BEAST Lab

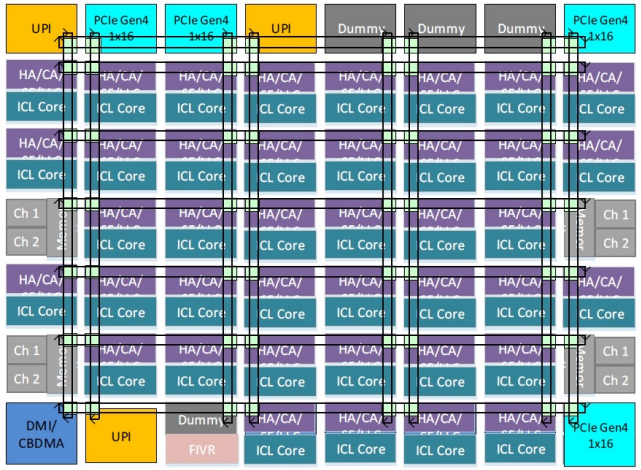
The Memory Hierarchy of Multi-core CPUs

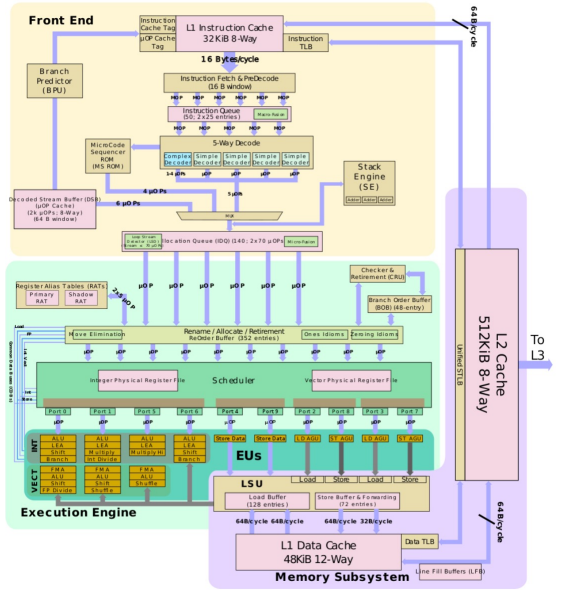
November 2, 2023 | Josef Weidendorfer

Example: Intel Ice Lake

Two systems in BEAST

- 2 sockets Intel Xeon (Icelake) Platinum 8360Y
  - 2x 36 = 72 cores
    - 2x 512bit vector units per core (8 x DP FMA)
    - 2 threads per core (”Hyper-Threading”)
    - 2.4 GHz base, Intel 10nm
- 512 GB main memory, 1.5 TB Optane NVRam

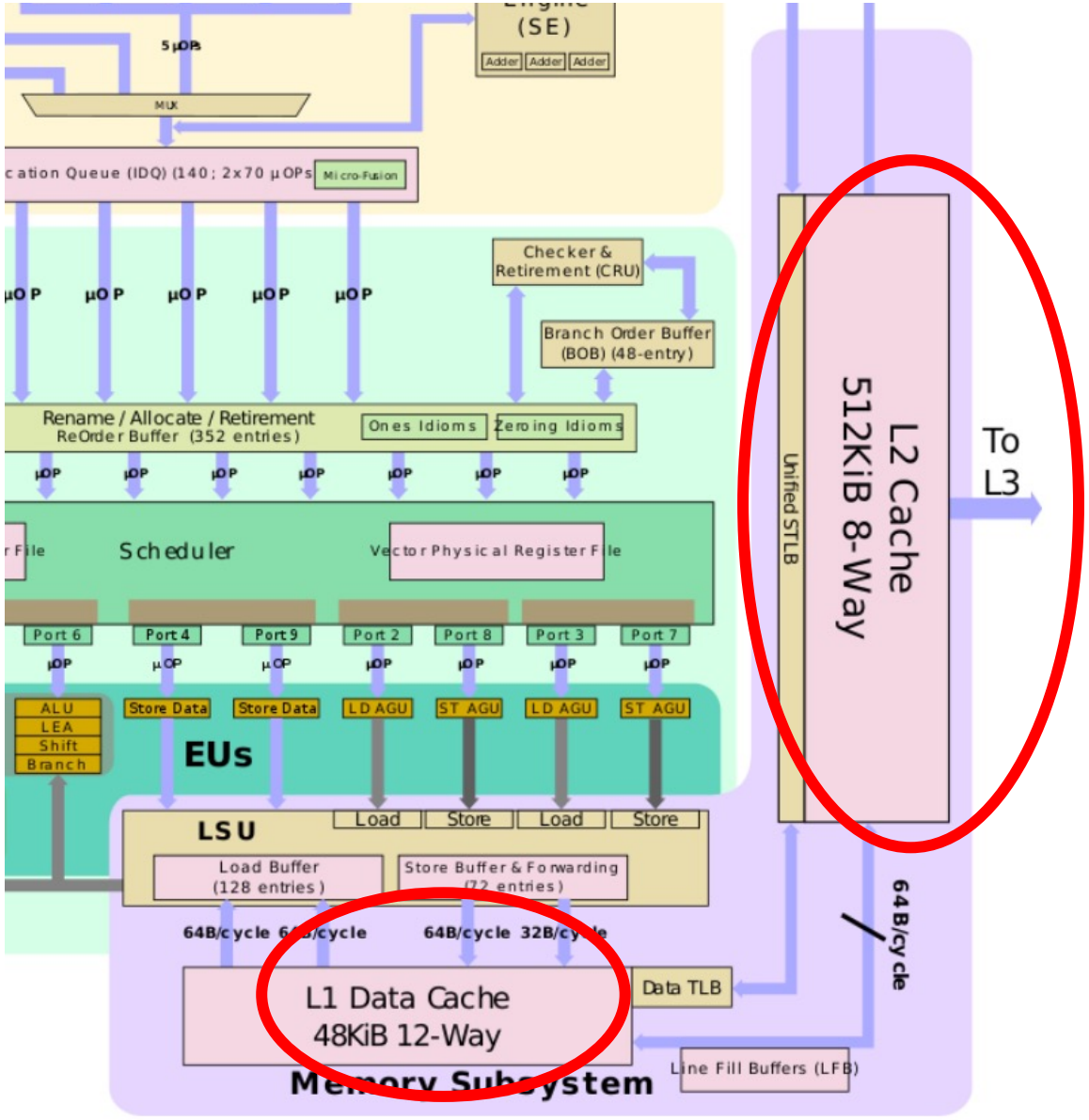


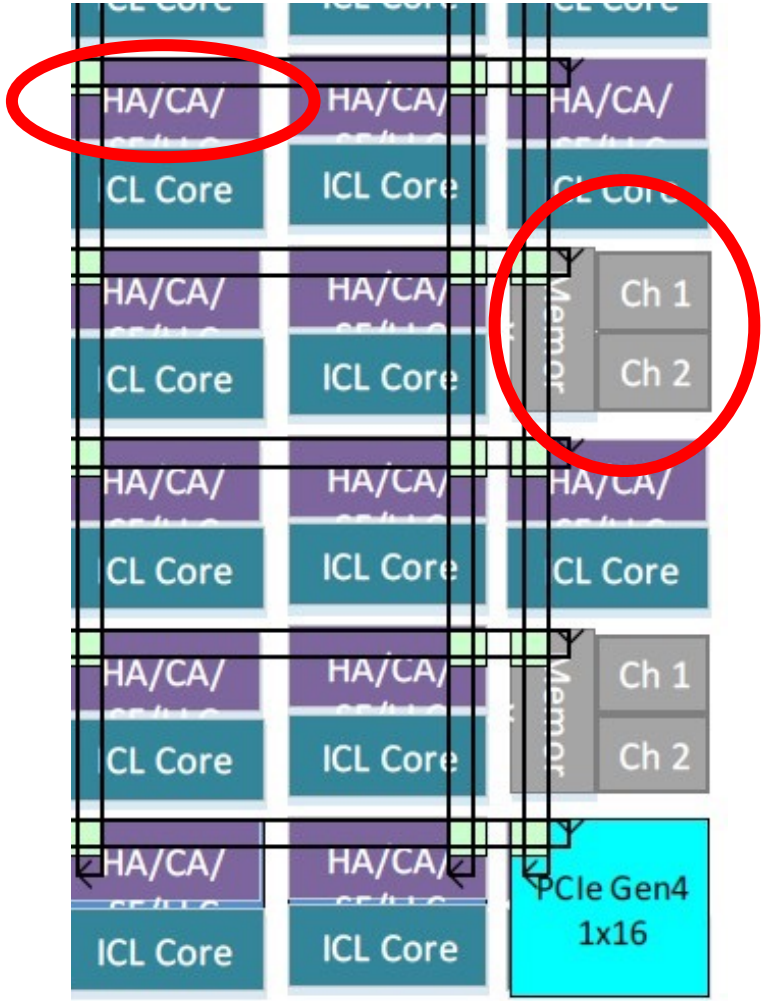


Links

- https://en.wikichip.org/wiki/intel/microarchitectures/ice_lake_(server)
- https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove

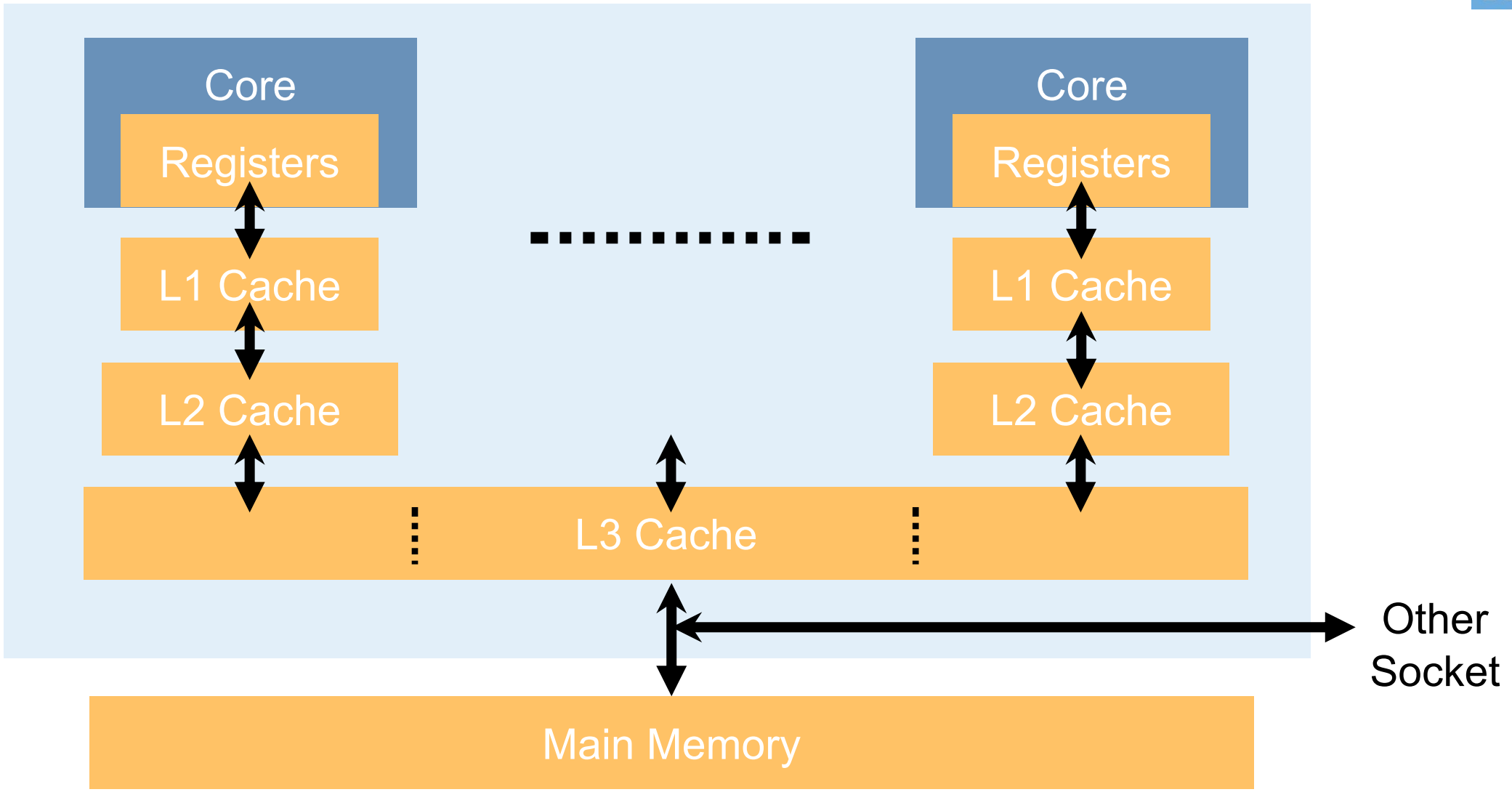
Example: Intel Ice Lake



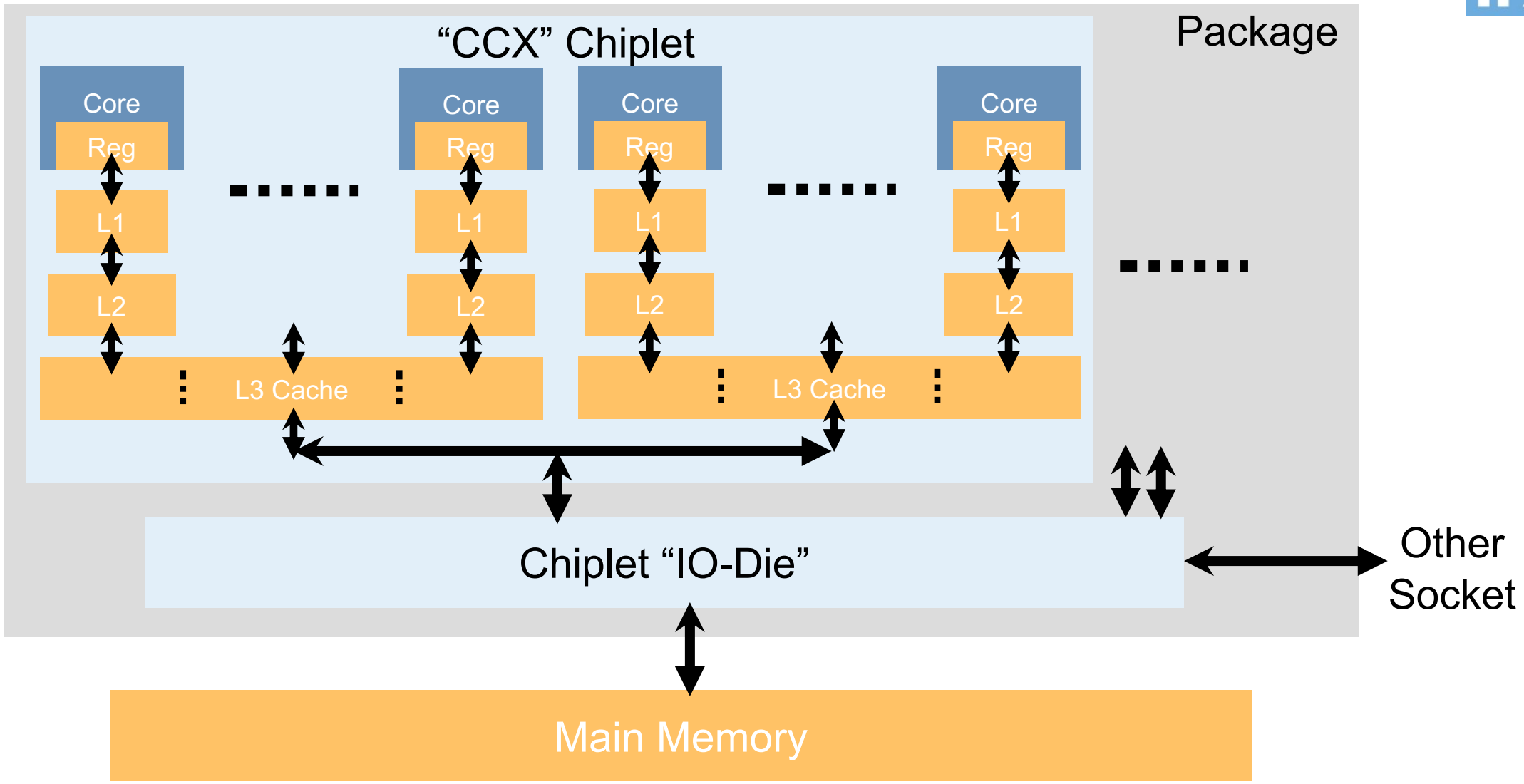


„Partitioned L3“ shared by all cores
(Location to store copy derived from address)

Example: Intel Ice Lake (similar to Skylake on SuperMUC-NG)



Example: AMD Milan



Cache and Cache-Levels: Motivation

Example: Intel Laptop from 2015 (AVX2)

- compute peak for 2 GHz CPU, 4 cores, 1 vector unit with 4 FP64 floats (FMA):
  2 * 10e9 /s * 4 * 4 * 2 Flops = 64 GFlop/s
- Triad (Assignment 1): A(i) = B(i) + C(i) * D(i)
  - per FP64 element: 3 * 8 B (Load) + 8 B (Store)
  - “Arithmetic Intensity”: 2 Flops / 32 Bytes transferred = 0.067 Flop/B
  - required bandwidth to allow peak: 64 GFlop/s / 0.067 Flop/B = 1024 GB/s
- Off-chip memory ~ 25 GB/s (Laptop) : 40x slower!
- Latency ~ 50ns: while accessing memory once, **one** core can do
  100 cycles * 8 Flop/cycle = 800 Flops

Solution: keep recently accessed data on-chip in a “cache”

Cache and Cache-Levels

How to decide what data to keep in the cache?

- Cache replacement policy: LRU (drop least-recently used if full)

Overhead: need to store both address (to check for existing copy) and data in cache

- Programs often access data at near-side addresses ➔ fetch/store data in blocks
- Can utilize parallel signal lanes to memory (multiple memory controllers + bursts)
- Implicit prefetch for near-side data
- “Cacheline” size: Ice Lake / Milan / ThunderX2: 64 Byte, A64FX: 256 Byte

Performance of on-chip caches depends on cache size

- Smaller caches are faster ➔ use a hierarchy of caches
- Ice Lake / Milan / ThunderX2: 3 levels, A64FX: 2 levels

Keep the Pipe filled: Memory Parallelism & Hardware Prefetching

Motivation: Assume synchronous, serialized memory accesses per core

- % of bandwidth usable?
- Best case: program does continuous memory accesses
  - 50 ns for 64 Bytes per core
    with sequential code (1 core): 64 / 50 ns ~ 1,2 GB/s (vs. 25 GB/s available on laptop)

Memory Parallelism

- caches + memory allow multiple outstanding loads (Ice Lake/Rome: > 8 per core)
- modern CPU cores can look ahead and trigger accesses speculatively (”Out-of-Order”)

Hardware Prefetching

- Cache predicts next memory accesses from recent access pattern
- Easier per core (L2), works very well for streams (up and down), can go wrong (!)

(Low-Level) Tuning Tips for Memory

Make use of caches

- Reorder your code such that you can **reuse** recently loaded data if possible

  Blocking / Tiling: instead of 10 times over full array, split into blocks and go 10x over each
  Q: How to take advantage of multiple cache levels (L1 / L2 / L3) ?
- Choose a memory layout + access order such that **all** data in a cache line can be used before being evicted

Make use of hardware prefetcher

- For data too large to fit into cache: use stream accesses as much as possible

Make use of memory parallelism

- **Independent** accesses in instruction stream can be triggered in parallel with OoO

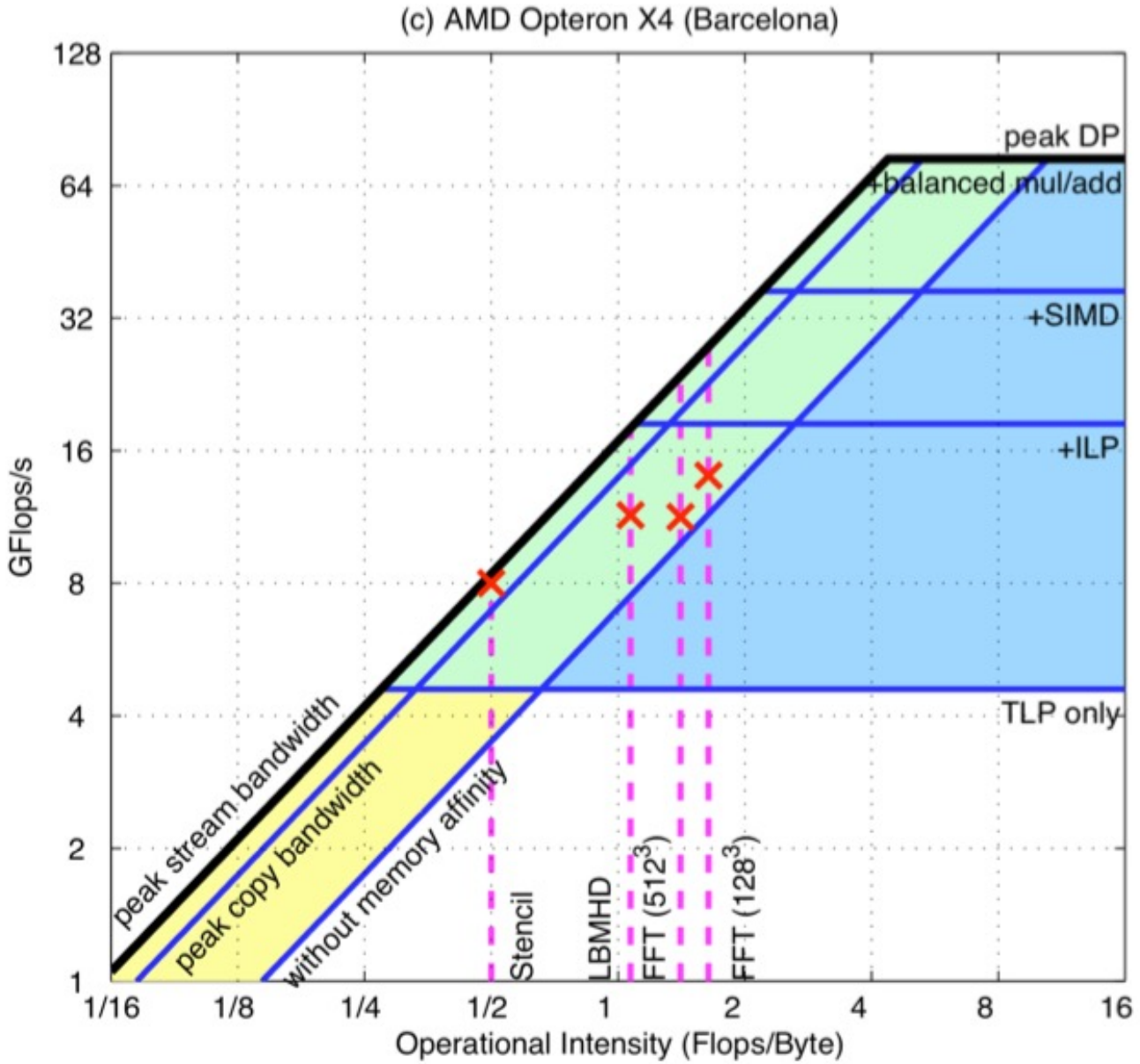
The Roofline Model – Visualization of System Limits

Given some characteristics of a given (parallel) code
- does it hit a hardware limit, or
- is there room for improvement?

Focus on full-system limits (of a server / of a cluster)
- main memory bandwidth
- peak performance

Characteristic: **Operational** Intensity
  - How many Flops are executed per Byte transferred to/from memory
  - Can be different to **Arithmetic** Intensity (due to accesses covered by caches)
    - Value is the same for vector triad for large vectors

The Roofline Model – Visualization of System Limits



[Williams et al: Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures]

The Roofline Model – Visualization of System Limits

Easy to use:

- Measure your code (full program / long-running parts): point in diagram
- How far are you from theoretical limits?

Extensions

- Limit vs. compute
  - L1 / L2 / L3 cache bandwidth, network, storage
  - Latency (L1 / L2 / L3 / main memory / network / storage)
  - Memory parallelism: how many accesses can be in-flight?
- Characterizations
  - Sensitivity to access latency
  - How many independent accesses could be triggered with X instruction look-ahead

Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities