Unsteady CFD for aerodynamics profiles

Jean-Marie Le Gouez, Onera Département MFN
Jean-Matthieu Etancelin, ROMEO

Thanks to Nikolay Markovskiy dev-tech at NVIDIA research Center, GB
Unsteady CFD for aerodynamics profiles

• Context

• State of the art for LES / ZDES unsteady Fluid Dynamics simulations of aerodynamics profiles

• Prototypes for new generation flow solvers

• NextFlow GPU prototype: Development stages, data models, programming languages, co-processing tools

• Capacity of TESLA networks for LES simulations

• Performance measurements, tracks for further optimizations

• Outlook
The recognized platforms for Research and Industrial applications:

- **elsA**: aerodynamics and aeroelasticity, in direct and adjoint modes, automatic shape optimization, numerous turbulence models, Zonal Detached Eddy Simulation (ZDES)

- **Cedre**: combustion and aerothermics, 2-phase flows, heat radiation, structural thermics and thermomechanics (Zebulon Onera code)

- **Sabrina/Funk**: unsteady aerodynamics, optimized for LES and aeroacoustics

- Their associated tools for application productivity: links with CAD, overlapping grids, AMR patches, grid deformation tools

- The multi-physics couplings libraries (collaborations with Cerfacs: OpenPalm)

- **Hundreds of thousands of code lines, mix of Fortran, C++, python**

- **Important usage by the aerospace industries**
The expectations of the external users:

• Extended simulation domains: effects of wake on downstream components blade-vortex interaction on helicopters, thermal loadings by the reactor jets on composite structures,

• Model of full systems and not only the individual components: multi-stage turbomachinery internal flows, couplings between the combustion chamber and the turbine aerodynamics, …

• More multi-scale effects: representation of technological effects to improve the overall flow system efficiency: grooves in the walls, local injectors for flow/acoustics control,

• Advanced usage of CFD: adjoint modes for automatic shape optimization and grid refinement, uncertainty management, input parameters defined as pdf,
Hard to conduct a deep reengineering of the existing solvers (important workload on the improvement of physical modelling with them)

Risks on their future scalability towards PetaFlops computing (expected within 3-5 years)

Development of prototypes is undertaken on the following subjects, for more modular solvers and couplers, optimized in their field:

Grid strategies:
- hybrids structured / unstructured
- overlapping, refined automatically,
- high order grids (curved faces),

Numerical schemes and non-linear solvers:
- Discontinuous Galerkin Method (AGHORA project)
- High order Finite Volumes (NextFlow project)

Multi-physics and multi-model couplings: CWIPI library, OpenPalm with Cerfacs

HPC: MPI / OpenMP, Vectorization on CPU (X86)
Heterogeneous Hardware architectures CPU / GPU
Improvement of predictive capabilities over the last 5 years: RANS / zonal LES of the flow around a High-Lift deployed wing

2009

- 2D steady RANS and 3D LES 7.5 Mpts

2009

- Mach 0.18
- Rey 1 400 000/corde

- Optimized on a CPU architecture MPI / OpenMP / vectorization
- CPU resources for 70ms of simulation: JADE computer (CINES)
- \( N_{xyz} \sim 2600 \text{ Mpts} \quad 4096 \text{ cores} / 10688 \text{ domains} \quad T_{CPU} \sim 6200000 \text{ h} \)

- Computed field of dilatation
- Identification of acoustic sources

2014

- LEISA project DLR / Onera cooperation
- ONERA FUNK software

Journée Scientifique ROMEO 11 Juin 2015
NextFlow : HO Finite Volume for RANS / LES
Demonstration of the feasibility of porting these algorithms on heterogeneous architectures
Starting from a Fortran / MPI base, with a management of a complex data model on partitioned grids, porting both on OpenMP / Fortran and C / Cuda, sharing an interface that does a transpose of work arrays, manages pointers on CPU and GPU memories,...

Preliminary work : i,j,k structure of the grid and the arrays, choice of the programming model (shared or distributed memory, acceleration by directives or dedicated language → choice of C / Cuda / structures

1st version : on the fully unstructured grid. Tests of fine partitioning relying on advanced cache usage on the GPU,

2nd version : hierarchical model, with an initial coarse grid which is dynamically refined on the GPU : this ensures a coalescent data access to global memory,

3rd version (on-going work) : 2.5 D model, periodic in spanwise direction, automatic vector processing on the CPU, and full data parallel model for the GPU in the homogeneous 3rd dimension

« Cela est bien dit, répondit Candide, mais il faut écrire notre Cuda »
1st Approach: Block Structuration of a Regular Linear Grid

Partition the mesh into small blocks

Map the GPU scalable structure

SM: Stream Multiprocessor
Advantage of the Block Structuration

- Bigger blocks provide
  - Better occupancy
  - Less latency due to kernel launch
  - Less transfers between blocks

- Smaller blocks provide
  - Much more data caching

- Final speedup wrt. to 2 hyperthreaded Westmere CPU: ~2
NXO-GPU Phase 2: Imposing a sub-structuration to the grid and data model
(inspired by the ‘tessellation’ mechanism in surface rendering)

Unique grid connectivity for the algorithm

Optimal to organize data for *coalescent memory access* during the algorithm and communication phases

Each coarse element in a block is allocated to an inner thread (threadId.x)

Hierarchical model for the grid: high order (quartic polynomial) triangles generated by gmsh refined on the GPU

the whole fine grid as such could remain unknown to the host CPU
Code structure

**Preprocessing**
Mesh generation and block and generic refinement generation

**Solver**
- Allocation and initialization of data structure from the modified mesh file
  - Fortran
- Computational routine
  - Fortran
  - GPU allocation and initialization binders
    - C
  - Computational binders
    - C
    - CUDA kernels
    - CUDA
  - Data fetching binder
    - C
- Time stepping

**Postprocessing**
Visualization and data analysis

Journée Scientifique ROMEO 11 Juin 2015
Results on a K20C : Max. Acceleration = 38 wrt to 2 Westmere sockets

Improvement of the Westmere CPU efficiency : OpenMP task-based ➞ the blocks are refined on the CPU also, then the K20C GPU / CPU acceleration drops to 13 ( 1 K20c = 150 Westmere cores)

In fact this method is memory bounded, and GPU bandwidth is critical. More CPU optimisation needed (cache blocking, vectorisation ?)

Flop count : around 80 Gflops/K20C

These are valuable flop, not Ax=b flop, but Riemann solver flop with high order (4th, 5th) extrapolated values, characteristic splitting, … : requires a very high memory traffic to permit theses flops

Thanks to the NVIDIA dev-tech department for their support, “ my flop is rich”
Version 3 : 2.5D periodic spanwise (cshift vectors), MULTI-GPU / MPI

Objective : one Billion cells on a cluster with 64 TESLA K20 (40 000 cells * 512 spanwise stations per partition)

The CPU (Fortran) and GPU (C/ Cuda) versions are in the same executable, for efficiency and accuracy comparisons

High CPU vectorisation (all variables are vectors of length 256 to 512) in the 3rd homogeneous direction

Full data parallel Cuda kernels coalesced
Version 3 : 2.5D periodic spanwise (cshift vectors), MULTI-GPU / MPI

PERFORMANCE LIMITERS

Three top kernels

iorfo=1,2

iorfo=3
Version 3 Initial Kernel Optimization (done by NVIDIA DevTech Great-Britain)

RESULTS

Comparison with Original code (1 partition)

Performance strategy: Increase occupancy, reduce registers’ use, reduce amount of operations with global memory

After some GPU optimizations: Acceleration 14 on one K20c with respect to 8 IVB cores, GPU memory Bandwith 150 GB/s

For LES, GPU memory size not too much of a problem: 20 million cells stored on a K20 (6 Gbytes), 40 millions on a K40

CPU time
Left column: IvyBridge (8 OMP threads)
Right column: K40
Full time steps and 7 main kernels
Version 3 : on-going work

Improve the efficiency of the communications between partitions: MPI/Cuda, GPUDirect,
Overlap communications with computations at the center of the partitions,
Verify the scalability curve from 1 to 128 accelerators,
Choose a strategy for the CPU usage:
  - identify algorithmic phases that are less pertinent for GPU,
  - specialize the CPU for co-processing tasks: Spatial Fourier transforms, management and computation of mesh refinement indices, preparation of cell metrics for the next time step for moving/deforming grids, computation of iso-value surfaces

Compare different versions of the Fluxes and Riemann solvers, which require less registers,
Compare the efficiency and accuracy to standard Fortran / MPI 2nd order codes, while using larger mesh size

Prepare for the specification of the next generation of CFD solvers