Wind Tunnel Simulation

Cicio Ionuț

01/01/2025

Contents

1 Introduction

You are provided with a sequential code that simulates in a simplified way, how the air pressure spreads in a Wind Tunnel, with fixed obstacles and flying particles.

- the simulation represents a zenithal view of the tunnel, by a 2-dimensional cut of the tunnel along the main air-flow axis
- the space is modelled as a set of discrete and evenly spaced points or positions
- a **bidimensional array** is used to represent the value of **air pressure at each point** in a given **time step**
- another **unidimensional array** stores the information of a collection of **flying particles** (they can be pushed and moved by the air flow), or **fixed structures** that cannot move
- all particles have a **mass** and a **resistance** to the flow of air, partially *blocking* and *deviating* it

Fan inlet

- there is an air inlet at the start of the tunnel with a big fan
- the values of air pressure introduced by the fan are represented in the row 0 of the array
- after a given number of simulation steps the fan **turning-phase** is slightly changed recomputing the values of air pressure introduced, also adding some random noise
- the fan phase is cyclically repeated
- every time step, the values of air pressure in the array are updated with the old values in upper rows, propagating the air flow to the lower rows
- on each iteration only some rows are updated
- they represent wave-fronts, separated by a given number of rows.

Fixed and flying particles

- the information of a particle includes:
  - position coordinates
  - a two dimensional velocity vector
  - mass
  - and air flow resistance
- all of them are numbers with decimals represented in fixed position
- the particles let some of the air flow to advance depending on their resistance
- the air flow that is blocked by the particle is deviated creating more pressure in the front of the particle
- the flying particles are also pushed by the air when the pressure in their front is enough to move their mass

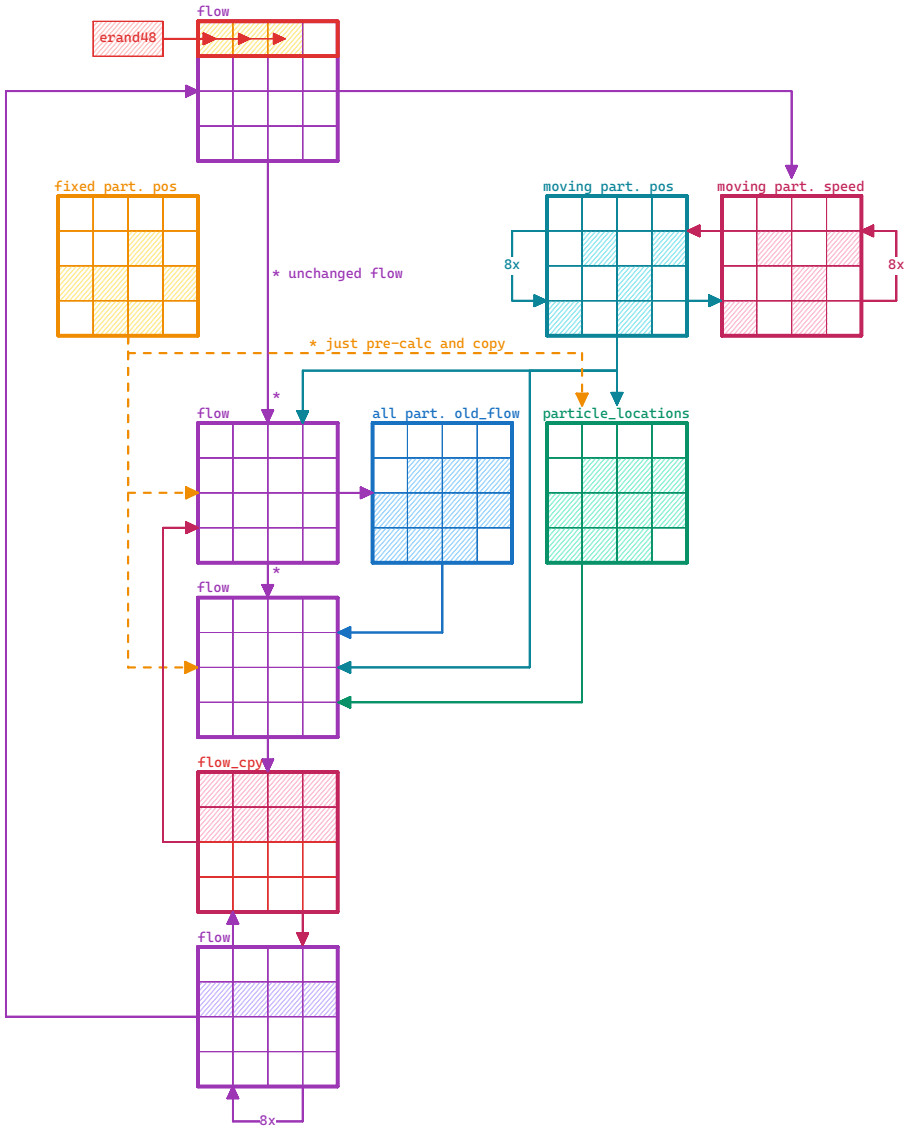
- the updates of deviated pressure and flying particles movements are done after a given number of time steps (the same as the distance between propagation wave fronts)

End of simulation

- the simulation ends:
  - after a fixed number of iterations (input argument)
  - or if the maximum variation of air pressure in any cell of the array is less than a given threshold
- at the end, the program outputs a results line with:
  - the number of iterations done
  - the value of the maximum variability of an update in the last iteration
  - and the air pressure value of several chosen array positions in three rows:
    - near the inlet
    - in the middle of the tunnel
    - in the last row of the tunnel;
    - or in the last row updated by the first wave front in case that the simulation stops before the air flow arrives at the last row

Symbols:

- `[ ]` One or more particles/obstacles in the position
- `.` Air pressu in the range $[0, 0.5)$
- `n` A number n represents a pressure between $[n, n + 1)$
- `*` Air pressure equal or greater than 10



2 Scalabilità

- rispetto alla dimensione della griglia
- rispetto alla dimensione dell'inlet
- rispetto al numero di particelle e al tipo

Il tutto testato con diversi numeri casuali (usando numeri primi)

3 Correttezza

- valgrind e roba cicli CPU etc...

- perf stat
- gprof

4 Architetture

- statistiche varie architetture e roba simile
  - getconf LEVEL3_CACHE_ASSOC
  - lscpu| grep -E '^Thread|^Core|^Socket|^CPU('

5 Alcune note

Task parallelism

- Partition various tasks among the cores
- The "temporal parallelism" we have seen in circuits is a specific type of task parallelism

Data parallelism

- Partition the data used in solving the problem among the cores
- Each core carries out similar operations on it's part of the data
- Similar to "spatial parallelism" in circuits

Which one can i use here? Maybe both? Tasks:
- update the inlet
- move the particles
- update the flow
- propagate

To propagate I need all the above… I can propagate multiple times for one of the above (a.k.a 8 times)

If I want to parallelize the data, I have to work on distributing the data of the matrices The problem with distributing the data is that some data depends on other data… maybe? Not as much as the tasks.

In practice, cores need to coordinate, for different reasons:

Communication e.g., one core sends its partial sum to another core

Load balancing share the work evenly so that one is not heavily loaded (what if some file to compress is much bigger than the others?)

- If not, p-1 one cores could wait for the slowest (wasted resources & power!)

Synchronization each core works at its own pace, make sure some core does not get too far ahead

- E.g., one core fills the list of files to compress. If the other ones start too early they might miss some files

Load balancing is secondary here… can be done at the start probably. Communication and Synchronization are the important ones, but how?

Foster's methodology

Partitioning divide the computation to be performed and the data operated on by the computation into small tasks. The focus here should be on identifying tasks that can be executed in parallel.

Communication determine what communication needs to be carried out among the tasks identified in the previous step

Agglomeration or aggregation combine tasks and communications identified in the first step into larger tasks. For example, if task A must be executed before task B can be executed, it may make sense to aggregate them into a single composite task.

Mapping assign the composite tasks identified in the previous step to processes/threads. This should be done so that communication is minimized, and each process/thread gets roughly the same amount of work.

Bibliography