

Trends in Multicore Architecture

Kevin Skadron

University of Virginia Dept. of Computer Science

LAVA Lab



Outline



Objective of this segment: explain why this tutorial is relevant to researchers in systems design

- Why multicore?
- Why GPUs?
- Why CUDA?

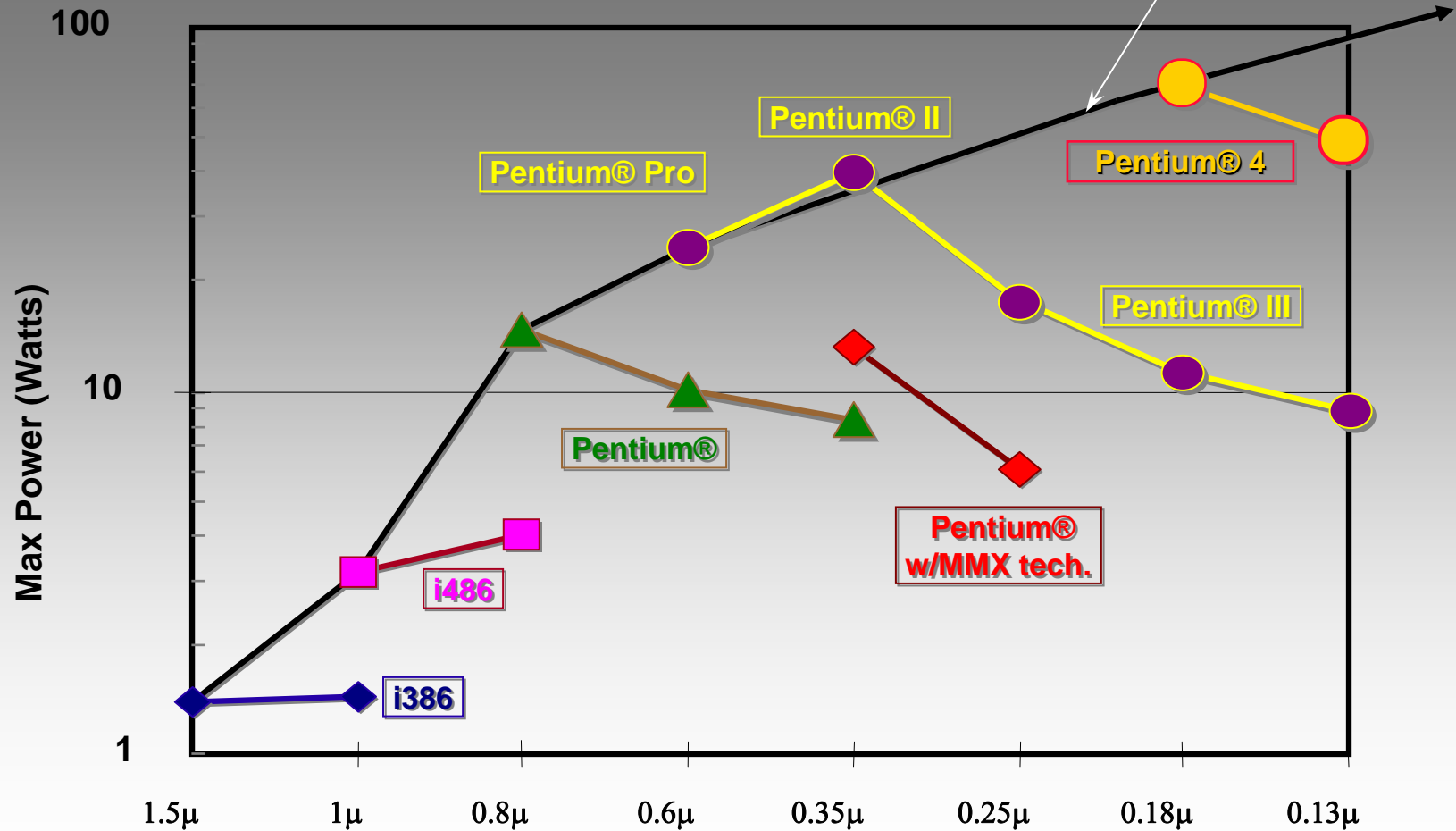
Why Multicore?



- **Why the dramatic paradigm shift? Combination of *both* “ILP wall” and “power wall”**
 - **Not *just* power**
 - **ILP wall: wider superscalar, more aggressive OO execution, have run out of steam**
 - **Power wall: only way to boost single-thread performance was to crank frequency**
 - **Aggressive circuits (expensive)**
 - **Very deep pipeline – 30+ stages? (expensive)**
 - **Power-saving techniques weren't able to compensate**
 - **This leaves only natural frequency growth due to technology scaling (~20-30% per generation)**
 - **Don't need expensive microarchitectures to obtain that scaling**

Actual Power

Core 2 Duo



Source: Intel

The Multi-core Revolution

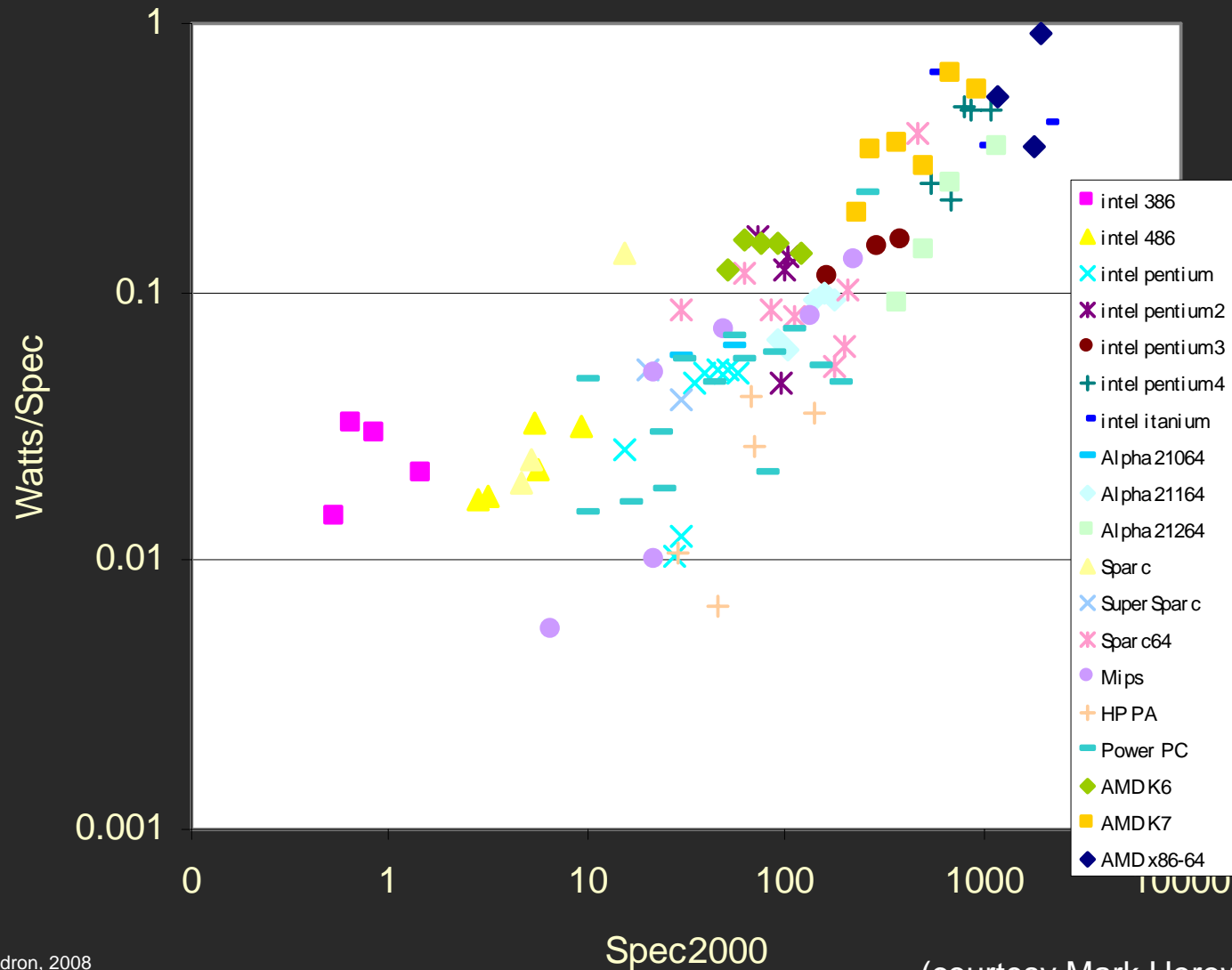


- **Can't make a single core faster**
- **Moore's Law \Rightarrow same core is 2X smaller per generation**
 - **Need to keep adding value to maintain average selling price**
 - **More and more cache doesn't cut it**
- **Use all those transistors to put multiple processors (cores) on a chip**
 - **2X cores per generation**
 - **Cores can potentially be optimized for power**
- **But harder to program, except for independent tasks**
 - **How many independent tasks are there to run at once?**

Single-core Watts/Spec



(through 2005)



Where Do GPUs Fit In?



- **Big-picture goal for processor designers: improve user benefit with Moore's Law**
 - Solve bigger problems
 - Improve user experience \Rightarrow induce them to buy new systems
- **Need scalable, programmable multicore**
 - **Scalable: doubling processing elements (PEs) ~doubles performance**
 - Multicore achieves this, *if* your program has scalable parallelism
 - **Programmable: easy to realize performance potential**
 - **GPUs can do this!**
 - GPUs provide a pool of cores with general-purpose instruction sets
 - Graphics has lots of parallelism *that scales to large # cores*
 - CUDA leverages this background
- **Need to maximize performance/mm²**
- **Need high volume to achieve commodity pricing**
 - GPUs of course leverage the 3D market

How do GPUs differ from CPUs?



Key: perf/mm²

- **Emphasize throughput, not per-thread latency**
- **Maximize number of PEs and utilization**
 - Amortize hardware in time—multithreading
 - Hide latency with computation, not caching
 - Spend area on PEs instead
 - Hide latencies with fast thread switch and many threads/PE
 - GPUs much more aggressive than today's CPUs: 24 active threads/PE on G80
- **Exploit SIMD efficiency**
 - Amortize hardware in space—share fetch/control among multiple PEs
 - 8 in the case of Tesla architecture
 - Note that SIMD \supset vector
 - NVIDIA's architecture is "scalar SIMD" (SIMT), AMD does both
- **High bandwidth to global memory**
 - Minimize amount of multithreading needed
 - G80 memory interface is 384-bit, R600 is 512-bit
- **Net result: 470 GFLOP/s and ~80 GB/s sustained in G80**
- **CPUs seem to be following similar trends**

How do GPUs differ from CPUs? (2)



- **Hardware thread creation and management**
 - New thread for each vertex/pixel
 - CPU: kernel or user-level software involvement
- **Virtualized cores**
 - Program is agnostic about physical number of cores
 - True for both 3D and general-purpose
 - CPU: number of threads generally $f(\# \text{ cores})$
- **Hardware barriers**
- **These characteristics simplify problem decomposition, scalability, and portability**
- **Nothing prevents non-graphics hardware from adopting these features**

How do GPUs differ from CPUs? (3)



- **Specialized graphics hardware**
(Here I only talk about what is exposed through CUDA)
 - **Texture path**
 - High-bandwidth gather, interpolation
 - **Constant memory**
 - Even higher-bandwidth access to small read-only data regions
 - **Transcendentals (reciprocal sqrt, trig, log2, etc.)**
 - **Different implementation of atomic memory operations**
 - GPU: handled in memory interface
 - CPU: generally handled with CPU involvement
 - **Local scratchpad in each core (a.k.a. per-block shared memory)**
 - **Memory system exploits spatial, not temporal locality**

How do GPUs differ from CPUs? (4)



- **Fundamental trends are actually very general**
 - **Exploit parallelism in time and space**
- **Other processor families are following similar paths (multithreading, SIMD, etc.)**
 - **Niagara**
 - **Larrabee**
 - **Network processors**
 - **Clearspeed**
 - **Cell BE**
 - **Many others....**
- **Alternative: heterogeneous**
 - **(Nomenclature note: asymmetric = single-ISA, heterogeneous = multi-ISA)**
 - **Cell BE**
 - **Fusion**

Why is CUDA Important?



- **Mass market platform**
 - Easy to buy and set up a system
- **Provides a solution for *manycore* parallelism**
 - Not limited to small core counts
 - Easy to learn abstractions for massive parallelism
- **Abstractions are not tied to a specific platform**
 - Doesn't depend on graphics pipeline; can be implemented on other platforms
 - Preliminary results suggest that CUDA programs run efficiently on multicore CPUs [Stratton'08]
- **Supports a wide range of application characteristics**
 - More general than streaming
 - Not limited to data parallelism

Why is CUDA Important? (2)



- **CUDA + GPUs facilitate multicore research *at scale***
 - 16, 8-way SIMD cores = 128 PEs
 - Simple programming model allows exploration of new algorithms, hardware bottlenecks, and parallel programming features
 - The whole community can learn from this
- **CUDA + GPUs provide a *real* platform...today**
 - Results are not theoretical
 - Increases interest from potential users, e.g. computational scientists
 - Boosts opportunities for interdisciplinary collaboration
- **Underlying ISA can be targeted with new languages**
 - Great opportunity for research in parallel languages
- **CUDA is teachable**
 - Undergrads can start writing real programs within a couple of weeks

Thank you



● Questions?