Towards an explicit, second-order, semi-Lagrangian 3D solver for Navier–Stokes equations on low-cost GPU architectures

Roberto Ferretti
Department of Mathematics and Physics, Roma Tre University
ferretti@mat.uniroma3.it
Catania, 21 Feb 2023

joint works (partly in progress) with
L. Bonaventura, S. Cacace, E. Carlini, E. Calzola, L. Rocchi
Outline

1 Introduction
   - Some basic concepts in SL schemes
   - The general idea in treating diffusion operators

2 2D Navier–Stokes Equations
   - Velocity–Pressure formulation
   - Boundary conditions for velocity

3 Numerical examples, projection scheme
   - Lid-driven cavity
   - Von Karman vortex street
   - Evaluation of the scheme

4 Second-order advection–diffusion solver

5 High-order treatment of boundary conditions

6 Efficient point location on unstructured grids
   - Quadtree/octree algorithms
   - Walking algorithms

7 Speed-up strategies for BW
   - Following characteristics
   - Using the previous time step
   - Using a neighbouring node
   - Evaluation of the various strategies

8 CUDA parallelization
Basic concepts on SL schemes – hyperbolic case

**Model equation:** linear, constant-coefficient advection

\[
\begin{align*}
    u_t(x, t) + au_x(x, t) &= 0 \\
    u(x, 0) &= u_0(x)
\end{align*}
\]

**Representation formula**

\[ u(x, t) = u_0(x - at) \]

**Time discretization**

\[ u(x_j, t_{n+1}) = u(x_j - a\Delta t, t_n) \]

Semi-Lagrangian (SL) schemes stem from the so-called **Courant–Isaacson–Rees (CIR) method** ('52) which discretizes the representation formula (instead of the equation)
Semi-Lagrangian (SL) discretization

\[ v_j^{n+1} = I[V^n](x_j - a\Delta t) \]

The most classical choice for the interpolation \( I[V] \) is a symmetric Lagrange interpolation on a structured uniform mesh. Various other options are possible, among which Galerkin projection.
Extension to second order operators (1)

In the constant coefficient, advection–diffusion case we have, via the Taylor expansion:

\[
\begin{align*}
  u(x_j + a\Delta t + \sqrt{2\nu\Delta t}) & = u(x_j) + \Delta t \, au_x(x_j) + \sqrt{2\nu \Delta t} \, u_x(x_j) + \\
  & + \Delta t \, \nu u_{xx}(x_j) + O(\Delta t^{3/2}) + O(\Delta t^2)
\end{align*}
\]

\[
\begin{align*}
  u(x_j + a\Delta t - \sqrt{2\nu\Delta t}) & = u(x_j) + \Delta t \, au_x(x_j) - \sqrt{2\nu \Delta t} \, u_x(x_j) + \\
  & + \Delta t \, \nu u_{xx}(x_j) - O(\Delta t^{3/2}) + O(\Delta t^2)
\end{align*}
\]
Extension to second order operators (1)

In the constant coefficient, **advection–diffusion case** we have, via the Taylor expansion:

\[
\begin{align*}
\frac{1}{2} \left[ u(x_j + a\Delta t + \sqrt{2\nu\Delta t}) + u(x_j + a\Delta t - \sqrt{2\nu\Delta t}) \right] &= u(x_j) + \Delta t \left[ au_x(x_j) + \nu u_{xx}(x_j) \right] + O(\Delta t^2) \\
\end{align*}
\]

First proposed for Dynamic Programming equations [KD01, CF95]
A first **upwinding** of magnitude $a \Delta t$ follows the **advection** (if advection occurs)
A first **upwinding** of magnitude $a \Delta t$ follows the **advection** (if advection occurs)

A second **symmetric displacement** of magnitude $\sqrt{2\nu\Delta t}$ is related to the **diffusion** (possibly asymmetric when close to the boundary)
Resolution of the smaller scales (1)

The numerical domain of dependence is made of two regions of reconstruction which are $2\sqrt{2\nu\Delta t}$ apart. This “hole” in the stencil may cause the smaller scales to be underresolved [F10].
Resolution of the smaller scales (1)

The numerical domain of dependence is made of **two regions of reconstruction** which are $2\sqrt{2\nu \Delta t}$ apart. This “hole” in the stencil may cause the smaller scales to be underresolved [F10].

Heat equation, $\Delta x = 0.02$, discontinuous initial condition

- $\Delta t = 0.1$
- $\Delta t = 0.01$
- $\Delta t = 0.001$
Resolution of the smaller scales (2)

- **Avoiding any such "hole"** in the numerical domain of dependence would require the **parabolic CFL condition** $\Delta t \sim \Delta x^2$

- Asymptotically, "holes" are filled at a given time $T$ under the weaker condition:

$$\Delta t = o \left( T^{2/3} \Delta x^{2/3} \right)$$

in particular, **hyperbolic type** $\Delta t / \Delta x$ relationships are acceptable

- **Adaptive strategies** of choice for the time step are possible
The first **upwinding** is driven by the advective term (higher order implementations are possible)
The first **upwinding** is driven by the advective term (higher order implementations are possible)

The **symmetric displacement** is scaled and replicated on each direction: \( \delta_k = \{(\pm \sqrt{4\nu \Delta t}, 0), (0, \pm \sqrt{4\nu \Delta t})\} \)
General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)

General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)
- **Explicit** (although first-order) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)
General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)

- **Explicit** (although **first-order**) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)

- **High-order** implementations for the hyperbolic part relatively easy to construct
General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)
- **Explicit** (although **first-order**) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)
- **High-order** implementations for the hyperbolic part relatively easy to construct
- **Large time steps** allowed with weak requirements on the $\Delta t/\Delta x$ relationship
Navier–Stokes Equations

**Goal**: integrating the SL advection–diffusion solver in a code for the incompressible NSE

\[
\begin{align*}
    u_t + (u \cdot \nabla)u - \nu \Delta u + \nabla p &= 0 \\
    \nabla \cdot u &= 0
\end{align*}
\]
Navier–Stokes Equations

**Goal:** integrating the SL advection–diffusion solver in a code for the incompressible NSE

\[
\begin{aligned}
    u_t + (u \cdot \nabla)u - \nu \Delta u + \nabla p &= 0 \\
    \nabla \cdot u &= 0
\end{aligned}
\]

This is typically achieved via intermediate formulations of the NSE:

**Vorticity–Streamfunction formulation**
- Suitable only for the 2D case
- Difficult and inaccurate treatment of boundary conditions for \( \omega \)
Navier–Stokes Equations

**Goal**: integrating the SL advection–diffusion solver in a code for the incompressible NSE

\[
\begin{aligned}
\begin{cases}
    u_t + (u \cdot \nabla)u - \nu \Delta u + \nabla p = 0 \\
    \nabla \cdot u = 0
\end{cases}
\end{aligned}
\]

This is typically achieved via intermediate formulations of the NSE:

**Vorticity–Streamfunction formulation**
- Suitable only for the 2D case
- Difficult and inaccurate treatment of boundary conditions for \( \omega \)

**Velocity–Pressure (projection) formulation**
- Suitable for both the 2D and the 3D case
- Easier treatment of boundary conditions for \( u \)
Navier–Stokes Equations: projection formulation

**General idea:** linearizing the NSE by neglecting the incompressibility constraint, then correcting the solution via a suitable pressure term

\[
\begin{align*}
\frac{u^{n+1/2} - u^n}{\Delta t} &= -(u^n \cdot \nabla)u^n + \nu \Delta u^n \\
-\Delta p^{n+1} &= -\frac{1}{\Delta t} \nabla \cdot u^{n+1/2} \\
u^{n+1} &= u^{n+1/2} - \Delta t \nabla p^{n+1}
\end{align*}
\]

with proper BCs (typically, Dirichlet in the first step and homogeneous Neumann in the second for no-slip boundary)
Navier–Stokes Equations: projection formulation

**General idea**: linearizing the NSE by neglecting the incompressibility constraint, then correcting the solution via a suitable pressure term

\[
\begin{align*}
\frac{u^{n+1/2} - u^n}{\Delta t} &= -(u^n \cdot \nabla)u^n + \nu \Delta u^n \\
-\Delta p^{n+1} &= -\frac{1}{\Delta t} \nabla \cdot u^{n+1/2} \\
u^{n+1} &= u^{n+1/2} - \Delta t \nabla p^{n+1}
\end{align*}
\]

with proper BCs (typically, Dirichlet in the first step and homogeneous Neumann in the second for no-slip boundary)

- **Critical operations**: advection–diffusion and Poisson solvers
- Time advancing is performed in the form of **fractional steps**
- Applicable to **3D problems**, easier treatment of BCs for \( u \)
- Easier treatment of **unstructured geometries**
- Avoids systematic errors for the numerical divergence of \( u \)
Navier–Stokes Equations: SL projection scheme

**SL scheme**

\[
\begin{align*}
\mathbf{u}^{n+1/2}_j &= \frac{1}{4} \sum_{k=1}^{4} I[\mathbf{U}^n](x_j - \Delta t \mathbf{u}^n_j + \delta_k) \\
- \Delta p^{n+1} &= - \frac{1}{\Delta t} \text{div}_\Delta \mathbf{u}^{n+1/2} \\
\mathbf{u}^{n+1} &= \mathbf{u}^{n+1/2} - \Delta t \nabla p^{n+1}
\end{align*}
\]

where \( \delta_k = \{(\pm \sqrt{4\nu\Delta t}, 0), (0, \pm \sqrt{4\nu\Delta t})\} \) (in 2D)

A **first order** implementation can be easily constructed via built-in Matlab functions:

- **Monotonized cubic** interpolation for \( I[\mathbf{U}^n] \) (structured case)
- **Triangle-based interpolation** from PDEtoolbox (unstructured case)
- **Sparse LU/Cholesky** solver for the Poisson eqn
- **Centered FD** (or FE discrete divergence) for \( \text{div}_\Delta \)
It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with reduced consistency rate) when close to the boundary.
Boundary conditions for velocity/vorticity (1)

- It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with reduced consistency rate) when close to the boundary.
- The **local drop in consistency** (to order $1/2$) occurs in a band of width $O(\sqrt{\Delta t})$, so that it **does not affect the global order**.

\[
\begin{align*}
\alpha^- + \delta^- &= \frac{1}{2} \\
\alpha^- + \delta^* - \alpha^- - \delta^- &= 0 \\
\alpha^- (\delta^- + 1)^2 + \alpha^- (\delta^* - 1)^2 &= 2 \nu \Delta t.
\end{align*}
\]
Boundary conditions for velocity/vorticity (1)

- It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with **reduced consistency rate**) when close to the boundary.

- The **local drop in consistency** (to order 1/2) occurs in a band of width $O(\sqrt{\Delta t})$, so that it **does not affect the global order**.

- Consistency conditions:

\[
\begin{align*}
\alpha^+ + \alpha^- &= 1/2 \\
\alpha^+ \delta^+ - \alpha^- \delta^- &= 0 \\
\alpha^+(\delta^+)^2 + \alpha^- (\delta^-)^2 &= 2\nu \Delta t.
\end{align*}
\]
Boundary conditions for velocity/vorticity (2)

Using these further degrees of freedom on weights and displacements and the consistency conditions, we can then enforce

**Dirichlet BCs via modified weights/displacements**

\[
\begin{align*}
\delta^- &= \delta_M, \\
\delta^+ &= \frac{4 \Delta t \nu}{\delta_M}, \\
\alpha^- &= \frac{1}{2} \frac{\delta^+}{\delta^+ + \delta^-}, \\
\alpha^+ &= \frac{1}{2} - \alpha^-.
\end{align*}
\]
Lid-driven cavity for the NSE, projection scheme (1)

Lid-driven cavity at $Re = 1000$
200 $\times$ 200 nodes, Courant number $\approx 5$
Lid-driven cavity for the NSE, projection scheme (2)

Lid-driven cavity at $Re = 10000$
$200 \times 200$ nodes, Courant number $\approx 5$
Von Karman vortex street, projection scheme

Von Karman vortex street at $Re = 100$, grid and solution
Evaluation of the scheme

Advantages

- Possibility of *(relatively)* large Courant numbers
- **Reduced complexity** for the advection–diffusion solver
- Accurate *transition between laminar and turbulent regime*
Evaluation of the scheme

**Advantages**
- Possibility of *(relatively)* large Courant numbers
- **Reduced complexity** for the advection–diffusion solver
- Accurate *transition between laminar and turbulent regime*

**Bottlenecks**
- Relatively *inaccurate treatment of BCs*
- Difficult to treat efficiently a *complicate geometry*
## Evaluation of the scheme

### Advantages
- Possibility of (relatively) large Courant numbers
- **Reduced complexity** for the advection–diffusion solver
- Accurate **transition between laminar and turbulent regime**

### Bottlenecks
- Relatively **inaccurate treatment of BCs**
- Difficult to treat efficiently a **complicate geometry**

### Improvements
- A **second-order** advection–diffusion solver
- **Higher order treatment of BCs**
- Efficient **point location strategies** for unstructured geometries
- **Scalability** on GPU architectures → C++ implementation
A possible second-order improvement of the previous AD solver is borrowed from the literature on approximation schemes for Stochastic Differential Equations, and is a **stochastic version of the Crank–Nicolson scheme**:

\[
Z_{k,i}^{n+1} = x_i - \frac{\Delta t}{2} \left( u(x_i, t_{n+1}) + u(Z_{k,i}^{n+1}, t_n) \right) + \delta_k.
\]

- In order to obtain second-order consistency, advection and diffusion operators **cannot any longer be considered as decoupled**.
- In this case, the increase in the order of approximation requires that moments of the probability density of \( \sqrt{2\nu \Delta W} \) are **reproduced by the discrete density up to the fifth moment**. This motivates the introduction of further displacements and weights.
Second-order advection–diffusion solver (2)

Passing from first- to second-order, the displacements $\delta_k$ and associated weights are modified as

- **Consistency analysis** for the resulting SL scheme is conceptually simple, but very technical
- **Well scalable** on SIMD architectures
High-order treatment of boundary conditions (1)

Boundary conditions are a typical bottleneck of SL schemes. To avoid the loss of consistency outlined above, we can compute the value at the foot of a characteristic by extrapolation [BCCF]

- To obtain a stable extrapolation, it should be performed on a coarser grid (of space step $h > \Delta x$)
High-order treatment of boundary conditions (1)

Boundary conditions are a typical bottleneck of SL schemes. To avoid the loss of consistency outlined above, we can compute the value at the foot of a characteristic by extrapolation [BCCF]

- To obtain a stable extrapolation, it should be performed on a coarser grid (of space step $h > \Delta x$)
- The analysis of this procedure provides a stable region of thickness $O(h)$ outside of the boundary, where extrapolation is allowed:

$$d(z_{k,i}^{n+1}, \Omega) < C(n_{ex})h$$

- A careful coupling between $h$ and $\Delta x$ is required to keep both stability and consistency rate
High-order treatment of boundary conditions (2)

- The **extrapolation grid** ($Q_2$ in this example) is constructed near the boundary.
- The coupling between $P_2$ internal interpolation and $P_2/Q_2$ extrapolation results in a **second-order treatment of BCs**.
- **Not easy** to obtain a general-purpose implementation on **genuinely unstructured geometries**.
An (approximate) **upwinding** is performed along the advection flow lines (higher order implementations are possible).
Upwinding and point location

- An (approximate) **upwinding** is performed along the advection flow lines (higher order implementations are possible)
- A local interpolation at the foot of a characteristic requires to **locate the point** on the space grid
Upwinding and point location

- An (approximate) upwinding is performed along the advection flow lines (higher order implementations are possible).
- A local interpolation at the foot of a characteristic requires to locate the point on the space grid.
- On structured grids, the point location has $O(1)$ complexity.
- On unstructured grids, the element containing the point should be located using adjacency information (no direct access).
The fast location of a point on an unstructured triangular/tetrahedral grid is a classical problem in computational geometry. Two major solutions are available:

- **Quadtree/octree algorithms**: the computational domain is structured in a tree, and the search requires to visit the tree.
  - Complexity of the visit is **logarithmic**
  - Requires to build an **additional data structure** with **additional memory occupation**
Algorithms for (efficient) point location

The fast location of a point on an unstructured triangular/tetrahedral grid is a classical problem in computational geometry. Two major solutions are available:

- **Quadtree/octree algorithms**: the computational domain is structured in a tree, and the search requires to visit the tree.
  - Complexity of the visit is logarithmic
  - Requires to build an additional data structure with additional memory occupation

- **Walking algorithms**: the algorithm walks along the computational domain towards the final element.
  - Complexity of the visit depends on the distance between the starting and the final element \(O(\sqrt[\text{d}]{\text{number of elements}})\) in the worst case, with \(d=\text{dimension}\)
  - No relevant additional data structure required
Quadtree(octree) algorithms

In this algorithm, a tree structure is associated to the mesh, starting from a rectangle containing all the mesh. A rectangle is refined into four subrectangles until:
In this algorithm, a **tree structure** is associated to the mesh, starting from a **rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or
In this algorithm, a **tree structure** is associated to the mesh, starting from a **rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or
- It intersects a number of triangles $n_t \in [1, q]$ ($q = 3$ in the example above) and no vertex, or
Quadtree(/octree) algorithms

In this algorithm, a **tree structure** is associated to the mesh, starting from a **rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or
- It intersects a number of triangles $n_t \in [1, q]$ ($q = 3$ in the example above) and no vertex, or
- It contains one vertex, regardless of the number of triangles
Walking algorithms (1)

We will use here the **barycentric walk**. In this algorithm, moving from an initial element:

![Diagram](image-url)
Walking algorithms (1)

We will use here the **barycentric walk**. In this algorithm, moving from **an initial element**:

- If the barycentric coordinates w.r.t. the current element are **all positive**, then the element contains the point; otherwise
We will use here the **barycentric walk**. In this algorithm, moving from an initial element:

- If the barycentric coordinates w.r.t. the current element are **all positive**, then the element contains the point; otherwise
- If (at least) **one barycentric coordinate is negative**, then the current element is updated by stepping to the element which faces the point associated to the **negative barycentric coordinate of largest magnitude**
Walking algorithms (2)

- In a **preliminary phase**, it might be necessary to construct the matrix of adjacency between elements.
- The memory occupation for this structure is **not critical**.
Walking algorithms (2)

- In a **preliminary phase**, it might be necessary to construct the matrix of adjacency between elements.
- The memory occupation for this structure is **not critical**.
- Each point location requires a walk from the initial to the final element. On a regular Delaunay triangulation with $N$ elements, the complexity is asymptotically linear w.r.t. the distance ($O(\sqrt{N})$ in the worst case).
# Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

## Quadtree
- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory
Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

Quadtree
- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory

Barycentric walk
- Complexity depends on initialization, slower in the general case
- Easy to code
- Requires little additional memory
Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

### Quadtree
- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory

### Barycentric walk
- Complexity depends on initialization, slower in the general case
- Easy to code
- Requires little additional memory

This motivates the search for a **clever choice** of the initial point $\overline{X}_j$ to initialize the BW (in practice, we can obtain $O(1)$ complexity [CF21])
Speed-up strategy A: following characteristics

The easiest choice is to start the walk **from the node** $x_j$ **itself**:

$$X_j = x_j.$$

- Already used, in particular in combination with **substepping** along trajectories
- Sensitive to an increase of the Courant number
Speed-up strategy B: using the previous time step

\[
\overline{X}_j = X^\Delta(x_j, t_n; t_{n-1})
\]

The second choice is to start the walk from the foot of the characteristic at the previous time step:

\[
\overline{X}_j = X^\Delta(x_j, t_n; t_{n-1}).
\]

- Provides the best guess in case of stationary advection terms
- Less sensitive to an increase of the Courant number
- Relatively scalable on SIMD architectures
Speed-up strategy C: using a neighbouring node

The third choice is to start the walk from the foot of the characteristic of a neighbouring node $x_i$, at the same time step:

$$X_j = X^\Delta(x_i, t_{n+1}; t_n).$$

- Requires to build a spanning tree of the grid
- Insensitive to an increase of the Courant number
- Intrinsically serial
Evaluation of the various strategies (1)

Once verified that complexity of the BW is \textbf{linear w.r.t. the distance}, i.e.,

\[ O \left( C_1 + C_2 \frac{\|X^\Delta(x_i, t_{n+1}; t_n) - X_i\|}{\Delta x} \right), \]

then the various strategies outlined result in a complexity
Evaluation of the various strategies (1)

Once verified that complexity of the BW is linear w.r.t. the distance, i.e.,
\[ O \left( C_1 + C_2 \frac{\|X^\Delta(x_i, t_{n+1}; t_n) - X_i\|}{\Delta x} \right), \]
than the various strategies outlined result in a complexity

(A) \[ O \left( C_1 + C_2 \frac{\|f(x_i, t_{n+1})\|\Delta t}{\Delta x} \right) \]
- Heavy dependence on the (local) Courant number
Evaluation of the various strategies (1)

Once verified that complexity of the BW is \textbf{linear w.r.t. the distance}, i.e.,

$$\mathcal{O} \left( C_1 + C_2 \frac{\|X^\Delta(x_i, t_{n+1}; t_n) - X_i\|}{\Delta x} \right),$$

then the various strategies outlined result in a complexity

\begin{itemize}
  \item[(A)] $\mathcal{O} \left( C_1 + C_2 \frac{\|f(x_i, t_{n+1})\|}{\Delta x} \Delta t \right)$
    \begin{itemize}
      \item Heavy dependence on the (local) Courant number
    \end{itemize}
  \item[(B)] $\mathcal{O} \left( C_1 + C_2 L_t \frac{\Delta t^2}{\Delta x} \right)$
    \begin{itemize}
      \item Light dependence on the (local) Courant number
      \item Takes advantage of (locally) constant in space advection terms
    \end{itemize}
\end{itemize}

In all three cases, we obtain $\mathcal{O}(1)$ cost under linear refinements.
Evaluation of the various strategies (1)

Once verified that complexity of the BW is linear \( \text{w.r.t. the distance} \), i.e.,

\[
O \left( C_1 + C_2 \frac{\| X^\Delta(x_i, t_{n+1}; t_n) - X_i \|}{\Delta x} \right),
\]

then the various strategies outlined result in a complexity

(A) \[
O \left( C_1 + C_2 \frac{\| f(x_i, t_{n+1}) \| \Delta t}{\Delta x} \right)
\]

- Heavy dependence on the (local) Courant number

(B) \[
O \left( C_1 + C_2 L_t \frac{\Delta t^2}{\Delta x} \right)
\]

- Light dependence on the (local) Courant number
- Takes advantage of (locally) stationary advection terms

(C) \[
O \left( (C_1 + C_2) + C_2 L_x \Delta t \right)
\]

- No dependence on the (local) Courant number
- Takes advantage of (locally) constant in space advection terms

In all three cases, we obtain \( O(1) \) cost under linear refinements
Evaluation of the various strategies (2)

Typical behaviour of the various strategies in term of CPU time, for varying time Lipschitz constant and as a function of the Courant number:

- Strategy A eventually becomes the worst choice at the increase of the Courant number
- Strategy B always performs the best for low or moderate Courant numbers
- Strategy C always performs the best for large Courant numbers
Evaluation of the various strategies (3)

At a comparison with a **structured triangulation**, the walking strategy (of type B in this case) **may even turn to be faster**. It is to be remarked that the computation of barycentric coordinates is performed in both cases to obtain the interpolate of the solution:
Evaluation of the various strategies (4)

Last, we perform a comparison of the search strategy B with the general-purpose Matlab function `pointLocation`:

To obtain a fair evaluation, the comparison is performed by replacing `pointLocation` with a Mex-compiled version of the BW, resulting in a speedup of 10–20 times.
3D + Boundary conditions

- Straightforward adaptation of the BW strategies to the 3D case
- However, BW may need to switch from an internal walk to a boundary walk and vice versa
CUDA parallelization

- On a SIMD architecture, we can set in parallel form:
  - Computation of feet of characteristics
  - Point location on the grid
  - Interpolation

- A careful parallelization allows to obtain relevant speed-ups with a relatively simple and unexpensive architecture

\[ u(x, y, z, t) = (0.5, 0.25 \cos(2\pi t), 0.25 \sin(\pi t)) \]

Hardware: Intel i9–9900K, 16 cores, 3.6 GHz, 32 GB RAM; Nvidia RTX–2080ti

Roberto Ferretti (Roma Tre)
CUDA parallelization

- On a SIMD architecture, we can set in parallel form:
  - Computation of feet of characteristics
  - Point location on the grid
  - Interpolation

- A careful parallelization allows to obtain relevant speed-ups with a relatively simple and unexpensive architecture

- **Test**: plain advection, time-dependent transport field and boundary conditions on the cylinder \((0, 2) \times B(0, 1)\).

\[
\mathbf{u}(x, y, z, t) = (0.5, 0.25 \cos(2\pi t), 0.25 \sin(\pi t))
\]

- **Scheme**: inviscid SL, point location via strategy B, \(P_2\) interpolation
  - Space grids from about 1M to about 9M elements
  - Average Courant numbers 4 to 8; number of time steps 16 to 64

- **Hardware**: Intel i9–9900K, 16 cores, 3.6 GHz, 32 GB RAM; Nvidia RTX–2080ti
CUDA parallelized code

- Maximum serial CPU time about 300s, maximum CPU time in parallel mode about 3.7s
- Speed-ups ranging from 70x to 93x, higher for larger Courant numbers and refined grids
A preliminary 2D test

- Airfoil at $Re = 5000$
- Barycentric walk-based point location, Chorin–Temam scheme, $P_2$ interpolation
Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes.
- Various drawbacks of this class of schemes have been removed, or at least reduced.
Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
- Various drawbacks of this class of schemes have been removed, or at least reduced
- First-order schemes work nicely, second-order in progress
Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
- Various drawbacks of this class of schemes have been removed, or at least reduced
- First-order schemes work nicely, second-order in progress
- GPU-based 3D implementation in progress, too – promising results so far
References


