Introduction to Computation Technologies in Deep Learning

Kai Jia
jiakai@megvii.com

Megvii Inc.

May. 30, 2018

Instruction: The Hardware/Software Interface

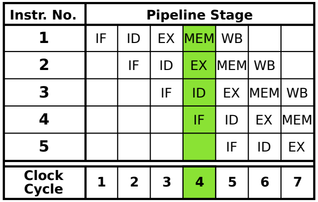
What is a program?

```
int sum(int *x) {
    return x[0] + x[1];
}
```

```
0000000000000000 <sum>:
    0:     8b 47 04                  mov     0x4(%rdi),%eax
    3:     03 07                     add     (%rdi),%eax
    5:     c3                        retq
```

Modern CPU Technologies

- **Pipeline** [1]



Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**

- **Superpipelining** increases stage number and simplifies each stage
- **Superscalar** dispatches multiple instructions to implement instruction-level parallelism
- **Out-of-order execution** executes according to availability of input data rather than original program order

[1] Image from `https://en.wikipedia.org/wiki/Instruction_pipelining`

Memory Hierarchy

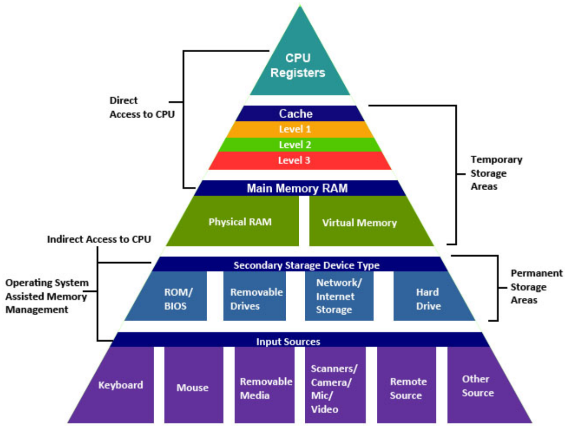


Image from `http://cse1.net/recaps/4-memory.html`

RAM Implementation
Static Random-Access Memory (SRAM)

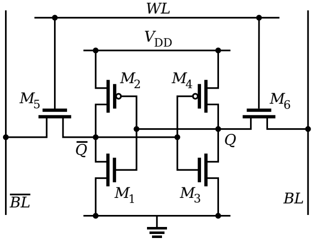


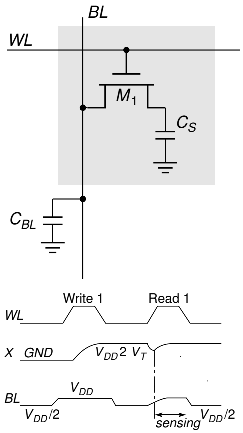
Image from `https://en.wikipedia.org/wiki/Static_random-access_memory`

Advantages:

1. Fast
2. Low power consumption
3. No refresh circuit

RAM Implementation

Dynamic Random-Access Memory (DRAM)



Advantages:

1. High density
2. Cheap

Refresh: periodically read blocks and write back.

Image from `http://docencia.ac.upc.edu/master/MIRI/NCD/docs/04-Memory%20Structures-2.pdf`

Cache Hierarchy

A hierarchical design for better trade-off between memory capacity and latency.

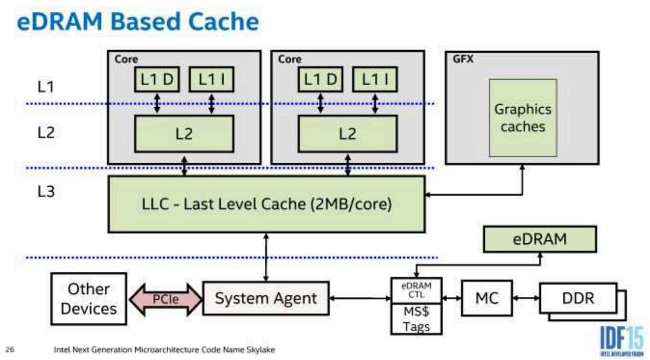


Image from

https://www.anandtech.com/show/9582/intel-skylake-mobile-desktop-launch-architecture-analysis/5

Cache Hierarchy

A hierarchical design for better trade-off between memory capacity and latency.

Core i7 Xeon 5500 Series

L1 hit	~ 4 cycles
L2 hit	~ 10 cycles
L3 hit line unshared	~ 40 cycles
L3 hit, shared line in another core	~ 65 cycles
L3 hit, modified in another core	~ 75 cycles
Remote L3	$\sim 100 - 300$ cycles
Local DRAM	~ 60 ns
Remote DRAM	~ 100 ns

source:

`https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf`

CPU Cache Structure

- Cache line

tag	data block	flag bits (valid, dirty)

- Indexing

tag (40bit)	index (6bit)	block offset (6bit)

CPU Cache Structure

- Cache line

tag	data block	flag bits (valid, dirty)

- Indexing

tag (40bit)	index (6bit)	block offset (6bit)

- Associativity

  Number of different tags to be kept under the same index

CPU Cache Structure

- Cache line

tag	data block	flag bits (valid, dirty)

- Indexing

tag (40bit)	index (6bit)	block offset (6bit)

- Associativity
  Number of different tags to be kept under the same index
- VIPT Addressing
  Virtually indexed, physically tagged (VIPT)
  - Solve aliasing problem
  - Simultaneous cache and TLB lookup

Interesting reading: `http://igoro.com/archive/gallery-of-processor-cache-effects/`

CPU Cache Structure

Example

```
$ grep -m1 name /proc/cpuinfo
model name      : Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz

$ cat /sys/devices/system/cpu/cpu0/cache/index0/{size,ways_of_associativity,coherency_line_size}
32K
8
64
```

- block offset: $\log_2 64 = 6\text{bit}$
- index: $\log_2(32\text{KiB}/64\text{B}/8) = 6\text{bit}$
- tag: $52 - 6 - 6 = 40\text{bit}$ (48-bit virtual memory and 52-bit physical memory)

Note that *block offset* and *index* together take 12 bits, which is equal to page size (4KiB), so VIPT can be easily implemented.

SIMD
Single instruction, multiple data

Store multiple data items in one register and process them in a single instruction.

Calculation of theoretical FLOPS[2]

$$FLOPS = f \cdot w \cdot IPC$$

f :	frequency
w :	SIMD width (number of floats per register)
IPC :	SIMD instructions per cycle

[2] floating point operations per second

SIMD

Single instruction, multiple data

Example

Intel® CPUs usually have $IPC = 2$. However if FMA is supported, IPC should be counted as 4 since 2 FMA instructions is essentially 4 floating point operations.

Intel® Xeon® Platinum 8180M[2]

# of Cores	28
Processor Base Frequency	2.50 GHz
Max Turbo Frequency	3.80 GHz
# of AVX-512 FMA Units	2

$$\begin{aligned} FLOPS &= 3.8 Gcyc/s \times 4 instr/cyc \ \times 16 float/instr \\ &= 243.2 GFLOPS \end{aligned}$$

$$FLOPS\_TOT = FLOPS \times 28 = 6.8 TFLOPS$$

[2] data available at
https://ark.intel.com/products/120498/Intel-Xeon-Platinum-8180M-Processor-38_5M-Cache-2_50-GHz

A MatMul Example

```
void matmul(float *a, float *b, float *c, int n) {
    for (int i = 0; i < n; ++ i) {
        for (int j = 0; j < n; ++ j) {
            float sum = 0;
            for (int k = 0; k < n; ++ k) {
                sum += a[i * n + k] * b[k * n + j];
            }
            c[i * n + j] = sum;
        }
    }
}
```

A MatMul Example

```
void matmul(float *a, float *b, float *c, int n) {
    for (int i = 0; i < n; ++ i) {
        for (int j = 0; j < n; ++ j) {
            float sum = 0;
            for (int k = 0; k < n; ++ k) {
                sum += a[i * n + k] * b[k * n + j];
            }
            c[i * n + j] = sum;
        }
    }
}
```

Swap the loops on j and k

NVIDIA GPU
A single instruction, multiple thread architecture

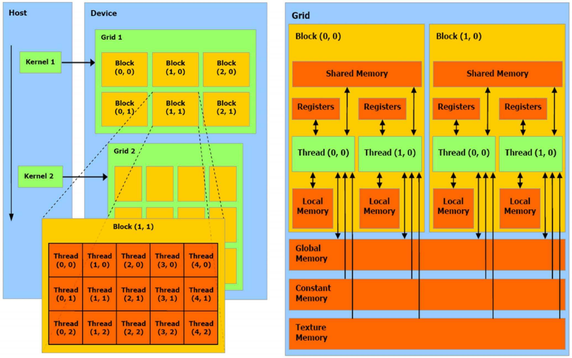


Image from[3]

[3]Marco Nobile et al. "cuTauLeaping: A GPU-Powered Tau-Leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems". In: 9 (Mar. 2014), e91963.

NVIDIA GPU

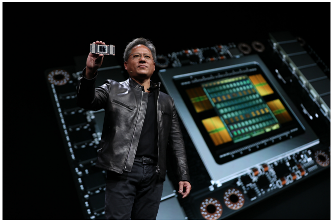
A single instruction, multiple thread architecture

```
__global__ void add(float *a, float *b, float *c, int n) {
    int id = blockIdx.x*blockDim.x+threadIdx.x;
    if (id < n)
        c[id] = a[id] + b[id];
}
```

- **Memory Coalescing**
  Adjacent threads access adjacent memories simultaneously
- **Parallelism**
  Divide the total work among many tiny threads

NVIDIA GPU

A single instruction, multiple thread architecture



Tesla V100 for NVLink

- 15.7 *TFLOPS* for single-precision
- 125 *TFLOPS* for half-precision

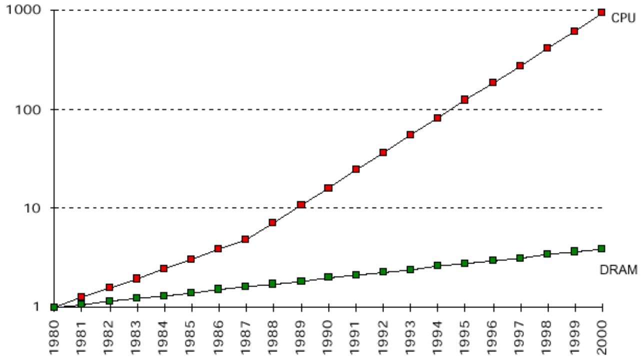
Image from `https://arstechnica.com/gadgets/2017/05/nvidia-tesla-v100-gpu-details/`

The Trend

- On cloud: high density computation
  e.g. Google TPU 3.0 pods are claimed to achieve 100PFLOPS
- On edge: low precision
  e.g. int8 in cDSP supported by Qualcomm's SNPE
- Automatic kernel tuning & generation
  An active research area. Typical projects include Halide[3], TVM[4] and TensorComprehension[5]

[3] `http://halide-lang.org/`
[4] `http://tvmlang.org/`
[5] `https://facebookresearch.github.io/TensorComprehensions/`

Unbalanced Development of Processor and Memory



Graph from[6]

[6] Carlos Carvalho. "The gap between processor and memory speeds". In: *Proc. of IEEE International Conference on Control and Automation*. 2002.

Challenges from NN Architecture

Computation-sparse structure seems to be beneficial.

Architecture	Computation	Memory
Small kernel	$\frac{k2^2}{k1^2}$	Param $\frac{k1^2}{k2^2}$
Large stride	$\frac{1}{s^2}$	Output $\frac{1}{s^2}$
Group/depthwise conv	$\frac{1}{g^2}$	Param $\frac{1}{g}$
Shuffle/concat	0	1

Roofline Model

A visualization method to characterize computation/memory

Performance P (FLOPS) is approximately a function of arithmetic intensity I (FLOP/byte) for a particular architecture[7].

Naïve Roofline

$$P = \min \begin{cases} \pi \\ \beta \times I \end{cases}$$

where π is the peak computing performance and β is the peak bandwidth.

[7] Samuel Webb Williams. *Auto-tuning performance on multicore computers.* University of California, Berkeley, 2008.

Roofline Model

A visualization method to characterize computation/memory

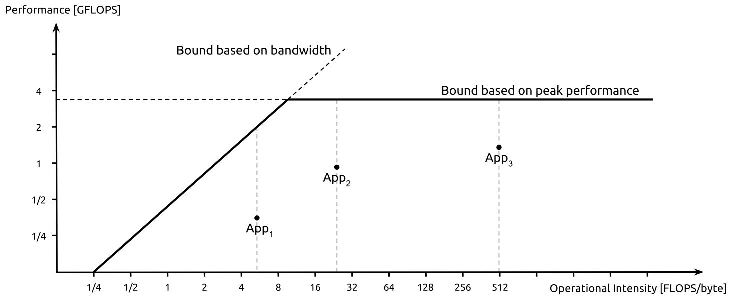


Image from `https://en.wikipedia.org/wiki/Roofline_model`

Thanks!

Questions and feedback are welcome :)