# Performance Analysis and Optimization

Raunak Borker

# Part1 Implementation Time Comparison

- Performance comparison over **25 steps**
  - **Python** implementation: **7.1sec**
  - **C++** implementation: **0.3sec**
  - **24x** speedup! (not good enough)

# Expectation before analyzing

- The **bottleneck** in performance could probably be due to **InternalForce** computations as it involves N computations for every element -
  total of $\mathbf{N^2}$ computations

# Profiling and Analysis

- gprofile results indicated **ComputeEnergy**, **InternalForce** and **PutInBox** methods are major time consumers

- **std::vector append** also showed up as one of the heavy users

- cachegrind indicated **~0.0% L1 miss rates**

# Part 1 Implementation

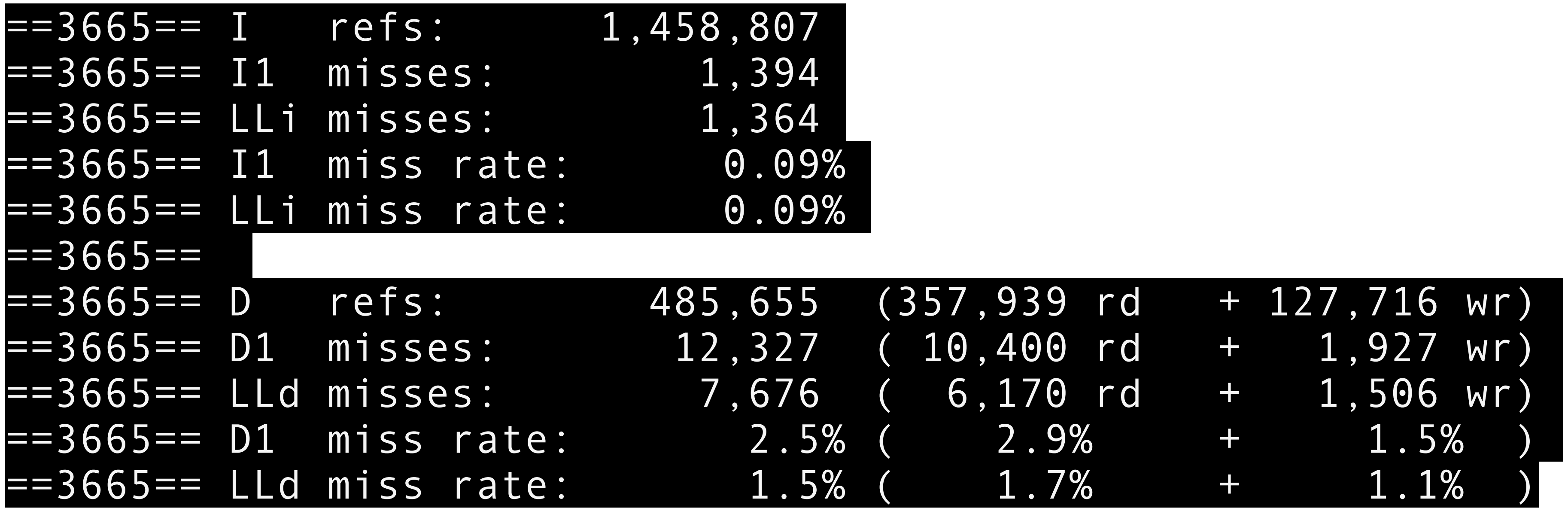
- Particle class (single particle)
  - Vectorial quantities like position, velocity, acceleration defined as **std::vector**
- ParticleBox class (collection of particles in a box)
  - Methods (for e.g. InternalForce, ComputeEnergy, Displacment) all returned **std::vector** as output

# Part 2 Implementation

- Particle class (single particle)
  - Vectorial quantities now defined as **std::array** replacing **std::vector**, since quantities are of a fixed size  (3 X 1)
- ParticleBox class (collection of particles in a box)
  - Methods mentioned previously now return void, but accept an additional input argument as a reference to **std::array** which stores the output

# Part 2 Implementation

- L1 miss rate was good in Part 1
- Part 2 - 2.5% miss rate

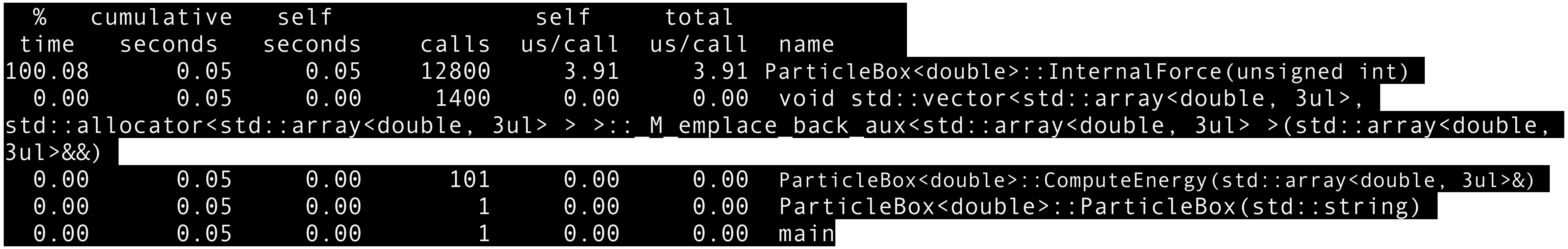


```
==3665== I   refs:      1,458,807
==3665== I1  misses:        1,394
==3665== LLi misses:        1,364
==3665== I1  miss rate:      0.09%
==3665== LLi miss rate:      0.09%
==3665==
==3665== D   refs:        485,655  (357,939 rd   + 127,716 wr)
==3665== D1  misses:       12,327  ( 10,400 rd   +   1,927 wr)
==3665== LLd misses:        7,676  (  6,170 rd   +   1,506 wr)
==3665== D1  miss rate:       2.5% (    2.9%     +     1.5%  )
==3665== LLd miss rate:       1.5% (    1.7%     +     1.1%  )
```

# Performance Comparison

- Performance comparison over **25 steps**
  - **Python** implementation: **7.1sec**
  - **C++** implementation: **0.038sec**
  - **187x** speedup!

# gprofile output (100 steps)



```
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  us/call  us/call  name    
100.08      0.05     0.05    12800     3.91     3.91 ParticleBox<double>::InternalForce(unsigned int)
  0.00      0.05     0.00     1400     0.00     0.00  void std::vector<std::array<double, 3ul>,
std::allocator<std::array<double, 3ul> > >::_M_emplace_back_aux<std::array<double, 3ul> >(std::array<double,
3ul>&&)
  0.00      0.05     0.00      101     0.00     0.00  ParticleBox<double>::ComputeEnergy(std::array<double, 3ul>&)
  0.00      0.05     0.00        1     0.00     0.00  ParticleBox<double>::ParticleBox(std::string)
  0.00      0.05     0.00        1     0.00     0.00  main
```

- As can be seen the initial expectation of **InternalForce** being bottleneck is now revealed
- This could possibly be due since this operation is not “vectorized”, i.e. the force contribution to particle *i* due to *j* is equal and opposite, but in the current implementation is computed twice!