Introduction to High-Performance Computing

Spoiler

Giorgio Amati

Corso di dottorato in Ingegneria Aeronautica e Spaziale 2024

g.amati@cineca.it / g.amaticode@gmail.com

Agenda

- ✓ **HPC: What it is?**
- ✓ **Spoiler…**
- ✓ Hardware: how it works
- ✓ Algorithm vs. Implementation
- ✓ Compiler
- ✓ Parallel Paradigm
- ✓ Conclusions & Comments

An example…..

Implementing Turbulence model (Smagorinsky Model) in a 3D Lattice Boltzmann model

$$\nu_{total} = \nu_0 + \underbrace{(C_S\Delta)^2|S|}_{\nu_t}$$

$$S_{ij} = -\frac{3\omega_s}{2\rho_{(0)}}\Pi_{ij}^{(neq)}$$

$$\Pi_{ij}^{(neq)} = \sum_q c_{qi}c_{qj}\, f_q^{(neq)}$$

An example…..

$$\Pi_{ij}^{(neq)} = \sum_{q} c_{qi} c_{qj} f_q^{(neq)}$$

- ✓ 19*6*2 multiplications
- ✓ 19*6 sums
- ✓ 19*6*3 loads

```
do i
   do j
      do k
…
         do pop=1,19
            Pxx = Pxx + cx(pop)*cx(pop)*neq(pop)
            Pyy = Pyy + cy(pop)*cy(pop)*neq(pop)
            Pzz = Pzz + cz(pop)*cz(pop)*neq(pop)
            Pxy = Pxy + cx(pop)*cy(pop)*neq(pop)
            Pxz = Pxz + cx(pop)*cz(pop)*neq(pop)
            Pyz = Pyz + cy(pop)*cz(pop)*neq(pop)
         enddo
…
```

first “optimization”

$$\Pi_{ij}^{(neq)} = \sum_q c_{qi} c_{qj} f_q^{(neq)}$$

- ✓ 19*6*2 multiplications
- ✓ 19*6 sums
- ✓ 19*6*2 loads

```
…
   Pxx = cx(01)*cx(01)*n01 + cx(02)*cx(02)*n02 + &
         cx(03)*cx(03)*n03 + cx(04)*cx(04)*n04 + &
         cx(05)*cx(05)*n05 + cx(06)*cx(06)*n06 + &
         cx(07)*cx(07)*n07 + cx(08)*cx(08)*n08 + &
         cx(09)*cx(09)*n09 + cx(10)*cx(10)*n10 + &
         cx(11)*cx(11)*n11 + cx(12)*cx(12)*n12 + &
         cx(13)*cx(13)*n13 + cx(14)*cx(14)*n14 + &
         cx(15)*cx(15)*n15 + cx(16)*cx(16)*n16 + &
         cx(17)*cx(17)*n17 + cx(18)*cx(18)*n18 + &
         cx(19)*cx(19)*n19
   Pyy = cy(01)*cy(01)*n01 + cy(02)*cy(02)*n02 + &
                   …
```

Some performance figures

- ✓ Left: time for 256^3 simulation without Smagorinsky Model
- ✓ Right: time for 256^3 simulation with Smagorinsky Model

```
# Time for section
# init    time   0.115159E+01
# loop    time   0.716699E+02
# coll    time   0.641728E+02
# bc      time   0.645609E+01
# diagno  time   0.102584E+01
```

```
# Time for section
# init    time   0.111767E+01
# loop    time   0.866046E+02
# coll    time   0.791241E+02
# bc      time   0.645086E+01
# diagno  time   0.101066E+01
```

- ✓ the Smagorinsky model introduce a 1.23x slowdown!
- ✓ Can be reduced?

Looking at operation

- ✓ For each Tensor element `Pij` we perform
  - 19*3 load operation
  - 19 sums
  - 19*2 products

```
Pxy = cx(01)*cy(01)*n01 + cx(02)*cy(02)*n02 + &
      cx(03)*cy(03)*n03 + cx(04)*cy(04)*n04 + &
      cx(05)*cy(05)*n05 + cx(06)*cy(06)*n06 + &
      cx(07)*cy(07)*n07 + cx(08)*cy(08)*n08 + &
      cx(09)*cy(09)*n09 + cx(10)*cy(10)*n10 + &
      cx(11)*cy(11)*n11 + cx(12)*cy(12)*n12 + &
      cx(13)*cy(13)*n13 + cx(14)*cy(14)*n14 + &
      cx(15)*cy(15)*n15 + cx(16)*cy(16)*n16 + &
      cx(17)*cy(17)*n17 + cx(18)*cy(18)*n18 + &
      cx(19)*cy(19)*n19
```

- ✓ But `cx(j),cy(j),cz(j)` can assume only values `-1,0,+1`
  - only 10 non zero values for `cx(j),cy(j),cz(j)`

Second optimization:

$$\Pi_{ij}^{(neq)} = \sum_q c_{qi} c_{qj} f_q^{(neq)}$$

```
Pxx =  n01 +n02 +n03 +n04 +n05 +n10 +n11 +n12 +n13 +n14
Pyy =  n01 +n03 +n07 +n08 +n09 +n10 +n12 +n16 +n17 +n18
Pzz =  n02 +n04 +n06 +n07 +n09 +n11 +n13 +n15 +n16 +n18
!
Pxz = -n02 +n04 +n11 -n13
Pxy = -n01 +n03 +n10 -n12
Pyz = +n07 -n09 +n16 -n18
…
```

Drastic operation reduction!

- ✓ Only 36 sums
- ✓ No multiplication at all
- ✓ no load of `cx(pop),cy(pop),cz(pop)` vectors

other performance figures

- ✓ Left: time for 256^3 simulation with Smagorinsky Model (no optimized)
- ✓ Right: time for 256^3 simulation with Smagorinsky Model (optimized)

```
# Time for section
# init    time   0.111767E+01
# loop    time   0.866046E+02
# coll    time   0.791241E+02
# bc      time   0.645086E+01
# diagno time    0.101066E+01
```

```
# Time for section
# init    time   0.112207E+01
# loop    time   0.730055E+02
# coll    time   0.655008E+02
# bc      time   0.646228E+01
# diagno time    0.102758E+01
```

- ✓ Now the Smagorinsky impact is negligible…

Lesson Learned

- ✓ Reduction operation/complexity is the key
- ✓ Monitoring performance help to understand if there’s room for improvement
- ✓ For a computer each operation has a cost, that could be extremely expensive
- ✓ Remove everything unnecessary
- ✓ Avoid rigid translation from the mathematical description of the algorithm