Around GPGPU: architecture, programming, and arithmetic

Sylvain Collange, Arénaire, LIP, ENS Lyon
David Defour, DALI, ELIAUS, Université de Perpignan

November 10, 2010

Key challenges for parallel architectures

- Scalability
  - Moving data is more expensive than computing
  - How to minimize data movement in a many-core architecture?
- Power efficiency
  - Power draw/dissipation is the current bottleneck
  - Power-directed design
- Programming model
  - How to write portable, reusable parallel software with minimal effort?
- Numerical accuracy
  - Confidence in a result produced after billions of operations?

Outline

- How a GPU works
- GPU programming guidelines
- Arithmetic features

GPU: a new architecture?

The Tesla SM uses ***a new processor architecture*** *we call single-instruction, multiple-thread (****SIMT****).*

Erik Lindholm et al., NVIDIA Tesla: a unified graphics and computing architecture, IEEE Micro, 2008

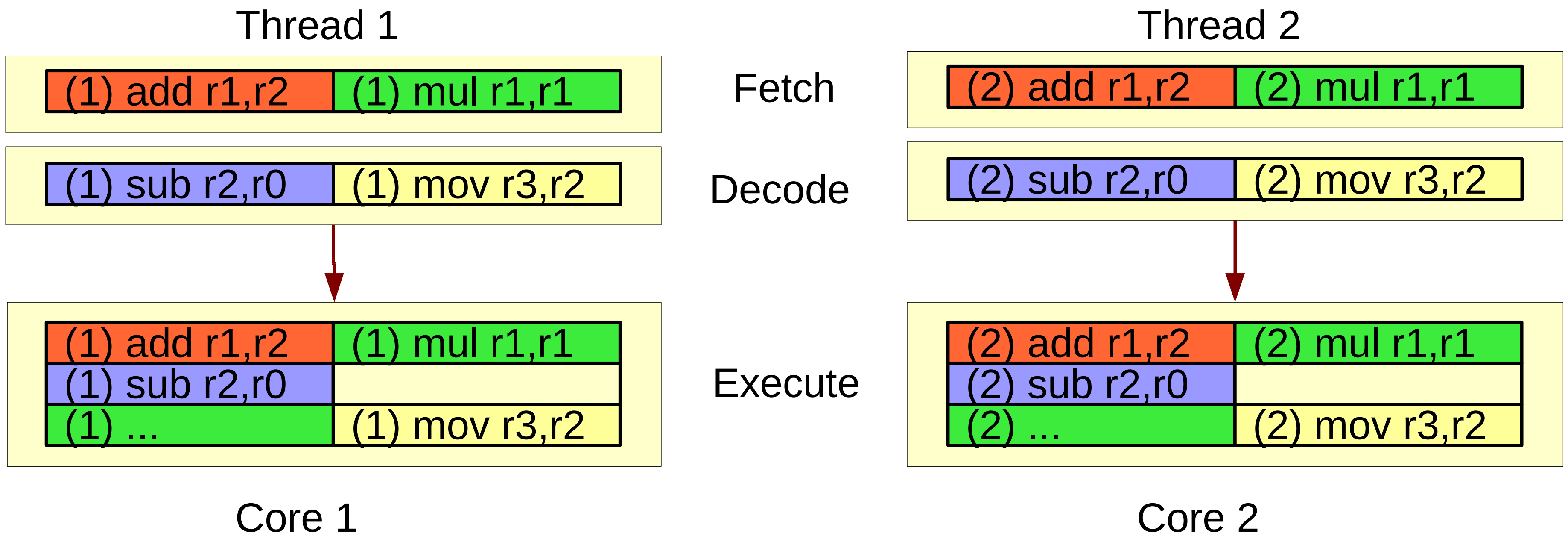
The Streaming Multiprocessor in reality is a ***highly threaded single-issue processor with SIMD****, although this is obscured by the overall complexity and marketing of the whole architecture.*

David Kanter, NVIDIA's GT200: Inside a Parallel Processor, Real Wold Tech, 2008

- Who is right?
- Difference with parallel processors from the 80's?

First step: MIMD

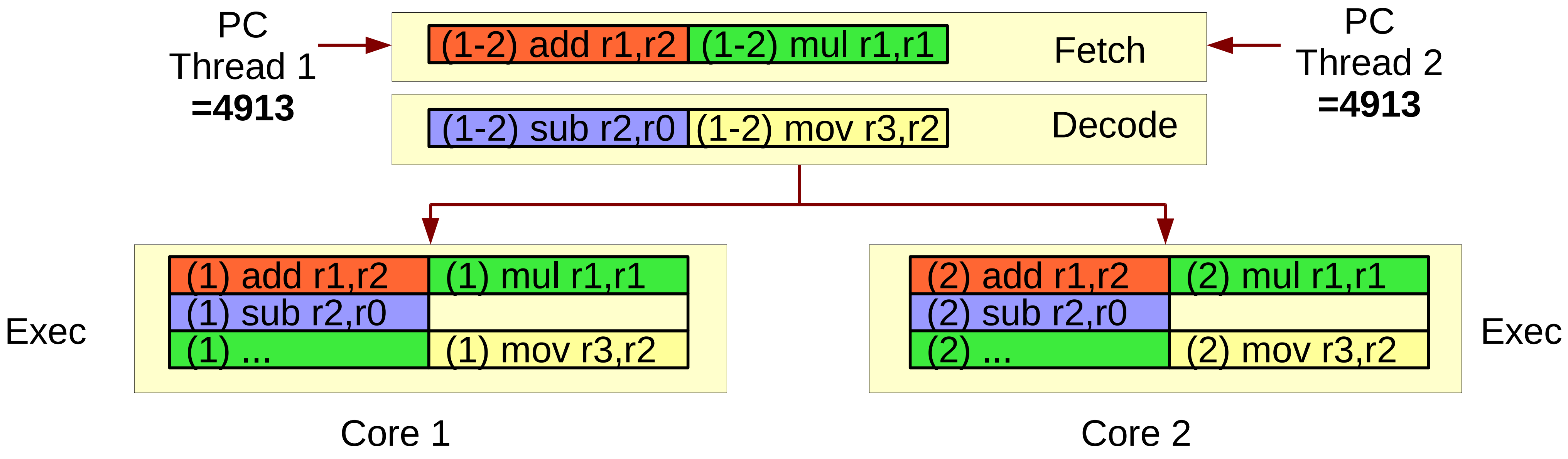
- Multiple small, independent cores



- Benefit from task / data parallelism

Second step: SIMT

- Share front-end (I$, F, D) between cores
- When both threads execute the same instruction
  - Fetch and decode it once, then broadcast it

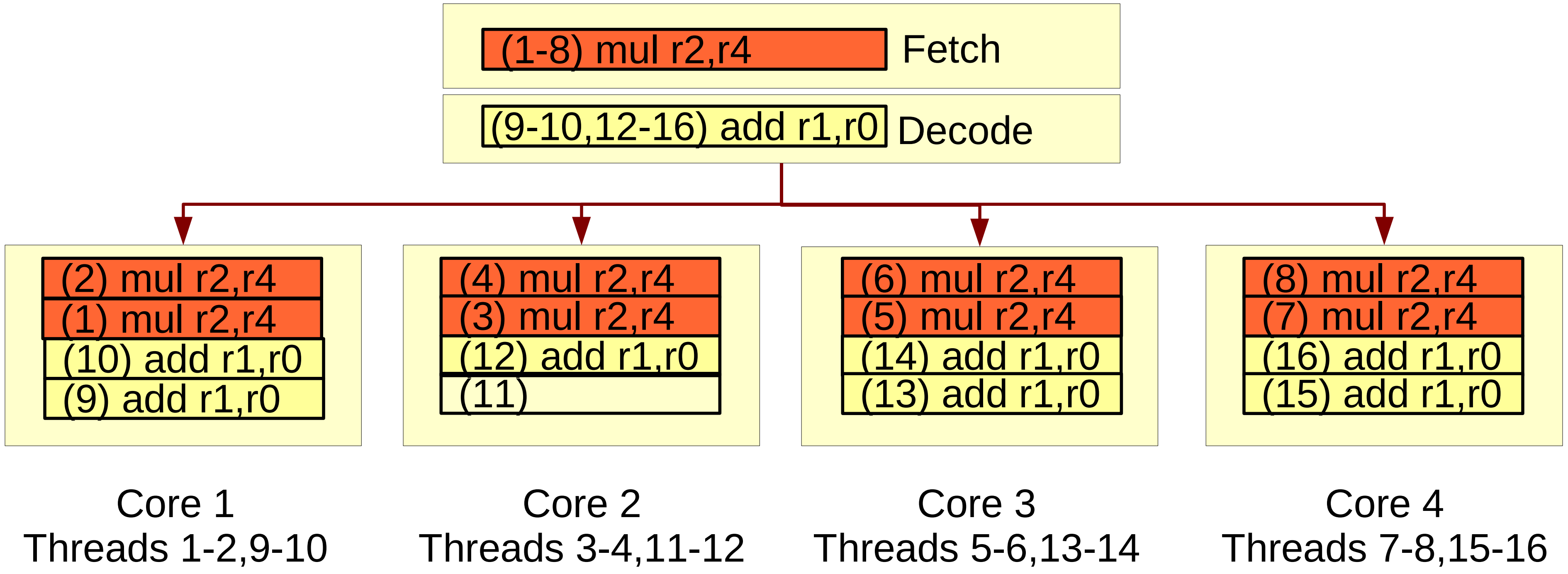


- Benefit from instruction regularity

A GPU

- More threads / core
- More cores / shared front-end
- Replicate instructions in time
- Share load-store unit, caches

- → Data parallelism
- → Instruction regularity
- → Data locality



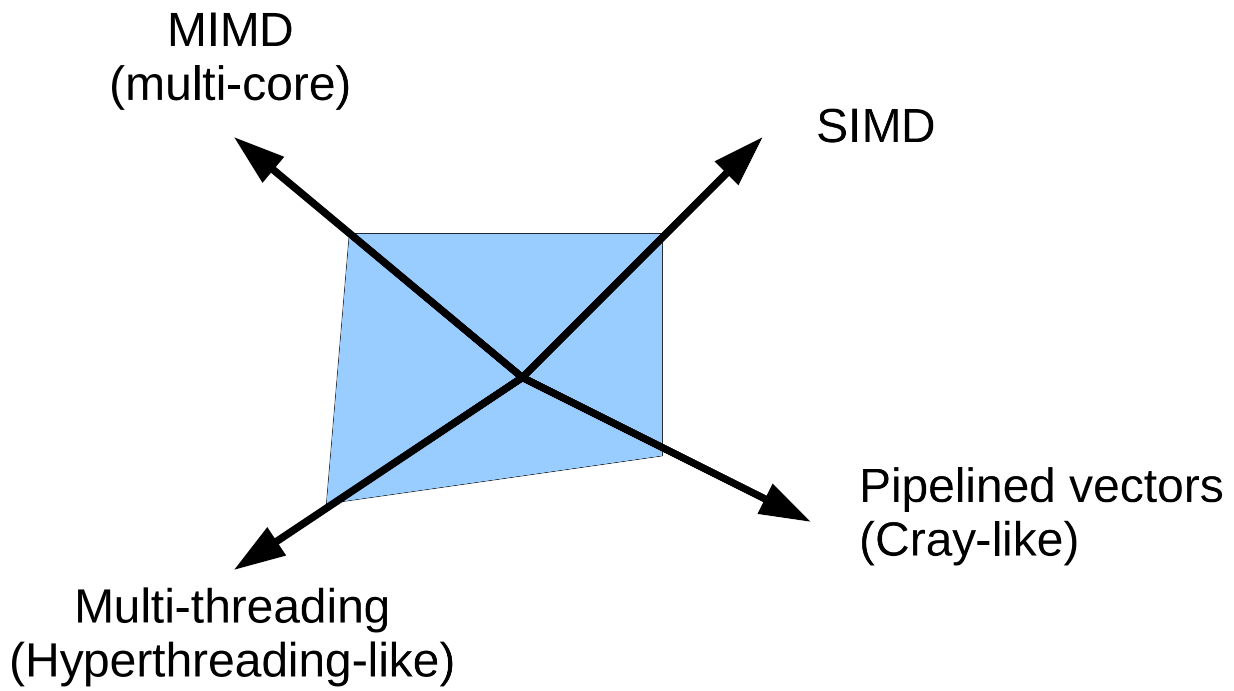
SIMD vs. SIMT

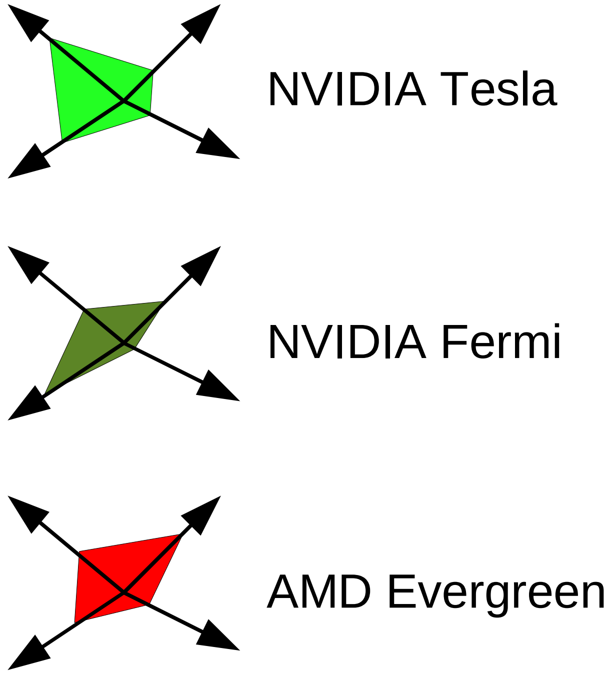
	SIMD or vector	SIMT
Vectorization	At compile-time	At runtime
Thread divergence	Software-managed Bit-masking, predication	Hardware-managed Stack, counters, multiple PCs…
Memory access	Vector load-store Gather-scatter	Gather-scatter with coalescing
And much more... [Glew09]		

- SIMT architecture : run SPMD code on SIMD units
  - Both authors are right...
  - SIMD units = only one possible implementation of SIMT

GPU design space

- What can we do with SPMD threads?





- This is the GPU architect's problem
- Programmer's point of view: just a bunch of threads
  - Microarchitecture-specific optimizations
  - Or just focus on locality and regularity

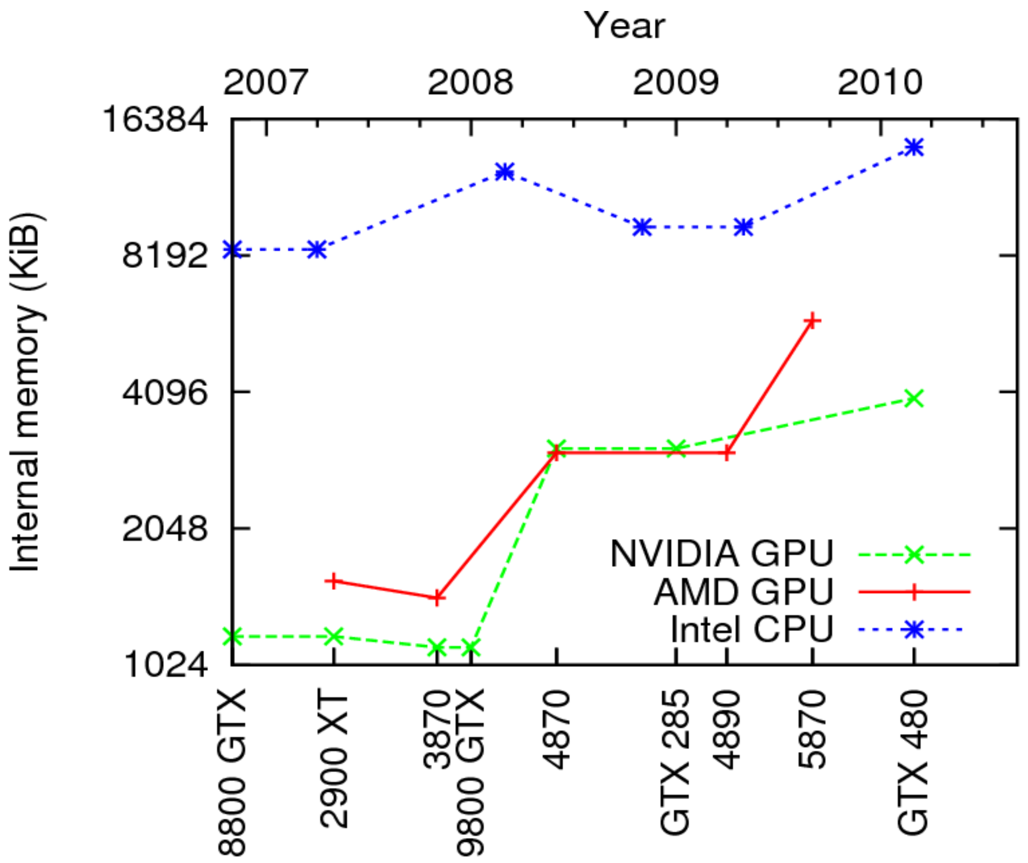
Outline

- How a GPU works
- GPU programming guidelines
  - Bottlenecks and limitations
  - Some recipes
- Arithmetic features

Where are my transistors gone?

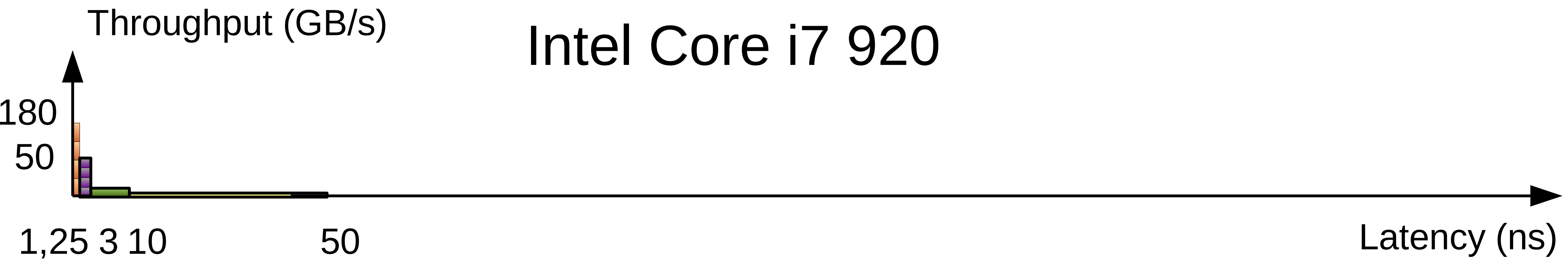
- Conventional wisdom
  - CPUs have huge amounts of cache
  - GPUs have almost none
- Reality check

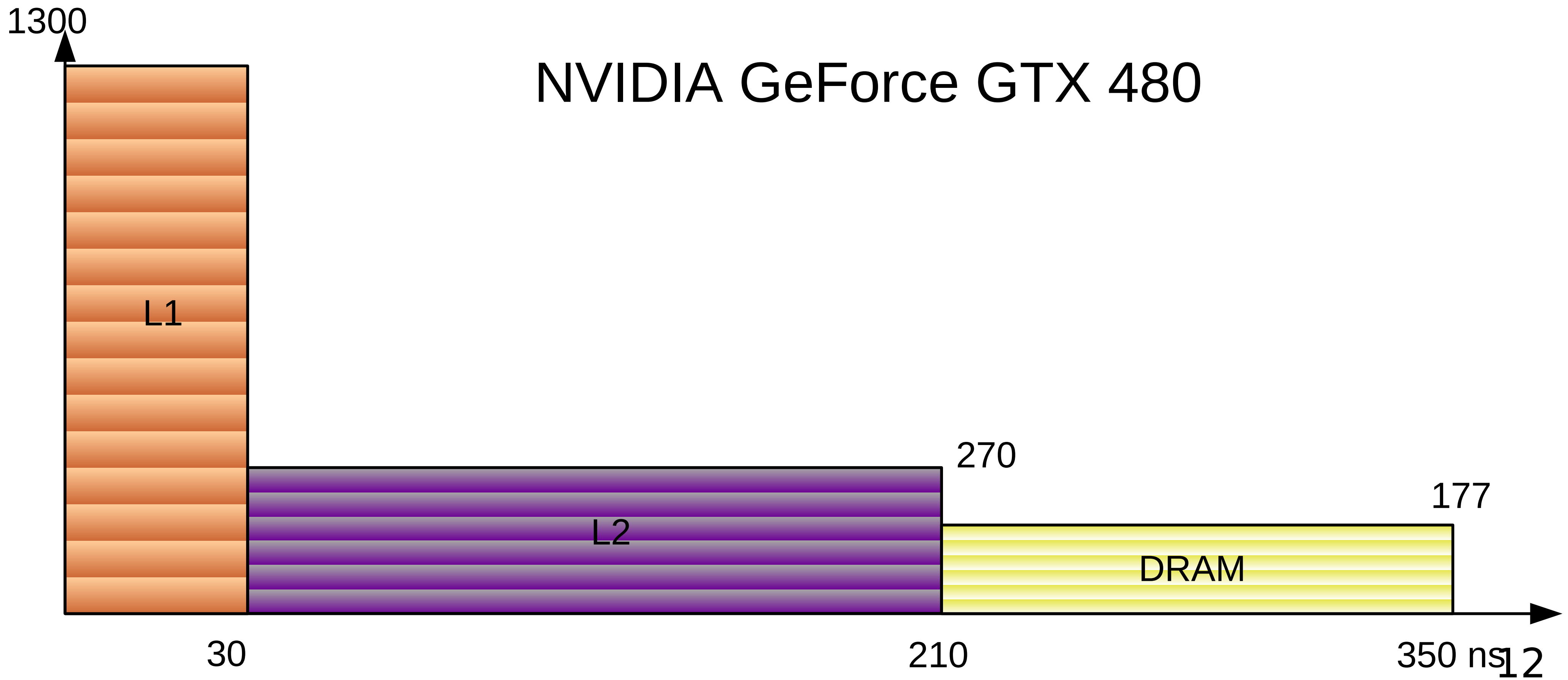
GPU	Register files + caches
NVIDIA GF100	3.9 MB
AMD Cypress	5.8 MB



- At this rate, will catch up with CPUs by 2012…

Little's law: data=throughput×latency





What about power?

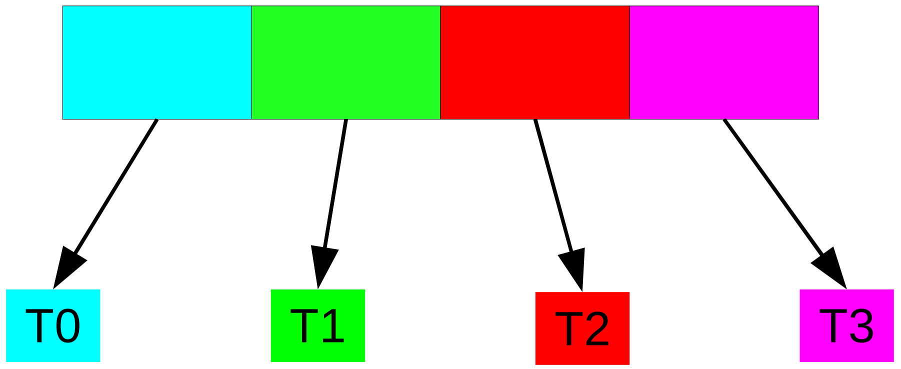
- Power measurements on NVIDIA GT200 [CDT09]

	Energy/op (nJ)	Total power (W)
Instruction control	1.8	18
32-way vector MAD	3.6	36
128-byte vector load	80	90

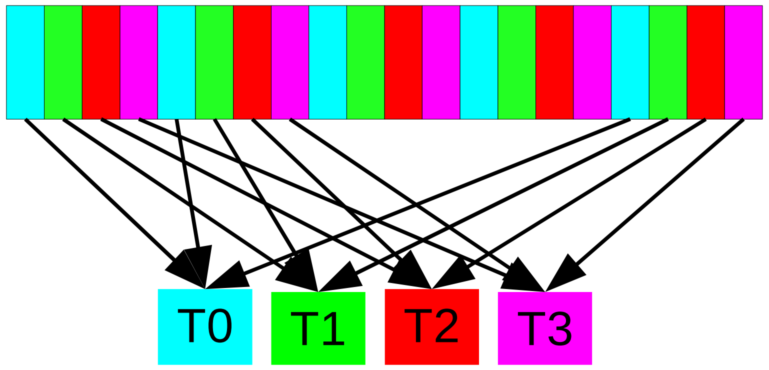
- Instruction overhead is under control
  - Thanks to SIMT
- FPUs are not so cheap
  - Once we put hundreds of them on a chip
- Memory is the killer

Guidelines: scheduling work

- On multicore / multiprocessor
  - Coarse-grained parallelism
  - **Decouple** tasks to reduce **conflicts** and inter-thread communication



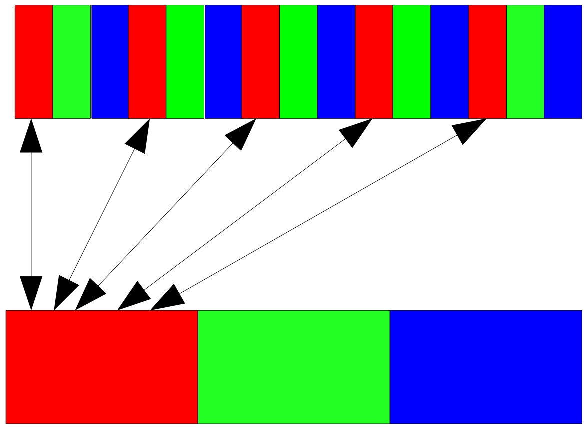
- On GPUs
  - Fine-grained parallelism
  - **Interleave** tasks
  - Exhibit **locality**: take advantage of local memory
  - Exhibit **regularity**: take advantage of SIMT units



Packing data

- Array of Structures (AoS)
  - Alignment?
  - Partial access (only blue)?
  - Access pattern on GPU?

```
struct Pixel {
    float r, g, b;
};
Pixel image_AoS[480][640];
```



- Structure of Arrays (SoA)
  - More GPU-friendly

```
struct Image {
    float R[480][640];
    float G[480][640];
    float B[480][640];
};
Image image_SoA;
```

- Prefer SoA in memory [Mici10]
  - Library to hide layout issues: [Strz10]

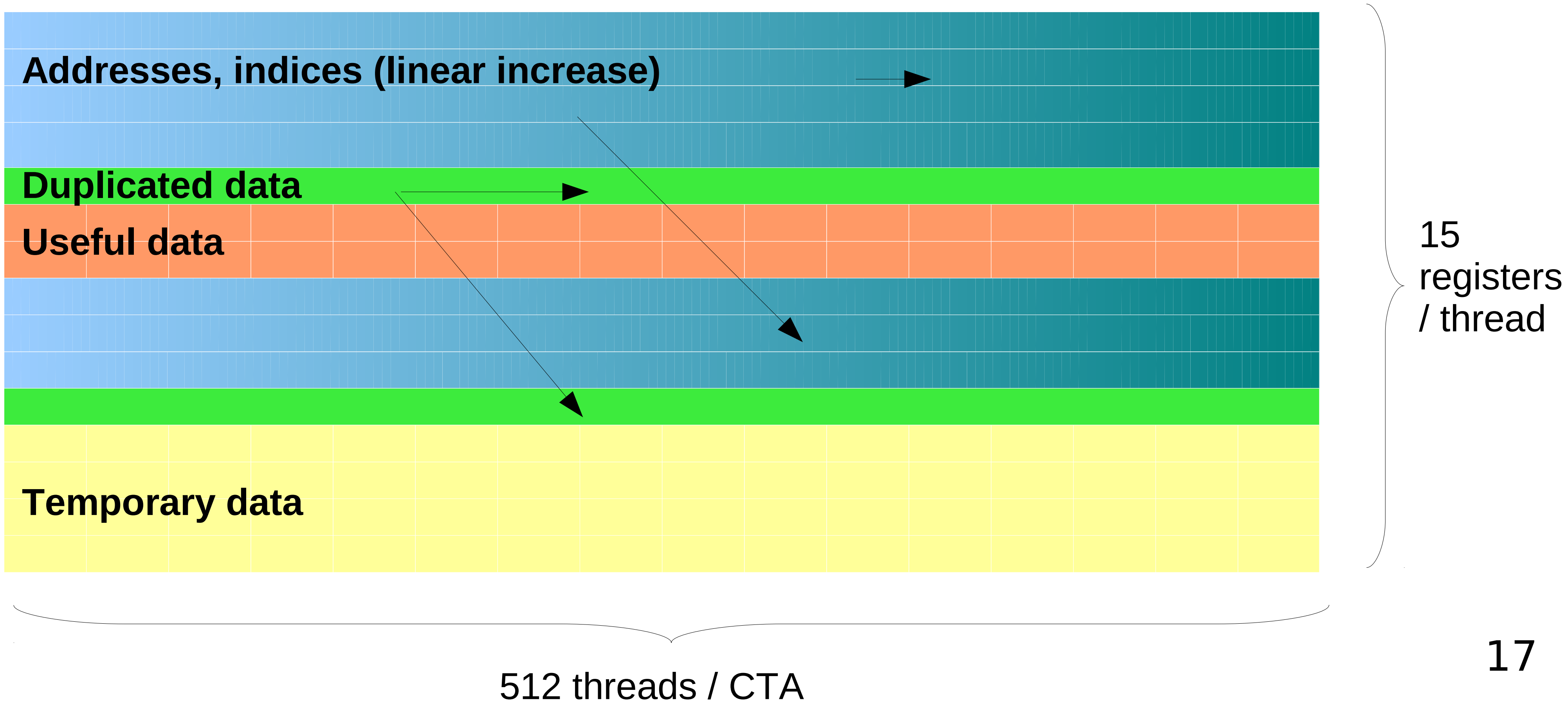
How many threads?

- As many as possible?
  - + Maximal data-parallelism
    - Latency hiding
  - − Locality
    - Store private data of each thread
  - − Thread management overhead
    - Initialization, redundant operations
- Instruction-Level Parallelism is not dead
  - Up to 5 pending loads/thread on Tesla, more on Fermi
  - Superscalar (supervector?) execution on GF104
  - VLIW on AMD architectures

Example : SGEMM from CUBLAS 1.1

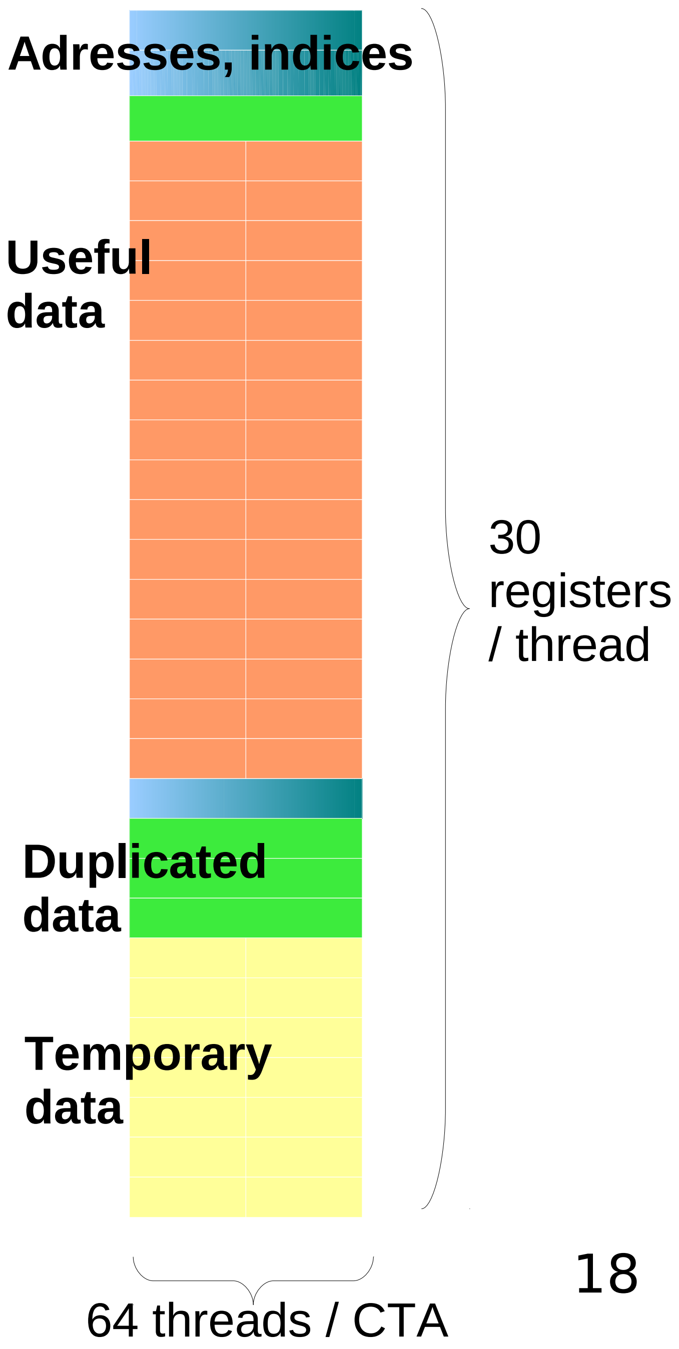
From: **Vasily Volkov**. Programming inverse memory hierarchy : case of stencils on GPUs. *ParCFD*, 2010.

- 512 threads / CTA, 15 registers / thread
- 9 registers / 15 contain redundant data
- Only 2 registers really needed



Fewer threads, more computations

- Volkov SGEMM
  - 8 elements computed / thread
  - Unrolled loops
  - Less traffic through shared memory, more through registers
- Overhead amortized
  - 1920 registers vs. 7680 for the same amount of work
  - Works for redundant computations too
- Success story
  - +60% compared to CUBLAS 1.1
  - Adopted in CUBLAS 2.0
- More in [Volk10]



Takeaway

- Distribute work and data
  - Favor SoA
  - Favor locality and regularity
  - Use common sense (avoid extraneous copies or indirections)
- More threads ≠ higher performance
  - Saturate instruction-level parallelism first (almost free)
  - Complete with data parallelism (expensive in terms of locality)
  - Compiler optimization: thread fusion?

Outline

- How a GPU works
- GPU programming guidelines
- Arithmetic features
  - IEEE-754?
  - A bit of history
  - FP capabilities

Every new generation is “now IEEE-754”

*The vector unit can perform four **IEEE single-precision** multiply, add, or multiply-add operations, as well as inner products, max, min, and so on.*

J. Montrym, H. Moreton, The **GeForce 6800**, IEEE Micro, 2005

*The floating-point add and multiply operations are **compatible with the IEEE 754 standard** for single-precision FP numbers, including not-a-number (NaN) and infinity values.*

Erik Lindholm et al., NVIDIA **Tesla**: a unified graphics and computing architecture, IEEE Micro, 2008

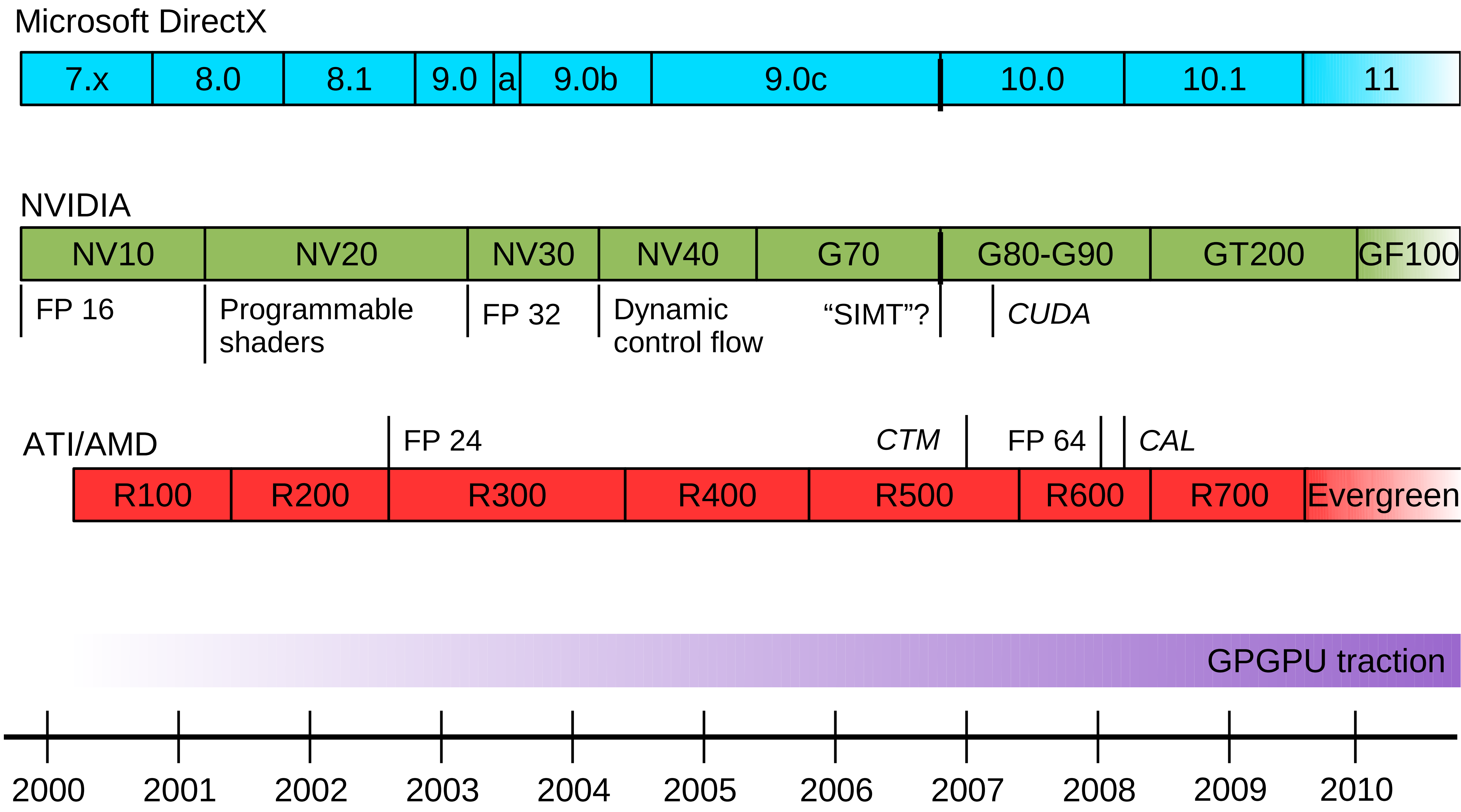
*Single precision floating point instructions now support subnormal numbers by default in hardware, as well as all **four IEEE 754-2008 rounding modes** (nearest, zero, positive infinity, and negative infinity).*

NVIDIA's next generation CUDA compute architecture: **Fermi** Whitepaper, 2009

*All compute devices **follow the IEEE 754-2008 standard** for binary floating-point arithmetic with the following **deviations**:*
[…2-page long bullet list…]

NVIDIA CUDA C Programming Guide, 2010

A short glimpse at recent GPU history



Arithmetic features

- 2006 (ATI R500, NVIDIA G70): “Cray-1-like” FP
  - Truncated multipliers, adders with 2 guard bits and no sticky
  - 41 / 41 ≠ 1
  - Same GPU, different units: different behavior
- 2007 (ATI R600, NVIDIA G80)
  - Correct IEEE-754 rounding to the nearest for +, ×
  - Integer arithmetic and logical ops
- 2008 (AMD R670, NVIDIA GT200)
  - Binary64
- 2010 (AMD Evergreen, NVIDIA GF100)
  - 4 mandatory IEEE rounding modes
  - FMA for both Binary32 and Binary64
  - Subnormals at full-speed

Hardware elementary functions

We therefore conclude that

(1) the entire function library should be included in the hardware if and only if COMPLEX data types and their corresponding arithmetic are formally introduced;

(2) the following error/accuracy criterion should be adopted and met by the implementation: [Correct rounding].

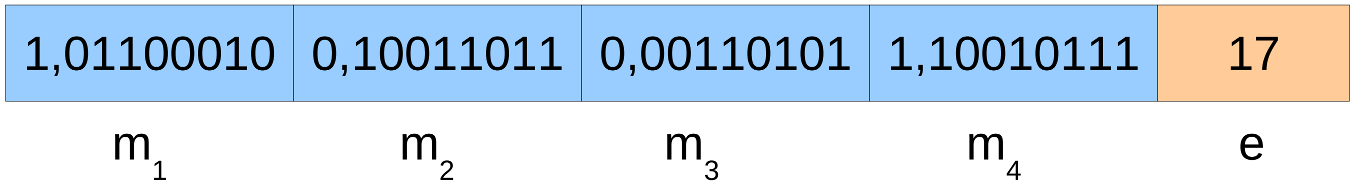
If either of these conditions is not met, then none of the elementary functions should be included in the hardware.

G. Paul, M.W. Wilson, Should the elementary function library be incorporated into computer instruction sets?, TOMS, 1976.

- 34 years later: still no complex datatypes nor correct rounding of elementary functions
- But we have hardware elementary functions on GPUs
  - $1/x$, $1/\sqrt{x}$, $\log_2$, 2^x, sin, cos
  - Accuracy: 22 to 23 bits
- Applications: graphics, physics, finance…

Graphics is bandwidth-starved too

- Lower-precision format: Binary16
  - 11-bit significand, 6-bit exponent
  - In IEEE-754:2008
- Block Floating-Point formats
  - One shared exponent, multiple significands
  - More compact storage for correlated FP data



$f_1=m_1 x 2^e$
$f_2=m_2 x 2^e$
$f_3=m_3 x 2^e$
$f_4=m_4 x 2^e$

- Lossy compression of textures in memory
  - Hardware-based on-the-fly decompression
- Lossless compression of frame buffer, depth buffer…

FMA

- Higher accuracy
  - One less rounding error
- Error-free transformations
  - FMA(a, b, -a×b)
- Different behavior than a×b+c
- Loss of symmetry (dot product…)
  - $a \times b + c \times d \neq c \times d + a \times b$
- In CUDA
  - fmaf(), fma() C functions
  - By default, compiler turns a*b+c expressions into FMAs
  - Use __fadd_rn(), __fmul_rn(), __dadd_rn(), __dmul_rn() in place of +, * to prevent FMAzation

Static rounding attributes

- On CPUs
  - Rounding mode as a mode for each thread
  - Get/set with e.g. fegetround() and fesetround()
- On NVIDIA GPUs
  - Rounding direction: flag in the instruction word
  - C intrinsics: __fadd_ru(), __fadd_rd(), __fmul_rz, __fmaf_rn…
- Benefit: zero-overhead mode switch
- Applications
  - Interval arithmetic
    - “Interval” CUDA SDK sample
    - 100× speedup for the same development effort
  - Stochastic arithmetic [JL10]

Conclusion

- GPU: throughput computing monster
  - Feed it with lots of threads (balanced ILP/DLP diet)
  - It likes: parallelism, locality, regularity (coherence)
- Specialized in FP arithmetic
  - From 8-bit fixed point to IEEE-754:2008 in 10 years
  - Now better FP support than on most CPUs
- Specialized in graphics
  - Exotic arithmetic units
- Can HPC learn from computer graphics?
  - Fixed-function units, memory compression?
- Next hardware feature?
  - Your feature?

FP OXO

	Format	FMA	UMA	Rounding	Subnormals	Inf, NaN	Flags	Exceptions
Intel X86	80-bit	✗	✗	4 Dynamic	Microcode	✓	✓	✓
IBM PowerPC	64-bit	✓	✗	4 Dyn.	Microcode	✓	✓	✓
Intel IA-64	82-bit	✓	✗	4 Dyn. + Stat.	Microcode	✓	✓	✓
IBM Cell SPU	32-bit	✓	✗	RZ	✗	✗	✓	✗
	64-bit	✓	✗	4 Dyn.	Output	✓	✓	✗
NVIDIA GT200	32-bit	✗	✓	2 Static	✗	✓	✗	✗
	64-bit	✓	✗	4 Static	✓	✓	✗	✗
AMD RV770	32-bit	✗	✓	RN	✗	✓	✗	✗
	64-bit	✗	✓	RN	✗	✓	✗	✗
NVIDIA GF100	32-bit	✓	✗	4 Static	✓	✓	✗	✗
	64-bit	✓	✗	4 Static	✓	✓	✗	✗
AMD Evergreen	32-bit	✓	✓	4 Dyn.	✓	✓	✓	✗
	64-bit	✓	✓	4 Dyn.	✓	✓	✓	✗

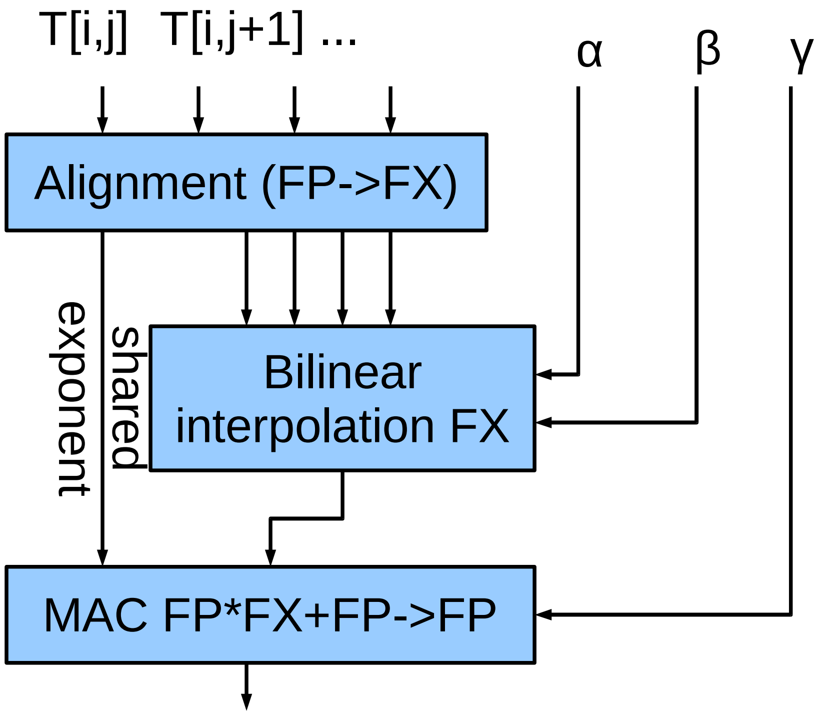
OpenCL 1.1	32-bit	Opt	N/A	RN	Opt	✓	✗	✗
	64-bit	✓		~~4 Static~~RN	✓	✓	✗	✗
Direct3D 11	32-bit	✗		RN	✗	✓	✗	✗
	64-bit	✗		RN	✓	✓	✗	✗

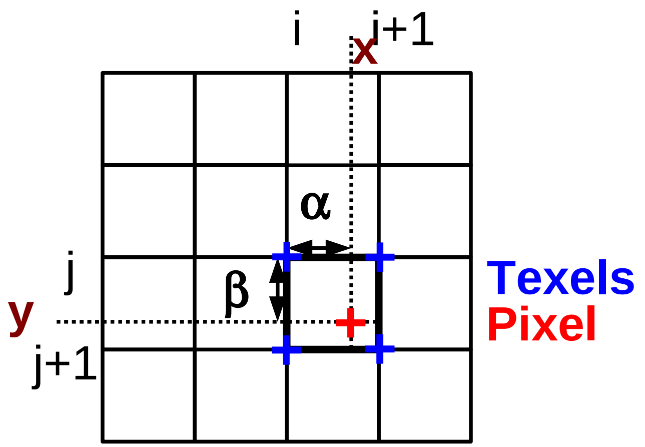
References

- [Glew09] Andy Glew. Coherent vector lane threading. Berkeley ParLab Seminar, 2009. http://parlab.eecs.berkeley.edu/seminars
- [CDT09] Sylvain Collange, David Defour, Arnaud Tisserand. Power consumption of GPUs from a software perspective. ICCS 2009.
- [Mici10] Paulius Micikevicius. Fundamental Performance Optimizations for GPUs. GTC 2010. http://developer.download.nvidia.com/compute/cuda/docs/GTC_2010_Arch
- [Strz10] Robert Strzodka. The Best of Both Worlds: Flexible Data Structures for Heterogeneous Computing. GTC 2010. http://developer.download.nvidia.com/compute/cuda/docs/GTC_2010_Arch
- [Volk10] Vasily Volkov. Better Performance at Lower Occupancy. GTC 2010. http://developer.download.nvidia.com/compute/cuda/docs/GTC_2010_Arch
- [ACD10] Mark Arnold, Sylvain Collange, David Defour. Implementing LNS using filtering units of GPUs. ICASSP, 2010.
- [JL10] Fabienne Jezequel, Jean-Luc Lamotte. Numerical validation of Slater integrals computation on GPU. SCAN 2010.

Texture filtering

- Fixed-function unit (ex NVIDIA GT200)
- Interpolate color of Pixels from Texels





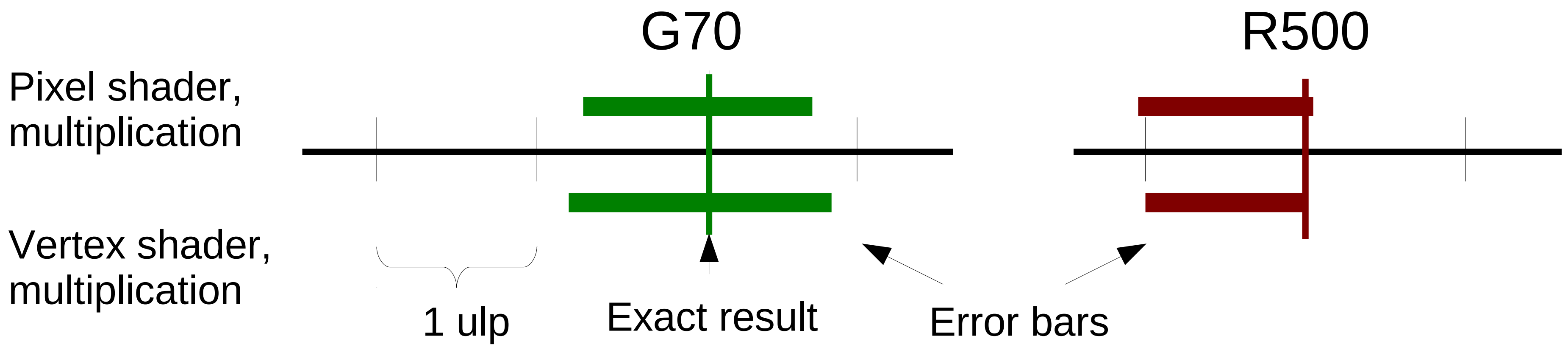
- Applications: graphics, image processing
- Can be hijacked to evaluate piecewise polynomials
  - Evaluate functions “for free” [ACD10]

Handling thread divergence

- Many techniques. e.g. Fermi:
  - Generic SIMT branch instruction
    - If all threads take the same path, treat as a branch
    - If not, fall back to predication
    - Handles nested control flow with a stack
  - Predication (for very short branches)
    - Take all paths, mask out unneeded calculations
  - Predicate-or-skip (for innermost conditionals)
    - Lighter version of generic mechanism
  - Select (for selective assignment)
- Compiler: selects which one to use
- Programmer: favor nondivergent conditionals
  - Regularity at algorithmic level

State of the art in 2006

- NVIDIA G70, ATI R500
- "Cray 1-like" floating-point arithmetic
  - Truncated multipliers
  - Adders with two guard bits and no sticky
  - 41 / 41 ≠ 1
  - Different behavior for different units on the same GPU



S. Collange, M. Daumas, D. Defour. État de l'intégration de la virgule flottante dans les processeurs graphiques. *RSTI – TSI 27/2008, p. 719 – 733.* 2008