CMT-bone – A Proxy Application for Compressible Multiphase Turbulent Flows

Tania Banerjee[1], Jason Hackl[2], Mrugesh Sringarpure[2], Tanzima Islam[3],
S. Balachandar[2], Thomas Jackson[2] and Sanjay Ranka[1]

[1]Department of Computer Information Science and Engineering, University of Florida
[2]Department of Mechanical and Aerospace Engineering, University of Florida
[3]Lawrence Livermore National Laboratory
[3]*islam3@llnl.gov*
[1,2]{*tmishra, jason.hackl,mrugeshs, bala1s, tlj, ranka*}*@ufl.edu*

***Abstract*—CMT-nek is a solver of the Navier-Stokes equations for compressible multiphase flows being developed at University of Florida. The objective of this application is to perform high fidelity, predictive simulations of particle laden explosively dispersed turbulent flows under conditions of extreme pressure and temperature. CMT has many environmental, ecological, industrial and national security applications. The physical processes underlying these applications are complex and cover a very wide range of spatial and temporal scales.**

CMT based applications require enormous computing power and are expected to be deployed on petascale and exascale supercomputers. They also have a very rich set of computation and data access patterns that are representative of other scientific applications. Hence, they have the potential to serve as an important benchmark for next generation supercomputers.

In this paper[1], we describe CMT-bone that serves as a proxy-app for CMT-nek. Proxy applications provide a simplified framework for computational scientists and system designers to investigate new node architectures, programming approaches algorithmic design choices and optimization techniques and is an important aspect of co-design. CMT-bone encapsulates key data structures and key compute and communication operations in CMT-nek and fully retains the workflow of CMT-nek, however, it simplifies on the number of variables allocated and the number of compute and communication operations performed at each step of the work. The reduction in compute and communication operations are kept proportional to avoid undue bias towards a particular operation. This is shown by a validation of the proxy-app with the main using the VERITAS tool developed at LLNL.

I. Introduction

CMT-nek is a proposed solver of the Navier-Stokes equations for compressible multiphase flows. The objective of this application is to perform high fidelity, predictive simulations of particle laden explosively dispersed turbulent flows under conditions of extreme pressure and temperature. The physical processes underlying explosive dispersal are complex and cover a very wide range of temporal and spatial scales. CMT has many environmental, ecological, industrial and national security applications. Medical applications such as needleless drug delivery, lithotripsy and micro-bubble-enhanced ultrasound imaging rely upon compression wave or shock interaction with particles and bubbles. CMT dominates the behavior of natural and ecological flows such as explosive volcanic eruptions, supernovae, and dust explosions in coal mines and grain silos. In many applications of national defense and security CMT plays an important role in our ability to accurately predict and control explosive dispersal of particles.

CMT-nek leverages Nek5000, an open-source spectral element based computational fluid dynamics code developed at Argonne National Laboratory for simulating unsteady incompressible fluid flow with thermal and passive scalar transport [1]. Nek5000 is a highly scalable code, with demonstrated strong scaling to over a million MPI ranks on ALCF BG/Q Mira[2]. It is, however, limited to low speed flows by its formulation and discretization. As a simulation workhorse, CMT-nek is expected to facilitate fundamental breakthroughs and development of better (physics-informed) models and closures for compressible multiphase turbulence. CMT-nek based applications will require enormous computing power and are expected to be deployed petascale and exascale supercomputers where energy and thermal issues are as paramount as the overall performance. They also have a very rich set of computation and data access patterns that are representative of other scientific applications. Hence, they have the potential to serve as an important benchmark for next generation supercomputers where multiobjective (performance, energy, and thermal isssues) optimization may be important.

CMT-nek offers a very rich set of computation and data access patterns representative of complex scientific applications:

1) Domain decomposition: CMT-nek uses discontinuous Galerkin methods to accomplish domain decomposition and hence parallelism. Groups of elements are distributed across processors even though an element is never subdivided. As a result, the dominant communication pattern is nearest neighbor point to point communication (with up to 26 surrounding processes) to exchange surface fluxes and migrating particles. Thus CMT-nek provides an interesting use case for studying domain decomposition and communication scaling on homogeneous as well as heterogeneous platforms.
2) Unique computation structures: The solution of the underlying solvers requires partial spatial derivative computation which involves dense matrix-matrix multiplication for three dimensional domains. Here data is accessed

[1]A preliminary version of this work will appear in Tania Banerjee, Jason Hackl, Mrugesh Sringarpure, Tanzima Islam, Siva Balachandar, Thomas Jackson and Sanjay Ranka, CMT-bone: A Proxy Application for Compressible Multiphase Turbulent Flows, Proceedings of HiPC 2016.

not only in unit strides but in strides of N and N^2, where N is the number of grid points in a cubic spectral element in any one direction. Another example of a computation intensive kernel is the dealiasing kernel that interpolates all conserved and primitive variables from a $N \times N \times N$ grid to a finer $M \times M \times M$ ($M > N$) grid to evaluate fluxes. Like derivative computation, dealiasing uses matrix-matrix multiplications. Some of the other computation patterns are vector updates, dot products and matrix-vector multiplications. The low computation to data access ratio of CMT-nek also makes it a representative application as it is a common characteristic across many scientific applications.

3) Coupled data structures: The solution to the underlying equation results in interaction between the particles and the spectral elements. Determining new location of particles following advection of particles as a result of forces exerted by the medium, calculating particle velocity and force field (comprising of drag, stress gradient, inviscid and viscous components) at all particle locations involve extensive computations per particle. Particle movement from one iteration to the next result in challenging load balancing issues as the number of particles assigned to a given element may be in-general highly non-uniform. Scalable algorithms for point particle tracking presents us a set of comprehensive computational and memory access patterns as well as an opportunity to study communication and load balancing algorithms.
4) Variable compute intensiveness: Different portions during a single iteration of the application utilize cpu, cache and memory modules variably. This makes it a challenging application to tradeoff peformance, energy and thermal objectives that are expected to be very important for next generation supercomputers.

Thus, CMT-nek has the potential to become an important application to evaluate current and future architectures.

In this paper, we describe CMT-bone, a proxy-app that is based on CMT-nek. Proxy applications are small, portable codes that capture the important computational aspects of the larger scientific codes known as parents. Testing, simulating or optimizing these codes is significantly easier than changing parent codes, and therefore proxy apps are used by computational scientists, software developers, and system designers to work together in understanding the required features and tradeoffs in developing algorithms and underlying hardware and system software. It is essential therefore to validate that a proxy application indeed encapsulates the desired characteristics of the parent. Examples of large scientific codes that have been studied and optimized using proxy applications are described in Section VI.

We have two important contributions in this paper.

1) We created a proxy application for a very complex physics code capable of modeling compressible multiphase turbulent flows with dispersed particles. CMT-bone uses different memory access patterns for volume data, surface data and particles. Hence, using CMT-bone we can evaluate a variety of architectures, communication network bottlenecks, and scalability issues. Additionally, the particle simulation capabilities present in CMT-bone will also allow us to study and model a variety of dynamic load balancing algorithms for particle tracking.
2) We have validated our proxy application against the parent application using VERITAS and present the validation results for specific regions of the code.

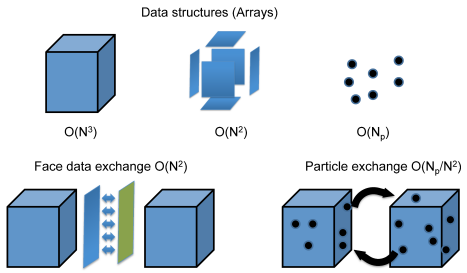


Fig. 1: Key datastructures and communication operations in CMT-nek

The rest of the paper is described as follows. In Section II, we describe our target application CMT-nek and its implementation details. Section III presents the implementation details of CMT-bone. The experimental results are presented in Section IV which validates CMT-bone as a proxy app for CMT-nek. Finally, conclusions are given in Section VII.

II. CMT-NEK

CMT-nek is a proposed solver of the compressible Navier-Stokes equations for multiphase flows. The objective of this application is to perform high fidelity, predictive simulations of particle laden explosively dispersed turbulent flows under conditions of extreme pressure and temperature. The three-dimensional Euler equations of gas dynamics are written in the form of a conservation law (Equation (3), Appendix A). These equations are reduced to a system of ordinary differential equations for the conserved variables U at each of $N_x \times N_y \times N_z$ quadrature nodes on each of N_{el} finite elements via the discontinuous Galerkin spectral element method. The end result appears in semi-discrete form of Equation (40). The two terms on the right hand side of Equation (40) must be evaluated on each element. The first term contributes to $\partial \mathbf{U}/\partial t$ at every quadrature point on the grid, and the second term is an additional effect added only to quadrature points lying on element faces. Figure 1 shows a schematic representation of important arrays and communication operations in CMT-nek. Most of the variables required for evaluating the terms in Equation (40) are stored in multidimensional arrays. All the variables associated with the flow can be classified into two categories - volume data and surface data.

Since CMT-nek is written in Fortran 77, multidimensional column-major arrays are the only data structure available. Table I lists array names, their dimensions, and the physical quantities they store. N_x, N_y, N_z are the number of GLL quadrature nodes in each direction of a given element Ω_e

TABLE I: Storage of volume data in CMT-nek.

Variable	Dimension	Physical quantity
vtrans(...,1)	$N_x, N_y, N_z, N_{el}, 1$	gas density ρ_g
vtrans(...,2)	$N_x, N_y, N_z, N_{el}, 1$	constant-pressure specific heat c_{pg}
vtrans(...,3)	$N_x, N_y, N_z, N_{el}, 1$	constant-volume specific heat c_{vg}
t(...,1)	$N_x, N_y, N_z, N_{el}, 1$	temperature T_g
pr	N_x, N_y, N_z, N_{el}	pressure p_g
vx, vy and vz	N_x, N_y, N_z, N_{el}	velocity components u_g, v_g, w_g
U	$N_x, N_y, N_z, 5, N_{el}$	conserved unknowns (Equation (4))
rx	$M_x, M_y, M_z, 9, N_{el}$	weighted metrics $\mathcal{M}_M\mathcal{I}\left[\frac{1}{J}\frac{dr_i}{dx_j}\right]$
vxd, vyd and vzd	M_x, M_y, M_z, N_{el}	$\mathcal{I}\mathbf{u}, \mathcal{I}\mathbf{v}, \mathcal{I}\mathbf{w}$

In nek5000. N_{el} is the number of elements on a given MPI rank. vtrans and t are arrays of rank 5 whose first four dimensions store volume data for all elements on a given MPI task, and whose outermost dimension correponds to one of ldimt passive scalars, such as gas density, specific heat or temperature, as shown in the Table I. We have repurposed them for thermodynamic quantities (Equations (10),(11)).

Surface data are stored contiguously for all faces on each element. Typically these variables are also rank-4 arrays with dimensions $(N_x, N_z, N_{faces}, N_{el})$. Here N_{faces} is the number of bounding surfaces $(3D)$ or edges $(2D)$ of an element. $N_{faces} = 6$ and 4 for $3D$ and $2D$ elements, respectively.

All the point particles are stored in a rank-2 array with dimensions (Pt_{prop}, N_{pt}), where N_{pt} is the number of particles owned by an MPI rank and Pt_{prop} is the number of particle properties being tracked. Typically we track information like particle location $(x^{(p)}, y^{(p)}, z^{(p)})$, particle velocity (u_p, v_p, w_p), fluid velocity at the particle location $(u^{(g)}, v^{(g)}, w^{(g)})$ and forces on the particles $(Fx^{(p)}, Fy^{(p)}, Fz^{(p)})$. This would correspond to 12 particle quantities. In general we will track additional quantities such as particle temperature, heat transfer, etc., and as a result we envision $Pt_{prop} > 20$.

It is convenient to explain the workflow and implementation of CMT-nek by using Equation (40) as a reference. The left-hand side of Equation (40) is explicitly discretized by a third-order total-variation-diminishing Runge-Kutta scheme[3]. This scheme advances the solution of the conservation laws from time t_n to $t_n + \Delta t$ by computing the terms on right-hand-side of Equation (40) explicitly. Figure 2 shows the workflow of computing these right-hand-side terms. After certain preliminary computations, the surface and volume integral terms are evaluated and the result is passed on to the time integration step. Each component of this work flow is further expanded to show the most intensive compute operations along with the communication operations in a sequence which represents the actual implementation.

A. Preliminary computations

Preliminary computations consists of several important steps that are necessary for evaluating surface and volume integrals. The workflow of this step is shown in Figure (3). Both, the surface and volume integral terms on the right-hand-side of Equation (40) need the flux vector $\mathbf{H}$ which is a function of conserved variables $\mathbf{U}$ and primitive variables $\rho_g, u_g, v_g, w_g, p_g$

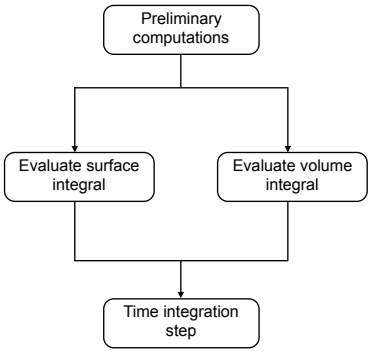


Fig. 2: A macro scale workflow of CMT-nek.

and T_g (see Equations (7),(8),(9)), grid metrics $\partial r_i/\partial x_j$ and interpolation operator $\mathcal{I}$. Since the current scope of CMT-nek does not include deforming and translating grids, the grid metrics and interpolation operators are computed once and stored for the rest of the simulation. Thus, the main objective of this block is to compute the primitive variables $\rho_g, u_g, v_g, w_g, T_g, p_g, a_g$, transport properties like λ, ν, κ and specific heats c_{pg}, c_{vg}. Figure 3 shows the steps in which these quantities are computed. Most of the computations in this block can be categorized as pointwise arithmetic with $O(N^3)$ computational workload, however, the nature and the number of arithmetic operations are dependent on the specific task. For example, the process of computing primitive variables like ρ_g, u_g, v_g, w_g involves a single pointwise division operation while the process of computing pressure p_g and temperature T_g will involve multiple pointwise addition, subtraction and division. It is important to note that when dealiasing is turned on, the velocity components are interpolated onto a fine mesh. This step is a matrix-vector multiplication with $O(N^4)$ computational workload. Also, it must be noted that the current CMT-nek framework enables users to implement equation of state and transport laws of their choice via user-defined subroutine userEOS and uservp.

B. Evaluate surface integral

After the primitive variables are computed, we can proceed to evaluate the surface integral term on on the right-hand-side of Equation (40). Figure 4 shows the sequence of steps involved in the evaluation of the surface integral term. Till the end of preliminary computations, all the variables are stored as volume data in a multidimensional array. Clearly, it will be cumbersome and inefficient to stride through grid points that lie on the surface of the elements or volume data multiple times to evaluate the surface integral term. Therefore, "$full\_to\_face$" mapping is generated at preprocessing which will extract the surface points from the volume data and store them contiguously in a separate "surface" array. Thus the first step in this block is to extract the surface points for all the necessary variables (primitive variables $\rho_g, u_g, v_g, w_g, T_g, p_g, a_g$, transport properties λ, ν, κ and specific heats c_{pg}, c_{vg}) and

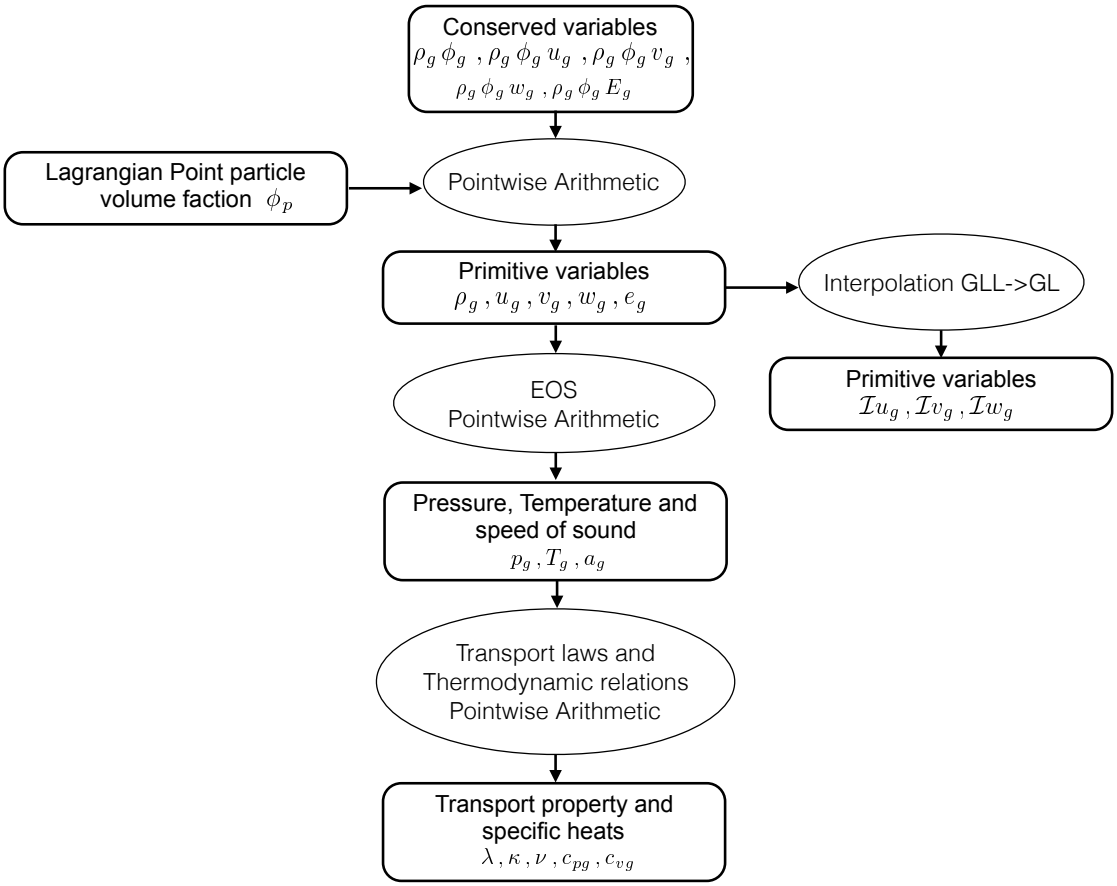


Fig. 3: Workflow of preliminary computations that precedes the evaluate surface and volume integral steps.

store them in a separate array so that all subsequent steps can be performed effieciently.

In discontinuous Galerkin method variables are permitted to have jumps (discontinuity) across the element faces. In other words, a gridpoint on any face will have two values associated with two elements who share that face. As a consequence of this, the "$full\_to\_face$" step provides only the surface data corresponding to all the faces on an element. This information is referred to as "Left state" in the workflow shown in Figure 4. Since nek5000 is based on unstructured collection of elements, an independent framework is in place that maps the connectivity of all the elements. Therefore, to obtain the surface data corresponding to the other element that shares the same face, one needs to read the connectivity map appropriately. At this stage it is important to emphasize that the computational domain is distributed and as a consequence the connectivity map has to be decoded across remote MPI ranks. To accomplish this, we utilize nek5000's, sophisticated communication framework along with the domain decomposition and element connectivity framework to obtain the surface data corresponding to the other element that shares the same face. Note that we refer to this information as the "right state" in the workflow. This process of getting the right state is referred to as "GS_op + Pointwise Arithmetic." Here the "Pointwise Arithmetic" step can be characterized as $O(N^2)$ computational workload.

Once the "left" and "right" states are available, we loop over all faces on all elements on a single MPI task. For each face, it interpolates all the variables on a single face to a finer grid of $M_x \times M_z$ points and evaluates the numerical flux $\mathbf{h}^*$ in Equation (39) at each of these surface nodes. It is to be noted that while $\mathbf{H}$ is the flux function, $\mathbf{h}^*$ represents the values of the flux function at the grid points, arranged into a computational vector. Our numerical flux is the Advection Upstream Splitting Method (AUSM+)[4]. This flux computation can be done for each point on the element faces independently. If a face has no neighbor, a boundary condition is applied instead. Once the fluxes are computed, they get projected back to the (N_x, N_z) GLL grid for a given face by tensor-nested matrix multiplication in two successive directions with the $N_x \times M_x$ matrix $\mathcal{I}_A^\top$. Once all faces are finished, they get mapped back to the cubic storage by "$face\_to\_full$" step.

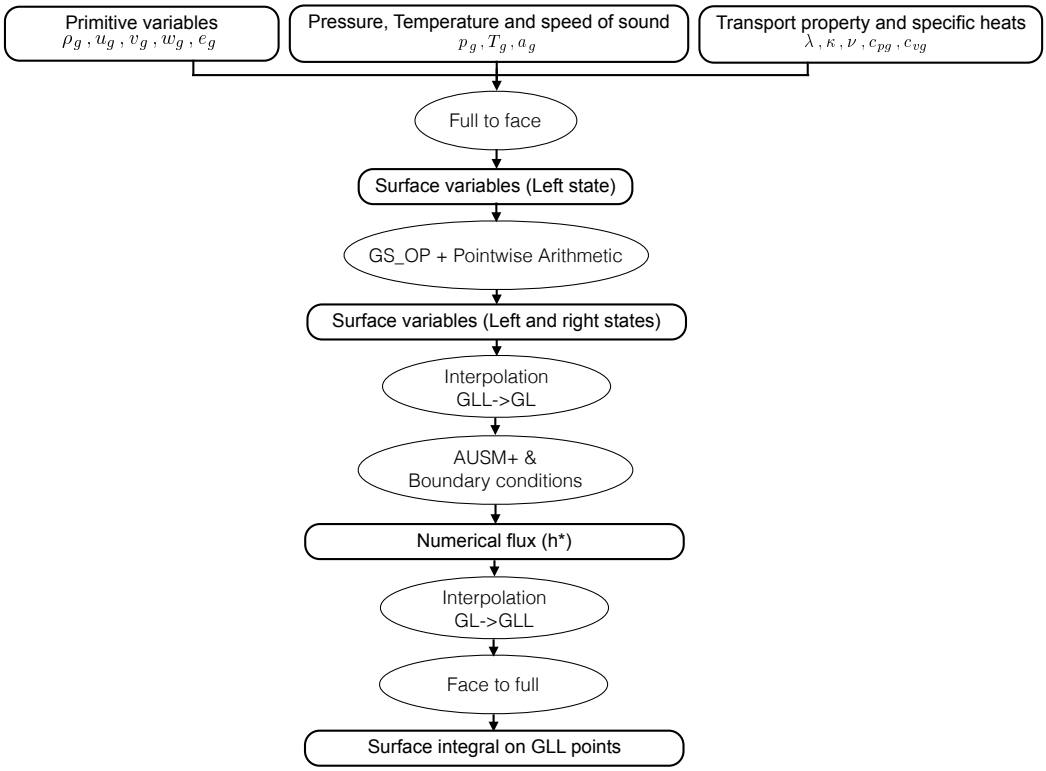


Fig. 4: Workflow of evaluate surface integral step

C. Evaluate volume integral

Next we evaluate the first term on the right-hand-side of Equation (40). All operations here will be performed on cubic data (N_x, N_y, N_z). When dealiasing is turned on, the conserved variables are interpolated on to the fine grid (M_x, M_y, M_z) and then combined with the primitive variables which are also interpolated on to the fine grid to get the flux vector $\mathbf{h}$. The process of computing the flux vector can be classified as "Pointwise Arithmetic" with $O(N^3)$ computational workload. To minimize the memory footprint of the code, $\mathbf{h}$ is only stored for one element, one equation at a time. Next the flux vector $\mathbf{h}$ is multiplied with the corresponding geometric metric vector "rx" and then the transpose of $\mathcal{D}$ is applied to it in each of the (r, s, t) directions. This step is of $O(M^4)$ computational workload when dealiasing is turned on and the grid is sampled onto a fine grid. Multiplication by the $N \times M$ matrix $\mathbf{J}^\top$ projects the right-hand-side volume-integral term back to the (N_x, N_y, N_z) grid. In this workflow, the computational workload of the interpolation steps is of $O(M \times N^3)$ for $GLL \rightarrow GL$ points and $O(N \times M^3)$ for $GL \rightarrow GLL$ points.

At the end, the result of the volume integral term is added to the contribution of the surface integral term to get the right-hand-side of Equation (40) for all equations and all elements. This result is then passed to the time integration step and the conserved variables are advanced to the next time step (stage in RK 3 time integration scheme).

After the conserved variable are updated, the control moves to the point particle tracking. In the current version of CMT-nek, Lagrangian point particle tracking is a user-defined source code that gets appended at compilation. Furthermore, the current code does not distribute the point particle force data back into Equation (40). In the point particle tracking code, we first interpolate the fluid velocity at the off-grid point particle location. Then the particle equations of motion (see Equation (14)) are solved and the point particles are moved to their new location with updated velocity. When the particles move to their new position, some of them may leave the spatial domain of one MPI rank and enter into the neighboring spatial domain owned by another MPI rank. At this stage, such particles are exchanged between corresponding MPI ranks so that all point particles are assigned to the appropriate MPI ranks. This concludes the point particle tracking and the control is passed back to the flow solver.

III. CMT-BONE

CMT-nek will be the primary simulation tool employed by the researchers at the Center to perform state-of-the-art simulations of compressible multiphase turbulence on some of the largest supercomputing facilities at the NNSA Tri-labs. Therefore, it is essential that the performance of CMT-nek is evaluated and existing algorithms tuned to ensure that these simulations are performed at the peak capacity of the code and the supercomputer. The use of parent application to evaluate the impact of various algorithmic design choices can be quite cumbersome, which drives the need to develop a representative proxy app that provides a simplified framework for computer scientists and system designers to investigate the

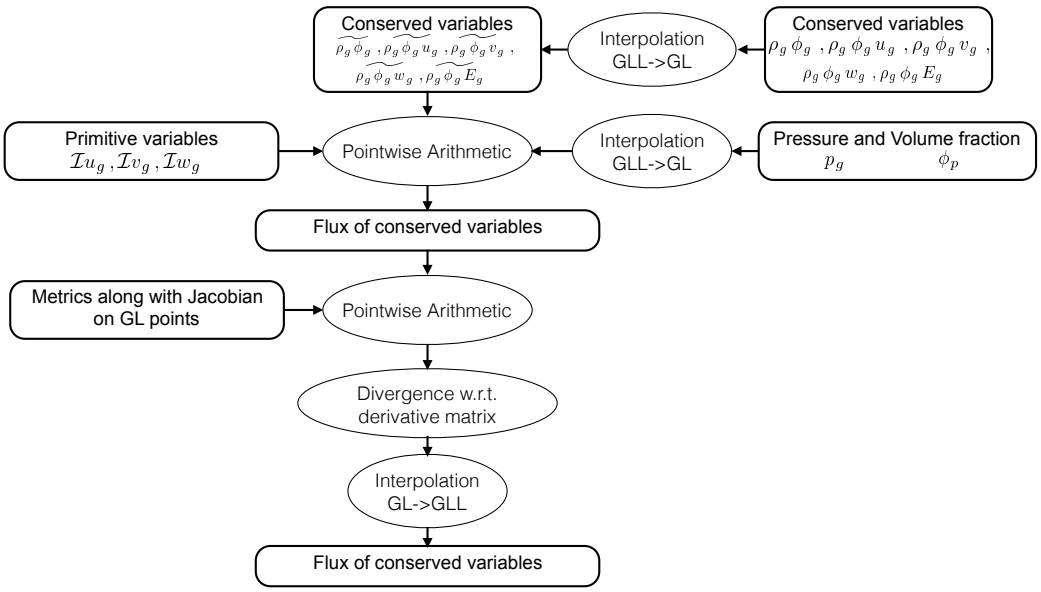


Fig. 5: Workflow of evaluate volume integral step

impact of node architectures and algorithmic design choices on the performance of the parent application.

CMT-bone will be a representative proxy application, if it encapsulates key data structures and key compute and communication operations in CMT-nek. From the detailed description of the CMT-nek implementation and the various workflows presented in the previous section, we can develop an abstraction of CMT-nek which will be a useful foundation on which CMTbone will be built. Figure 6 shows an abstract representation which highlights the important arrays, computations and communication operations in CMT-nek. To put this schematic in the context of the implementation details and the workflows, we can clearly see the following

- The evaluate surface integral described in II-B is represented by the first four blocks. After close inspection of Figure 4 and Figure 6, it is obvious that the "$full\_to\_face$" step is represented by "Volume to surface" block, "GS_op" step is represented by "Face data exchange" block. While the "computation for face points" is an abstraction of "AUSM+" and "Boundary condition" step in Figure 4. Finally the "$face\_to\_full$" step in Figure 4 is represented by "Surface to volume" block in Figure 6.
- The next two blocks - "Pointwise computation for volume points " and "Derivative computation volume points" represent the evaluate volume integral section described in II-C.
- The last four blocks represents the Lagrangian point particle tracking in CMT-nek. Note that the last block "Distribute point particle force on volume data" is currently under development and therefore not described in the earlier sections.

CMT-bone fully retains the workflow of CMT-nek; however, it simplifies on the number of variables allocated and the number of compute and communication operations performed at each step of the work. For example, CMT-bone implementation captures the basic steps in CMT-nek to solve Equation 24. As a result, CMT-bone uses the data structures described in the previous section to store volume and surface data. This first version of CMT-bone uses all the five conserved variables of CMT-nek; namely, mass, energy and the three components of momentum. CMT-bone also uses a subset of primitive variables used in CMT-nek and the primitive variables in CMT-bone are pressure, density and the three components of velocity. Particle processing in CMT-bone is identical to that in CMT-nek.

CMT-bone retains the key operations of CMT-nek. For example, CMT-bone processes operations described in Sections II-A, II-B, II-C. However, it trims out operations such as computation of diffusive fluxes, and treatment of viscous flows. In future versions of CMT-bone we will reduce the number of variables further with a caveat that the reduction in compute and communication operations are kept proportional to avoid undue bias towards a particular operation. To ensure that the mini-app represents the parent application in its most mature state, CMT-bone and CMT-nek development is done concurrently. As new capabilities are added and tested in CMT-nek, their abstractions will be added in CMT-bone.

Table II is a tabular representation of the implementation and CMT-nek workflow shown in Figure 6. Its last column gives the number of variables that are operated at each step in CMT-nek and the corresponding number in CMT-bone.

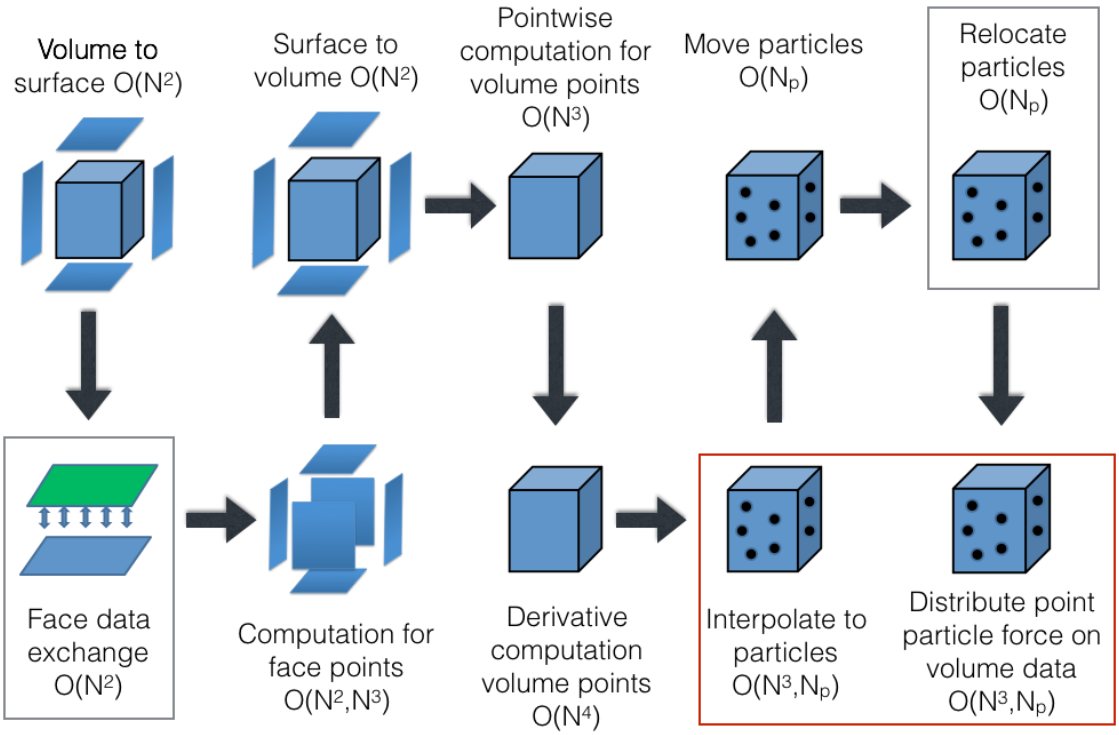


Fig. 6: CMT-nek and CMT-bone workflow that shows the sequence of key compute and communication operations on various data stuctures

TABLE II: Key subroutines in CMT-nek implementation and their corresponding description in the CMT-nek workflow shown in Figure 6.

step	CMT-nek subroutine	CMT-nek workflow description	number of variables	
			CMT-nek	CMT-bone
1	primitive_variables	*pointwise computation for volume points*	9	5
2	fillq	*Volume to surface*	15	10
3	gs_op	*face data exchange*	18	10
4	ausm	*computation for face points*	5	5
5	addfacetofull	*surface to volume*	5	5
6	*evaluate_conv_h*	*Pointwise compuation for volume points*	15	15
7	flux_div_integral	*Derivative compuation for volume points*	15	15
8	baryweights_findpts_eval	*Interpolate to particles*	3	3
9	update_stokes_particles	*Move particles*	3	3
10	crystal_tuple_transfer	*Relocate particles*	3	3
11	-	*Distribute point particle force on volume data*	-	-

IV. Validation Approach

We compared the performance of CMT-bone against that of CMT-nek in order to study which resource utilizations of CMT-bone are representatives of CMT-nek, if at all. We ran our experiments on Cab, a 431 Tflop Intel Xeon Linux cluster with $1,296$ nodes. Each node on Cab is a dual socket Sandy Bridge processor with 32 GB memory, 20 MB shared L3 cache across 8 cores on each socket, 32 KB L2 cache private to each core, and L1 data and instruction caches of size 32 KB each.

A. Input Parameters

CMT-nek flow constitutes of four important stages as listed below:

1) Point computations: This is where the primitive variables are computed for each grid point and is encapsulated in **`point_compute_kernel`**.
2) Communication step: This is where surface data is exchanged between neighbors and is considered the most communication intensive. The code in this region is encapsulated in **`comm_kernel`**.
3) Computation step: This is where fluxes and spatial partial derivatives of the fluxes are computed. This is the key computation step in CMT-nek and is referred to as **`compute_kernel`**.
4) Processing particles: This is where particles are initialized, randomly distributed and tracked. This region is represented in **`particles_kernel`**.

TABLE III: Resource groups for Intel Xeon

Resource Group	Intel Xeon
Floating point unit	FP
Branch unit	BR
Prefetch events	PREFETCH
L1 cache	L1
L2 cache	L2
Last level L3 cache	L3
Translation look aside buffer	TLB
Memory	MEM
Remote socket traffic	OFFCORE
PCI express bus	PCIE

TABLE IV: Workloads used in our experiments

Number of Elements	Workload Id	Polynomial Order
512	1	5
	2	7
	3	9
	4	11
	5	13
	6	15
1024	7	5
	8	7
	9	9
	10	11
	11	13
	12	15

CMT-bone captures these stages as well. The **particles_kernel** is identical in CMT-nek and CMT-bone. Hence we did not validate the **particles_kernel**.

We performed on-node performance validation of the remaining kernels by scaling both applications from 1 through 16 processes on one core of Cab. We varied workload by changing the polynomial order to $5, 7, 9, 11, 13, 15$[2] and the number of spectral elements 512, and 1024.

B. Resource Groups

For on-node performance comparison of CMT-nek and CMT-bone, we used the proxy application validation framework VERITAS developed by Islam et al. in [5]. VERITAS categorizes hardware performance counters on an architecture into high-level resource groups and uses machine-learning techniques to attribute the efficiency loss with scale of both the applications to their utilization of high-level resources on Intel Xeon. Table III presents the high-level resource groups on Intel Xeon.

C. Validation Methodology

VERITAS defines validating a proxy application as comparing how closely the data subspaces of two applications match. The validation process has two steps: *(1)* identify which resource utilization behaviors impact scalability in CMT-nek; *(2)* identify which resource utilization behaviors are covered by CMT-bone and how well.

1) Identifying important resources: To identify the resources that are the bottlenecks in the parent application, VERITAS uses efficiency loss with scale as a reference. Efficiency loss increases as an application uses more cores on a multi-core machine. Since every performance event has a non-zero penalty, VERITAS uses the total penalty incurred by a performance metric to compare against efficiency loss of the application. In fact, the more closely the penalty of a performance metric follows the growth of efficiency loss as an application scales, the more important that metric is (and in turn the corresponding resource is) in predicting efficiency loss of the application.

Based on all the performance metrics collected from CMT-nek workloads, VERITAS identifies a sparse subset of performance metrics that can describe the characteristics of efficiency loss and then uses these metrics to build a predictive linear model. Using the inferred sparse model, VERITAS combines per-metric importances to estimate resource-wise importances. Thus, VERITAS enables users to obtain a broad understanding of which hardware resources lead to performance bottlenecks in an application, as well as the relative importance of these resources in determining performance of the parent application. This knowledge helps the users to develop a proxy application which nicely covers the hardware resources that are important in determining the behavior of the parent application.

The underlying assumption in sparse representations is that the data is drawn from a union of low-dimensional subspaces, which need not be completely disjoint or orthogonal. VERITAS employs the idea that a complex pattern can be effectively decomposed into a small set of diverse, elementary patterns. Mathematically, this assumption is expressed as follows. Given a data sample $\mathbf{y} \in \mathbb{R}^n$ and a dictionary of K representative patterns $\mathbf{D} \in \mathbb{R}^{n \times K}$, the sparse representation $\mathbf{a} \in \mathbb{R}^K$ can be obtained as

$$\min_{\mathbf{a}} \|\mathbf{y} - \mathbf{D}\mathbf{a}\|_2^2 \text{ s.t. } \|\mathbf{a}\|_0 \leq \kappa. \quad (1)$$

where $\|.\|_0$ denotes the ℓ_0 norm that counts the total number of non-zero entries in a vector, $\|.\|_2$ is the ℓ_2 norm, and κ is the desired sparsity. In our case, $\mathbf{y}$ is the efficiency loss (or loss in runtime), n is the number of workers (or the number of workloads), the dictionary $\mathbf{D}$ is the collection of K performance metrics. Note that, this optimization problem solves for the sparsest set of performance metrics that can reconstruct the efficiency loss. In particular, VERITAS uses the Orthogonal Matching Pursuit (OMP) algorithm proposed in [6] to select that smallest diverse subset of metrics to describe the performance loss.

VERITAS uses two parameters to tune the complexity and the sparsity of the model it builds to explain efficiency loss. The complexity parameter controls how sophisticated the model needs to be in order to explain the characteristics of performance data better. Higher value for this parameter means more number of resources will be considered in order to explain the performance loss as the applications scale. This parameter is used to identify higher-order bottlenecks of an application, whose resource utilization that may become a problem when the immediate bottleneck is removed. However, the sparsity parameter that controls how many counters are highly correlated with the performance loss are also considered per resource. The higher the value of the sparsity parameter, the more dense the model is, and hence the model might select a large number of counters that only contribute marginally. The

[2]Odd N_x ensures even polynomial order and integral whole numbers for M, the number of points for overintegration.

idea is to build a simple model with high enough sparsity so that only a small number of performance metrics are dominant causes of efficiency loss and selected by the model.

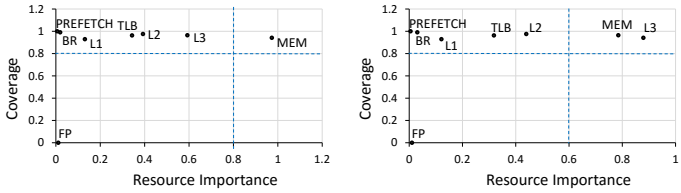
2) Comparing proxy to parent: The problem of comparing proxy to parent can be posed as measuring the compatibility between the two feature spaces defined by a set of performance metrics. In statistical modeling, it is common to assume that underlying data distribution can be described using a fewer degrees of freedom. VERITAS uses Principle Component Analysis (PCA) to identify the directions of maximal variance and projects the data onto the subspace. Consequently, instead of comparing the high-dimensional feature spaces directly, VERITAS compares their corresponding low-dimensional representatives. A linear subspace can be conveniently represented using its basis $\mathbf{B} \in \mathbb{R}^{C \times d}$, where C is the dimensionality of the data, and d is the dimensionality of the subspace. The collection of all $d-$dimensional linear subspaces form the Grassmannian $G(d, C)$, a smooth Riemannian manifold, which allows effective geometric and statistical analysis of subspaces. VERITAS constructs the geodesic curve between the two subspaces and estimates a dissimilarity measure between them using the computed dissimilarity. Note that, for a given resource group, the lower the resource subspace dissimilarity measured, the higher the compatibility between the two applications.

VERITAS considers the two applications to be compatible, with respect to a given resource group, when (a) the two subspaces are geometrically well-aligned, and (b) the data in the projected subspaces are similarly distributed. The first can be obtained by computing the principle angles between the basis vectors along the principle components. In cases where there are multiple workloads, the dissimilarity measure is obtained as the average of the individual workloads. VERITAS defines the *Coverage* measure for each resource r as

$$Coverage_r = 1 - \mathbf{dissimilarity}_r, \forall r. \quad (2)$$

D. How to Interpret Results

Using Figure 8a we will explain the various characteristics of the charts generated by VERITAS. Along the X-axis, the importance of resources in predicting efficiency loss is plotted (between 0 and 1 where 0 means the lowest and 1 means the highest importance), and along the Y-axis, the quality of match (or Coverage) for each resource is plotted (again between 0 and 1 where 0 means "does not cover" and 1 means "covers 100% behavior"). $Coverage \geq 0.8$ means that on average the proxy covers more than 80% of the utilization behavior of a certain resource. Similarly, Resource Importance ≥ 0.8 means that on average a resource utilization behavior affects scalability of an application with more than 80% probability. In our analysis, we assume $Coverage \geq 0.8$ as good coverage, and Resource Importance ≥ 0.8 as important to cover (hence the dotted line through 0.8). However, this is a user defined threshold that can be passed to VERITAS as an input. VERITAS also generates per workload charts. Such a chart shown in (Figure 9b) shows the coverage behavior for the important resources in red for ease of identification.



(a) $\kappa = 10$ (b) $\kappa = 15$

Fig. 7: Results for different κ for $\tau = 10$

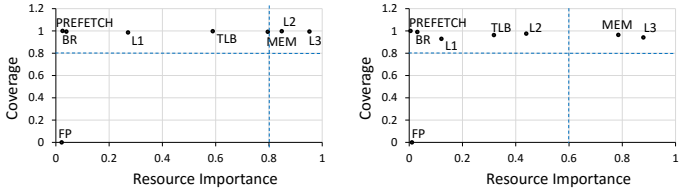
V. Results

In this section, we first present the rational behind the choice for the sparsity and complexity parameters in Section V-A followed by the validation results from our experiments separately for the three different kernels in code in Sections V-B through V-D.

A. Determining Parameters:

In order to determine the model parameters κ(sparsity) and τ (model complexity) for a kernel being validated, we vary both these parameters and identify a reasonable setting that can be used for the rest of the analysis for that kernel. Due to lack of space, we only present the results of the **`compute_kernel`** here. The results of the other kernels are similar and not presented in this paper.

Figure 7 shows a subset of results for a combination of different values for parameters κ and τ. We observed that decreasing sparsity (increasing κ, i.e. involving more performance metrics for a resource) does not significantly change the importance of resources. Also, using $\tau = 10$ or more encourages OMP to include different resources in the solutions. This phenomenon results in L3 resource becoming important. Higher values of τ or κ do not change the importance of the resources in any significant way. Hence, for the rest of the experiments with **`compute_kernel`**, we used $\kappa = 15$ and $\tau = 10$.

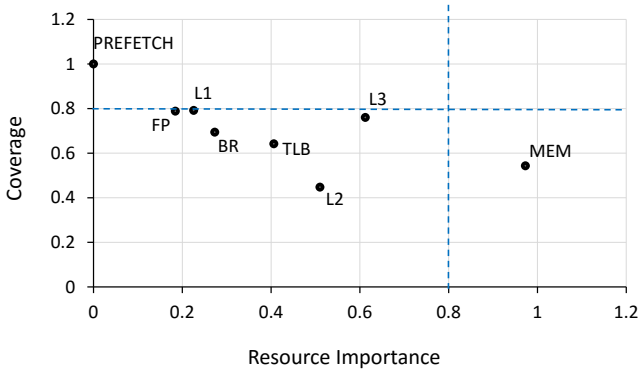


(a) **`point_compute_kernel`** (b) **`compute_kernel`**

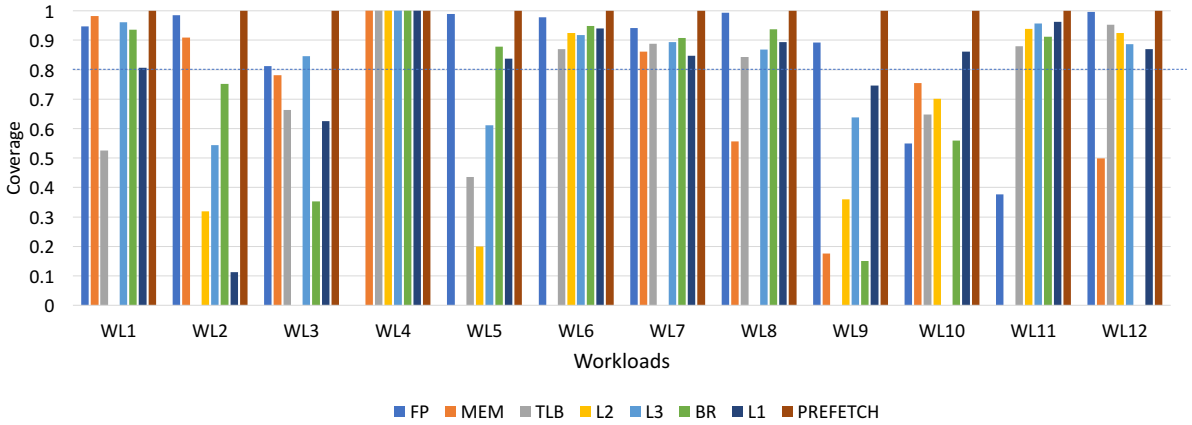
Fig. 8: CMT-bone captures all the resource utilization behavior of CMT-nek for the two kernels.

B. Point computations:

To validate the point computation kernel, VERITAS compares the resource utilization behavior of the kernel for CMT-nek and CMT-bone. This kernel essentially performs a number of vector-vector multiplications to compute the primitive variables at every grid point.



(a) Resource utilization of CMT-nek and coverage of CMT-bone.



(b) The MEM resource coverage is poor for most of the workloads

Fig. 9: For **`comm_kernel`**, CMT-bone does not cover the memory utilization behavior of CMT-nek with scale for multiple workloads.

Figure 8a shows that the memory utilization behavior of CMT-nek is the most predictive of the efficiency loss of **`point_compute_kernel`** with scale in CMT-nek and the corresponding kernel in CMT-bone captures that behavior very well (with $Coverage > 0.8$). Figure 8a also shows that the performance behavior of the **`point_compute_kernel`** in CMT-bone matches all resource utilization behavior of the kernel in CMT-nek on Xeon, hence it is an ideal proxy.

C. Computation step:

The key computation step in CMT-nek essentially performs matrix-matrix multiplication for computing fluxes and spatial partial derivatives of fluxes.

Figure 8b reveals that the major cause for efficiency loss with scale in the **`compute_kernel`** of CMT-nek is L3 and node memory utilization and CMT-bone covers these resource behavior perfectly. Further investigation reveals that the gradient and the volume integral computations in this module perform a number of matrix-matrix multiplications where matrices are accessed in strides of 1, N and N^2. This results in a large number of L3 store misses which in turn generates a large number of node memory store operations.

D. Communication step:

Even though the performance of this module is dominated by inter-node communication, our analysis identifies the key computation behavior that cause efficiency loss in this kernel with scale, since we collect on-node hardware performance counters.

Figure 9a shows that even though memory utilization is the most important behavior that impacts scalability (Resource Importance ≥ 0.9) of CMT-nek, the **`comm_kernel`** in CMT-bone does not capture this behavior ($Coverage < 0.5$). Figure 9b shows the performance characteristics for each workload with scale and it can be seen that the performance behavior of CMT-nek and CMT-bone differs significantly for the different workloads specially for resources such as MEM.

The runtimes for CMT-bone and CMT-nek for the four different regions corresponding to workload 12 are plotted in Figures 10 through 12. These figures show that runtimes for both the applications are comparable.

VI. Related Work

In this section we present some existing benchmarks and proxy apps that are part of the CORAL benchmarks [7]. The

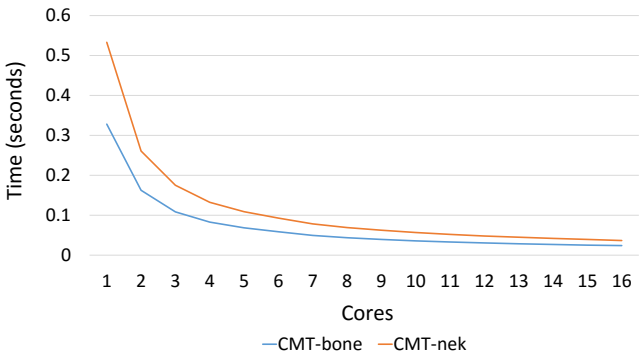


Fig. 10: Comparison of run times of CMT-nek and CMT-bone for the **`point compute kernel`**.

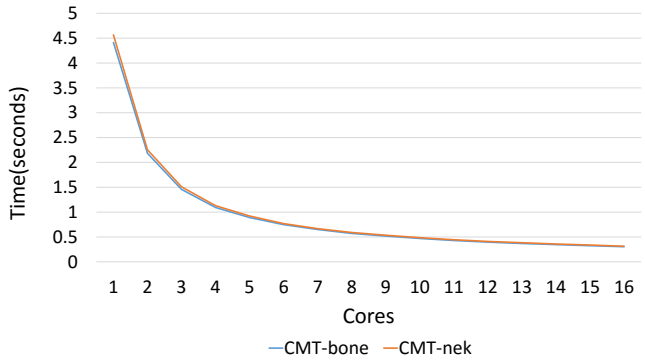


Fig. 11: Comparison of run times of CMT-nek and CMT-bone for the **`compute_kernel`**.

CORAL benchmarks were developed in a collaboration among Oak Ridge National Laboratory, Argonne National Laboratory and Lawrence Livermore National Laboratory.

Nekbone [8], is based on Nek5000, and captures the principal computation and communication kernel. Like Nek5000, Nekbone solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient solver along with a simple preconditioner. The entire computational domain is partitioned into high-order quadrilateral elements that are mapped to the processors using spectral bisection.

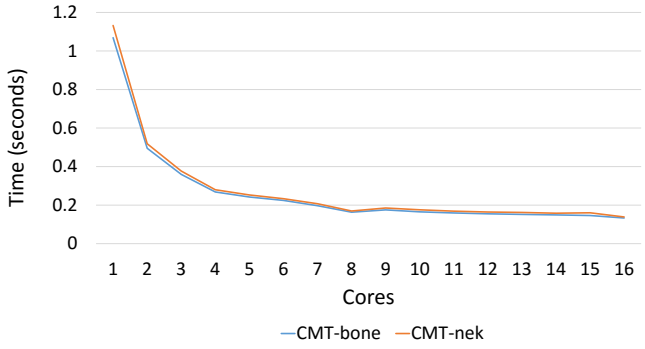


Fig. 12: Comparison of run times of CMT-nek and CMT-bone for the **`comm_kernel`**.

LULESH is a proxy application for shock hydrodynamics and represents the behavior of hydrodynamics code such as ALE3D at Lawrence Livermore National Laboratory (LLNL). LULESH has been used to evaluate the strengths and weaknesses of different existing (MPI, OpenMP, MPI+OpenMP, CUDA) as well as newer (Chapel, Charm++, Liszt, Loci) parallel programming models [9]. LULESH has also been used in [10] for analyzing factors impacting performance and energy consumption of OpenMP applications and minimizing energy usage by throttling concurrency based on runtime feedback of performance and energy consumption.

AMG2013 [11] is a parallel algebraic multi-grid solver that solves a linear system of equations arising from problems on unstructured grids. AMG2013 is a proxy application derived from the BoomerAMG solver in the hypre library being developed at LLNL. AMG2013 is a synchronous code exhibiting surface-to-volume relationship in the communications and computations patterns. AMG2013 also has low computation to data access ratio and is a memory intensive application.

Qbox [12] is a first-principle molecular dynamics code that is used to compute properties of materials. A Qbox run stresses memory bandwidth, is characterized by high floating-point intensity and uses global collectives. The two main computational operations dense linear algebra implemented in a parallel fashion using ScaLAPACK library, as well as a custom 3D Fast Fourier Transform. The primary data structure is the electronic wavefunction that is distributed across MPI processes. The main computation operation are matrix multiplication, Gram-Schmidt orthogonalization and parallel FFTs.

UMT2013 [13] is a single physics package code for deterministic radiation transport in unstructured mesh. It has high computation intensity, large message sizes and stresses memory bandwidth. Utilizing the distributed memory of a platform, the benchmark provides impressive weak scaling to very large core counts. Validation of a number of parent application and corresponding proxy applications is analyzed in [5], such as OpenMC and XSBench. OpenMC is the parent application that computes the path of particle neutron through a nuclear reactor using Monte Carlo simulations. XSBench is a proxy of OpenMC that represents the most compute intensive kernels of OpenMC accounting for about 85% of the total runtime of OpenMC.

Effective implementation of CMT-bone on parallel machines will require designing dynamic load balancing algorithms due to variable density of particles and particle movement. The methods in [14] show how particles and spectral elements can be mapped in a simple architecture independent representation, such as a one dimensional array, while still preserving node locality. The methods in [15] present optimizations for fast mapping on hybrid architectures.

VII. Conclusions

In this paper, we have presented the key implementation components of CMT-nek that was used to develop CMT-bone. CMT-Bone has several interesting features that can be used to

evaluate different memory access patterns for volume data, surface data and particles. Hence, using CMT-bone we can evaluate a variety of architectures, communication network bottlenecks, and scalability issues. Additionally, the particle simulation capabilities present in CMT-bone will also allow us to study and model a variety of dynamic load balancing algorithms for particle tracking. CMT-bone was validated to be a representative application of CMT-nek for the computation intensive region of the code, using VERITAS.

ACKNOWLEDGEMENT

This work was funded by the U.S. Department of Energy, National Nuclear Security Administration, Advanced Simulation and Computing Program, as a Cooperative Agreement under the Predictive Science Academic Alliance Program, Contract No. DOE-NA0002378.

REFERENCES

[1] P. F. Fischer, J. Lottes, S. Kerkemeier, K. Heisey, A. Obabko, O. Marin, and E. Merzari. http://nek5000.mcs.anl.gov, 2014.

[2] P.F. Fischer H.M. Tufo. Terascale spectral element algorithms and implementations. In *In Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 68. ACM, 1999.

[3] S. Gottlieb and C.-W. Shu. Total variation-diminishing Runge-Kutta schemes. *Math. Comp.*, 67:73–85, 1998.

[4] M. S. Liou. A sequel to AUSM: AUSM+. *J. Comp. Phys.*, 129:362–382, 1996.

[5] Tanzima Zerin Islam, Jayaraman J. Thiagarajan, Abhinav Bhatele, Martin Schulz, and Todd Gamblin. A machine-learning framework for performance coverage analysis of proxy applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[6] A.M. Bruckstein, Michael Elad, and Michael Zibulevsky. On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations. *IEEE Transactions on Information Theory*, 54(11):4813–4820, 2008.

[7] Coral benchmark codes. https://asc.llnl.gov/CORAL-benchmarks, 2014.

[8] Proxy-apps for thermal hydraulics. https://cesar.mcs.anl.gov/content/software/thermal_hydraulics.

[9] I. Karlin, A. Bhatele, J. Keasler, B.L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C.H. Still. Exploring traditional and emerging parallel programming models using a proxy application. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 919–932, May 2013.

[10] A.K. Porterfield, S.L. Olivier, S. Bhalachandra, and J.F. Prins. Power measurement and concurrency throttling for energy reduction in openmp programs. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 884–891, 2013.

[11] Co-design at lawrence livermore national lab. https://codesign.llnl.gov/amg2013.php.

[12] Qbox qb@ll branch. https://asc.llnl.gov/CORAL-benchmarks/Summaries/QBox_Summary_v1.2.pdf.

[13] Umt. https://asc.llnl.gov/CORAL-benchmarks/Summaries/UMT2013_Summary_v1.2.pdf.

[14] Chao-Wei Ou, Manoj Gunwani, and Sanjay Ranka. Architecture-independent locality-improving transformations of computational graphs embedded in k-dimensions. In *Proceedings of the 9th International Conference on Supercomputing*, ICS '95, pages 289–298, New York, NY, USA, 1995. ACM.

[15] Maher Kaddoura, Chao-Wei Ou, and Sanjay Ranka. Runtime support for parallelization of data-parallel applications on adaptive and nonuniform computational environments. *Juornal of Parallel and Distributed Computing*, 43:163–168, 1997.

[16] M. O. Deville, P. F. Fischer, and E.H. Mund. *High-order methods for incompressible fluid flow*. Cambridge University Press, Cambridge, 2002.

[17] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: Algorithms, analysis, and applications*. Springer, New York, 2008.

[18] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods I: Fundamentals in single domains*. Springer-Verlag, New York, 2006.

[19] B. Fornberg. *A practical guide to pseudospectral methods*. Cambridge University Press, New York, 1998.

[20] R. M. Kirby and G. E. Karniadakis. De-aliasing on non-uniform grids: algorithms and applications. *J. Comp. Phys.*, 191:249–264, 2003.

APPENDIX A

A. Governing Equations

The three-dimensional Euler equations of gas dynamics are written in the form of a conservation law

$$\frac{\partial U_m}{\partial t} + \nabla \cdot \mathbf{H}_m = R_m, \tag{3}$$

where U_m is the m^{th} of five conserved variables,

$$\mathbf{U} = \phi_g \begin{bmatrix} \rho_g \\ \rho_g u_g \\ \rho_g v_g \\ \rho_g w_g \\ \rho_g E_g \end{bmatrix}, \tag{4}$$

for a gas occupying the fraction ϕ_g of an infinitesimal volume V surrounding a given point $\mathbf{x}$,

$$\phi_g(\mathbf{x}) \equiv \lim_{V \to 0} \frac{V_g}{V}. \tag{5}$$

The corresponding volume fraction occupied by particles at a given point is $\phi_p(\mathbf{x}) \equiv 1 - \phi_g(\mathbf{x})$. The gas velocity $\mathbf{u}$ and spatial coordinate $\mathbf{x}$ are

$$\mathbf{u} = \begin{bmatrix} u_g \\ v_g \\ w_g \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{6}$$

ρ_g is the gas density, E_g is the mass-specific total energy $e_g + \frac{1}{2}|\mathbf{u}|^2$ of the gas, e is the gas internal energy, and p_g is the thermodynamic gas pressure.

$\mathbf{H}_m \in \mathbb{R}^3$ is the flux vector of equation m. For fluid mass U_1,

$$\mathbf{H}_1 = \phi_g \rho_g \mathbf{u} = U_{2-4}, \tag{7}$$

and, for gas momentum U_{2-4},

$$\mathbf{H}_2 = \phi_g \begin{bmatrix} (\rho_g u_g)\, u_g + p_g \\ (\rho_g u_g)\, v_g \\ (\rho_g u_g)\, w_g \end{bmatrix}, \mathbf{H}_3 = \phi_g \begin{bmatrix} (\rho_g v_g)\, u_g \\ (\rho_g v_g)\, v_g + p_g \\ (\rho_g v_g)\, w_g \end{bmatrix},$$
$$\mathbf{H}_4 = \phi_g \begin{bmatrix} (\rho_g w_g)\, u_g \\ (\rho_g w_g)\, v_g \\ (\rho_g w_g)\, w_g + p_g \end{bmatrix}, \tag{8}$$

and, for gas total energy $\rho_g E_g$,

$$\mathbf{H}_5 = \phi_g \mathbf{u} \left(\rho_g E_g + p_g\right). \tag{9}$$

The system is closed by an equation of state,

$$[p_g, T_g] = \text{EOS}\left(\rho_g, e_g\right). \tag{10}$$

Treatment of the particles and the source term due to the particles is deferred for the present time. For the present discussion, $R_m = 0, m = \{1, \ldots, 5\}$ Internal energy per unit mass $e = E - \frac{1}{2}|\mathbf{u}|^2$ is related to gas temperature T by the intensive property c_v, the constant-volume specific heat, such that

$$e_g = \int c_{vg}(T) dT. \tag{11}$$

Generally, Equation 11 must be solved for temperature T implicitly, iteratively, or via tabulation. For calorically perfect gases, c_{vg} is constant. For both thermally and calorically perfect gases, pressure is obtained last via

$$p_g = \rho_g R_g T_g, \tag{12}$$

where the specific gas constant $R = (\gamma - 1)\, c_{vg}$ requires the specification of $\gamma = c_{pg}/c_{vg}$, the ratio of constant-pressure specific heat c_{pg} to c_{vg}.

The current version of CMT-nek has one-way coupled point particle tracking capability which means that the point particles are advected based on the forces exerted by the gas, however, the particle motion itself does not modify or affect the surrounding flow. The equation of motion are as shown below

$$\frac{d\mathbf{x}^{(p)}}{dt} = \mathbf{v}^{(p)}, \tag{13}$$

$$\frac{d\mathbf{v}^{(p)}}{dt} = \frac{1}{m^{(p)}} \left(\mathbf{F}^{(p)}_{QS} + \mathbf{F}^{(p)}_{SG} + \mathbf{F}^{(p)}_{IU} + \mathbf{F}^{(p)}_{VU} \right). \tag{14}$$

In the above, superscript (p) refers to the p^{th} of M_p particles. $\mathbf{x}^{(p)}$ and $\mathbf{v}^{(p)}$ is the location and velocity of the point particle p. The Eulerian quantities are represented by superscript (g) and refer to interpolated values at particle position $\tilde{x}^{(p)}$. $\mathbf{F}^{(p)}_{QS}$ is the quasi-steady drag term, $\mathbf{F}^{(p)}_{SG}$ is the stress gradient force, $\mathbf{F}^{(p)}_{IV}$ is the inviscid-unsteady force and $\mathbf{F}^{(p)}_{VU}$ is the viscous-unsteady force. The expression for these forces are given below

$$\mathbf{F}^{(p)}_{QS} = 3\pi\mu^{(g)} d^{(p)} \left(\mathbf{u}^{(g)} - \mathbf{v}^{(p)} \right) f_1(Re^{(p)}, M^{(p)}) f_2(\phi^{(p)}), \tag{15}$$

$$\mathbf{F}^{(p)}_{SG} = V^{(p)} \rho^{(g)} \frac{D\mathbf{u}^{(g)}}{Dt}, \tag{16}$$

$$\mathbf{F}^{(p)}_{IU} \approx \frac{1}{2} V^{(p)} f_3(M^{(p)}) f_4(\phi^{(p)}) \left[\rho^{(g)} \frac{D\mathbf{u}^{(g)}}{Dt} - \frac{d\rho^{(g)}\mathbf{v}^{(p)}}{dt} \right], \tag{17}$$

$$\mathbf{F}^{(p)}_{VU} = 3\pi\mu^{(g)} d^{(p)} \int_{-\infty}^{t} K_{VU} \left((t-\xi), \mathbf{Re}^{(p)} \right) \left[\rho^{(g)} \frac{D\mathbf{u}^{(g)}}{Dt} - \frac{d\rho^{(g)}\mathbf{v}^{(p)}}{dt} \right] d\xi, \tag{18}$$

where $\mu^{(g)}$ is the viscosity of gas, $\rho^{(g)}$ is the density of gas, $d^{(p)}$ is the diameter of the point particle, $V^{(p)}$ is the volume associated with the point particle, $Re^{(p)}$ is the Reynolds number based on the magnitude of relative velocity of the particle with respective to the surrounding flow, $Ma^{(p)}$ is the Mach number based on the relative velocity of the particle. $f_1(Re^{(p)}, M^{(p)})$ is the finite Mach number and finite Reynolds number correction and $f_2(\phi^{(p)})$ is the volume fraction correction to the quasi-steady drag term. $f_3(M^{(p)})$ and $f_4(\phi^{(p)})$ are the finite mach number and volume fraction correction to the inviscid-unsteady (kernel) force term, respectively. is the volume fraction correction to the inviscid-unsteady (kernel) force term. K_{VU} is the viscous-unsteady kernel for computing the viscous unsteady force. Full expressions for these correction terms can be found in [].

B. Discontinuous Galerkin scheme

a) The weighted residual statement: Discontinuous Galerkin methods solve Equation 3 by partitioning the domain Ω into nelt nonoverlapping elements, the e^{th} of which is Ω_e. In the following section, we consider a single conserved variable U_m in Equation 4 and omit the subscript m. We approximate U by a truncated series of basis functions whose domain is Ω_e and enforcing a weighted-residual statement such that

$$\int_{\Omega_e} v(\mathbf{x}) \frac{\partial U(\mathbf{x})}{\partial t} dV + \int_{\Omega_e} (\nabla \cdot \mathbf{H})\, v(\mathbf{x}) dV = 0 \tag{19}$$

for *all* test functions $v : \mathbb{R}^3 \to \mathbb{R}$ in the same function space where U is approximated[16], [17]. We then integrate the flux divergence $v\nabla \cdot \mathbf{H}$ by parts:

$$\int_{\Omega_e} v(\mathbf{x}) \frac{\partial U(\mathbf{x})}{\partial t} dV = \int_{\Omega_e} (\nabla v) \cdot \mathbf{H} dV - \int_{\partial\Omega_e} v(\mathbf{x}) \mathbf{H} \cdot \hat{\mathbf{n}} dA \tag{20}$$

where $\hat{\mathbf{n}}(\mathbf{x})$ is the unit normal vector to $\partial\Omega_e$ facing outward from Ω_e.

The central and distinguishing feature of discontinuous Galerkin methods is the replacement of $\mathbf{H} \cdot \hat{\mathbf{n}}$ in the surface integral in Equation 20 with a numerical flux $\mathbf{H}^* \cdot \hat{\mathbf{n}}$ that

1) weakly enforces flux continuity (or boundary conditions should $\partial\Omega_e \subset \partial\Omega$) at the element interfaces without demanding continuity of U there, and
2) ensures physical correctness of the solution independent of functional representations within the element.

It is important to introduce some notation now; at a given point $\mathbf{x}$ on $\partial\Omega_e$,

$$U^-(\mathbf{x}) \equiv U(\mathbf{x}) \text{ taken from the } \textbf{interior} \text{ of } \Omega_e \tag{21}$$

$$U^+(\mathbf{x}) \equiv U(\mathbf{x}) \text{ taken from } \textbf{outside} \text{ of } \Omega_e \tag{22}$$

where $\mathbf{U}^+$ is understood to be equal to $\mathbf{U}^-$ for the neighboring element sharing $\partial\Omega_e$ with the e^{th} element. The numerical flux must be consistent:

$$\mathbf{H}^*(\mathbf{U}^-, \mathbf{U}^+) = \mathbf{H}(\mathbf{U}) \text{ if } \mathbf{U}^+ = \mathbf{U}^-, \tag{23}$$

and will be discussed further in later sections.

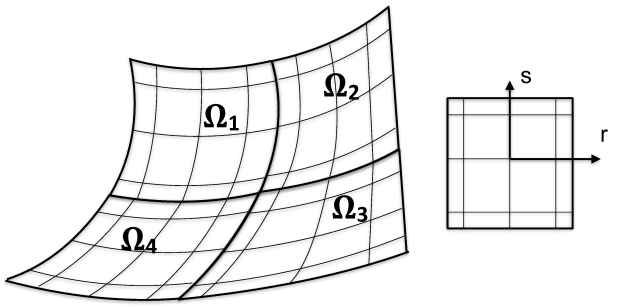


Fig. 13: A schematic representation of a collection of spectral elements making up a computational domain. A schematic representation of a reference element is also shown.

b) Spectral element discretization: Inside the e^{th} element Ω_e the unknowns $\mathbf{U}$ are represented in some finite-dimensional space of functions χ. The representation of a single (scalar component of Equation 4) conserved variable U is written as a truncated series expansion

$$U(x) \approx \sum_{i=1}^{N} \widehat{U}_i \phi_i(x), x \in \Omega_e, \tag{24}$$

that is, a linear combination of basis functions spanning χ $\{\phi_i : \Omega_e \to \mathbb{R}, i = 1, \dots N\} \subset \chi$, each of which is weighted by some coefficient $\widehat{U}_i$. Note that Equation (24) refers to a one-dimensional problem. In this simplification the element is just a line segment. Furthermore,

1) Coordinate x in each element is **isoparametrically mapped** onto the reference element $r \in [-1, 1]$. Figure 13 shows a schematic of 2 dimensional space divided into elements. Also a schematic of a reference element is shown in the figure.
   - This transformation has, $\forall \mathbf{x} \in \Omega_e$:
     a) Jacobian $\text{Jac}(\mathbf{x}) \equiv |\partial \mathbf{x}/\partial \mathbf{r}|$
     b) Metrics $\partial \mathbf{r}/\partial \mathbf{x}$
   - χ is the **space** of $(N-1)^{\text{th}}$-order polynomials $\mathbb{P}^{N-1}$.
   - The **coefficients** $\widehat{U}_i = U(x_i), i = 1 \dots N$ are the values of the unknown at grid points x_i within Ω_e. Thus, Equation 24 is said to be a **nodal** approximation to U.
   - The i^{th} **basis function** $\phi_i = l_i(x)$ is the Lagrange interpolating polynomial associated with the i^{th} node x_i.
     – This representation extends to higher dimensions by tensor-nesting the Lagrange polynomials in each of the coordinate directions. For example, in three dimensional space we have directions r, s, and t on the cubic $[-1, 1]^3$ reference element.
2) The grid points on the reference element are the N **Gauss-Legendre-Lobatto** (GLL) quadrature nodes[18] .

The next major numerical approximation underlying the method is to approximate all integrals by Gaussian quadrature on the N GLL points in the reference element. Quadrature approximates an integral by a weighted sum of nodal values,

$$\int_{-1}^{1} f(r) dr \approx \sum_{i=1}^{N} \omega_i f(r_i), \tag{25}$$

where the quadrature weight for the i^{th} point ω_i depends on the locations of the quadrature nodes r_i. Formulas for determining quadrature weights for the GLL nodes are well-known [18], [16].

Nodal values at grid points are arranged into vectors along a given reference-element coordinate (r is shown):

$$\mathbf{U} \equiv \begin{bmatrix} U(r_1) \\ \vdots \\ U(r_N) \end{bmatrix}, \mathbf{v} \equiv \begin{bmatrix} v(r_1) \\ \vdots \\ v(r_N) \end{bmatrix}, \mathbf{h}_1 \equiv \begin{bmatrix} H_x(U(r_1)) \\ \vdots \\ H_x(U(r_N)) \end{bmatrix}. \tag{26}$$

Then defining

$$\text{diag}(\mathbf{v}) \equiv \begin{bmatrix} \diagdown & & 0 \\ & v(r_i) & \\ 0 & & \diagdown \end{bmatrix}, \tag{27}$$

will allow us to write vectors of N nodal values as $N \times N$ diagonal matrices:

$$J \equiv \text{diag}(\text{Jac}) \tag{28}$$

$$\mathcal{M} \equiv \text{diag}(\omega_j). \tag{29}$$

Finally, the derivative of a function approximated by Equation 24 at its N nodes r_i may itself be approximated by finite differences. Many authors[16], [17], [19] provide the details of how to represent $(N-1)^{\text{th}}$-order-accurate finite differences on each of N points as matrix-vector products between a $N \times N$ differentiation matrix $\mathcal{D}$ and a vector of nodal values:

$$\mathcal{D}\mathbf{v} \approx \left[\left.\frac{dv}{dr}\right|_{r_1}, \cdots, \left.\frac{dv}{dr}\right|_{r_N} \right]^{\top} \tag{30}$$

For the rest of this section, subscripts refer to Cartesian components of vectors in $\mathbb{R}^3$, so r_j is the j^{th} component of $(r_1, r_2, r_3)^{\top} = (r, s, t)^{\top}$ and x_i is the i^{th} component of $\mathbf{x}$ (Equation 6). $\mathcal{D}_j$ refers to an appropriately tensor-nested differentiation operator in the direction of r_j ([16], Chapter 4). Einstein summation convention is applied to spatial coordinates, and bold-faced quantities are vectors of grid points (Equation 26), not spatial vectors. $\mathcal{M}$ is also tensor-nested: $M_{mno} = \omega_m \omega_n \omega_o$ for the grid point with indices m, n, o.

Additionally defining $\mathcal{E}$ as an indicator that is zero for all grid points except those on the element faces $\partial\Omega_e$, the discontinuous Galerkin weighted-residual statement Equation 20 may be written as a semidiscrete governing equation in matrix form as

$$\mathbf{v}^{\top} \left[J\mathcal{M} \frac{\partial \mathbf{u}}{\partial t} \right] \approx \left[\left[\text{diag}\left(\frac{dr_i}{dx_j} \right) \right] \mathcal{D}_i \mathbf{v} \right]^{\top} [\mathcal{M}\mathbf{h}_j] - \mathbf{v}^{\top} \left[\mathcal{E}\mathcal{A}\left(J_A \mathbf{h}_j^* \hat{n}_j \right) \right] \tag{31}$$

where J_A is the Jacobian of the transformation mapping a given face (selected by $\mathcal{E}$) to the reference $[-1,1]^2$ square. By equating coefficients of the nodal values of the test function $\mathbf{v}$, the weighted residual theorem is satisfied for all possible $v \in \chi$ and the semidiscrete system reads (for the vector of N GLL points in a given element along a given line in rst space)

$$\mathcal{M}\frac{\partial \mathbf{u}}{\partial t} = \mathcal{D}_i^\top \left[\text{diag}\left(\frac{1}{J}\frac{dr_i}{dx_j}\right)\mathcal{M}\mathbf{h}_j \right] - \mathcal{E}\left[\mathcal{A}\left(\frac{J_A}{J}\mathbf{h}_j^* \hat{n}_j\right)\right], \tag{32}$$

where parentheses represent scalar collocation of different values with one another at the same grid point.

c) Managing aliasing error: It is well-known that Gaussian quadrature on N GLL points exactly integrates a polynomial of order $2(N-1)-1$. However, the test function, unknowns, and field variables like pressure and velocity are all polynomials of order $N-1$; their products produce integrands that may not be exactly integrated on only N points. The functional forms of the Jacobian and the metrics also affect how many points are needed for exact quadrature, especially on deformed elements with curved faces. Errors from inexact quadrature are said to "alias" onto the degrees of freedom, and the provision of more grid points for quadrature is called "overintegration[20]" or **dealiasing**.

The standard sources [16], [17], [19] on high-order methods based on polynomials also represent polynomial interpolation from N points represented by Equation 24 to M other points as a product between an $M \times N$ **interpolation matrix** $\mathcal{I}$ and N-vectors like those in Equation 26. Avoiding the details of such an operation, let

$$\mathcal{I}\mathbf{v} = \begin{bmatrix} v(r_1) \\ \vdots \\ v(r_M) \end{bmatrix} \tag{33}$$

be the matrix-vector product that interpolates $\mathbf{v}$ from N GLL points to $M > N$ Gauss-Legendre (GL) quadrature nodes, a slightly different choice of nodes with quadrature weights that can integrate polynomials of order $2M-1$ exactly. Considering a longitudinal flux of momentum (e.g., z-momentum in the z-direction), Equation 8 says

$$H_z = (\rho w)\, w_g + p_g = U_4 w_g + p_g. \tag{34}$$

An inner product of the interpolating polynomial Equation 24 representing H_z in Equation 34 with that of the test function gradient $\partial v/\partial r_j$ would be the integral of the product of three polynomials: U_4 & w (both[3] $\in \mathbb{P}^{N-1}$), and $\partial v/\partial r_j \in \mathbb{P}^{N-2}$. This integrand has degree $3N-4$, and to avoid multiplicative aliasing errors, Gaussian quadrature (Equation 25) needs each polynomial in Equation 34 on M GL points such that $2M-1 > 3N-4$. Thus, $M = 3(N-1)/2$ (or greater, depending on the polynomial order of the grid metrics and Jacobian), and the vector of grid points $\mathbf{h}_3$ must be formed by interpolating $\mathbf{U}_4$, $\mathbf{w}$ and $\mathbf{p}$ from the N-point GLL grid to the M-point GL grid, forming

$$\mathbf{h}_{3,M} = [\text{diag}\,(\mathcal{I}\mathbf{U}_4)]\,[\mathcal{I}\mathbf{w}_g] + \mathcal{I}\mathbf{p}_g, \tag{35}$$

interpolating the metrics onto the M-point grid, and repeating the manipulations that led from Equation 31 to Equation 32. The first right-hand-side term in Equation 32

$$\mathcal{D}_i^\top \left[\text{diag}\left(\frac{1}{J}\frac{dr_i}{dx_j}\right)\mathcal{M}\mathbf{h}_j \right], \tag{36}$$

is replaced by

$$\mathcal{I}^\top \mathcal{D}_{i,M}^\top \left[\text{diag}\left(\mathcal{I}\left[\frac{1}{J}\frac{dr_i}{dx_j}\right]\right)\mathcal{M}_M \mathbf{h}_{j,M} \right], \tag{37}$$

where the subscript M denotes the both the mass matrix and differentiation operators that have been independently computed for M Gauss-Legendre points.

The surface integral in Equation 32 must be re-evaluated in the same way. The interpolation operator $\mathcal{I}$ is tensor-nested; it is multiplied by a vector of point values along each "line" of GLL points for each line in each of three directions successively. The operator $\mathcal{I}_A$ is the 1D GLL interpolation matrix nested twice instead of three times, and it interpolates grid points on an element face from a 2D plane of GLL points to a 2D plane of Gauss-Legendre points. Likewise, we use a diagonal matrix $\mathcal{A}$, whose elements are $M_{ij} = \omega_i \omega_j$ at a given crossing of GLL lines, to handle quadrature for surface integrals.

$$\mathcal{E}\mathcal{A}\left(\frac{J_A}{J}\mathbf{h}_j^* \hat{n}_j\right) \tag{38}$$

becomes

$$\mathcal{E}\mathcal{I}_A^\top \left[\mathcal{A}_M \text{diag}\left(\mathcal{I}_A \frac{J_A}{J}\right) \text{diag}\left(\mathbf{h}_{j,M}^*\right) [\mathcal{I}_A \hat{\mathbf{n}}_j] \right], \tag{39}$$

where both the surface normal vector n_j and the Jacobian ratio J_A/J have been interpolated onto the M^2 GL face points.

The only remaining steps are to substitute Equations 37 and 39 into Equation 32 and multiply through by $\mathcal{M}^{-1}$ to get our final semi-discrete system.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} =& \mathcal{M}^{-1}\mathcal{I}^\top \mathcal{D}_{i,M}^\top \left[\text{diag}\left(\mathcal{I}\left[\frac{1}{J}\frac{dr_i}{dx_j}\right]\right)\mathcal{M}_M \mathbf{h}_{j,M} \right] \\ & - \mathcal{M}^{-1}\mathcal{E}\mathcal{I}_A^\top \left[\mathcal{A}_M \text{diag}\left(\mathcal{I}_A \frac{J_A}{J}\right) \text{diag}\left(\mathbf{h}_{j,M}^*\right) [\mathcal{I}_A \hat{\mathbf{n}}_j] \right], \end{aligned} \tag{40}$$

We note here that, ordinarily, inverting the mass matrix "lifts" surface integrals into volume nodes. Under the assumptions of spectral element methods, however, the mass matrix is diagonal by construction, and no lifting occurs; surface integrals only directly affect surface nodes.

[3] $w = U_4/U_1$ is technically a rational function, but it responds favorably to overintegration. We treat the thermodynamic pressure p as a polynomial in spite of the nonlinearity of Equation 10.