Sapienza Università di Roma

Faculty of Information Engineering, Informatics and Statistics
Department of Computer Science

High-Performance Computing

Author:
Simone Lidonnici

6 october 2025

Contents

1 Introduction

In the last decade the application of High-Performance Computing have grown a lot in different fields like engineering, science and AI.
This is because sometimes is too difficult or too expensive to do a real life experiment, so the simulation is the only way to have some data about different phenomenon.
Also the increase in power of CPUs is not enough to keep up with the increase of computing power demand. So instead of improving the single core performance of CPUs the manufacturers decided to implement the **chiplet paradigm**. The chiplet paradigm consist in multiple chips, each with different functionalities and also different silicon dies, in a single CPU.

1.1 Top500 list

The Top500 is a list of all the best supercomputers and data centers in the world, calculated using a specific workload HPL that solve a system of linear equation.
At the start of Top500, in 1993, the majority of supercomputer were monolith, so a single specialized architecture with high power. After the discoveries of microprocessor the best clusters became a group of machines connected between them that work in parallel as a single powerful machine. Now the architectures are heterogeneous, with CPUs augmented by accelerators, the most common being GPUs. In the future the ipotesis is that specialized architecture will come back and the nodes will become even more heterogeneous.

1.2 Components of HPC System

An HPC system is composed by different devices:

- Processors (CPU/GPU): do the computation.
- Memory (RAM): high-speed storage for active computation.
- Interconnects: high-speed network to transfer data.
- Storage
- Software: operating system, schedulers, parallel programming tools.

Today the nodes of an HPC system can be very different between them, with general nodes (CPU based), high-throughput computational nodes (GPU based) and storage nodes with RAM and capacity storage.
The interconnection is also heterogeneous, with 3 main type of communication:

- **Intra-node**: communication between CPU,GPU and local storage. These communication have ultra-low latency ($<$ 100 ns) and high bandwidth ($>$ 100 GBps).
- **Intra-rack**: communication between nodes in the same rack. These communication have low latency (1-2 μs), high throughput and collective operations.
- **Inter-rack**: communication between different racks in a large cluster. Depends on the topology used (dragonfly, fat-tree,...). The major characteristics is the scalability that can handle hundred of thousands of node and also fault tolerance and congestion control.

1.3 AI and Machine Learning takeoff

Recently Machine Learning and AI have had a huge takeoff thanks to the amount of data available (on Internet for example) and the increase of computational power.

Deep Learning needs to do a lot of matrix multiplication, that can also be done on 16-bit floating point because there is no need to have the precision of 32 bit. This caused HPC to move towards this different rappresentation of numbers and also the creation of new standard for 16-bit floating point. The IEEE Standard for 16-bit floating point is divided in 1 bit sign, 5 bit exponent and 10 bit fraction. For 32-bit the IEEE Standard is divided in 1 bit sign, 8 bit exponent and 23 bit fraction. The new standard for 16-bit, the **Google Bfloat16**, is divided in 1 bit sign, 8 bit exponent and 7 bit fraction. It has more range than the IEEE Standard but less precision, also is easier to transform a 32-bit IEEE into a Bfloat16 simply by discarding the last 16 bit (or adding 16 0's in the other way).

1.4 Performance

The performance of a supercomputer are measured in Flop/s, the number of 64-bit floatingpoint operation per second. Of an HPC system we can calculate the **theoretical peak performance** by counting the number of operation completed in a period of time. For example a CPU with 2.1GHz and 24 core that can complete 32 operation per cycle per core has a theoretical peak performance of 1.61 Tflop/s.

To improve the performance of a Single-Node HPC there are 3 main optimization that can be done:

- Algorithms and Data Structures: use efficient algorithms and appropriate data structures.
- Memory: moving data has a big impact on the performance and we need to balance the number of operation (Flop/s) with the transfer from memory to CPU (Words/s).
- Exploit hardware parallelism: there are two main type of parallelism:
  - Data parallelism: exploit Single Instruction Multiple Data (SIMD) by using vectorized instructions. This allow to execute the same operation on different data at the same time.
  - Thread parallelism: use different thread on different cores to cooperate. This is done on Single-Node HPC with OpenMP.

2
CPU Architecture

2.1 Istruction Set Architecture (ISA)

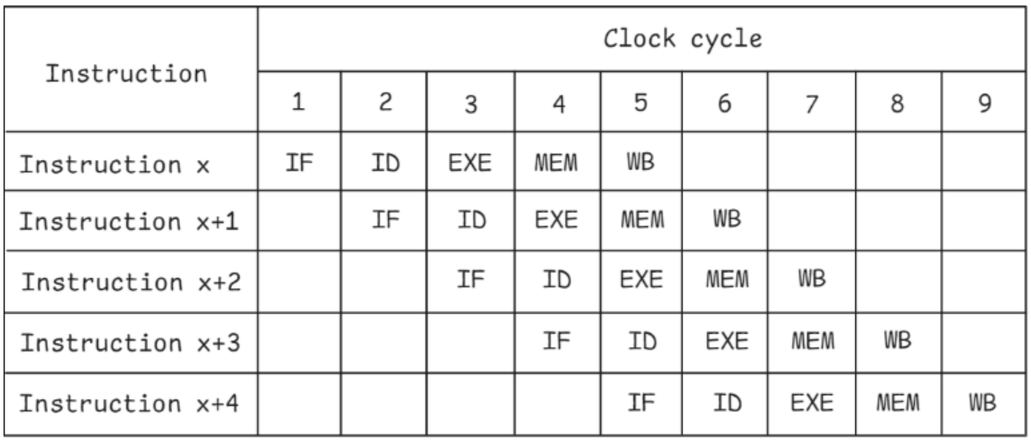
The **instruction set architecture (ISA)** is the contract between the software and the hardware, which defines the rules of communication. There are two main categories:

- Register-based: where data can be accessed only using load and store istructions, for example ARM.
- Register-memory: where operations can be done with data in registers and memory, for example x86.

Every architecture extends the base instruction set with other instructions, for example different number rappresentation, vector processing and matrix instructions.

2.2 Pipeline

Pipeline is used in CPUs to make multiple instructions overlap during the execution. The basic pipeline execute instruction in 5 stages (fetch, decode, execute, memory and write), but high-end processor have 10-20 pipelane stages.



Instruction	Clock cycle								
	1	2	3	4	5	6	7	8	9
Instruction x	IF	ID	EXE	MEM	WB				
Instruction x+1		IF	ID	EXE	MEM	WB			
Instruction x+2			IF	ID	EXE	MEM	WB		
Instruction x+3				IF	ID	EXE	MEM	WB	
Instruction x+4					IF	ID	EXE	MEM	WB

The **throughput** of a pipelined CPU is defined as the number of instructions that exit the pipeline per unit of time. The **latency** for any given instruction is the total time through all the stages of the pipeline. The time required to move an instruction from one stage to the next is the **clock cycle** of the CPU.

2.2.1 Pipeline hazards

The hazards that can appear in a pipeline can be of three types:

- **Structural hazard**: caused by resurce confict. The only way to solve them is replicate hardware parts, like execution units or decoders.
- **Control hazard**: caused by a change in the program flow, for example a branch.
- **Data hazard**: caused by data dependencies in the program.

Data hazards can be classified into three types:

- Read-after-Write (RAW):

  ```
  R1 = R0 ADD 1
  R2 = R1 ADD 2
  ```

  Can be resolved with data forwarding/bypassing. Bypassing is more effective longer is the pipeline.

- Write-after-Read (WAR):

  ```
  R1 = R0 ADD 1
  R0 = R2 ADD 2
  ```

  Occur only in superscalar of Out-of-Order processors and can be resolved by **Register renaming**.

- Write-after-Write (WAW):

  ```
  R1 = R0 ADD 1
  R1 = R0 MUL 3
  ```

  Can be also resolved by register renaming.

Most modern CPUs are **Out of Order (OoO)** processors, means that sequential instructions can enter the execute stage in any order. An instruction is **retired** when the result is visuble in the architectural state, so in a register visible from outside. CPUs have physical registers for temporal data and architectural registers where the architectural state is visible. A CPU must retire the istructions in program order, this is called **in-order retire**.
Modern CPUs are also **superscalar**, so they can issue more than one instruction for clock cycle, the maximum number of instruction issued in the same clock cycle is defined as **issue width**.

2.3 Static scheduling

Compiler divide the instructions in groups that can be issued on the same clock cycle (they don't have dependencies and all the resources are available), called **issue packet**. An issue packet is essencially a very long instruction word (VLIW).

This compilers have to analyze the data dependencies to make the correct groups. They have problems with memory access conflict and also with instruction that requires unpredictable amount of time (memory latency or cache miss).
This type of compilers are very good in some cases, like machine learning where the data is accessed in a predictable way.

2.4 Dynamic scheduling

In the dynamic scheduling CPUs use register renaming (Tomasulo algorithm) to eliminate data dependencies of WAR and WAW types. It not resolve the RAW dependancies, known as **dependencies chain**, that are also fund in loop where an iteration depends on the previous iteration.
A **Reordered buffer (ROB)** is used to insert instruction, that can be executed out of order but retired in order. The number of entries determines how far the hardware can look to schedule instruction independent between each other. It also use the Tomasulo algorithm to do register renaming when instruction enter the ROB. The ROB place the instruction in the Reservoir Station.
In the **Reservoir Station (RS)** the instructions wait the operands to be available. In the RS the instruction are executed in any order and also support speculative execution.

2.4.1 Speculative Execution

To avoid the performance loss by the control hazards is used a technique called hardware branch prediction. The CPU predict the most likely path of the branch and start executing the istruction of that path. If the prediction is correct, the performance loss is avoided, but if it's wrong all the result of the speculative execution are voided. The ROB allows an easy roll back in the case of wrong prediction.
There are three type of branches:

- Unconditional jump and direct calls: always taken.
- Indirect calls and jumps: have many targe, generated by switch or function pointer.
- Conditional branches: only two outcome possible, taken or not taken. Are divided in two types:
  - Forward: generated by if-else statement and have a high chance of not being taken because frequenly they are error checking.
  - Backward: generated by loop and are usually taken.

All prediction mechanisms exploit two principles:

- Temporal correlation: if a branch is taken is a good prediction if it will be taken again. This is also known as local correlation.
- Spatial correlation: close branches may resolve in a correlated manner (a preferred path of execution). This is also known as global correlation.

The branch prediction unit (BPU) is composed of:

- Branch target buffer (BTB): which caches the target addresses for every taken branch.
- Pattern History Table (PHT): Stores past branch behavior to predict future outcomes. Often implemented with 2-bit counters.
- Return Address Stack (RAS): used for predicting the return address of function.

2.5 Thread-level parallelism

There are three main techniques to exploit Thread-Level Parallelism:

- Multicore systems
- Simultaneous multithreading
- Hybrid architectures

2.5.1 Multicore systems

The idea of Multicore design is to replicate multiple processor cores on a single chip and let them serve different programs at the same time.
Since each core generates heat when it's working multicore processors with more cores usually reduce clock speeds due to this heat dissipation problem.
Cores in a multicore system are connected to each other and to shared resources, such as cache and memory controllers. This shared resources frequently become the source of performance issues in a multicore system.

2.5.2 Simultaneous multithreading (SMT)

SMT allows multiple software threads to run in the same clock cycle on the same physical core using shared resources.
To support SMT, a CPU must replicate the architectural state (program counter, registers) to maintain thread context. Other CPU resources can be shared.
The problem with SMT is caused by competition for L1 and L2 caches. Since they are shared between multiple logical cores, they could lack space and force exit of data that will be used by another thread in the future. SMT makes harder to measure the performance of an application that run on SMT core.

2.5.3 Hybrid architecture

There are hybrid CPU in which different types of cores are put in the same processor. Typically, more powerful cores are coupled with relatively slower cores.
The scheduling in hybrid architectures follows some considerations:

- Do not use big cores for background work.
- Use smaller cores for low importance task and bigger core for high important tasks.

- When assigning a new task, use an idle big core first. In the case of SMT, use big cores with both logical threads idle. After that, use idle small cores. After that, use sibling logical threads of big cores.

2.6 Memory hierarchy

Programs access a small proportion of their address space at any time, following the **principle of locality**, that can be intended in two ways:

- Temporal locality: Data accessed recently is likely to be accessed again soon.
- Spatial locality: Data near those accessed recently is likely to be accessed soon.

To take advantage of locality every time an item is accessed the nearby items are copied from disk to DRAM and from DRAM to SRAM.
In the DRAM the data is stored as a charge in a capacitor, so it must be periodically refreshed. The data is stored in a rectangular way. The access and refresh can be done in parallel on an entire row, also consecutive accesses can be done in burst (on the same clock cycle). Is possible to do DRAM banking, allowing Simultaneous access to multiple DRAMs.
HBM (High Bandwidth Memory) is a new type of CPU/GPU memory that vertically stacks memory chips. HBM shortens the distance data needs to travel to reach a processor. HBM has a memory bus of 1024 bits for each HBM stack. This allows HBM to achieve ultra-high bandwidth.

2.6.1 Cache

The cache is the level of the memory hierarchy closest to the CPU, and can have different level of assoctiveness:

- Direct Mapped: every address in memory can be inserted in only one cache block, determined by his address.
- Fully associative: every address can be inserted in every cache block. Is expensive to search because needs to view every entries.
- Set associative: there are different sets with n entries per set. Is easier to search because needs only to check the n entries in the same set.

The performance of a cache can be measured in a simplified way with the formula:

$$\text{Memory stall cycles} = \text{Memory accesses} \cdot \text{Miss rate} \cdot \text{Miss penalty}$$

Example:
We have a cache with:

- I-cache miss rate = 2%
- D-cache miss rate = 4%
- Miss penalty = 100 cycles

- Base CPI (clock per instruction)= 2
- Load and stores are 36% of instructions

The stall cycles per istructions will be:

$$\text{I-cache} = 0.02 \cdot 100 = 2 \text{ cc}$$
$$\text{D-cache} = 0.36 \cdot 0.04 \cdot 100 = 1.44 \text{ cc}$$

So the actual CPI of the cache:

$$\text{CPI} = 2 + 2 + 1.44 = 5.44 \text{ cc}$$

Another important parameter for performance is the average memory access time (AMAT), calculated:

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \cdot \text{Miss penalty}$$

In modern architectures there are level of caches, often 2 levels but sometimes also 3:

- L-1 cache: is attached to the CPU and is small but fast. Focuses on minimal hit time.
- L-2 cache: is bigger and slower than L-1 but still faster than main memory. Focus on low miss rate.
- L-3 cache: is a shared cache that exists in some high-end systems.

The first 2 levels are private to every core.
Example:
We have a L-1 cache with:

- I-cache miss rate = 2%
- Main memory access time = 100 ns
- Base CPI (clock per instruction)= 1
- Clock rate = 4 GHz

And a L-2 cache with:

- Access time = 5 ns
- Global miss rate = 0.5%

So the miss penalties are:

- L-2 Hit = $\frac{5 \text{ ns}}{0.25 \text{ ns}}$ = 20 cycles
- L-2 Miss = $\frac{100 \text{ ns}}{0.25 \text{ ns}}$ = 400 cycles

The actual CPI of the caches:

$$\text{CPI} = 1 + 0.02 \cdot 20 + 0.005 \cdot 400 = 3.4$$

Out-of-order CPUs can execute instructions during cache miss, so pending store stays in load-/store unit, dependent instructions wait in reservation stations and independent instructions continue. Effect of miss depends on program data flow.
The are three main sources of cache miss:

- Compulsory misses (aka cold start misses): first access to a block.
- Capacity misses: due to finite cache size where a replaced block is later accessed again.
- Conflict misses (aka collision misses): in a non-fully associative cache due to competition for entries in a set. Would not occur in a fully associative cache of the same total size.

One method to avoid cache misses is to prefetch data into caches prior to when the pipeline demands it. The miss penalty can be mostly hidden if the prefetch request is issued sufficiently ahead in the pipeline. Most CPUs provide implicit hardware-based prefetching and explicit software prefetching.

Hardware prefetchers observe the behavior of a running application and initiate prefetching on repetitive patterns of cache misses. However, hardware prefetching works for a limited set of commonly used data access patterns.

Software memory prefetching complements prefetching done by hardware. Developers can specify which memory locations are needed ahead of time via dedicated instruction. Compilers can also automatically add prefetch instructions into the code to request data before it is required.

An important problem with caches is the cache coherence, when one CPU modify data in his cache and the same data is also in cache of another CPU. To solve this problem a common protocol used is the **invalidating snooping protocol**, in which every time a CPU modify data for a value it sends a message on a bus to invalidate the old value. So, other CPUs with the same value in cache have to update the value from memory again.

The data are invalidated in lines, so even if the second CPU wants to read a value that is not the same as the one modified from the first CPU but in the same line, it has to reload the line from memory. This problem is called **false sharing**.

2.6.2 Main Memory

Main memory uses DRAM (Dynamic Random Access Memory) modules that supports large capacities at reasonable cost points. A DRAM module is organized as a set of DRAM chips. The memory rank describes how many sets of DRAM chips exist on a module.

Multiple DRAM modules can also increase memory bandwidth. The multi-channel architectures increase the width of the memory bus, allowing DRAM modules to be accessed simultaneously. Interleaving spreads adjacent addresses within a page across multiple memory devices.

The **interleaving granularity** controls how big each striped block is before switching to the next channel:

- Fine granularity (cache line or 4KB page level): maximizes parallelism and bandwidth utilization. Best for streaming, bandwidth-heavy HPC workloads (CFD, linear algebra, tensor ops).
- Coarse granularity (2MB, 1GB): improves locality (important for NUMA-aware applications like MPI). Best when software is explicitly NUMA/hugepage-aware (e.g., HPC MPI jobs pinned to sockets).

2.6.3 Virtual Memory

The **virtual memory** allows sharing the physical memory attached to a CPU with all the processes executing on it. Provides a protection mechanism that prevents access to the memory allocated to a given process from other processes. Provides relocation, which is the ability to load a program anywhere in physical memory without changing the addresses in the program.

Virtual memory is divided into pages and the page table can be either single-level or nested. Of the 64 bit of a virtual address the 16 most significant bits are not used (Some applications use those unused bits to keep metadata, also known as pointer tagging*).

Failure to provide a physical address mapping is called a page fault. It occurs if the operating system is committed to allocating a page or an accessed page was swapped out to disk and is not currently stored in RAM.

The **translation lookaside buffer (TLB)** caches the most recently used translations. TLBs are often designed as a hierarchy of L1 ITLB (Instructions), and L1 DTLB (Data), followed by a shared (instructions and data) L2 STLB. To speed up the handling of TLB misses, CPUs have a mechanism called a hardware page walker. Such a unit can perform a page walk directly in hardware by issuing the required instructions to traverse the page table.

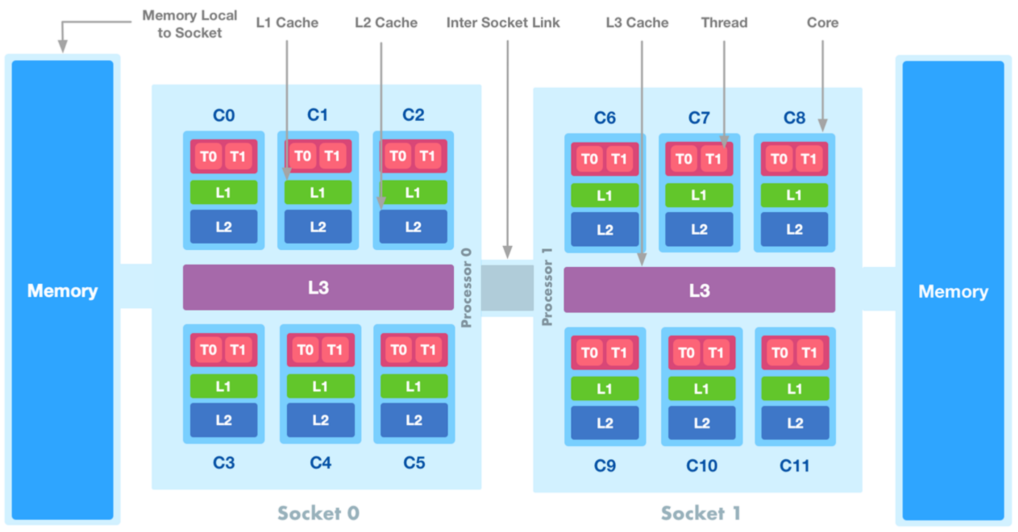
Virtual addresses can be saved in cache, the way in which they are saved depends on the cache level. The L-1 cache is Virtually Indexed Physically Tagged and the L-2 and L-3 are Physically Indexed Physically Tagged.

Having a small page size makes it possible to manage the available memory more efficiently and reduce fragmentation, but it requires having more page table entries to cover the same memory region. Huge Pages (2MB or 1GB) drastically reduce the pressure on the TLB hierarchy since fewer TLB entries are required. It greatly increases the chance of a TLB hit. The downsides of using huge pages are memory fragmentation and, in some cases, nondeterministic page allocation latency because it is harder for the OS to find a contiguous chunk of memory. If this cannot be found, the OS needs to reorganize the pages, resulting in a longer allocation latency.

2.7 Compute Node Architecture

A compute node can be considered a basic building block of computing systems today. A node consists of one or more sockets. A socket is hardware on the motherboard where the CPU is mounted. A processor has components like ALU, FPU, Registers, Control Unit, and Caches are collectively called a core of a processor.

Each socket has main memory which is accessible to all processors in a node. If the processor can access its own socket memory faster than the remote socket then the design is referred to as Non-Uniform Memory Access (NUMA).



The **uncore** in an x86 processor refers to the components and functions of the CPU that are not part of the individual processing cores but are integrated onto the same silicon die. Key Components and Functions of the uncore are:

- Last-Level Cache (LLC): typically the L3 cache, which is shared among all processor cores.
- Memory Controllers (MBox): these manage how the CPU accesses the system's main memory (RAM).
- Interconnects (PBox): these provide high-speed communication pathways between the cores themselves, and between the CPU and other system components.
- PCI Express (PCIe) and I/O Controllers: these handle high-speed peripheral connections, such as graphics cards and storage devices.
- System Configuration Controller (UBox): this component, along with a router (RBox), manages connections between various uncore elements like the PBox, CBox, and MBox.
- Power Control Logic (PWR): the uncore plays a significant role in power management, including dynamic uncore frequency scaling to optimize for both energy efficiency and performance based on workload.

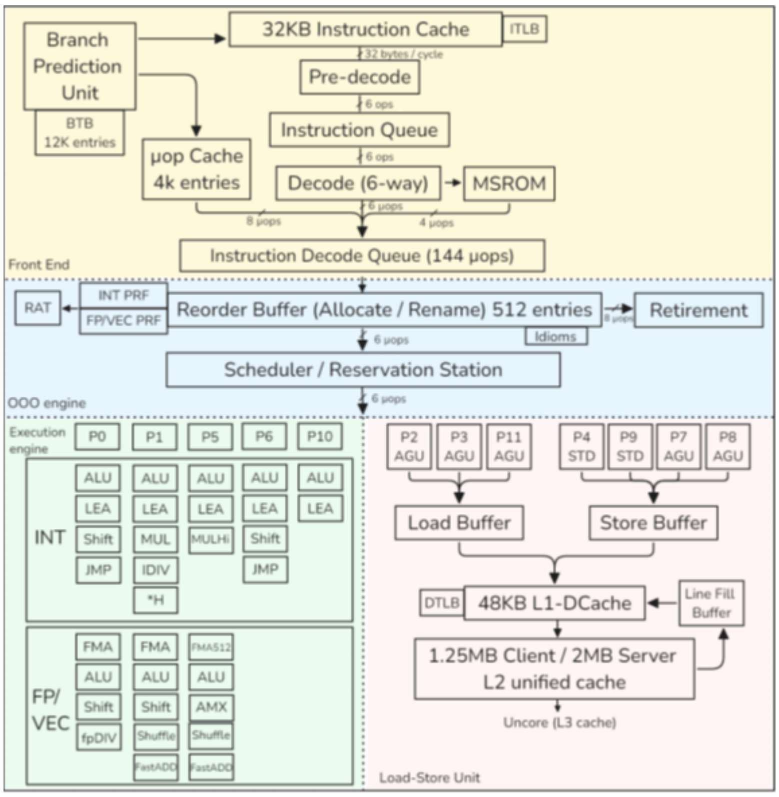
2.8 Core design

We take as example the implementation of Intel's 12th-generation core, Golden Cove. The core is split into 4 parts:

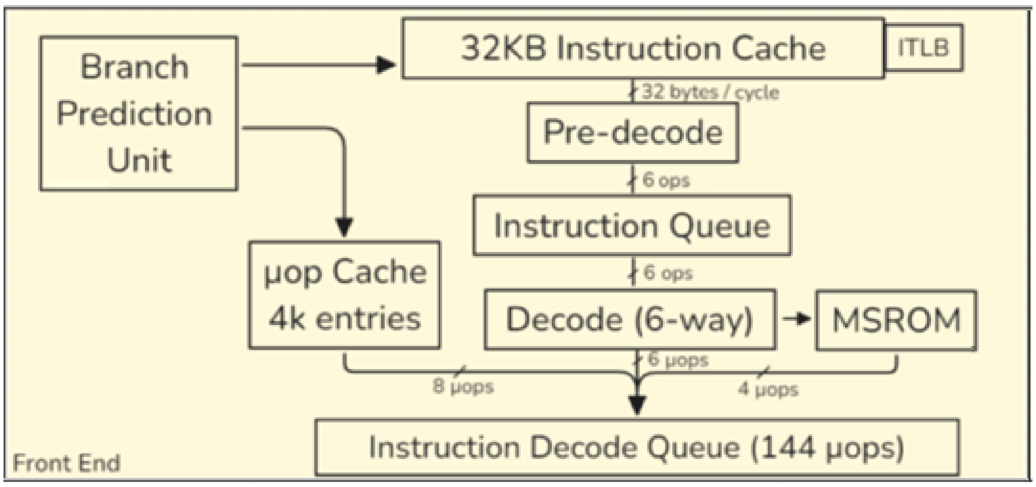
- Front-end: in-order frontend that fetches and decodes x86 instructions into micro-ops (μops).
- Back-end: 6-wide superscalar, out-of-order backend.
- Execution engine: perform multiple operations in parallel.

- Load-Store Unit: interact with the CPU memory.

The Golden Cove core supports 2-way SMT. It has a 32KB L1 I-cache, a 48KB L1 D-cache and a 1.25MB L2 cache.



2.8.1 Front-end



The BPU predicts the target of branch instructions and provide the next instruction to fetch based on this prediction. The frontend fetches 32 bytes per cycle of x86 instructions from the L1 I-cache. This is shared among the two threads.

The pre-decode stage marks the boundaries of the variable length x86 instructions and moves up to 6 instructions (macroinstructions) to the Instruction Queue.

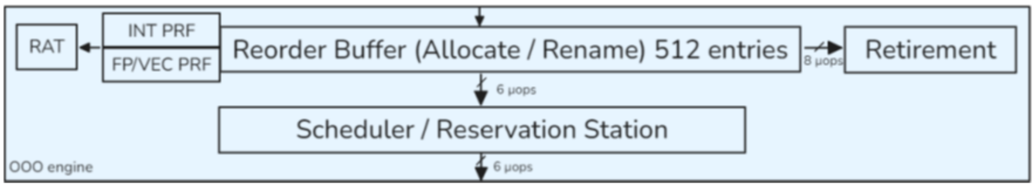
The 6-way decoder converts the complex macro-ops into fixed-length μops. Decoded μops are queued into the Instruction Decode Queue.

The μop cache, also called Decoded Stream Buffer (DSB) stores the more recently used decoded instruction. The DSB can provide 8 μops per cycle and can hold up to 4K entries. The DSB reduces decoder power consumption, latency for instruction fetch and front-end bottlenecks.

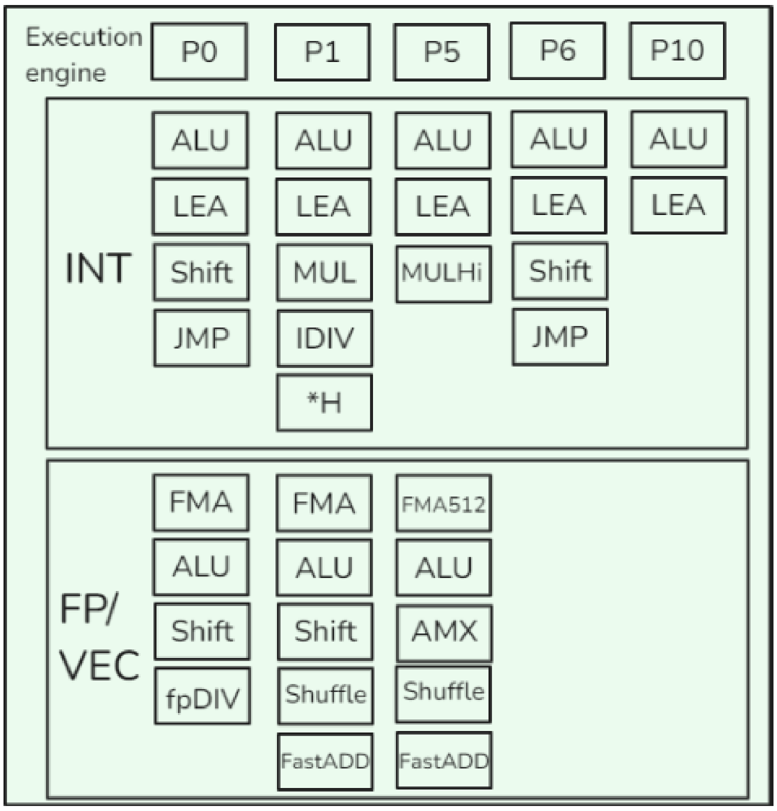
The MSROM (Microcode Sequencer) manages some “complex” or “rare” x86 instructions. Those sequences are predefined microprograms stored in the MSROM. So the MSROM is essentially a library of μop routines burned into the CPU.

If SMT is enabled the two SMT threads alternate every cycle to access the instruction fetch, the pre-decode, the instruction queue and the decoder. The DSB and the Instruction Decode Queue is partitioned (72 entries for each thread) between the two threads.

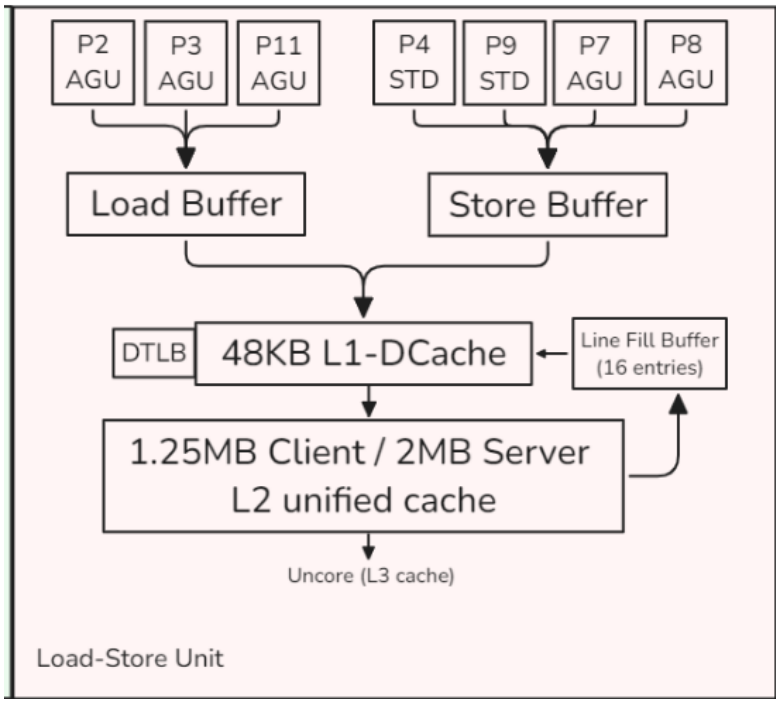
2.8.2 Back-end


RAT
INT PRF
FP/VEC PRF
Reorder Buffer (Allocate / Rename) 512 entries
8 µops
Retirement
6 µops
Scheduler / Reservation Station
OOO engine
6 µops

2.8.3 Execution engine



2.8.4 Load Store Unit


P2 AGU
P3 AGU
P11 AGU
P4 STD
P9 STD
P7 AGU
P8 AGU
Load Buffer
Store Buffer
DTLB
48KB L1-DCache
Line Fill Buffer (16 entries)
1.25MB Client / 2MB Server L2 unified cache
Uncore (L3 cache)
Load-Store Unit