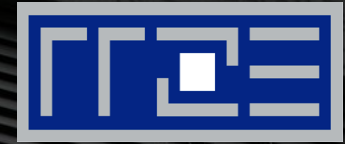


**ERLANGEN REGIONAL
COMPUTING CENTER [RRZE]**



Performance Analysis with LIKWID

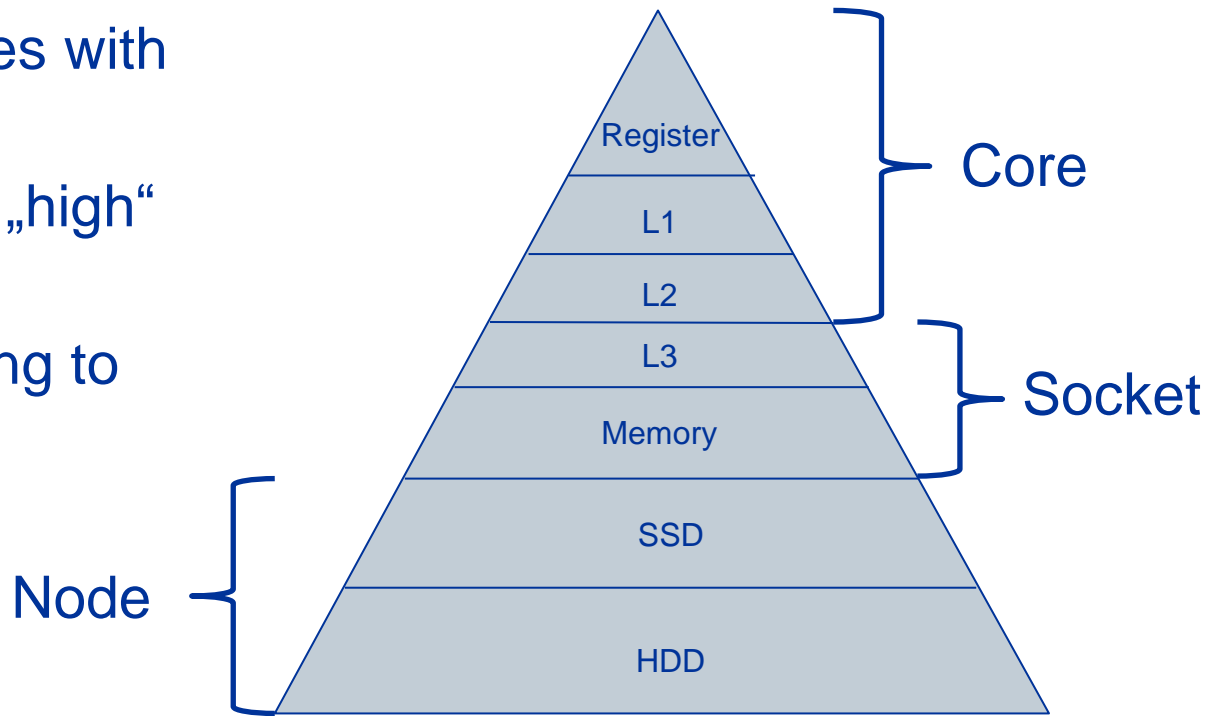
Thomas Röhl (HPC @ Uni Erlangen)

Agenda

- LIKWID
- Identify performance limitations
 - Bandwidth saturation
 - False-sharing of cache lines
 - Pipeline stalls
 - Excess data sizes
- DEMO (if we have time)

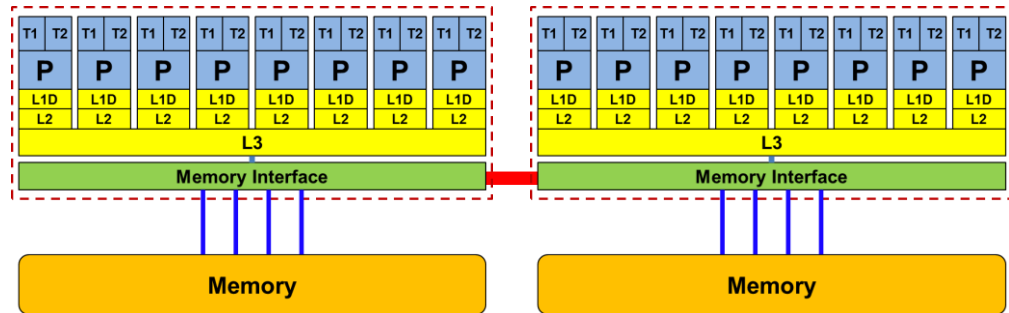
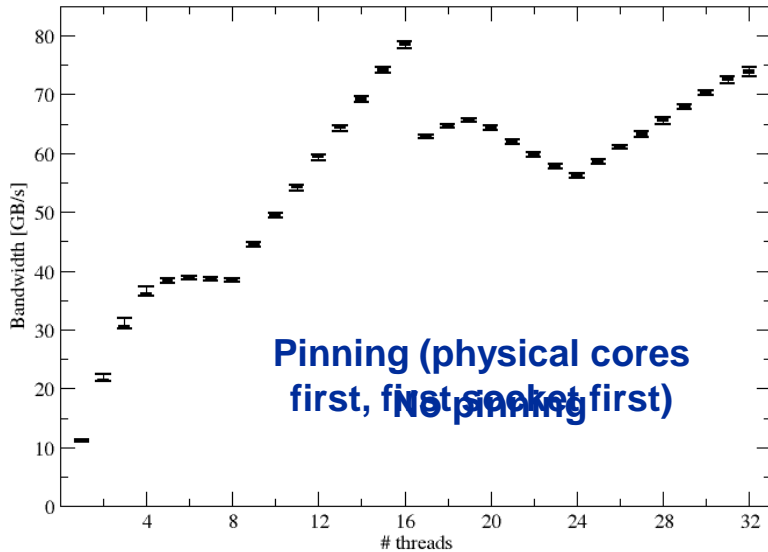
Importance of Affinity

- Bandwidth decreases with each level
- Try to keep data as „high“ as possible
- Pin threads according to data locality



Importance of Affinity

STREAM benchmark on 16-core Sandy Bridge



LIKWID

Overview

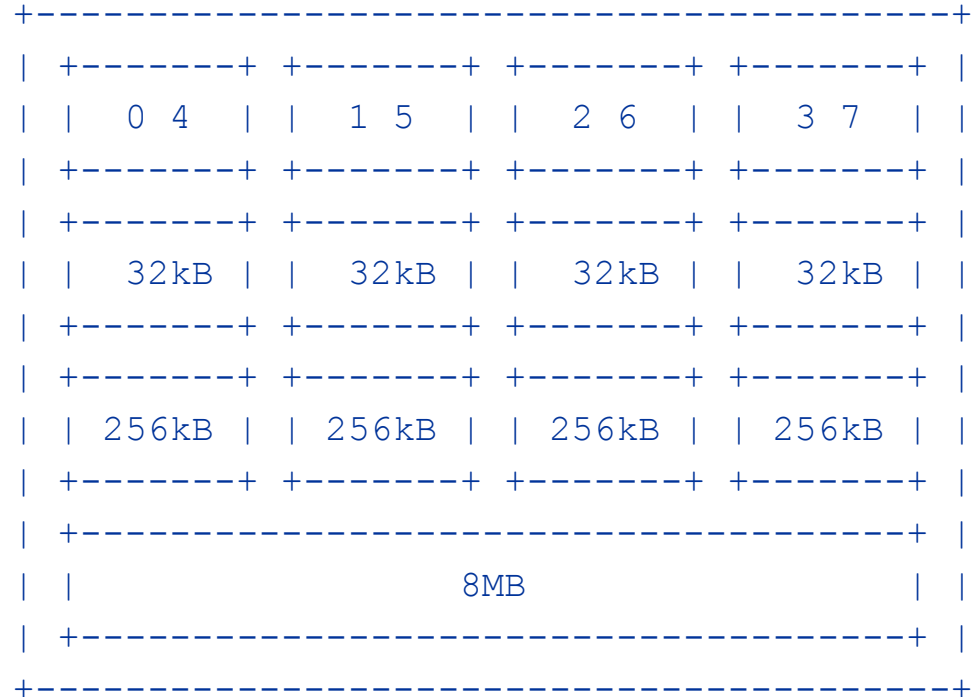
- „Like I know what I do“
- Set of tools:
 - Topology information
 - Process/Thread pinning
 - Hardware performance monitoring
 - Low-level benchmarking
 - CPU frequency manipulation
 - CPU feature manipulation (prefetchers)

System topology with LIKWID

likwid-topology

- Thread topology
- Cache topology
- NUMA topology
- Graphical topology

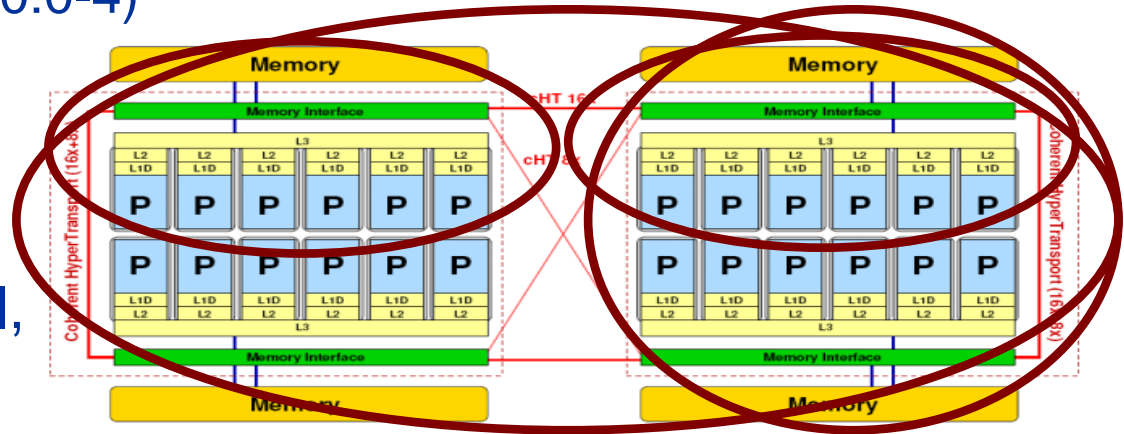
Socket 0:



Affinity with LIKWID

likwid-pin

- LIKWID defines affinity domains:
 - Node (N:0-19)
 - Last Level Cache (C0:0-4)
 - Socket (S0:0-4)
 - NUMA domain (M0:0-4)
- Also physical, logical, function-based selection



LIKWID

Performance groups

- Event names are not intuitive -> difficult selection
- Performance groups combine event set and derived metrics
- Easy event set management
- Get counter results and derived metrics (bandwidth, ratios, ...)

- Examples:

- `likwid-perfctr -c S0:0-3@S1:0-3 -g L3 ./a.out`

- `likwid-perfctr -C E:N:10:1:2 -g FLOPS_DP ./a.out`

No pinning

L2/L3 traffic

Pinning

10 threads, 1 out of 2

Double-precision floating point ops

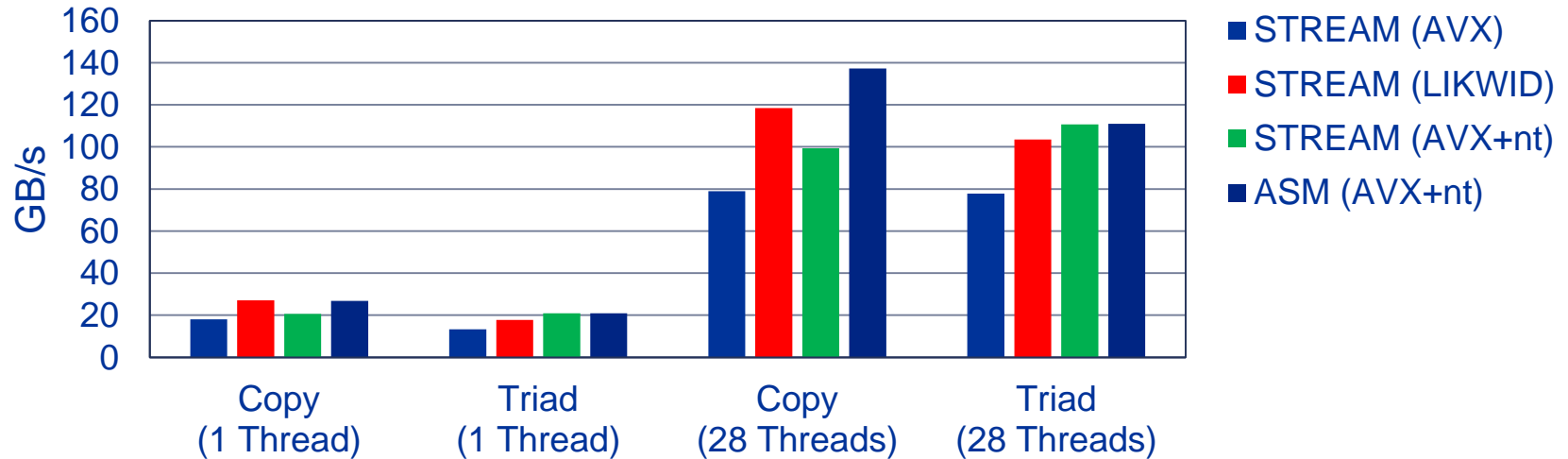
Hands on patterns – Bandwidth limitation

- How fast can a memory copy be?
- Often only the application data is regarded
- Is there hidden traffic?
 - Caused by instructions added by compiler
 - Caused by hardware behavior
 - Management traffic
- Does the compiler emit the most performant code?

Hands on – Bandwidth limitation

STREAM bandwidth on Intel Haswell EP (14C, 2.3GHz)

Memory bandwidth comparison



Hands on – Bandwidth limitation

STREAM bandwidth

- CL stores cause Read-for-Ownership
 - Implicit, even in assembly
 - Detectable by hardware performance counters
 - For pure data loops force non-temporal stores
 - Think about the data flow of a loop
 - Pin the threads
-
- For higher languages: identify hidden data operations

Hands on – False-sharing

Parallelize histogram code

- Serial histogram code

```
hist[16] = { 0 };  
for(j=0; j<1000000000; ++j) {  
    hist[rand_r(&seed) & 0xf]++;  
}
```

← **Data race!**

- Solution: Own histogram per thread

```
hist[16][MAX_THREADS] = { 0 };  
#pragma omp parallel for firstprivate(seed)  
for(j=0; j<1000000000; ++j)  
    hist[rand_r(&seed) & 0xf][tid]++;  
// combine thread histograms
```

Hands on – False-sharing

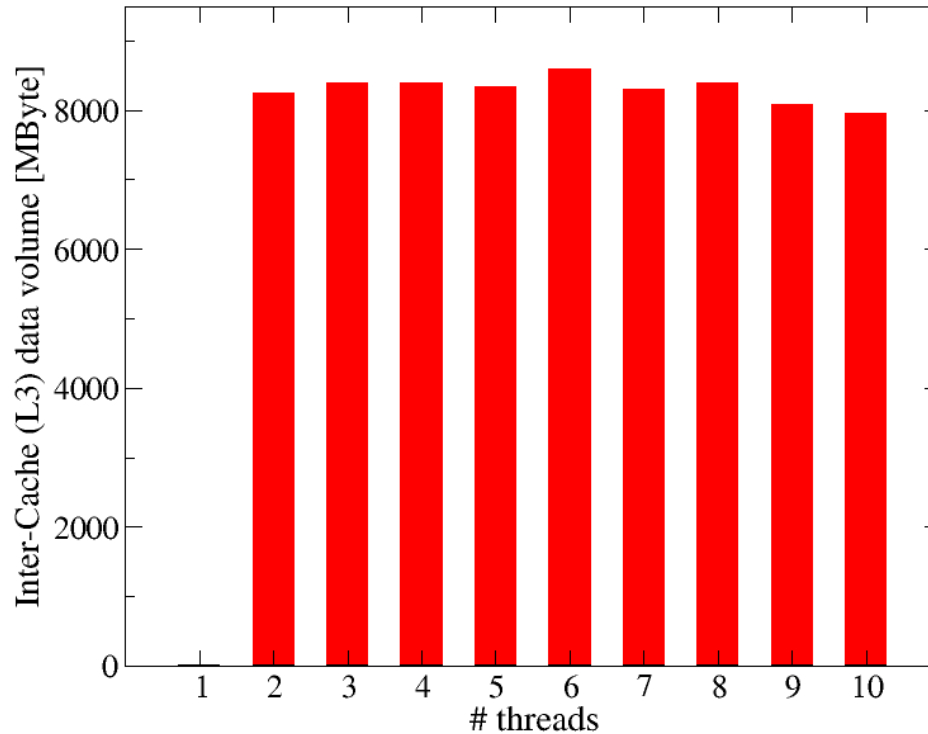
Parallelize histogram code



■ 64B cache line

Hands on – False-sharing

Parallelize histogram code



Single thread:

4.85 s

10 threads:

7.58 s

**After Optimization
with thread-local
histogram**

10 threads: 0.61 s

Intel IvyBridge EP

Hands on – False-sharing

Problems

- CL sharing is normal in almost every code
 - Global data structures (stop criterion, memory addresses, ...)
 - Common structs (static information)
- Cannot differentiate between required shared CL and falsely shared CL
- Counters not accurate (Haswell: up to 50% deviation)
- No information about cache flushes / memory barriers

Hands on – Pipeline stalls

Vector summation

- Often used operation

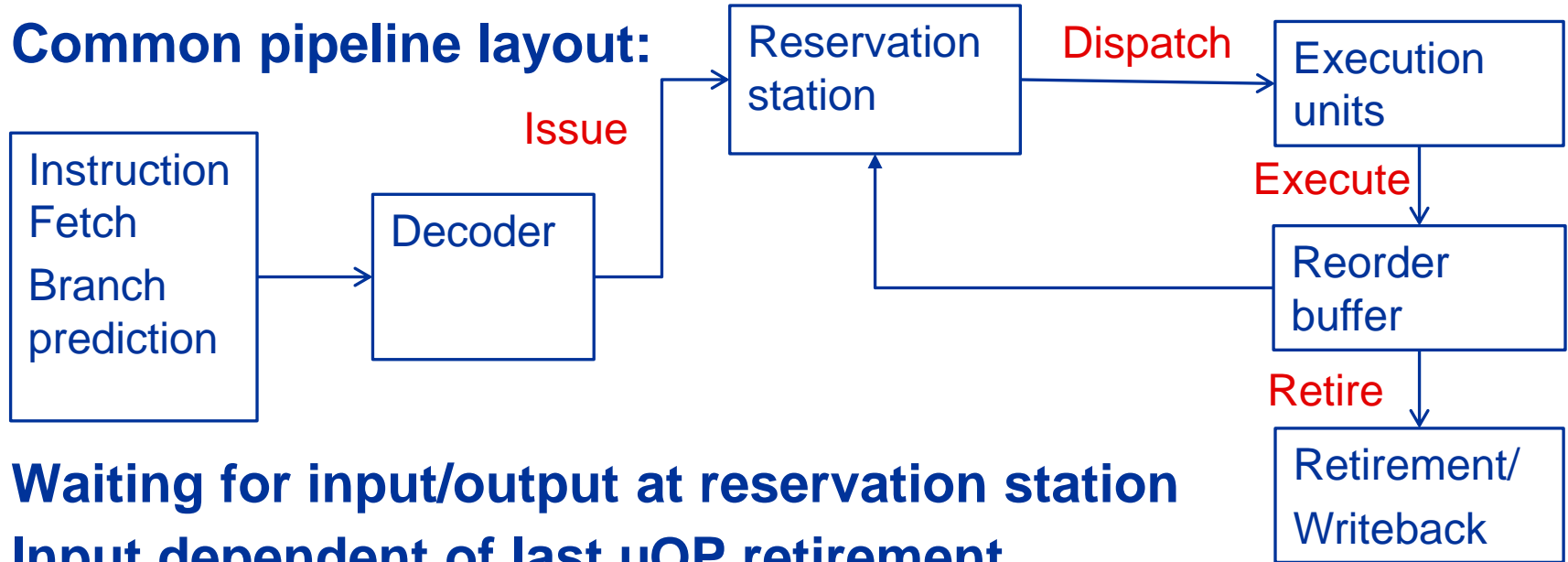
```
sum = 0;
for(j=0; j<N; ++j) {
    sum += vector[j];
}
```

- Performance insensitive of data size
- In-core throughput far from design limit
- Performance model and measurements do not fit
- Almost no options for optimization

Hands on – Pipeline stalls

Vector summation

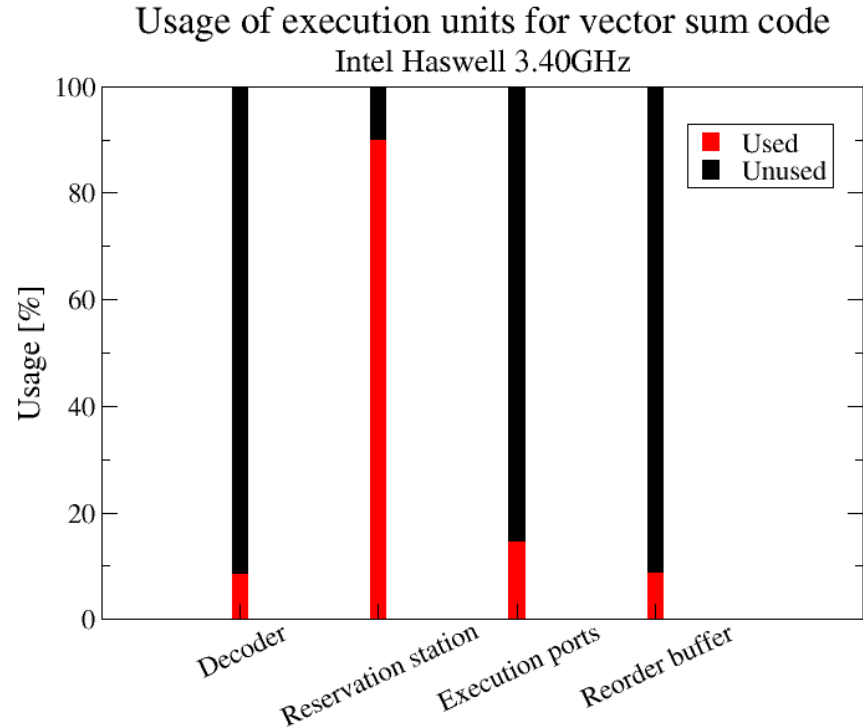
- Common pipeline layout:



- Waiting for input/output at reservation station
- Input dependent of last μ OP retirement
- Each stall induces penalty

Hands on – Pipeline stalls

Vector summation



Bottleneck at reservation station! No chance for optimization!

Hands on – Pipeline stalls

Vector summation

- Loop unrolling no improvement
- Avoid small loops acting on single variables
- Better code mixture (try to do anything else with the data)
- Parallization helps a bit
 - Each thread as bottleneck
 - Shorter aggregated stall time per thread
 - Less penelties

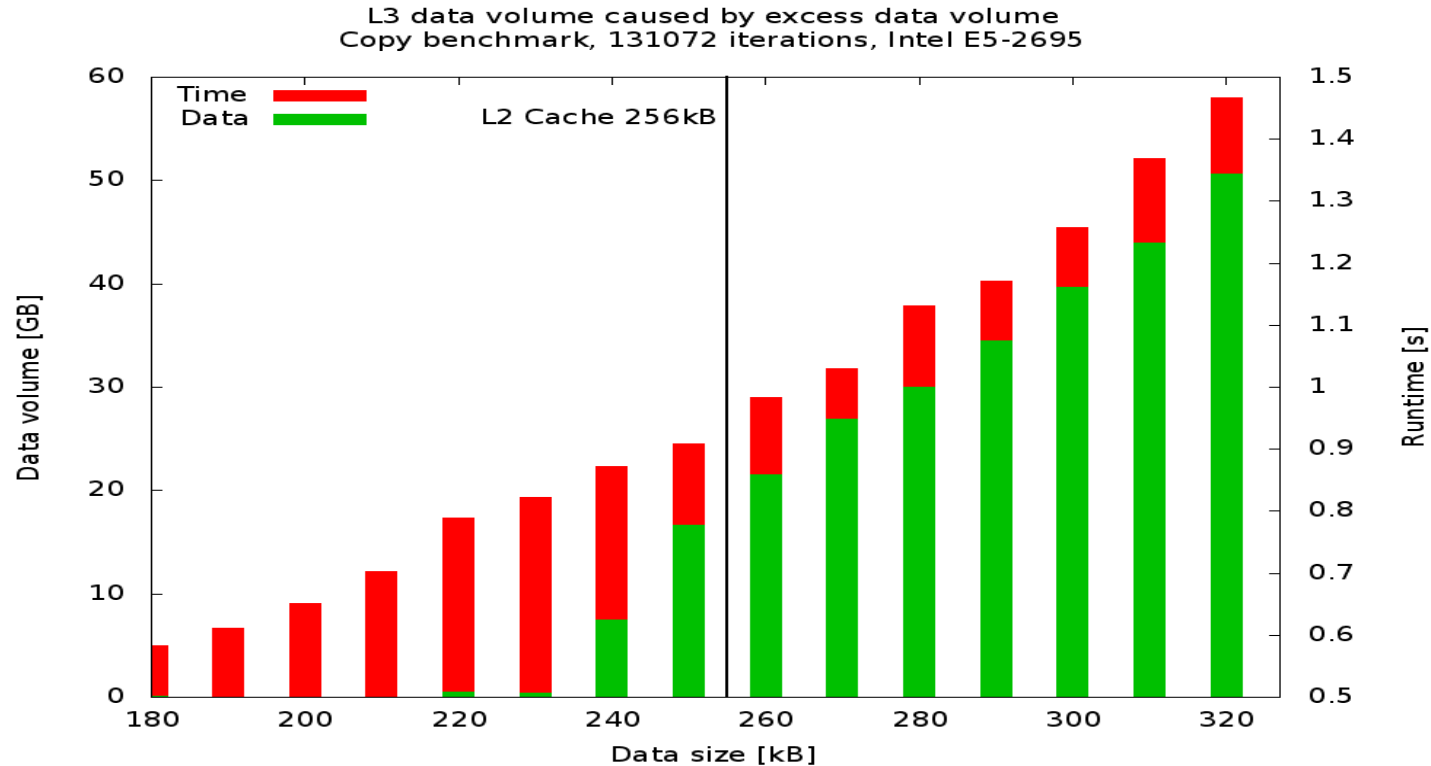
Hands on – Excess data sizes

Copy benchmark

- Data structures do not fit into the cache
- Unaligned objects/structs
- Improper blocking in cache layer

Hands on – Excess data sizes

Copy benchmark



Hands on – Excess data sizes

Copy benchmark

- Accesses to L3 slow down
- Getting worse in „lower“ level in memory hierarchy
 - ➔ Try to „block“ your data in the „highest“ cache layer

Conclusion

- Control execution environment
 - System topology
 - Pinning processes/threads
- Get behavior of hardware running software
 - Huge amount of events
 - Performance groups combine useful events

DEMO

(if we have time)

Thank you for your attention

Questions?

- LIKWID: <https://github.com/RRZE-HPC/likwid>
- STREAM: <http://www.cs.virginia.edu/stream/>