Parallel Algorithms for Unsteady Navier-Stokes Flow on GPU Architectures

Bahareh Mostafazadeh, Ferran Marti, Feng Liu, Aparna Chandramowlishwaran
University of California, Irvine
HPC Factory

Motivation & Contributions

- *What are the limitations of today's CFD simulations?*
  - Production CFD codes operate at 3–5% of the architectures peak performance
  - Only exploit homogenous multicore and distributed memory systems
  - Deterioration in convergence rate of the numerical solver with
    - increasing problem size
    - large parallel partitioning
- *What do we gain from a high-performance CFD solver?*
  - Increased efficiency by achieving a high percentage of the theoretical peak machine performance
  - Parallel algorithms that map onto heterogenous multi-CPU multi-GPU
  - architectures
  - Enable next-generation science and engineering problems

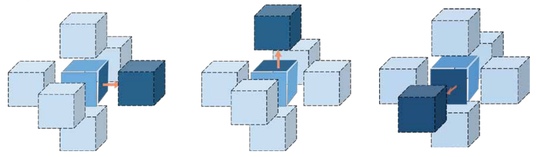
Navier-Stokes Flow

Governing equations

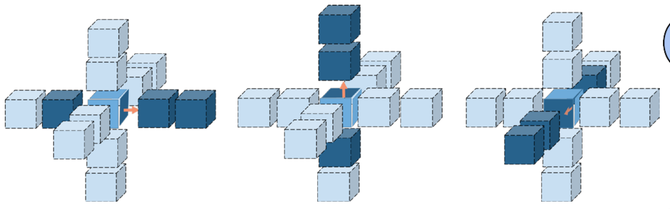
$$\frac{\partial}{\partial t}\iiint_V W\,dV + \iint_S (F_e - F_v).n\,dS = 0 \qquad W = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}$$

$F_e : EulerFlux = InviscidFlux + ArtificialDissipation$
$F_v : ViscousFlux$

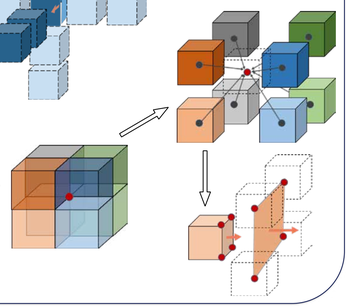
- To solve the Navier Stokes equations above we use:
  - 5 stage Runge Kutta time stepping scheme
  - Central difference with artificial dissipation (2nd order accurate in space)
- Flux calculation (yellow box) takes up ~ 90% of the total execution time
- **Inviscid flux**
  - 5 point stencil in 2D and 7 point stencil in 3D

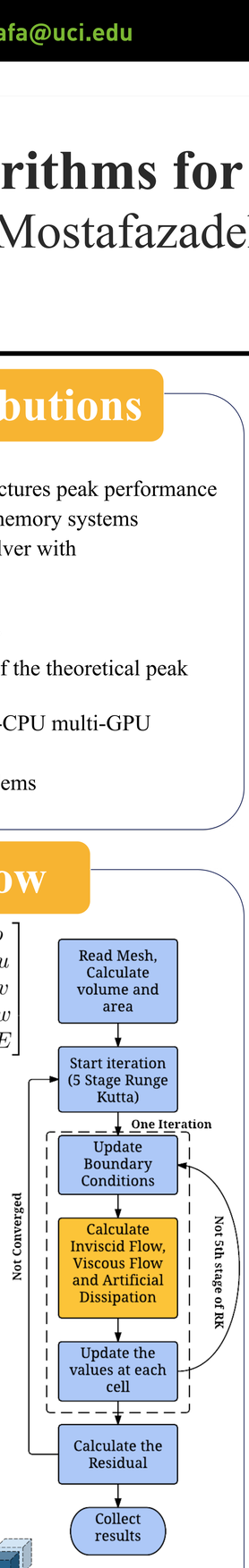


- **Artificial dissipation**
  - 9 point stencil in 2D and 13 point stencil in 3D



- **Viscous flux**
  - Auxiliary grid(dashed line cells in figure, centered at the vertex of original cell) used for calculating the gradients
  - 27 point stencil in 3D, 9 point stencil in 2D





HiPer Algorithm

HiPer was designed to overcome the limitations of today's simulations, with the goal of scalability on current and future heterogeneous systems in mind.

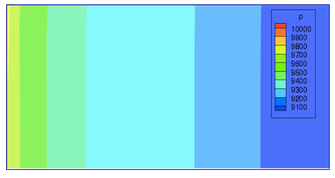
Challenges on GPUs

- Shared memory limitations
  - Cannot store the values of W and surfaces for the entire grid
- Multiple access patterns
  - Inviscid flux stencils are cell centered and read x-cells in each direction
    - Can use registers to store neighboring cells in Z direction
  - Gradient stencils (needed in viscous flux) are centered at the vertex
    - Complex memory access pattern
- Fused stencils
  - Viscous flux calculation for one surface :
    - Four 8 point stencils (to calculate values at vertex) + one 4 point stencil (average the previously obtained results for flux at the surface)
    - Expensive synchronization after values at vertices are calculated
    - Could instead be fused into a non-trivial 18 point stencil

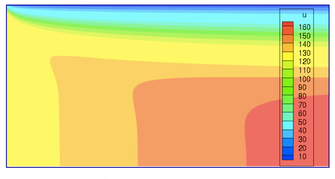
Verification

We use the following simulation in order to verify the obtained results:

- 2D compressible laminar channel flow
- Grid size of 160 x 40, Half channel with symmetry boundary conditions



pressure contour



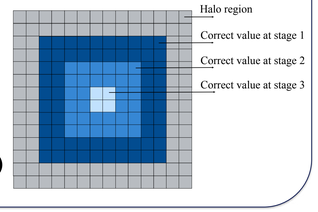
velocity contour

The Evolution of HiPer

We divide the grid into thread blocks with tunable size, and assign the calculation of each cell to one thread.

The baseline implementation only uses the global memory; We then further improved the performance, considering the GPU architecture, with the following optimizations:

- Utilizing the shared memory
  - Stores W and pressure for all cells within a thread block
  - \+ Reduces cost of memory accesses
  - \- Need to copy updated W and pressure back to global memory after each stage of Runge Kutta, to synchronize within thread blocks
- Reducing the cost of synchronization by merging all the 5 stages of RK
  - Does the same calculation without synchronizing after each stage
  - \- Introduces error, since values of neighbors are not up-to-date
- Decreasing the error due to the previous merge optimization
  - Reduce the number of stages to 3
  - Temporal blocking : Redundantly compute the values of previous stages to obtain correct results for a smaller block of cells(details shown in Figure)

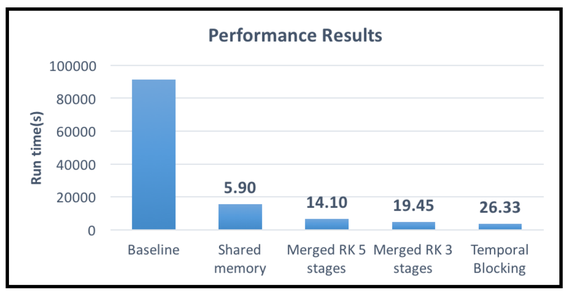


Performance Results

Our GPU code shows 27X speedup compared to the baseline code.

Experiments were run on a system with Intel Xeon E5-2630 CPU and a Tesla K40 GPU.

The results are for a single precision channel flow simulation on a grid of size 2000 x 500 (1 million cells) and CFL = 1.0.



✦ The number above each bar is speedup compared against Baseline.

Future Work

As future work, we plan to continue with this trend in improving these simulations, and add more features to HiPer. The following are some of our future goals:

- **Supporting Multigrid**
  - One of the most common acceleration techniques used in CFD
- **Supporting Turbulence models**
  - This is essential to capture the physics in numerous real-world applications
- **Heterogeneous Algorithms**
  - Computational resources are wasted when either CPU or GPU is left idle. We plan to partition the grid to utilize the entire system to its full potential
- **Supporting other schemes and methods**
  - Exploration of both implicit and explicit numerical schemes in light of the GPU architecture
  - "Best" scheme could be different for each application

References

[1] Liu, F. and Zheng, X., "A Strongly-Coupled Time-Marching Method for Solving the Navier-Stokes and k-ω Turbulence Model Equations with Multigrid," Journal of Computational Physics, Vol. 128, No. 2, pp. 289-300, 1996.
[2] Paulius Micikevicius. 2009. 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units* (GPGPU-2). ACM, New York, NY, USA, 79-84.
[3] A. D. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-d Blocking Optimization for Stencil Computations on Modern CPUs and GPUs," in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, pp. 113. IEEE Computer Society, Washington, DC, USA, 2010