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 2024

Assignment 4 – Task 1

Peak resource delivery

a) 64 (cores) x 4 (AVX) x 2 (flops/FMA) x 2 (FMA/cy) x 2.4 (Gcy/s) = 2.4576 Tflop/s

b) 8 (channels) x 8 (byte/cy) x 3.2 (Gcy/s) = 204.8 Gbyte/s

c) Amount of floating point computations: 2 x 10^9 flops
Amount of data transfer: 32 x 10^9 byte

```
for(int i=0; i<10^9; ++i)
  a[i] = b[i] + s * c[i];
```

Time to do the flops at peak flop rate:

$$T_{flops} = \frac{2 \times 10^9 \text{ flops}}{2.4576 \text{ Tflops/s}} = 814 \text{ µs}$$

Time to transfer the data at peak BW:

$$T_{BW} = \frac{32 \times 10^9 \text{ byte}}{204.8 \text{ Gbyte/s}} = 156 \text{ ms}$$

→ Under the given conditions, the bottleneck is clearly the data transfer

Assignment 4 – Task 2

a) Locality of access (`a[][]` in memory, assuming good loop order):

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

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

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

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

Assignment 4 – Task 2

b) Assuming standard stores are used → write-allocate applies

`y[]` does not count (comes from register)
`x[]` comes from a cache N-1 times
or from memory if too large → 0 or 8 B/it.
`a[][]` always must be updated in memory,
no reuse → 16 B/it (incl. write-allocate)

L3 cache size (L_c) impacts the threshold N

→ `x[]` fitting into (half of) the cache is at $N < \frac{L_c / 2}{8B}$

NT stores can be used
→ B_c reduction by 8 B/it in all cases

If `x[]` fits into:	B_c in memory [B/it]
L1	16
L2	16
L3	16
Memory	24

Assignment 4 – Task 3

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

for(i=0; i<N; ++i) {
  a[i] += b[i] * c[i];
}
z[0] = 0.;
for(i=1; i<N; ++i) {
  z[i] = a[i-1] * 0.5f;
}
```


```
z[0]=0;
a[0] += b[0] * c[0];
for(i=1; i<N; ++i) {
  a[i] += b[i] * c[i];
  z[i] = a[i-1] * 0.5f;
}
```

$$B_c = \frac{16 + 12}{3}\frac{B}{F} = 9.33\ \text{B/F or } 28\ \text{B/it}$$

$$B_c = \frac{16+8}{3}\frac{B}{F} = 8\ \text{B/F or } 24\ \text{B/it}$$

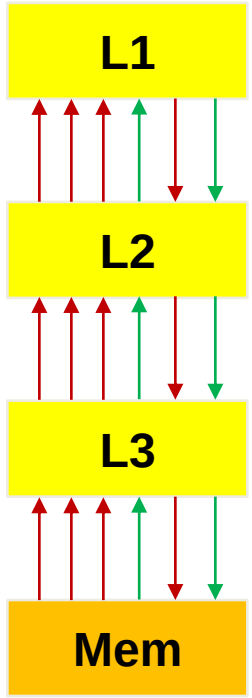
a) Is the code SIMD vectorizable?

- Original: yes (trivially)
- Optimized: yes, because the backward reference to `a[i-1]` is harmless

Assignment 4 – Task 3

b) Data transfer modeling for optimized code on hypothetical CPU

```
for(i=1; i<N; ++i){
  a[i] += b[i] * c[i];
  z[i] = a[i-1] * 0.5f;
}
```



b_S=110 GB/s → 1.28 cy/CL @ 2.2 GHz

- L1-L2: 6 CLs → 6 cy
- L2-L3: 6 CLs → 24 cy
- Memory: 6 CLs * 1.28 cy/CL = 7.7 cy