WIND - Lattice-Boltzmann Fluid Dynamics

Antonino Natale, 200705

Luglio 2020

Sommario

I metodi reticolari di Boltzmann, in fluidodinamica computazionale, sono un insieme di tecniche usate per la simulazione dei fluidi. L'equazione di Boltzmann viene risolta a partire dalle equazioni di Navier-Strokes per simulare il flusso di un fluido newtoniano mediante modelli di collisione. In questo documento, mi occuperò principalmente di questo particolare automa cellulare sfruttando un approccio volto all'efficenza e scalabilità richiesta dalla complessità computazionale dello stesso.

Indice

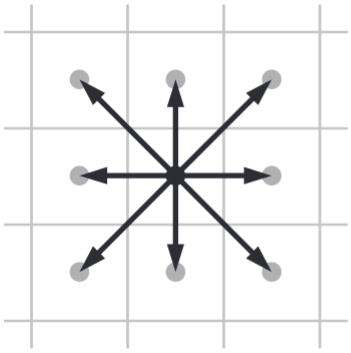
1 Introduzione

1.1 Dinamica dei fluidi

La fluidodinamica è una branchia notoriamente difficile della fisica, dove pochissimi problemi possono essere risolti analiticamente. Un fluido, può subire oscillazioni lineari proprio come un solido, ma a differenza di quest'ultimo può anche scorrere attorno ad ostacoli e, in molte circostanze anche ruotare vorticosamente su stesso. Questo comportamento è intrinsecamente non lineare e difficile da comprendere in modo quantitativo. Normalmente, descriviamo lo stato macroscopico di un fluido in termini di densità ρ, e di velocità $\vec{u}$, entrambi in funzione di posizione e tempo. A queste funzioni seguono un insieme di equazioni differenziali parziali, chiamate *equazioni di Navier-Strokes*, le quali derivano dalle leggi di Newton e da alcune assunzioni empiriche riguardo il comportamento dei fluidi. A causa della complessità analitica delle equazioni di Navier-Strokes, nei problemi di fluidodinamica, si preferisce un approccio computazionale. Nascono quindi, i *Computational Fluid Dynamics (CFD)*, ovvero metodi di discretizzazione delle equazioni di Navier-Strokes, con conseguente risoluzione numerica per date condizioni a contorno.

1.2 I reticoli di Boltzmann

Nell'approccio dei reticoli di Boltzmann, discretizziamo sia spazio che tempo, affinchè siano permesse solo ed esclusivamente certe velocità vettoriali discrete. Nel progetto, la struttura reticolare utilizzata è la **D2Q9**, dove si assume uno spazio a due dimensioni xy e nove possibili componenti vettoriali $\vec{e}$. Una di queste componenti è zero, mentre le altre otto descrivono una particella e i suoi possibili movimenti nelle otto direzioni adiacenti al reticolo - orizzontale, verticale e diagonale.



$$\vec{e}_0 = 0$$
$$\vec{e}_1 = (1,0) \qquad \vec{e}_5 = (1,1)$$
$$\vec{e}_2 = (0,1) \qquad \vec{e}_6 = (-1,1)$$
$$\vec{e}_3 = (-1,0) \qquad \vec{e}_7 = (-1,-1)$$
$$\vec{e}_4 = (0,-1) \qquad \vec{e}_8 = (1,-1)$$

L'obiettivo è quello di associare delle probabilità w_i a questi nove vettori velocità, al fine di garantire l'accuratezza della distribuzione di Boltzmann. Per un fluido in quiete con velocità $\vec{u} = 0$, la probabilità ottimale risulta essere $\frac{4}{9}$; per uno in movimento nelle quattro direzioni cardinali è $\frac{1}{9}$, e per le diagonali $\frac{1}{36}$.

$$w_0 = \frac{4}{9} \tag{1}$$

$$w_1 = w_2 = w_3 = w_4 = \frac{1}{9} \tag{2}$$

$$w_5 = w_6 = w_7 = w_8 = \frac{1}{36} \tag{3}$$

1.2.1 Equazione di equilibrio

Nell'algoritmo dei reticoli di Boltzmann, elementi di particolare rilevanza sono le nove densità microscopiche $n_i(x, y)$, che scorrono attraverso le nove componenti vettoriali $\vec{e_{ii}}$ di ogni reticolo. In un dato istante di tempo, per ogni reticolo, queste nove densità possono avere valori positivi che sommati tra loro restituiscono la densità macroscopica ρ, e i componenti x e y della vettore velocità macroscopica $\vec{u}$. I risultati di queste due variabili costituiscono le componenti essenziali per l'equazione di equilibrio termico:

$$n_i^{eq} = \rho w_i[1 + 3\vec{e}_i \cdot \vec{u} + \frac{9}{2}(\vec{e}_i \cdot \vec{u})^2 - \frac{3}{2}|\vec{u}|^2] \tag{4}$$

Il primo passo è quindi quello di calcolare, per ogni step della simulazione, la (**??**) ad ogni singolo reticolo che compone il fluido. Applicando quindi:

$$n_i^{new} = n_i^{old} + \omega(n_i^{eq} - n_i^{old}) \tag{5}$$

Lasciamo che le nove densità microscopiche si avvicinino, poco per volta, all'equilibrio termico T. Dove ω è un valore scalare che indica la *viscosità* del fluido, ovvero quanto più tempo le collisioni impiegano nel processo di equilibrio - può assumere valori da 0 a 2.

1.2.2 Algoritmo

Per ogni step della simulazione si applicano i seguenti passi:

1. **Collisione.** Per ogni reticolo:
   (a) Dalle nove densità microscopiche n_i si costruiscono ρ e $\vec{u}$;
   (b) Si applica la (**??**) per calcolare l'equilibrio termico T;
   (c) Si aggiornano le nove densità microscopiche in accordo con l'equazione (**??**).
2. **Scorrimento.** Per ogni reticolo, si scorrono le densità n_i che lo compongono nei rispettivi reticoli adiacenti, copiandone l'appropriato valore;
3. **Rimbalzo.** Per ogni barriera - condizione a contorno - si inverte lo scorrere delle densità n_i alterandone il vettore $\vec{u}$.

2 Calcolo Parallelo

2.1 Message Passing Interface

MPI è uno standard di comunicazione tra nodi appartenenti ad un cluster di computer che eseguono un programma parallelo. Lo scopo nel progetto è di accelerare, mediante parallelizzazione su più nodi, l'esecuzione dell'algoritmo dei reticoli di Boltzmann, al fine di avere una simulazione interattiva e in tempo reale dello scorrimento di un fluido newtoniano attraverso degli ostacoli inseriti dall'utente. Realizzare ciò, ha richiesto la considerazione di un nuovo approccio che fosse efficente, in termini di prestazioni, e scalabile, ovvero, utilizzabile in un cluster più complesso costruito su più nodi, anche remoti. Prima di descrivere il metodo utilizzato è doveroso introdurre due concetti fondamentali di MPI.

2.1.1 Window Object e Memoria condivisa

I nodi che esistono all'interno di uno stesso dispositivo fisico sono in grado di interfacciarsi tra loro mediante tecniche di memoria condivisa. Ossia, richiedono al sistema operativo un'area di memoria visibile ad entrambi i processi, nella quale inserire direttamente informazioni e dati senza ricorrere all'utilizzo delle routine `MPI_Send/MPI_Recv`. MPI, allora, introduce una nuova entità, chiamata `Window` con lo scopo di gestire zone di memoria condivisa. In particolare, nel progetto, ho utilizzato la funzione:

```
MPI_Win_allocate_shared (
    unit_size * sizeof(struct unit),
    sizeof(struct unit),
    MPI_INFO_NULL,
    MPI_COMM_LOCAL,
    &units,
    &MPI_LOCAL_WINDOW
);
```

Il risultato è analogo alla funzione `MPI_Scatter()`, ossia, l'area di memoria viene partizionata per singolo nodo. La differenza è che ciò permette di allocare zone di memoria adiacenti e condivise per ogni singolo nodo all'interno di un dato gruppo `MPI_COMM_LOCAL`.

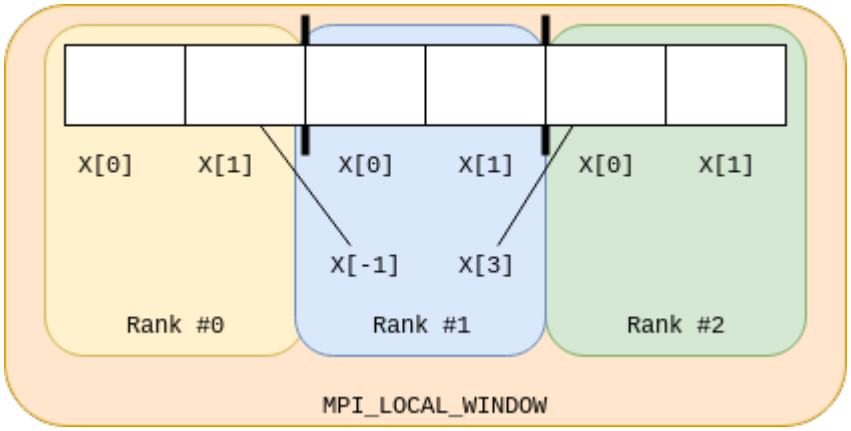


Figura 1: Area di memoria condivisa all'interno di un Window Object

2.1.2 Comunicatori

Un comunicatore è un oggetto che descrive un gruppo di nodi all'interno di un cluster. Per questo progetto si è reso necessario suddividere i nodi al fine creare gruppi di comunicatori locali che interagissero tra loro e condividessero la stesso oggetto `Window`. Il motivo, è principalmente prestazionale; i gruppi sono suddivisi secondo il criterio `MPI_COMM_TYPE_SHARED`, ovvero gruppi di nodi, facenti parte dello stesso dispositivo fisico, che possono creare un area di memoria condivisa.

```
MPI_Comm_split_type (
    MPI_COMM_WORLD,
    MPI_COMM_TYPE_SHARED,
    world_rank,
    MPI_INFO_NULL,
    &MPI_COMM_LOCAL
);
```

Ciò permette ad ogni singolo nodo all'interno di ogni gruppo `MPI_COMM_LOCAL` di condividere, in modo diretto e veloce, i propri dati sulla simulazione in corso.

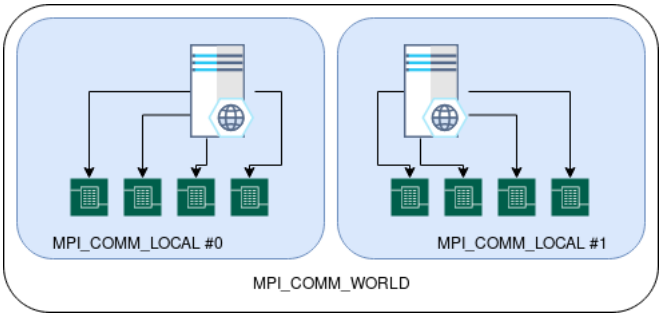


Figura 2: Topologia dei nodi

2.2 Parallelizzazione e Ghost Cells

Sfruttando i concetti di **??** e **??**, ho deciso di suddividere i nodi per gruppi `MPI_COMM_LOCAL` dal gruppo globale più grande `MPI_COMM_WORLD` ed assegnare loro un `local_rank` e un `world_rank` di riferimento. Ad ogni `MPI_COMM_LOCAL` ho allocato una `MPI_LOCAL_WINDOW` di dimensioni pari ai reticoli che ogni nodo deve gestire. Infine, l'algoritmo si sviluppa principalmente in tre fasi:

1. **Collisione.** Per ogni nodo locale, eseguo i calcoli di **??**
2. **Scorrimento Locale.** Per ogni nodo locale applico lo scorrimento descritto in **??** accedendo alle aree condivise di memoria degli altri nodi locali appartenenti allo stesso gruppo `MPI_COMM_LOCAL`
3. **Scorrimento Globale.** Per ogni nodo locale adiacente ad un altro `MPI_COMM_LOCAL` esterno, utilizzo le routine `MPI_Send/MPI_Recv` al fine di ottenere le *Ghost Cells* ed applicare a quest'ultime lo scorrimento dei reticoli.

2.2.1 Vantaggi

I vantaggi risiedono principalmente nel numero ridotto di chiamate ad `MPI_Send/MPI_Recv`, con conseguente diminuzione drastica dei tempi di esecuzione, necessari al fine di ottenere una simulazione rapida e in tempo reale.

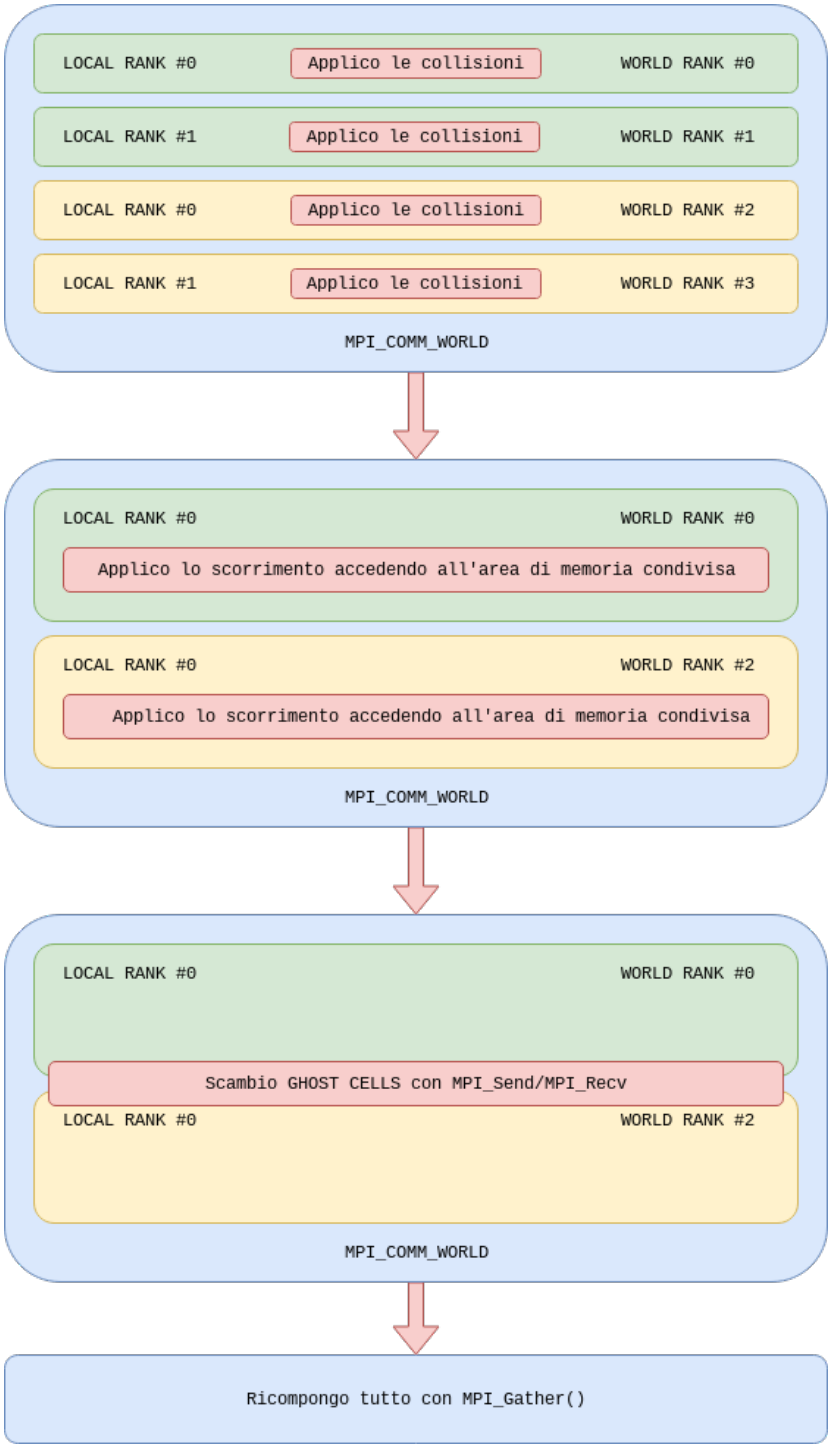


Figura 3: Algoritmo in parallelo

3 Progetto

Infine, veniamo al codice. Tutto ciò che è stato accennato nelle prime sezioni è trascritto in C++ all'interno del progetto, con l'ausilio di `OpenMPI` e `Allegro`.

3.1 Variabili

3.1.1 Units

Matrice di locale di reticoli di Boltzmann. Viene assegnata ad ogni nodo presente in `MPI_COMM_WORLD` ed è definita come `struct unit* units;`

3.1.2 Frame

Matrice globale di reticoli di Boltzmann. Viene allocata solo ed esclusivamente dal nodo primario di `MPI_COMM_WORLD`, ossia con `world_rank = 0`. Il suo scopo è puramente di disegno e raccolta delle computazioni dei singoli nodi ed è definita come `struct unit* frame;`

3.1.3 Ghost Cells

Matrice mono-dimensionale che descrive la riga, qualora vi fosse, superiore o inferiore del reticolo locale `units`. Sono due e sono definiti come `struct unit* up_units` e `struct unit* bottom_units`.

3.1.4 Velocità e viscosità fluido

Entità che descrivono la velocità $\vec{u}$ e la viscosità ω del fluido, vedi **??**. Sono definite come `v2d flow_speed;` e `double flow_viscosity;`

3.1.5 Booleane di controllo

Permettono di interagire con l'esecuzione della simulazione, sono: `running`, `paused`, `resetting`.

3.1.6 MPI_COMM_LOCAL

Comunicatore creato a partire dal gruppo principale `MPI_COMM_WORLD` che contiene nodi facenti parte dello stesso dispositivo fisico.

```
MPI_Comm_split_type (
    MPI_COMM_WORLD,
    MPI_COMM_TYPE_SHARED,
    world_rank,
    MPI_INFO_NULL,
    &MPI_COMM_LOCAL
);
```

3.1.7 MPI_LOCAL_WINDOW

Window Object che descrive l'area di memoria condivisa all'interno di ogni singolo `MPI_COMM_LOCAL`

```
MPI_Win_allocate_shared (
    unit_size * sizeof(struct unit),
    sizeof(struct unit),
    MPI_INFO_NULL,
    MPI_COMM_LOCAL,
    &units,
    &MPI_LOCAL_WINDOW
);
```

3.1.8 Ranks e num_procs

Entità di tipo intero che identificano informazioni riguardo al gruppo globale/locale di appartenenza.

```
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_num_procs);

MPI_Comm_rank(MPI_COMM_LOCAL, &local_rank);
MPI_Comm_size(MPI_COMM_LOCAL, &local_num_procs);
```

3.2 Strutture Dati

Sono presenti diversi tipi di `MPI_DataType`. Il loro scopo è quello di ottimizzare il processamento dei dati da parte delle routine di MPI.

3.2.1 Vettore

L'entita che descrive un vettore bidimensionale è definita come `struct v2d` e contiene:

- Componente X
- Componente Y

```
MPI_Datatype v2d_types[] = { MPI_DOUBLE, MPI_DOUBLE };
MPI_Aint v2d_offsets[] = { 0, sizeof(double) };
int v2d_blocks[]       = { 1, 1 };

MPI_Type_create_struct (
    2,
    v2d_blocks,
    v2d_offsets,
    v2d_types,
    &MPI_TYPE_V2D
);

MPI_Type_commit(&MPI_TYPE_V2D);
```

3.2.2 Reticolo

L'entità che descrive il reticolo di Boltzmann è definita come `struct unit` e contiene:

- Tipo di unita: Fluido, Barriera
- Le nove densità microscopiche n_i
- Densità macroscopica ρ
- Vettore velocità $\vec{u}$
- Intensità del vortice limitrofo

```
MPI_Datatype unit_types[] =
    { MPI_DOUBLE, MPI_CXX_BOOL, MPI_TYPE_V2D, MPI_DOUBLE,
       MPI_DOUBLE };

MPI_Aint unit_offsets[] = {
    offsetof(unit, n),
    offsetof(unit, barrier),
    offsetof(unit, u),
    offsetof(unit, rho),
    offsetof(unit, curl),
};

int unit_blocks[] = { 9, 1, 1, 1, 1 };

MPI_Type_create_struct (
    5,
    unit_blocks,
    unit_offsets,
    unit_types,
    &MPI_TYPE_UNIT
);

MPI_Type_commit(&MPI_TYPE_UNIT);
```

3.3 Avvio

All'avvio, il nodo primario globale, ossia con `world_rank = 0` alloca le strutture di cui necessita per il disegno e inizializza `Allegro`, istanzia una finestra ed effettua il polling degli eventi, gestendo grafica e input utente. In seguito, ogni nodo, dopo aver istanziato le entità di MPI, da inizio all'esecuzione dell'algoritmo descritto in **??**. Il fluido, inizialmente è in uno stato di quiete, ovvero di equilibrio termico. Non appena l'utente disegna, alterando la morfologia delle condizioni a contorno, avviene la destabilizzazione del fluido stesso, portando a tale risultato:

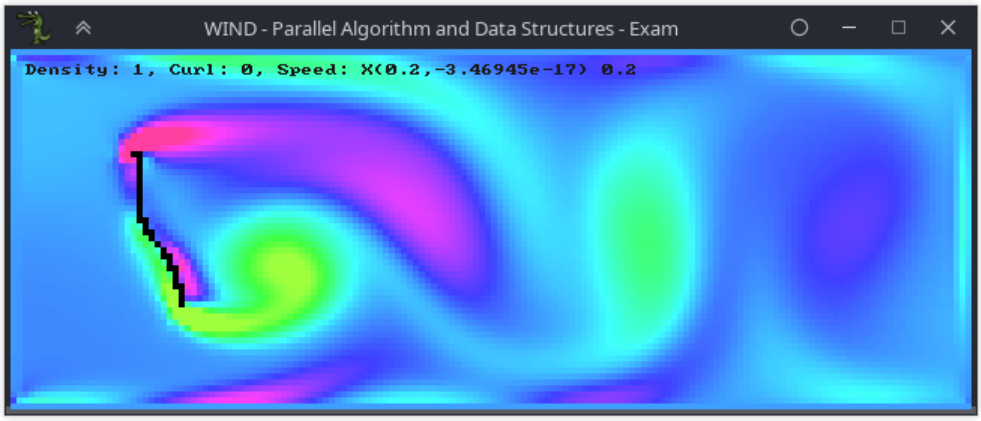


Figura 4: Avvio e disegno di una barriera

Come si può notare dalla figura (**??**), la posizione e rotazione della barriera al centro crea una turbolenza con il susseguirsi di vortici instabili di colore verde e fucsia.

3.4 Istruzioni

L'utente può interagire con la simulazione attraverso Mouse e Tastiera con i seguenti comandi.

3.4.1 Tastiera

- **R**: Sospende/Riprende la simulazione
- **C**: Cancella tutte la barriere
- **I**: Mostra topologia dei nodi globali
- **U**: Mostra topologia dei nodi locali
- **1 .. 5**: Cambia modalità di visualizzazione:
  1. Vortici
  2. Densità
  3. Velocità totale

4. Velocità X
5. Velocità Y

- **NumPad**: Cambia direzione del fluido.

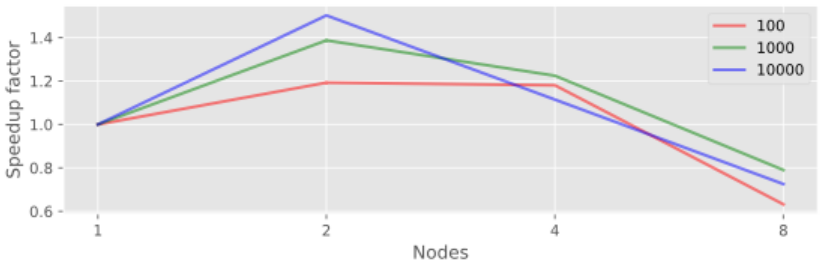
3.4.2 Mouse

L'utente può disegnare le barriere sullo schermo cliccando con il tasto sinistro del mouse.

3.5 Benchmarks

I test sono eseguiti su:

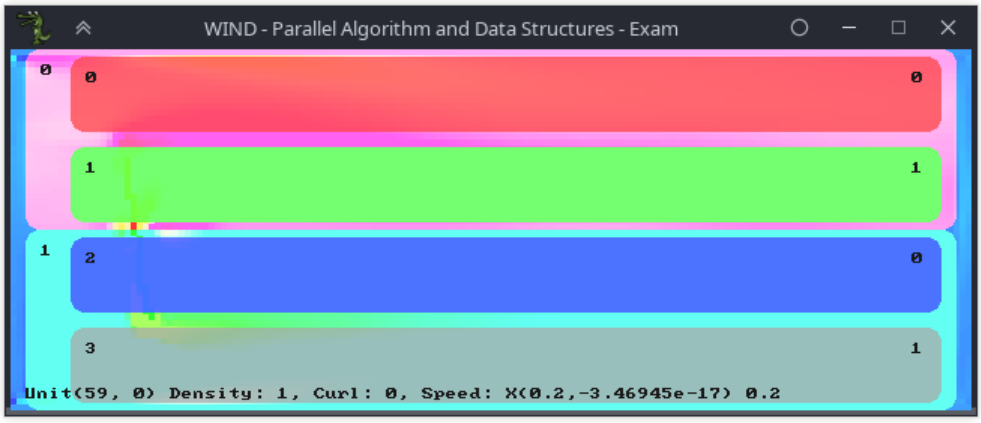
```
Linux 5.4.44-1-LTS
Intel(R) Core(TM) i3-4160 CPU @ 3.60GHz (Dual Core)
```



Steps	1	2	4	8
100	0.088214	0.073977	0.074715	0.139526
1000	0.870845	0.627775	0.711142	1.10187
10000	8.569142	5.704306	7.687985	11.809135

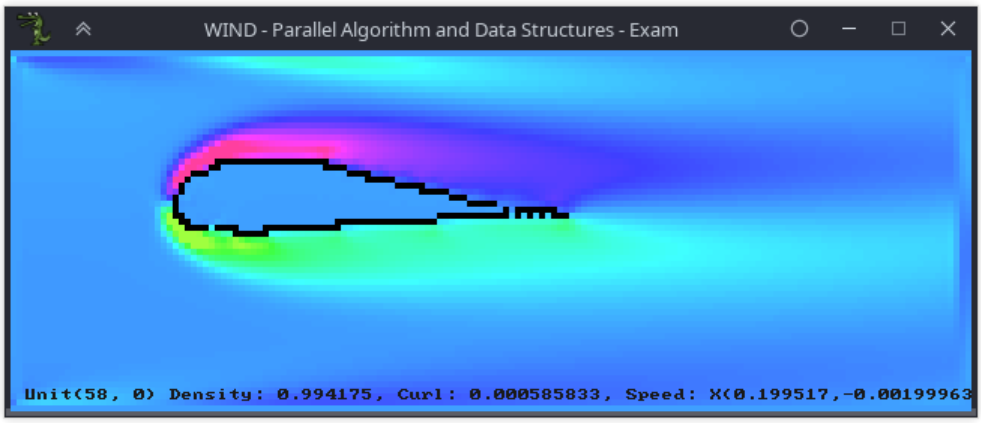
Benchmark e Speedup factor

3.6 Screenshots

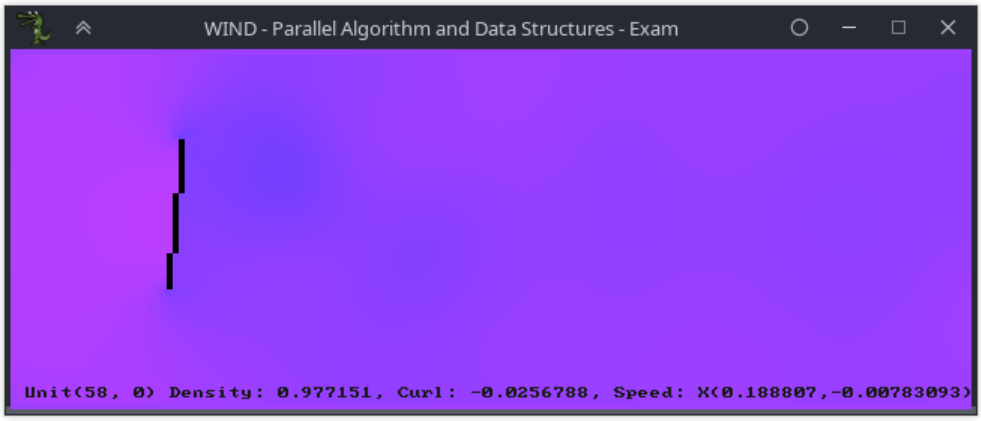


Esempio di Topologia. A sinistra il rank globale, a destra quello locale

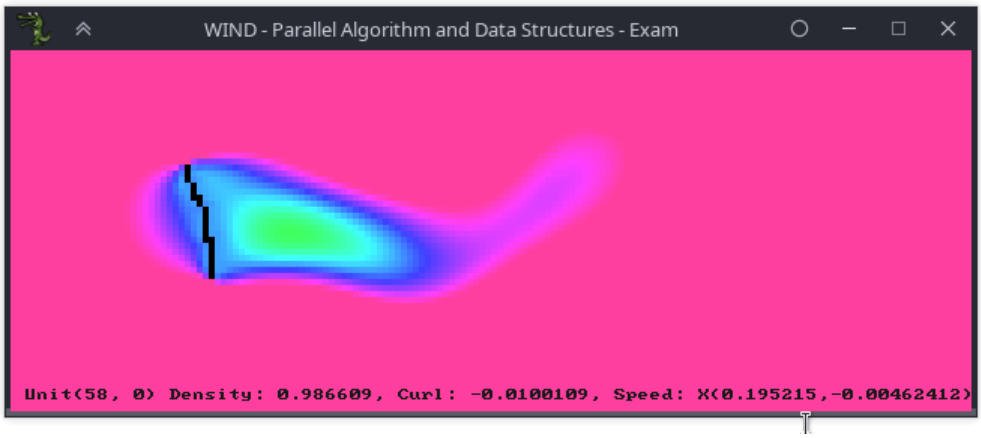
Questo è un esempio di topologia dove sono presenti due `MPI_COMM_LOCAL` e quattro nodi distruibuiti: due per il primo gruppo, due per il secondo.



Piano a profilo aerodinamico



Densità fluido



Velocità $\vec{u}$ totale