Modeling Continuum PDEs using the Discontinuous Galerkin Method with OpenACC

Shiva Gopalakrishnan and Mandar Gurav

Scalable Algorithms and Numerical Methods in Computing (SATANIC) Lab
Department of Mechanical Engineering
Indian Institute of Technology Bombay
Motivation: Complex Large Scale Simulations

- Tsunami Modeling
- Coastal Inundation from Storm surges and Tsunamis
- Non-equilibrium flows in Injectors
- Jets in crossflows
March towards Exascale

- Growth in supercomputing performance

credit: top500.org

- Are current numerical methods scalable?
- Are current numerical methods power efficient?
Parallel Scaling of Finite Volume Methods

Lid driven cavity: Incompressible flow solver using OpenFOAM (Opensource FVM).

credit: hpc.ntnu.no
Element Based Galerkin methods

- All EBG methods partition the domain into computational elements and then approximate a function via basis functions.
Solution Vector Approximation

- For the canonical equation

\[
\frac{\partial q}{\partial t} + \frac{\partial f}{\partial x} = 0
\]

where \( q = q(x, t) \), \( f = f(x, t) \) and \( f = qu \).

- We approximate the solution variable as

\[
q_N(x, t) = \sum_{i=0}^{N} \psi_i(x)q_i(t)
\]

where \( f_N = f(q_N(x, t)) \).

- \( q \) being the expansion coefficients, \( \psi \) the basis functions and the \( N \) the order of the polynomial.
Differential to Integral form

- Substituting the approximation into the PDE yields

\[ \frac{\partial q_N}{\partial t} + \frac{\partial f_N}{\partial x} = r \neq 0 \]

Since we have used a finite dimensional approximation.

- We resolve this by multiplying the approximation with a test function \( \psi \) and integrating to get

\[ \int_{\Omega_e} \psi_i \frac{\partial q_N}{\partial t} d\Omega_e + \int_{\Omega_e} \psi_i \frac{\partial f_N}{\partial x} d\Omega_e = \int_{\Omega_e} \psi_i r d\Omega_e \equiv 0 \]
Differential to Integral form

- where the domain is partitioned as

\[ \Omega = \bigcup_{e=1}^{N_e} \Omega_e \]

defines the total domain and \( e = 1, 2, \ldots, N_e \) are the elements
Weak Integral form

- Using calculus identities we can simplify the weak integral system into the form

\[
\int_{\Omega_e} \psi_i \frac{\partial q_N}{\partial t} \, d\Omega_e + \int_{\Omega_e} (\psi_i f_N) \, d\Omega_e - \int_{\Omega_e} \frac{\partial \psi_i}{\partial x} f_N \, d\Omega_e = 0
\]

Integrating the second term gives:

\[
\int_{\Omega_e} \psi_i \frac{\partial q_N}{\partial t} \, d\Omega_e + [\psi_i f_N]_{\Gamma_e} - \int_{\Omega_e} \frac{\partial \psi_i}{\partial x} f_N \, d\Omega_e = 0
\]

where the term in the square brackets is evaluated at the boundary \( \Gamma_e \) of the element \( \Omega_e \).
Discontinuous Galerkin Method

- The equation

\[ \int_{\Omega_e} \psi_i \frac{\partial q_N}{\partial t} d\Omega_e + [\psi_i f_N]_{\Gamma_e} - \int_{\Omega_e} \frac{\partial \psi_i}{\partial x} f_N d\Omega_e = 0 \]

represents the (weak) integral form of the original differential equation.

- The term \([\psi_i f_n]_{\Gamma_e}\) allows neighbouring elements to communicate.
Basis functions

N=1

N=2

N=3
Discontinuous Galerkin Method

Applying DG to the Constitutive equations to obtain the weak form

\[
\int_{\Omega_e} \left( \frac{\partial q_N^{(e)}}{\partial t} - F_N^{(e)} \cdot \nabla - S_N^{(e)} \right) \psi_i (x) \, dx
\]

\[
= - \sum_{l=1}^{3} \int_{\Gamma_e} \psi_i (x) n^{(e,l)} \cdot F_N^{(*,l)} \, dx
\]

Rusanov Numerical Flux

\[
F_N^{(*,l)} = \frac{1}{2} \left[ F_N \left( q_N^{(e)} \right) + F_N \left( q_N^{(l)} \right) - |\lambda^{(l)}| \left( q_N^{(l)} - q_N^{(e)} \right) n^{(e,l)} \right]
\]
Matrix form of semi–discrete equations

Using the polynomial approximation \( q_N = \sum_{i=1}^{M_N} \psi_i q_i \)

\[
\int_{\Omega_e} \psi_i \psi_j d\mathbf{x} \frac{\partial q^{(e)}}{\partial t} - F_j^{(e)} \cdot \int_{\Omega_e} \nabla \psi_i \psi_j d\mathbf{x} - \int_{\Omega_e} \psi_i \psi_j d\mathbf{x} S_j^{(e)} \\
= - \sum_{l=1}^{3} \int_{\Gamma_e} \psi_i \psi_j n^{(e,l)} d\mathbf{x} \cdot (F^{(*,l)})_j
\]

Defining element matrices as

\[
M_{ij}^{(e)} = \int_{\Omega_e} \psi_i \psi_j d\mathbf{x}, \quad M_{ij}^{(e,l)} = \int_{\Gamma_e} \psi_i \psi_j n^{(e,l)} d\mathbf{x}, \quad D_{ij}^{(e)} = \int_{\Omega_e} \nabla \psi_i \psi_j d\mathbf{x}
\]
Matrix form of semi–discrete equations

\[ M_{ij}^{(e)} \frac{\partial q_{i}^{(e)}}{\partial t} - (D_{ij}^{(e)})^T F_{j}^{(e)} - M_{ij}^{(e)} S_{j}^{(e)} = - \sum_{l=1}^{3} (M_{ij}^{(e,l)})^T (F^{(\ast,l)})_{j} \]

Eliminating mass matrix on LHS

\[ \hat{D}^{(e)} = (M^{(e)})^{-1} D^{(e)}, \quad \hat{M}^{(e,l)} = (M^{(e,l)})^{-1} M^{(e,l)} \]

\[ \frac{\partial q_{i}^{(e)}}{\partial t} - (\hat{D}_{ij}^{(e)})^T F_{j}^{(e)} - S_{j}^{(e)} = - \sum_{l=1}^{3} (\hat{M}_{ij}^{(e,l)})^T (F^{(\ast,l)})_{j} \]
Matrix form of semi–discrete equations

\[
M_{ij}^{(e)} \frac{\partial q^{(e)}}{\partial t} - (D_{ij}^{(e)})^T F_j^{(e)} - M_{ij}^{(e)} S_j^{(e)} = - \sum_{l=1}^{3} (\hat{M}_{ij}^{(e,l)})^T (F^{(e,l)})_j
\]

Volume Integration (offload)

Eliminating mass matrix on LHS

\[
\hat{D}^{(e)} = (M^{(e)})^{-1} D^{(e)}, \quad \hat{M}^{(e,l)} = (M^{(e,l)})^{-1} M^{(e,l)}
\]

\[
\frac{\partial q^{(e)}}{\partial t} - (\hat{D}_{ij}^{(e)})^T F_j^{(e)} - S_j^{(e)} = - \sum_{l=1}^{3} (\hat{M}_{ij}^{(e,l)})^T (F^{(e,l)})_j
\]

Flux Integration (offload)
Finite volume Stencil
Sparsity Pattern: Finite volume

Unordered Matrix

Ordered Matrix RCM

Credit: Mathworks.com
Sparsity Pattern: Discontinuous Galerkin

Mass Matrix

Differentiation Matrix
Speedup on GPUs

![Speedup on GPUs Graph](image)
Speedup on GPUs: Optimised for $N=4$
Scaling: Discontinuous Galerkin with N=4

(a) wallclock time

(b) speedup

Girlado et al, Continuous and discontinuous Galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode, JCP (2012)
Scaling: Discontinuous Galerkin with N=8

(a) wallclock time

(b) speedup

Girlado et al, Continuous and discontinuous Galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode, JCP (2012)
Error vs Computational Efficiency

![Graph showing error vs computational efficiency for different DG and FDM methods. The x-axis represents the total number of floating point operations (in GFlop), and the y-axis represents the L2 norm. The graph compares different DG methods with varying polynomial degrees (DG N=3, DG N=5, DG N=7, DG N=9) and a FDM 3rd Upwind method.](image-url)
Power Efficiency

L2 Norm vs Energy consumed

DG N=3
FDM 3rd Upwind
MEANDG Framework

- C++ based framework.
- Fully three dimensional. Support for Hexahedral, Tetrahedral and transitional prism, pyramid cells.
- Complete abstraction. Discrete operators can work with Scalar, Vector and Tensor objects.
- Can quickly develop solvers based on Continuum PDEs.
- Currently solver for Advection, Euler and Navier–Stokes Equations are present.
- Parallel implementation using OpenMP, OpenACC and MPI.
Geometry Support

(a) Hexahedra (b) Prism (c) Tetrahedra (d) Pyramid

- Higher order support through cardinal Lagrange polynomials.
- Polynomials upto 16th order have been tested.
- Natural support for $h$ and $p$ refinement.
Complex Geometry

- Flow past a motorbike.
Three dimensional Westervelt Equations

- Discontinuous Galerkin code based on the Westervelt equation to simulate transient acoustic wave propagation in the brain and skull.
- Collaborators: James F. Kelly, Michigan State University and Simone Marras, Rutgers University
- Ongoing, only 12 routines have been parallelized via OpenACC.
- Speedup: 4.62
- GPU used: Nvidia V100 (PSG Cluster)
GPU Bootcamp at IIT Bombay

- 13 research groups with approximately 30 researchers and 6 mentors. Held May 7th and 8th 2019
- Application domains
  - Computational Fluid Dynamics
  - Materials Science
  - Physics
  - Computational Biology
  - Earth systems.
- Groups had either serial code or MPI parallel code.
- With OpenACC the max speedup achieved by a group was 40x. most groups reported some amount of speedup.
Conclusions

- To solve complex problems we need more detailed simulation capability which in turn requires more than ever computational power.
- Current numerical methods technology has limits on issues of scaling.
- Newer methods are required. DG promises to show linear scaling up to thousands, if not hundreds of thousands of processors.
- Power efficiency is desired and DG demonstrates power savings to large extent.
- Numerical methods have to adapt to newer computational hardware rather than vice versa.