Multi-GPU High-Order Unstructured Solver for Compressible Navier-Stokes Equations

P. Castonguay, D. Williams, P. Vincent, and A. Jameson

Aeronautics and Astronautics Department, Stanford University

Introduction

Algorithm: Unstructured high-order compressible flow solver for Navier-Stokes equations

Why a higher order flow solver?

For applications with low error tolerance, high order methods are more cost effective

Why is higher order suitable for the GPU?

More operations per degree of freedom
Local operations within cells allow use of shared memory

Objectives: Develop a fast, high-order multi-GPU solver to simulate viscous compressible flows over complex geometries such as micro air vehicles, birds, and Formula 1 cars

Solution Method

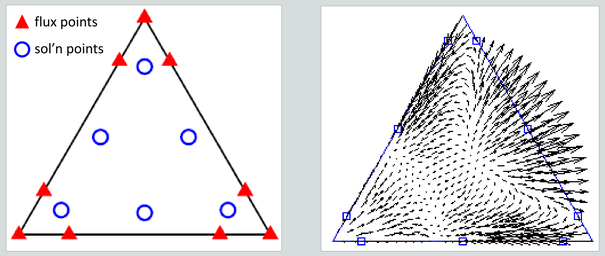


Figure 1: Reference Element (left). Correction function (right)

Flux Reconstruction Approach:

1. Use solution points to define polynomial representation of solution inside the cell
2. Use flux points to communicate information between cells

   Flux Exchange: Find a common flux at points on the boundary between cells

   Flux Reconstruction: Propagate new flux information back into each cell interior using higher order correction polynomial (see figure above)
3. Update the solution using new information

Types of Operations:

Matrix multiplication: Ex: Extrapolate state between flux and solution points

Algebraic: Ex: Find common flux in Flux Exchange procedure

Matrix Multiplication Example

Strategy: Make use of shared and texture memory. Allow multiple cells per block

Challenges: Avoid bank-conflicts and non-coalesced memory accesses
Extrapolation matrix M is too big to fit in shared memory for 3D grids

$$MQ_S = Q_E$$

$$M : Texture$$

$$Q_S : Shared$$

$$Q_E : Global$$

```
for (i=0;i<n_qpts;i++)
{
    m = i*n_ed_fgpts+ifp;            m1 = n_fields*n_qpts*ic_loc+i;

    q0 +=  fetch_double(t_interp_mat_0,m)*s_q_qpts[m1];  m1 += n_qpts;
    q1 +=  fetch_double(t_interp_mat_0,m)*s_q_qpts[m1];  m1 += n_qpts;
    q2 +=  fetch_double(t_interp_mat_0,m)*s_q_qpts[m1];  m1 += n_qpts;
    q3 +=  fetch_double(t_interp_mat_0,m)*s_q_qpts[m1];
}
```

Figure 2: Matrix multiplication example. Extrapolate state from solution points to flux points

Single-GPU Results

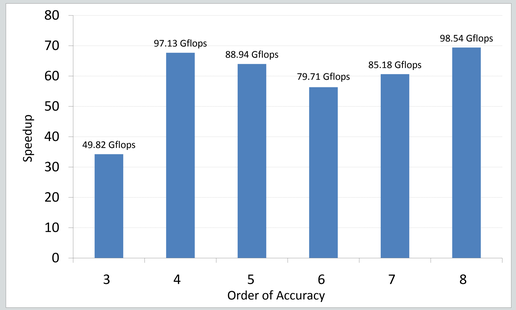


Figure 3: Overall double precision performance on Tesla C2050 relative to 1 core of Xeon X5670 vs. order of accuracy for quads

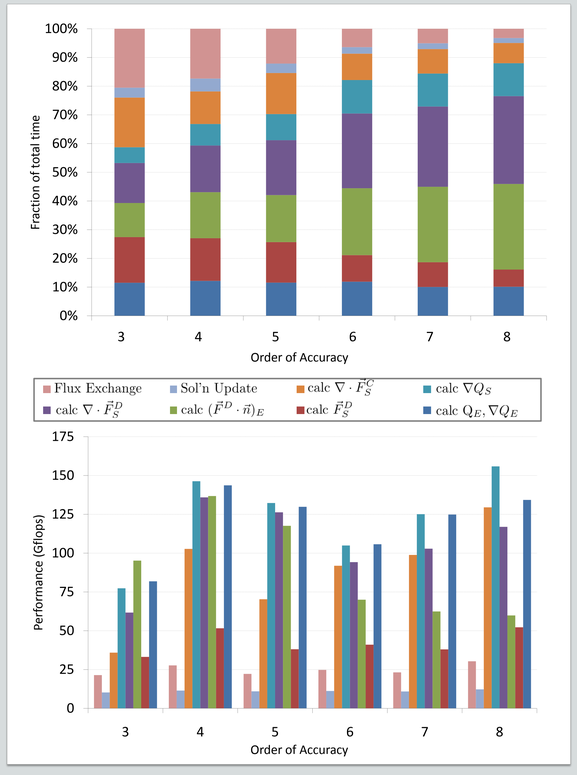


Figure 4: (Top) Profile of GPU algorithm vs. order of accuracy for quads (Bottom) Performance per kernel vs. order of accuracy for quads

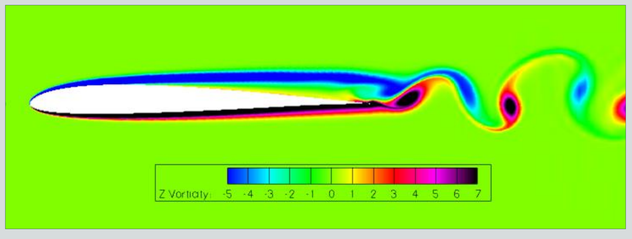


Figure 5: Spanwise vorticity contours for unsteady flow over SD7003 airfoil at Re = 10000, M=0.2

Multi-GPU Results

Approach:

Use mixed MPI-CUDA implementation to make use of multiple GPUs working in parallel

Divide mesh between processors (figure on the right)

Overlap CUDA computation with the CUDA memory copy and MPI communication (see pseudo-code below)

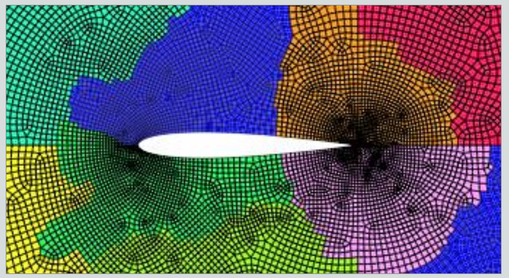
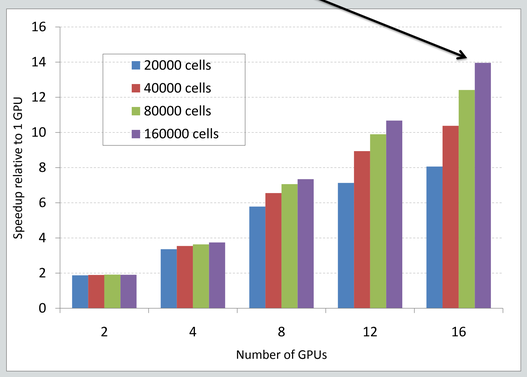


Figure 6: Mesh partition obtained using ParMetis

1.2 Teraflops Double Precision!



Pseudo-Code:

```
// Some code here

Pack_mpi_buf<<<grid_size,block_size>>>;

Cuda_Memcpy(DeviceToHost);
MPI_Isend(mpi_out_buf);
MPI_Irecv(mpi_in_buf);

// Compute discontinuous residual (cell local)

MPI_Wait();
Cuda_Memcpy(HostToDevice);

// Compute continuous residual (using neighboring cell information)
```

Figure 7: Speedup for the MPI-CUDA implementation relative to single GPU code for 6th order calculation

Scalability:

- Multi-GPU code run on up to 16 GPUs
- 2 Tesla C2050s per compute node
- C2050 specs:
  - Memory Bandwidth: 144 GB/s
  - Single precision peak: 1.03 Tflops
  - Double precision peak: 515 Gflops
- 2 Xeon X5670 CPUs per compute node
- PCIe x16 slots
- InfiniBand Interconnect

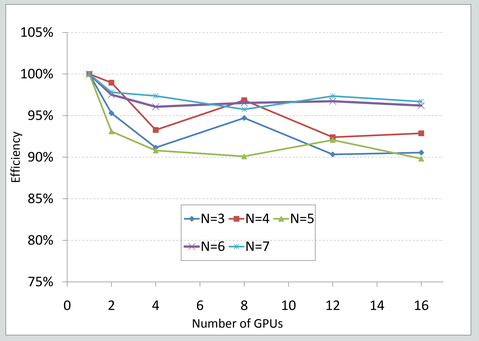


Figure 8: Weak scalability of the MPI-CUDA implementation. Domain size varies from 10000 to 160000 cells

Conclusions and Future Work

Summary:

1. Developed a 2D unstructured compressible viscous flow solver on multiple GPUs
2. Achieved 70x speedup (double precision) on single GPU relative to 1 core of Xeon X5670 CPU
3. Obtained good scalability on up to 16 GPUs and peak performance of 1.2 Teraflops

Future work will involve extension to 3D and application to more computationally intensive cases