NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

# Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2026

# Assignment 2 – Task 1 a)

## Peak Performance

- Clock frequency of 2.05 GHz, but only 0.9 * 2.05 GHz for “hot code”
- 126 cores
- ShmAVX-1024 instruction set (1024-bit wide SIMD registers); each core is capable of retiring one full-width double-precision FMA instruction per cycle

$$P_{peak} = n_{cores} * n_{super}^{FP} * n_{FMA} * n_{SIMD} * f$$

# Assignment 2 – Task 1 a)

## Peak Performance

- Clock frequency of 2.05 GHz, but only 0.9 * 2.05 GHz for “hot code”
- 126 cores
- ShmAVX-1024 instruction set (1024-bit wide SIMD registers); each core is capable of retiring one full-width double-precision FMA instruction per cycle

$$P_{peak} = n_{cores} * n_{super}^{FP} * n_{FMA} * n_{SIMD} * f$$

$$P_{peak} = 126 * 1 \left[\frac{instr}{cy}\right] * 2 * 16 \left[\frac{flops}{instr}\right] * 1.845 \left[\frac{Gcy}{s}\right]$$

$$P_{peak} = 126 * 1 \left[\frac{instr}{cy}\right] * 2 * 16 \left[\frac{flops}{instr}\right] * 1.845 \left[\frac{Gcy}{s}\right] \approx 7.439 \left[\frac{Tflops}{s}\right]$$

# Assignment 2 – Task 1 a)

## Peak Performance

- Clock frequency of 2.05 GHz, but only 0.9 * 2.05 GHz for “hot code”
- 126 cores
- ShmAVX-1024 instruction set (1024-bit wide SIMD registers); each core is capable of retiring one full-width double-precision FMA instruction per cycle

$$P_{peak} = n_{cores} * n_{super}^{FP} * n_{FMA} * n_{SIMD} * f$$

$$P_{peak} = 126 * 1 \left[\frac{instr}{cy}\right] * 2 * 16 \left[\frac{flops}{instr}\right] * 1.845 \left[\frac{Gcy}{s}\right]$$

$$P_{peak} = 126 * 1 \left[\frac{\cancel{instr}}{cy}\right] * 2 * 16 \left[\frac{flops}{\cancel{instr}}\right] * 1.845 \left[\frac{Gcy}{s}\right] \approx 7.439 \left[\frac{Tflops}{s}\right]$$

# Assignment 2 – Task 1 a)

## Peak Performance

- Clock frequency of 2.05 GHz, but only 0.9 * 2.05 GHz for “hot code”
- 126 cores
- ShmAVX-1024 instruction set (1024-bit wide SIMD registers); each core is capable of retiring one full-width double-precision FMA instruction per cycle

$$P_{peak} = n_{cores} * n_{super}^{FP} * n_{FMA} * n_{SIMD} * f$$

$$P_{peak} = 126 * 1 \left[\frac{instr}{cy}\right] * 2 * 16 \left[\frac{flops}{instr}\right] * 1.845 \left[\frac{Gcy}{s}\right]$$

$$P_{peak} = 126 * 1 \left[\frac{\cancel{instr}}{\cancel{cy}}\right] * 2 * 16 \left[\frac{flops}{\cancel{instr}}\right] * 1.845 \left[\frac{\cancel{Gcy}}{s}\right] \approx 7.439 \left[\frac{Tflops}{s}\right]$$

# Assignment 2 – Task 1 a)

## Peak Performance

- Clock frequency of 2.05 GHz, but only 0.9 * 2.05 GHz for “hot code”
- 126 cores
- ShmAVX-1024 instruction set (1024-bit wide SIMD registers); each core is capable of retiring one full-width double-precision FMA instruction per cycle

$$P_{peak} = n_{cores} * n_{super}^{FP} * n_{FMA} * n_{SIMD} * f$$

When in doubt, think **units**!

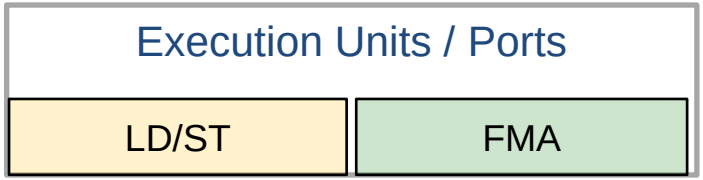
$$P_{peak} = 126 * 1 \left[\frac{instr}{cy}\right] * 2 * 16 \left[\frac{flops}{instr}\right] * 1.845 \left[\frac{Gcy}{s}\right]$$

$$P_{peak} = 126 * 1 \left[\frac{\cancel{instr}}{\cancel{cy}}\right] * 2 * 16 \left[\frac{flops}{\cancel{instr}}\right] * 1.845 \left[\frac{\cancel{Gcy}}{s}\right] \approx 7.439 \left[\frac{Tflops}{s}\right]$$

# Assignment 2 – Task 1 b)

## Execution Units

```
double a[...],b[...];
double s=0.0, t=1.234;
// a[] and b[] contain sensible data
for(int i=0; i<N; ++i) {
  s += a[i]*a[i];
  b[i] *= t;
}
```

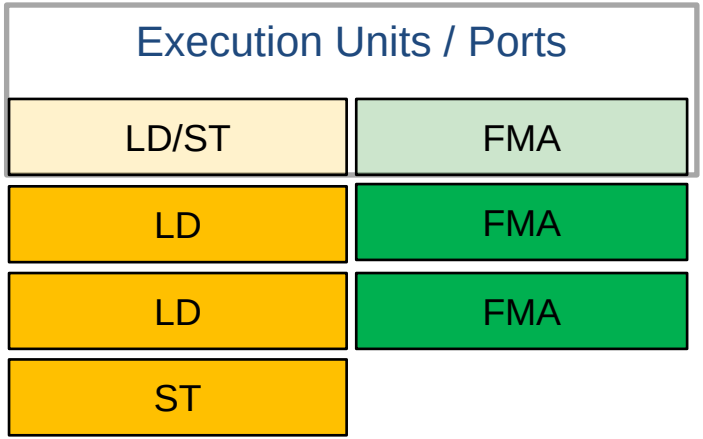



- Capability of executing 1 FMA, and 1 LOAD or 1 STORE instruction per cycle
- 1024-bit wide SIMD capability, for all instructions

# Assignment 2 – Task 1 b)

## Execution Units

```
double a[...],b[...];
double s=0.0, t=1.234;
// a[] and b[] contain sensible data
for(int i=0; i<N; ++i) {
  s += a[i]*a[i];
  b[i] *= t;
}
```



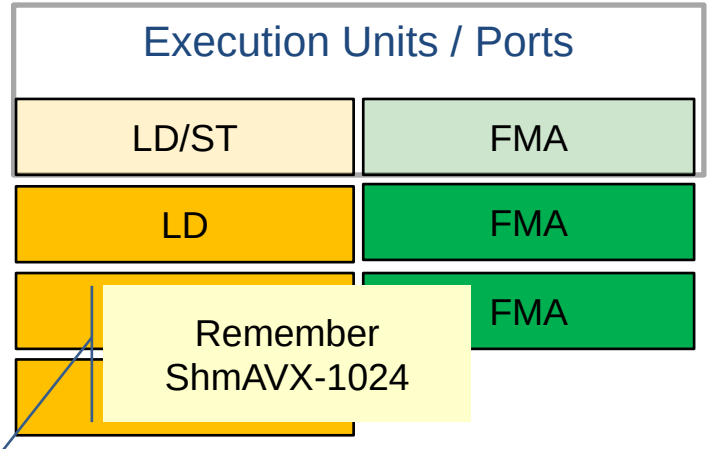

- Suppose the MULT uses the FMA port and adds “zero”
- Two FMAs, 2 LOAD, and 1 STORE

$$P_{\max} = 126 * \frac{1[iter]}{3[cy]} * 1.845 \left[\frac{Gcy}{s}\right] = 126 * \frac{3*16[flops]}{3[cy]} * 1.845 \left[\frac{Gcy}{s}\right] = 3.720 \left[\frac{Gflops}{s}\right]$$

# Assignment 2 – Task 1 b)

## Execution Units

```
double a[...],b[...];
double s=0.0, t=1.234;
// a[] and b[] contain sensible data
for(int i=0; i<N; ++i) {
  s += a[i]*a[i];
  b[i] *= t;
}
```



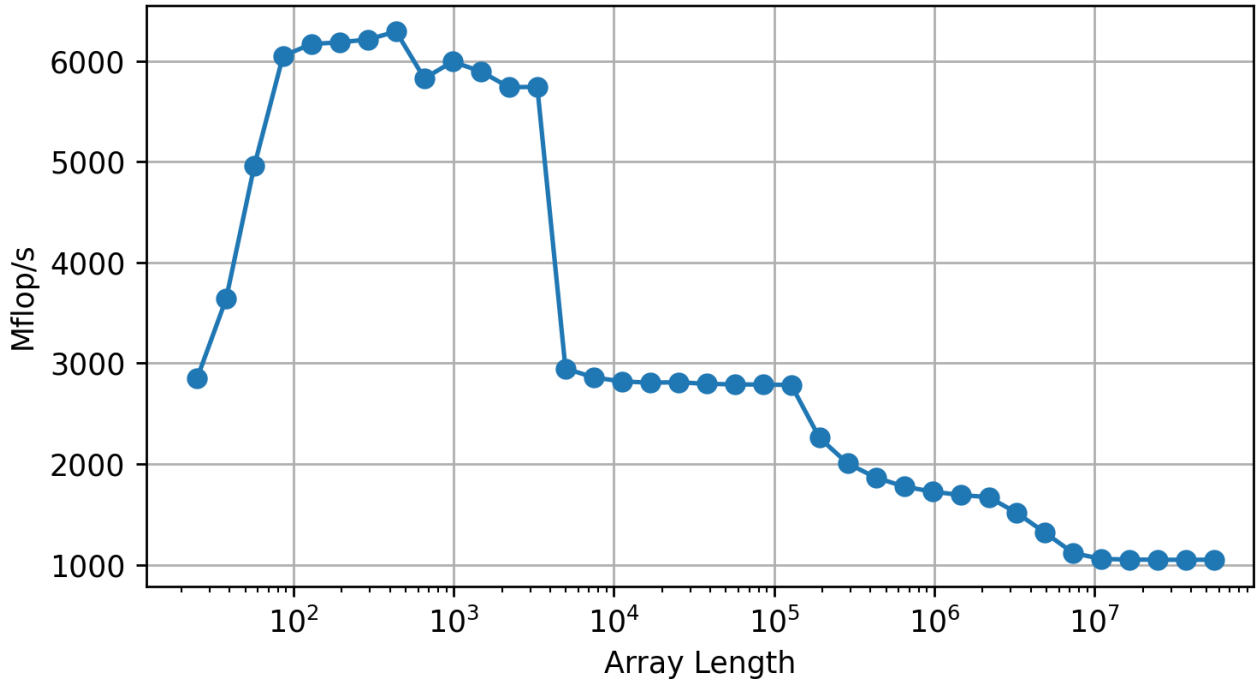

- Suppose the MULT uses the FMA port and adds “zero”
- Two FMAs, 2 LOAD, and 1 STORE

$$P_{\max} = 126 * \frac{1[iter]}{3[cy]} * 1.845 \left[\frac{Gcy}{s}\right] = 126 * \frac{3*16[flops]}{3[cy]} * 1.845 \left[\frac{Gcy}{s}\right] = 3.720 \left[\frac{Gflops}{s}\right]$$

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

a) Accumulate: `a[i] = a[i] + b[i]`




icx compiler
-O3 -xHost -fno-alias

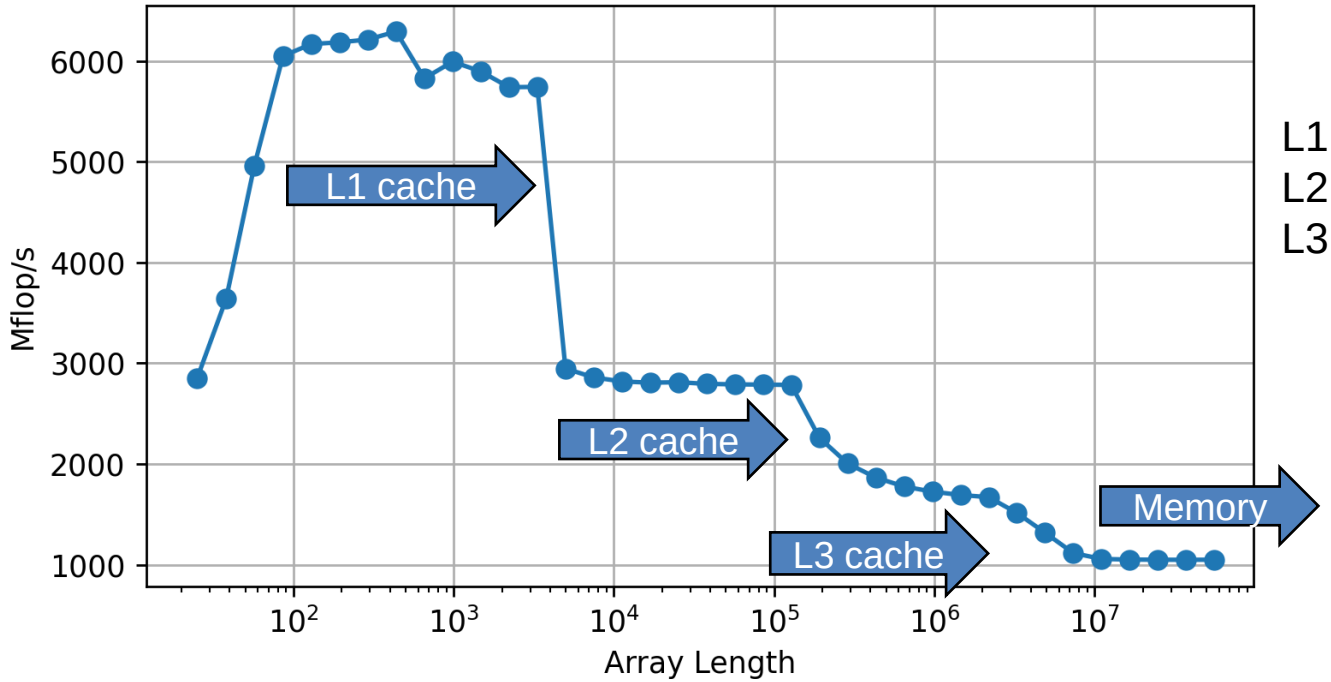
L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

a) Accumulate: `a[i] = a[i] + b[i]`

icx compiler
-O3 -xHost -fno-alias

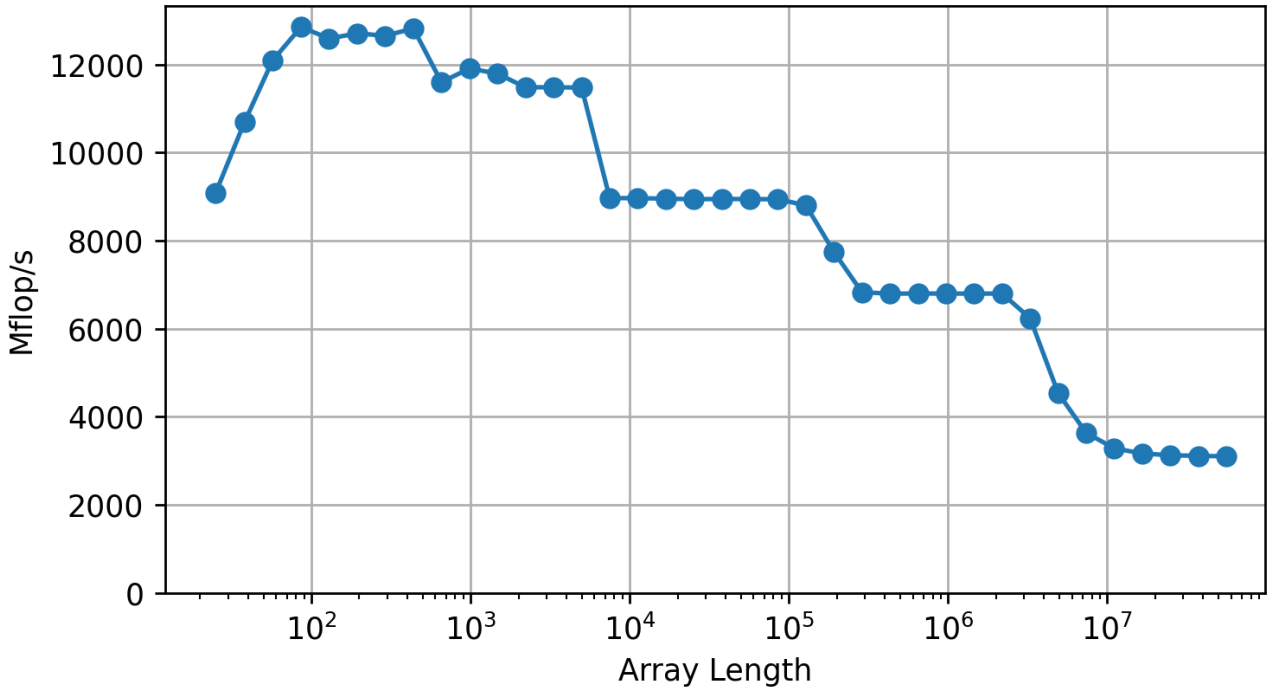



L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

a) Update: `a[i] = s * a[i] + t`



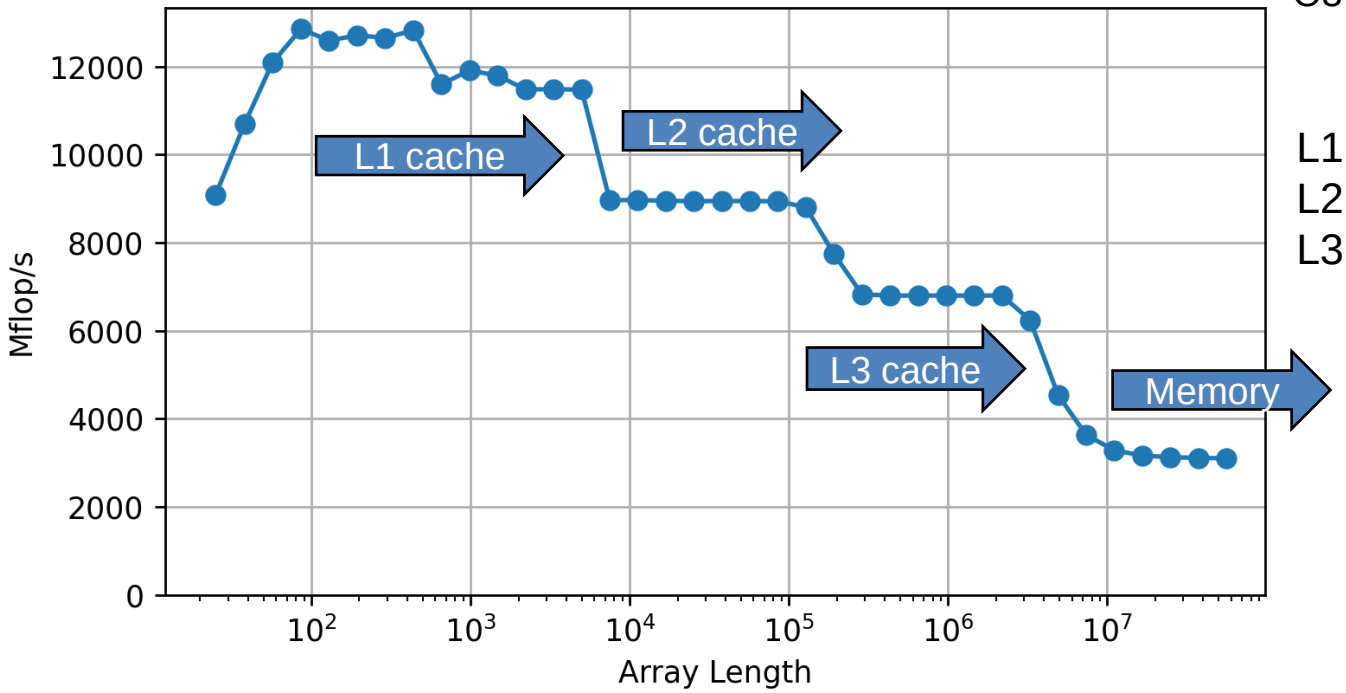

icx compiler
-O3 -xHost -fno-alias

L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

a) Update: `a[i] = s * a[i] + t`



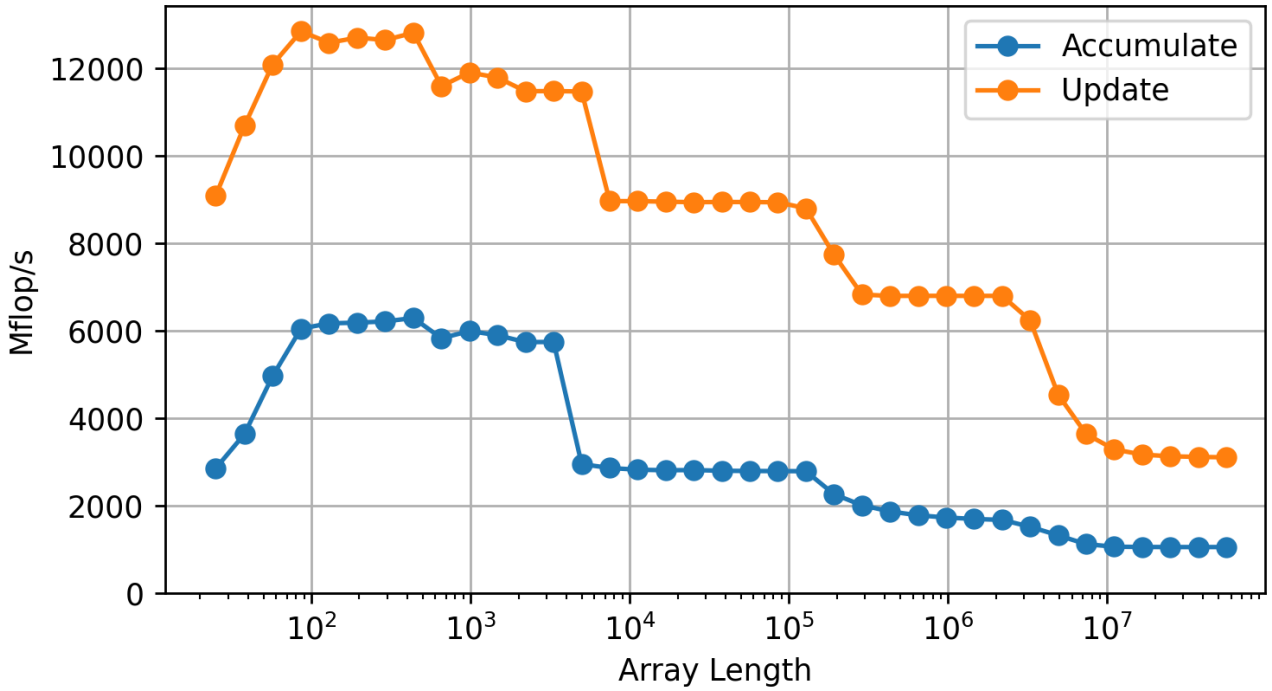

icx compiler
-O3 -xHost -fno-alias

L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

d) Differences

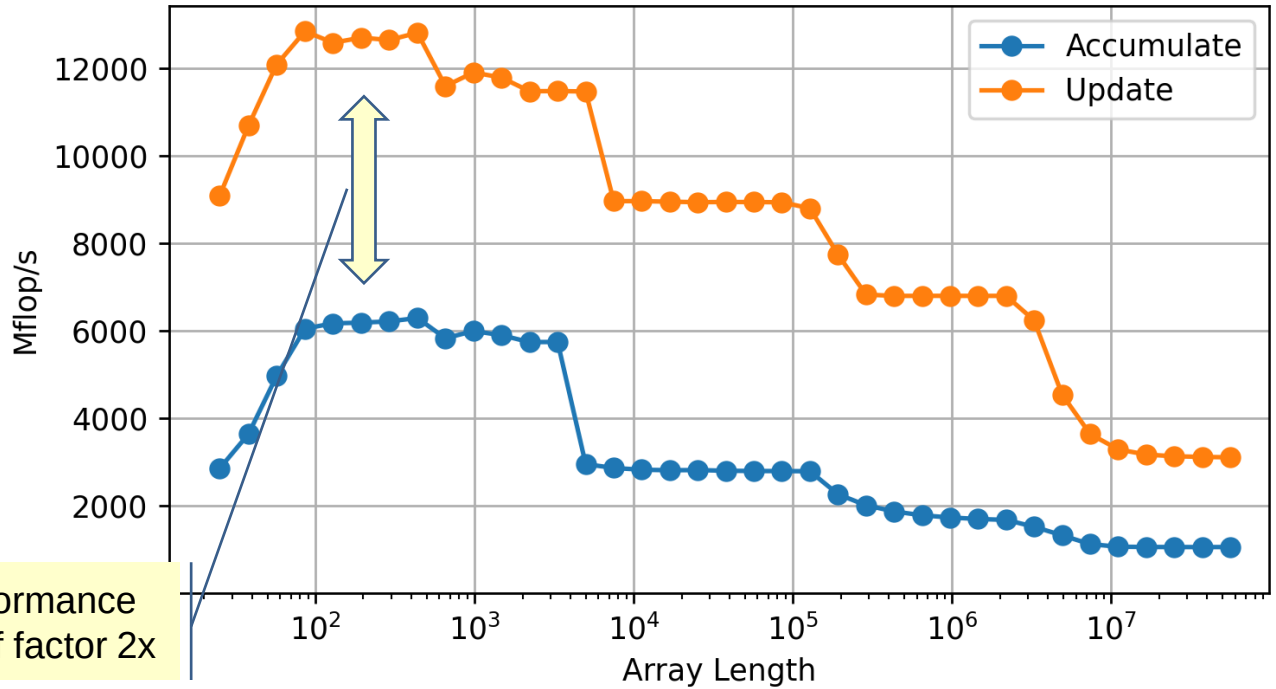



L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

d) Differences



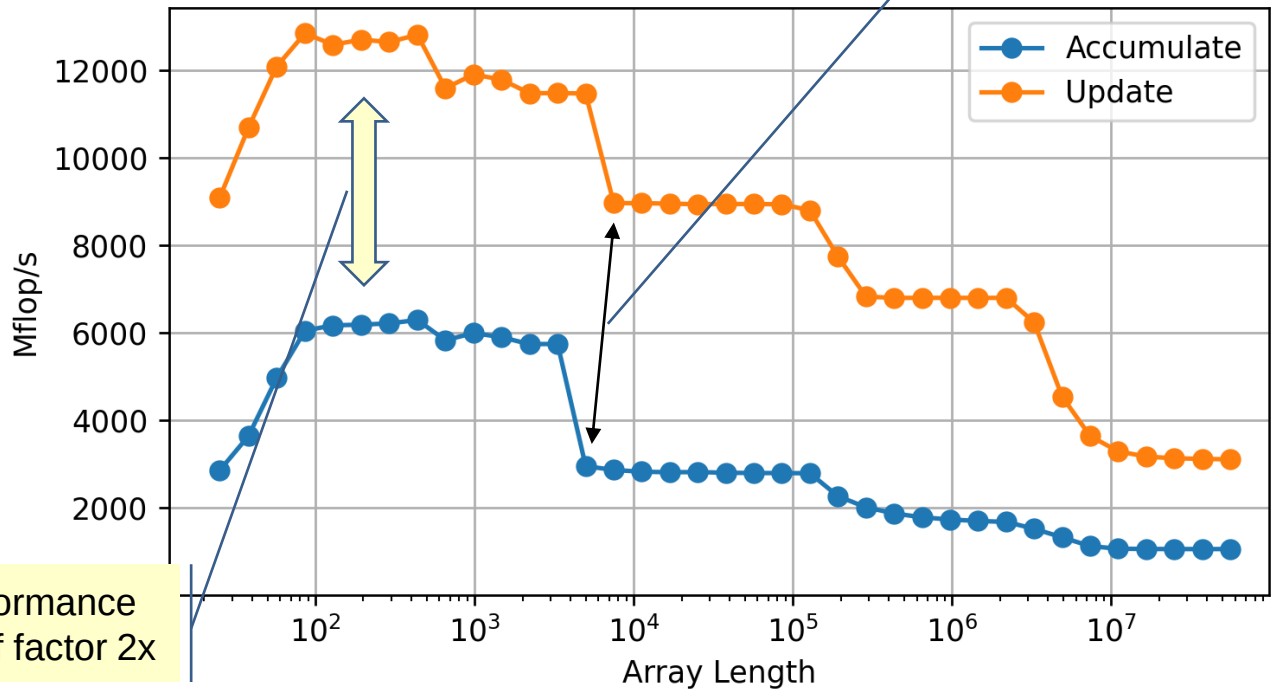

Performance diff. of factor 2x

L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB

# Assignment 2 – Task 2

## Loop Kernel Benchmarking

d) Differences




Drops appear 2x later. Why?

Performance diff. of factor 2x

L1 cache: 48 KiB
L2 cache: 1.25 MiB
L3 cache: 54 MiB