Generic Refinement and Block Partitioning enabling efficient GPU CFD on Unstructured Grids

Matthieu Lefebvre¹, Jean-Marie Le Gouez²

¹PhD at Onera, now post-doc at Princeton, department of Geosciences, mll5@princeton.edu
²Director of CFD and Aeroacoustics Department, Onera, Châtillon
Efficient GPU CFD on Unstructured Grids

- Context
- Objectives
- Development stages, data models, languages
- Results of CFD method implementations
- Outlook
Context: Simulation & HPC

CFD legacy softwares

- elsA: aerodynamics and aeroelasticity,
- Cedre: combustion and aerothermics,
- Sabrina/Space/Funk: LES, Aeroacoustics, Euler equations in perturbations, linear or not
- Hundreds of thousands lines, mixed Fortran, C++, Python for elsA
- High usage in aerospace industry

External user expectations: need solutions to process
- Larger flow domains: effect of wakes on downstream components: BVI,
- Full system modeling like multistage aerodynamics in turbomachinery, coupling with the combustion
- More multiscale: near-wall modeling, technological details to gain in system efficiency, flow control,
- Up-to-date CFD usage: Shape optimization, building and use of response surfaces, robust design, i.e. effect of existing uncertainties on input data: on-going integration in mainstream codes
Context: Simulation & HPC

Internal users expectations, as front end to the external users:
New class of methods to develop better models of flow behavior
- confidence that novel numerical schemes will better preserve their models from spoiling by numerical diffusion/dispersion (still not a consensus),
- more efficiency: accuracy versus number of cells, grid convergence
- Objective: Billion cells, long unsteady runs, deforming & overlapping meshes (LES, DES, Aeracoustics)

Evidences:
1. Ongoing heavy development plans of the legacy software to develop & validate new functionalities ➔ risks on the scalability towards Petaflop’s computing
2. Innovative research at ONERA: numerous prototypes
   - Grid strategies:
     - structured
     - unstructured
     - overset, self generated, adaptive
   - associated to model certification, international benchmarks
   - Numerical schemes and non-linear solvers: DGM (h-p-M adaptation), VMS turbulence (Variational multiscale with hierarchy in basis functions), High Order FV
   - Mesh/model/physics coupling strategies
Objectives: Simulation & HPC

Proceed with innovative solver prototypes, h,p-Model adaptive : DGM, Hybrid&Overset FV ?, Isogeometric FE with high order curved boundary elements

Analyse the hierarchy in hardware computing core implementations, coprocessor farms, the associated hierarchy in memory access,

Specify modular solvers on partitioned grids, adaptive and hierarchical data models, reuse (rewrite, simplify) existing modules code lines (already subdomain&MPI)

Follow the way for a CFD Domain Specific Language
Generic GPU Finite Volume prototype

Uses a High Order in space FV method (NXO)

From a Fortran basis, which uses complex data structures on partitioned grids, calls the coprocessor modules (C subroutines, CUDA kernels), mocks-up and arranges the data model in C and Cuda, manages the pointers on GPU memory and thread submission for high GPU efficiency on TESLA

1st (easy) stage : on i,j,k structured grids multi-GPU implementation
2nd stage : on an existing arbitrary mesh of triangles

2a : domain partitioning of fine linear triangles (gmsh) addressed to blocks of threads : efficient usage of cache
2b : partitioning and generic refinement of an unstructured coarse mesh of triangles (high geometrical order triangles by gmsh) : efficient coalescent memory accesses in all the solver inner stages (kernel computations element-based, face-based, node-based)
Euler equations with immersed boundaries i,j,k grid

2nd order schemes
Roe Fluxes / MUSCL
**Cartesian structuration**

Used to determine language / API / tools to use
CUDA C/Fortran, OpenCL
PGI Accelerator

2D and 3D solvers
Cartesian Euler: Multi-GPUs Communication Scheme

CUDA C model preferred
2 Fermi GPUs
  collaboration through the pthread library

Flat plate
25 M points
Mach 0.7
20 degree incidence
3D Experiments – Speedup and Scalability Results

Speedup 1 M2050 vs. 2 M2050: up to 2.11
Challenge: Accelerating Non “GPU Compliant” Codes

- Unstructured grids
- Large stencils (up to the third neighbor to compute the fluxes for each cell interface, very high memory stress
- Reaches 4th-order spatial accuracy
- However algorithmic efficiency is hampered by the numerous indirect memory accesses through the arbitrary connectivity lists accessed at successive stages of the algorithm (cell-, face-, node-, stencil-based descriptions).

On a GPU such non consecutive memory fetches are penalizing.

NXO
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 an Inner Structuration to the Grid
(inspired by the “tessellation” mechanism in surface rendering)

Hints for further optimization:
• relax the grid coordinates once the substructuration is done
• the whole fine grid as such could remain unknown to the host CPU

‘red’ block partition
NXO-GPU Phase 3: Imposing an Inner Structuration to the Grid

Unique grid connectivity for the algorithm
Unique subsets of cell groups for communications
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)

To improve the percentage of coalescent communication:
- Renumber the coarse cells in a partition
- Change the orientation (permutation of the sides)

Sub-structured interface vector for communication
Code structure

Preprocessing
Mesh generation and block and generic refinement generation

Solver
Allocation and initialization of data structure from the modified mesh file
Computational routine
GPU allocation and initialization binders
Computational binders
CUDA kernels
Data fetching binder

Postprocessing
Visualization and data analysis
Stencils and Ghost cells

Coupling mechanisms:

- Identify ghost cells with real fine cells from neighbor coarse cells (transfer its metrics)

- or:

  - do an overset grid interpolation to obtain the volume average of the conserved variables on the extruded ghost cells

Reference element to be mapped on curvilinear cells
Code structure

CPU Data structure

Block List

---

1

Block

- Conservative variables
- Stencils coefficients

---

Coarse Element

- Generic geometry arrays
- Connectivity between fine elements

Array filling: face

Fine element number

Coarse element

1..*
Code structure

Block List + ... 1..* Block + ...

Block List + ... 1..* Block + ...

Block "C pointer" List + ...

Coarse Element + ...

Fortran 2003 "C pointer" localisation

Fortran 2003 interface

Fortran

C / CUDA
```c
typedef struct {
    int number_elements;   
    int number_faces;      
    int size_stencil;      
    double *wall_distance; 
    double *dissipation_rate;
    // ...                   
    int number_adherent_faces; 
    int number_open_faces;  
    int *adherent_faces;    
    int *open_faces;        
} block_data_t;
```

```c
TYPE block_data_t
    integer(C_INT) :: number_elements
    integer(C_INT) :: number_faces
    integer(C_INT) :: size_stencil
    real(C_DOUBLE), dimension(:), allocatable :: wall_distance
    real(C_DOUBLE), dimension(:), allocatable :: dissipation_rate
    // ...                     
    real(C_DOUBLE), dimension(:), allocatable :: fluid_u
    real(C_DOUBLE), dimension(:), allocatable :: fluid_v
    // ...                     
    real(C_DOUBLE), dimension(:), allocatable :: producted_kinetic_energy
    real(C_DOUBLE), dimension(:), allocatable :: pseudo_timestep
    real(C_DOUBLE), dimension(:), allocatable :: ro
    real(C_DOUBLE), dimension(:), allocatable :: sfx
    real(C_DOUBLE), dimension(:), allocatable :: sfy
    real(C_DOUBLE), dimension(:), allocatable :: temperature
    real(C_DOUBLE), dimension(:), allocatable :: total_energy
    real(C_DOUBLE), dimension(:), allocatable :: viscosity
    // ...                     
    integer(C_INT), dimension(:), allocatable :: face_vertices
    integer(C_INT), dimension(:), allocatable :: index_element_faces
    integer(C_INT), dimension(:), allocatable :: index_face_elements
    integer(C_INT), dimension(:), allocatable :: number_elements_stencils
    integer(C_INT), dimension(:), allocatable :: stencil_indexes
    integer(C_INT), dimension(:), allocatable :: index_stencil_faces
    // inter coarse element communications
    integer(C_INT), dimension(:), allocatable :: message_src

    integer(C_INT) :: number_adherent_faces
    integer(C_INT) :: number_open_faces
    integer(C_INT), dimension(:), allocatable :: adherent_faces
    integer(C_INT), dimension(:), allocatable :: open_faces
END TYPE block_data_t
```
Measured efficiency on a Tesla 2050 (with respect to 2 Cpu Xeon 5650, OMP loop-based)

Recent results on a K20C : Max. Acceleration = 38 wrt to 2 Westmere sockets, very good scaling from the C2050 with the number of cores (2000 / 480), due to a lower pressure on registers ?
Conclusion

Generic refinement methodology permits coalesced data accesses in global memory
Exchange step between coarse cells ~ 10% global time

Refinement level adaptable to the target computer architecture
Refinement level adaptable to the solution (clustering in space and blocks of threads of coarse cells with identical refinement)

Natural extension of the partitioned structure to multi-GPUs,

Further works: extension to 3D and other types of elements

A prototype of hierarchical data model, further methods (DGM, 2nd order compact schemes on unstructured grids) and turbulence models will be ported in new kernels.

At present non linear implicit in real time by dual time stepping, a linearized implicit formulation will be implemented