Multiple Cores, Multiple Pipes, Multiple Threads – Do we have more Parallelism than we can handle?

David B. Kirk

Outline

- Types of Parallelism
- Processor / System Parallelism
- Application / Problem Parallelism
- CPU Multi-core Roadmaps
- GPU Evolution and Performance Growth
- Programming Models
- Mapping Application Parallelism onto Processors
- The Problem of Parallelism

Processor / System Parallelism

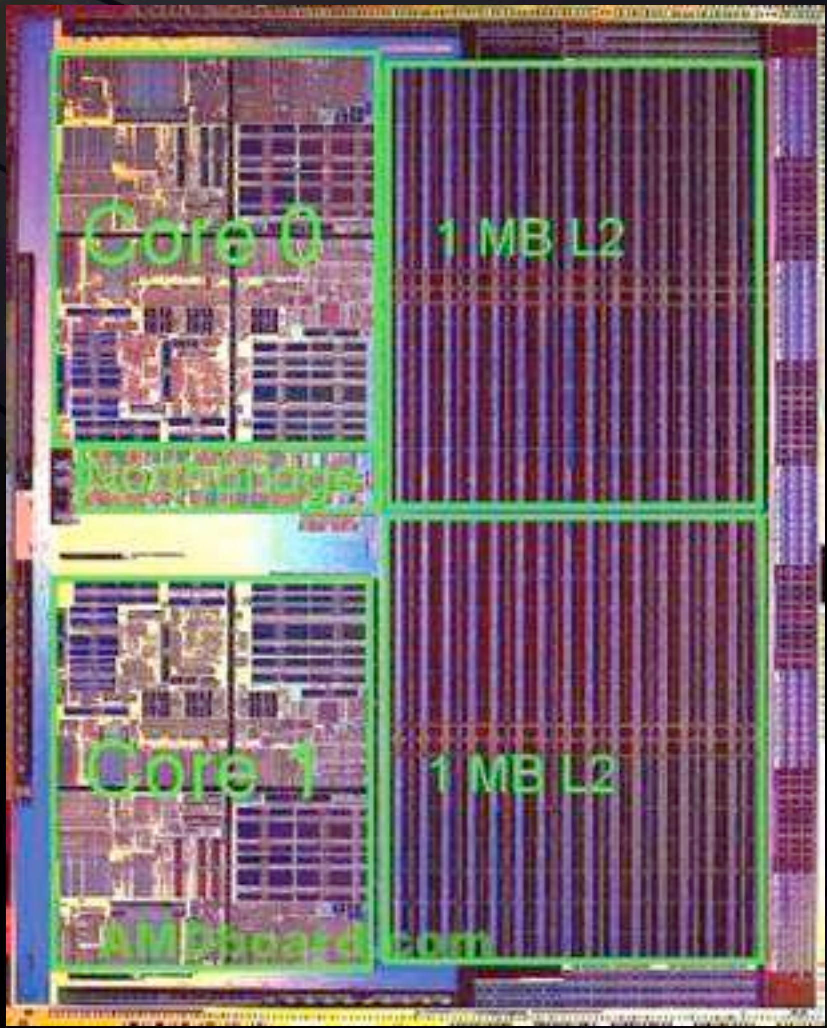
- Single- vs. Multi- core
- Fine- vs. Coarse- grained
- Single- vs. Multi- pipeline
- Vector vs. Scalar math
- Data- vs. Thread- vs. Instruction-level- parallel
  - Not mutually exclusive
- Single- vs. Multi-threaded processor
- Message-passing vs. Shared Memory communication
- SISD, SIMD, MIMD…
- Tightly- vs. Loosely-connected cores & threads

Application / Problem Parallelism

- If there is no parallelism in the workload, processor/system parallelism doesn't matter!
- Large problems can (often) be more easily parallelized
- "Good" parallel behavior
  - Many inputs/results
  - Parallel structure – many similar computation paths
  - Little interaction between data/threads
- Data parallelism easy to map to machine "automagically"
- Task parallelism requires programmer forethought

General Purpose Processors

- Single-core…
- Dual-core…
- Multi-core
- Limited multi-threading
- No special support for parallel programming
  - Coarse-grained
  - Thread-based
  - Requires programmer awareness



Processor die photo courtesy of AMD

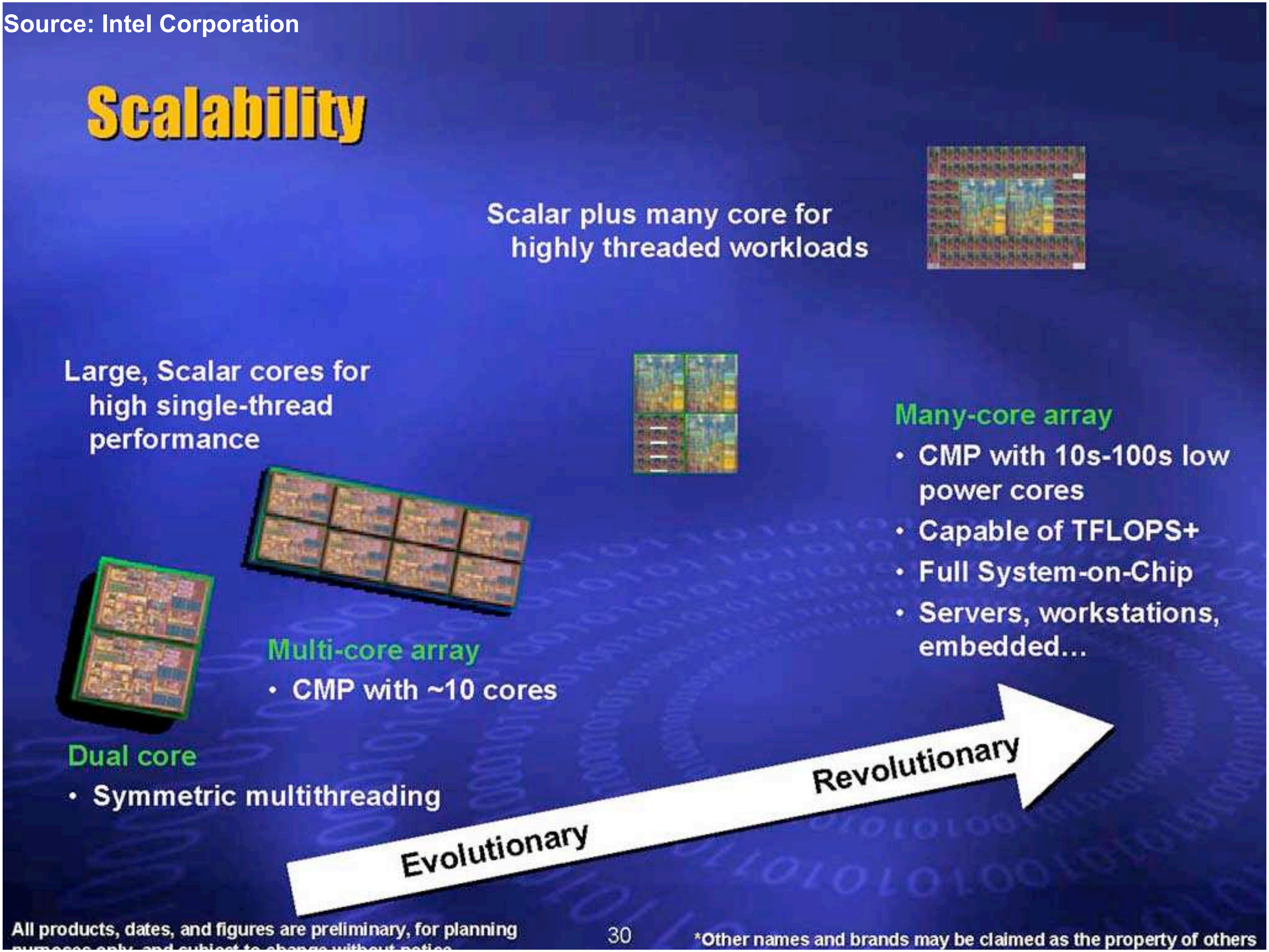
Source: Intel Corporation

The Move to Intel Multi-core

Platform	Current	2005	2006	2007
Itanium® processor MP	Itanium® 2 Processor	Montecito	Montvale	Tukwila Poulson
Itanium® processor DP	Itanium® 2 Processor - 3M (Fanwood)	Millington	DP Montvale	Dimona
MP Server	Intel® Xeon™ Processor MP	64-bit Intel® Xeon™ processor MP	Paxville Tulsa	Whitefield
DP Server / WS	64-bit Intel® Xeon™ Processor w/ 2MB cache		Dempsey	Woodcrest
Desktop Client	Pentium® 4 processor	Pentium® Processor Extreme Edition Pentium D Processor	Presler	Conroe
		Pentium® 4 processor	Cedar Mill	
Mobile Client	Pentium® M processor		Yonah	Merom
			Yonah	

Single core | Dual-core | >= 2 cores

All products and dates are preliminary and subject to change without notice.


Source: Intel Corporation
Scalability
Scalar plus many core for highly threaded workloads
Large, Scalar cores for high single-thread performance
Many-core array
• CMP with 10s-100s low power cores
• Capable of TFLOPS+
• Full System-on-Chip
• Servers, workstations, embedded...
Multi-core array
• CMP with ~10 cores
Dual core
• Symmetric multithreading
Revolutionary
Evolutionary
All products, dates, and figures are preliminary, for planning
30
*Other names and brands may be claimed as the property of others

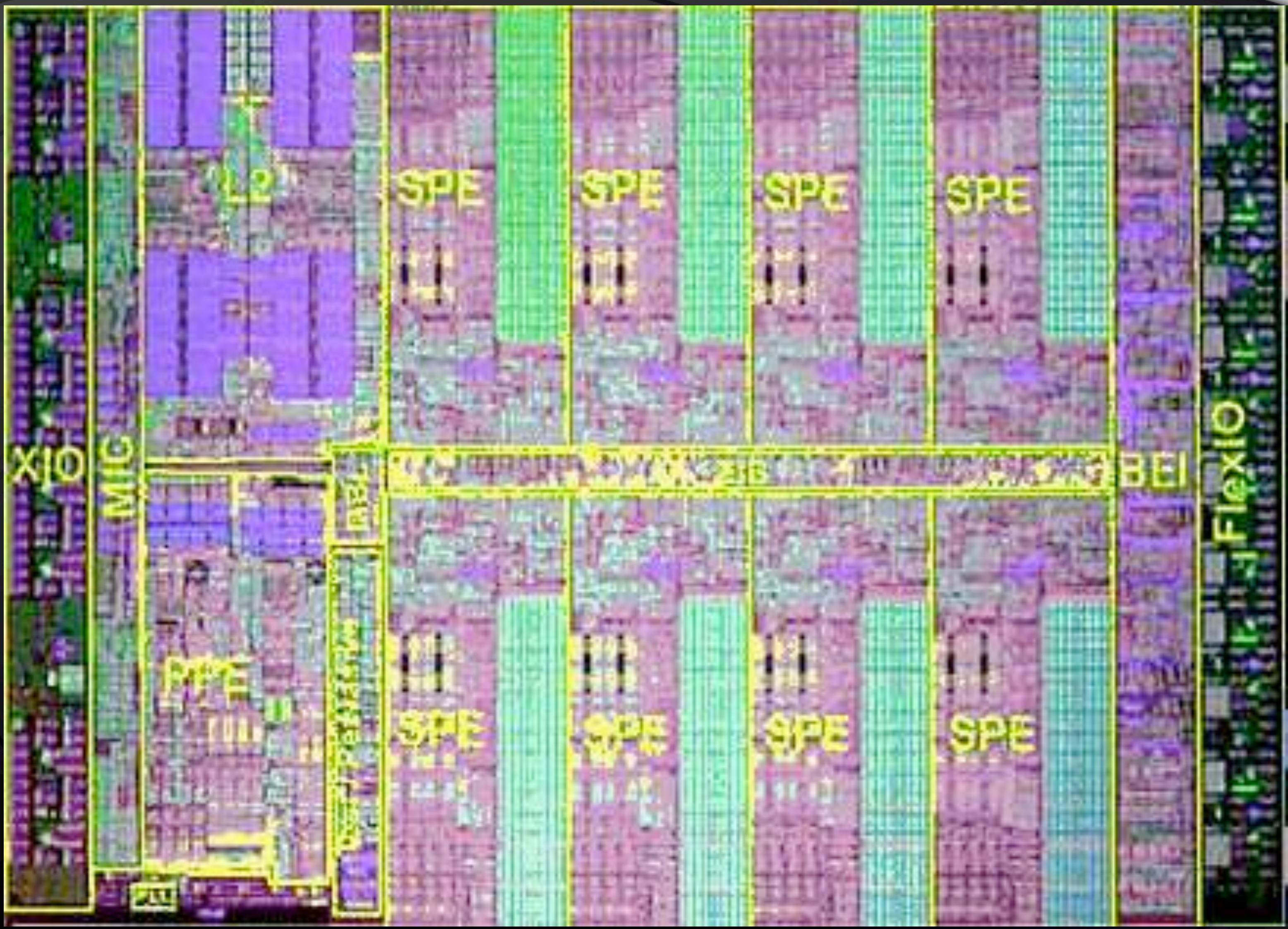
CPU Approach to Parallelism

- Multi-core
- Limited multi-threading
- Coarse-grained
- Scalar (usually)
- Explicitly Threaded
  - Application parallelism must exist at coarse scale
  - Programmer writes independent program thread for each core

Multi-Core Application Improvement: General Purpose CPU

- Single-thread Applications:
  - 0% speedup
- Multi-thread Applications:
  - Up to X times speedup, where X is # of cores * hardware multi-threading
  - Potential for cache problems / data sharing & migration
- % of Applications that are Multi-threaded
  - Essentially 0 ☹
- But… HUGE potential ☺
  - Multi-threaded OS's will benefit
  - Just add software for apps…

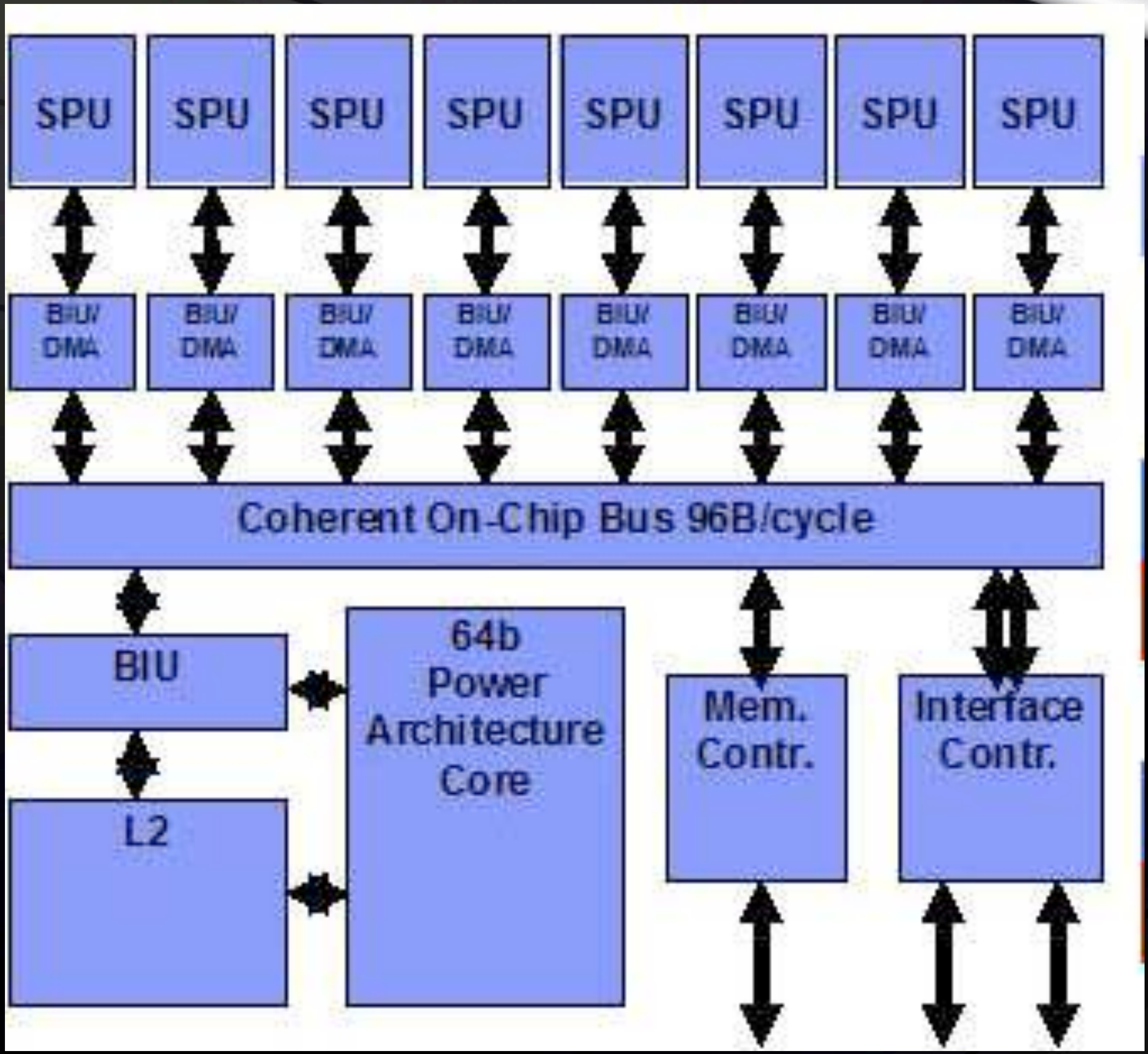
Special-purpose Parallel CPUs



Cell Processor die photo courtesy of IBM

Cell Processor

- What is the programming model?
- Can you expect programmers to explicitly schedule work / data flow?



Cell Processor Approach to Parallelism

- Multi-core
- Coarse-grained
- Vector
- Explicitly Threaded
  - Programmer writes independent program thread code for each core
- Explicit Data Sharing
  - Programmer must copy or DMA data between cores

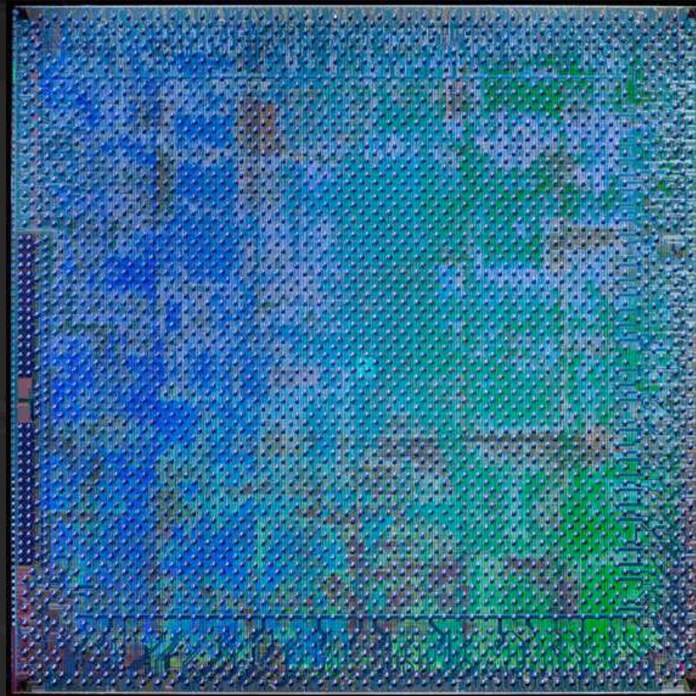
Multi-Core Application Improvement: Special Purpose CPU (Cell)

- Single-thread Applications:
  - 0% speedup
- Multi-thread Applications:
  - Up to X times speedup, where X is # of cores
  - Up to Y times speedup, where Y is vector width
  - Explicit software management of cache / data sharing & migration

- % of Applications that are Multi-threaded
  - Essentially 100% ☺ (all applications are written custom)

- But… HUGE software development effort ☹

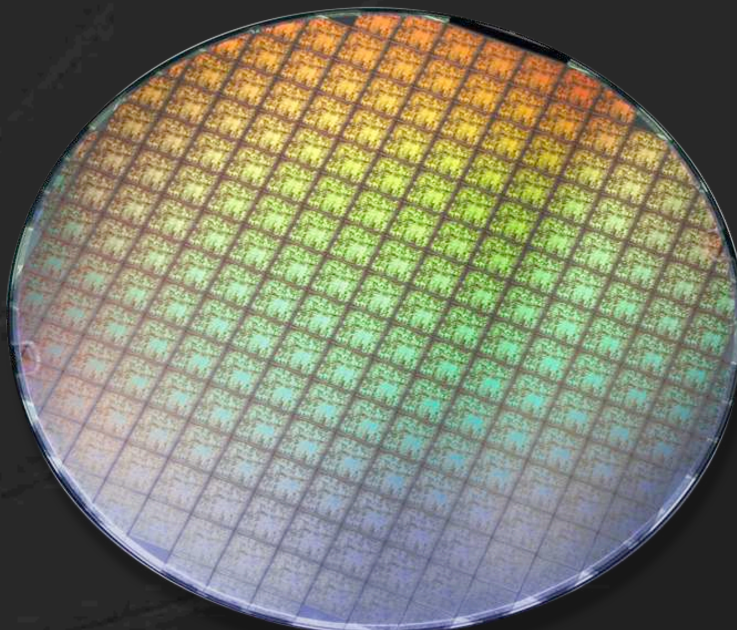
GeForce 7800 GTX: Most *Capable* Graphics Processor Ever Built



302M Transistors

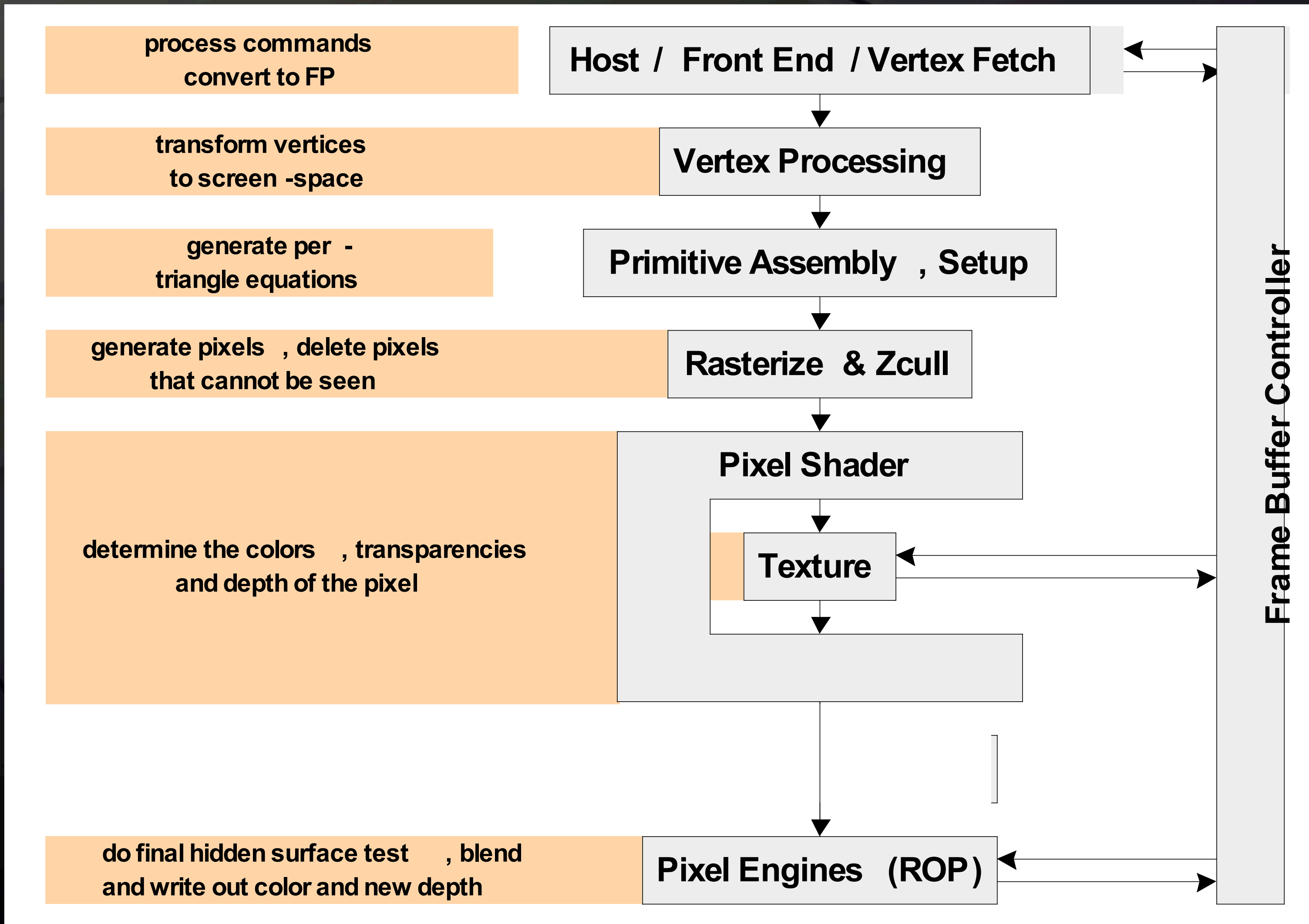
- \+ XBOX GPU (60M)
- \+ PS2 Graphics Synthesizer (43M)
- \+ Game Cube Flipper (51M)
- \+ Game Cube Gekko (21M)
- \+ XBOX Pentium3 CPU (9M)
- \+ PS2 Emotion Engine (10.5M)
- \+ Athlon FX 55 (105.9M)

300.4M



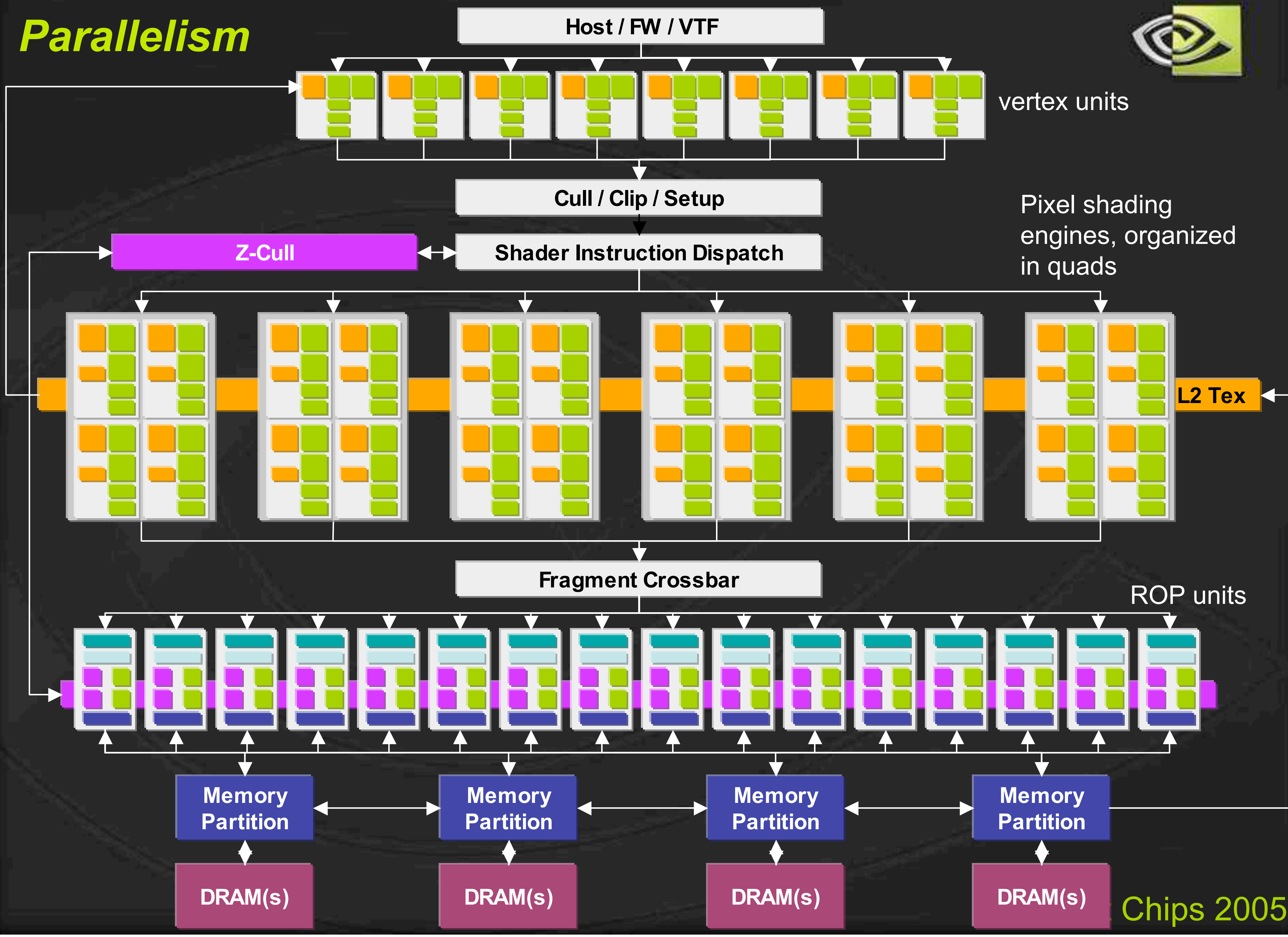

IEEE Hot Chips 2005

The Life of a Triangle

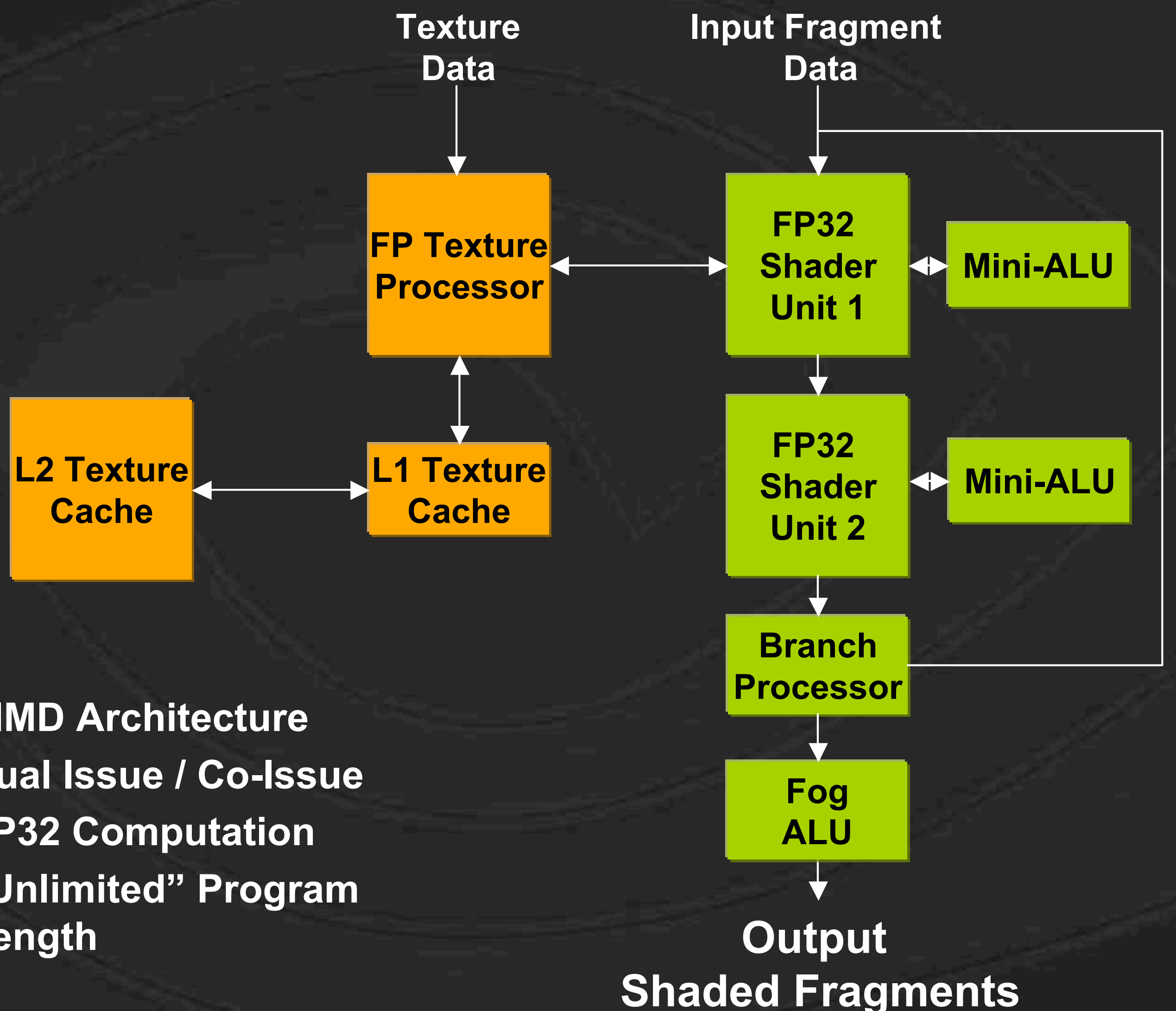


GeForce 7800

Parallelism



Detail of a single pixel shader pipeline



- SIMD Architecture
- Dual Issue / Co-Issue
- FP32 Computation
- “Unlimited” Program Length

Arithmetic Density

Delivered 32-bit Floating Point performance

- **Big GFlop #'s are nice…**

  …but what can you actually measure from a basic test program?

	Clock	vec4 MAD Ginstructions	**Gflops**
GeForce 6800 Ultra	425	6.7568	**54.0544**
GeForce 7800 GTX	430	20.6331	**165.0648**

Using a test pixel shader program that simply measures how many 4-component MAD instructions can be executed per second.

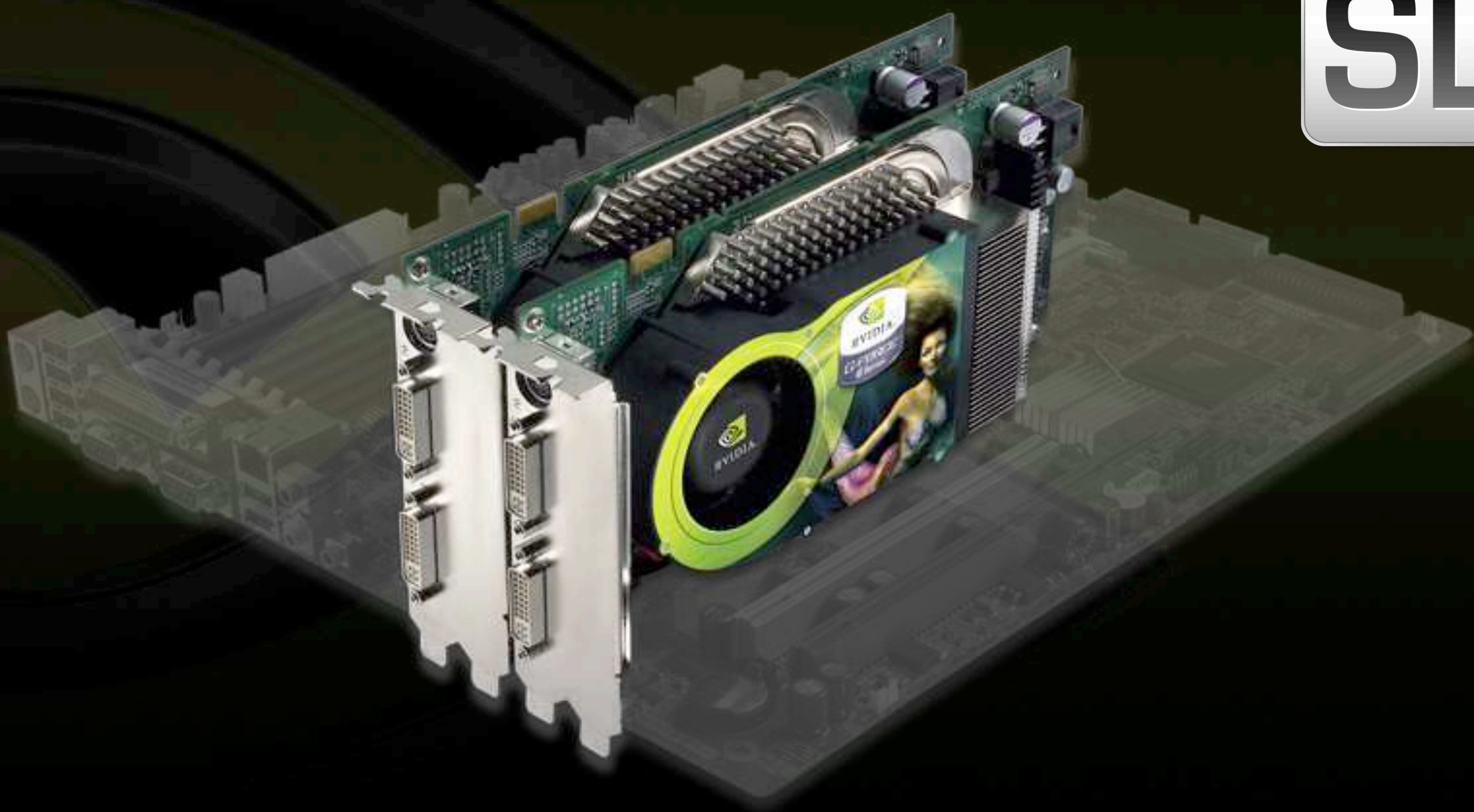
GPU Approach to Parallelism

- Single-core
- Multi-pipeline
- Multi-threaded
- Fine-grained
- Vector
- Explicitly and Implicitly Threaded
  - Programmer writes sequential program thread code for shader processors
  - Thread instances are spawned automatically
- Data Parallel
  - Threads don't communicate, except at start/finish

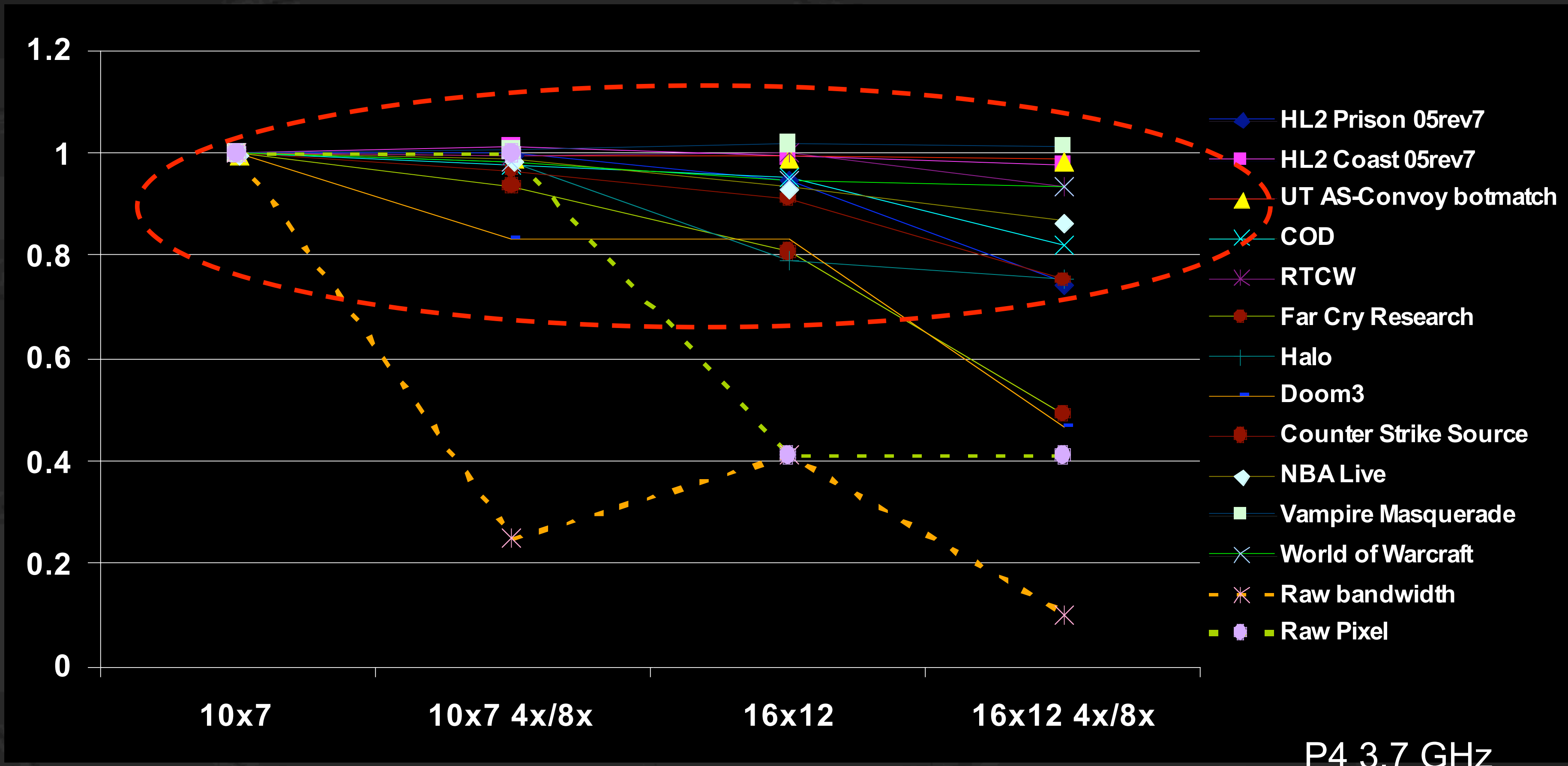
Multi-Pipeline Application Improvement: GPU

- Multi-thread Applications:
  - Up to X times speedup, where X is # of pipelines
  - Exploits x4 vector FP MADs
  - Very little software management of cache / data sharing & migration
- % of Applications that are Multi-threaded
  - Essentially 100% ☺ (all applications are written custom)
- Again… HUGE software development effort ☹
- Limited by CPU throughput ☹

SLI – Dual GPUs in a Single System

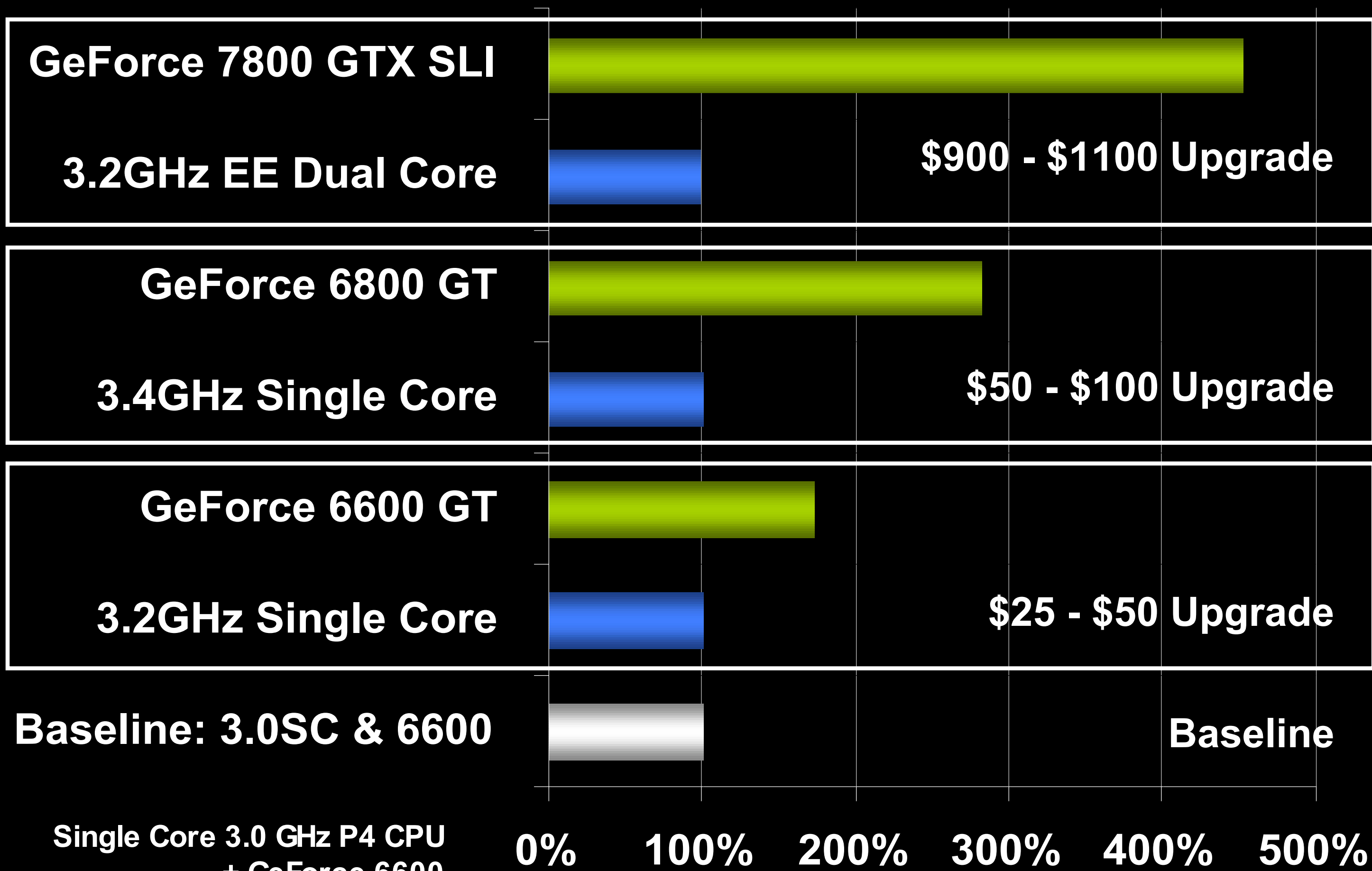


CPU Limitedness – Exploitation of GPU Parallelism limited by CPU



P4 3.7 GHz
1 GB RAM

Game Performance Benefits of Dual-core vs. GPU Upgrade



Single Core 3.0 GHz P4 CPU
+ GeForce 6600
Performance improvement based on Doom3
1024x768x32, 4x AA, 8x Aniso filtering

GPU Programming Languages

- DX9 (Direct X)
  - assembly coding for vertex and pixel
  - HLSL (High-Level Shading Language)
- OpenGL 1.3+
  - assembly coding for vertex and pixel
  - GLSLang
  - Cg
- Brook for GPUs (Stanford)
  - HLL for GPGPU layered on DX/OGL
  - http://graphics.stanford.edu/projects/brookgpu/
- SH for GPUs (Waterloo)
  - Metaprogramming Language for GPGPU
  - http://libsh.sourceforge.net/docs.html
- Others...

Importance of Data Parallelism for GPUs

- GPUs are designed for graphics
  - Highly parallel tasks
- GPUs process *independent* vertices & fragments
  - Temporary registers are zeroed
  - No shared or static data
  - No read-modify-write buffers
  - Opportunity exists when # of independent results is large
- Data-parallel processing
  - GPUs architecture is ALU-heavy
    - Multiple vertex & pixel pipelines, multiple ALUs per pipe
  - Large memory latency, but HUGE memory bandwidth
  - Hide memory latency (with more computation)

Language Support for Parallelism

- Most widely-used programming languages are terrible at exposing potential parallelism

```
for (int i=0; i<100; i++) {
    // compute i'th element ...
}
```

- LISP and other functional languages are marginally better

```
(+ (func 1 4) (func 23 9) (func 6 2))
```

- Some direct support: Fortran 90, HPF
  - Not in common use for development anymore
- Research in true parallel languages has stagnated
  - Early 1990's: lots of research in C*, DPCE, etc.
  - Late 1990's on: research moved to JVM, managed code, etc

Parallel Programming: Not just for GPUs

- CPU's benefit, too
  - SSE, SSE2, MMX, etc.
  - Hyperthreading
  - Multi-core processors announced from Intel, AMD, etc.
  - Playstation2 Emotion Unit
  - MPI, OpenMP packages for coarse grain communication
- Efficient execution of:
  - Parallel code on a serial processor: EASY
  - Serial code on a parallel processor: HARD
- The impact of power consumption further justifies more research in this area – parallelism is the future

PC Graphics growth (225%/yr)
Sustainable Growth on Capability Curve

Season	tech	#trans	Gflop*	Mpix	Mpoly	Mvector
...						
spring/00	0.18	25M	35	800	30M	50M
fall/00	0.15	50M	150	1.2G	75M	100M
spring/01	0.15	55M	180	2.0G	100M	200M
fall/01	0.13	100M	280	4.0G	200M	400M
...						
spring/03 (NV30)	0.13	140M	500	8.0G**	300M	300M
spring/04 (NV40)	0.13	220M	1000	25.6G**	600M	600M

* Special purpose math, not all general purpose programmable math

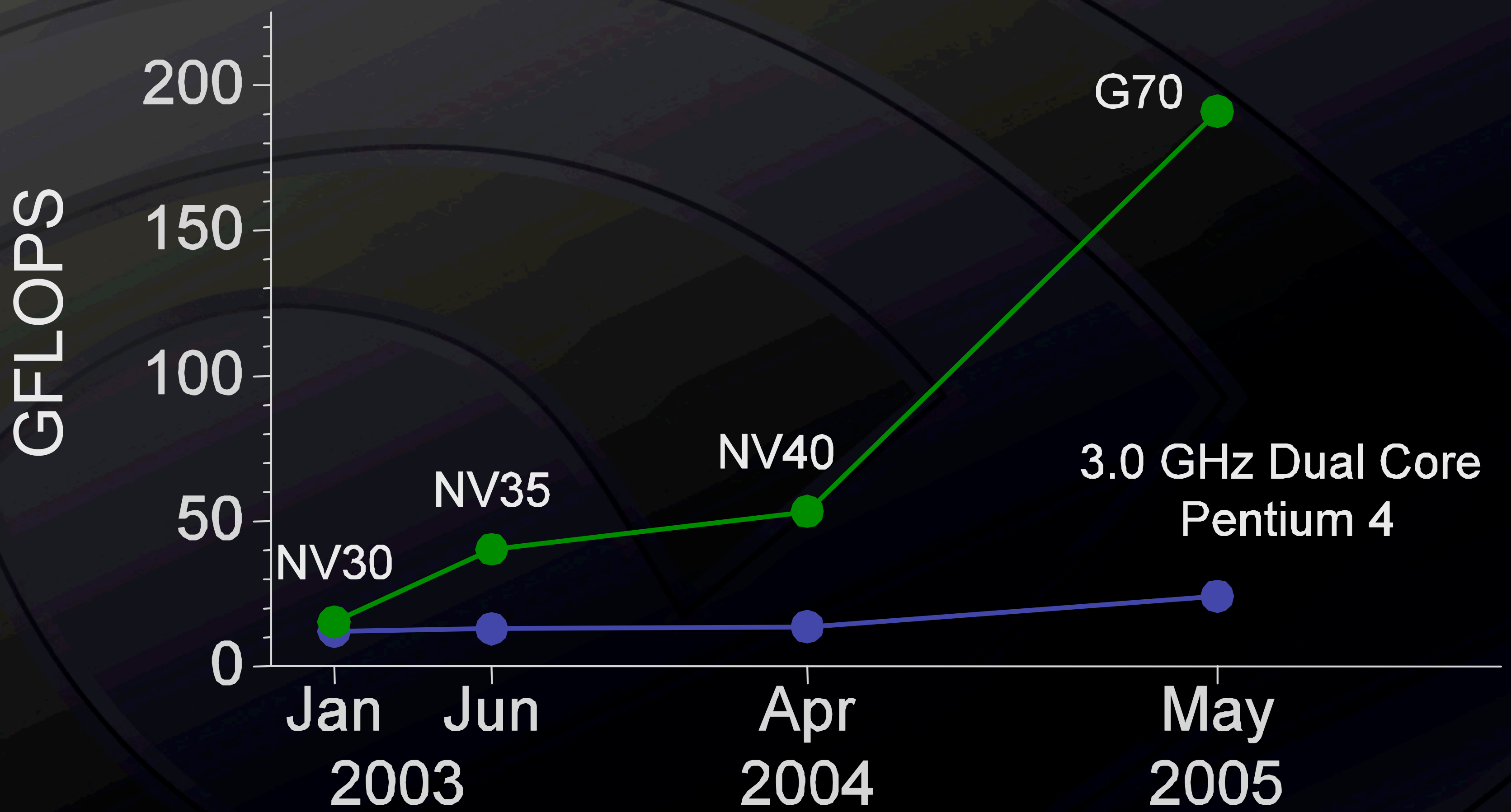
** Samples (multiple color values within a pixel, for smooth edges)

GPUs Continue to Accelerate above Moore's Law, but that's not all...

- As pixel/vertex/triangle growth slows and plateaus...
- Other performance factors increase
  - Number of color samples per pixel (Anti-aliasing)
  - Number of calculations per pixel/vertex
  - Flexibility of programming model
    - Looping, branching, multiple I/O access, multiple math ops/clock
    - High-level language programmability
  - Number of "General Purpose" programmable 32bit Gigaflops per pixel – demand grows without bounds

- GPUs become General Purpose parallel processors
  - What happens if you compare GPUs to microprocessors?

Sustained SP MAD GFLOPS



CPU / GPU Design Strategies / Tactics

- CPU Strategy: Make the workload (one compute thread) run as fast as possible
  - Tactics
    - Cacheing
    - Instruction/Data Prefetch
    - "hyperthreading"
    - Speculative Execution
    - → limited by "perimeter" – communication bandwidth
    - Multi-core will help... a little
- GPU Strategy: Make the workload (as many threads as possible) run as fast as possible
  - Tactics
    - Parallelism (1000s of threads)
    - Pipelining
    - → limited by "area" – compute capability

Application Matches for GPU:
Any Large-Scale, Parallel, Feed-forward, Math- and Data-intensive Problem

- Real-time Graphics (of course!)
- Image Processing and Analysis
  - Telescope, Surveillance, Sensor Data
  - Volume Data
- Correlation - Radio Telescope, SETI
- Monte Carlo Simulation - Neutron Transport
- Neural Networks
  - Speech Recognition
  - Handwriting Recognition
- Ray Tracing
- Physical Modeling and Simulation
- Video Processing

Example: Black-Scholes options pricing

- Widely-used model for pricing call/put options
- Implemented in ~15 lines of Cg, use combinations of input parameters as separate simulations (fragments)
- Performance:
  - Fast (~3GHz) P4, good C++: ~3.0 MBSOPS (1X)
  - Quadro FX 3000, Cg: ~2.8 MBSOPS (~.9X)
  - Quadro FX 4400, Cg: ~14.4 MBSOPS (4.8X)
  - Quadro FX 4400, Cg, 100 runs: ~176.0 MBSOPS (59X)

    (remove test/data transfer bandwidth overhead)
- How?
  - CPU: ~11GFLOPS, slow exp(), log(), sqrt(), fast mem access
  - GPU: ~65GFLOPS, fast exp(), log(), sqrt(), slow mem access
  - Black-Scholes has high ratio of math to memory access
  - GPU has Parallelism

So, What's the Problem?

- The Good News:
  - CPUs and GPUs are increasingly parallel
  - GPUs are already highly parallel
  - Workloads – Graphics and GP – are highly parallel
  - Moore's Law and the "capability curve" continue to be our friends
- The Not-so-Good News:
  - Parallel programming is hard
  - Language and Tool support for parallelism is poor
  - Computer Science Education is not focused on parallel programming

We Are Approaching a Crisis in Programming Skills (lack)

- Intel, AMD, IBM/Sony/Toshiba (Cell), Sun (Niagara) have all announced Multi- or Many-core roadmaps
- NVIDIA and other GPUs are already “Multi-core” ☺
  - Less of a crisis, due to GPU threaded programming model
- Analysts predict > 50% of processors shipped in next 5 years will be >1 core
- Who will program these devices?
- How will the value of multi-core and multi-threading be exploited?

Call to Action

- Research
  - Explore new ways of Parallel Programming
  - Explore new Threading models
  - Make parallelism easier to express/exploit
- Industry (processor vendors)
  - Make exploitation of Multi-core easier
  - Explore “transparent” application speedups
- Consequences of Failure to Act
  - Multi-core not valued by market

Questions?