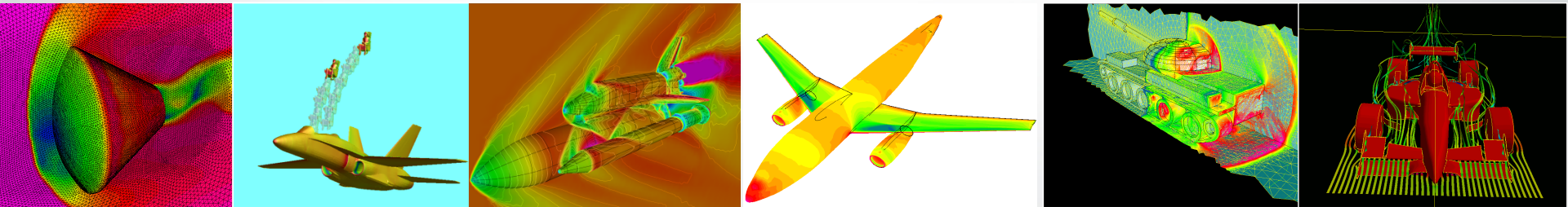
Development of a Portable, GPU-Accelerated High-Order Discontinuous Galerkin CFD Code for Compressible Flows on Hybrid Grids

Yidong Xia, Lixiang Luo, Jialin Lou, Hong Luo and Jack Edwards
Department of Mechanical and Aerospace Engineering
North Carolina State University

and

Frank Mueller and Nishanth Balasubramanian
Department of Computer Science
North Carolina State University

February 7, 2014



People at NC State

Faculty (Co-PI's)

Dr. Hong Luo
Professor, Department of Mechanical and Aerospace Engineering

Dr. Jack Edward
Professor, Department of Mechanical and Aerospace Engineering

Dr. Frank Mueller
Professor, Department of Computer Science

Postdocs

Dr. Yidong (Tim) Xia
Current principal developer of the RDGFLO code

Dr. Lixiang (Eric) Luo
Investigation of frontier challenges and solutions in GPU computing; developer of INCOMP3D

Students

Mr. Jialin (Johnny) Lou
Ph.D. student of Aerospace Engineering; current developer of the RDGFLO code

Mr. Nishanth Balasubramanian
Master student of Computer Science; investigation of GPU computing models for CFD programming

Outline

① A brief overview

② Motivation and objective

③ Governing equations of fluid dynamics

④ Discontinuous Galerkin formulation

⑤ OpenACC-based parallelism

⑥ Numerical examples

⑦ Concluding remarks

⑧ Future work

GPU Computing in CFD – Overview

GPGPU

General-purpose computing on graphics processing units

- The utilization of GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by CPU.

Why?

The great potential and scalability of GPGPU for CFD applications!

How?

Offload the computing-intensive portion from the host CPUs to the GPU devices.

Programming models

- **OpenCL**: the currently dominant open GPGPU programming language
- **NVIDIA's CUDA**: the dominant proprietary framework
- **OpenACC**: a collection of directives designed to simplify parallel programming

GPU Computing in CFD – Overview

A variety of applications (selected)

- **For the finite difference method (FDM)**
  - ❑ E. Elsen, P. LeGresley, and E. Darve.
    Large Calculation of the Flow over a Hypersonic Vehicle Using a GPU.
    J. Comput. Phys., 227(24):10148–10161, 2008.
- **For the finite volume method (FVM)**
  - ❑ A. Corrigan, F. Camelli, R. Löhner, and J. Wallin.
    Running Unstructured Grid-based CFD Solvers on Modern Graphics Hardware.
    Int. J. Numer. Methods Fluids, 66(2):221–229, 2011.
- **For the spectral difference method (SDM)**
  - ❑ B. Zimmerman, Z. Wang, and M. Visbal.
    High-Order Spectral Difference: Verification and Acceleration Using GPU Computing.
    AIAA Paper, 2013-2491, 2013.
- **For the discontinuous Galerkin method (DGM)**
  - ❑ A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven
    Nodal Discontinuous Galerkin Methods on Graphics Processors.
    J. Comput. Phys., 228(21):7863–7882, 2009.

Observation: most GPU-related CFD solvers are based on CUDA

Motivation & Objectives

Motivation

Tap the power of GPU parallel computing for the aerodynamic design of Unmanned Aerial Vehicle (UAV) with CFD toolkits

MQ-1 Predator UAV (Source: grabcad.com CAD by Chao Wey)

Motivation & Objectives

Objectives

A portable, efficient and ultimately competitive GPU parallelization strategy for extensible and sustainable high-fidelity CFD code development

An unavoidable debate – CUDA, OpenCL or OpenACC?

It really depends on your needs, e.g., for us, **cross-platform portability**

- Support from **a wide range of compiler and accelerator vendors**
- C/C++ and **Fortran**
- Best performance
- Less fine-tuning effort, **especially for a legacy code package**
- Available computing resource

GPU-Computing Framework

Option 1. – based on NVIDIA's CUDA

- **Why CUDA?**
  - ❑ Mature and ever-updating GPU parallelization standards for HPC
  - ❑ Currently wide user community support
  - ❑ Achievable optimal performance with fine-tuning
  - ❑ Strong GPU-accelerated library support, e.g., CULA tools
  - ❑ ……
- **Why not CUDA?**
  - ❑ Complex and explicit layout of threads on GPU for each kernel function
  - ❑ Excessive workload to upgrade an existing CFD package
  - ❑ Uncertainty in the vendor's long-term development strategy
  - ❑ Constrained **portability** of the developed code on non-CUDA devices

CUDA alone can not satisfy not only our primary design goals, but also many others who hesitate to adopt GPU computing!

GPU-Computing Framework

Option 2. – based on OpenACC

Directives for accelerators http://www.openacc-standard.org/

- **Why OpenACC?**
  - ❑ Simple directive-based GPU parallelization strategy, similar to OpenMP
  - ❑ Multi-compiler / multi-platform support
  - ❑ Growing supporting community
- **Why not OpenACC?**
  - ❑ If a fine-tuned, best-performance code is what you pursue
  - ❑ A number of limitations compared with CUDA

Our choice: OpenACC meets most of our design requirements!

GPU-Computing Framework

Desirable features of the resulting CFD code

- **Multi-compiler compatibility**
  - ❑ GNU Fortran compiler
  - ❑ Intel Fortran compiler
  - ❑ PGI Accelerator Fortran compiler (with **OpenACC** support)
  - ❑ CAPS Fortran compiler (with **OpenACC** support)

- **Cross-platform portability**
  - ❑ Intel CPUs
  - ❑ AMD CPUs / APUs (with potential **OpenACC** support in 2014)
  - ❑ NVIDIA CUDA-enabled GPUs (with **OpenACC** support)

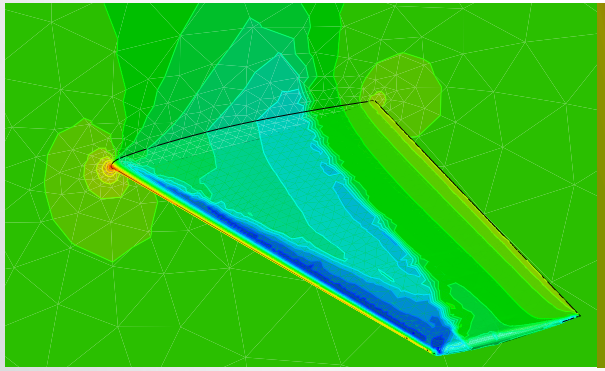
- **Extensible and sustainable programming schemes**
- **Competitive performance**

Legacy CFD Package

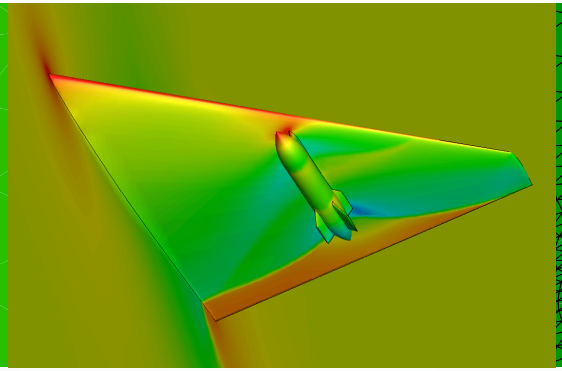
RDGFLO – a baseline code for OpenACC-based GPU parallelization

- **A Reconstructed Discontinuous Galerkin finite element FLOw solver**
  - ❑ High-order solution of compressible flows on 3-D hybrid grids
  - ❑ Explicit / implicit solution schemes
  - ❑ Domain-partition based MPI parallel computing

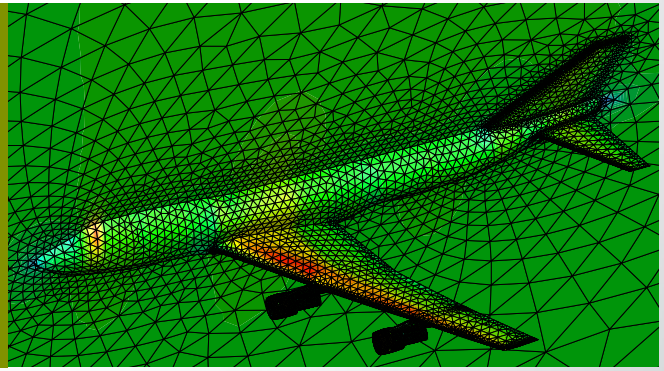
Gallery



ONERA M6 wing



wing/pylon/finned-store configuration



Boeing 747 aircraft

Governing Equations of Fluid Dynamics

The Navier-Stokes equations for unsteady compressible flows

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_k(\mathbf{U}, t)}{\partial \mathbf{x}_k} = \frac{\partial \mathbf{G}_k(\mathbf{U}, \nabla \mathbf{U}, t)}{\partial \mathbf{x}_k}$$

where the summation convention is used. The conservative variable vector **U**, advective flux vector **F**, and viscous flux **G** are defined by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix} \qquad \mathbf{F}_j = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p\delta_{ij} \\ u_j(\rho e + p) \end{pmatrix} \qquad \mathbf{G}_j = \begin{pmatrix} 0 \\ \tau_{ij} \\ u_l \tau_{ij} + q_j \end{pmatrix}$$

Governing Equations of Fluid Dynamics

The Navier-Stokes equations for unsteady compressible flows (cont.)

The pressure p can be computed from the equation of state (EOS)

$$p = (\gamma - 1)(\rho e - \frac{1}{2}\rho u_k u_k)$$

which is valid for perfect gas. The ratio of specific heats γ is assumed to be constant and equal to 1.4.

The viscous stress tensor τ_{ij} and heat flux vector q_j are given by

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij} \qquad q_j = \frac{1}{\gamma - 1}\frac{\mu}{Pr}\frac{\partial T}{\partial x_j}$$

where T is the temperature of the fluid, Pr the laminar Prandtl number, which is 0.7 for air. μ represents the molecular viscosity, which can be determined through the Sutherlands law

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0}\right)^{3/2}\frac{T_0 + S}{T + S}$$

where μ_0 is the viscosity at the reference temperature T_0 and S = 110K.

Discontinuous Galerkin Method

Weak formulation of the governing equations

$$\frac{d}{dt}\int_{\Omega_e}\mathbf{U}_h B_i\,d\Omega+\int_{\Gamma_e}\mathbf{F}_k\mathbf{n}_k B_i\,d\Gamma-\int_{\Omega_e}\mathbf{F}_k\frac{\partial B_i}{\partial \mathbf{x}_k}d\Omega=$$

$$\int_{\Gamma_e}\mathbf{G}_k\mathbf{n}_k B_i\,d\Gamma-\int_{\Omega_e}\mathbf{G}_k\frac{\partial B_i}{\partial \mathbf{x}_k}d\Omega,\ 1\le i\le N$$

where $B_i(\mathbf{x})$ is the basis function of polynomials of degree p, and N is the dimension of the polynomial space p.

The Taylor-basis discontinuous Galerkin (DG(p)) solution in each element

$$\mathbf{U}=\sum_{i=1}^{N}\mathbf{U}_i B_i$$

For example, the underlying piecewise linear polynomial DG(P1) solution

$$\mathbf{U}=\bar{\mathbf{U}}+\left(\frac{\partial \mathbf{U}}{\partial x}\Delta x\right)\frac{x-x_c}{\Delta x}+\left(\frac{\partial \mathbf{U}}{\partial y}\Delta y\right)\frac{y-y_c}{\Delta y}+\left(\frac{\partial \mathbf{U}}{\partial z}\Delta z\right)\frac{z-z_c}{\Delta z}$$

where (x_c, y_c, z_c) is the coordinate of the element center.

Discontinuous Galerkin Method

Hierarchical WENO reconstruction

- **A quadratic polynomial DG(P2) solution is obtained via a hierarchical WENO reconstruction approach in each element**

  1) Least-squares reconstruction to obtain an initial quadratic solution
  2) WENO reconstruction for the second derivatives to maintain linear stability
  3) WENO reconstruction for the first derivatives to maintain nonlinear stability

- **Reference articles**

  - ❑ H. Luo, Y. Xia, S. Li, and R. Nourgaliev.
    A Hermite WENO Reconstruction-Based Discontinuous Galerkin Method for the Euler Equations on Tetrahedral grids.
    J. Comput. Phys. 231(16):5489–5503, 2012.
  - ❑ H. Luo, Y. Xia, S. Spiegel, R. Nourgaliev, and Z. Jiang.
    A Reconstructed Discontinuous Galerkin Method Based on a Hierarchical WENO Reconstruction for Compressible Flows on Tetrahedral Grids.
    J. Comput. Phys. 236:477–492, 2013.

Discontinuous Galerkin Method

Semi-discrete form

A system of ordinary differential equations (ODEs) in time

$$\boldsymbol{M}\frac{d\mathbf{U}}{dt} = \mathbf{R}$$

where $\boldsymbol{M}$ is the mass matrix and $\boldsymbol{R}$ is the residual vector.

Three-stage TVD Runge-Kutta (TVDRK3) time stepping

$$\mathbf{U}^{(1)} = \mathbf{U}^{n} + \Delta t \boldsymbol{M}^{-1}\mathbf{R}(\mathbf{U}^{n})$$

$$\mathbf{U}^{(2)} = \frac{3}{4}\mathbf{U}^{n} + \frac{3}{4}\left(\mathbf{U}^{(1)} + \Delta t \boldsymbol{M}^{-1}\mathbf{R}(\mathbf{U}^{(1)})\right)$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\mathbf{U}^{n} + \frac{2}{3}\left(\mathbf{U}^{(2)} + \Delta t \boldsymbol{M}^{-1}\mathbf{R}(\mathbf{U}^{(2)})\right)$$

Design of OpenACC Parallel Regions

Computing-intensive regions

1. ~65% Internal & boundary face integral: loop over mesh faces T_e

$$\int_{\Gamma_e} \mathbf{F}_k \mathbf{n}_k B_i \, d\Gamma \qquad \int_{\Gamma_e} \mathbf{G}_k \mathbf{n}_k B_i \, d\Gamma$$

2. ~25% Domain integral: loop over mesh elements Ω_e

$$\int_{\Omega_e} \mathbf{F}_k \frac{\partial B_i}{\partial \mathbf{x}_k} d\Omega \qquad \int_{\Omega_e} \mathbf{G}_k \frac{\partial B_i}{\partial \mathbf{x}_k} d\Omega$$

3. ~5% TVDRK3 time stepping: loop over mesh elements Ω_e

Design of OpenACC Parallel Regions

Example: a readily vectorizable region

Domain integral: loop over mesh elements Ω_e $\int_{\Omega_e} \mathbf{F}_{ki}\, \partial B / \partial \mathbf{x}_k \, d\Omega$

OpenMP version

```
!$omp parallel
!$omp do
do ie = 1, Nelem
   do ig = 1, Ngp
      !... contribution to this element
      rhsel(:, :, ie) = rhsel(:, :, ie) + flux
   enddo
enddo
!$omp end parallel
```

OpenACC version

```
!$acc parallel
!$acc loop
do ie = 1, Nelem
   do ig = 1, Ngp
      !... contribution to this element
      rhsel(:, :, ie) = rhsel(:, :, ie) + flux
   enddo
enddo
!$acc end parallel
```

Design of OpenACC Parallel Regions

Example: a region that is not readily vectorizable

Face integral: loop over mesh faces T_e $\int_{\Gamma_e} \mathbf{F}_k \mathbf{n}_k B_i \, d\Gamma$

OpenMP version

```
!$omp parallel
!$omp do
do ifa = Njfac+1, Nafac
   do ig = 1, Ngp
      !... contribution to the face-left element
      rhsel(:, :, iel) = rhsel(:, :, iel) - flux
      !... contribution to the face-right element
      rhsel(:, :, ier) = rhsel(:, :, ier) + flux
   enddo
enddo
!$omp end parallel
```

OpenACC version

```
!$acc parallel
!$acc do
do ifa = Njfac+1, Nafac
   do ig = 1, Ngp
      !... contribution to the face-left element
      rhsel(:, :, iel) = rhsel(:, :, iel) - flux
      !... contribution to the face-right element
      rhsel(:, :, ier) = rhsel(:, :, ier) + flux
   enddo
enddo
!$acc end parallel
```

"race condition" – multiple writes to the same elemental residual vector!

Design of OpenACC Parallel Regions

A common method for avoiding "race condition" in face integral

- **Perform face integral at the element level**
  - ❑ All the computing is implemented as loops over mesh elements
- **Overheads**
  - ❑ Lead to redundant computation of face integrals (doubled!)
  - ❑ Require an additional element-face connectivity array
- **Reference articles on unstructured DG/FV methods (with CUDA)**
  - ❑ A. Corrigan et al. Running Unstructured Grid-based CFD Solvers on Modern Graphics Hardware. *Int. J. Numer. Methods Fluids*, 66(2):221–229, 2011.
  - ❑ Tristan Cabel and Stephane Lanteri. Discontinuous Galerkin Time-Domain Solver on GPU Based Systems. Plafrim meeting, May 31, 2011

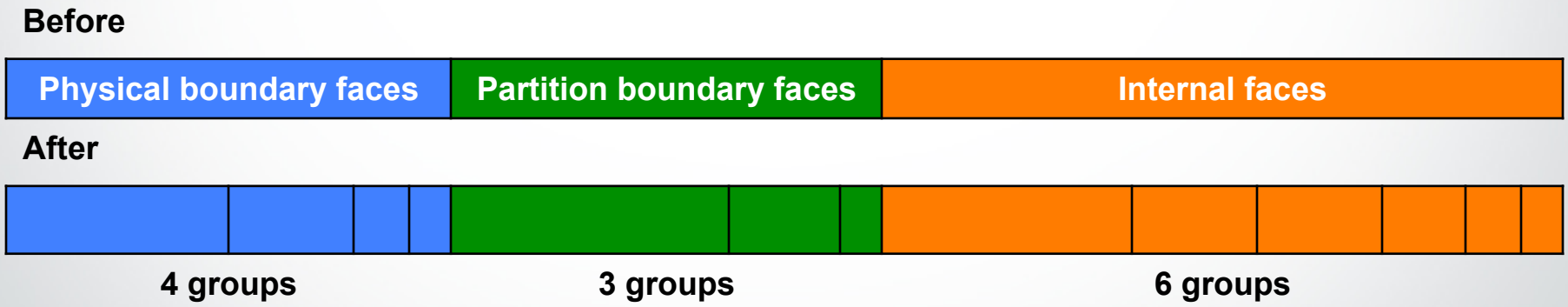
This design approach requires a major rebuild in code structures!

Design of OpenACC Parallel Regions

An alternative approach to avoiding "race condition" in face integral

- **Face renumbering (coloring method)**
  - ❑ renumber the faces and split them into several groups.
  - ❑ Any two faces that share a common cell do not reside in the same group.
  - ❑ The original face loops are nested in a sequential loop over groups.

An example of "face renumbering & grouping"



Design of OpenACC Parallel Regions

Example: face loops after "face renumbering & grouping"

The original face loops are nested in a sequential loop over groups.

OpenMP version

```
Nfac1 = Njfac
do ipass = 1, Npass_ift
   Nfac0 = Nfac1 + 1
   Nfac1 = fpass_ift(ipass)
   !$omp parallel
   !$omp do
   do ifa = Nfac0, Nfac1
      do ig = 1, Ngp
         !... contribution to the face-left element
         rhsel(:, :, iel) = rhsel(:, :, iel) - flux
         !... contribution to the face-right element
         rhsel(:, :, ier) = rhsel(:, :, ier) + flux
      enddo
   enddo
   !$omp end parallel
enddo
```

OpenACC version

```
Nfac1 = Njfac
do ipass = 1, Npass_ift
   Nfac0 = Nfac1 + 1
   Nfac1 = fpass_ift(ipass)
   !$acc parallel
   !$acc do
   do ifa = Nfac0, Nfac1
      do ig = 1, Ngp
         !... contribution to the face-left element
         rhsel(:, :, iel) = rhsel(:, :, iel) - flux
         !... contribution to the face-right element
         rhsel(:, :, ier) = rhsel(:, :, ier) + flux
      enddo
   enddo
   !$acc end parallel
enddo
```

Design of OpenACC Parallel Regions

"Face renumbering & grouping" vs. "element loops"

- **Advantages**
  - ❑ No redundant computation.
  - ❑ Least intrusion to the original code structures.
  - ❑ Recoverable to CPU parallel computing.
- **Disadvantages**
  - ❑ Sequential groups lead to overheads in initializing more parallel kernels.
- **Remarks**
  1. No direct comparison of these two strategies is yet conducted for our solver. It is unknown which will render better performance on GPU.
  2. Since **portability** is our design priority, the current approach is a preferred, although **optimal performance** might be compromised.

Hardware/Software Resource

CPU

- **2-way SMPs with AMD Opteron 6128 (Many Core) with 8 cores per socket (16 cores per node)**
  - ❑ 32 GB DRAM
  - ❑ 2.0 GHz core-speed for 6128 Opteron (single core)

GPU

- **NVIDIA Tesla K20c**
  - ❑ Memory amount: **5.0 GB**
  - ❑ Stream processors: **2496**

Software (64 bit)

- **Operating system**
  - ❑ CentOS 5.7 Linux x86 64
- **Compilation & runtime suite**
  - ❑ PGI Accelerator Fortran Compiler (ver. 13.4)
  - ❑ OpenMPI (ver. 1.5.5)

Performance Assessment

Timing measurements

Unit running time T_{unit}

$$T_{\text{unit}} = \frac{T_{\text{run}}}{\text{Ntime} \times \text{Nelem}} \times 10^6 \ \text{(microsecond)}$$

where the running time T_{run} refers to the time recorded for completing the time marching loop with a given iteration number Ntime.

Control sets (simulations in double-precision arithmetic)	
GPU-1	Parallel computing with **1 GPU device**
CPU-1	Serial computing with **1 CPU core**
CPU-16	Parallel computing with **16 CPU cores** (domain partitioning + MPI)

Example 1. Subsonic Flow past a Sphere

Subsonic flow past a sphere at $M_\infty = 0.50$ and $\alpha = 0°$.

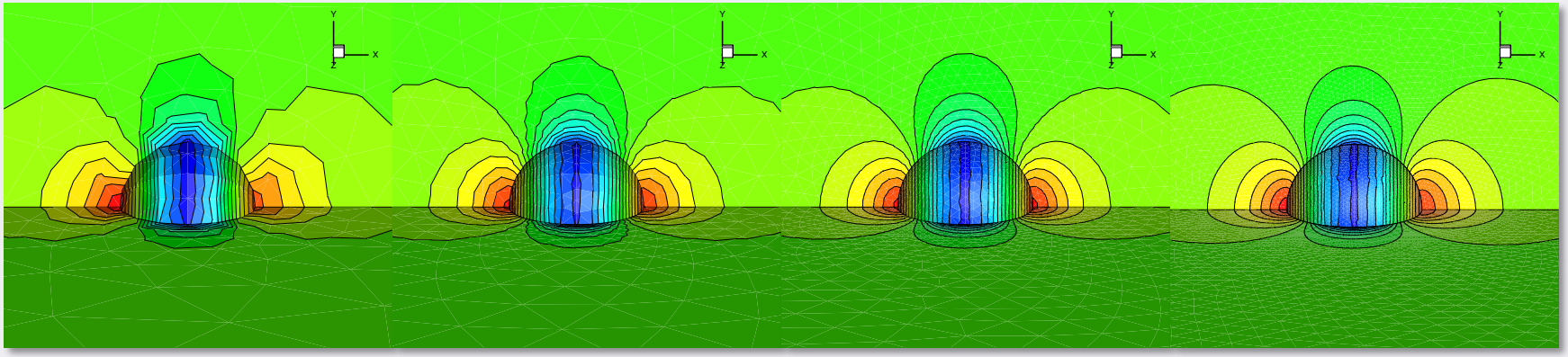
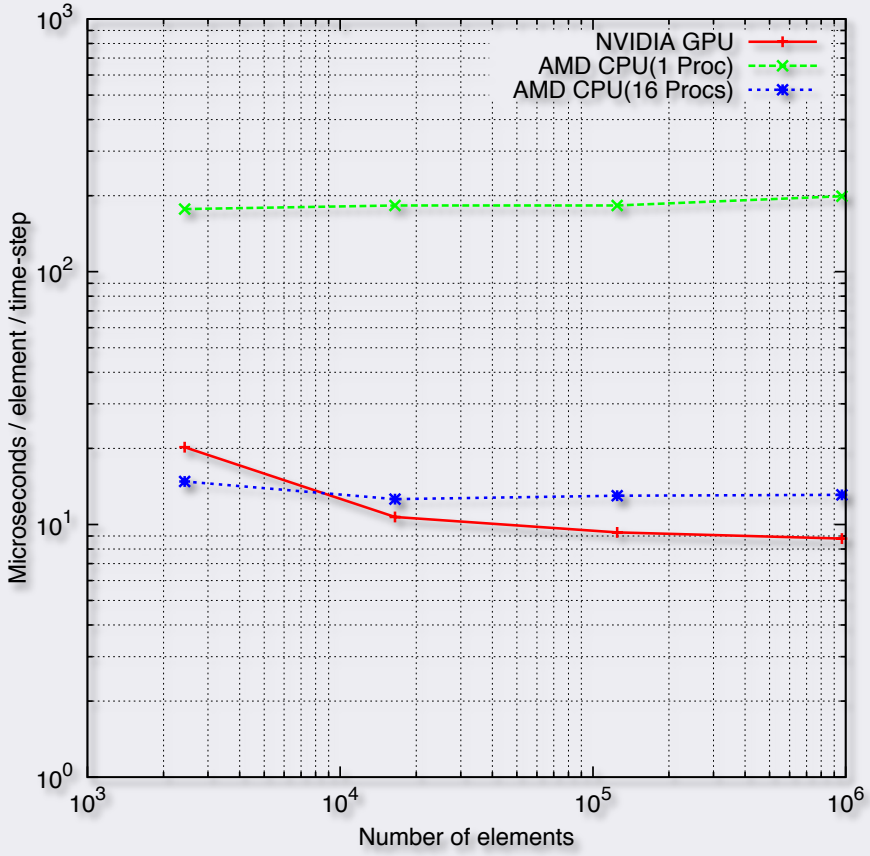


Figure. Pressure contours plotted on the surface triangular meshes of a sequence of four successively refined tetrahedral grids

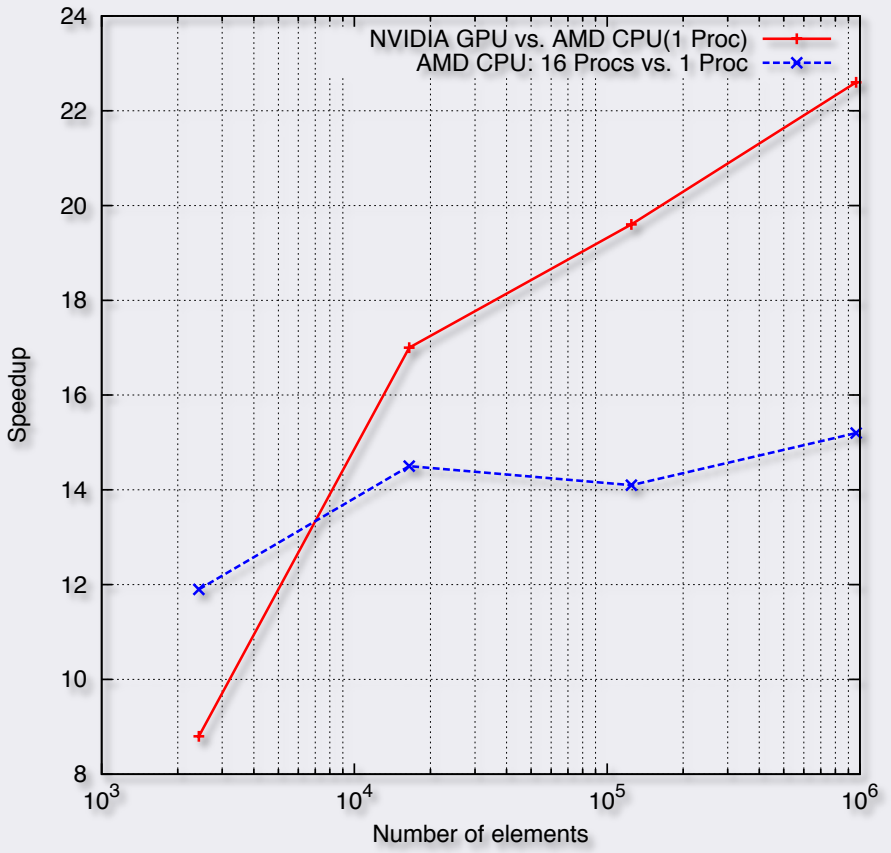
	Unit time (microsecond)			Speedup	
Nelem	GPU-1	CPU-1	CPU-16	vs. CPU-1	vs. CPU-16
2,426	20.2	176.8	14.8	8.8	0.73
16,467	10.7	182.8	12.6	17.0	1.18
124,706	9.3	182.8	13.0	19.6	1.40
966,497	8.8	198.9	13.1	22.6	1.49

Example 1. Subsonic Flow past a Sphere

Subsonic flow past a sphere at $M_\infty = 0.50$ and $\alpha = 0°$.



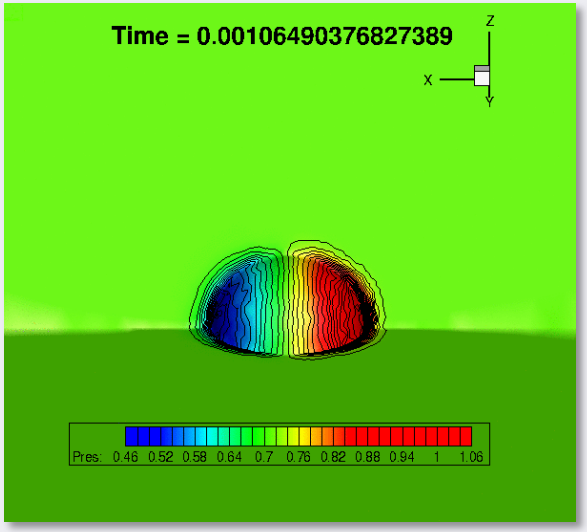
Unit running time vs. Nr. of elements



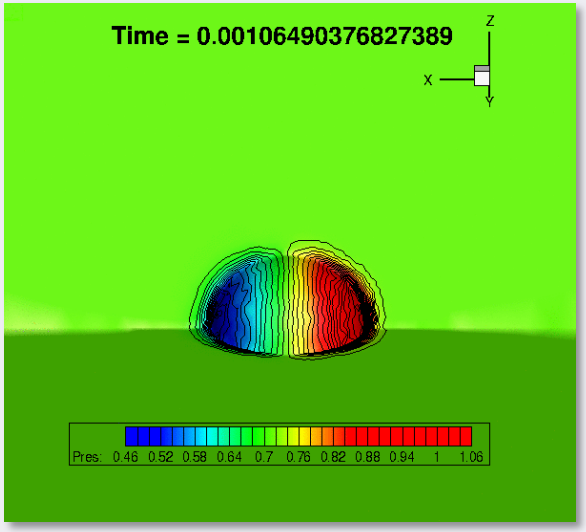
Speedup vs. Nr. of elements

Example 1. Subsonic Flow past a Sphere

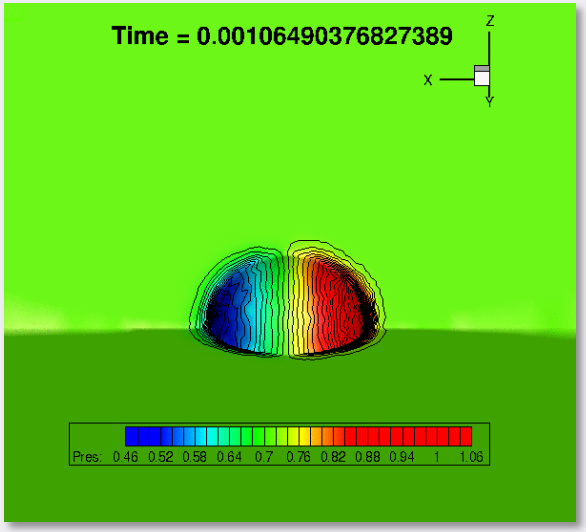
Subsonic flow past a sphere at $M_\infty = 0.50$ and $\alpha = 0°$.



GPU-1



CPU-16



CPU-1

Anim. Comparison of runtime performance rendered by pressure contours on the surface meshes

Example 2. Transonic Flow over a Boeing 747

Transonic flow over a Boeing 747 aircraft at M_∞ = 0.85 and α = 2◦

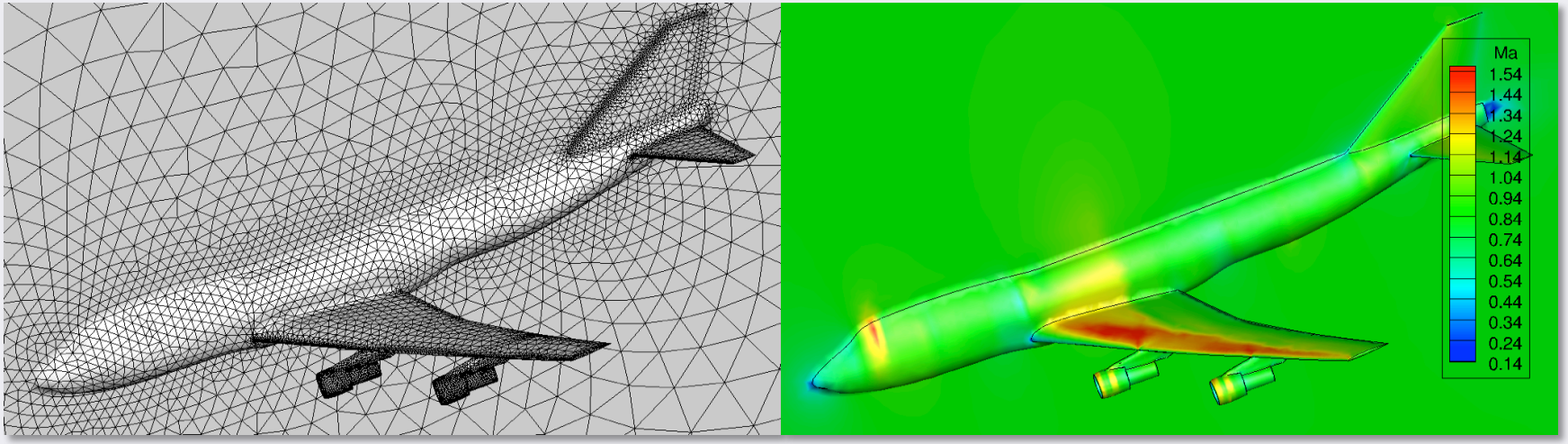
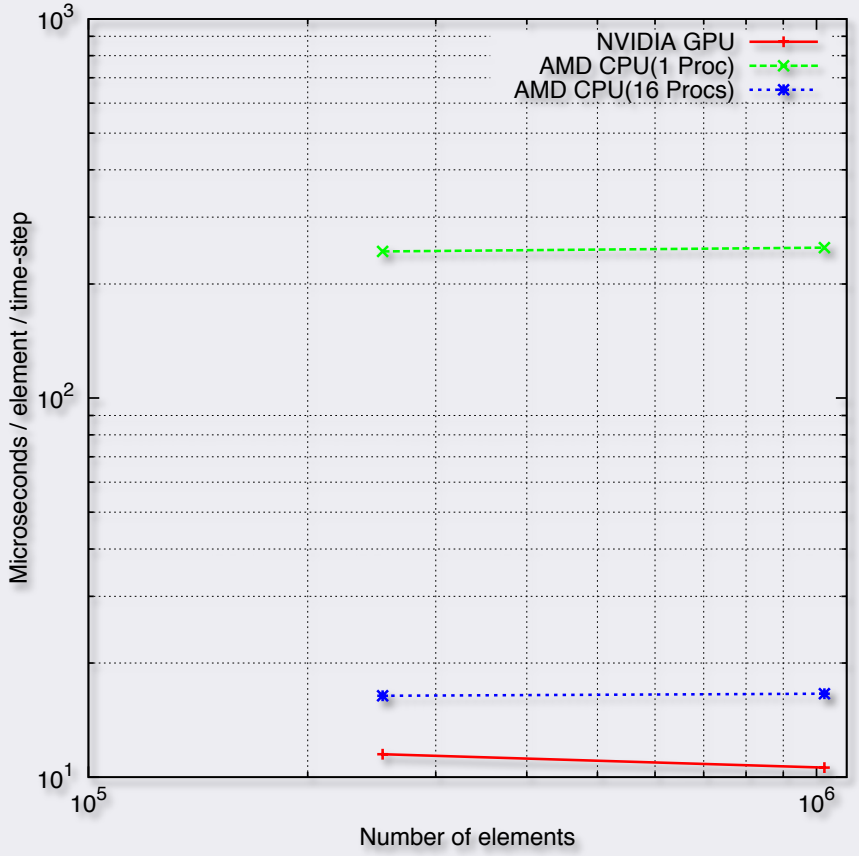


Figure. Mach number contours plotted on the surface triangular meshes

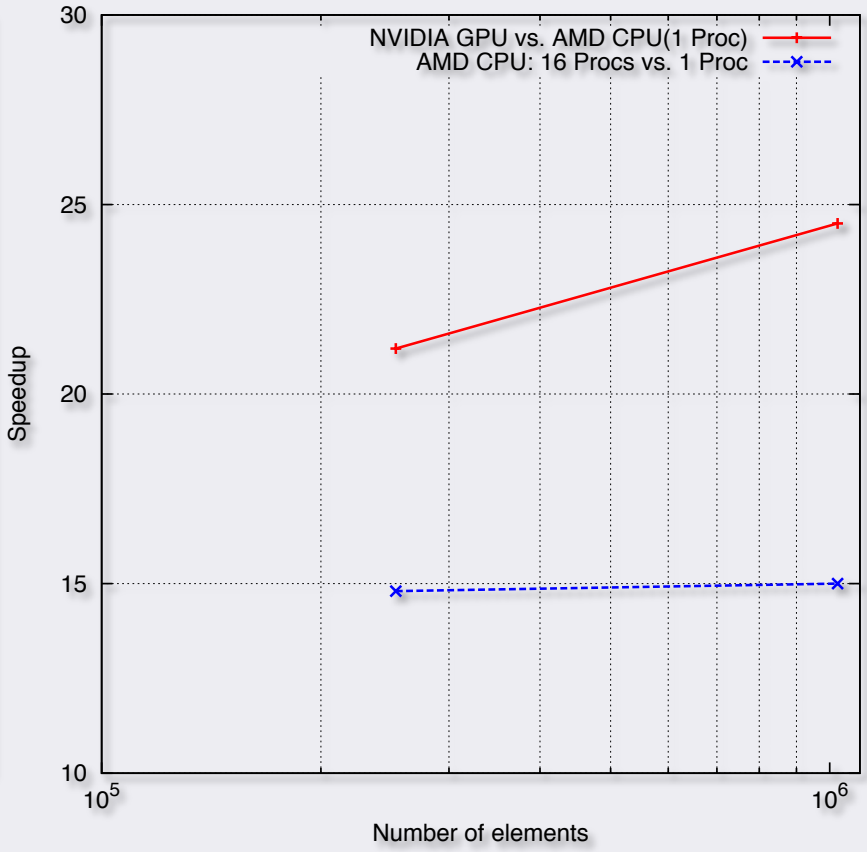
	Unit time (microsecond)			Speedup	
Nelem	GPU-1	CPU-1	CPU-16	vs. CPU-1	vs. CPU-16
253,577	11.5	243.8	16.4	21.2	1.43
1,025,170	10.6	249.4	16.6	24.5	1.57

Example 2. Transonic Flow over a Boeing 747

Transonic flow over a Boeing 747 aircraft at $M_\infty = 0.85$ and $\alpha = 2^\circ$



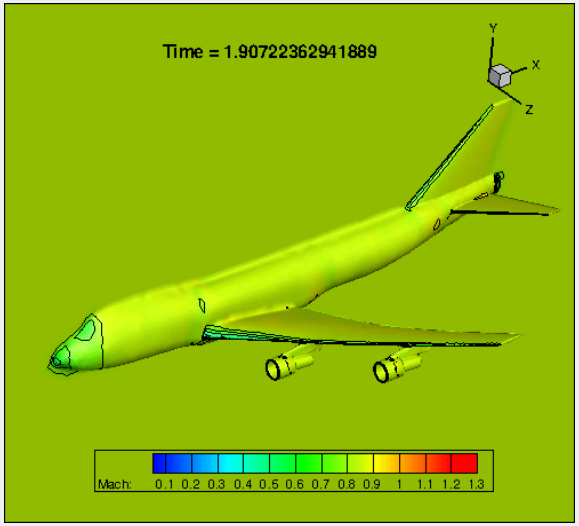
Unit running time vs. Nr. of elements



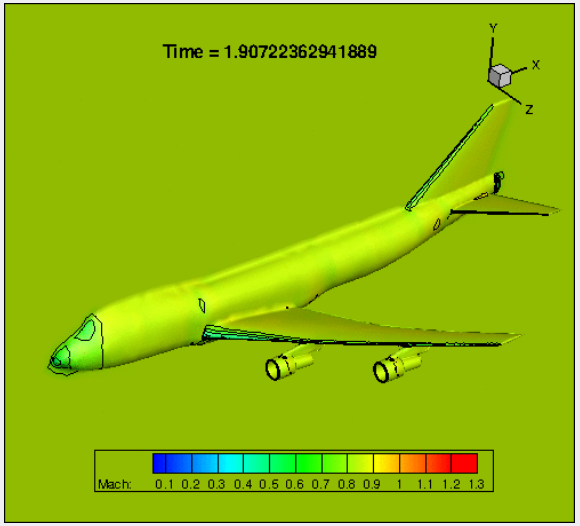
Speedup vs. Nr. of elements

Example 2. Transonic Flow over a Boeing 747

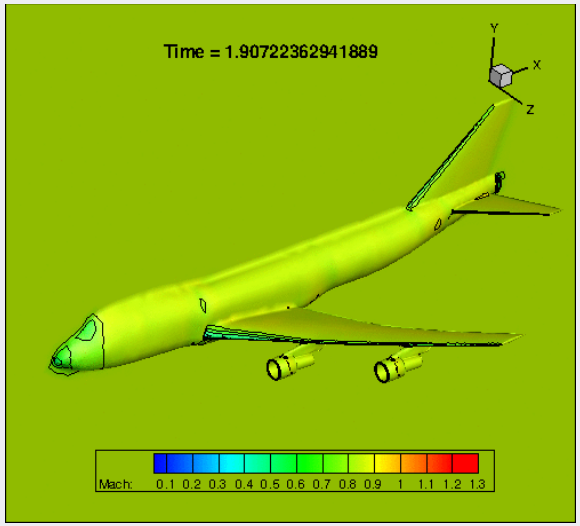
Transonic flow over a Boeing 747 aircraft at M_∞ = 0.85 and α = 2◦



GPU-1



CPU-16

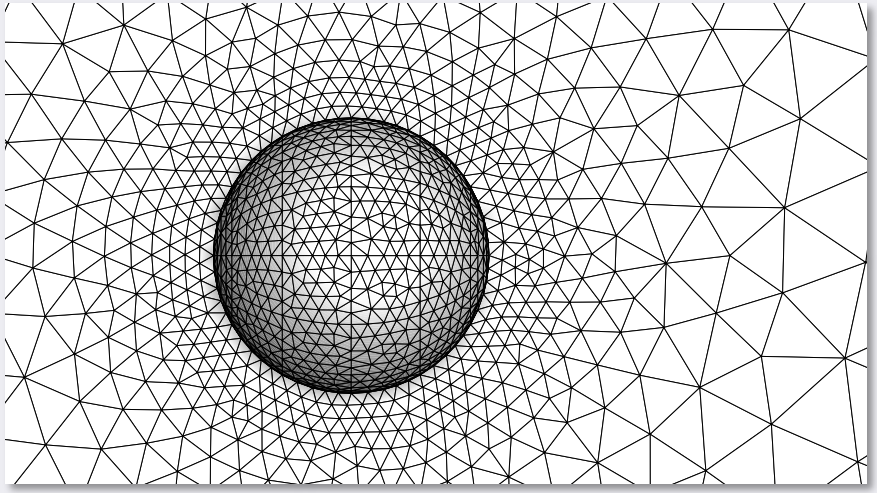


CPU-1

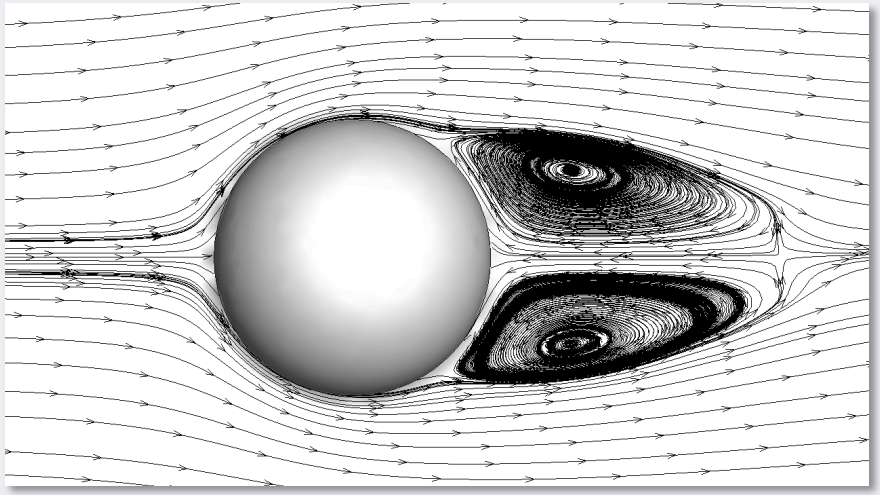
Anim. Comparison of runtime performance rendered by Mach number contours on the surface meshes

Example 3. Viscous Flow past a Sphere

Subsonic flow past a sphere at M_∞ = 0.5 and Re = 118



Triangular surface meshes

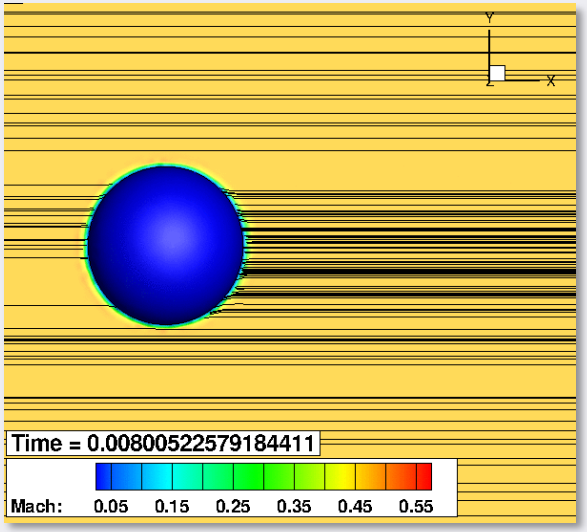


Streamtraces on the symmetry plane

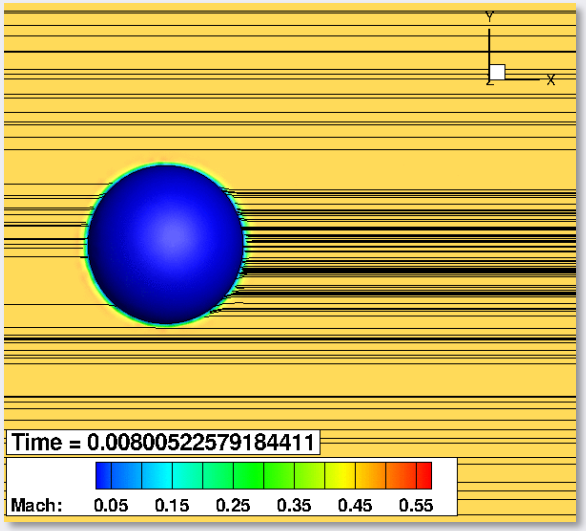
	Unit time (microsecond)			Speedup	
Nelem	GPU-1	CPU-1	CPU-16	vs. CPU-1	vs. CPU-16
200,416	14.6	259.9	20.5	17.8	1.41
925,995	13.9	257.2	20.6	18.5	1.48

Example 3. Viscous Flow past a Sphere

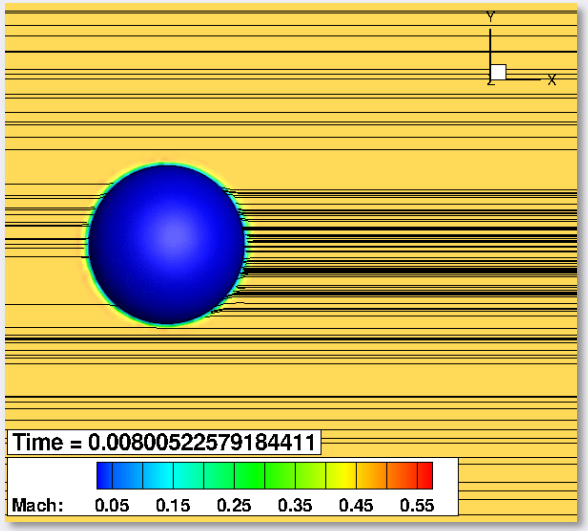
Subsonic flow past a sphere at M_∞ = 0.5 and Re = 118



GPU-1



CPU-16

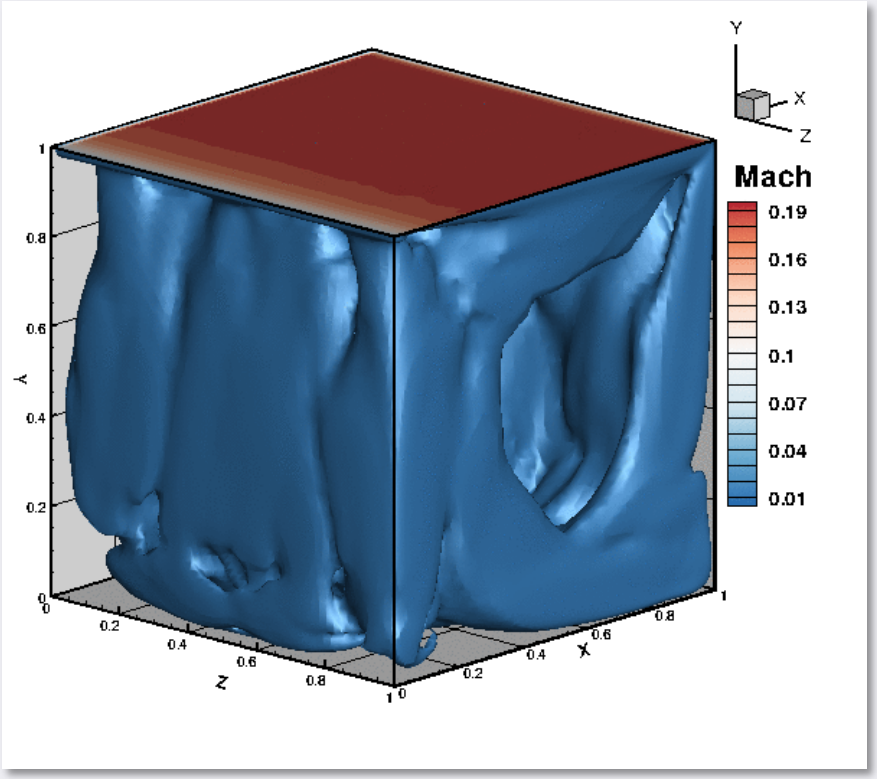


CPU-1

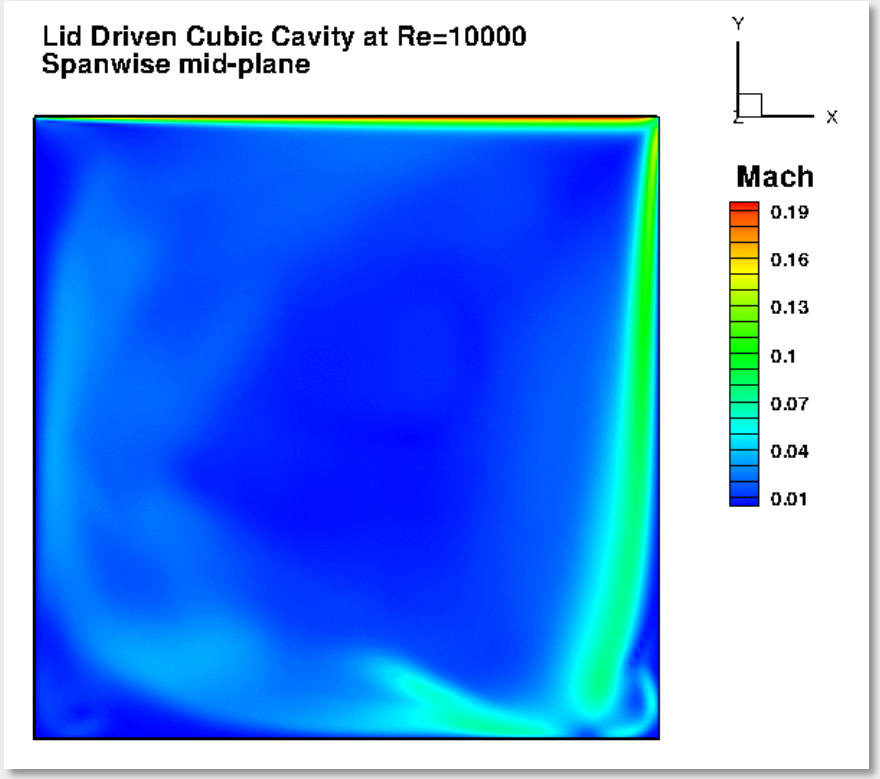
Anim. Comparison of runtime performance rendered by Mach number contours along with streamtraces on the symmetry plane

Example 4. Large Eddy Simulation (to be completed soon…)

Lid-driven cubical cavity at M_b = 0.2 and Re = 10,000



Instantaneous Mach number iso-surfaces in the cubical domain



Instantaneous Mach number contours on spanwise mid-plane

Simulation of incompressible flows using a compressible flow solver!

Concluding Remarks

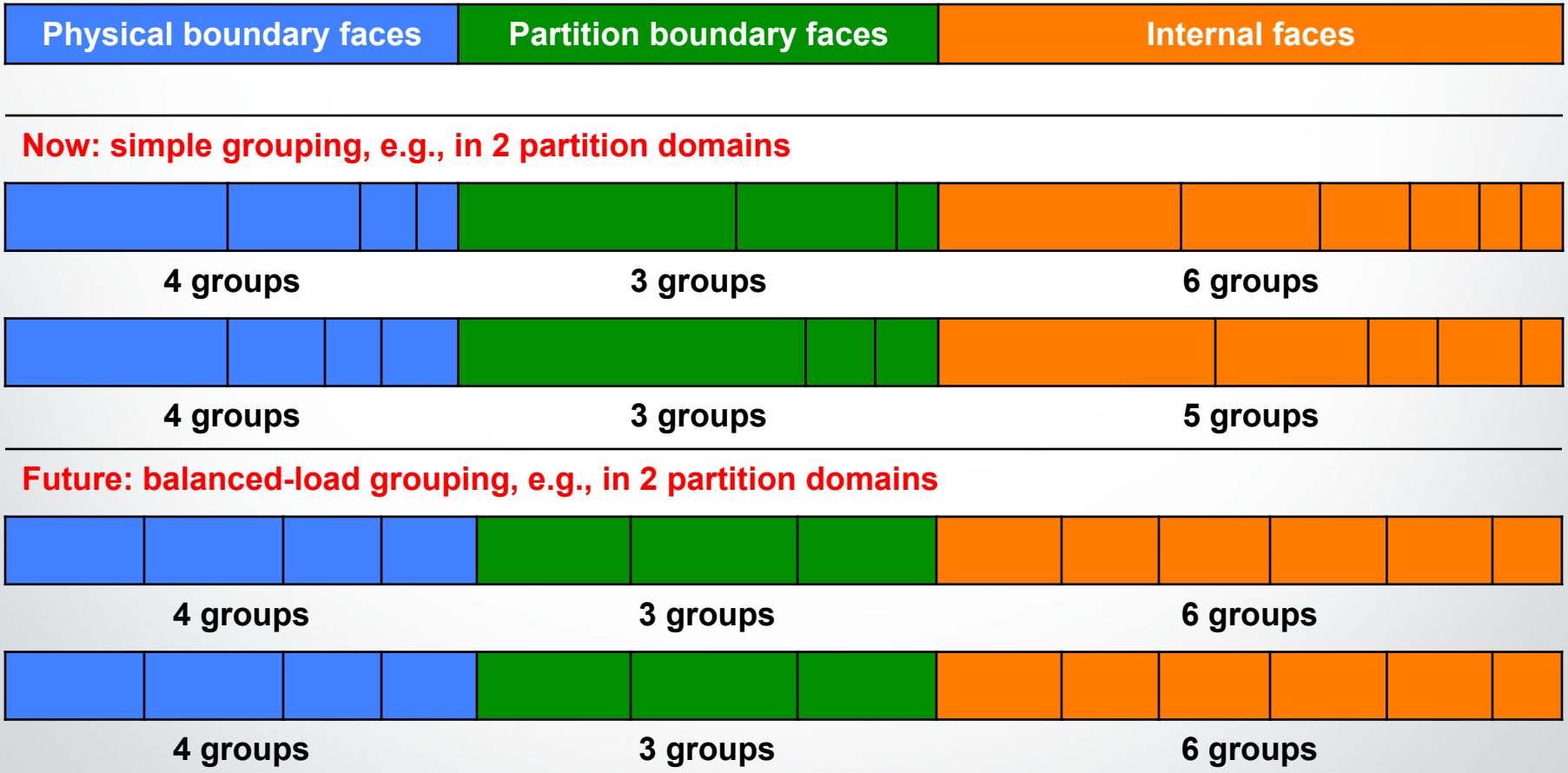
RDGFLO with OpenACC

- **Performance on single GPU**
  - ❑ An average scaling factor of over **20.0x** vs. 1 CPU core
  - ❑ An average scaling factor of over **1.4x** vs. 16 CPU cores
- **Extensibility**
  - ❑ The current parallel strategy can be applied to more functionalities
- **Verified compilers**
  - ❑ PGI Fortran
- **Verified platform**
  - ❑ NVIDIA CUDA-enabled GPUs
- **Limitation**
  - ❑ Memory constraint on GPU device
  - ❑ Balance between [optimal GPU parallelism] and [code portability]
- **Publications**
  - ❑ Y. Xia, L. Luo, H. Luo, J. Edwards, J. Lou, and F. Mueller. OpenACC-based GPU Acceleration of a 3-D Unstructured Discontinuous Galerkin Method. 52nd AIAA Aerospace Sciences Meeting, AIAA-2014-1129, January 2014.

Future Work & Prospect

1. Multi-GPU parallelization via MPI

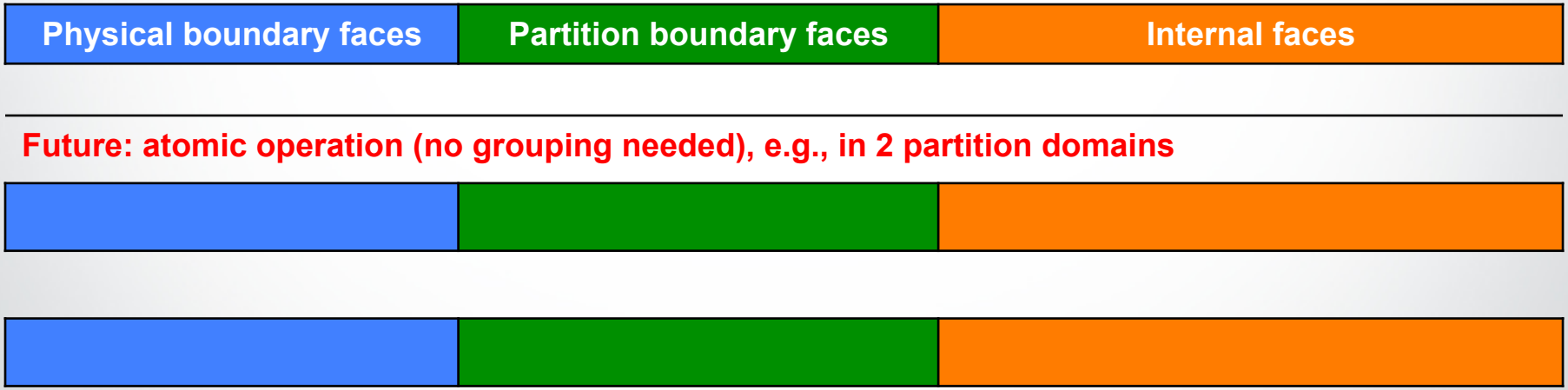
- Face grouping algorithm for balanced load over partition-domain regions



Future Work & Prospect

1. Multi-GPU parallelization via MPI (cont.)

- **Atomic operation: !$acc atomic**
  - ❑ Declared in OpenACC 2.0
  - ❑ Not yet implemented in PGI Accelerator
  - ❑ Why we need it?
    - Significant simplification of programming
    - Potential huge improvement of scalability



Future Work & Prospect

2. Implicit time integration

- **Preconditioned linear solver, e.g., GMRES-LU+SGS algorithm**
  - ❑ The inverse of the approximate block diagonal Jacobian matrix is the least we need for preconditioning
  - ❑ Any direct algorithm of matrix inverse requires some auxiliary arrays, as below

OpenMP version

```
real*8 A(20,20,Nel)
real*8 B(20,20)
real*8 C(20)

!$omp parallel
!$omp do
do ie = 1, Nel
   !... Direct algorithm to invert A(20,20,ie)
   !... with the aid of B(20,20) and C(20)
enddo
!$omp end parallel
```

OpenACC version

```
real*8 A(20,20,Nel)
real*8 B(20,20)
real*8 C(20)

!$acc parallel
!$acc loop
do ie = 1, Nel
   !... Direct algorithm to invert A(20,20,ie)
   !... with the aid of B(20,20) and C(20)
enddo
!$acc end parallel
```

However, the direct algorithm in the OpenACC ver. is practically useless, due to very limited "shared memory" (48~64KB) / thread block on GPU!

Future Work & Prospect

2. Implicit time integration (cont.)

- **Solution: iterative algorithm for matrix inverse**
  - ❑ Pros
    - No need of auxiliary arrays, suitable for OpenACC parallel regions
    - Adequate solution efficiency for diagonal-dominant matrix
  - ❑ Cons
    - To be explored…

OpenMP version

```
real*8 A(20,20,Nel)
real*8 B(20,20)
real*8 C(20)

!$omp parallel
!$omp do
do ie = 1, Nel
   !... Direct algorithm to invert A(20,20,ie)
   !... with the aid of B(20,20) and C(20)
enddo
!$omp end parallel
```

OpenACC version

```
real*8 A(20,20,Nel)

!$acc parallel
!$acc loop
do ie = 1, Nel
   !... Iterative algorithm to invert A(20,20,ie)
   !... with no need of auxiliary arrays
enddo
!$acc end parallel
```

Development Plans

Recent work

Multi-GPU parallelization via MPI

- ❑ A balanced face grouping algorithm is required over domain partitions.

Challenges for implementing implicit time integration

- ❑ Algorithms suitable for GPU parallel computing
- ❑ memory limitation on GPU devices.

Goals

A portable, GPU-accelerated high-order CFD toolkit

- ❑ A complete framework for the reconstructed discontinuous Galerkin (RDG) method for compressible flows on hybrid grids

Impact

A potential, competitive parallel-computing model of CFD programming for the next-generation HPC hardware/software