GPU Computing

E. Carlinet, J. Chazalon {`firstname.lastname@lrde.epita.fr`}

Oct 21

EPITA Research & Development Laboratory (LRDE)

Slides generated on October 8, 2021

GPU vs CPU architectures

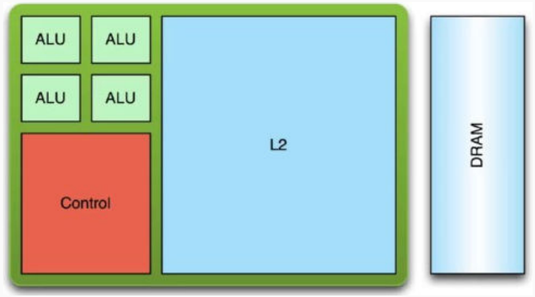
GPU vs CPU architectures

GPU vs CPU architectures

How to explain that:

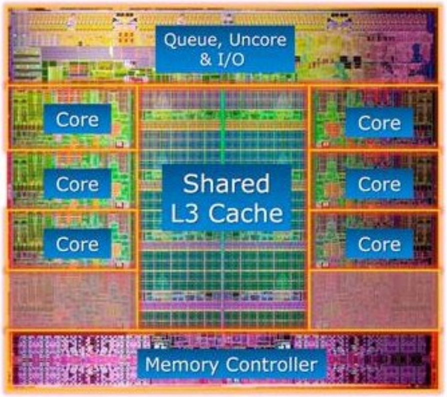
- CPU are low-latency
- GPU are high-throughput

It’s all about data... the CPU:

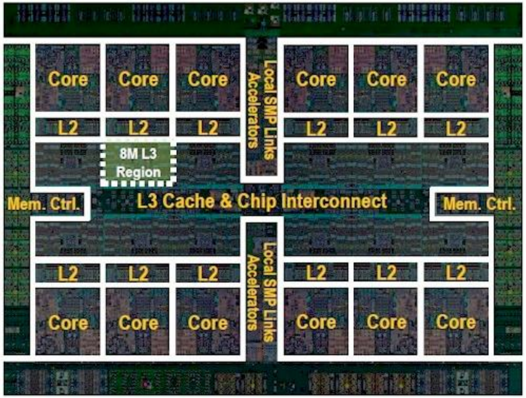


- Optimized for low-latency **access** (many memory caches)
- Control logic for out-of-order and speculative execution

Intel i7

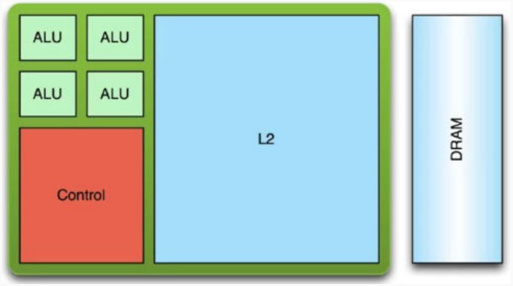


IBM Power 8 (2014)



It’s all about data... the GPU:

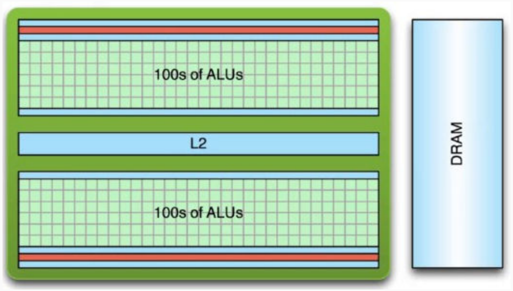
CPU



- Low-latency **access**
- Many control logic

But how... ?

GPU

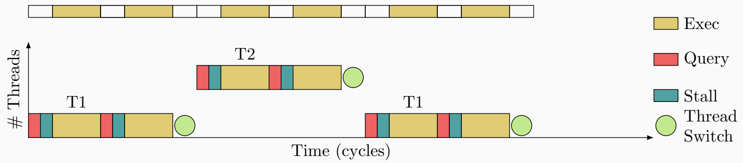


- Throughput computation (ALUs)
- **Tolerant to memory latency**

Hiding latency With thread parallelism & pipelining

So... you want to hide the latency of getting data from global memory... how ?

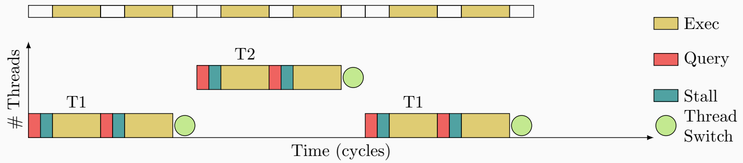
1 CPU Core



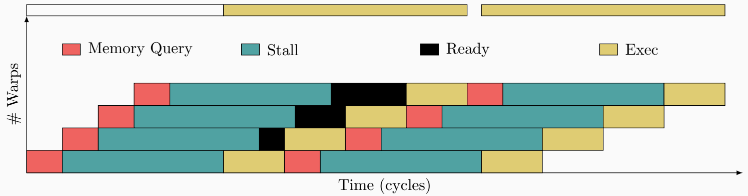
Hiding latency With thread parallelism & pipelining

So… you want to hide the latency of getting data from global memory… how ?

1 CPU Core



1 GPU SMP (Streaming Multiprocessor)



CPU

- low-latency memory to get data ready
- each thread context switch has a cost

GPU

- memory latency hidden by **pipelining**
- context switch is free

Latency hiding

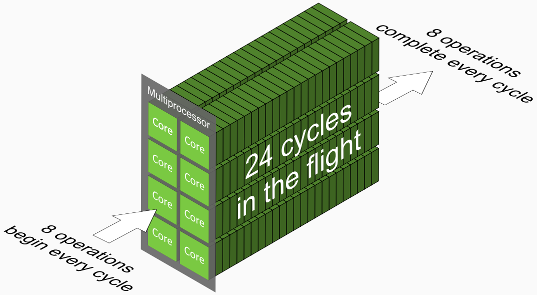
- = do other operations when waiting for data
- = having a lot of parallelism
- = having a lot of data
- will run faster
- but not faster than the peak
- what is the peak by the way ?

It’s all about data... Little’s law

- Customer arrival rate: Throughput
- Customer time spent: Latency
- Avergage customer count: Concurrency (Data in the pipe)” = throughput * Latency

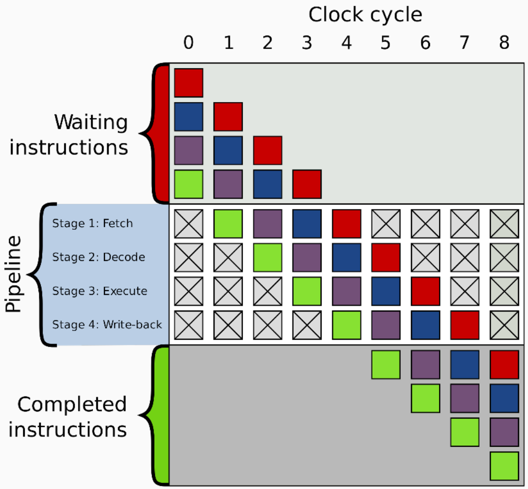
Concurrency is the number of items processed at the same time.

	Latency	Peak Throughput	Needed Concurrency
GPU-arithmetic	24 cycles	8 IPC	192 inst
GPU-memory	350 ns	190 GB/s	65K



Hiding latency

With thread parallelism & pipelining
Note that pipeling exists on CPUs (cycle de Von Neumann):



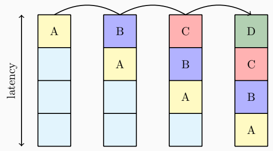
- IF instruction fetch
- ID instruction decode
- OPF Operand fetch
- EX execute
- WB result write back

Pipeline at **Instruction Level** vs pipeline at **Thread (Warp) Level**

More about forms of parallelism (the why!)

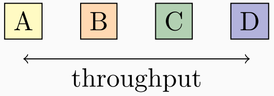
Vertical parallelism for latency hiding

Pipelining keeps units busy when waiting for dependencies, memory



Horizontal parallelism for throughput

More units working in parallel



More about forms of parallelism (the how!)

Instruction-Level Parallelism (ILP)

Between independent instructions.

```
1. add r3 ← r1, r2
2. mul r0 ← r0, r1
3. sub r1 ← r3, r0
```

’1 and ’2 run concurrently

More about forms of parallelism (the how!)

Instruction-Level Parallelism (ILP)

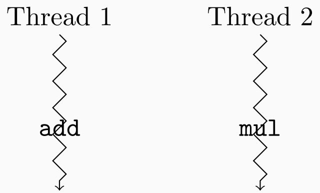
Between independent instructions.

```
1. add r3 ← r1, r2
2. mul r0 ← r0, r1
3. sub r1 ← r3, r0
```

Thread-Level Parallelism (TLP)

Between independent execution contexts: threads

’1 and ’2 run concurrently



More about forms of parallelism (the how!)

Instruction-Level Parallelism (ILP)

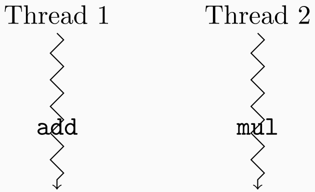
Between independent instructions.

```
1. add r3 ← r1, r2
2. mul r0 ← r0, r1
3. sub r1 ← r3, r0
```

’1 and ’2 run concurrently

Thread-Level Parallelism (TLP)

Between independent execution contexts: threads



Data-Level Parallelism (DLP)

Between elements of a vector: same operation on several elements

```
vadd r ← a, b
```

$$\begin{array}{ccc} a_1 & a_2 & a_3 \\ & + & \\ b_1 & b_2 & b_3 \\ \hline r_1 & r_2 & r_3 \end{array}$$

Extracting parallelism

	Horizontal	Vertical
ILP	Superscalar	Pipeline
TLP	Multi-cores / SMT	Interleaved / Switch-on-event multi-threading
DLP	SIMD / SIMT	

Parallel architectures & parallelism

CPU (Intel Haswell)

	Hor.	Vert.
ILP	8	✓
TLP	4	2
DLP	8	

- 8 ALUs for executing non-dependent instructions
- 4 cores. Physical cores
- 4*2 logical hyper-cores
- Lane of 8x32bits SIMD registers supporting AVX 256

General-purpose multi-cores: balance ILP, TLP and DLP

Parallel architectures & parallelism

CPU (Intel Haswell)

	Hor.	Vert.
ILP	8	✓
TLP	4	2
DLP	8	

- 8 ALUs for executing non-dependent instructions
- 4 cores. Physical cores
- 4*2 logical hyper-cores
- Lane of 8x32bits SIMD registers supporting AVX 256

General-purpose multi-cores: balance ILP, TLP and DLP

GPU (NVidia Kepler)

	Hor.	Vert.
ILP	2	
TLP	16x4	64
DLP	32	

- Dual instruction issue for executing non-dependent instructions
- 16 Multiprocessors (physical cores) Can execute 4 simultaneous warps
- Multithreading (64 warps / SM)
- 128 (4 x 32) CUDA cores. SIMT of width 32

GPU: focus on DLP, TLP horizontal and vertical.

- All processors use hardware to turn parallelism into performance
- GPUs focus on Thread-level and Data-level parallelism