Why are Graphics Systems so Fast?

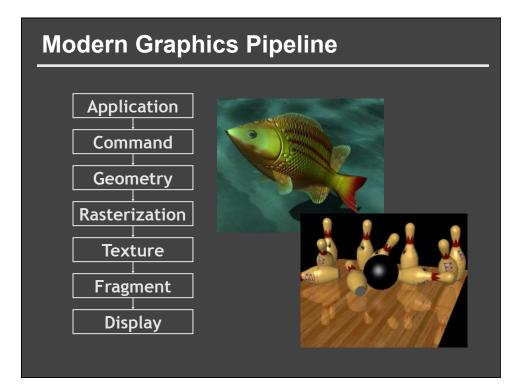
Pat Hanrahan

Pervasive Parallelism Laboratory Stanford University

> PACT Keynote September 14, 2009

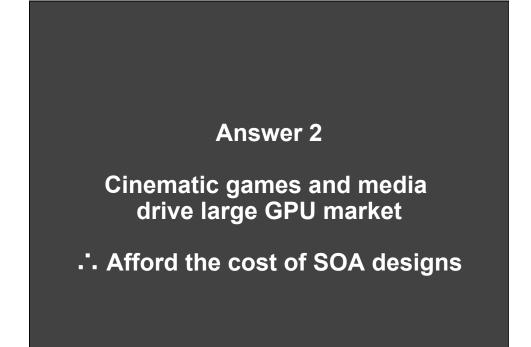
Answer 1

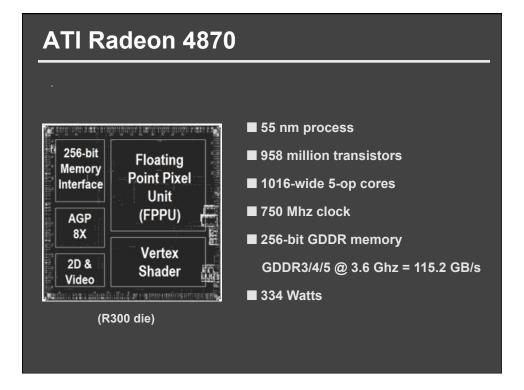
Simulating virtual worlds requires high performance



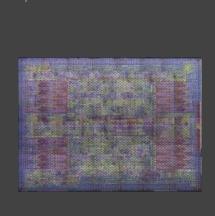
NVIDIA Historicals

Year	Product	Tri rate	CAGR	Tex rate	CAGR
1998	Riva ZX	3m	-	100m	-
1999	Riva TNT2	9m	3.0	350m	3.5
2000	GeForce2 GTS	25m	2.8	664m	1.9
2001	GeForce3	30m	1.2	800m	1.2
2002	GeForce Ti 4600	60m	2.0	1200m	1.5
2003	GeForce FX	167m	2.8	2000m	1.7
2004	GeForce 6800 Ultra	170m	1.0	6800m	2.7
2005	GeForce 7800 GTX	940m	3.9	10300m	2.0
2006	GeForce 7900 GTX	1400m	1.5	15600m	1.4
2007	GeForce 8800 GTX	1800m	1.3	36800m	2.3
2008	GeForce GTX 280			48160m	1.3
			1.7		1.8





NVIDIA GTX 280



- 65 nm TSMC process
- 1.4 biillion transistors
- 575 mm^2
- 240 scalar processors

1.3 Ghz clock rate

■ 512-bit GDDR memory
 GDDR @ 1.1 Ghz = 141.7 GB/s
 ■ 236 Watts

Answer 3

GPUs efficiently use semiconductor technology

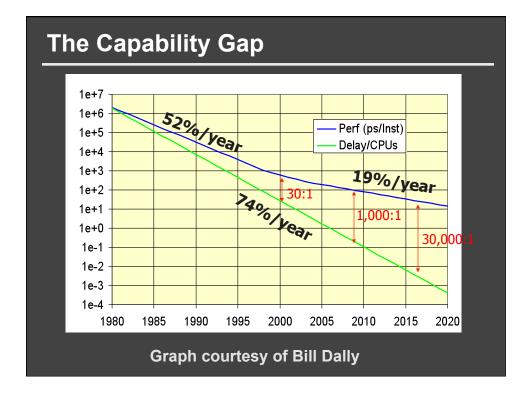
Scaling Laws

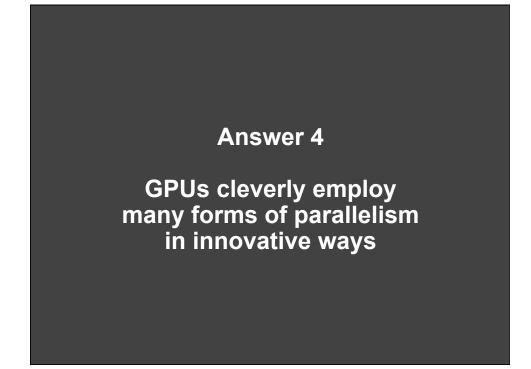
Moore's Law

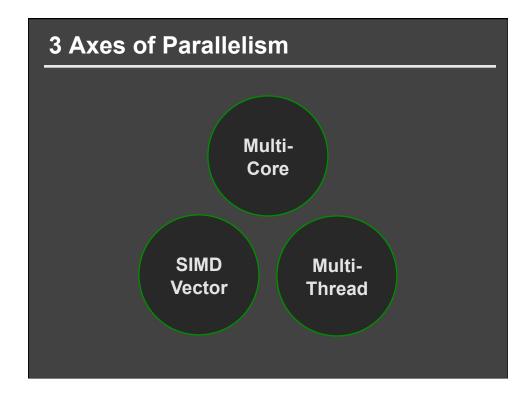
- Number of transistors doubles every 18 months
- Number of transistors increases by ~50% / yr
- Feature size decreases by ~25% / yr
- Gate delay decreases with feature size by ~25% / yr

Semiconductor capability = Number of transistors / Switching speed

50% (number) + 25% (speed)



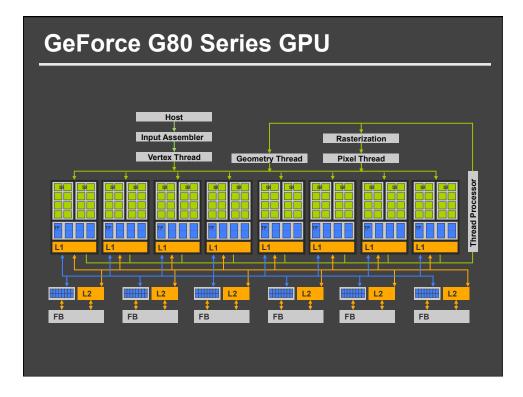


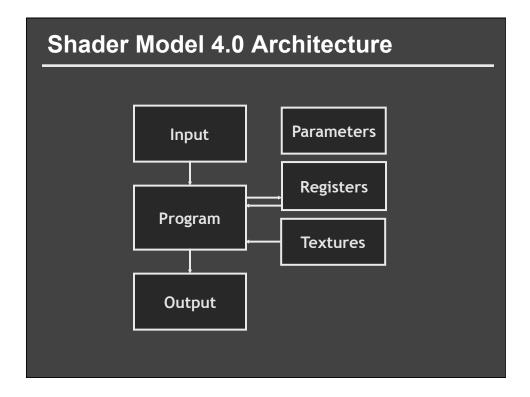


GPU Architectures

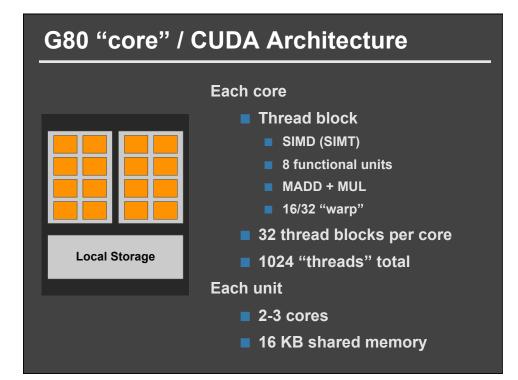
A Closer Look at GPUs Kayvon Fatahalian and Mike Houston Communications of the ACM. Vol. 51, No. 10 (October 2008)

From Shader Code to a Teraflop: How Shader Cores Work Kayvon Fatahalian Beyond Programming Shading, SIGGRAPH 2009 Course Notes

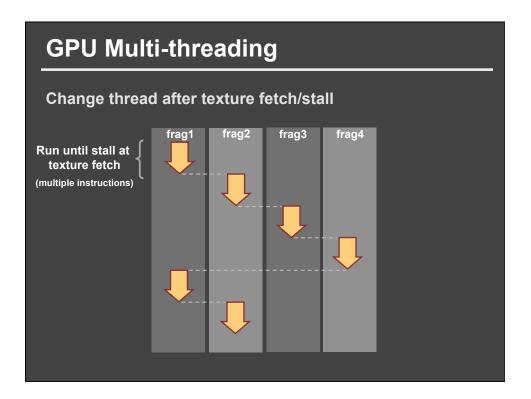


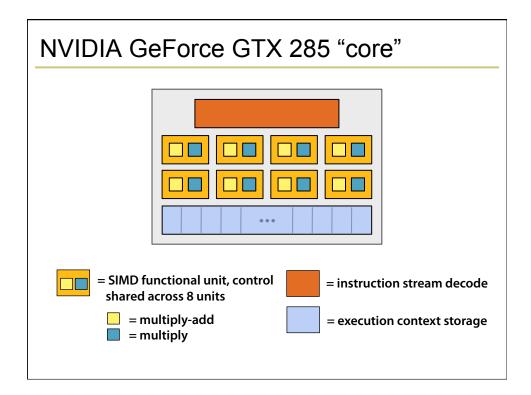


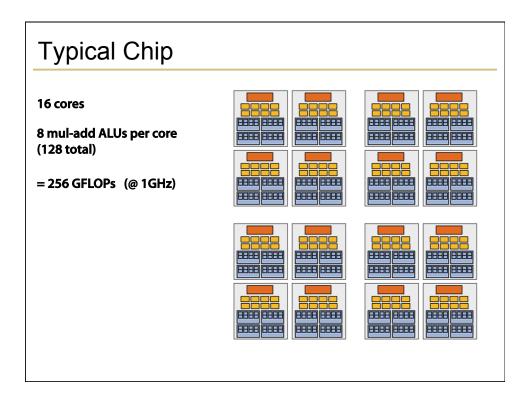
<pre>DP4 o[HPOS].x, c[0], v[OPOS]; # Transform pc DP4 o[HPOS].y, c[1], v[OPOS]; DP4 o[HPOS].z, c[2], v[OPOS]; DP4 o[HPOS].w, c[3], v[OPOS]; DP3 R0.x, c[4], v[NRML]; # Transform nc</pre>	
DP4 o[HPOS].y, c[1], v[OPOS]; DP4 o[HPOS].z, c[2], v[OPOS]; DP4 o[HPOS].w, c[3], v[OPOS];	
DP4 $o[HPOS].z, c[2], v[OPOS];$ DP4 $o[HPOS].w, c[3], v[OPOS];$	os.
DP4 $o[HPOS].w, c[3], v[OPOS];$	
DP3 R0.x, c[4], v[NRML]; # Transform no	
	ormal.
DP3 R0.y, $c[5]$, $v[NRML]$;	
DP3 R0.z, c[6], v [NRML];	
DP3 R1.x, c[32], R0; # R1.x = L DO1	r n'
DP3 R1.y, c[33], R0; # R1.y = H DO1	r n'
MOV R1.w, c[38].x; # R1.w = specu	ılar
LIT R2, R1; # Compute ligh	nting
MAD R3, c[35].x, R2.y, c[35].y; # diffuse + an	nbient
<pre>MAD o[COL0].xyz, c[36], R2.z, R3; # + specular</pre>	
END	

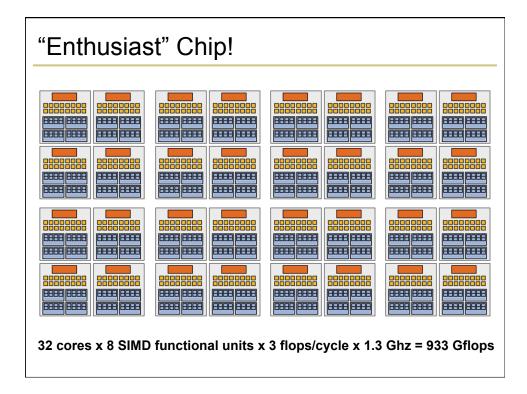


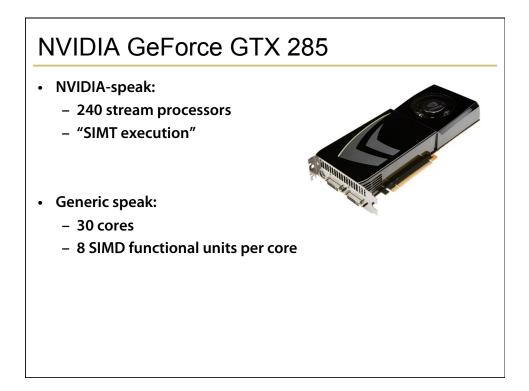
Critical	Inner Loop	for Graphics	
ps_2_0 DCL DCL DCL_2D TEX1D MUL	t0.xy v0.xyzw s0 r0, t0, s0 r1, r0, v0	<pre># Interpolate t0.xy # Interpolate v0.xyzw # Declaration - no code # TEXTURE LOAD! # Multiply</pre>	
MOU MOV	oC0, r1	# Multiply # Store to framebuffer	
The program must run at 100% efficiency Short inner loop Very little state (few registers) Random memory (texture) access			

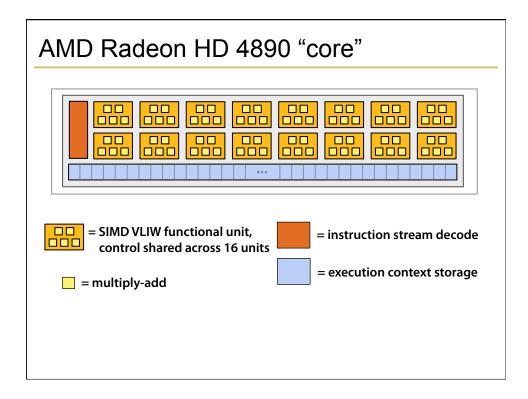


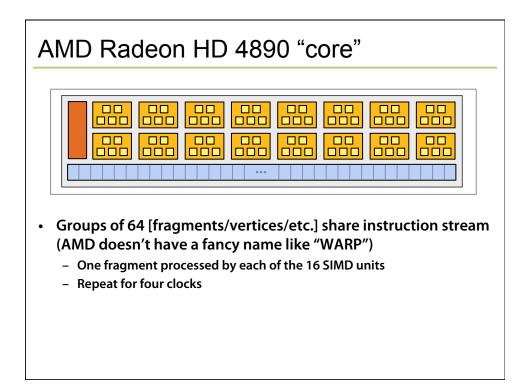


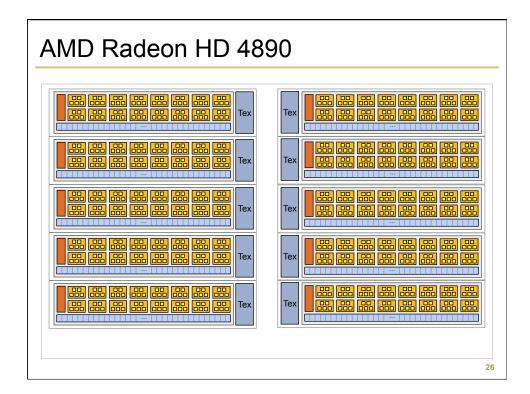












AMD Radeon HD 4890

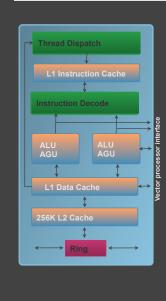
- AMD-speak:
 - 800 stream processors
 - HW-managed instruction stream sharing (like "SIMT")
- Generic speak:
 - 10 cores
 - 16 SIMD functional units per core
 - 5 ALUs per VLIW unit per SIMD lane



27

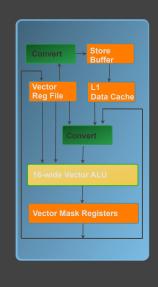
Larrabee: A many-core x86 architecture for visual computing, D. Carmean, E. Sprangle, T. Forsythe, M. Abrash, L. Seiler, A. Lake, P. Dubey, S. Junkins, J. Sugerman, P. Hanrahan, SIGGRAPH 2008 (IEEE Micro 2009, Top Pick)

Larrabee Core



Separate scalar and vector units Separate register files In-order IA scalar core Vector unit: 16 32-bit ops/clock Short execution pipelines Fast access from L1 cache Direct connection to L2 cache Prefetch to manage L1/L2 caches

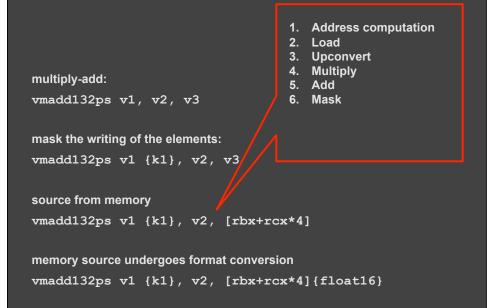
Vector Processing Unit

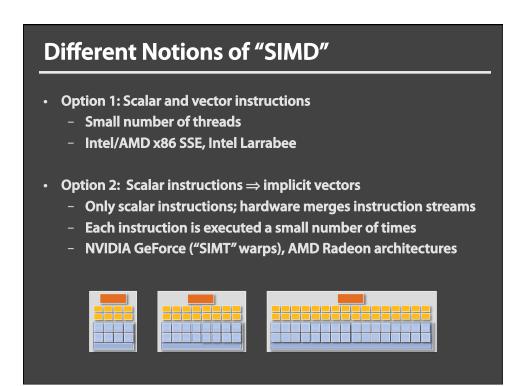


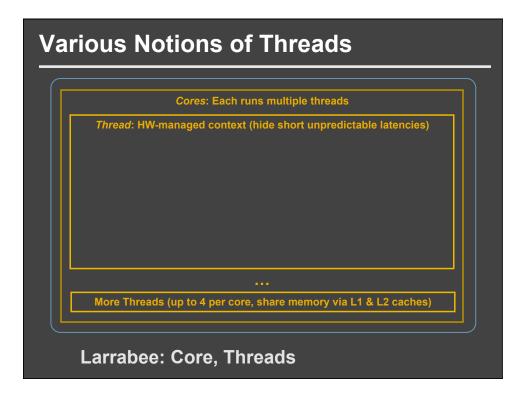
Vector instructions support

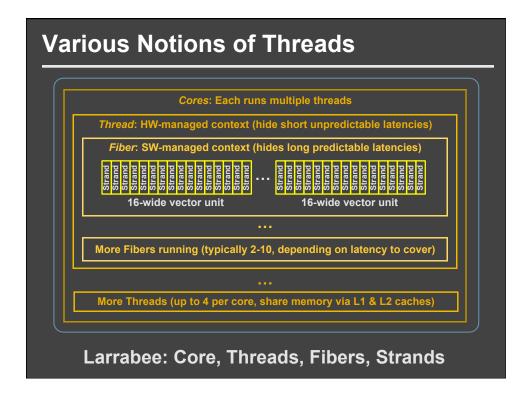
- Fast, wide read from L1 cache
- Numeric type conversion and data
- Rearrange the lanes on register read
- Fused multiply add (three arguments)
- Int32, Float32 and Float64 data
- Augmented vector instruction set
 - Scatter/gather for vector load/store
 - Mask registers select lanes

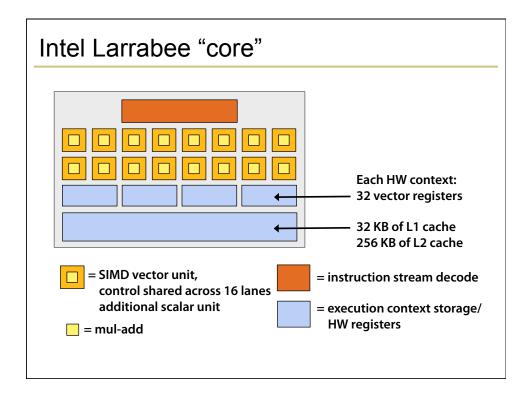
Example LRBni Vector Instructions

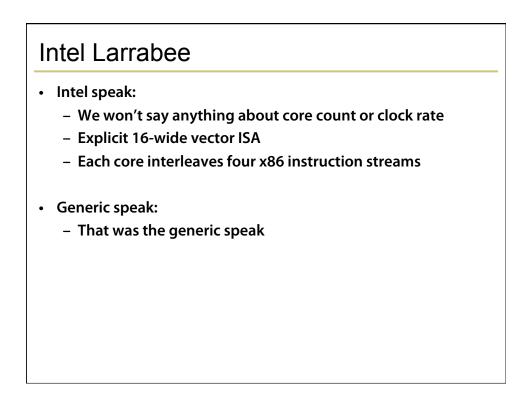


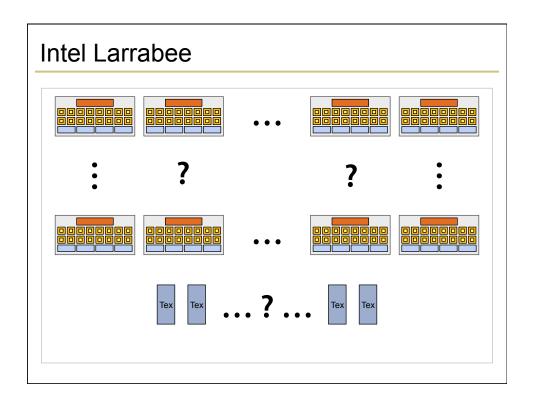


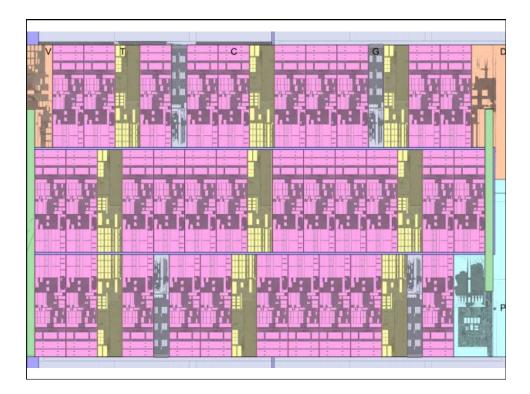












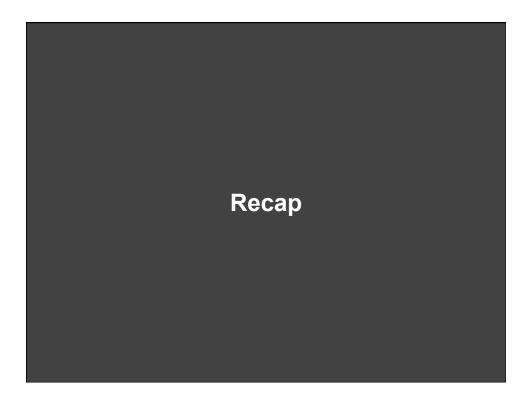
Larrabee

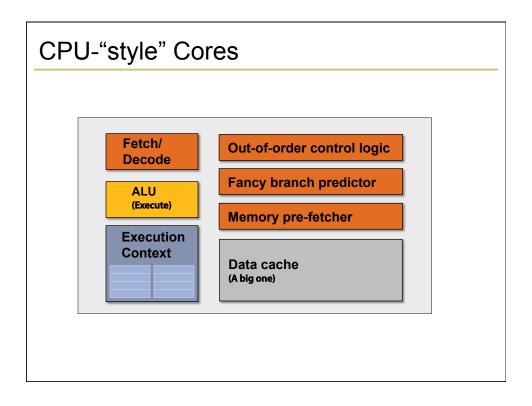
Each Larrabee core is a complete IA core

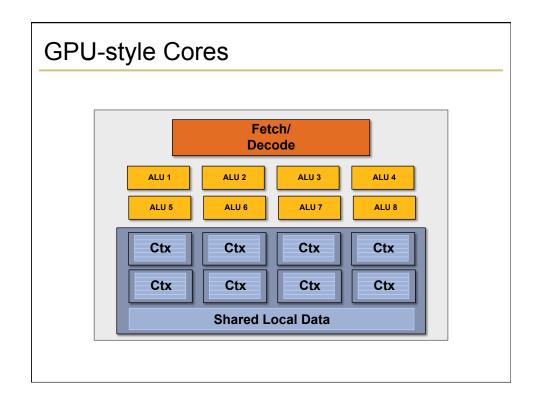
- Context switching & pre-emptive multi-tasking
- Virtual memory and page swapping
- Fully coherent caches at all levels of the hierarchy

Efficient inter-block communication

- Ring bus for full inter-processor communication
- Low latency high bandwidth L1 and L2 caches
- Fast synchronization between cores and caches







Three Key Ideas

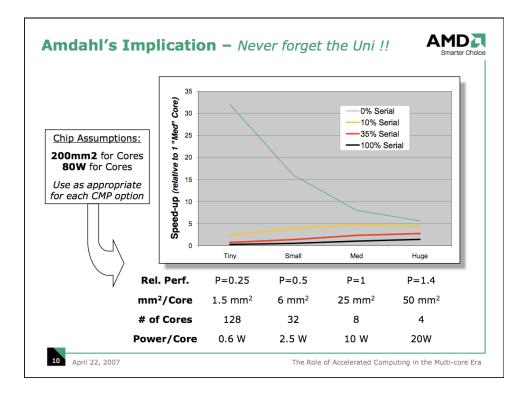
- **1.** Simplify the core.
 - Remove high-overhead logic to control out of order execution, branch predication, etc.
- 2. Exploit the efficiency of SIMD processing
 - Share instructions and replicate functional units
- 3. Use many threads to hide memory latency
 - Smaller caches, but still need thread state
 - If you have enough thread state, never a stall

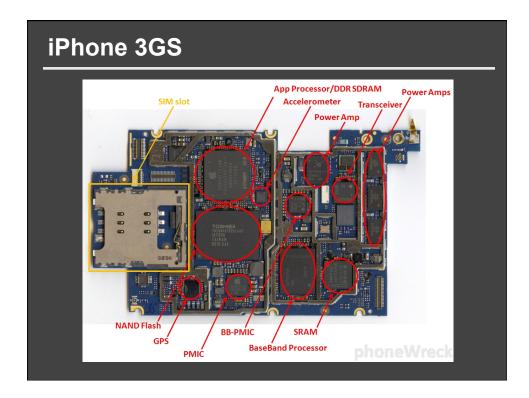
Optimizing for Throughput

Hypothetical Core design experiment: Specify a throughput-optimized processor with same area and power of a standard dual core CPU

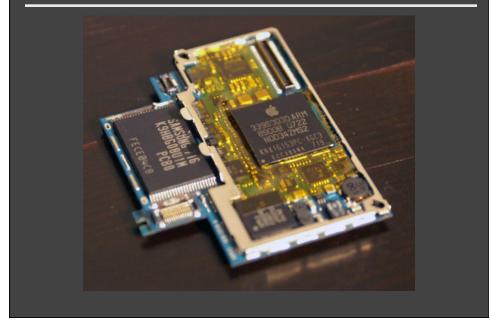
# CPU cores	2 out of order	10 in-order
Instructions per issue	4 per clock	2 per dock
VPU lanes per core	4-wide SSE	16-wide
L2 cache size	4 MB	4 MB
Single-stream	4 per clock	2 per clock
Vector throughput	8 per clock	160 per dock

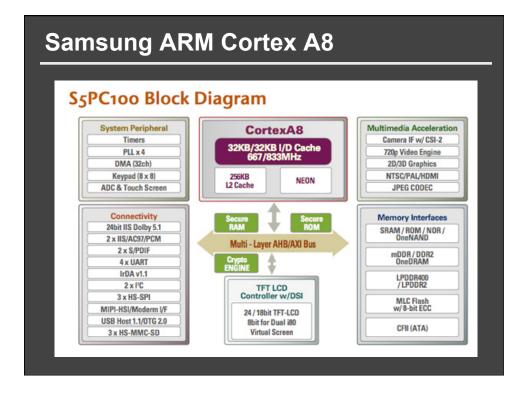
20 times greater throughput for same area and power

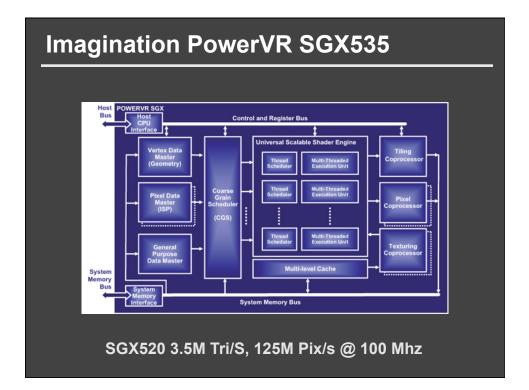


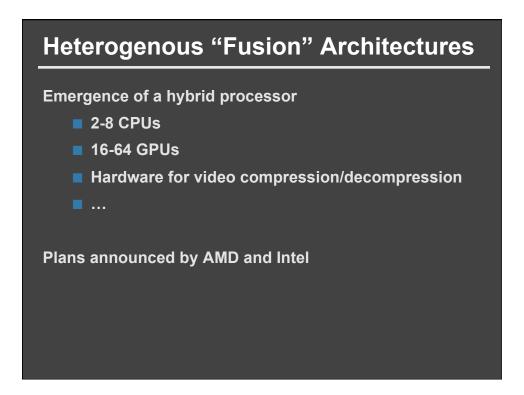


Apple/Samsung SoC (CPU, GPU, Mem)









Answer 5

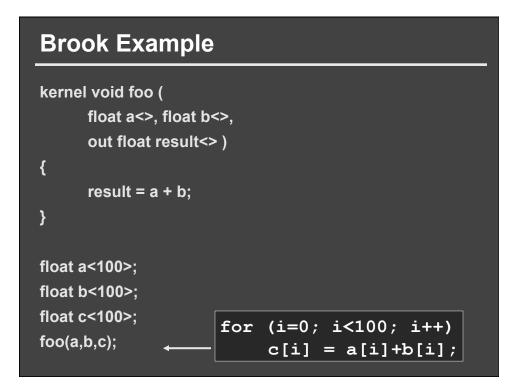
Graphics Systems are Programmed at a High-Level of Abstraction (Utilize Domain-Specific Languages)

Brook

Ian Buck PhD Thesis Stanford University

Brook for GPUs: Stream computing on graphics hardware, I. Buck, T. Foley, D. Horn, J. Sugarman, K. Fatahalian, M. Houston, P. Hanrahan, SIGGRAPH 2004

CUDA: Scalable parallel programming made clear, J. Nickolls, I. Buck, K. Skadron, and M. Garland, ACM Queue, April 2008



Current Statistics: September 13, 2009

Client type	Current TFLOPS*	Active Processors
Windows	215	225,721
Mac OS X/Intel	22	5,063
Linux	77	45,028
ATI	1,027	10,069
NVIDIA	1,992	16,736
PS/3	1,075	38,110
Total	4,412	347,825

Domain-Specific Languages

Graphics (GRAMPS) Molecular dynamics (GROMACS) Physical simulation on meshes (Liszt) Data-parallel programming (Kore) Statistics/machine learning and data analysis (Bern) Computer vision and imaging Brain simulation Autonomous vehicles



Questions and Answers

Why are graphics systems so fast?

- 1. Simulating virtual worlds requires high performance
- 2. Cinematic games and media drive large GPU market
- 3. GPUs (more) efficiently use semiconductor resources
- 4. GPUs employ many forms of parallelism in innovative ways (core, thread, vector)
- 5. GPUs are programmed at a high-level

Why are other compuer systems so slow / inefficient?

Architectural Issues

High-throughput processor design

- SIMD vs. blocked threads (SIMT)
- Software- vs. hardware-managed threads

Processor of the future likely to be a hybrid CPU/GPU

- Why? Heterogeneous workload
- Small number of traditional CPU cores running a moderate number of sequential tasks
- Large number of high-throughput GPU cores running data-parallel work
- Special hardware for tasks that need to be power efficient

Opportunities

Current hardware not optimal

Incredible opportunity for architectural innovation

Current software environment immature

Incredible opportunity for reinventing parallel computing software, programming environments and language

Acknowledgements

Bill Dally	lan Buck
Alex Aiken	Kayvon Fatahalian
Eric Darve	Tim Foley
Vijay Pande	Daniel Horn
Bill Mark	Michael Houston
John Owens	Jeremy Sugerman
Kurt Akeley	Doug Carmean
Mark Horowitz	Michael Abrash
Funding: DARPA, DOE, ATI,	IBM, INTEL, NVIDIA, SONY

