HIGH PERFORMANCE COMPUTING for SCIENCE & ENGINEERING (HPCSE) I

TUTORIAL 02: CACHE USAGE OPTIMIZATION

Noah Baumann (noabauma@student.ethz.ch)

Computational **S**cience and **E**ngineering **Lab**
ETH Zürich

05.10.2022

Outline

2nd Class: Cache Size & Cache Speed

- Roofline model & Operational Intensity
- Matrix-Vector multiplication example
- HW1 Q5 Optional Exercise
- Particle Simulation Programming Exercise

II. Roofline Model

How to compute Operational Intensity from a given Kernel?

$$OI = \frac{W}{Q}\,[FLOP/byte]$$

W = amount of work / i.e floating point operations required

Q = memory transfer / i.e access from DRAM to lowest level cache

Example 1

```
float in[N], out[N];
for (int i=1; i<N-1; i++)
      out[i] = in[i-1]-2*in[i]+in[i+1]
```

float=4 byte, double=8 byte

A. Amount of flops W

For every `i`: `out[i] = in[i-1]-2*in[i]+in[i+1]` 3 flop

Loop over: `for (int i=1; i<N-1; i++)->` (N-2) repetitions

Total = 3(N-2) FLOPs

B. Memory acceses Q

Depends on cache size!

`out[I] = in[i-1]-2*in[i]+in[i+1]`

	For every `i`	Total Q	Total [bytes]	OI [flop/B]
1. No cache (we read directly from slow memory) every data accessed is counted	4	4(N-2)	4(N-2)x4	$\frac{3}{16}$
2. Perfect cache (infinite size cache) data is read & written ONLY ONCE	2	N+(N-2)	(2N-2)x4	$\approx \frac{3}{8}$

II. Roofline Model

How to compute Operational Intensity?

$$OI = \frac{W}{Q}\,[FLOP/byte]$$

Example 2 Matrix multiplication (Naive)

```
double A[N,N], B[N,N], C[N,N];
for (int j=0; j<N; j++)
    for (int i=0; i<N; i++)
        for (int k=0; k<N; k++)
            C[i,j] = C[i,j] + A[i,k]*B[k,j]
```

= 1 MUL + 1 ADD

A. Amount of flops W ? For every `i,j`: `C[i,j] = C[i,j] + A[i,k]*B[k,j]` 2 FLOPs

Loop over N*N*N —> Total = $2N^3$ FLOPs

B. Memory acceses Q ? For every `i,j`: `C[i,j] = C[i,j] + A[i,k]*B[k,j]`

	For every `i,j`	Total Q	Total [bytes]	OI [flop/B]
1. Perfect cache (small N - fits in cache) →	3 read	$3N^2$	$3N^2 \times 8$	$\frac{2N^3}{24N^2} = \mathcal{O}(N)$
2. More realistic cache →	For every `C[i,j]` element: – read a row of `A (N)` = 2N read – read a column of `B (N)` – `read & write 1 element C` = 2N + 1	$(2N+1)N^2$	$(2N+1)N^2 \times 8$	$\frac{2N^3}{8(2N^3+N^2)} \approx \frac{1}{8}$

"ZEN 3" OVERVIEW

2 THREADS PER CORE (SMT)

STATE-OF-THE-ART BRANCH PREDICTOR

CACHES
- I-cache 32k, 8-way
- Op-cache, 4K instructions
- D-cache 32k, 8-way
- L2 cache 512k, 8-way

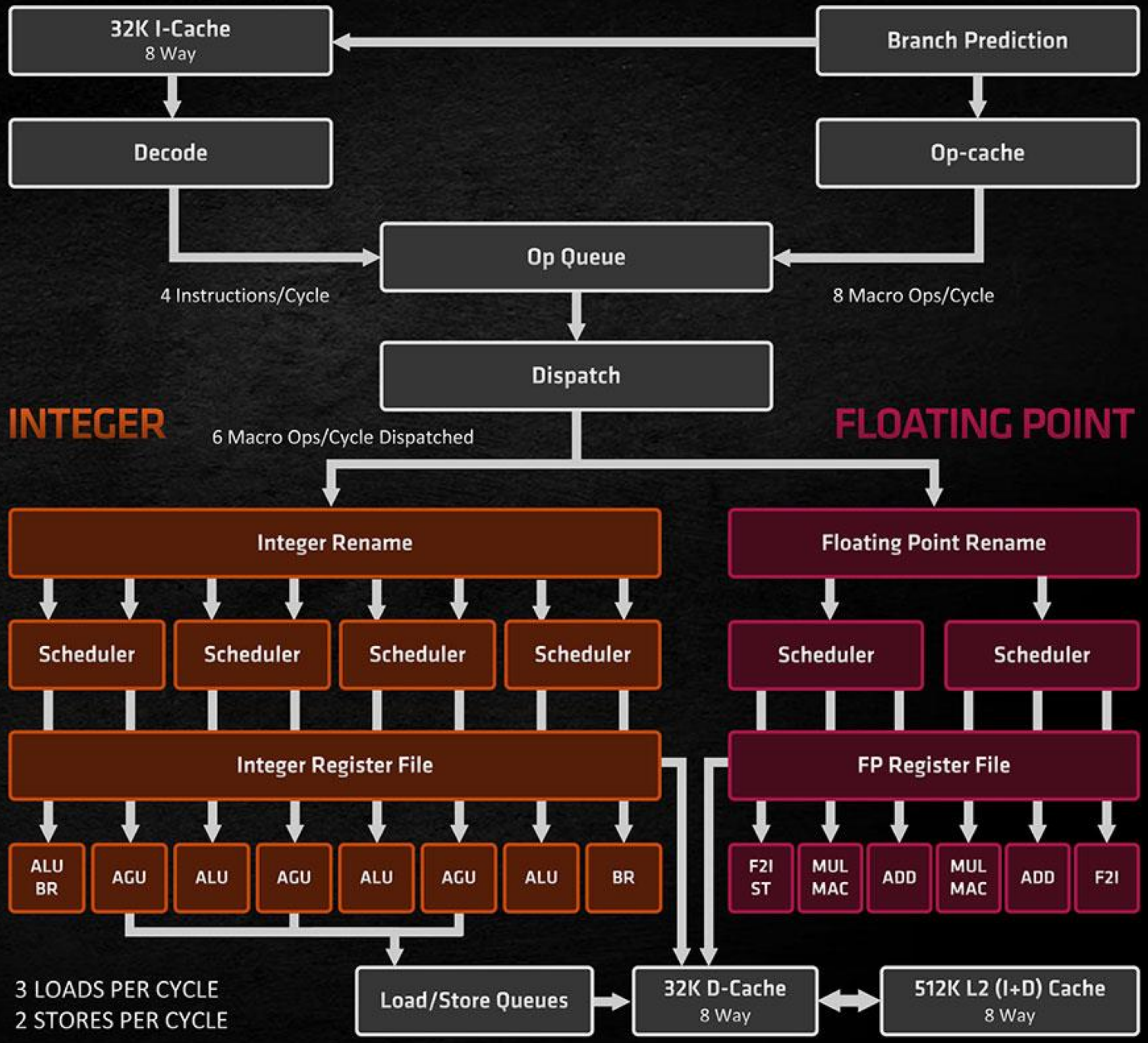
DECODE
- 4 instructions / cycle from decode or 8 ops from Op-cache
- 6 ops / cycle dispatched to Integer or Floating Point

EXECUTION CAPABILITIES
- 4 integer units
- Dedicated branch and store data units
- 3 address generations per cycle

3 MEMORY OPS PER CYCLE
- Max 2 can be stores

TLBs
- L1 64 entries I & D, all page sizes
- L2 512 I, 2K D, everything but 1G

TWO 256-BIT FP MULTIPLY ACCUMULATE / CYCLE



Remember!

Homework 01:

We consider bith read + write & only write memory accesses.

But you have to justify your solution!

Roofline model:

- Great Question:

Do we change the Roofline model if we use doubles instead of floats?

Depends:
If vectorization is allowed and you only consider doubles: yes!
Else: no

AVX512: 512-bit -> 8 doubles or 16 floats
AVX2: 256-bit -> 4 doubles or 8 floats

Exercise

Matrix-vector multiplication coding example

Cache Size & Cache Speed

From HW1, Optional Question 5

Iterate:

```
for(i : M)
  k = a[k]
```

Variant 1: $a_k = random\ once{-}cycle\ permutation$

index k	0	1	2	3	4	5	…	N-1
array element a_k	2	9	5	15	71	91	…	

Variant 2: $a_k = (k+1)\%N$

k	0	1	2	3	4	5	…	N-1
a_k	1	2	3	4	5	6	…	

Variant 3: $a_k = \left(k + \frac{cache\ line\ size}{sizeof(int)}\right)\%N$

k	0	1	2	3	4	5	…	N-1
a_k	16	17	18	19	20	21	…	

Cache Size & Cache Speed

Variant 1: $a_k = random\ once{-}cycle\ permutation$

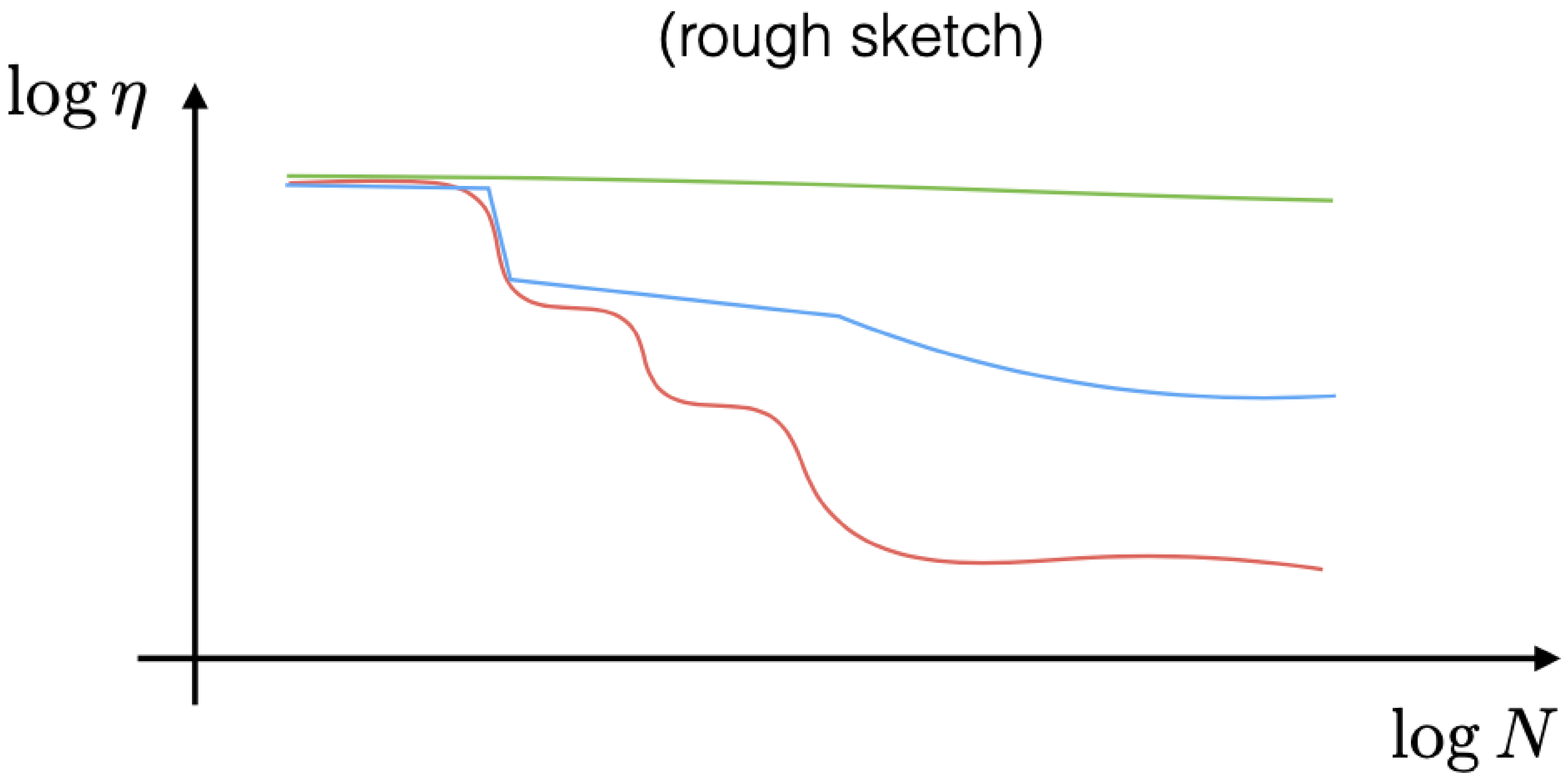
Variant 2: $a_k = (k+1)\%N$

Variant 3: $a_k = \left(k + \frac{cache\ line\ size}{sizeof(int)}\right)\%N$

Iterate:

```
for(i : M)
  k = a[k]
```

k	0	1	2	3	4	5	…									N-1
a_k	2	9	5	15	71	91	…									



Question:

$$\eta = \frac{M}{total\ time} = ?$$

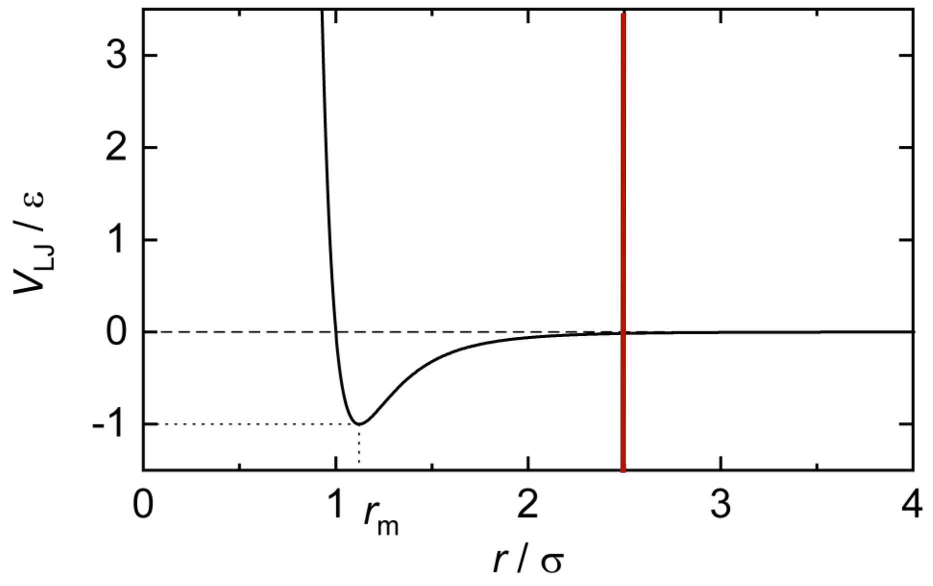
$$for N \to large$$

Particle Simulation

Example: Particles in a “Lennard-Jones” Potential

physical constants

$$U(r_{ij}) = 4\epsilon[(\frac{\sigma}{r_{ij}})^{12} - (\frac{\sigma}{r_{ij}})^{6}]$$



$$r_c = 2.5\sigma$$

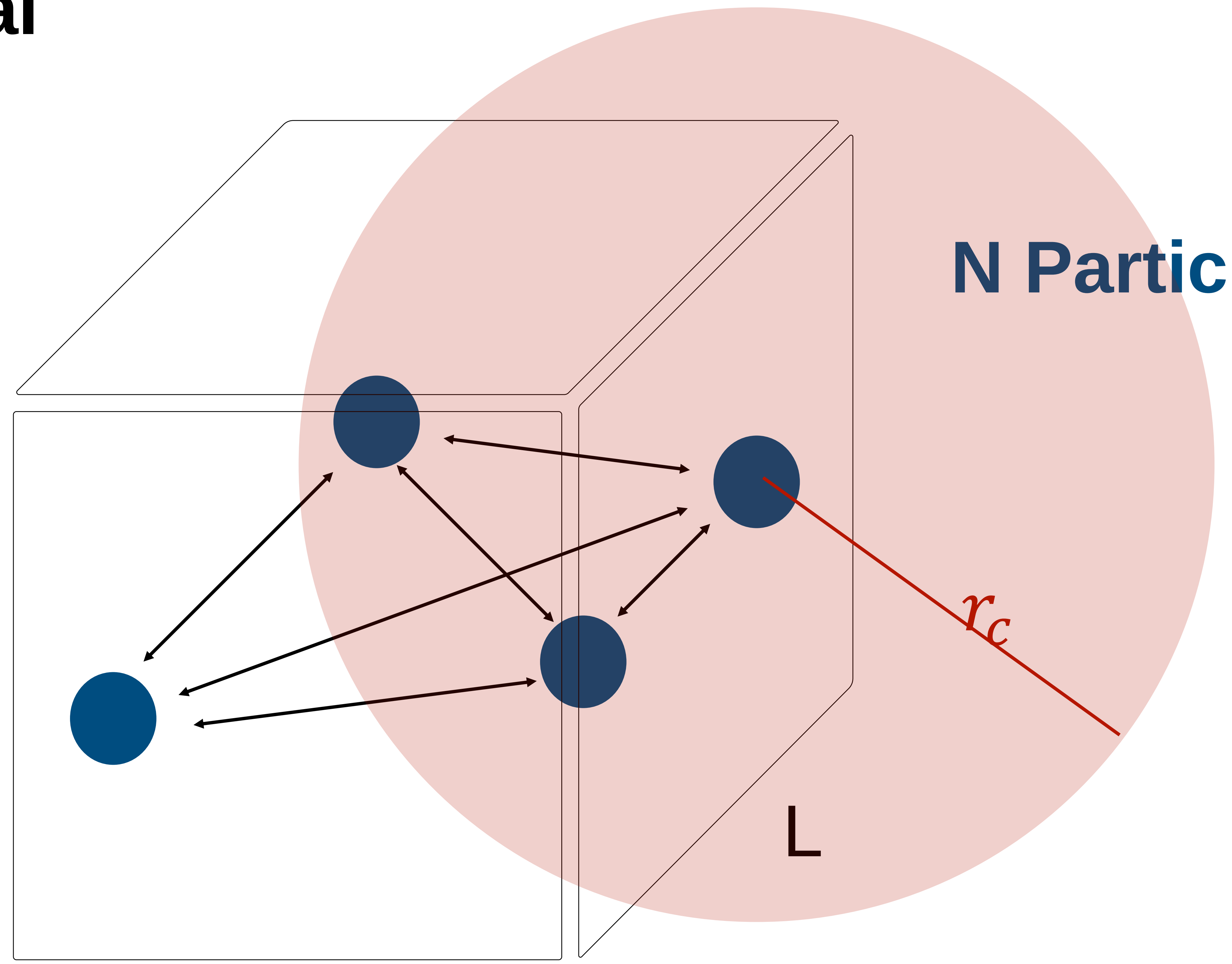
$$F(r) = -\frac{\partial U(r)}{\partial r}$$

$$U(r_{ij}) \approx 0 \, for \, r_{ij} > r_c$$

$$\vec{f}_{ij} = -\frac{24\epsilon}{r_{ij}}(2(\frac{\sigma}{r_{ij}})^{12} - (\frac{\sigma}{r_{ij}})^{6})\vec{r}_{ij}$$

$$\vec{f}_i = \sum_{i \neq j}^{N} \vec{f}_{ij} \qquad \vec{a}_i = \frac{\vec{f}_j}{m_i}$$

$$\vec{v}_{i,t+1} = \vec{v}_{i,t+1} + a_i\delta t \qquad \vec{x}_{i,t+1} = \vec{x}_{i,t+1} + v_{i,t+1}\delta t$$



How many force-terms to compute?

$$\frac{N^2 - N}{2} = \mathcal{O}(N^2) \qquad \text{Memory } \mathcal{O}(N)$$

Pseudocode “Naive” Particle Simulation

Simulation Loops

```
simulateParticles( std::vector<particle>& particles )
{
  for(i = 0; i < N; i++)
    for(j = i+1; j < N; j++)
       checkDistance(particle i, particle j)
       calculateForce(particle i, particle j)
       updateForce(particle i)
       updateForce(particle j)

  for(auto& p : particles)
    updateVelocity(p)
    move(p)
}
```

80 Bytes Object

```
struct particle {
  double x;
  double y;
  double z;
  double vx;
  double vy;
  double vz;
  double fx;
  double fy;
  double fz;
  double m;
 };
```

spatial coordinates

velocities

placeholder for forces

mass

How can we optimize data movement?

Potential Improvements

- E.g. we have 10 particles
- Calculate the forces for N(N-1)/2 cases

Particle j

Particle i

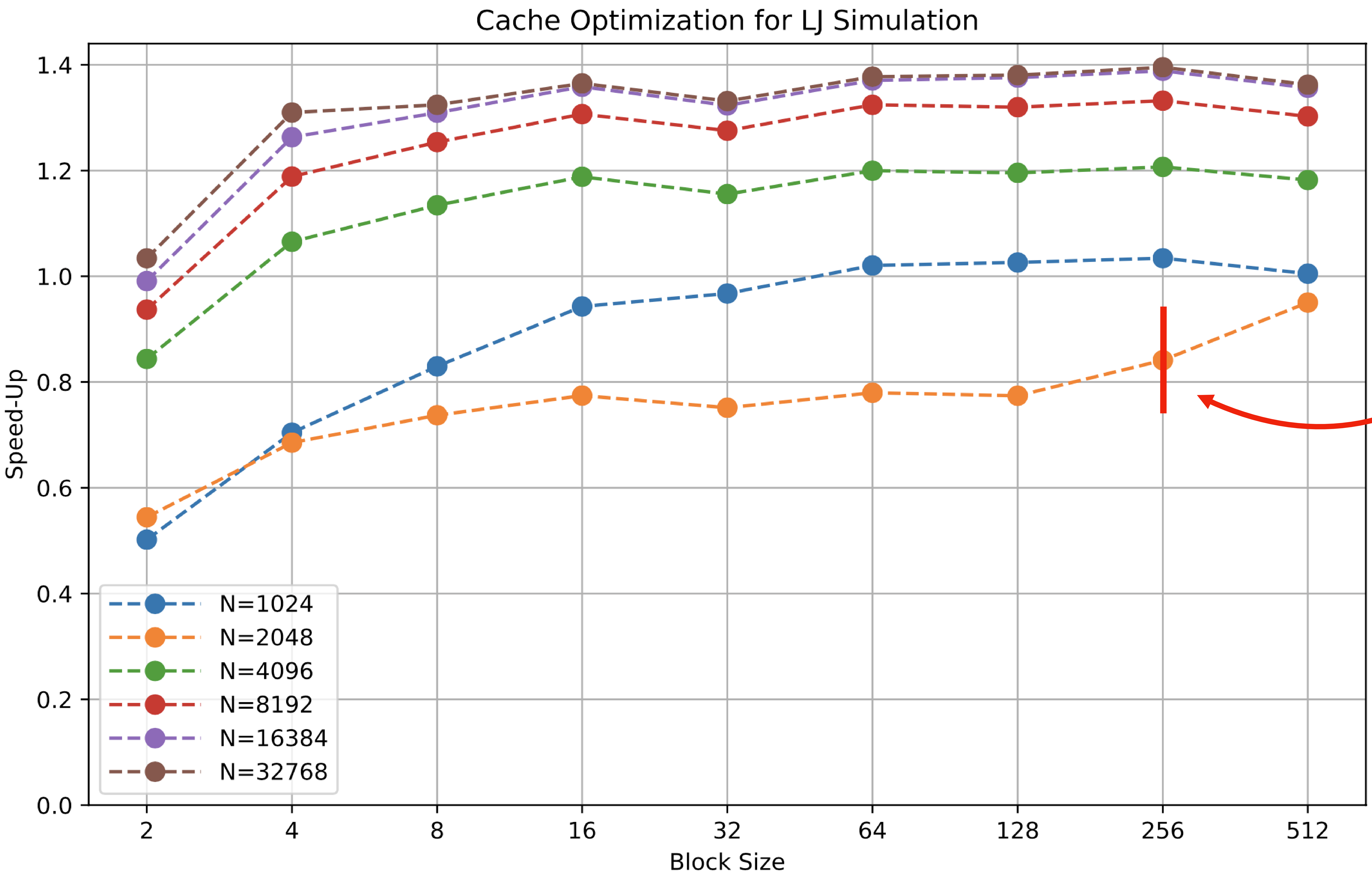
i,j=0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
		1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
			2,3	2,4	2,5	2,6	2,7	2,8	2,9
				3,4	3,5	3,6	3,7	3,8	3,9
					4,5	4,6	4,7	4,8	4,9
						5,6	5,7	5,8	5,9
							6,7	6,8	6,9
								7,8	7,9
									8,9

Potential Improvements

1. **Iterating over particles in blocks (similar idea as in matrix-matrix multiplication)**
2. **Instead of updating the position of the particles in a separate loop, we update the particle the last time its force is updated**
3. Remove force placeholder from particle object and directly update the velocity in the inner-most loop
4. Change the structure of the code to structure of array (SoA) instead of AoS, such that we can avoid unnecessary data movement for velocity and mass if cut-off radius breached

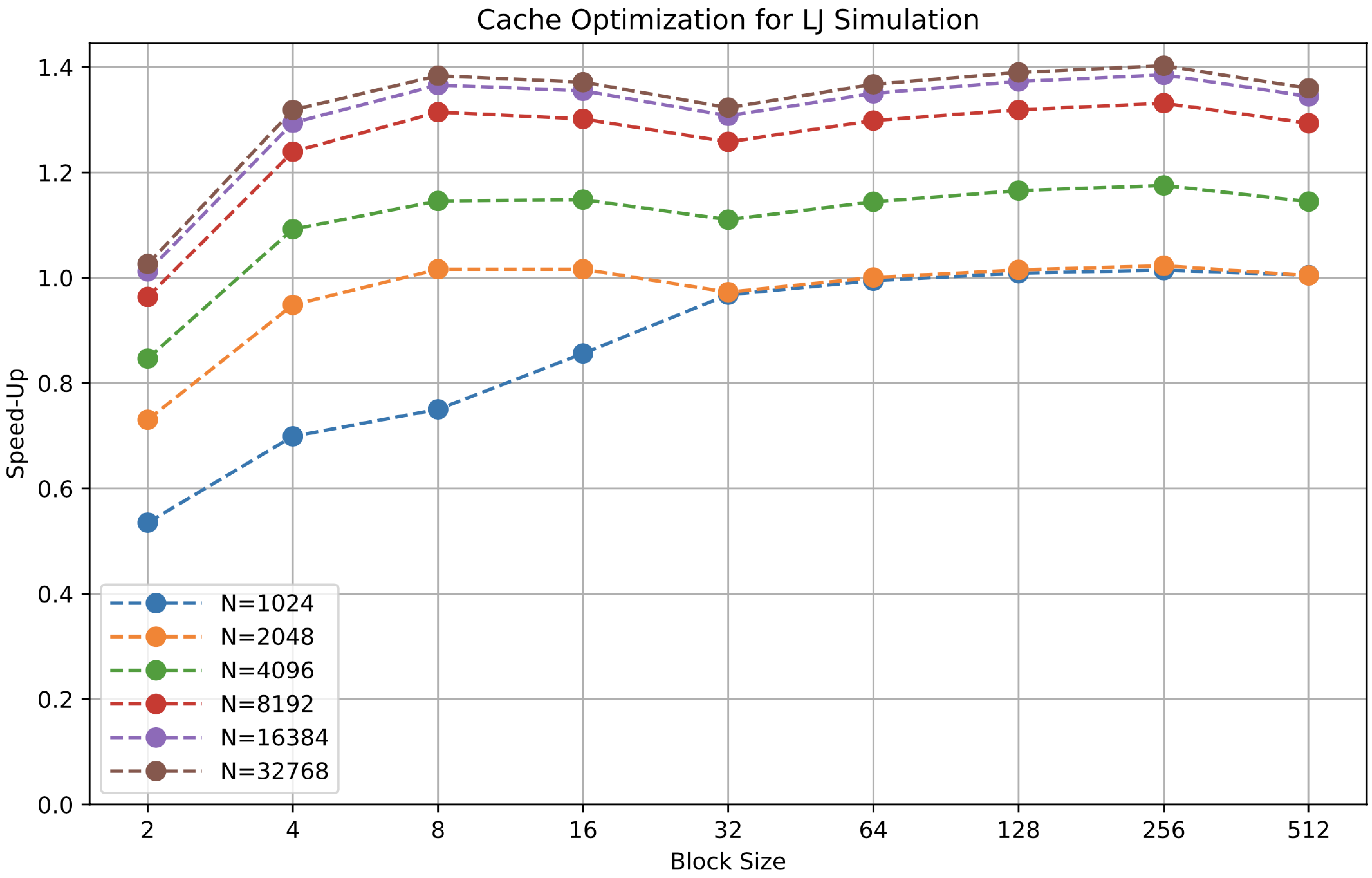
not today

(1) Improvement Blocking



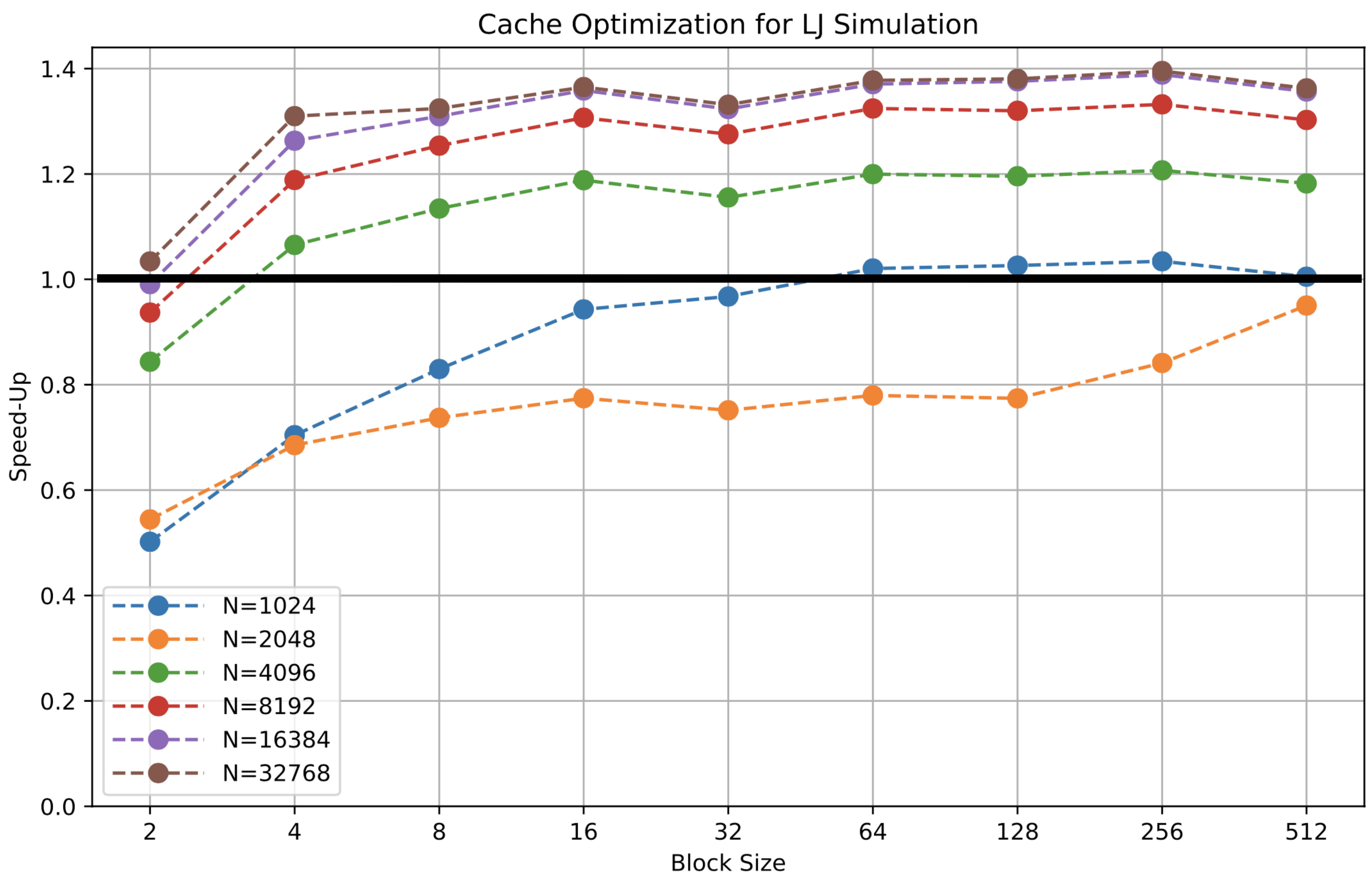
error bars missing!!

(2) Improvement Particle Move

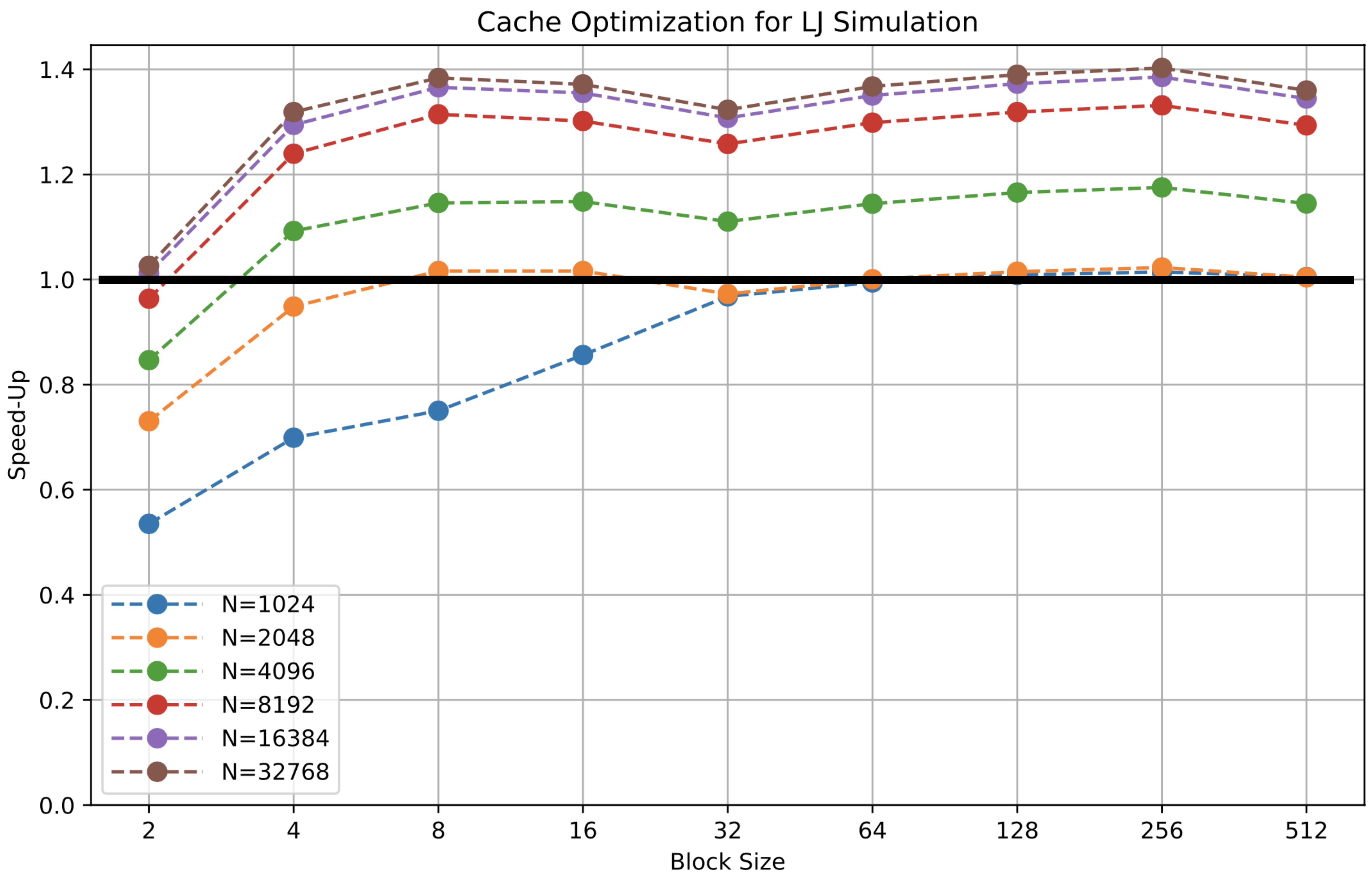


Comparison

(1) Blocking



(2) Particle Move



For which block size did you expect the most speed-up?

get_sysinfo

L1 Cache 4000 Bytes

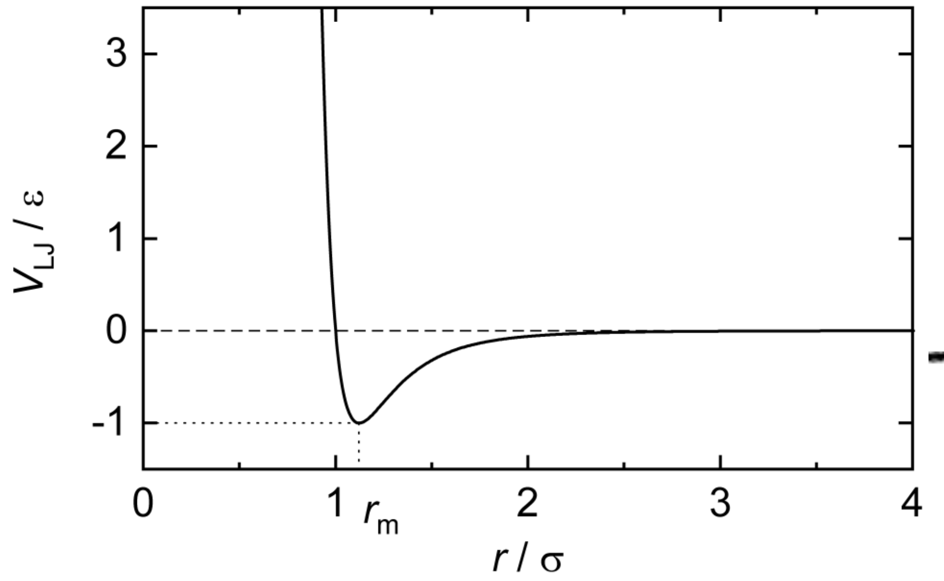
80 Bytes per Particle

→ 50 Particles fit in L1 →

Processing 2 blocks in the inner loops

Optimal Block Size 25 Particles?

Simulation



Simulation time 0.1 sec (100 frames each 1ms)

Increment for integrator

$$\delta t = 0.001$$

Physical constants for LJ-Potential

$$\sigma = 0.2$$

$$\epsilon = 0.001$$

Random Particle Initialization

$$N = 512$$

$$\rho = \frac{L^3}{N} = 1$$

$$m_i \sim \mathcal{N}(10,1)$$

$$v_i^{(0)} \sim \mathcal{N}(0,1)$$

$$(x,y,z)_i^{(0)} \sim \mathcal{U}_{(0,L)}^3$$

$$\rightarrow v_{avg} \approx 2m/s$$