

Microbenchmarking for architectural exploration

Probing of the memory hierarchy

Saturation effects



Motivation for Microbenchmarking as a tool

- Isolate small kernels to:
 - Separate influences
 - Determine specific machine capabilities (light speed)
 - Gain experience about software/hardware interaction
 - Determine programming model overhead
 - ...
- Possibilities:
 - Readymade benchmark collections (epcc OpenMP, IMB)
 - STREAM benchmark for memory bandwidth
 - Implement own benchmarks (difficult and error prone)
 - **likwid-bench** tool: Offers collection of benchmarks and framework for rapid development of assembly code kernels

The parallel vector triad benchmark - A “swiss army knife” for microbenchmarking

```
double striad_seq(double* restrict a, double* restrict b, double* restrict c,
double* restrict d, int N, int iter) {
    double S, E;
    S = getTimeStamp();
    for(int j = 0; j < iter; j++) {
#pragma vector aligned
        for (int i = 0; i < N; i++) {
            a[i] = b[i] + d[i] * c[i];
        }
        if (a[N/2] > 2000) printf("Ai = %f\n",a[N-1]);
    }
    E = getTimeStamp();
    return E-S;
}
```

Required to get optimal code with Intel compiler icc! New icx unclear

Keeps smarty-pants compilers from doing “clever” stuff

- Report performance for different **N**, choose **iter** so that accurate time measurement is possible
- This kernel is limited by data transfer performance for all memory levels on all architectures, ever!

A better way – use a microbenchmarking tool

- Microbenchmarking in high-level language is often difficult
- Solution: assembly-based **microbenchmarking framework**
 - e.g., **likwid-bench**

```
$ likwid-bench -t triad_avx512_fma -W S0:28kB:1
```

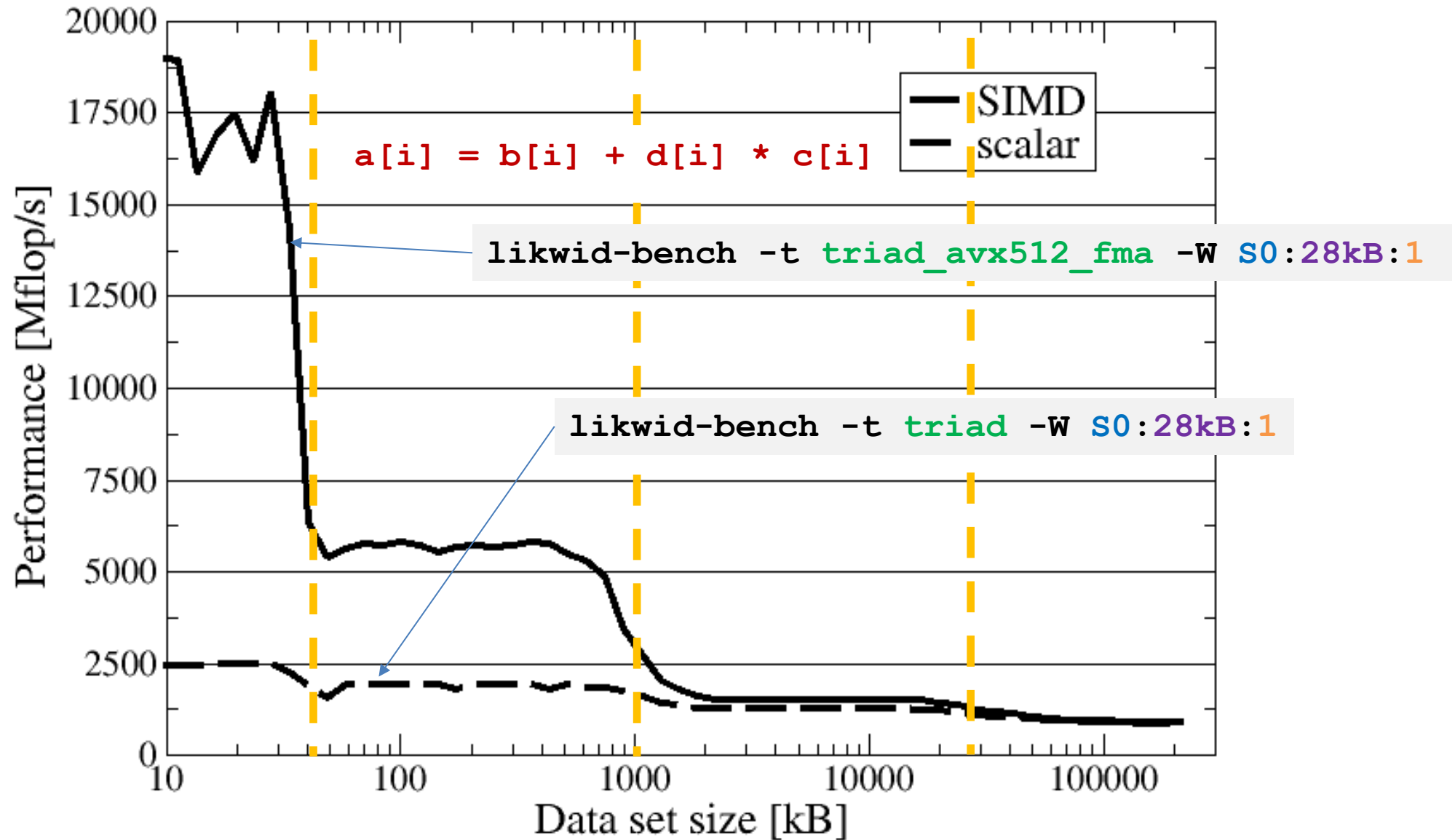
benchmark type

topological entity (see likwid-pin)

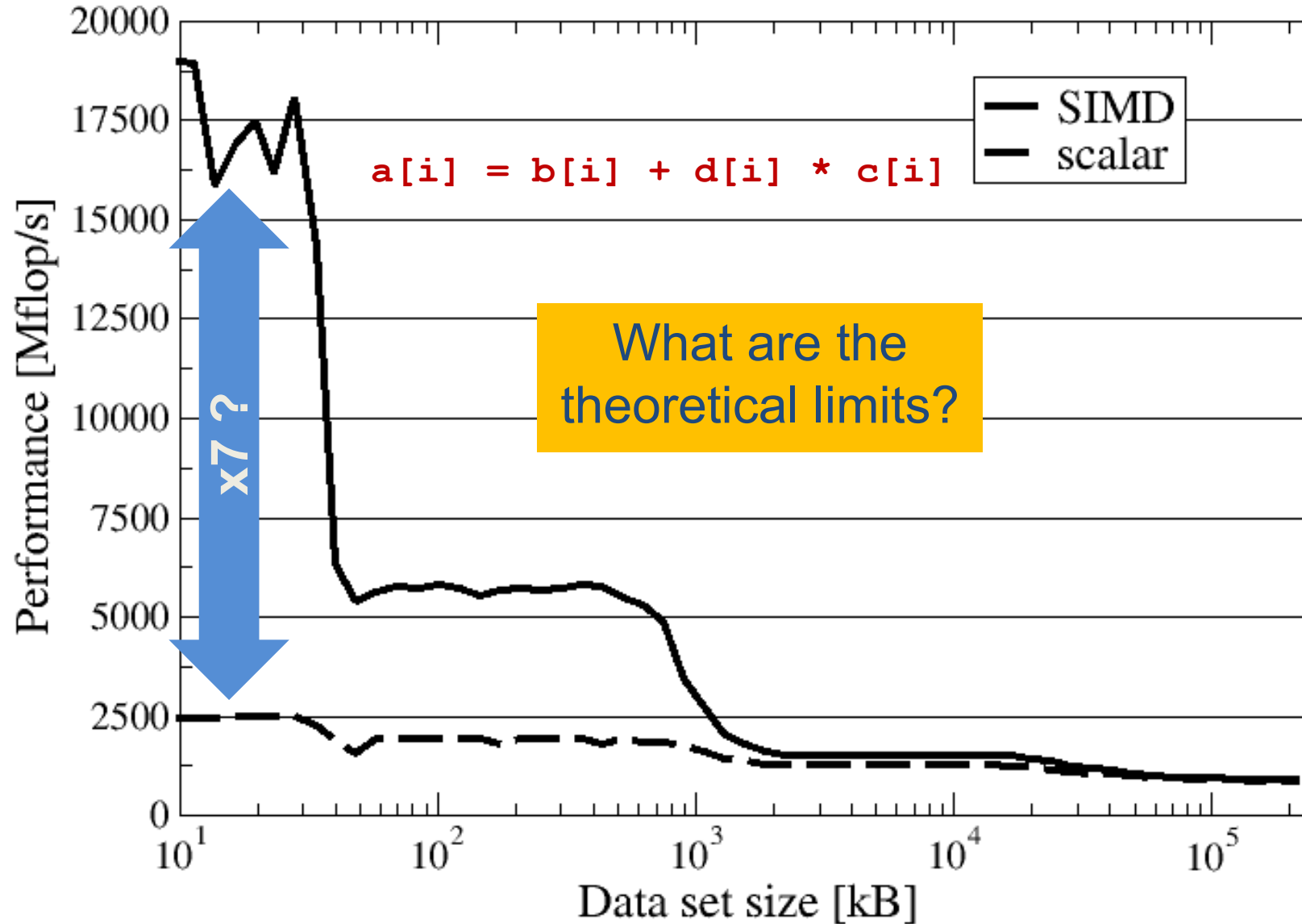
working set

of threads

Schönaauer triad on one CascadeLake core 2.5GHz



Schönauer triad on one CascadeLake core 2.5GHz

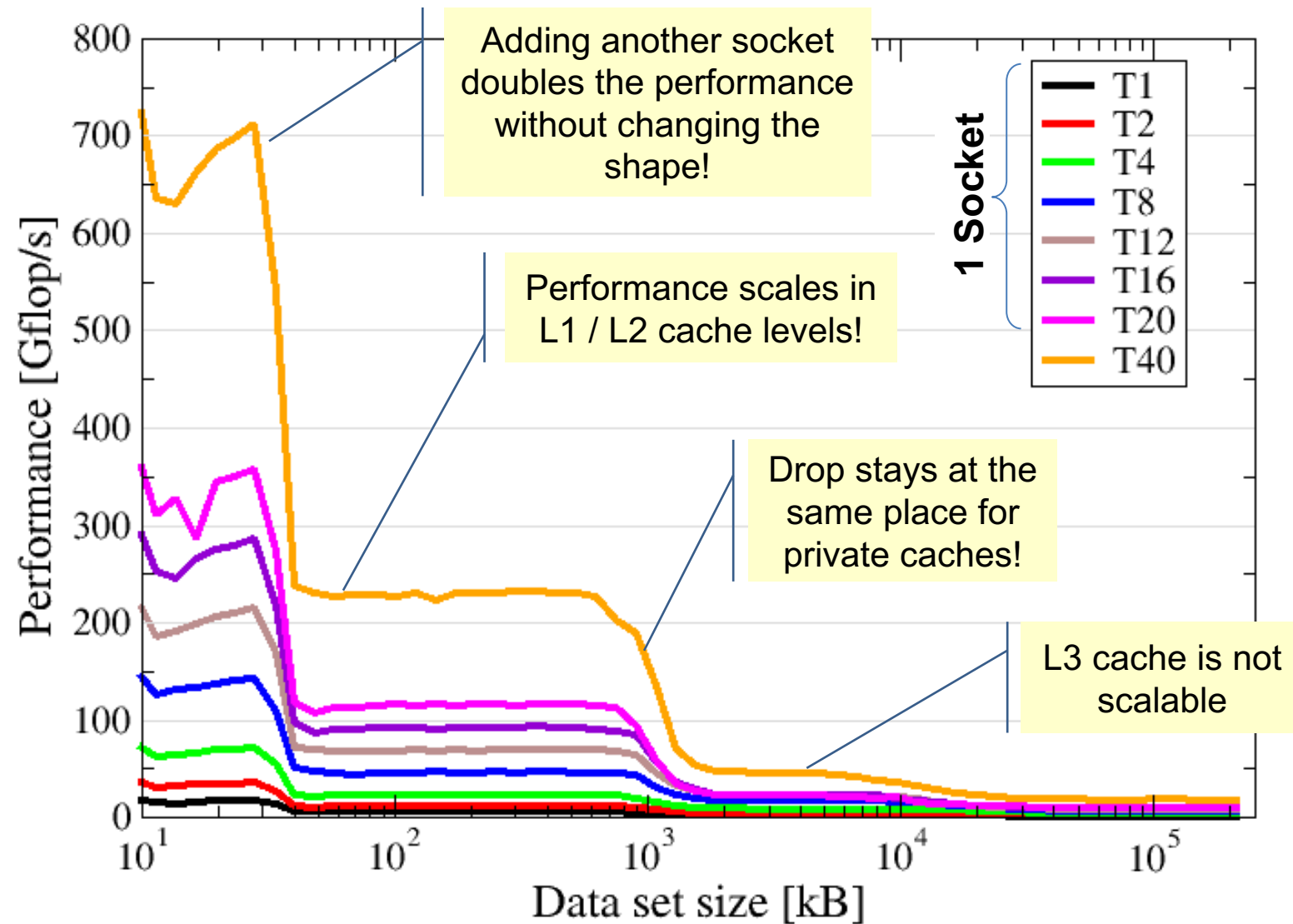


Throughput triad on one CascadeLake node (2.5 GHz)

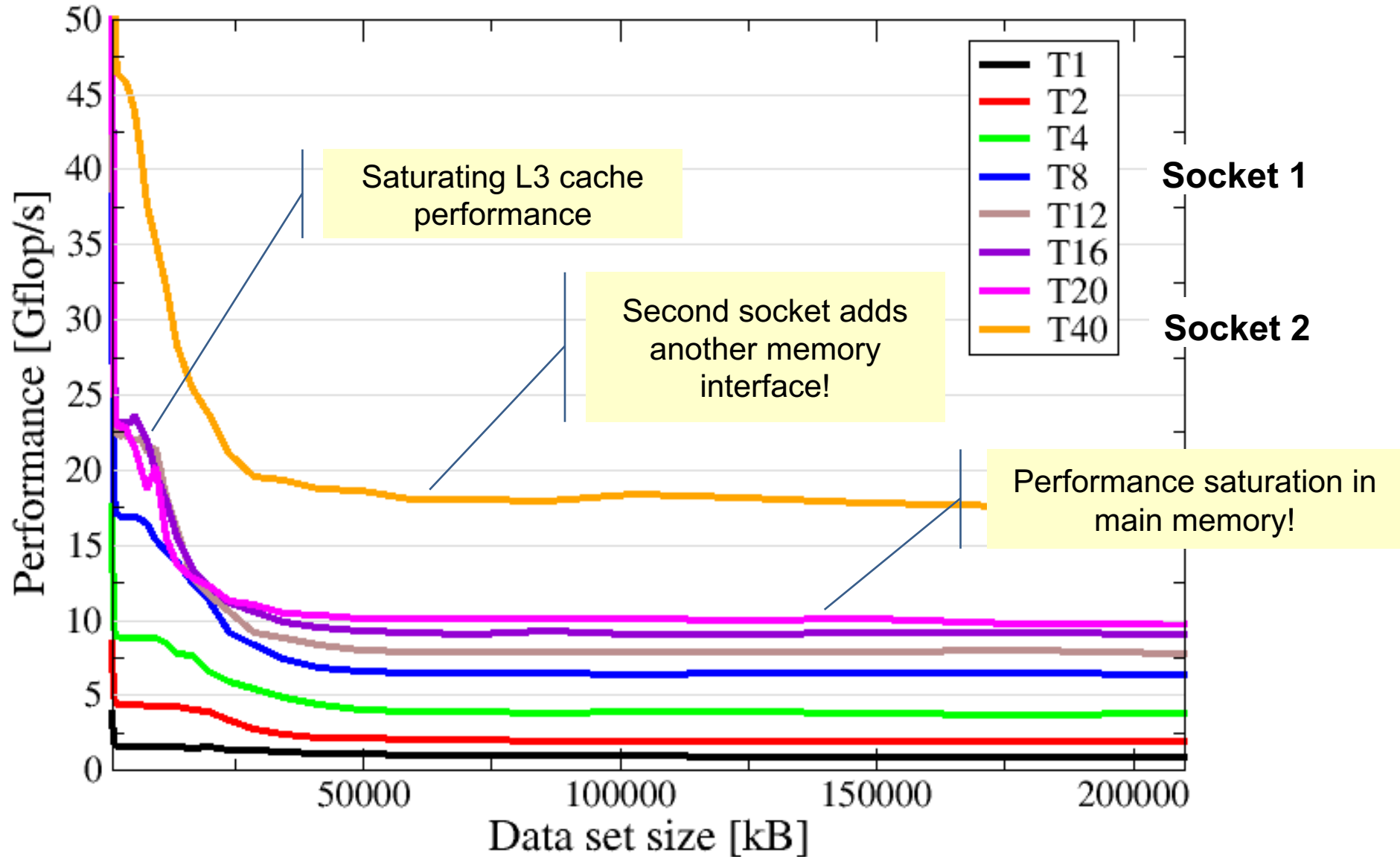
- How does the bandwidth scale across cores?
- Are there any bottlenecks?
- How large are the caches?

```
likwid-bench \  
-t triad_avx512_fma \  
-W S0:$size:$threads:1:2
```

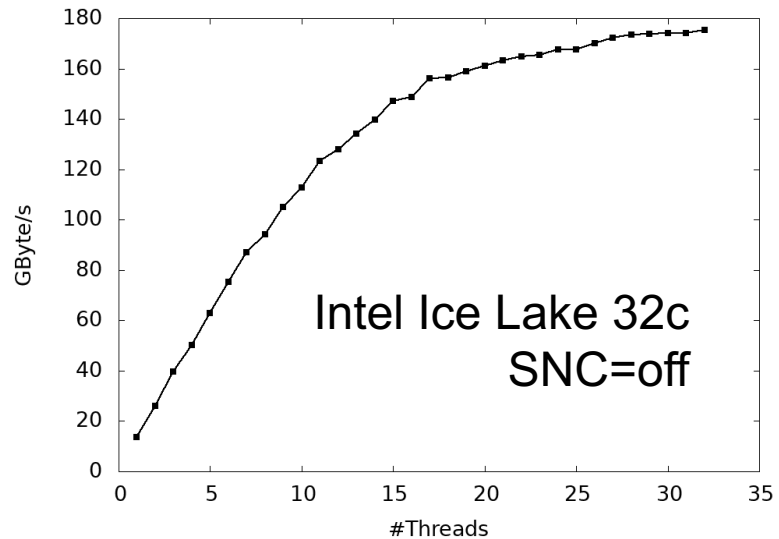
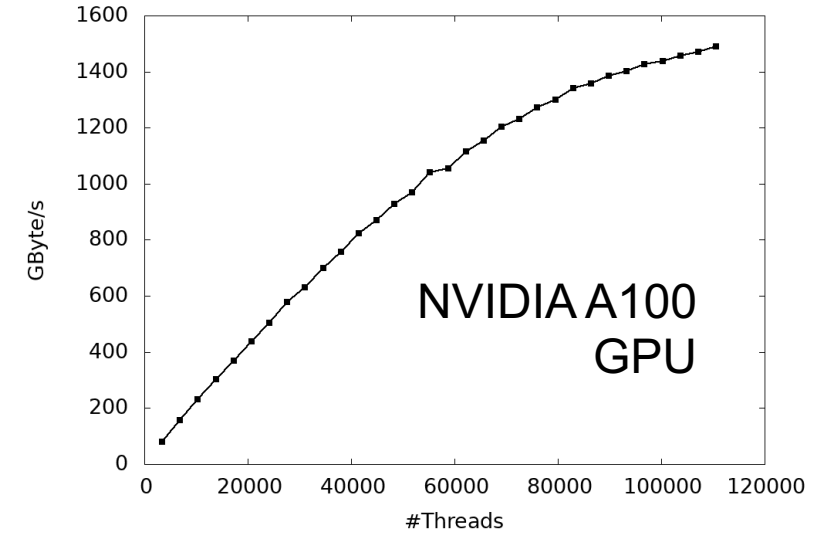
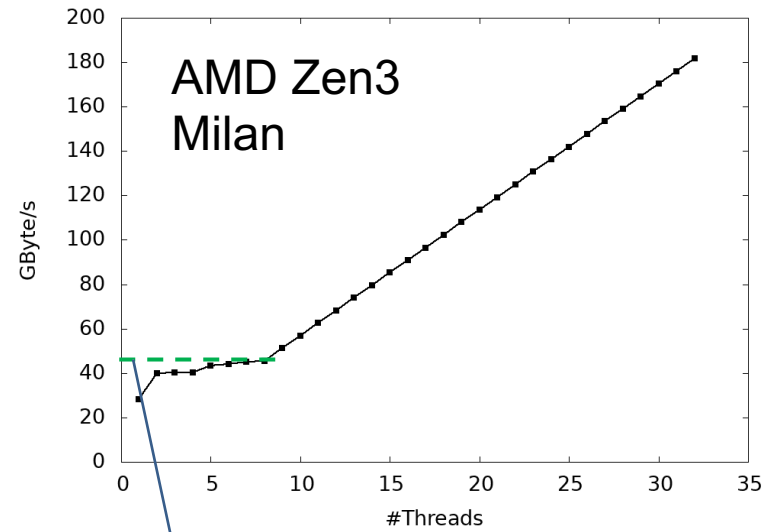
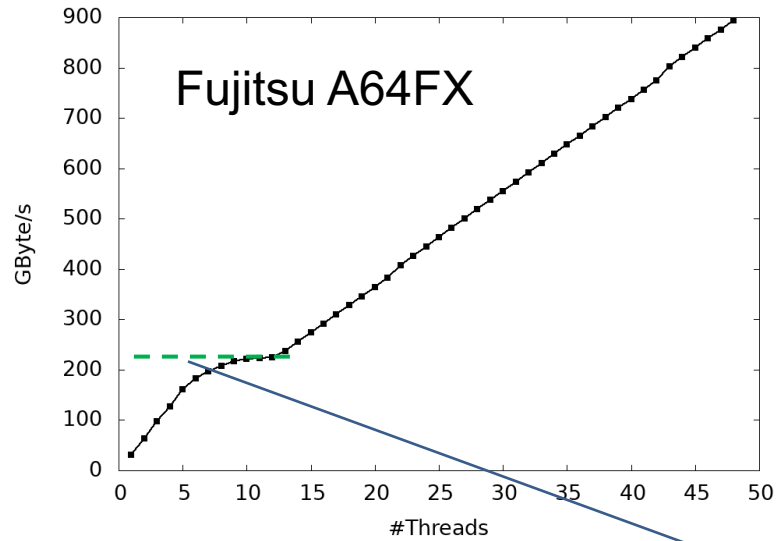
- Scan **\$size** and **\$threads**
- Pin threads in **chunks of 1** with **distance of 2** (skip SMT threads)



Throughput triad on CascadeLake (memory close-up)

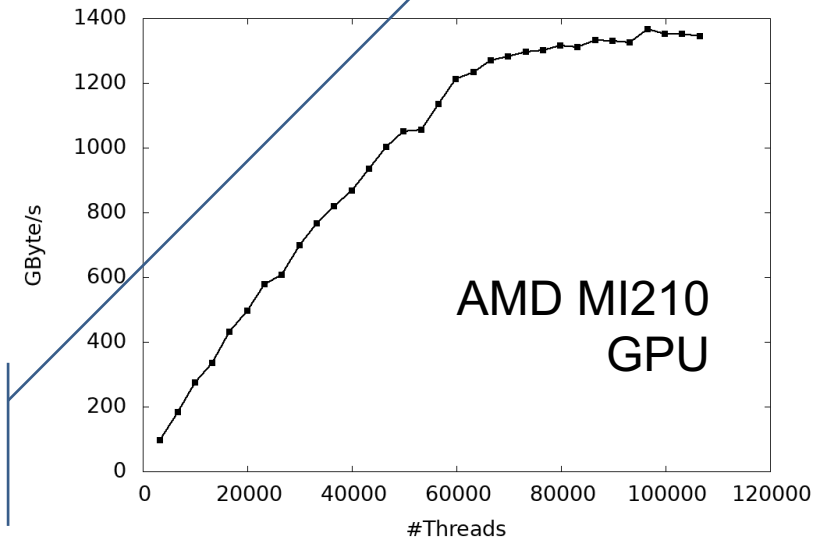


Memory bandwidth saturation (read-only)



Bandwidth saturation on 1st ccNUMA domain

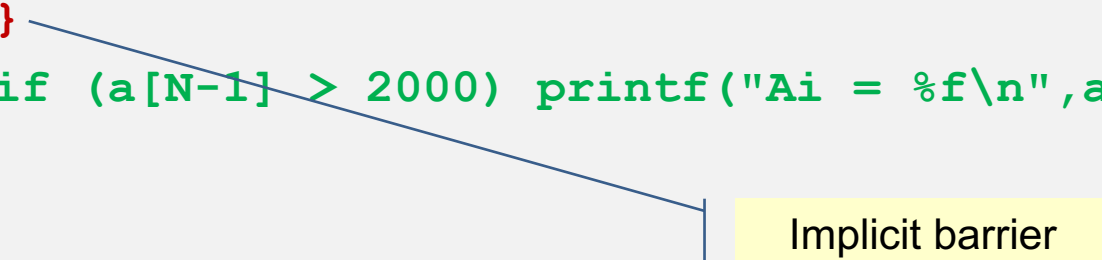
Massive thread parallelism needed on GPUs to saturate



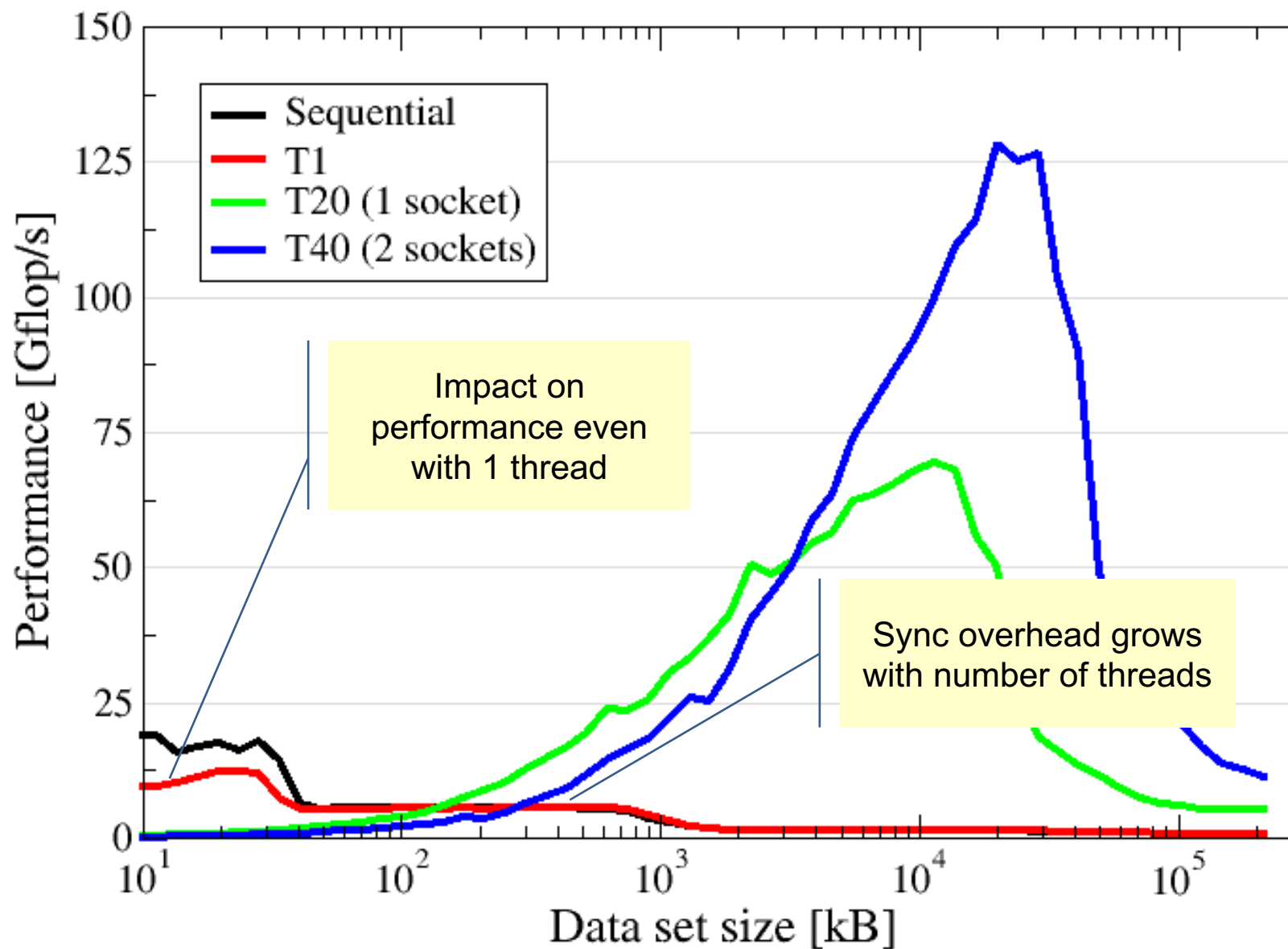
The OpenMP-parallel vector triad benchmark

OpenMP worksharing in the benchmark loop

```
S = getTimeStamp();
#pragma omp parallel
{
    for(int j = 0; j < iter; j++) {
#pragma omp for
#pragma vector aligned
        for (int i=0; i<N; i++) {
            a[i] = b[i] + d[i] * c[i];
        }
        if (a[N-1] > 2000) printf("Ai = %f\n",a[N-1]);
    }
}
E = getTimeStamp();
```

A blue line originates from the closing brace of the innermost loop (the one containing the vector-aligned loop) and points to a yellow rectangular box labeled "Implicit barrier". This box is positioned between the inner loop and the outer loop's closing brace, indicating that a barrier is implicitly placed at the end of the inner loop.

OpenMP vector triad on CascadeLake node (2.2 GHz)



Conclusions from the microbenchmarks

- **Microbenchmarks** can yield **surprisingly deep insights**
- **Affinity matters!**
 - Almost all performance properties depend on the position of
 - Data
 - Threads/processes
 - Consequences
 - **Know where your threads are running**
 - **Know where your data is** (see later for that)
- **Bandwidth bottlenecks are ubiquitous**
- **Synchronization overhead** may be an issue
 - ... and depends on the system topology!
 - Many-core poses new challenges in terms of synchronization