furthermore, the computational expense which is especially apparent at high k is mitigated by the use of GPU architectures.

2 The Conservative Level Set Method

The conservative level set method is constructed under the assumption of divergence-free velocity fields, allowing the advection equation to treat the level set scalar G as a conserved variable

$$\frac{\partial G}{\partial t} + \nabla \cdot (G\boldsymbol{u}) = 0 \tag{1}$$

Then, the interface is defined as the $G = 0.5$ isosurface, where G has a hyperbolic tangent profile in the vicinity of the interface.

$$G(\boldsymbol{x}, t) = \frac{1}{2}\left(\tanh\left(\frac{\phi(\boldsymbol{x}, t)}{2\varepsilon}\right) + 1\right) \tag{2}$$

Here, ϕ is the signed distance function and the thickness of the profile is proportional to ε, which is set to half the cell width Δx. This profile is chosen since it is the exact solution of a

conservative reinitialization equation,

$$\frac{\partial G}{\partial \tau} + \nabla \cdot (G\,(1-G)\,\hat{\boldsymbol{n}}) = \nabla \cdot (\varepsilon\,(\nabla G \cdot \hat{\boldsymbol{n}})\,\hat{\boldsymbol{n}})\,, \tag{3}$$

where $\hat{\boldsymbol{n}}$ is the normal to the interface. Reinitialization is necessary since advection does not maintain the initial level set scalar profile; rather, advecting the interface deforms the scalar by introducing dissipative errors. Therefore, between advection steps it is necessary to reinitialize G in pseudo-time.

3 The Runge-Kutta Discontinuous Galerkin Method

The discontinuous Galerkin (DG) method, originally developed by Reed and Hill [18], is motivated by arbitrarily high convergence rates that can be achieved with a small stencil, containing only immediate neighbors. DG accomplishes this by allowing sub-cell variation and storing information about derivatives locally in the form of basis function coefficients. LeSaint and Raviart [12] proved that this method can formally achieve a $k+1$ order convergence rate with k^{th} degree polynomials on linear problems, while Cockburn and Shu [1] found this to also be achievable for nonlinear problems in practice. This section describes the scheme construction.

DG involves first spectrally discretizing the solution variables in each cell by projecting them into a local basis $\{b_i\}$ which is dependent on sub-cell coordinates $\boldsymbol{\xi} \in \mathscr{K} = [-1,1]^3$ bounded by a cube domain.

$$\boldsymbol{f}(\boldsymbol{x},t) \;= \sum_{i=1}^{N_f} \boldsymbol{f}_i^{\kappa}(t)\, b_i(\boldsymbol{\xi})\,, \tag{4}$$

The series is truncated at N_f (for the purposes of this paper, $N_g = N_u = N_n$ and $\Delta x = \Delta y = \Delta z$). In this sense, a finite volume method is equivalent to a DG method with $N_g = N_u = N_n = 1$. The normalized Legendre polynomial basis is selected for their orthonormality, and are constructed using Gram-Schmidt orthonormalization on the space of 3D monomials $\xi^\alpha \eta^\beta \zeta^\gamma$. Then, for a maximum monomial degree k, the number of terms in the spectral expansion is $N_g = (k+1)^3$.

These expansions are then substituted into Eq. (1) and Eq. (3), with spatial derivative variables changed to sub-cell coordinates. By performing an inner product with b_n (integrate over the cell domain $\mathscr{K}$), taking advantage of orthonormality, and using the divergence theorem, we arrive at a system of coupled ordinary differential equations describing the time evolution the DG coefficients g_n^κ. The RKDG method has been implemented in the context of the CLS method previously by Czajkowski and Desjardins [3]. However, their method held the velocity and normal vectors constant within a cell, only expanding the level set scalar to full order in the discontinuous basis. The result was a scheme limited to second order. Here, all variables

are projected into the DG basis at full order, allowing for higher convergence rates.

$$\frac{\mathrm{d}g_n^\kappa}{\mathrm{d}t} = u_k^{\kappa,j} g_i^\kappa \frac{2}{\Delta x} \int_{\mathscr{K}} b_k b_i \frac{\partial b_n}{\partial \xi_j} \,\mathrm{d}V + u_k^{\mathrm{up},j} g_i^{\mathrm{up}} \frac{2}{\Delta x} \int_{\partial\mathscr{K}} N_j b_k^{\mathrm{up}} b_i^{\mathrm{up}} b_n \,\mathrm{d}S \tag{5}$$

$$\begin{aligned}
\frac{\mathrm{d}g_n^\kappa}{\mathrm{d}\tau} &= n_k^{\kappa,j} g_i^\kappa \frac{2}{\Delta x} \int_{\mathscr{K}} b_k b_i \frac{\partial b_n}{\partial \xi_j} \,\mathrm{d}V - n_k^{\kappa,j} g_i^\kappa g_l^\kappa \frac{2}{\Delta x} \int_{\mathscr{K}} b_k b_i b_l \frac{\partial b_n}{\partial \xi_j} \,\mathrm{d}V \\
&\quad - \varepsilon g_i^\kappa n_k^{\kappa,a} n_l^{\kappa,d} \left(\frac{2}{\Delta x}\right)^2 \int_{\mathscr{K}} \frac{\mathrm{d}b_i}{\mathrm{d}\xi_a} b_k b_l \frac{\mathrm{d}b_n}{\mathrm{d}\xi_d} \,\mathrm{d}V \\
&\quad - \frac{1}{\Delta x}(g_i^{f+} \hat{\boldsymbol{n}}_k^{f+} + g_i^{f-} \hat{\boldsymbol{n}}_k^{f-}) \cdot \hat{\boldsymbol{N}}^f \int_{\partial^f\mathscr{K}} b_i^{f-} b_k^{f-} b_n \,\mathrm{d}S \\
&\quad + \frac{1}{\Delta x}(g_i^{f+} g_j^{f+} \hat{\boldsymbol{n}}_k^{f+} + g_i^{f-} g_j^{f-} \hat{\boldsymbol{n}}_k^{f-}) \cdot \hat{\boldsymbol{N}}^f \int_{\partial^f\mathscr{K}} b_i^{f-} b_j^{f-} b_k^{f-} b_n \,\mathrm{d}S \\
&\quad + \frac{C^f}{\Delta x}\left(g_i^{f+} \int_{\partial^f\mathscr{K}} b_i^{f+} b_n \,\mathrm{d}S - g_i^{f-} \int_{\partial^f\mathscr{K}} b_i^{f-} b_n \,\mathrm{d}S\right) \\
&\quad + \varepsilon \left(\frac{2}{\Delta x}\right)^2 \tilde{g}_i^f \tilde{n}_j^{f,k} \tilde{\hat{\boldsymbol{n}}}_l^f \cdot \hat{\boldsymbol{N}}^f (1 - \delta_{kf}/2) \int_{\tilde{\xi}_f=0} \frac{\partial \tilde{b}_i^f}{\partial \tilde{\xi}_k} \tilde{b}_j^f \tilde{b}_l^f b_n \,\mathrm{d}\tilde{S}^f
\end{aligned} \tag{6}$$

The advection equation flux is chosen by upwinding, while local Lax-Friedrichs is used for the nonlinear term in the reinitialization equation. The $f\pm$ cells refer to the cell on the $\pm$ side of face f. The diffusive flux is handled via a reconstruction method described by Luo et al. [14]. Here, a tilde refers to coefficients associated with a shared basis (also designated by a tilde) across two neighboring cells. These coefficients are calculated by

$$\tilde{f}_{n,\kappa}^\alpha = f_{i,\kappa}^{\alpha-} \int_{\widetilde{\mathscr{K}}^{\alpha-}} b_i \tilde{b}_n^\alpha \,\mathrm{d}\widetilde{V}^\alpha + f_{i,\kappa}^{\alpha+} \int_{\widetilde{\mathscr{K}}^{\alpha+}} b_i \tilde{b}_n^\alpha \,\mathrm{d}\widetilde{V}^\alpha \tag{7}$$

where $\widetilde{\mathscr{K}}^{\alpha\pm}$ refers to the $\pm$ half of the domain $\widetilde{\mathscr{K}}$ shared between two neighboring cells. Finally, time stepping is performed by an explicit $k+1$ order Runge-Kutta total variation diminishing (TVD) approach, as used in [1] and described by Gottlieb [10].

The time step size is limited by CFL conditions, found through von Neumann stability analysis. The CFL condition for convective terms is provided by [1]:

$$\max|\boldsymbol{f}'(G)| \frac{\Delta t}{\Delta x} \leq \frac{1}{2k+1} \tag{8}$$

The flux function derivatives, knowing that $|\hat{\boldsymbol{n}}| = 1$ and the CLS method restricts $0 \leq G \leq 1$, are

$$\begin{aligned}
\text{advection:} \quad & \max|\boldsymbol{f}'(G)| = \max|\boldsymbol{u}| \\
\text{reinitialization convective term:} \quad & \max|\boldsymbol{f}'(G)| = 1
\end{aligned} \tag{9}$$

The diffusive term in the reinitialization equation restricts time step size by [13]:

$$\varepsilon \frac{\Delta t}{\Delta x^2} \leq \frac{\beta(k)}{(2k+1)^2 \sqrt{d}} \tag{10}$$

Table 1: Stable values for β presented by Lörcher et al. [13]

k	1	2	3	4	5	6	7
β	1.46	0.80	0.40	0.24	0.16	0.12	0.09

where $\beta(k)$ is a function of polynomial order (several values are given in Table 1) and d is the number of dimensions. In practice, $\varepsilon \sim \Delta x$ so that Δt scales with Δx rather than its square.

Lastly, notice that all of the above integrals are written only in terms of Legendre polynomials and their derivatives. We take advantage of this by precomputing them symbolically using software such as SymPy or Mathematica, and read the data into arrays at the beginning of the simulation. This can save significant compute time in avoiding quadrature, largely because the resulting arrays of integral values are sparse, and especially so in 3D at high order (see Table 2).

Table 2: Matrix Fill Fraction for Advection Integral Arrays

Polynomial	2D Simulation Integrals			3D Simulation Integrals		
Degree	# of elements	Volume	Surface	# of elements	Volume	Surface
1	64	12.5%	50.0%	512	6.25%	25.0%
2	729	10.6%	40.7%	19683	4.30%	16.6%
3	4096	10.1%	35.9%	262144	3.63%	12.9%
4	15625	9.68%	33.6%	1953125	3.25%	11.3%

4 Computation

4.1 Compressed Row Storage

To avoid unnecessary data transfer, floating point operations, and to ensure maximum occupancy of GPU hardware, a compressed row storage technique is essential. There are several storage techniques, including ELLPACK, compressed diagonal storage, compressed row storage (CRS) , and many others. For our purposes, CRS has shown to be the most effective since multiple dimensions within a row can be most easily accessed in parallel.

Our CRS format, based on the work of Duff et al. [8], involves a series of several 1D arrays to represent a data structure, for example the 3D data structure $Z_{n,k,i}$. The first array, `Z`, contains the values of all nonzero array elements. Two arrays `start` and `end` contain the 1D array locations bounding a given row n. Then, two arrays `i2` and `i3` contain the full 2D array coordinates k and i associated with element l in row n of the array `Z`.

4.2 GPU Programming Model

Using CUDA terminology, a GPU operates by executing a function called a *kernel* in parallel on a cluster of *threads*, which are organized into *blocks* with resources allocated to groups of 32 threads called *warps*. Threads and blocks then have associated integers for identification. Eq. (5) is solved by assigning a single cell to each block, updating all level set scalar coefficients

for that cell. Then, threads within a block share the workload of tensor-vector multiplication.

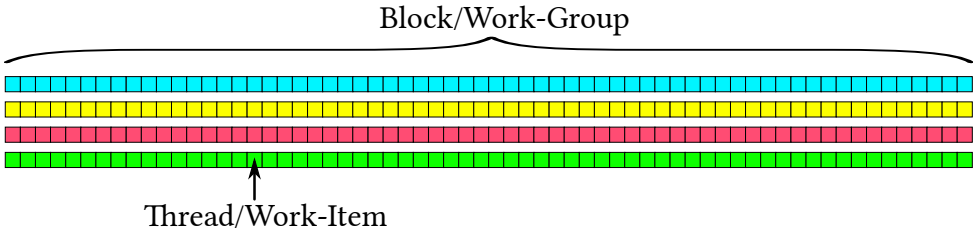


Figure 1: CUDA/OpenCL Execution Model

A second aspect of GPU programming is the use of different memory spaces for array storage: i) *global* memory, which is available to all threads but invokes an additional 400-600 clock cycles of latency when accessed [15] (for comparison, memory read/write time itself is 8 operations per clock cycle), ii) *shared* memory, which is accessible to members within the same block and may be accessed ∼100x faster than global memory [21] since the latency is significantly reduced, and iii) small *private* registers, which is not shared between threads but is slightly faster than shared memory. An excellent description of this is given by Scarpino [21]. Currently, integral arrays are stored in global memory. Since all g and u coefficients associated with a cell are accessed frequently by a block, storing solution variable coefficients in shared memory before calculating a given tensor multiplication term provides a 10-15% speedup. Unfortunately, shared memory cannot be dynamically allocated. Rather, the CPU must send a request to reserve variable sized blocks of memory before queuing kernel execution.

In Alg. 1, Eq. (5) is considered a series of equations of the form $\Delta g_n^\kappa \mathrel{+}= \sum_{k=1}^{N_u} u_k^\kappa \sum_{i=1}^{N_g} g_i^\kappa Z_{n,k,i}$ with coefficients for velocity `u`, level set scalar `g`, and an integral array `Z`. Each thread has its own instance of the variable `my_dg`, in which it sums together a subset of the above equation. Following the CRS format, we loop over a single integer l that corresponds to nonzero elements of the compressed array `Z[l]`. Each call of the multiplication routine evaluates a term for one row n, so we loop through a subset of `Z` bounded by two integers `start[n]` and `end[n]`.

In order to take advantage of memory coalescence and evenly distribute the workload, and hence reduce runtime, the local group of threads must align their access to the global array `Z` by their local id number [15]. For example, thread 7 will access the array element located immediately after the memory accessed by thread 6 and immediately before the memory accessed thread 8. To accomplish this, threads begin the loop offset by their local id and step through the loop by the local block size. Furthermore, the zeroth thread should access array elements that are multiples of 32, the warp size [2]. This optimization alone provides an additional 10-15% speedup.

Algorithm 1 GPU 3D Array Multiplication

```
tiX ⟵ local ID of thread
ntX ⟵ block size
term ⟵ 0.0
for l ⟵ start[n]+tiX, end[n] with step size ntX do
    term += u[i2[l]] * g[i3[l]] * Z[l]        ▷ multiply u and g coeff associated with l
end for
declare shared array partialsum of length ntX
partialsum[tiX] ⟵ term                        ▷ save private result to local array
return reduction_sum_within_tile(partialsum)
```

5 Results

5.1 Accuracy and Convergence

Here we demonstrate the accuracy and convergence of the method via several test cases. The first is the method of manufactured solutions (MMS), originally developed by Salari and Knupp [20] at Sandia National Laboratory. An excellent overview of the method is given by Roache [19]. In short, it allows arbitrary selection of an exact solution to a PDE by modifying it with an additional source term, thereby testing convergence rates and accuracy of the numerical method while neglecting the physics. A final test involves reinitialization of a circle with exact normal vectors. This test requires a much smaller thickness ϵ than the MMS test, increasing the possible time-step size allowed by Eq. (10) and the feasibility of testing high polynomial degree k.

5.1.1 Advection MMS

To test the advection equation, Eq. (1), with MMS, it is modified with a source term.

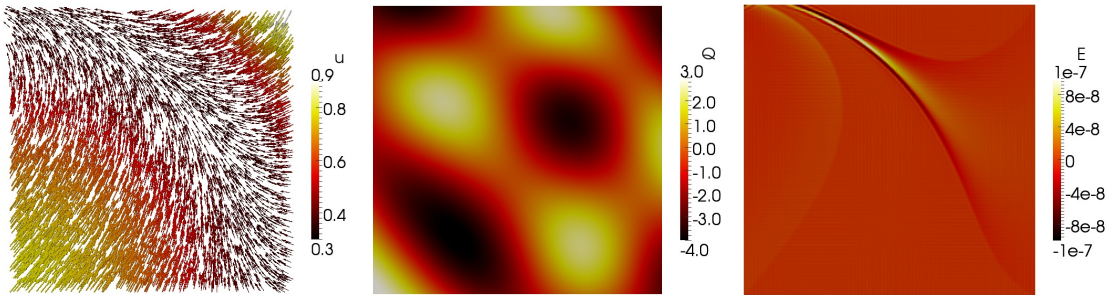
$$\frac{\partial G}{\partial t} + \nabla \cdot (G\boldsymbol{u}) = Q(\boldsymbol{x}, t)\,. \tag{11}$$

The source term is evaluated from an arbitrarily chosen exact solution and velocity field:

$$Q(\boldsymbol{x}, t) = \frac{\partial G_{\text{ex}}(\boldsymbol{x}, t)}{\partial t} + \nabla \cdot \left(G_{\text{ex}}(\boldsymbol{x}, t)\, \boldsymbol{u}_{\text{ex}}(\boldsymbol{x}, t) \right). \tag{12}$$

Finally, the RKDG scheme and code are tested on the unit-sized domain $[0, 1]^2$ with the following exact solution, prescribed velocity, and resulting source term:

$$\begin{aligned}
G_{\text{ex}}(x, y) &= \frac{1}{2} + \sin(2\pi x)\cos(2\pi y) \\
\boldsymbol{u}_{\text{ex}}(x, y) &= (\frac{1}{2} - \sin(x^2 + y^2))\hat{\boldsymbol{x}} + (\cos(x^2 + y^2) - \frac{2}{5})\hat{\boldsymbol{y}} \\
\implies Q(x, y) &= -2\cos(x^2 + y^2)x(1/2 + \sin(2\pi x)\cos(2\pi y)) \\
&\quad + 2(0.5 - \sin(x^2 + y^2))\cos(2\pi x)\pi\cos(2\pi y) \\
&\quad - 2\sin(x^2 + y^2)y(1/2 + \sin(2\pi x)\cos(2\pi y)) \\
&\quad - 2(\cos(x^2 + y^2) - 0.4)\sin(2\pi x)\sin(2\pi y)\pi
\end{aligned} \tag{13}$$



(a) velocity field $\boldsymbol{u}$ (b) source term Q (c) final error

Figure 2: MMS Advection Test

The velocity field, source term, and error at the final converged state at $t = 4.3$ are shown in Fig. 2. Comparing the final error to the velocity field, it is shown that the error predominantly lies in compressive regions where velocity vectors converge.

Table 3: Advection MMS test case error norms and order of convergence under grid refinement for RKDG-4.

Δx	L_∞	order	L_1	order
1/10	2.65e-5	-	3.37e-6	-
1/20	2.13e-6	3.6	1.03e-7	5.0
1/40	1.16e-7	4.2	3.32e-9	5.0
1/80	6.32e-9	4.2	1.07e-10	5.0
1/160	3.95e-10	4.0	3.43e-12	5.0

The errors produced for $k = 4$ are listed in Table 3. The results show a $k + 1$ order convergence rate in the L_1 norm with only a k^{th} order convergence rate in the L_∞ norm. The reduced convergence for L_∞ requires further study, but can possibly be attributed to the upwind scheme or convergent velocity field.

5.1.2 Reinitialization MMS

The reinitialization equation, Eq. (3), is modified with a source term as below:

$$\frac{\partial G}{\partial \tau} + \nabla \cdot \left(G \left(1 - G \right) \hat{\boldsymbol{n}} \right) = \nabla \cdot \left(\varepsilon \left(\nabla G \cdot \hat{\boldsymbol{n}} \right) \hat{\boldsymbol{n}} \right) + Q(\boldsymbol{x}, t) \,. \tag{14}$$

It should be noted that MMS does not require the normal vector to conform in any sense to the normal of any surface in this system, since it does not verify CLS. As such, there is also no need for it to be normalized. The source term is evaluated from the exact solution and prescribed normal vector field:

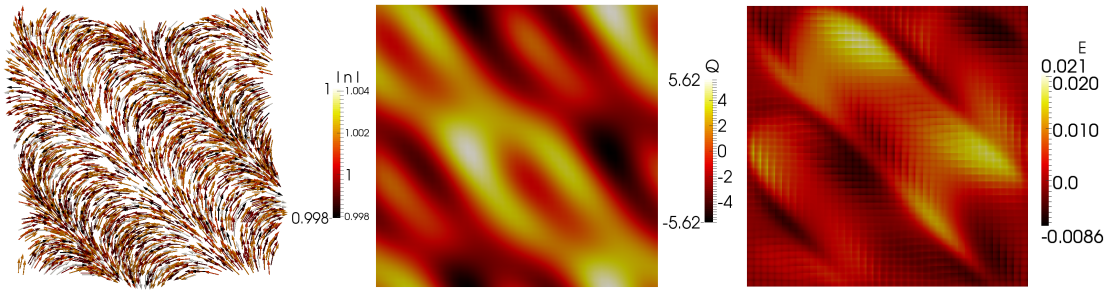
$$\begin{aligned} Q(\boldsymbol{x}, t) \;=\; & \frac{\partial G_{\text{ex}}(\boldsymbol{x}, \tau)}{\partial \tau} + \nabla \cdot (G_{\text{ex}}(\boldsymbol{x}, \tau)(1 - G_{\text{ex}}(\boldsymbol{x}, \tau))\hat{\boldsymbol{n}}_{\text{ex}}(\boldsymbol{x}, \tau)) \\ & - \nabla \cdot (\varepsilon(\nabla G_{\text{ex}}(\boldsymbol{x}, \tau) \cdot \hat{\boldsymbol{n}}_{\text{ex}}(\boldsymbol{x}, \tau))\hat{\boldsymbol{n}}_{\text{ex}}(\boldsymbol{x}, \tau)). \end{aligned} \tag{15}$$

The RKDG scheme for reinitialization was tested on the unit sized domain $[-0.5, 0.5]^2$ with the following exact solution and velocity:

$$\begin{aligned} G_{\text{ex}}(x,y) &= \frac{1}{2}(1+\cos(2\pi x)\cos(2\pi y)) \\ \hat{\boldsymbol{n}}_{\text{ex}}(x,y) &= \sin(2\pi(x+y))\hat{\boldsymbol{x}} + \cos(2\pi(x+y))\hat{\boldsymbol{y}}\,. \end{aligned} \tag{16}$$

Here the source term is much longer and therefore omitted, although it is easily derived from Eq. (15). The diffusivity constant ϵ is chosen to be 0.08 so that the amplitudes of convective and diffusive terms are equal.

The normal vectors, source term, and error at the final converged state at $t = 2.8$ are shown in Fig. 3. Comparing the final error to the normal field, it is found that, similar to advection, the error predominantly lies in compressive regions where normal vectors converge.



(a) normal field $\hat{\boldsymbol{n}}$ (b) source term Q (c) final error

Figure 3: MMS Reinitialization Test

Table 4: Reinitialization MMS test case error norms and order of convergence under grid refinement for RKDG-1.

Δx	L_∞	order	L_1	order
1/10	1.24e-1	-	2.75e-2	-
1/20	7.73e-2	0.68	1.11e-2	1.32
1/40	3.15e-2	1.30	3.57e-3	1.63
1/80	1.02e-2	1.63	1.02e-3	1.80

The results for $k = 1$ are listed in Table 4 and show a convergence rate that approaches $k+1$ in both the L_1 and L_∞ norms. Tests for higher k and finer grids are expensive to perform because the diffusive CFL restriction, Eq. (10), and the requirement that ϵ remain constant for grid refinement studies causes the scheme to become prohibitively expensive. Modifying the scheme to an explicit-implicit time stepping method similar to the fractional-step method may be used to prevent this, however, and is left to future work. Furthermore, performing reinitialization on GPU hardware will lessen the computational cost.

5.1.3 Circle Test

To assess reinitialization, a test case was developed which involves a circle of radius R_0 placed at the origin of a unit-sized $[-0.5, 0.5]^2$ domain. The level set scalar is initialized to

$$G(\boldsymbol{x}) = \frac{1}{2}\left(\tanh\left(\frac{R_0 - \sqrt{x^2+y^2}}{2\epsilon_0}\right) + 1\right). \tag{17}$$

Reinitialization then sharpens the interface from thickness $\varepsilon_0 \to \varepsilon$, the latter of which is used in Eq. (3). As the circle is refined, the 0.5-isosurface is transported outward in order to conserve G. The new radius R can be computed by assuming that G is transported only in the set normal direction and not tangent to the interface. Then, the integral over G from $r = 0, \infty$ remains unchanged under reinitialization and R can be computed in terms of R_0 , ε_0 , and ε.

$$\int_0^\infty G(r, R, \varepsilon) \ \mathrm{d}r = \int_0^\infty G\left(r, R_0, \varepsilon_0\right) \ \mathrm{d}r$$

The results of a refinement study for $k = 3$ polynomials are shown in Table 5, which shows the L_∞ and L_1 norms of the error at steady state together with the associated orders of convergence. Reinitialization is evaluated for a circle of initial radius $R_0 = 0.25$ and thickness $\varepsilon_0 = 0.025$ refined to $\varepsilon = 0.0125$. The final radius R is found to be 0.253063 from Eq. (5.1.3). As expected, the L_1 and L_∞ norms of the error converge with $k + 1$ order.

Table 5: Error norms of circle test and their order of convergence under grid refinement for RKDG-CLS-3.

Δx	L_∞	order	L_1	order
1/20	5.47e-2	-	3.24e-3	-
1/40	3.32e-3	4.04	1.68e-4	4.27
1/80	1.84e-4	4.17	9.40e-6	4.16

5.2 Speed

The previously described DG advection MMS test case was executed in 3D on a NVIDIA Tesla K20 GPU (with 128 threads/block) and an Intel Xeon E5-2620. Both algorithms take advantage of sparsity and are implemented on equidistant Cartesian meshes in unit sized domains. Since the RKDG-CLS method verification has been described through the previous test cases, this test is limited to compute times and assurance that the CPU and GPU give equivalent results (within 10^5 times machine epsilon at double precision). This is done by randomizing the level set scalar and velocity coefficients, then performing a single RK advection step and comparing the CPU and GPU runtime and output.

These tests produce several interesting trends, shown in Fig. 4. First, low-degree polynomials show little benefit from the GPU, simply because the integral arrays are not large enough for all threads to be operating simultaneously. A similar drawback arises if sparsity is not exploited, where the GPU instead slows down computation since many threads end up multiplying by zero, wasting significant FLOPs. One way to remedy this in practice is to use smaller work-group sizes when dealing with low-degree polynomials. However, this solution has limitations since GPUs are most efficient when the block size is a multiple of 32 (the warp size), as previously mentioned.

Second, the data indicates the GPU is increasingly advantageous as it is given more work. With more degrees of freedom and operations, whether from refining the grid or increasing the number of polynomials, the total speedup increases. This compliments the effectiveness

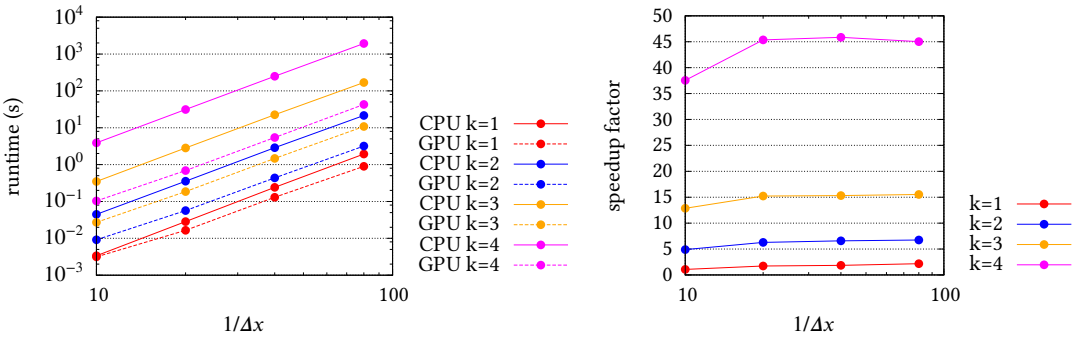


Figure 4: Compute Time and Speedup of One 3D Advection RK Step

of high-order DG and reflects the streaming memory model on the GPU, where one warp's memory latencies are masked by another's floating-point operations.

6 Summary

We have presented an arbitrary-order Runge-Kutta discontinuous Galerkin conservative level set method, expanding on the approach proposed by [3] by using high order velocity field and normal vector expansions, as well as accelerating the method on GPU hardware. As a result, we obtain the full formal order of the method as $k+1$ for k^{th} order Legendre polynomial basis functions, as shown by MMS test cases. Furthermore, the use of GPU hardware has improved computation speed for advection by over 45x for 4^{th} order polynomials.

In future work, we intend to also accelerate the reinitialization algorithm via GPU hardware, integrate improvements from the accurate conservative level set method proposed by [4] calculating the normal vectors from $\boldsymbol{G}$ in the vicinity of the interface alone, and finally calculate curvature in a DG sense. This will allow the complete GPU-RKDG-ACLS method to be coupled to a flow solver for engineering applications.

Acknowledgments

We are grateful for the support provided by the National Science Foundation grant CBET-1054272, the 2012 Stanford CTR summer program, and the 2013 Los Alamos Computational Physics student workshop. We also would like to acknowledge several groups for maintaining or contributing to the hardware on which this project was developed and tested, including the ASU Advanced Computing Center for the Saguaro cluster, Los Alamos National Laboratory for the Moonlight ASC and Darwin CCS-7 clusters, and NVIDIA for their Academic Hardware Donation.

References

[1] B. Cockburn and C.-W. Shu. *J. Sci. Comput.* 16 (2001), pp. 173–261.

[2] *CUDA C Programming Guide*. Version 5.5. NVIDIA Corporation. 2013.

[3] M. F. Czajkowski and O. Desjardins. *ILASS Americas 23rd Annual Conference on Liquid Atomization and Spray Systems* (2011).

[4] O. Desjardins, V. Moureau, and H. Pitsch. *J. Comput. Phys.* 227 (2008), pp. 8395–8416.

[5] O. Desjardins and H. Pitsch. *J. Comput. Phys.* 228 (2009), pp. 1658–1677.

[6] O. Desjardins and H. Pitsch. *Atom. Sprays* 20 (2010), pp. 311–336.

[7] O. Desjardins et al. *ILASS Americas 20th Annual Conference on Liquid Atomization and Spray Systems* (2007).

[8] I. Duff, R. Grimes, and J. Lewis. *User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I).* 1992.

[9] D. Enright et al. *J. Comput. Phys.* 183 (2002), pp. 83–116.

[10] S. Gottlieb. *J. Sci. Comput.* 25 (2005), pp. 105–128.

[11] M. Herrmann. *J. Comput. Phys.* 227 (2008), pp. 2674–2706.

[12] P. LeSaint and P. A. Raviart. *Mathematical Aspects of Finite Elements in Partial Differential Equations*. Ed. by C. de Boor. Academic Press, NY, 1974, pp. 89–123.

[13] F. Lörcher, G. Gassner, and C.-D. Munz. *J. Comput. Phys.* 227 (2008), pp. 5649–5670.

[14] H. Luo et al. *J. Comput. Phys.* 229 (2010), pp. 6961–6978.

[15] *NVIDIA OpenCL Best Practices Guide*. Version 1.0. NVIDIA Corporation. 2009.

[16] E. Olsson and G. Kreiss. *J. Comput. Phys.* 210 (2005), pp. 225–246.

[17] E. Olsson, G. Kreiss, and S. Zahedi. *J. Comput. Phys.* 225 (2007), pp. 785–807.

[18] W. H. Reed and T. R. Hill. *Triangular mesh methods for the neutron transport equation.* Tech. rep. LA-UR-73-479. Los Alamos National Laboratory, 1973.

[19] P. J. Roache. *J. Fluids Eng.* 124 (2002), pp. 4–10.

[20] K. Salari and P. Knupp. *Code verification by the method of manufactured solutions*. Tech. rep. SAND2000-1444. Sandia National Laboratory, 2000.

[21] M. Scarpino. *OpenCL in Action: How to Accelerate Graphics and Computation*. Shelter Island, NY: Manning Publications Co., 2012.

[22] J. A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Science*. Cambridge: Cambridge University Press, 1996.

[23] M. Sussman and E. G. Puckett. *J. Comput. Phys.* 162 (2000), pp. 301–337.