Hardware technology
Optimization training at CINES

Adrien CASSAGNE
adrien.cassagne@cines.fr

2014/09/30
Contents

1. Main processor principles
2. Several levels of parallelism
3. Multi-node overview
4. Presentation of accelerators
Basically, what is a processor?

- Memory and compute units
  - Memory is a way to load/store data (input/output)
  - Compute units allow us to transform data
Basically, what is a processor?

- In fact we can be more precise, a CPU is composed by:
  - A control flow (if, switch, instructions placement, ...)
  - Logical units, LU (==, !=, >, <, ...)
  - Arithmetic units, AU (+, *, -, /, ...)

More precise vision of a processor and its memory
Clock cycle and frequency

- At each clock cycle the CPU is able to perform an elementary task.
- The frequency is the number of cycles per second (in Hertz).
- Modern CPUs clock rates range between 1GHz and 4Ghz, this is very fast!
  - $1\text{Ghz} = 10^9\text{Hz}$
- RAM does not operate at the same frequency as the CPU: between 0.5Ghz and 1.6GHz.
  - This is slower than the CPU!
  - How can the CPU operate fully if memory is slower? We will see that later on...
Okay the CPU is fast, but how is this technically achieved?

- First, you have to know the basic cycle of an instruction:
  1. Fetch instruction: the instruction is copied from the memory
  2. Decode instruction: the instruction is interpreted by the CPU
  3. Execute instruction: the instruction is executed

- Let’s take a look on how it works on an old school CPU (a slow one):

  ![Pipelining model diagram]

  No pipeline
Okay the CPU is fast, but how can it technically achieve that?

- According to the previous slide, we can divide an instruction in 3 sub-instructions.
- We can divide an instruction in sub-instructions which take an equal amount of time.
- This is the pipelining strategy:
  - Uses internal parallelism.
  - The number of tasks that can be carried out in parallel is the number of pipeline stages.

3-stage pipeline:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Instruction</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1</td>
<td>IF 1, ID 1, EX 1</td>
<td></td>
</tr>
<tr>
<td>Cycle 2</td>
<td>IF 2, ID 2, EX 2</td>
<td></td>
</tr>
<tr>
<td>Cycle 3</td>
<td>IF 3, ID 3, EX 3</td>
<td></td>
</tr>
</tbody>
</table>

Time
There is a time before the pipeline is optimal: this is called "the pipeline latency"

- 2 cycles here

If we do not consider this latency, we are 3 times faster with the pipeline strategy (3-stage pipeline)

3-stage pipeline, yellow is optimal
Pipelining model

- Today CPUs have between 10-stage and 20-stage pipelines but the principle is still the same
- Pipelining is efficient but what about branches (if statements)?
  - They are very problematic and sometimes "if" statements can destroy pipeline efficiency
  - The branch predictor mechanism tries to minimize this effect...

3-stage pipeline, yellow is optimal
Today CPUs are superscalar

They can do same sub-instructions in parallel, this is also called Instruction Level Parallelism (ILP)

- Between 3-way and 6-way superscalar
- So, now we can achieve more than one operation in one CPU clock cycle
  - In fact we can achieve almost 5 instructions at each cycle...
  - We will see that in more detail later

3-stage pipeline, 2-way superscalar
In modern CPUs the pipeline is split into two main parts:
- Front end: fetches and decodes instructions
- Back end or execution engine: executes instructions

Front end and back end in our 3-stage pipeline
Sandy Bridge pipeline: front end

(Picture from Anandtech)
Sandy Bridge pipeline: back end

(Picture from Anandtech)
Out-of-Order execution

- Depending on ports availability, the processor can change the order of execution of the instructions: Out-of-Order execution
  - Ports utilization maximisation
  - Sometimes it’s difficult to understand what the CPU really does (hard to predict)

```c
1 int a, b, c, d, e, f, g, h;
2 c = a + b; // first instruction to be executed
3 \n3 // no dependency
4 e = c * d; // third instruction to be executed
5 \n5 // dependency with the c variable
6 h = f * g; // second instruction to be executed
7 \n7 // no dependency
```

Code example, Out-of-Order execution
Previously we saw how fast a modern CPU is and how it manages instructions.

But there is still a problem: how to feed the beast?

- Remember, memory is very slow compared to the compute capacity of a CPU.
- And a CPU needs input data in order to compute results...

Any ideas on how to solve this problem?
Add a layer of faster memory between CPU and RAM: cache memory!

- Faster memory is expensive, and takes a lot of physical space
- So, this memory is way smaller than the RAM
- 3 cache levels (on-chip memory):
  - L1 is the fastest but also the smallest (32 Ko): (access time approx. 1 cycle)
  - L2 is slower than L1 but it is also bigger (256 Ko): (access time approx. 10 cycles)
  - L3 is slower than L2 but much faster than RAM and it is bigger than L2 (3 Mo to 20 Mo): (access time approx. 30 cycles)

- RAM latency is approximately 100 cycles
First load with memory hierarchy

Load from RAM: 100 cycles
First load with memory hierarchy

Copying loaded data into L3
First load with memory hierarchy

Load from L3: 30 cycles
First load with memory hierarchy

Central Process Unit (CPU)

control flow

LU

AU

L1

L2

L3

Random Access Memory (RAM)

memory

Copying loaded data into L2
First load with memory hierarchy

Central Process Unit (CPU)

control flow
LU | AU

L1 L2 L3

Random Access Memory (RAM)

memory

Load from L2: 10 cycles
First load with memory hierarchy

Copying loaded data into L2
First load with memory hierarchy

Load from L1: 1 cycle
Second load with memory hierarchy

- Here we can see the benefit of the cache hierarchy
- If the data are in the L1 cache, the load takes only 1 cycle!
- Data reuse is required if we want to feed the CPU
- Data are loaded by lines of words, so contiguous words accesses are fast
Store with memory hierarchy

- Store data from CPU to RAM
- Stored data are also put in caches
- Load data we have recently stored is efficient (this data is in the L1)
Several level of parallelism inside a CPU

- The simplest way to increase performance is to increase the clock frequency of the processor
  - No code modification required
  - But energy consumption directly depends on the clock frequency \((e \approx f^2)\)
  - And we can’t increase clock frequency indefinitely
- This is why we prefer making things in parallel
  - No impact on clock frequency
  - CPU performance is also improved this way, but sometime we have to modify the code!
Several levels of parallelism inside a CPU

There are two different types of parallelism:

- **Automatic parallelism**, which can be indirectly controlled by user:
  - Pipeline
  - Instruction Level Parallelism (superscalar processors)

- **Manual parallelism**, directly controlled by the user:
  - Vectorization
  - Simultaneous multi threading or Hyper Threading
  - Multi-core architecture
Vectorization

- Traditionally an instruction works on scalar values
- But there is another type of instruction that allows us to work on vectors
- This strategy is the less energy consuming!
- This is also called SIMD (Single Instruction Multiple Data) instructions
- Modern CPUs tend to become more and more SIMD (because of the energy efficiency)
Vectorization

- A scalar instruction works with standard scalar registers

![Scalar instruction diagram]

- A vector instruction works with special vector registers

- Performing a vector instruction takes the same amount of cycles as a standard scalar instruction

![Vector instruction diagram]
Sandy Bridge processors have AVX-256 bits instructions (Advanced Vector Extensions)

- 256 bits is the size of AVX registers
- 256 bits = 4 double precision numbers (double)
- 256 bits = 8 single precision numbers (float, int)

- One simple precision number requires 32 bits (or 4 bytes)
- One double precision number requires 64 bits (or 8 bytes)
New hardware supports the very famous Hyper Threading technology (same as SMT), but what is it really?

- Possibility to manage several hardware threads in a processor core simultaneously
  - Without adding arithmetic and logic units (max. reachable performance does not change with SMT)
  - Increase the pressure on ports
  - This is a mechanism to maximize CPU usage
    - Gain depends on the problem and the implementation
    - Sometimes this is useless
Simultaneous Multi Threading

- Today, Intel cores can manage until 2 hardware threads
- This reduces bubbles in the pipeline

2-way Simultaneous Multi Threading (Wikipedia)
Multi-core architecture

- Current architectures are multi-core
- The idea is to put processors together
  - Each core has a lot of independence
  - All the previous features are available in each core
  - In fact, this is not totally true: cores also share some resources...
    - L3 cache is shared by all the cores of a processor!
    - Memory (RAM) management is also shared
- To use these cores fully we have to create multiple threads (multi-threading) or to create multiple processes
  - Be careful, threads can be used on different cores but also in a same core with SMT mechanism
  - Unlike SMT, multi-core architecture really increases the maximum reachable performance (by the number of cores)
Multi-core architecture

Central Process Unit (CPU)

Quad core processor architecture
Contents

1. Main processor principles
2. Several levels of parallelism
3. Multi-node overview
   - Basics
4. Presentation of accelerators
How to make a supercomputer?

- In the previous sections we described the mechanisms inside a node (inside a single computer).
- Sometimes a node is not powerful enough to simulate a complete phenomena.
- So, what can we do?
  - Apply the *Divide and conquer* strategy!
  - Split the work and distribute it over several nodes.
  - We have just created an other level of parallelism: the node.
- How to use node parallelism is not in the range of this course.
A very basic supercomputer

Very basic supercomputer architecture

- Here, the switch inter-connects all nodes
- Cluster max. performance = 8 x node performance
What is an accelerator?

- Basically, this is something made to decrease the restitution time of codes
- An hardware separated from the CPU
  - Today, accelerators are connected to the CPU through a the PCI-Express bus
- A very parallel hardware (even more than CPUs)
- It has its own memory

![Diagram of an accelerator in a node]
Graphical Process Unit (GPU)

- Built for image calculations
- Very parallel architecture
- Less control units than traditional CPU but more compute units
- Faster global memory than CPU memory (RAM)
- Very attractive performance/energy ratio (much better than the CPU ratio)
- Adapted to massively parallel scientific computations
- But requires a specific code to work well!
GPU architecture

Nvidia Kepler architecture (full GK110)
Xeon Phi

- First massively parallel accelerator from Intel
- Built specifically for HPC
- Very young architecture made from the union of many small x86 processors
  - Based on Pentium 3 (or Atom) architecture
  - But with very large AVX 512 bits instructions: remember those instructions are very interesting for energy efficiency!
  - Uses 2-way SMT
- Faster memory than traditional CPU memory
- Runs a real operating system based on Linux!
Xeon Phi architecture

Intel Knights Corner architecture