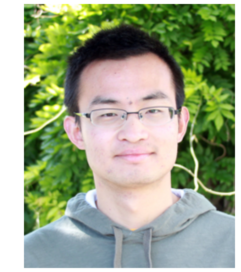
Accelerating the BLAST code with hybrid MPI+OpenMP+CUDA programming on CPU-GPU clusters

TINGXING DONG[1,3] WITH TZANIO KOLEV[1] AND ROBERT RIEBEN[2] AND VESELIN DOBREV[1]

[1]Center for Applied Scientific Computing, LLNL; [2]Weapons and Complex Integration, LLNL

[3]Innovative Computing Laboratory, University of Tennessee, Knoxville



Abstract: The BLAST[1] code implements a high-order numerical algorithm that solves the equations of compressible hydrodynamics using the Finite Element Method in a moving Lagrangian frame. We accelerate the computational intensive parts (80%–95%) of BLAST with hybrid MPI + OpenMP + CUDA programming on CPU-GPU clusters. Our test shows that 12 CPU cores and 2 GPUs delivered 21x speedup compared to a single Intel Xeon core a 2x speedup over 12 MPI tasks.

BLAST

- Supports unstructured curvilinear meshes.
- High order field representations.
- Exact discrete energy conservation by construction.
- Multiple options for basis functions and quadrature orders.
- Reduces to classical staggered-grid hydro algorithms under simplifying assumptions.

Lagrangian Hydrodynamics

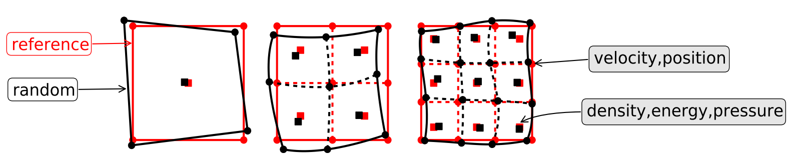
On semi-discrete level our method can be written as

$$\textbf{Momentum Conservation:}\quad \frac{d\mathbf{v}}{dt} = -\mathbf{M}_\mathbf{v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:}\quad \frac{d\mathbf{e}}{dt} = \mathbf{M}_\mathbf{e}^{-1}\mathbf{F}^\mathbf{T}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:}\quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M}_\mathbf{v}$ and $\mathbf{M}_\mathbf{e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step. The right side of the first two equations take more than 80% of the total time and therefore are the computational hot spots of the algorithm.



Types of Zones: (left to right) bilinear (Q1-Q0), biquadratic (Q2-Q1), and bicubic (Q3-Q2) zones and corresponding degrees of freedom.

Reference:[1] V.A.Dobrev, Tz.V.Kolev, R.N.Rieben. High order curvilinear finite element methods for Lagrangian hydrodynamics. SIAM J.Sci.Comp.12.

Corner Force Matrix F

The computational kernel of our method is the evaluation of the **Generalized Corner Force** matrix, which is constructed by three loops:

– Loop over all domains
  – Loop over zones in the domain
    – Loop over quadrature points in this zone

Each quadrature point computes hydro forces asscociated with it absoutely independently. $\mathbf{F}$ varies with basis functions, dimension, etc, and can be arbitrarily expensive.

$$(\mathbf{F}_z)_{ij} = \int_{\Omega_z(t)} (\sigma : \nabla\vec{w}_i)\,\phi_j \approx \sum_k \alpha_k \hat{\sigma}(\vec{\hat{q}}_k) : \mathbf{J}_z^{-1}(\vec{\hat{q}}_k)\hat{\nabla}\hat{\vec{w}}_i(\vec{\hat{q}}_k)\,\hat{\phi}_j(\vec{\hat{q}}_k)|\mathbf{J}_z(\vec{\hat{q}}_k)|$$

- The quantities α_k, $\hat{\nabla}\hat{\vec{w}}_i$, $\hat{\phi}_j(\vec{\hat{q}}_k)$ do not change in time and can be put into constant memory.
- The evaluation of the stress values $\hat{\sigma}(\vec{\hat{q}}_k)$ requires significant amount of computations (SVD, eigenvectors, EOS, etc.).

CUDA Implementation of Corner Force

1. **Loop over qudrature points** and ompute part of $\mathbf{F}$ based on $\mathbf{v}$, $\mathbf{e}$, $\mathbf{x}$ (transferred from CPU) and work space allocated on GPU.
2. **Loop over zones.** Each zone does a matrix-matrix transpose multiplication and assemble the matrix $\mathbf{F}$ which stays on the GPU.
3. **Compute** $\mathbf{F}\cdot\mathbf{1}$ and either return result to the CPU or keep on the GPU depending on our CUDA-CG solver turning off/on.
4. **Compute** $\mathbf{F}^\mathbf{T}\cdot\mathbf{v}$ with results staying on GPU.
5. A custom Conjugate Gradient (CG) solver for $\mathbf{M}_\mathbf{v}^{-1}(\mathbf{F}\cdot\mathbf{1})$ based on cuBLAS/cuSPARSE with a diagonal preconditioner.
6. Sparse (CSR) matrix vector multiplication to compute $\mathbf{M}_\mathbf{e}^{-1}(\mathbf{F}^\mathbf{T}\cdot\mathbf{v})$ by calling a cuSPARSE routine.

CUDA + OpenMP Implementation of Corner Force

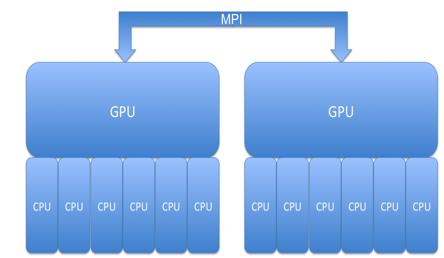
- CPU host thread launchs CUDA kernels and returns immediately.
- Host thread spawns OpenMP threads and distributes the loop over zones between threads.
- Each thread allocates working space and executes like normal serial code.
- OpenMP is used to harness 6 CPU cores.
- Synchronization between CPU and GPU to complete $\mathbf{F}$

MPI + CUDA + OpenMP

- **Two layers of parallelism**
- MPI-based parallel domain-decomposition and communication between CPUs
- CUDA and OpenMP based parallel corner forces in BLAST
- One GPU is attached to one MPI task.
- Auto tuning: a scheduler to find the optimal ratio of workload between 1 GPU and 6 CPU cores.

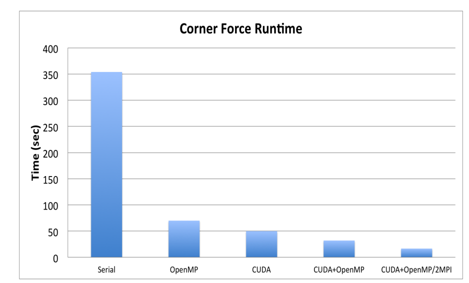
Case	Method	Ratio
Triple-pt	Q3Q2	7:3
Sedov 3D	Q2Q1	6:4
Sedov 2D	Q2Q1	5:5

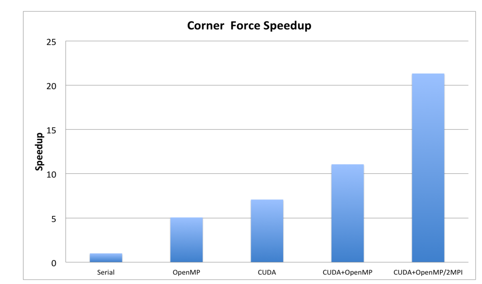
Optimal workload ratio of 1 M2050 to 6 Xeon cores



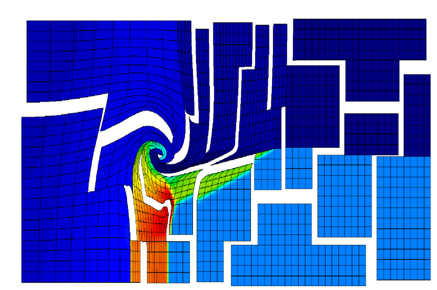
MPI + CUDA + OpenMP hierarchy

Performance

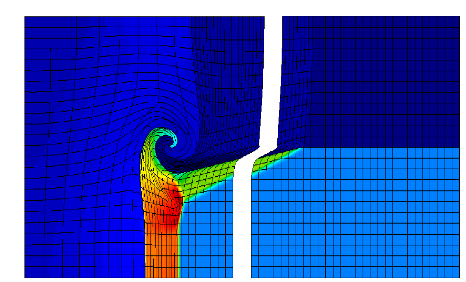




Test Results



12 MPI tasks



2MPI tasks, each with 1M2050+6Xeon Cores

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-571572