

GPU-Accelerated Discontinuous Galerkin Methods: 30x Speedup on 345 Billion Unknowns

Andrew C. Kirby

*Lincoln Laboratory Supercomputing Center
Massachusetts Institute of Technology
Lexington, MA, USA*

Dimitri J. Mavriplis

*Mechanical Engineering
University of Wyoming
Laramie, WY, USA*

Abstract—A discontinuous Galerkin method for the discretization of the compressible Euler equations, the governing equations of inviscid fluid dynamics, on Cartesian meshes is developed for use of Graphical Processing Units via OCCA, a unified approach to performance portability on multi-threaded hardware architectures. A 30x time-to-solution speedup over CPU-only implementations using non-CUDA-Aware MPI communications is demonstrated up to 1,536 NVIDIA V100 GPUs and parallel strong scalability is shown up to 6,144 NVIDIA V100 GPUs for a problem containing 345 billion unknowns. A comparison of CUDA-Aware MPI communication to non-GPUDirect communication is performed demonstrating an additional 24% speedup on eight nodes composed of 32 NVIDIA V100 GPUs.

Index Terms—Computational Fluid Dynamics, Discontinuous Galerkin Methods, OCCA, GPU Computing

I. INTRODUCTION

A transformation in computing architectures has emerged over the past decade in the quest to keep Moore’s Law alive, leading to significant advancements in Artificial Intelligence and Machine Learning techniques [1], [2]. The success of Machine Learning has caused a role reversal, now becoming a primary driver in the development of massively-parallel thread-based hardware indicative of future exascale-era architectures [3]. Traditional Computational Science applications such as Computational Fluid Dynamics (CFD) has eschewed the use of Graphical Processing Units (GPU) due to their programming complexity to achieve high performance. However, in this volatile heterogeneous computing landscape, algorithms suitable for these platforms are now forefront in the advancement of these fields.

Low-order discretizations such as the finite volume method [4], [5] have been the industrial standard in CFD for the last 40 years. However, these algorithms suffer from low arithmetic intensity¹ and were never conceived with massive parallelism in mind, making them potentially ill-suited for heterogeneous systems containing GPUs. According to the NASA CFD Vision 2030 Study [6], “Beyond potential advantages in improved accuracy per degree of freedom, higher-order methods may more effectively utilize new HPC hardware through increased levels of computation per degree of freedom.” To leverage these new architectures and enable

higher-fidelity simulations, algorithmic advancement of high-order methods, such as the discontinuous Galerkin method [7] and the flux-reconstruction method [8], have emerged within the Computational Science community. These high-order methods have higher-algorithmic intensity and naturally-tiered parallelism through inter-element distributed memory strategies and intra-element shared memory parallelism.

This work develops a discontinuous Galerkin (DG) method to discretize the governing equations of fluid dynamics concerning aerospace and atmospheric applications. The discontinuous Galerkin method exhibits highly-parallelizable computational kernels due to its nearest-neighbor-only element communication pattern and high-intensity localized compute stencil, making them well suitable for GPUs in a distributed computing environment. Implementation of a DG methods for hyperbolic systems of equations on GPUs was introduced in [9] and extension and optimization of this work conducted in [10]–[15].

We employ an thread-programming abstraction model known as OCCA [16], an open-source library, enabling the use of GPUs, CPUs, and FPGAs, to achieve performance portability. OCCA allows developers to write performance-portable code in the OCCA Kernel Language (OKL) [17] which is an intermediate representation for shared-memory parallelism. The intermediate representation is just-in-time compiled to various backends including serial code, OpenMP, CUDA, and OpenCL. Efficient and scalable implementations of high-order discretizations for atmospheric conditions using OCCA have been demonstrated by Abdi et. al in [18] and [19].

A nodal DG method with collocation of integration and solution points is assumed enabling several arithmetic simplifications. Further, we conform to Cartesian meshes resulting in constant mesh parameters further simplifying the discretization. This is done to establish a performance ceiling for future algorithmic developments involving dynamically adapting unstructured meshes and variable polynomial order element-wise solutions, i.e. hp —adaption. All computations involved in the discretization of the governing equations are ported to the GPU for this work including spatial and temporal residuals. The performance of the method is analyzed and comparisons of network communications via CUDA-Aware MPI are performed. Lastly, we demonstrate the parallel scalability of the implementation up to 6,144 NVIDIA V100 GPUs.

This work was partially supported under ONR Grant N00014-16-1-2737.

¹Arithmetic intensity is the ratio of floating-point operations performed to the amount of memory used in the calculation.

II. DISCONTINUOUS GALERKIN METHOD

A. Governing Equations

The governing equations utilized in this work are the three-dimensional compressible inviscid Euler equations, which are written in conservative form:

$$\frac{\partial \mathbf{Q}(\mathbf{x}, t)}{\partial t} + \vec{\nabla} \cdot \mathbf{F}(\mathbf{Q}(\mathbf{x}, t)) = 0 \quad (1)$$

representing the conservation of mass, momentum, and energy for a fluid. The solution vector \mathbf{Q} represents the conservative flow variables and the matrix \mathbf{F} represents the Cartesian flux components. \mathbf{Q} and \mathbf{F} are defined as follows:

$$\mathbf{Q} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{Bmatrix}, \mathbf{F} = \begin{Bmatrix} \mathbf{F}^1 & \mathbf{F}^2 & \mathbf{F}^3 \\ \rho u & \rho v & \rho w \\ \rho u^2 + p & \rho uv & \rho uw \\ \rho uv & \rho v^2 + p & \rho vw \\ \rho uw & \rho vw & \rho w^2 + p \\ \rho uH & \rho vH & \rho wH \end{Bmatrix} \quad (2)$$

where ρ is density, u, v, w are velocity components in each spatial coordinate direction, p is pressure, E is total internal energy, and $H = E + \frac{p}{\rho}$ is total enthalpy. These equations are closed using the ideal gas equation of state:

$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2} \rho (u^2 + v^2 + w^2) \quad (3)$$

where $\gamma = 1.4$ is the ratio of specific heats.

B. Discretization

This work constrains the computational domain to Cartesian meshes. Under these assumptions, arithmetic simplifications are performed to increase computational performance. Within this setting, the governing equations are discretized via the discontinuous Galerkin (DG) method [20] assuming a weighted residual formulation. A weak formulation is established by taking the inner product of the governing equations and set of piecewise continuous test functions, where the inner product is defined as multiplying each equation by a set of test functions $\psi_s \in l(x)$, the Lagrange basis polynomials up to degree N , and integrating over the Cartesian element volume Ω_k :

$$\int_{\Omega_k} \left(\frac{\partial \mathbf{Q}}{\partial t} + \vec{\nabla} \cdot \mathbf{F} \right) \psi_s(\mathbf{x}) d\mathbf{x} = 0 \quad (4)$$

Integrating (4) by parts, the weak-form residual $\mathbf{R}_s^{\text{Weak}}$ is defined as:

$$\begin{aligned} \mathbf{R}_s^{\text{Weak}} &= \int_{\Omega_k} \frac{\partial \mathbf{Q}}{\partial t} \psi_s(\mathbf{x}) d\mathbf{x} \\ &\quad - \int_{\Omega_k} (\mathbf{F} \cdot \vec{\nabla}) \psi_s(\mathbf{x}) d\mathbf{x} \\ &\quad + \int_{\Gamma_k} (\mathbf{F}^* \cdot \vec{n}) \psi_s(\mathbf{x}|_{\Gamma_k}) d\Gamma_k = 0 \end{aligned} \quad (5)$$

The discrete residual contains integrals over mesh element boundaries Γ where special treatment is needed for the fluxes. The advective fluxes \mathbf{F}^* are calculated using an approximate Riemann solver, namely the Lax-Friedrichs method [21]. To complete the spatial discretization, the solution is approximated as follows:

$$\mathbf{Q}(x, t) = \sum_{s=1}^N \hat{Q}_s(t) \phi_s(x) \quad (6)$$

where $\phi_s(x)$ are chosen to be the same as the test functions $\psi_s(x)$. A collocation approach is chosen for computational efficiency by choosing the basis and test functions to be Lagrange interpolation polynomials

$$\ell_s(x) = \prod_{i=1, i \neq s}^N \frac{(x - \xi_i)}{(\xi_s - \xi_i)}, \quad s = 1, \dots, N \quad (7)$$

where $N = p + 1$: p is the user-chosen polynomial degree, and numerical integration via Gauss-Legendre quadrature is used. Lastly, we implement a low-storage explicit Runge-Kutta method to discretize the temporal component of (4).

III. CPU IMPLEMENTATION

This work serves within the larger computational framework known as the Wyoming Wind and Aerospace Applications Komputation Environment (WAKE3D) [22]–[24]. WAKE3D has been demonstrated on various aerodynamics problems [25] and wind energy applications [26]–[29]. The framework is composed of multiple software components, namely NSU3D [30], DG4est [27], [31], and TIOGA [32], [33]. Figure 1 illustrates the simulation of a Siemens SWT-2.3-93 wind turbine using WAKE3D.

The computational kernel within the DG4est component, known as CartDG [34]–[36], is the primary focus of this work. CartDG is a discontinuous Galerkin method designed for computational efficiency on Cartesian meshes utilizing tensor-product collocation-based basis functions. The CPU implementation has achieved over 10% sustained peak of theoretical compute performance [37] using the viscous formulation as shown in Figure 2. CartDG utilizes the Message Passage Interface (MPI) for distributed-memory computation and enables computation-communication overlap to hide communication latency. It has been demonstrated to scale to over one million MPI ranks [22] on ALCF Mira Supercomputer.

IV. GPU IMPLEMENTATION

All computational kernels herein are ported via OCCA including the spatial volume and surface residuals and a five-stage low-storage Runge-Kutta temporal discretization. Discontinuous Galerkin methods exhibit multiple levels of parallelism: coarse-grain parallelism via mesh decomposition and fine-grain parallelism via single instruction, multiple data within a mesh element where element-local degrees-of-freedom are coupled. In this work, the MPI programming model is employed for coarse-grain parallelism and OCCA



Fig. 1. Turbulent wake structure of a wind turbine containing 1.6 billion degrees-of-freedom (8 billion unknowns). This simulation used the CPU implementation of this work on 20,000+ CPU cores.

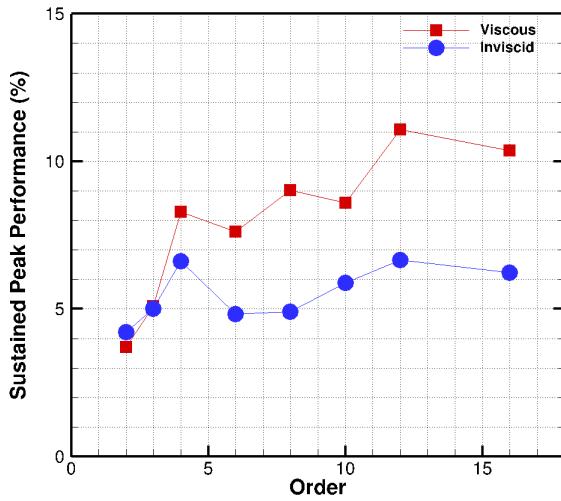


Fig. 2. Sustained peak performance of CartDG on Intel Xeon E5-2697V2 processors for various solution orders.

[16] is utilized for fine-grain parallelism. We select the CUDA-backend within OCCA to execute all computational kernels on NVIDIA V100 GPUs. OCCA provides a kernel language known as OKL, a simple extension of the C-language, enabling explicit architecture-independent fine-grain parallelized code. The sample code shown illustrates the spatial residual volume kernel (second term) from (4) using OKL. Note the similarity of the OKL-based code to the C programming language interlaced with simple decorators, e.g. @kernel, @exclusive, @shared, @outer, @inner.

The discontinuous Galerkin method discretizes mesh elements into multiple solution modes per variable which depend on the solution polynomial degree established a priori by the user. For example, in three-dimensional space, suppose the polynomial degree $p = 6$, then the number of modes per conservative variable within an element is $(p+1)^3 = 343$. To map the discretization to the hardware, we assign one hardware thread per mode, which is seen in the code sample indicated by the `cubeThreads` definition. The `@inner` attribute assigns a thread to each index (i, j, k) mode.

V. PERFORMANCE RESULTS: MIT SATORI

The GPU implementation of all computational kernels of CartDG is tested on the MIT Satori supercomputer. Satori is composed of 64 IBM Power9 nodes, each containing four

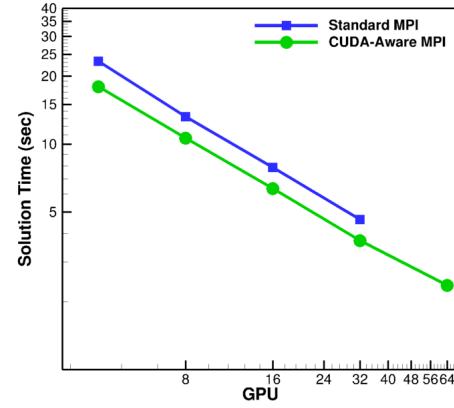


Fig. 3. Parallel strong scalability of CartDG on MIT Satori using up to 64 GPUs. A 29% speedup is achieved at 4 GPUs contained in a single node and a 24% speedup is achieved at 32 GPUs when using GPUDirect-enabled CUDA-Aware MPI compared to standard MPI.

TABLE I
CUDA-AWARE MPI COMMUNICATION COMPARISON TO REGULAR MPI COMMUNICATION USING NVIDIA V100 GPUs.

MIT Satori: CUDA-Aware MPI
Problem Size: 452,984,832 DOF.

GPUs	Solution Time (sec)		
	ON	OFF	Speedup
4	17.94	23.22	1.29x
8	10.60	13.22	1.25x
16	6.34	7.87	1.24x
32	3.73	4.62	1.24x

NVIDIA V100 GPUs and two AC-922 22-core processors. The communication interconnect contains NVIDIA GPUDirect, which enables CUDA-Aware MPI communication protocols over 50 GB/s NVLink fabric linking the GPUs together within a node. The problem chosen for performance testing consists of a periodic domain containing $128 \times 128 \times 128$ mesh elements and solution polynomial degree of $p = 6$, totaling 452,984,832 degrees-of-freedom (DOF) (2,264,924,160 unknowns).

Figure 3 demonstrates the parallel strong scaling performance of the implementation comparing GPUDirect communication to non-CUDA-Aware MPI communication. As seen in the figure, the solver scales well to 16 nodes. Table I demonstrates that the GPUDirect CUDA-Aware MPI provides a 29% speedup using a single node and a 24% speedup across multiple nodes.

```

1 #define cubeThreads \
2     for (int k = 0; k < p_TM; ++k; @inner(2)) \
3         for (int j = 0; j < p_TM; ++j; @inner(1)) \
4             for (int i = 0; i < p_TM; ++i; @inner(0))
5
6 /* Inviscid Flux Volume Kernel—Total Operations: nelements*(38*N^3 + 30*N^4) */
7 @kernel void residual_inviscid_volume_occa(
8     /* solution */ @restrict const double *Q,
9     /* residual */ @restrict double *R,
10    /* basis functions */ @restrict const double *dphiw,
11    /* mesh element count */ const int nelements,
12    /* ratio of specific heats minus one */ const double gml,
13    /* geometry factor */ const double vgeo){
14     for (int e = 0; e < nelements; ++e; @outer) {
15         const int tm_nf = p_TM*p_NF; // p_NF: # of fields = 5
16         const int tm_tm_nf = p_TM*tm_nf; // p_TM: # of modes = p+1
17         const int nf_tm_ALL = p_TM*tm_tm_nf;
18         const int elem_idx = e*nf_tm_ALL;
19
20         @exclusive double r_Q[p_NF]; // thread-local solution
21         @exclusive double r_R[p_NF]; // thread-local residual
22
23         @shared double s_fluxX[p_NQP][p_NQP][p_NQP][p_NF]; // s_fluxY, s_fluxZ
24         @shared double s_dphiw[p_TM][p_TM]; // basis with quadrature weights
25
26         // load solution and reset residual per thread memory
27         cubeThreads {
28             const int idx1 = i*p_NF + j*tm_nf + k*tm_tm_nf + elem_idx;
29             #pragma unroll p_NF
30             for (int n = 0; n < p_NF; ++n) {
31                 r_Q[n] = Q[idx1+n];
32                 r_R[n] = 0.0;
33             }
34             // fetch transpose(dphiw) to shared memory
35             if(k == 0) s_dphiw[i][j] = dphiw[j*p_TM+i];
36         } @barrier("localMemFence");
37
38         // volume flux: ops = 38*N^3
39         cubeThreads{
40             const double oneOrho = 1.0/r_Q[0]; // inverse density
41             const double u = r_Q[1]*oneOrho*vgeo; // dimensional x-velocity
42             const double v = r_Q[2]*oneOrho*vgeo; // dimensional y-velocity
43             const double w = r_Q[3]*oneOrho*vgeo; // dimensional z-velocity
44             const double p = (gml)*(r_Q[4] - 0.5*r_Q[0]*(u*u + v*v + w*w));
45             const double Ep = r_Q[4] + p;
46
47             // convective flux: x-component
48             s_fluxX[k][j][i][0] = r_Q[0]*(u);
49             s_fluxX[k][j][i][1] = r_Q[1]*(u) + p*vgeo;
50             s_fluxX[k][j][i][2] = r_Q[2]*(u);
51             s_fluxX[k][j][i][3] = r_Q[3]*(u);
52             s_fluxX[k][j][i][4] = Ep*(u);
53             // convective fluxes: y-component and z-component
54         } @barrier("localMemFence");
55
56         // residual per thread and update global residual memory: ops = 10*N^4
57         cubeThreads {
58             int idx1 = i*p_NF + j*tm_nf + k*tm_tm_nf + elem_idx;
59             #pragma unroll p_NQP
60             for (int qp = 0; qp < p_NQP; ++qp)
61                 #pragma unroll p_NF
62                 for(int n = 0; n < p_NF; ++n)
63                     r_R[n] += s_dphiw[i][qp]*s_fluxX[k][j][qp][n];
64                     r_R[n] += s_dphiw[j][qp]*s_fluxY[k][qp][i][n];
65                     r_R[n] += s_dphiw[k][qp]*s_fluxZ[qp][j][i][n];
66
67             #pragma unroll p_NF
68             for(int n = 0; n < p_NF; ++n) R[idx1+n] = r_R[n];
69         }
70     }
71 }
```

VI. PERFORMANCE RESULTS: ORNL SUMMIT

We test the same implementation on ORNL Summit². Summit has nearly the same architecture as MIT Satori but contains 6 NVIDIA V100 GPUs and 2 IBM AC-922 22-core processors per node. Summit contains 4,608 nodes totaling 27,648 GPUs and 202,752 CPU cores and has achieved 148.6 PetaFLOPS in the LINPACK benchmark [38], [39].

A. Strong Scalability

We perform a second parallel strong-scalability test on ORNL Summit using a $512 \times 512 \times 768$ mesh composed of $p = 6$, seventh-order accurate elements, totaling 69,055,021,056 degrees-of-freedom (345,275,105,280 unknowns). This problem size is chosen to give approximately 11 million degrees-of-freedom per GPU when using all GPUs on 1,024 nodes. The strong-scaling test measures the wall-clock time to solve 100 Low-Storage Explicit Runge-Kutta 5-stage 4th-order (LSERK45) time steps. The benchmark used 128, 256, 512, 920, and 1024 nodes. Figure 4 displays the scaling results. We note that these results do not utilize CUDA-Aware MPI; the information required by neighboring cores is first transferred to the host CPU, exchanged via MPI, then transferred to the GPU. Table IV tabulates the wall-clock times at various node counts for the CPU and GPU implementations.

TABLE II

GPU KERNEL PERFORMANCE FOR 512 NODES (NON-CUDA-AWARE COMMUNICATION). PROBLEM SIZE: 69,055,021,056 DEGREES-OF-FREEDOM. (THEORETICAL PEAK ASSUMED 21.5 PETAFLOPS FOR 512 NODES).

ORNL Summit: GPU Performance			
Kernel	Time (sec)	PetaFlops	Achieved Theoretical Peak
Volume	1.77	4.84	22.5%
Surface	2.15	2.45	11.4%
Update-Project	6.00	0.49	2.3%
Communication	10.18	—	—
Overall	20.45	0.82	3.8%

TABLE III

CUDA-AWARE MPI COMPARISON TO REGULAR MPI ON ONE NODE USING SIX NVIDIA V100GPUS DEMONSTRATING A 40% SPEEDUP.

ORNL Summit: CUDA-Aware MPI
Problem Size: 21,337,344 DOF.

Single Node		Time (sec)	
Kernel	ON	OFF	
Volume	1.67	1.68	
Surface	2.04	2.04	
Update-Project	5.70	5.70	
Communication	0.65	5.13	
Overall	10.37	14.55	

B. GPU Performance

As demonstrated in the strong-scaling figure, the performance improvement is approximately 30x when using all six GPUs compared to using both CPUs per node only (using 44 cores per node). Table II demonstrates the computational

²These results were generated during early acceptance testing of ORNL Summit in 2018. GPUDirect MPI issues across multiple nodes were not fully resolved, thus, all multi-node results herein utilized non-CUDA-Aware MPI.

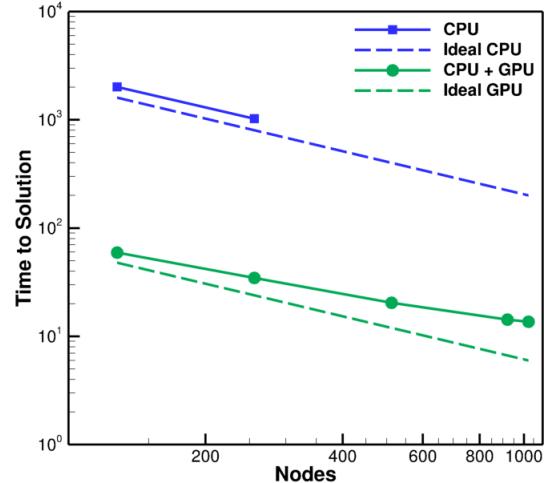


Fig. 4. Parallel strong scalability of CartDG on ORNL Summit using up to 1,024 nodes (6,144 GPUs/45,056 CPU cores).

kernel breakdown of the GPU simulation on 512 nodes. The volume integration kernel corresponding to the code sample achieved nearly 5 PetaFLOPs of performance on 512 nodes. Additionally, the communication overhead is large in comparison to the computational time. This is due to not utilizing the CUDA-Aware MPI on Summit, which allows for faster GPU-GPU communication on a node and overlap of computation and communication.

A communication benchmark is conducted using a single node with six GPUs interconnected with NVLink for a problem containing 21,337,344 DOFs. As shown in Table III, CartDG achieves an 8x speedup in communication time using the GPUDirect CUDA-Aware MPI on one node corresponding to a 40% overall speedup in time to solution. This result corroborates the results achieved on MIT Satori and illustrates a significant point-to-point communication improvement when using GPUDirect CUDA-Aware MPI. This contrasts the result found by [40] which tested a similar high-order solver on ORNL Summit indicating a *slow-down* in performance when using CUDA-Aware MPI.

VII. CONCLUSION

This work developed of a high-order discontinuous Galerkin method on Cartesian meshes for the use of heterogeneous architectures including GPUs by porting all computational kernels via OCCA. Performance benchmarks on MIT Satori and ORNL Summit demonstrated significant speedups achieved including an overall 30x time to solution over the CPU-only implementation. Further, the benefits of utilizing GPUDirect for point-to-point communication were shown for this application, improving the overall performance by 24% across multiple nodes and an 8x improvement in communication time within one node. If the CUDA-Aware MPI performance improvement holds at scale, as indicated by the results performed on MIT Satori, a GPU-to-CPU performance improvement of 40-50x may be expected.

TABLE IV
ORNL SUMMIT PARALLEL STRONG SCALABILITY COMPARISON OF CPU VS GPU (NON-CUDA-AWARE COMMUNICATION).

ORNL Summit: Parallel Strong Scalability						
Problem Size: 69,055,021,056 degrees-of-freedom.						
Each node contains 2 IBM POWER9 processors (44 cores per node) and 6 NVIDIA V100 GPUs.						
CPU	GPU	Speedup				
Nodes	Time (sec)	Scalability %	Nodes	Time (sec)	Scalability %	CPU/GPU
128	2011.2	100.0	128	59.37	100.0	33.9x
256	1021.9	98.43	256	34.75	85.42	29.4x
512	—	—	512	20.45	72.58	—
920	—	—	920	14.26	57.94	—
1,024	—	—	1,024	13.67	54.29	—

ACKNOWLEDGMENT

The authors acknowledge the MIT Research Computing Project and Oak Ridge National Laboratory for providing HPC resources that have contributed to the research results reported in this paper.

REFERENCES

- [1] T. Ben-Nun and T. Hoefer, “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” ACM Computing Surveys (CSUR) 52, no. 4 (2019): 1-43.
- [2] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, “Survey and benchmarking of machine learning accelerators,” arXiv preprint arXiv:1908.11348 (2019).
- [3] P. Messina, “The exascale computing project,” Computing in Science & Engineering 19, no. 3 (2017): 63-67.
- [4] A. Jameson, and D. J. Mavriplis, “Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh,” AIAA Journal 24, no. 4 (1986): 611-618.
- [5] D. J. Mavriplis, “Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes,” AIAA Journal 26, no. 7 (1988): 824-831.
- [6] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. J. Mavriplis, “CFD vision 2030 study: a path to revolutionary computational aerosciences,” NASA Technical Reports, 2014.
- [7] J. S. Hesthaven, and T. Warburton, Nodal discontinuous Galerkin methods: algorithms, analysis, and applications. Springer Science & Business Media, 2007.
- [8] F. D. Witherden, A. M. Farrington, and P. E. Vincent, “PyFR: An open source framework for solving advectiondiffusion type problems on streaming architectures using the flux reconstruction approach,” Computer Physics Communications 185, no. 11 (2014): 3028-3040.
- [9] A. Klckner, T. Warburton, J. Bridge, J. S. Hesthaven, “Nodal discontinuous Galerkin methods on graphics processors,” Journal of Computational Physics, 228(21), pp. 7863-7882, 2009.
- [10] R. Gandham, D. Medina, and T. Warburton, “GPU accelerated discontinuous Galerkin methods for shallow water equations,” Communications in Computational Physics, 18 (1), pp. 3764, 2015.
- [11] A. Modave, A. St-Cyr, and T. Warburton, “GPU performance analysis of a nodal discontinuous Galerkin method for acoustic and elastic models,” Computers & Geosciences, 91, pp. 6476, 2016.
- [12] J. Chan, Z. Wang, A. Modave, J. F. Remacle, and T. Warburton, “GPU-accelerated discontinuous Galerkin methods on hybrid meshes,” Journal of Computational Physics, 318, pp. 142168, 2016.
- [13] A. Karakus, T. Warburton, M. H. Aksel, and C. Sert, “A GPU-accelerated adaptive discontinuous Galerkin method for level set equation,” International Journal of Computational Fluid Dynamics, 30 (1), pp. 5668, 2016.
- [14] A. Karakus, T. Warburton, M. H. Aksel, and C. Sert, “A GPU accelerated level set reinitialization for an adaptive discontinuous Galerkin method,” Computers & Mathematics with Applications, 72 (3), pp. 755767, 2016.
- [15] A. Karakus, N. Chalmers, K. wiryドowicz, and T. Warburton, “A GPU accelerated discontinuous Galerkin incompressible flow solver,” Journal of Computational Physics, 390, pp.380-404, 2019.
- [16] D. S. Medina, A St-Cyr, and T. Warburton, “OCCA: A unified approach to multi-threading languages,” arXiv preprint arXiv:1403.0968 (2014).
- [17] D. Medina, “OKL: A unified language for parallel architectures,” Ph. D. Dissertation, Rice University, 2015.
- [18] D. S. Abdi, L. C. Wilcox, T. C. Warburton, and F. X. Giraldo, “A GPU-accelerated continuous and discontinuous Galerkin non-hydrostatic atmospheric model,” The International Journal of High Performance Computing Applications, 33(1), pp.81-109, 2019.
- [19] D. S. Abdi, F. X. Giraldo, E. M. Constantinescu, L. E. Carr, L. C. Wilcox, and T. C. Warburton, “Acceleration of the IMplicitEXplicit non-hydrostatic unified model of the atmosphere on manycore processors,” The International Journal of High Performance Computing Applications, 33(2), pp.242-267, 2019.
- [20] B. Cockburn, G. E. Karniadakis, and C. Shu, Discontinuous Galerkin methods: theory, computation and applications. Vol. 11. Springer Science & Business Media, 2012.
- [21] A. Harten, P. D. Lax, and B. van Leer, “On upstream differencing and Godunov-type schemes for hyperbolic conservation laws,” SIAM review 25, no. 1 (1983): 35-61.
- [22] M. J. Brazell, A. C. Kirby, J. Sitaraman, and D. J. Mavriplis, “A multi-solver overset mesh Approach for 3D mixed element variable order discretizations,” AIAA Paper 2016-0053, 54th AIAA Aerospace Sciences Meeting, San Diego CA, January 2016.
- [23] A. C. Kirby, and D. J. Mavriplis, “High Fidelity Blade-Resolved Wind Plant Modeling,” SC17, 2017.
- [24] A. C. Kirby, Z. Yang, D. J. Mavriplis, E. P. Duque, and B. J. Whitlock, “Visualization and data analytics challenges of large-scale high-fidelity numerical simulations of wind energy applications,” AIAA Paper 2018-1171, 56th AIAA Aerospace Sciences Meeting, Kissimmee FL, January 2018.
- [25] K. Kara, A. C. Kirby and D. J. Mavriplis, “Hover Predictions of S76 Rotor Using a High Order Discontinuous Galerkin Off-Body Discretization,” AIAA Paper 2020-0771 Presented at the AIAA Scitech 2020 Conference, Orlando FL, January 2020.
- [26] A. C. Kirby, A. Hassanzadeh, D. J. Mavriplis, and J. W. Naughton, “Wind Turbine Wake Dynamics Analysis Using a High-Fidelity Simulation Framework with Blade-Resolved Turbine Models,” AIAA Paper 2018-0256, Wind Energy Symposium, Kissimmee FL, January 2018.
- [27] A. C. Kirby, M. J. Brazell, Z. Yang, R. Roy, B. R. Ahrabi, M. K. Stoellinger, J. Sitaraman, and D. J. Mavriplis, “Wind farm simulations using an overset hp-adaptive approach with blade-resolved turbine models,” The International Journal of High Performance Computing Applications 33, no. 5 (2019): 897-923.
- [28] A. Kirby, M. Brazell, J. Sitaraman, D. Mavriplis, “An Overset Adaptive High-Order Approach for Blade-Resolved Wind Energy Applications,” AHS Forum 72, West Palm Beach FL, May 2016.
- [29] A. P. Edmonds, A. Hassanzadeh, A. C. Kirby, D. J. Mavriplis, and J. W. Naughton, “Effects of Blade Load Distributions on Wind Turbine Wake Evolution Using Blade-Resolved Computational Fluid Dynamics Simulations,” AIAA Paper 2019-2081, 57th AIAA Aerospace Sciences Meeting, San Diego CA, January 2019.
- [30] D. Mavriplis, M. Long, T. Lake and M. Langlois, “NSU3D results for the second AIAA high-lift prediction workshop,” Journal of Aircraft 52, no. 4 (2015): 1063-1081.
- [31] M. K. Stoellinger, A. P. Edmonds, A. C. Kirby, D. J. Mavriplis and S. Heinz, “Dynamic SGS modeling in LES using DG with kinetic energy preserving flux schemes,” AIAA Paper 2019-1648, 57th AIAA Aerospace Sciences Meeting, San Diego CA, January 2019.
- [32] M. J. Brazell, J. Sitaraman, and D. J. Mavriplis, “An overset mesh

- approach for 3D mixed element high-order discretizations," Journal of Computational Physics 322 (2016): 33-51.
- [33] J. Sitaraman, D. Jude, and M. J. Brazell, "Enhancements to Overset Methods for Improved Accuracy and Solution Convergence," AIAA Paper 2020-1528, AIAA Scitech Forum, 2020.
 - [34] A. C. Kirby, D. J. Mavriplis, and A. M. Wissink, "An Adaptive Explicit 3D Discontinuous Galerkin Solver For Unsteady Problems," AIAA Paper 2015-3046, 22nd AIAA Computational Fluid Dynamics Conference, Dallas TX, June 2015.
 - [35] Ma. J. Brazell, M. J. Brazell, M. K. Stoellinger, D. J. Mavriplis, and A. C. Kirby, "Using LES in a Discontinuous Galerkin method with constant and dynamic SGS models," AIAA Paper 2015-0060, 53rd AIAA Aerospace Sciences Meeting, Kissimmee FL, January 2015.
 - [36] A. C. Kirby, "Enabling High-order Methods for Extreme-scale Simulations," University of Wyoming, 2018.
 - [37] M. J. Brazell, A. C. Kirby, and D. J. Mavriplis, "A High-order Discontinuous-Galerkin Octree-Based AMR Solver for Overset Simulations," AIAA Paper 2017-3944, 23rd AIAA Computational Fluid Dynamics Conference, Denver CO, June 2017.
 - [38] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," Concurrency and Computation: practice and experience 15, no. 9 (2003): 803-820.
 - [39] J. A. Kahle, J. Moreno, and D. Dreps, "2.1 Summit and Sierra: Designing AI/HPC Supercomputers," In 2019 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 42-43. IEEE, 2019.
 - [40] J. Romero, J. Crabbill, J. E. Watkins, F. D. Witherden, and A. Jameson, "ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method," Computer Physics Communications (2020): 107169.