Algorithmen für die Echtzeitgrafik

Algorithmen für die Echtzeitgrafik

Daniel Scherzer

scherzer@cg.tuwien.ac.at

LBI Virtual Archeology

Why Parallel Programming?

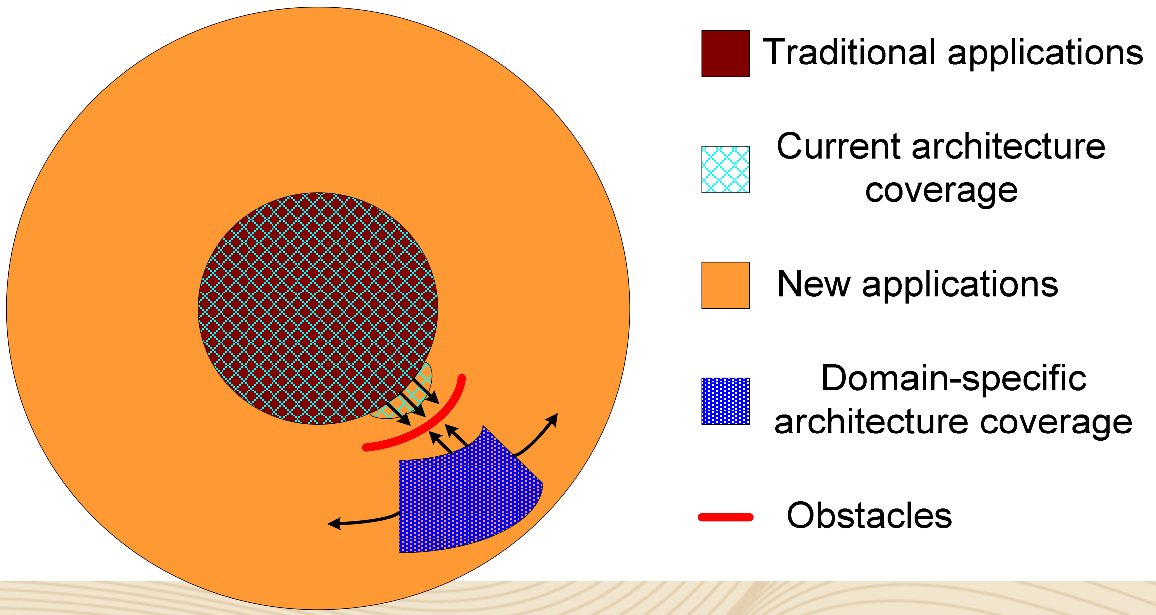
Applications

Future Apps Reflect a Concurrent World

- “Supercomputing applications”
  - New applications in “future” mass computing market
    - Molecular dynamics simulation
    - Video and audio coding and manipulation
    - 3D imaging and visualization
    - Consumer game physics
    - Virtual reality products
    - …
  - “Super-apps” represent and model physical, concurrent world (huge amount of data, streaming, …)
- Various granularities of parallelism exist, but…
  - programming model must not hinder parallel implementation
  - data delivery needs careful management

Stretching Traditional Architectures

- Traditional Apps:
  - Sequential / hard to parallelize
  - Covered by CPUs
- New Apps:
  - Huge amount of data
  - Partly parallelizable



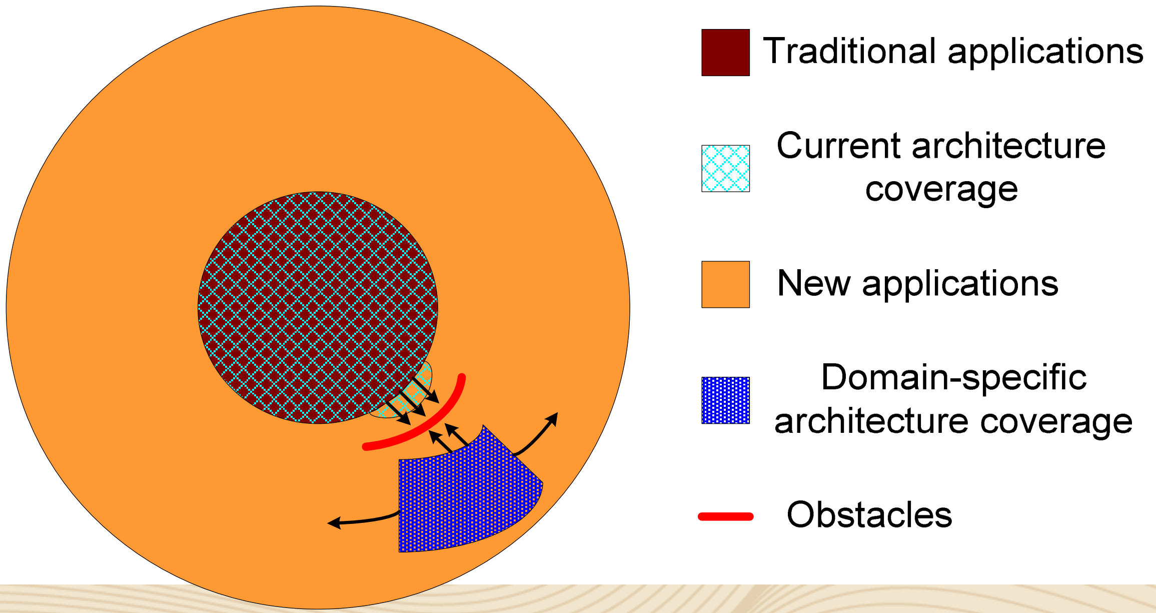
Traditional applications
Current architecture coverage
New applications
Domain-specific architecture coverage
Obstacles

Stretching Traditional Architectures

- Tradit. parallel architectures cover some super-apps
  - DSP, GPU, network apps, scientific
  - But with specifically designed hardware for problem
    - Extension hard or impossible

→ Grow mainstream architectures “out” (more cores..)

→ Or domain-specific architectures “in” (CUDA, OpenCL)


Traditional applications
Current architecture coverage
New applications
Domain-specific architecture coverage
Obstacles

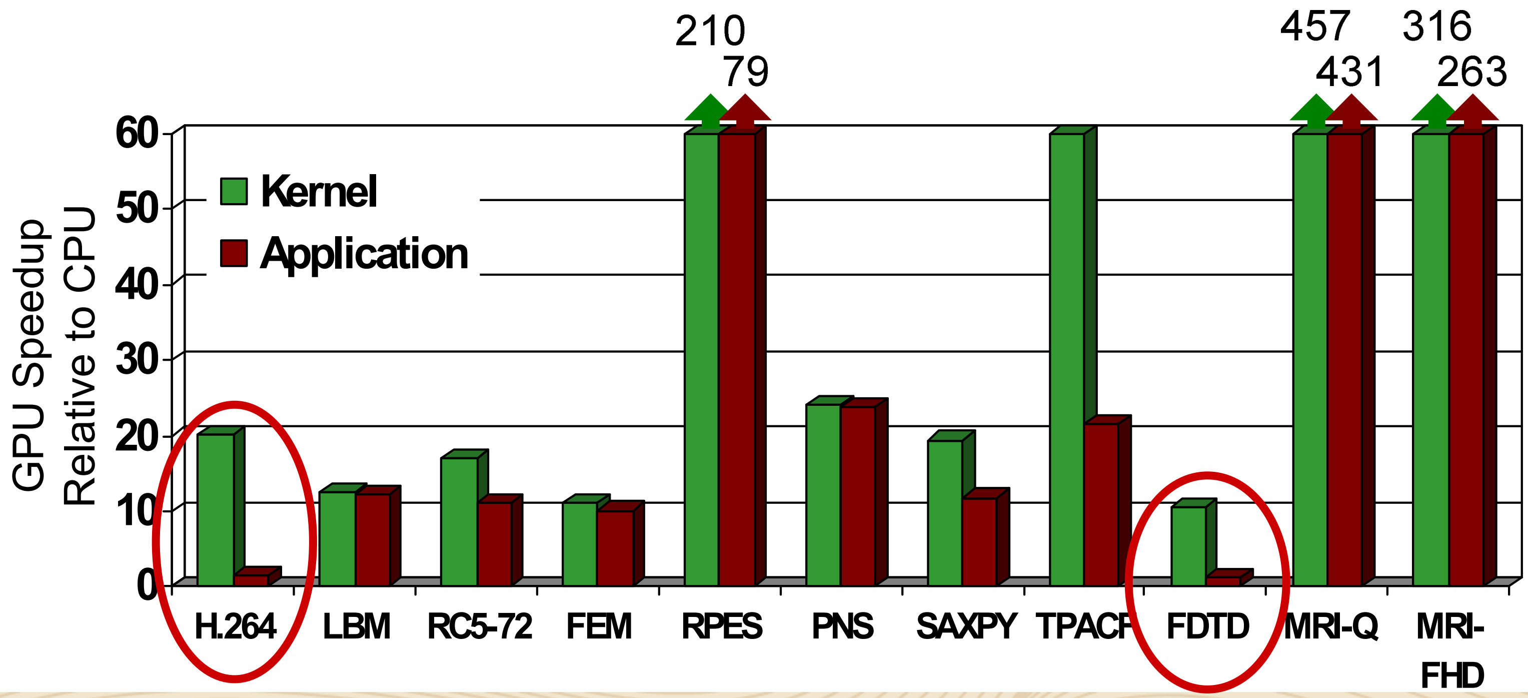
Special-purpose processors always choke off real algorithmic creativity - Jim Blinn

Example Applications (CUDA)

Application	Description	Source	Kernel	% time
H.264	SPEC '06 version, change in guess vector	34,811	194	35%
LBM	SPEC '06 version,fluid simulation; change to single precision and print fewer reports	1,481	285	>99%
RC5-72	Code cracking; Distributed.net RC5-72 challenge client code	1,979	218	>99%
FEM	Finite element modeling, simulation of 3D graded materials	1,874	146	99%
RPES	Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion	1,104	281	99%
PNS	Petri Net simulation of a distributed system	322	160	>99%
SAXPY	Single-precision implementation of saxpy, used in Linpack's Gaussian elim. Routine	952	31	>99%
TRACF	Two Point Angular Correlation Function	536	98	96%
FDTD	Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation	1,365	93	16%
MRI-Q	Computing a matrix Q, a scanner's configuration in MRI reconstruction	490	33	>99%

Speedup of Applications

- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in kernel typical, as long as kernel can occupy enough parallel threads
- 25× to 400× speedup if function's data requirements and control flow suit the GPU

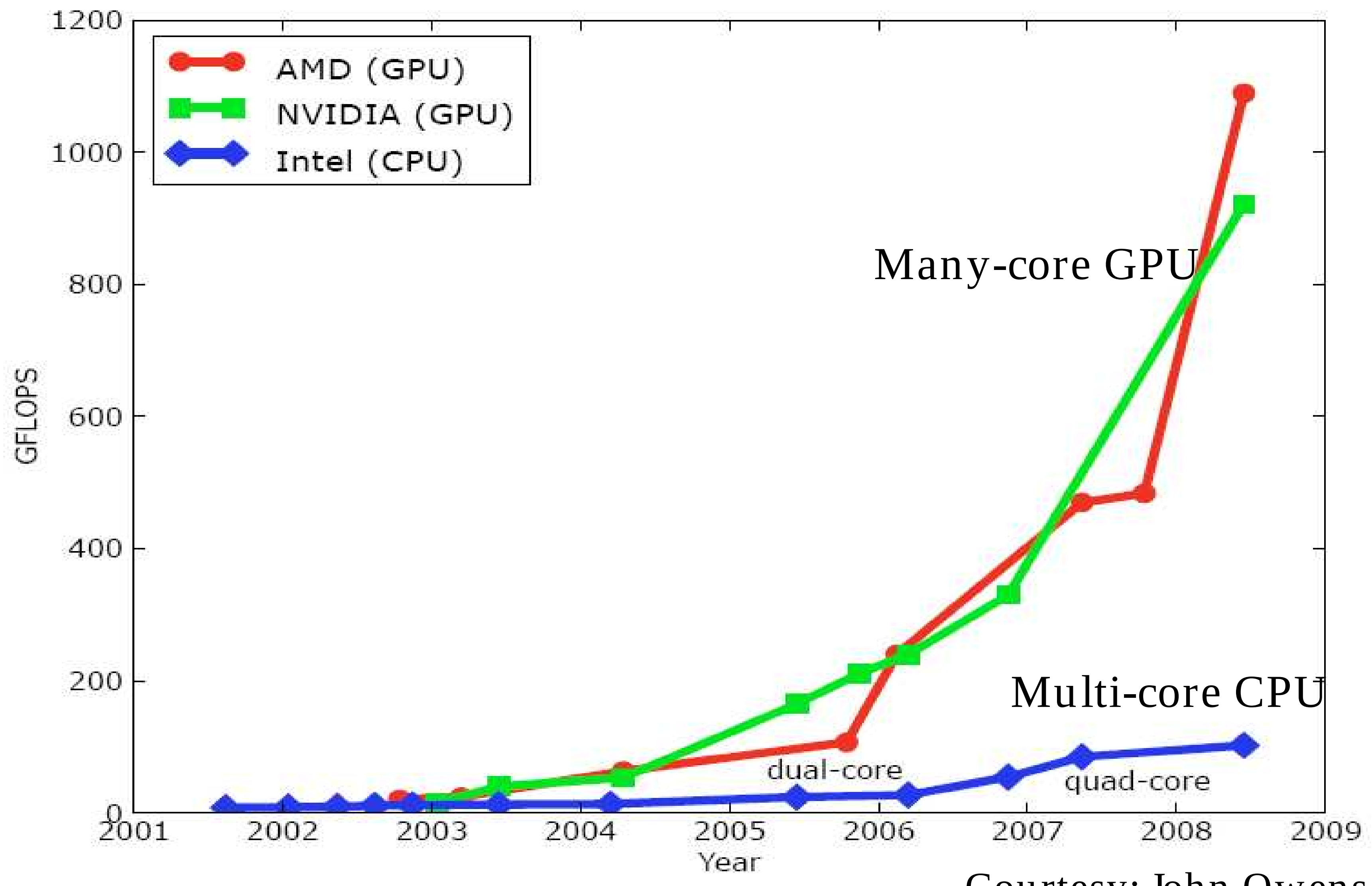


Why Parallel Programming?

Best Bang for the Buck –GPUs and Money

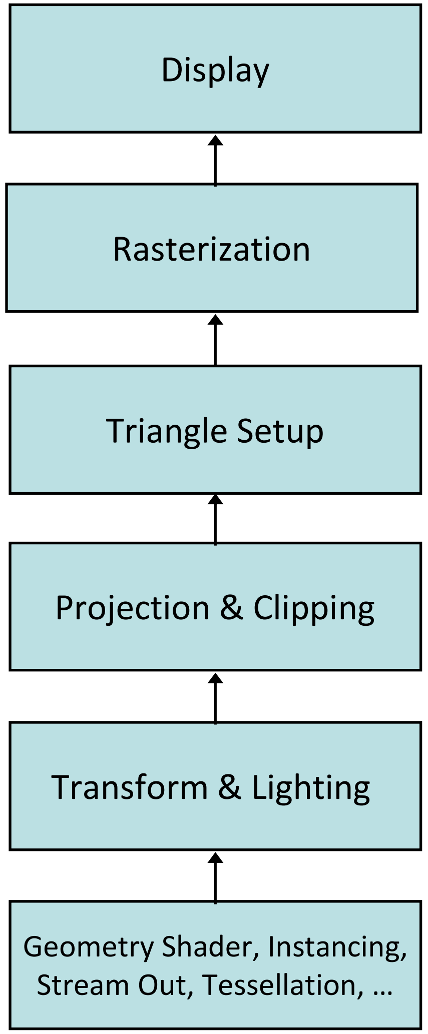
The display is the computer.
– Jen-Hsun Huang, CEO of NVIDIA

CPU vs. GPU



Courtesy: John Owens

GPUs: Upstream Over Time



The dark ages (early-mid 1990's), when there were only frame buffers for normal PC's.

Some accelerators were no more than a simple chip that sped up linear interpolation along a single span, so increasing fill rate.

Once even high-end systems supported just triangle setup and fill. CPU sent triangle with color and depth per vertex and it's rendered.

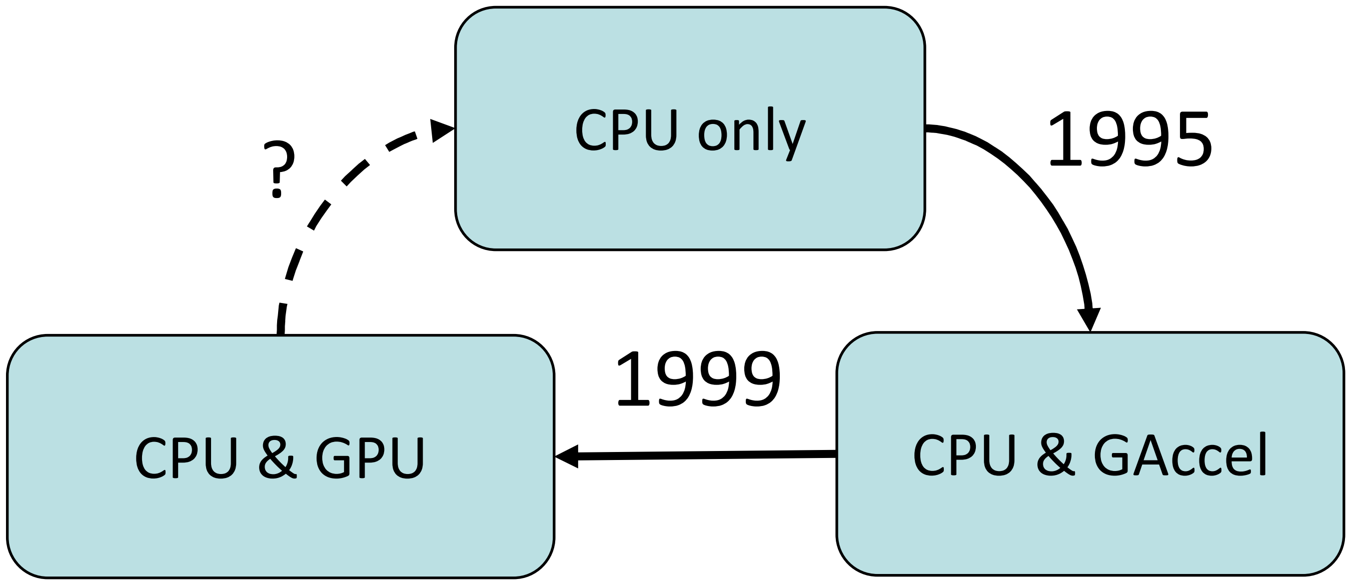
This is where pipelines start for PC commodity graphics, 1995-1998. Seminal event is 3dfx's Voodoo in October 1996.

This part of the pipeline reaches the consumer level with the introduction of the NVIDIA GeForce256, Fall 1999.

More and more moves to the GPU – what is the best division of labor? Should it even be a pipeline, or something more general?

Wheel of Reincarnation

Coined by Myer and Sutherland, 1968.



Will the wheel turn again?

Spending Transistors

- CPU
  - Control logic (ILP)
  - Memory
- GPU
  - (used to) spend on algorithm logic

Spending Transistors

- CPU and GPU heading towards each other
- CPU
  - SSE through SSE5
  - 128 bit registers
  - 256 bits data path with AVX
- GPU
  - Unified shaders
  - Large pools of registers
  - Less fixed-function stages
  - Multiple paths out of GPU

Modern Processor Trends

- Moore's Law: ~1.6x transistors every year (10x every 5 years)
  - DRAM capacity (per year)
    - 1.6x from 1980-1992
    - 1.4x 1996-2002
- DRAM bandwidth (per year)
  - 1.25x = 25%, (10x every 10 years)
- DRAM latency (per year)
  - 1.05x = **5%** (10x every 48 years).
- Bandwidth improves by at least the square of the improvement in latency [Patterson2004].

Memory & Latency

- “Cache is King”
- Missing the L2 cache and going to main memory is death, *10-50 slower.
  Why secondary rays usually stink.
- CPUs focus on very fast caches,
  GPUs (used to) try to hide latency via many threads.

Opportunity: Latency Hiding

```
for i = 1 to N: a[i] = b[i] * c + d[i]
```

sizeof(a && b && d) > cache size

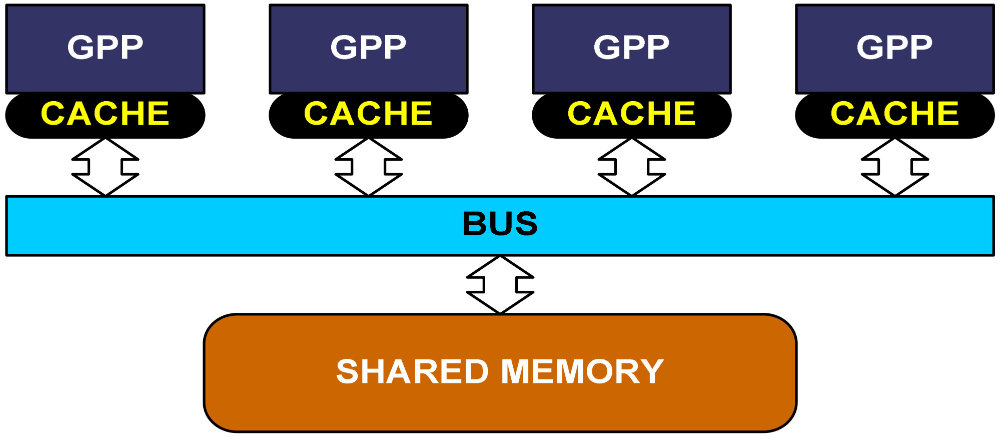
But sizeof(a) + sizeof(b || d) < cache size

Instead, to hide memory access:

```
for i = 1 to N: t[i] = b[i]
for i = 1 to N: t[i] *= c
for i = 1 to N: t[i] += d[i]
for i = 1 to N: a[i] = t[i]
```

Speedup of *10-50 possible

Memory Wall: Bandwidth



- Multi-core
  - Private caches allow some cores to continue when others suffer from memory latency
  - Ability to tolerate memory latency comes with increased memory bandwidth (traffic to caches)
- Coherence/consistency maintains illusion of monolithic memory

The Three Walls

- Instruction Level Parallelism (ILP) mined out
  - Branch prediction
  - Out of order processing
  - Control improvements
- Memory (access latency)
  - Load and store is slow
- Power (reason for multi-core)
  - GHz peaked in 2005 at around 3.8 GHz.
  - Diminishing returns
    - Increasing power does not linearly increase processing speed. 1.6x speed costs ~2-2.5x power and ~2-3x die area.

The Future: Parallelism

- Design must change!
  - Intel: in 2006 gave plan of 80 cores by 2011
  - Berkeley: could have thousand cores on a chip
- Migration towards multi-processing
  - Provide other threads of execution while waiting for memory
  - Big caches
  - Increase memory bandwidth to compensate for long latency
  - **Do not solve problem!**

The Future: Parallelism

- Tradeoff: large, fast core vs. many slower cores.
  - All tasks need to run reasonably, serial and parallel
  - Implies a hybrid: some fast cores, many small ones
  - The “HPU”: what is our goal? Solve for “H”
    (but Cell died!?)
  - Intel Turbo Boost [Knight Rider 1982]

Tearing Down the Memory Wall

- Traditional software model
  - Arbitrary data access
  - Flat monolithic memory
- Identifying private data and localize access
  - Eliminate unnecessary access and update to main memory
  - High compute to memory access ratio
  - Key to programming massively parallel processors