GPGPU: what it is and why you should care

Alexander Titov

About me

Alexander Titov

- Hardware Architect
- 11 years of C++ experience (HW simulation)
- Teaching Computer Architecture and Design

alexander.titov@atitov.com

alexander-titov-cpu

Preface

- **GPGPU** is performing **G**eneral-**P**urpose computation on Graphics Processing Units (**GPU**) instead of CPU
- Goal is to understand the basics of GPGPU that are common for all the HW vendors and for all the SW APIs
- This talk is **not**…
  - a proper intro to CUDA, OpenCL, SYCL or any other framework
  - a discussion on what HW vendor is better, what API is better, etc.
  - able to make you a good GPGPU programmer

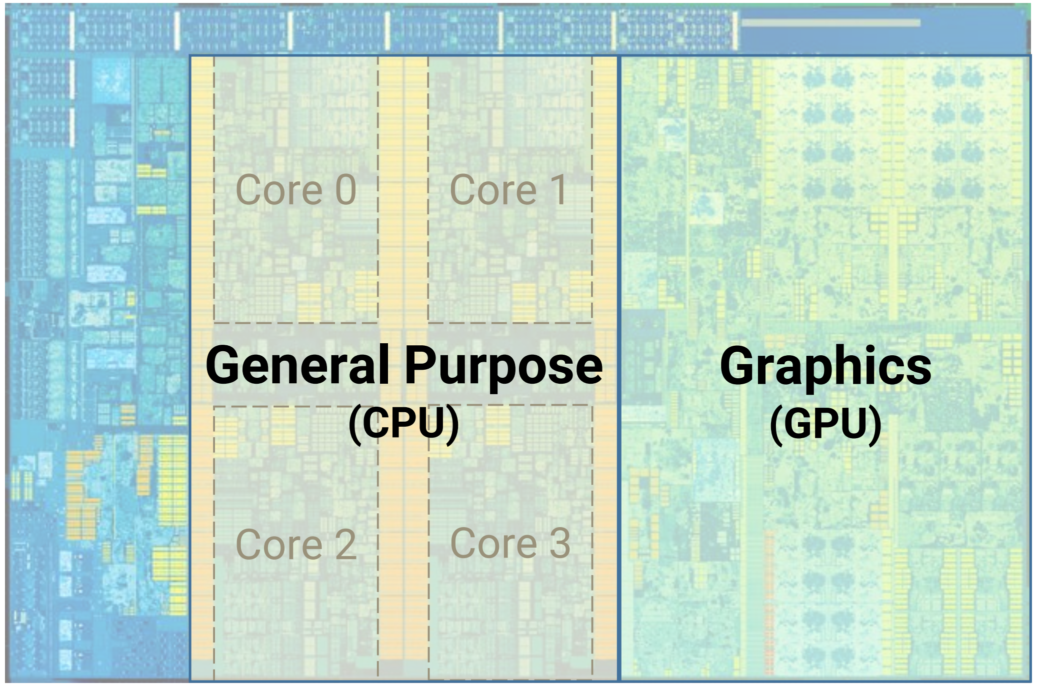
GPGPU Myths (?)

1. GPU is killing CPU
2. GPU will make your applications 1000x faster
3. GPU HW is “magic” and it is very different from CPU HW
4. GPGPU API (e.g., CUDA) is “magic”. It is much easier to write high-performance code for GPU than for CPU

Why GPGPU?

- GPU *dramatically* accelerates *some* general-purpose applications compared to CPU

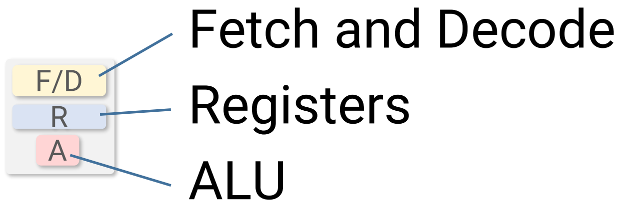
- GPU is already everywhere
  - Desktops, laptops, mobile… not servers (yet)
- GPU is not a second-class citizen
  - GPU area ≥ CPU area



Core i5-7400T, Kaby Lake, Jan 2017
Gen9 GT2 <- the smallest GPU configuration

CPU Evolution

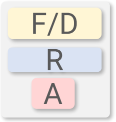
- Extremely simple: one instruction at a cycle, everything is in order
- No parallelism



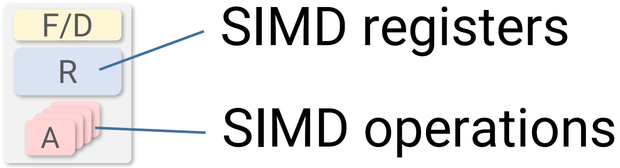
In Order

CPU Evolution

- Single-Instruction-Multiple-Data (SIMD) operations to leverage Data Level Parallelism (DLP)



In Order



In Order

SIMD (DLP)

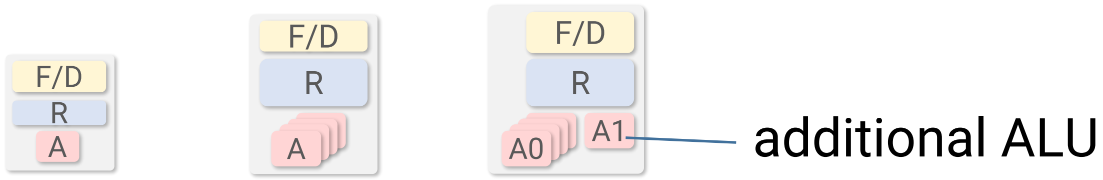
Example of a SIMD operation:

```
B[0] = add A[0], 1
B[1] = add A[1], 1
B[2] = add A[2], 1
B[3] = add A[3], 1
```

```
B[0:3] = vadd A[0:3], 1
```

CPU Evolution

- SuperScalar approach: if two (or more) consecutive instructions are independent, execute them in parallel
- A very limited solution to exploit Instruction Level Parallelism (ILP)



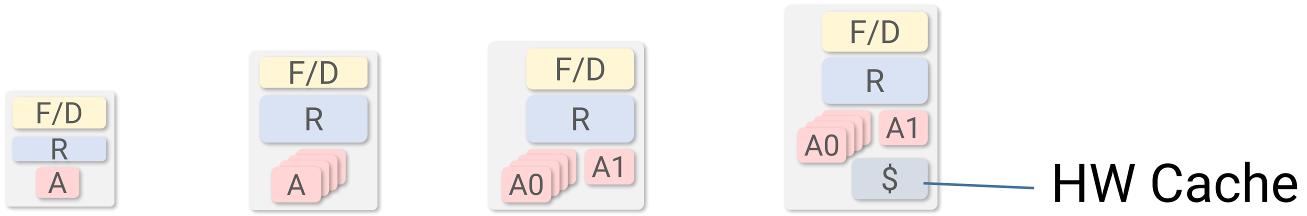
In Order

In Order
SIMD (DLP)

In Order
SIMD (DLP)
SuperScalar (ILP)

CPU Evolution

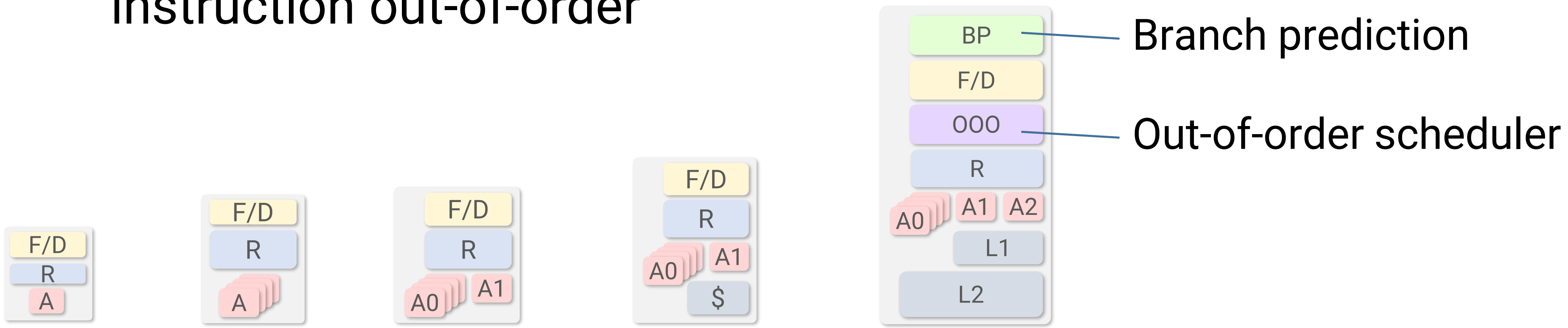
- Memory access latency is becoming a problem
- Adding HW Caches and prefetch to mitigate it



In Order	In Order	In Order	In Order
	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)
		SuperScalar (ILP)	SuperScalar (ILP)
			Caches

CPU Evolution

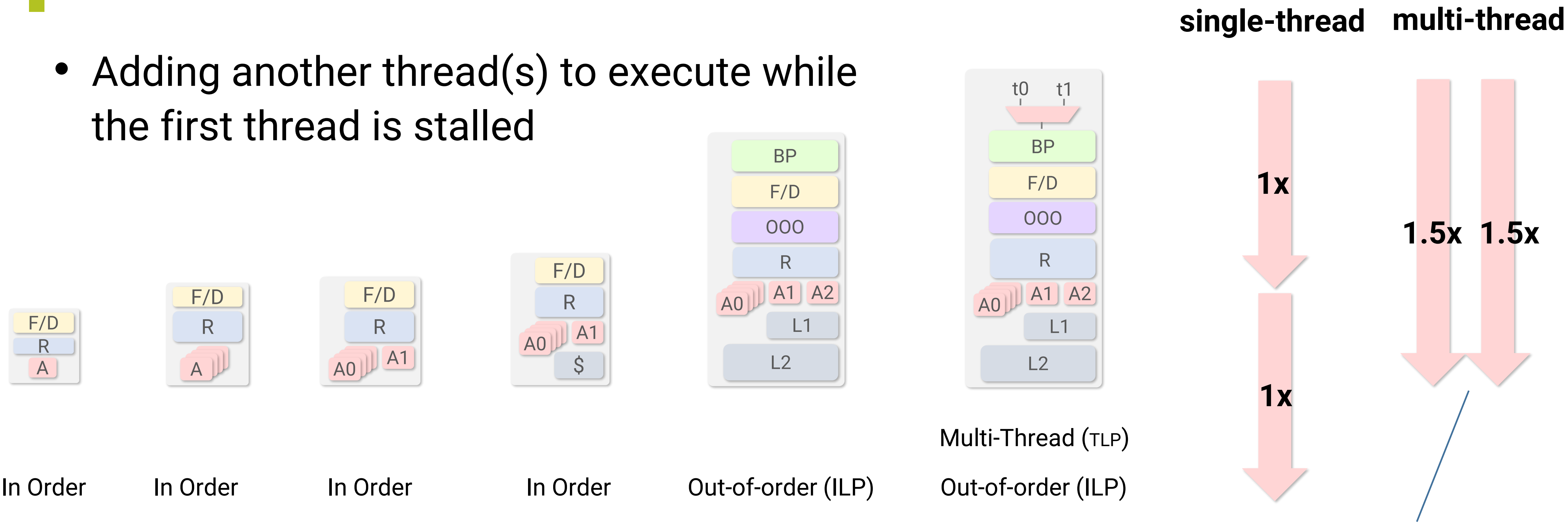
- Use sophisticated HW to extract more ILP by executing independent instruction out-of-order



In Order	In Order	In Order	In Order	Out-of-order (ILP)
	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)
		SuperScalar (ILP)	SuperScalar (ILP)	SuperScalar (ILP)
			Caches	Caches

CPU Evolution

- Adding another thread(s) to execute while the first thread is stalled

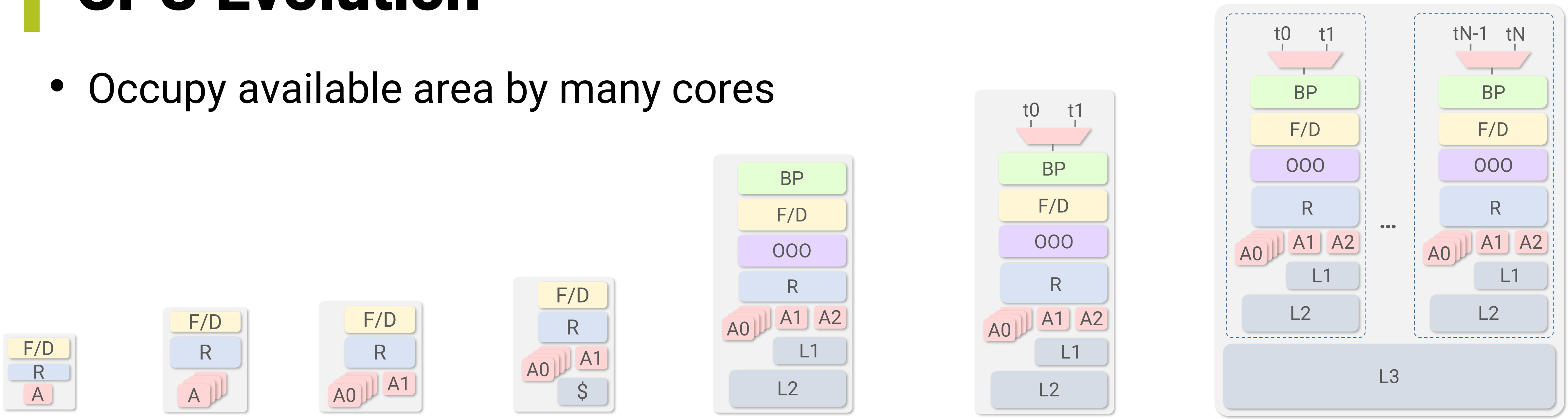


In Order	In Order	In Order	In Order	Out-of-order (ILP)	Multi-Thread (TLP)
	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	Out-of-order (ILP)
		SuperScalar (ILP)	SuperScalar (ILP)	SuperScalar (ILP)	SIMD (DLP)
			Caches	Caches	SuperScalar (ILP)
					Caches

throughput is higher,
but each thread is slower

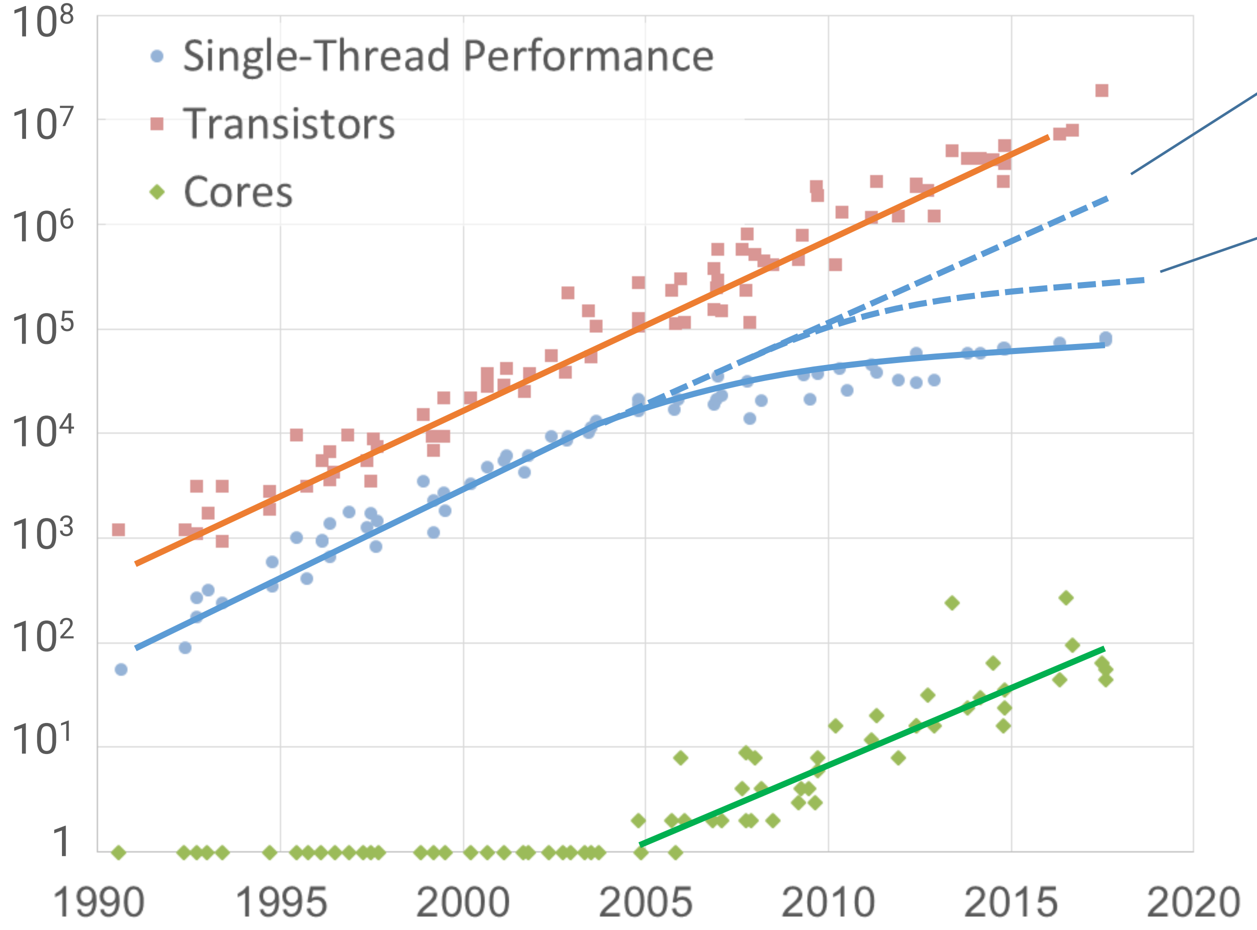
CPU Evolution

- Occupy available area by many cores



					Multi-Thread (TLP)	Multi-Thread-Core (TLP)
In Order	In Order	In Order	In Order	Out-of-order (ILP)	Out-of-order (ILP)	Out-of-order (ILP)
	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)	SIMD (DLP)
		SuperScalar (ILP)	SuperScalar (ILP)	SuperScalar (ILP)	SuperScalar (ILP)	SuperScalar (ILP)
			Caches	Caches	Caches	Caches

CPU Evolution: Performance Trends



Data up to 2010 collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. Data for 2010-2017 by K. Rupp

About 5-10% of applications are well scaled → **need many simple cores**

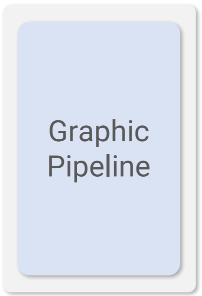
Majority of applications cannot utilize more than 4-8 cores
→ **need a few large cores**

- Cannot optimize CPUs for 5-10% applications
- Need to develop another device?
  → **No, it is already available**

Myth: GPU is killing CPU BUSTED!

GPU Evolution (toward GPGPU)

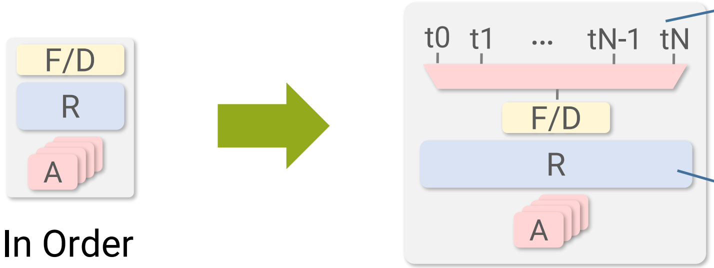
- Initially, GPU was designed strictly for rendering 2D and 3D
- Graphic pipeline was fixed and could not be programmed



Non-Programmable

GPU Evolution (toward GPGPU)

- Impossible to create special HW for all the demanded rendering functionality → programmability is needed
- What is a typical rendering task?
  - the same actions on multiple independent elements (e.g., turn pixels color to gray) → it is SIMD!
  - processing time of a single element is not important, only total time matter → it is throughput!



Many threads to hide latencies (8-16 in practice)

Large register file to keep registers of all the thread

In Order

SIMD (DLP)

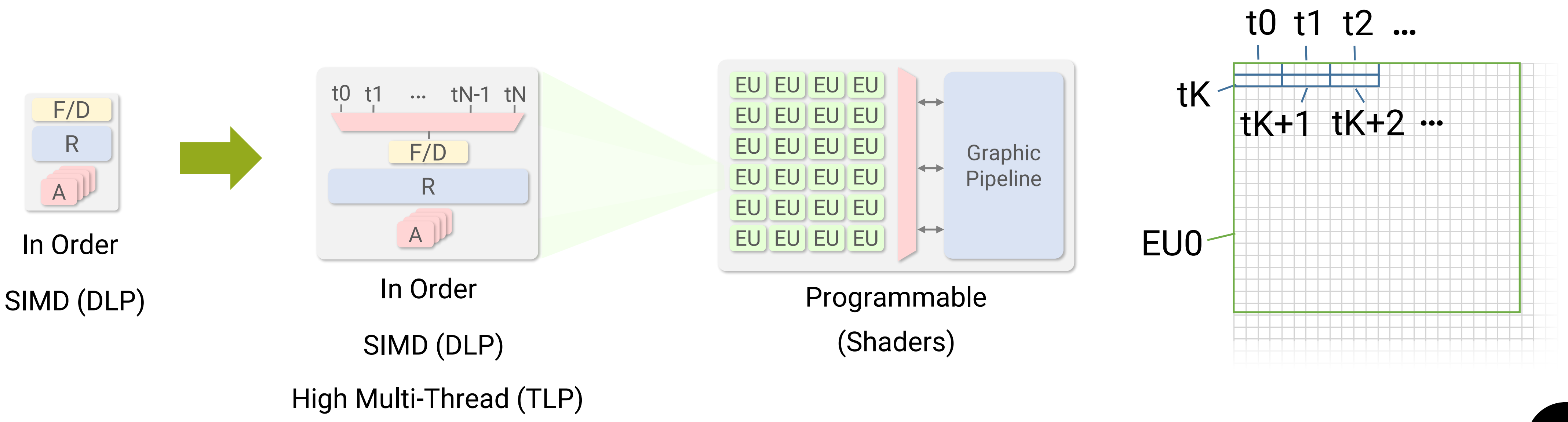
In Order

SIMD (DLP)

High Multi-Thread (TLP)

GPU Evolution (toward GPGPU)

- Impossible to create special HW for all the demanded rendering functionality → programmability is needed
- What is a typical rendering task?
  - the same actions on multiple independent elements (e.g., turn pixels color to gray) → it is SIMD!
  - processing time of a single element is not important, only total time matter → it is throughput!



CPU HW vs GPU HW

- Similar HW techniques, just optimized for different purposes

CPU	GPU
good for everything the best for single-thread	good for throughput only
large ILP	no ILP
large DLP (SIMD)	large DLP (SIMD)
medium TLP	extreme TLP

Myth: GPU HW is “magic” and it is very different from CPU HW

How Fast GPU vs. CPU?

- There are a lot of misunderstanding and black marketing:

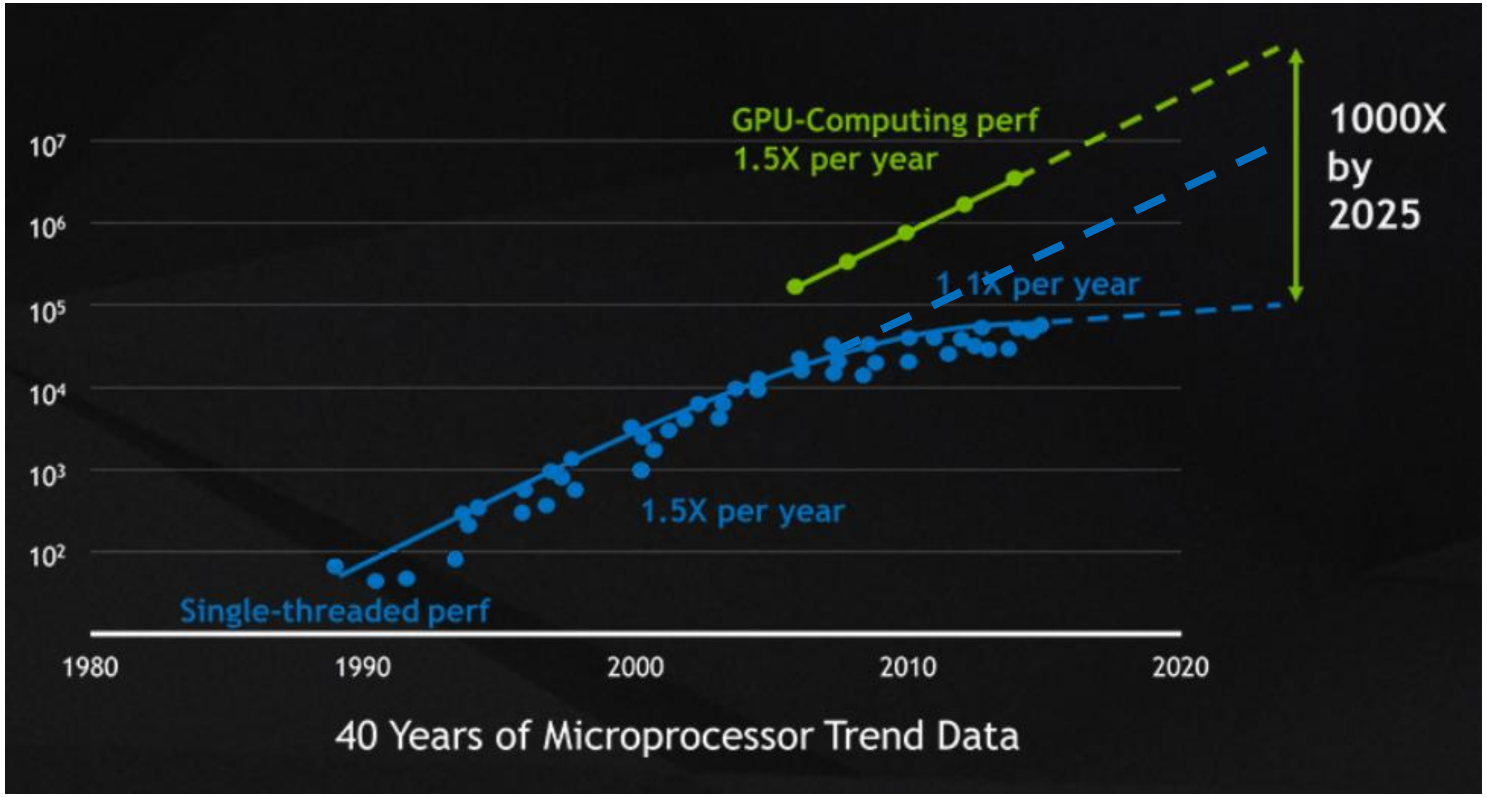
	Google	Stanford
Number of cores	1K CPUs = 16K cores	3GPUs = 18K cores
Cost	$5B	$33K
Training time	week	week

What is a core on GPU?

Are CPU and GPU cores are comparable?

How Fast GPU vs. CPU?

- There are a lot of misunderstanding and black marketing:



Need to compare with throughput applications on many cores

How Fast GPU vs. CPU?

	2080 Ti (Turing), 2017Y	Xeon 8168 (Skylake), 2017Y	GPU / CPU
Cores	~~4352~~	24	~~181x~~
Compute capacity	14 TFLOPS (FP32) 110 TFLOPS (FP16 tensor cores)	~3 TFLOPS (FP32)	5x 36x
Memory BW	616 GB/s	120 GB/s	5x
Area	754 mm2	694 mm2	~1x
Price	1000$	6000x	6x

- It is reasonable to say that GPU is up to ~10x faster than CPU (36x for ML)

Myth: GPU will make your applications 1000x faster

GPGPU Programming

- There are many GPGPU APIs for C/C++:
  - **CUDA**: many features, easy-to-use, but not open and not portable (only NVIDIA)
  - **OpenCL**: less features, verbose, but open and portable (Intel, AMD, NVIDIA, etc.)
  - **SYCL**: based on OpenCL, easy-to-use, more C++ oriented, but still in development
- All APIs are based on the offload model: host (CPU) prepares and controls everything, device (GPU) only execute
- Code is divided in the two parts:
  - the device code (kernel) written in a subset of C/C++
  - the host code in your C/C++ program

Kernel Code (Open CL)

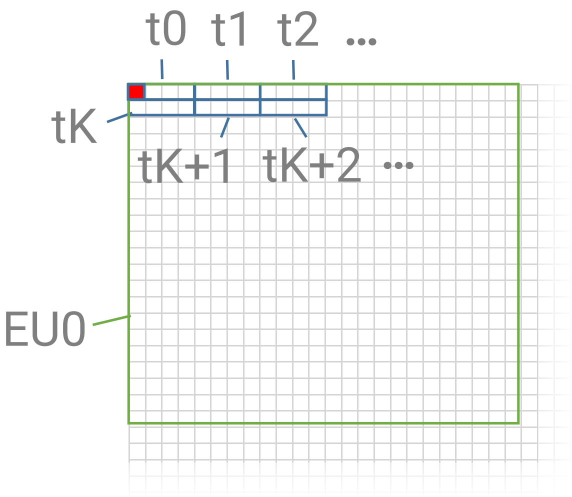
- How to program this crazy mix of vectors and threads?
- Big Idea: as it is SIMD, write code for a single data element only
  - The kernel compiler forms vectors and create threads on its own

Traditional Loop

```
void
mul(const float *a,
    const float *b,
          float *c,
    const int n)
{
  for (int i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

Data parallel OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)

{
  int i = get_global_id(0);
  c[i] = a[i] * b[i];
}
```



Kernel Code (Open CL)

- Is everything so simple? → in toy examples – yes, but not in the real world
- Problem: Divergent control flow decreases utilization

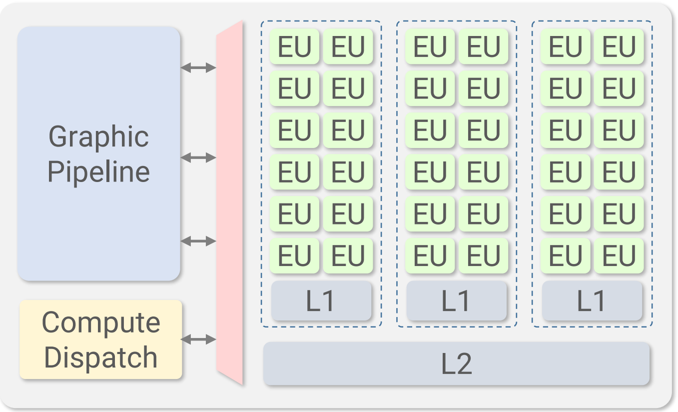
```
__kernel void
foo(__global const float *a,
    __global const float *b,
    __global       float *c)

{
  int i = get_global_id(0);
  if (i % 2)
     c[i] = a[i] * a[i];
  else
     c[i] = b[i] * b[i];
}
```

```
mask = {0, 1, 0, 1, 0, 1, 0, 1}
c[0:7] = vmul a[0:7], a[0:7], mask
c[0:7] = vmul b[0:7], b[0:7], !mask
```

half of elements is calculated, but dropped

Kernel Code (Open CL)

- Is everything so simple? → in toy examples – yes, but not in the real world
- Problem: Divergent control flow decreases utilization
- Problem: Tuning for GPU layout is required



Hand on OpenCL Workshop, UoB-HPC, 2018

Matrix Multiplication Approaches	GFLOP/s	
	CPU	GPU
Sequential C (not OpenCL)	0.85	N/A
C(i,j) per work-item, all global	111.8	70.3
C row per work-item, all global	61.8	9.1
C row per work-item, A row private	9.6	24.9
C row per work-item, A private, B local	12.3	55.4
Block oriented approach using local	138.0	1,801.8
	11.5% of peak	21.2% of peak

Myth: It is much easier to write high-performance code for GPU than for CPU

Conclusions

- 5-10x speedup is good to invest
- APIs are mature for production, but still evolve
- Entry threshold is medium
- Writing high-performance code is as hard as on CPU

Many thanks!

Questions welcome :)

Alexander Titov

alexander.titov@atitov.com

alexander-titov-cpu