Some examples for CFD instant computations on GPU

F. De Vuyst

Centre de Mathématiques et leurs Applications
CMLA, CNRS UMR 8536
ENS Cachan
« I have a dream » ...

- Get instant computations

- Interact with the flow

- Perform interactive CAD/parameter design and see the impact on the objective function

Q : is it a pure dream ? Realistic in (say) 20 years ? What kind of computer ? Office workstation ? ...
Outline

1. Lattice Boltzmann method for 2D unsteady Navier-Stokes equations
2. Finite Volume method for 2D compressible Euler equations
3. Lagrange-remap schemes
4. Experience feedback
5. Demo
nVIDIA GPU FERMI compute architecture

Scheduler & dispatch

Streaming multiprocessor (SM)  Registers & L1 cache

(2*16)*16 = 512 cores
Typical NVIDIA GPU in use

NVIDIA GeForce GTX 690

- 3072 cores !!
- Base Clock 915 MHz
- Boost Clock 1019 Mhz
- Memory bandwidth 6.0 Gbps
- Mem 4096 MB (2048 MB per GPU)
- DRAM bus memory 512-bit GDDR5

Theoretical peak performance : about 6 TFlops !! (single precision)
1. Lattice Boltzmann method for 2D unsteady Navier-Stokes equations

- A mesoscopic paradigm to model PDE problems
- A « smoothed » Lattice Gas automaton model
- Navier-Stokes case: based on a very simple approximation and discretization of the Boltzmann-type equation
Lattice BGK model

Based on a simple discretization of the Boltzmann equation with BGK approximation for the collision term:

\[
f(x + e_i \Delta t, e_i, t + \Delta t) - f(x, e_i, t) = \frac{1}{\tau} \left[ f_{eq}^\rho (\rho, \mathbf{u}) - f(x, e_i, t) \right]
\]

\[
\Delta x = \Delta t = 1
\]

Relaxation toward some equilibrium...

\[e_0 = 0, \ e_1 = (1, 0), \ e_2 = (1, 1), \ldots\]

Moments:

\[
\sum_{i \in \mathcal{S}} f(x, e_i, t) = \rho(x, t),
\]

\[
\sum_{i \in \mathcal{S}} e_i f(x, e_i, t) = \rho \mathbf{u}(x, t).
\]
Lattice BGK model

- Unknows \( f_i(x, t) \equiv f(x, e_i, t), \; i \in \{0, \ldots, 8\} \).

- LB equation

  \[
  f_i(x + e_i, t + \Delta t) = f_i(x, t) + \omega \left[ f_i^{eq}(\rho(x, t), u(x, t)) - f_i(x, t) \right]
  \]

- Requirements

  \[
  \sum_{i \in S} f_i^{eq}(\rho, u) = \rho, \quad \omega = \frac{1}{\tau}.
  \]

  + some galilean invariance conditions

  \[
  \sum e_i f_i^{eq}(\rho, u) = \rho u.
  \]
2D and 3D lattices & equilibrium distribution

\[ f_i^{eq} = \rho w_i \left[ 1 + 3u \cdot e_i + \frac{9}{2}(u \cdot e_i)^2 - \frac{3}{2}u \cdot u \right] \]

For D2Q9 we have:

\[ w_i = \begin{cases} 
4/9 & i = 0 \\
1/9 & i = 1, 2, 3, 4 \\
1/36 & i = 5, 6, 7, 8 
\end{cases} \]

For D3Q15 the weights are:

\[ w_i = \begin{cases} 
2/9 & i = 0 \\
1/9 & i = 1 - 6 \\
1/72 & i = 7 - 14 
\end{cases} \]

For D3Q19 the weights are:

\[ w_i = \begin{cases} 
1/3 & i = 0 \\
1/18 & i = 1 - 6 \\
1/36 & i = 7 - 18 
\end{cases} \]

Source: [A. Wagner 08]
Practical implementation « stream-and-collide »

1. Stream step

\[ \tilde{f}_i(x, t) = f_i(x + e_i, t) \]

2. Collision step

- Compute the moments

\[ (\rho, \rho u)(x, t) = \sum_{i \in S} (1, e_i) \tilde{f}_i(x, t) \]

- Compute the equilibrium function

\[ \tilde{f}_i^{eq} = \rho w_i \ldots \]

- Collision step:

\[ f_i(x, t + \Delta t) = \tilde{f}_i(x, t) + \omega \left[ \tilde{f}_i^{eq}(\rho(x, t), u(x, t)) - \tilde{f}_i(x, t) \right] \]

« ET VOILA ! »
Connection with Navier-Stokes equations (multiscale Chapman-Enskog analysis)

\[ \nabla \cdot u = 0 + O(\Delta t^2), \]

\[ \frac{\partial u}{\partial t} + u \cdot \nabla u + \frac{1}{\rho} \nabla p - \nu \Delta u = O(\Delta t^2 + \Delta t M^2). \]

with \[ \nu = \frac{1}{3} (2\tau - 1). \]

Requires: \[ u = M \ll 1. \]

Artificial EOS: \[ p = p(\rho) = \rho c^2. \]

Von Neuman linear stability analysis of LBGK scheme:

\[ \tau > \frac{1}{2} \iff \omega < 2. \]
Benefits

- Very simple to implement (explicit schemes)
- $O(100\text{-lines})$ Navier-Stokes code with Matlab!
- Wall and fluid boundary conditions OK
- Complex geometries by immersed boundary method (IBM)
- Appears to be suitable for GPU parallel computing (stream-and-collide procedure)

[Qian, d'Humières, Lallemand 92], [Ginzburg, d'Humières, Krafczyk, Lallemand, 2002], [Chikatamarla, Ansumali, Karlin 06] [Tosi, Ubertini, Succi 06], [Brownlee, Gorban, Levesley 06], [Xu, Luan, Tang, Tao 09] [Dellacherie 13]

Finite Volume interpretation [Dubois, Lallemand 08]
2. Finite Volume method for 2D compressible Euler equations

\[
\begin{align*}
\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) &= 0, \\
\partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p &= 0, \\
\partial_t (\rho E) + \nabla \cdot ((\rho E + p) \mathbf{u}) &= 0.
\end{align*}
\]

\[
E = e + \frac{1}{2} |\mathbf{u}|^2, \quad e = \frac{1}{\gamma - 1} \frac{p}{\rho}, \quad \gamma \in (1, 3].
\]

- Nonlinear hyperbolic conservative system
- Solutions generally develop discontinuities
- Numerical framework : (stable) finite volumes
Flux Difference Splitting (FDS) vs Flux Vector Splitting (FVS) methods on GPU

FDS case

i) **Send** states at interfaces

ii) Computes numerical fluxes

\[
\Phi(U^n_K, U^n_L, \nu_{KL}) = \frac{F(U^n_K) + F(U^n_L)}{2} - \frac{1}{2} \int_{\Gamma_{KL}} |A|(s) U'(s) \, ds
\]

iii) **Send** fluxes at cell centers

iv) **Update** states
Flux Difference Splitting (FDS) vs Flux Vector Splitting (FVS) methods on GPU

FVS case

\[ \Phi(U^n_K, U^n_L, \nu_{KL}) = F^+(U^n_K, \nu_{KL}) + F^-(U^n_L, \nu_{KL}) \]

i) Compute FVS at cell centers

ii) Send \( F^+ \) and \( F^- \) to the neighboring cells

iii) Update states

\[ \rightarrow \text{FVS appears to reduce DRAM communications by 2.} \]

Exemple of Flux Vector Splitting

Van Leer's Flux Vector Splitting with Hänel's energy flux correction:

\[
F^+_{\rho} = \frac{\rho c}{4} (M + 1)^2 \mathbb{1}_{|M| \leq 1} + \max(u, 0) \mathbb{1}_{|M| > 1}
\]

\[
p^+ = \frac{p}{4} (M + 1)^2 (2 - M) \mathbb{1}_{|M| \leq 1} + p \mathbb{1}_{M > 1}
\]

\[
F^+_{\rho u} = u F^+_{\rho} + p^+, \quad F^+_{\rho v} = v F^+_{\rho}, \quad F^+_{\rho E} = H F^+_{\rho}, \quad H = E + p/\rho.
\]

\[
F^-(U) = F(U) - F^+(U).
\]

Quite simple to compute
GPU CUDA implementation

C CUDA kernel prototypes (example)

```c
//
// CUDA kernel prototypes
//
__global__ void init_kernel (int pitch, var *d_vars);
__global__ void setPlotData_kernel(int pitch, var *d_vars, float *plot_data);
__global__ void computePrimitiveVariables_kernel(int pitch, var *d_vars, int solid_data);
__global__ void computeFVSx_kernel(int pitch, var *d_vars, flux *fpp, flux *fmm, int solid_data);
__global__ void computeFVSy_kernel(int pitch, var *d_vars, flux *gpp, flux *gmm, int solid_data);
__global__ void updateScheme_kernel(float hx, float hy, float dt,
int pitch, var *d_vars, flux *fpp, flux *fmm, flux *gpp, flux *gmm, int solid_data);
```

Time advance

```
for (int i=0; i<FREQ; i++) {
//
computePrimitiveVariables_kernel<<<grid, block>>>(pitch, d_vars, solid_data);
// Compute FVS in x direction
computeFVSx_kernel<<<grid, block>>>(pitch, d_vars, fpp, fmm, solid_data);
// Compute FVS in y direction
computeFVSy_kernel<<<grid, block>>>(pitch, d_vars, gpp, gmm, solid_data);
// Apply the Finite Volume scheme
updateScheme_kernel<<<grid, block>>>(hx, hy, dt, pitch, d_vars, fpp, fmm, gpp, gmm, solid_data);
}
```

Very easy!
Ex3 – Remapped-Lagrange Eulerian solvers
Lagrange-remap (LR) schemes

- Define a piecewise $C^1$ Lagrange transformation operator $\mathcal{L}(x, t; t_0, x_0) : dx/dt = u(x), \ x(t = 0) = x_0$.
- Use the Reynolds transport theorem:

$$\frac{d}{dt} \int_{\Omega_t} q(x, t) \, dx = \int_{\Omega_t} \{\partial_t q + \nabla \cdot (qu)\} \, dx$$

for any moving domain $\Omega_t$, quantity $q$.
- Finite volume approximation
- Move the mesh according to a Lagrange operator $\mathcal{L}$ over $[t^n, t^{n+1}]$
- Project the quantities on the initial mesh.
Conservative reformulation

Geometric conservation law (GCL):

\[ h_j^{n+1,L} = h + \Delta t \left( u_{j+1/2}^{n+1/2,L} - u_{j-1/2}^{n+1/2,L} \right). \]

Mass conservation law:

\[ \rho_j^{n+1,L} h_j^{n+1,L} = h \rho_j^n. \]
Conservative reformulation (2)

\[ \rho_{j}^{n+1,L} = \frac{\rho_{j}^{n}}{1 + \frac{\Delta t}{h} (\Delta u)_{j}^{n+1/2,L}}, \quad (\Delta u)_{j}^{n+1/2,L} := u_{j+1/2}^{n+1/2,L} - u_{j-1/2}^{n+1/2,L}. \]

Projection step:

\[ \rho_{j}^{n+1} = \frac{1}{h} \int_{l_{j}} \mathcal{I} \rho^{n+1,L}(x) \, dx = \frac{1}{h} \int_{l_{j}^{n+1,L}} \ldots - \ldots + \ldots. \]
Conservative reformulation (3)

Mass balance rewriting: under some convenient CFL condition, we have

\[ h\rho_j^{n+1} = h_j^{n+1,L} \rho_j^{n+1,L} - \Delta t \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L} + \Delta t \rho_{j+1/2}^{upw,n+1} u_{j-1/2}^{n+1/2,L} \]

\[ = h\rho_j^n - \Delta t \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L} + \Delta t \rho_{j+1/2}^{upw,n+1} u_{j-1/2}^{n+1/2,L} \]

in the form

\[ \rho_j^{n+1} = \rho_j^n - \frac{\Delta t}{h} \left( \Phi_{m,j+1/2}^{n+1/2,n+1} - \Phi_{m,j-1/2}^{n+1/2,n+1} \right), \]

\[ \Phi_{m,j+1/2}^{n+1/2,n+1} = \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L}. \]

[De Vuyst, Fochesato, Loubère, Saas, Motte, Ghidaglia, preprint paper 2013]
Remark

First-order (1st-order remap) LR schemes are actually 5-point schemes!

\[ U_{j+1/2}^{n+1} = \Phi_{j+1/2} \left( U_{j-1}^n, U_j^n, U_{j+1}^n, U_{j+2}^n, \Delta t \right). \]

→ Large stencil method: limited GPU performance because of lot of memory reads.
2D case
Recent results

The Lagrange-schemes have been modified in order to lead to reduced spatial stencils (paper in progress)

→ Expected to obtain far better performance on GPU.
Data structures: technical aspects
Coalesced memory is a critical keypoint

Q: Arrays-of-Structs (AoS) or Structs-of-Arrays (SoA)?

\[ U = (\rho, \rho u, \rho v, \rho E)^T \]

\[ U[i2d(i,j)].\rho = \ldots \quad \text{// AoS} \]
\[ \quad \text{// Coalesced memory for struct} \]

\[ U.\rho[i2d(i,j)] = \ldots \quad \text{// SoA} \]
\[ \quad \text{// Coalesced memory for array} \]

Option: AoSoA, forced alignment, texture memory, other?
4. Experience feedback for the PDE community

- Reasonable (suboptimal) speedup (say 10 or 20) is easy to reach

- GPU Parallel computing easy for pbs on cartesian grids

- Better performance: tradeoff to find between code readability and performance.

- Chose suitable data structures, not too complex for memory coalescence, read/write transaction: AoS vs SoA vs SoAoS, byte alignment …

- Instant interactive computations made possible, today in 2D, certainly in 3D in 2-3 years…
5. Ongoing/future works

- 2D air-water 2-fluid flow with interface capturing on GPU
- LB for fluid & moving rigid bodies (gas-solid flows)
- Thermal-CFD coupling (Boussinesq) using LBM
- Exploration of new suitable paradigm shifts
- Splitting operator strategies (tuning computations and communications, important keyword)
Papers


Acknowledgements

- Collaborators from LMT: C. Rey, R. Bennacer, M. Cremonesi
- CMLA GPU team: C. Labourdette, F. Pascal, L. Quivy, N. Vayatis
- Ext. collaborators: F. Salvarani (U. Pavia), S. Faure, L. Gouarin, B. Graille (LMO Orsay)

Institut Farman Grant, NVIDIA Grant, LRC MESO, LMH Labex
Thank you for your attention