Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2025

Assignment 4 – Task 1

Peak performance and bandwidth

a) 48 (cores) x 8 (AVX-512) x 2 (FMA/cy) x 2 (flops/FMA) x 2.0 (Gcy/s) = 3.072 Tflop/s

b) 12 (channels) x 8 (byte/cy) x 3.2 (Gcy/s) = 307.2 GB/s

c) Amount of floating-point computations: 2 x 10^9 flops
Amount of data transfer: 4 x 8 x 10^9 byte

```
for(int i=0; i<10⁹; ++i)
  a[i] = b[i] + s * c[i];
```

Time to do the flops at peak flop rate: $T_{\mathrm{flops}} = \frac{2\times 10^9 \text{ flops}}{3.072 \text{ Tflops/s}} = 651\ \mu\mathrm{s}$

Time to transfer the data at peak BW: $T_{\mathrm{BW}} = \frac{32\times 10^9 \text{ byte}}{307.2 \text{ Gbyte/s}} = 104 \text{ ms}$

→ **The bottleneck is clearly the data transfer**

→ **In-core model not helpful (LD/ST-bound), better: P_{max}**

Assignment 4 – Task 2

```
for(j=0; j<N; ++j)
  for(i=0; i<N; ++i)
    a[j][i] = x[i] * y[j];
```

Outer product

a) Locality of access (**`a[][]`** in memory, **`N`** is large):

`a[][]` purely spatial locality (each element accessed exactly once in linear order)

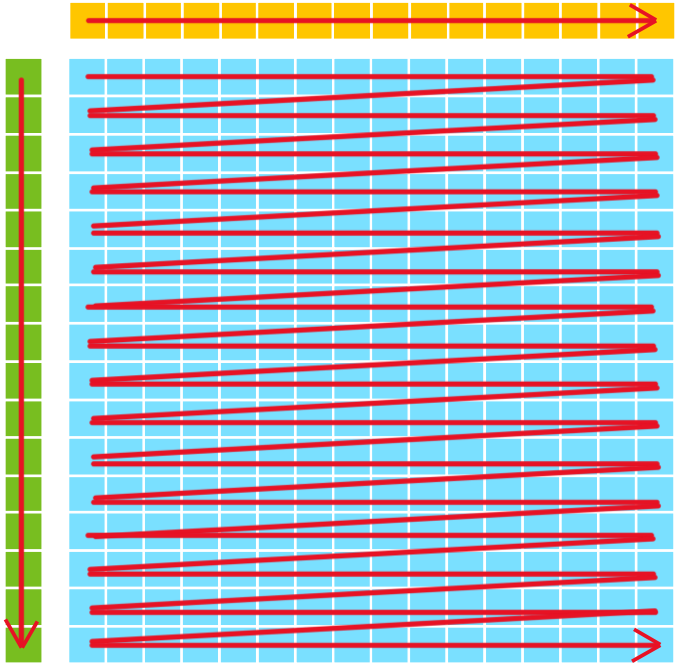
`x[]` spatial and temporal locality:
all elements accessed in linear order,
each element reused N-1 times if array x[ ] fits into some cache

`y[]` spatial and temporal locality:
all elements accessed in linear order,
each element reused N-1 times from L1 or register

Assignment 4 – Task 2

Outer product

b)

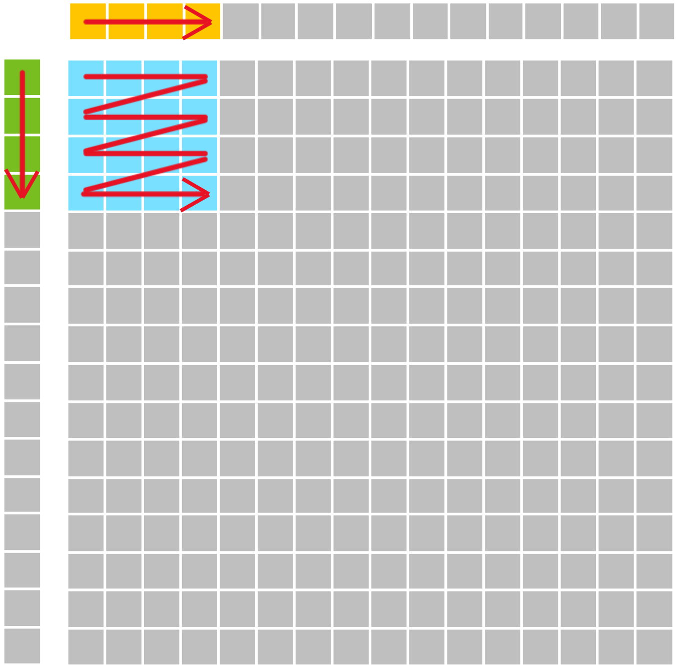


```
for(j=0; j<N; ++j)
  for(i=0; i<N; ++i)
    a[j][i] = x[i] * y[j];
```

Assignment 4 – Task 2

Outer product

b)



```
for(jb=0; jb<N; jb+=bsize)
  for(ib=0; ib<N; ib+=bsize)
    for(j=jb; j<min(N,jb+bsize); ++j)
      for(i=ib; i<min(N,ib+bsize); ++i)
        a[j][i] = x[i] * y[j];
```

Tile size: $bsize^2$ x 8B

With 1 MiB L2 cache:

$$8 \times \mathrm{bsize}^2 < 2^{20}$$

→ $\mathrm{bsize} < 362$

Assignment 4 – Task 2

Outer product

c) $N = 10^4$ ➔ size of `x[]`: L2 > 80 kB > L1 ➔ `x[]` stays in L2

```
for(jb=0; jb<N; jb+=bsize)
  for(ib=0; ib<N; ib+=bsize)
    for(j=jb; j<min(N,jb+bsize); ++j)
      for(i=ib; i<min(N,ib+bsize); ++i)
        a[j][i] = x[i] * y[j];
```

	B_C in Memory	B_C in L3	B_C in L2	B_C in L1
untiled	16 B/it	16 B/it	24 B/it	32 (24) B/it
tiled	16 B/it	16 B/it	16 B/it	32 (24) B/it

➔ tiling improves performance due to **the reuse of `x[]`**

Assignment 4 – Task 3

Optimizing for code balance

a)

```
float a[],b[],c[],z[];

for(i=0; i<N; ++i) {
  a[i] += b[i] * c[i];
}
for(i=0; i<N; ++i) {
  z[i] = a[i] * 0.5f;
}
```

Loop fusion

```
float a[],b[],c[],z[];

for(i=0; i<N; ++i) {
  a[i] += b[i] * c[i];
  z[i] = a[i] * 0.5f;
}
```

$$B_c = \frac{16+12}{3} \frac{\mathrm{B}}{\mathrm{Flop}} = \mathbf{9.33\ B/Flop}$$

$$B_c = \frac{16+8}{3} \frac{\mathrm{B}}{\mathrm{Flop}} = \mathbf{8\ B/Flop}$$

Assignment 4 – Task 3

Optimizing for code balance

b)

- L1-L2: 64 B/cy half-duplex → 1 cy/CL
- L2-L3: 16 B/cy half-duplex → 4 cy/CL
- Mem BW: 110 GB/s, half-duplex → 1.28 cy/CL
- 2.2 GHz clock freq

```
float a[],b[],c[],z[];

for(i=0; i<N; ++i) {
  a[i] += b[i] * c[i];
  z[i] = a[i] * 0.5f;
}
```

L1-L2: **6 cy**
L2-L3: **24 cy**
Memory: **7.7 cy**