# GPUs – Cheap Supercomputing

Graham Pullan (Engineering)

Cambridge Many-Core Workshop

28 October 2008

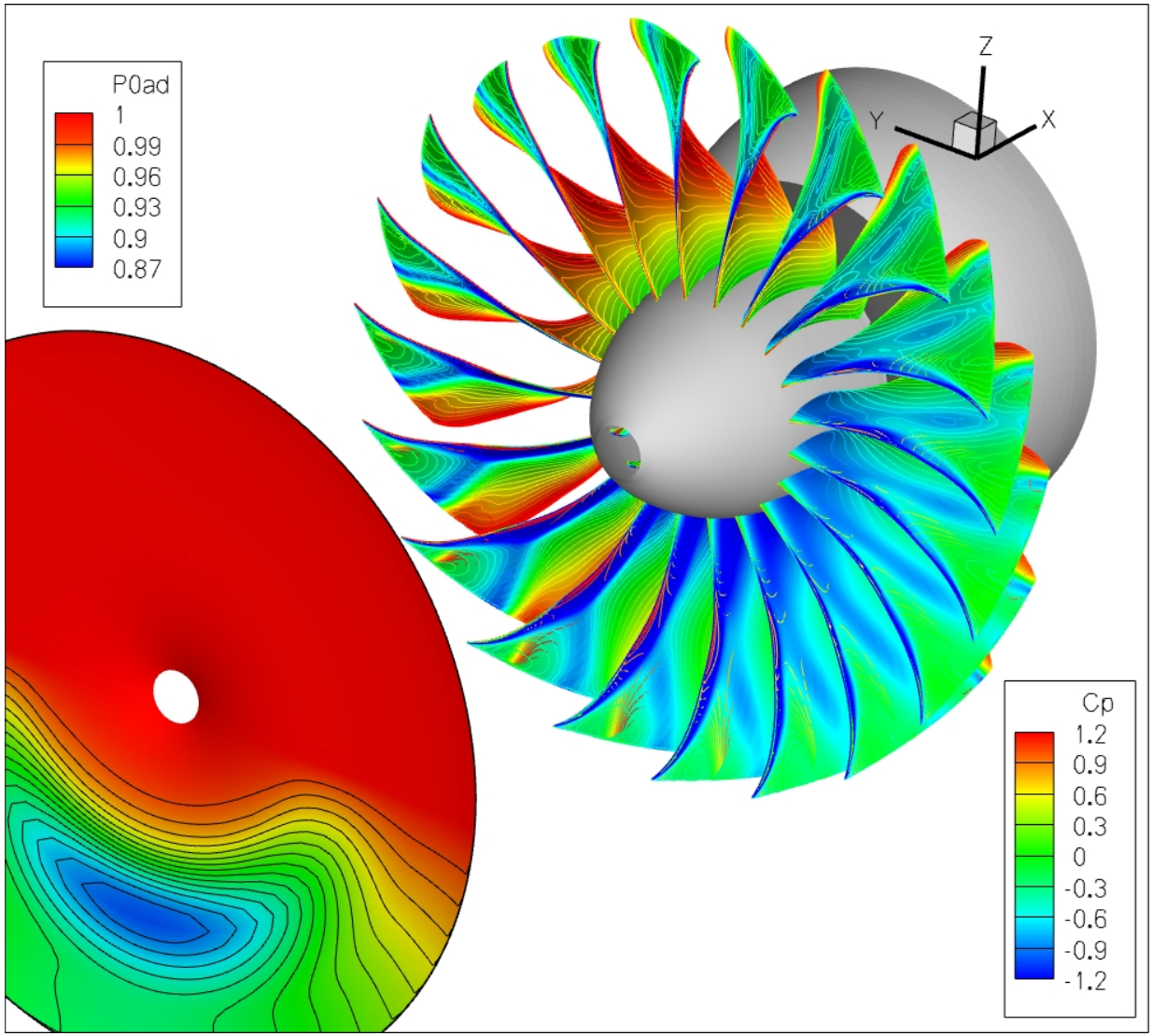
# Coming up...

- My background
- CPUs and GPUs
- GPU models: old and new
- An example
- Alternative devices
- Conclusions

# Part 1: My background

# Turbomachinery

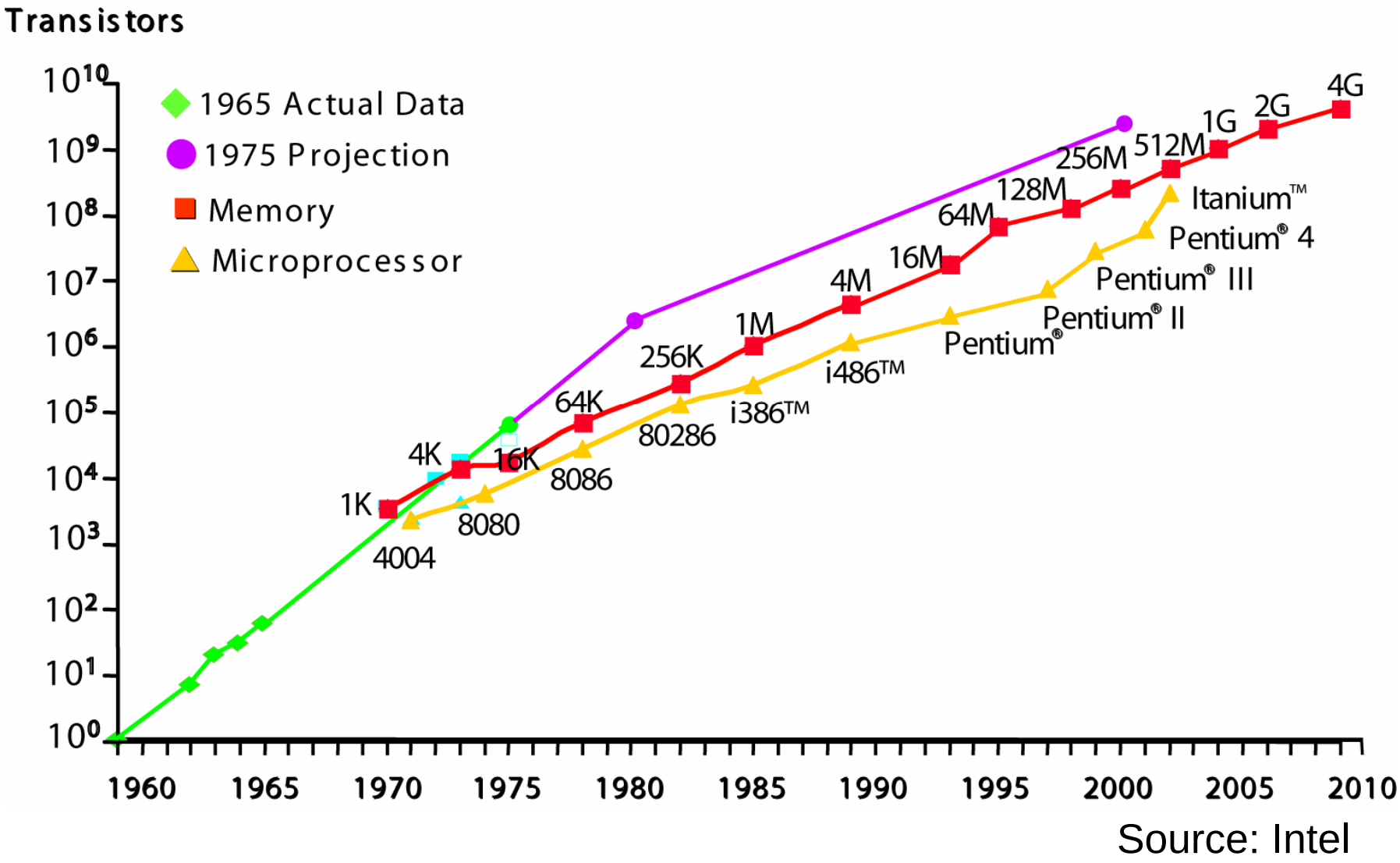
# Engine calculation




Courtesy Vicente Jerez Fidalgo, Whittle Lab

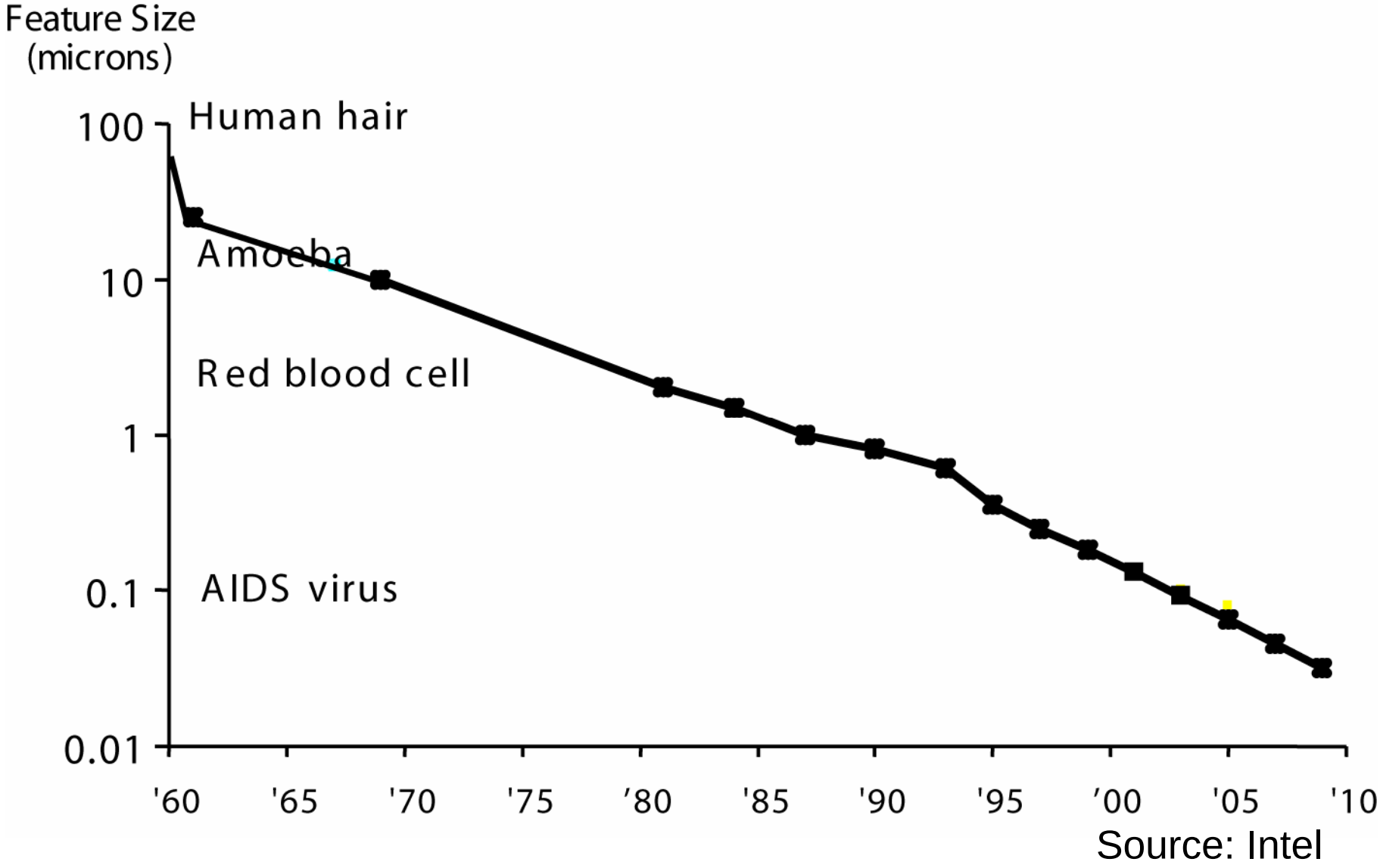
# Part 2: CPUs and GPUs

# Was Moore right?



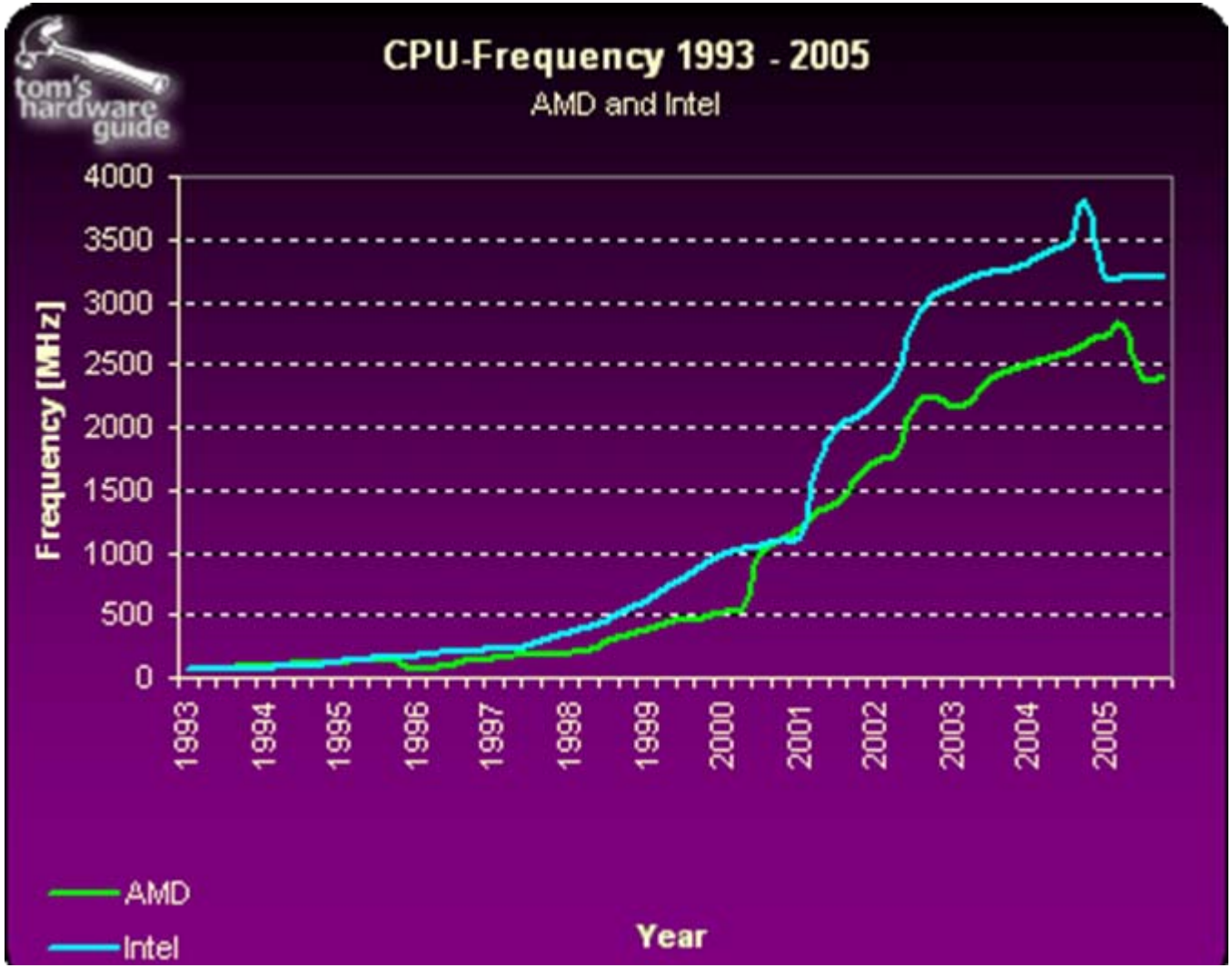

Source: Intel

# Feature size




Source: Intel

# Clock speed




Source: Tom's Hardware

# What to do with all these transistors?

# Parallel computing

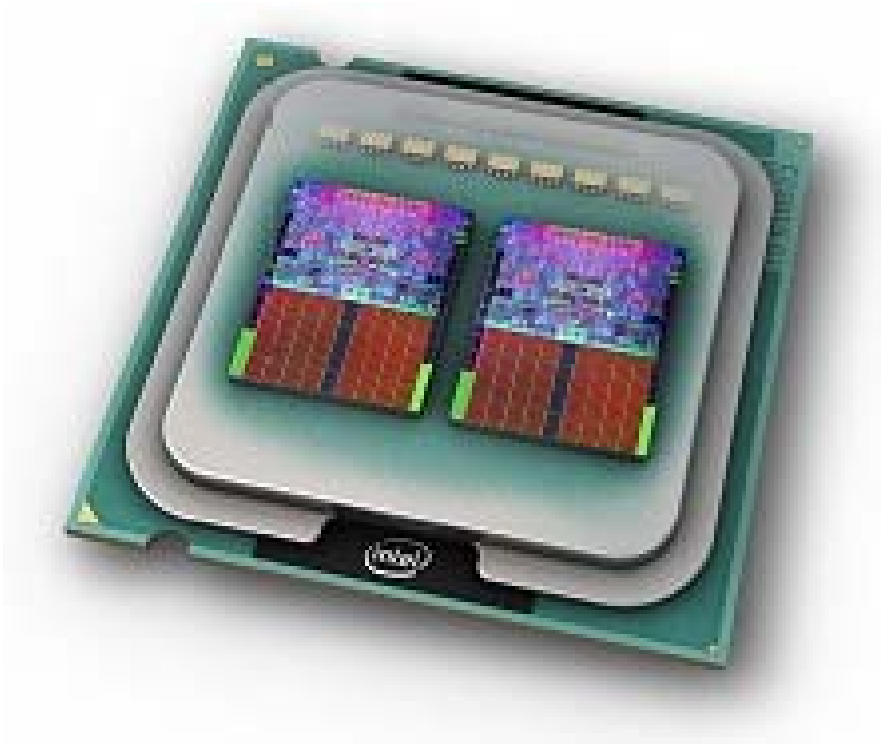
Multi-core chips are either:

– Instruction parallel
(Multiple Instruction, Multiple Data) – MIMD

or

– Data parallel
(Single Instruction, Multiple Data) – SIMD

# Today's commodity MIMD chips: CPUs



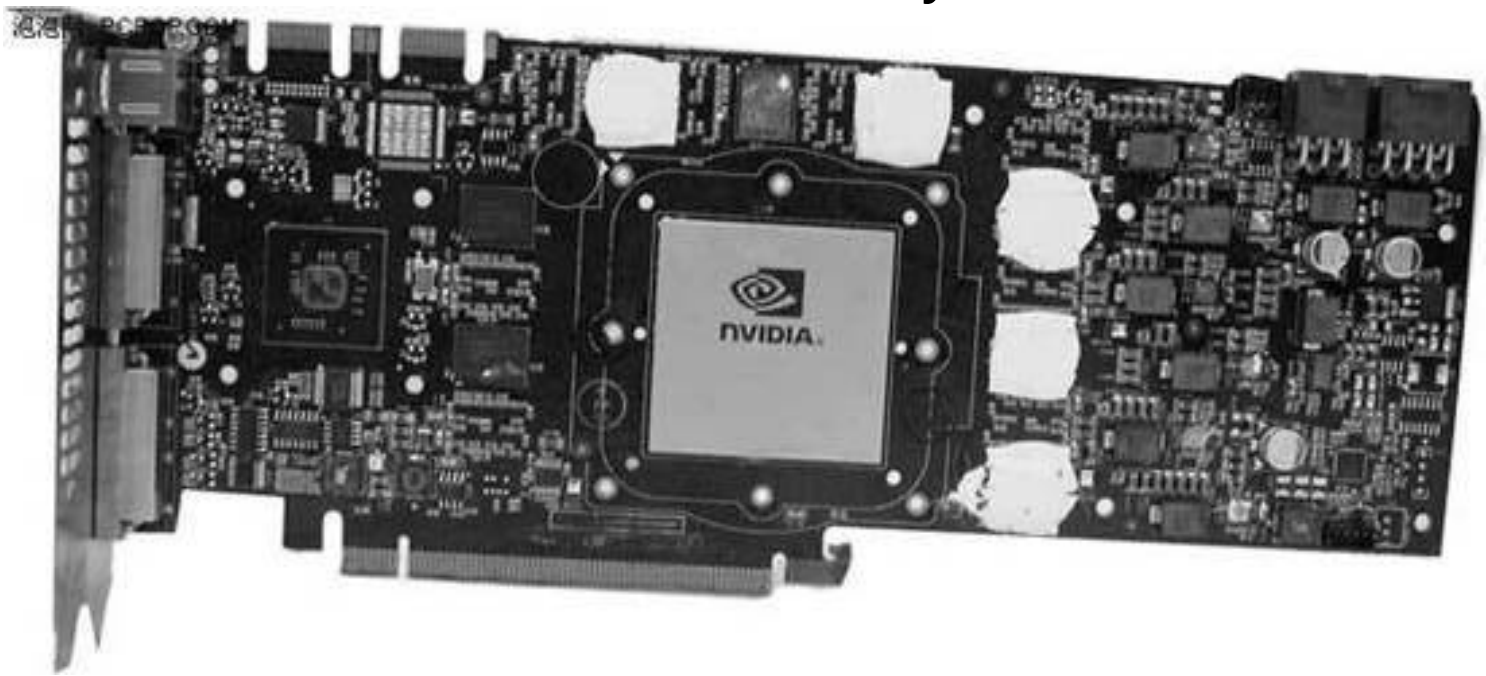
Intel Core 2 Quad

- 4 cores
- 3. GHz
- 45nm features
- 820 million transistors
- 12 MB on chip memory

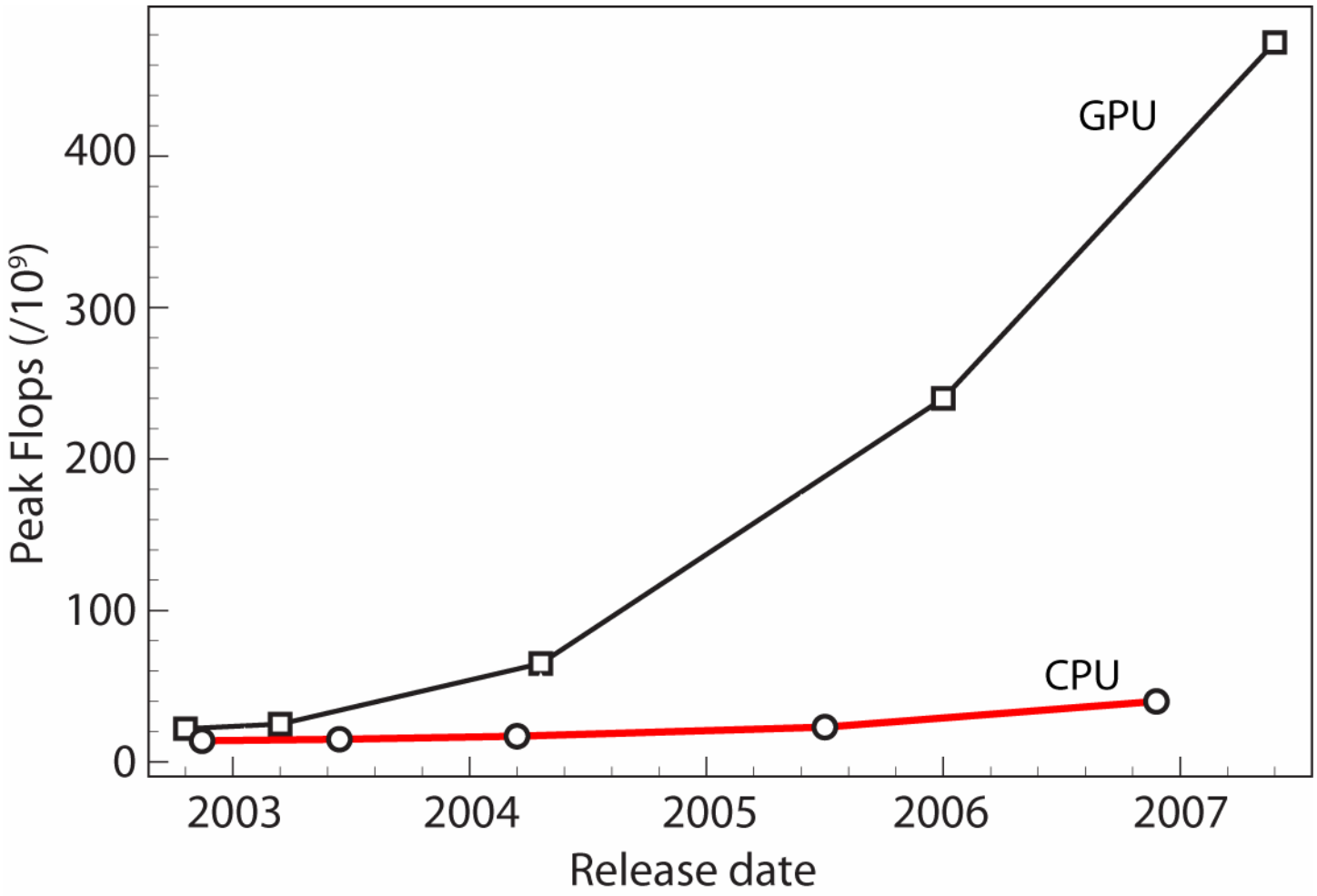
# Today's commodity SIMD chips: GPUs

NVIDIA 8800 GTX

- 240 cores 1.3 GHz
- 65nm features 1400 million transistors
- 1GB on board memory

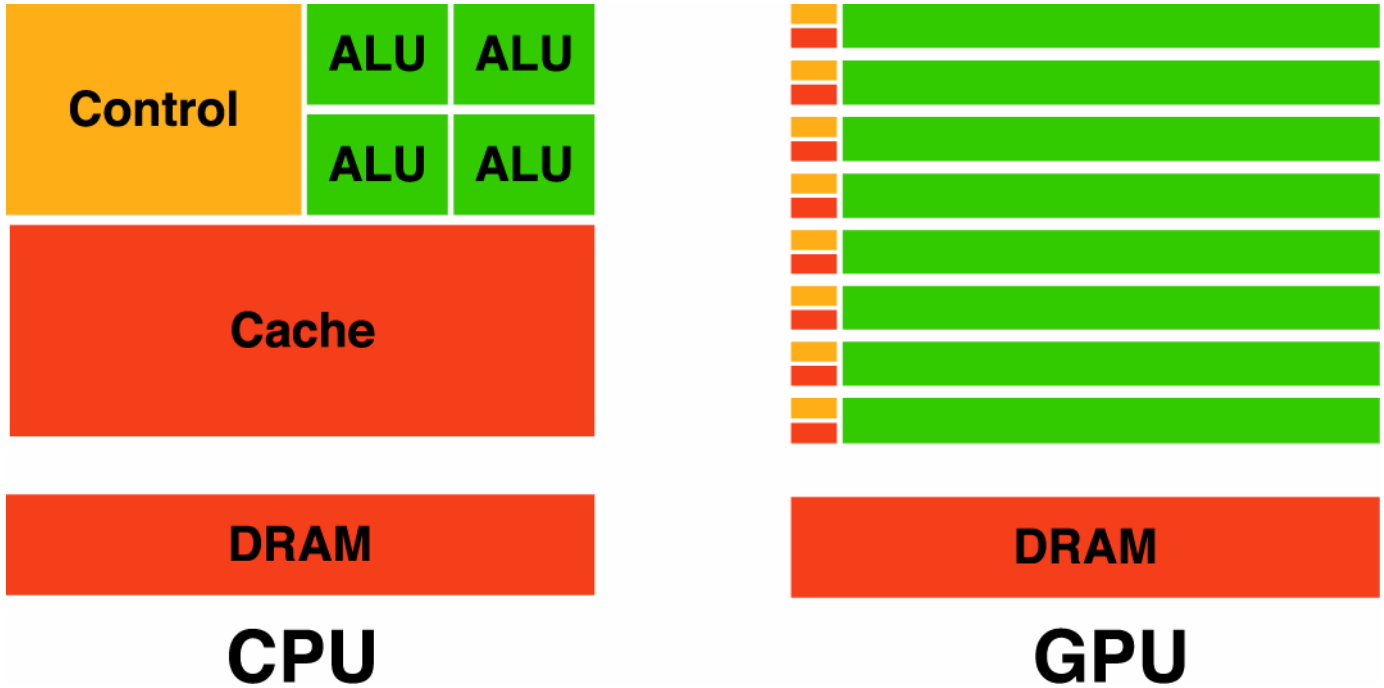


# CPUs vs GPUs


GPU
CPU
400
300
200
100
0
Peak Flops (/10⁹)
2003
2004
2005
2006
2007
Release date

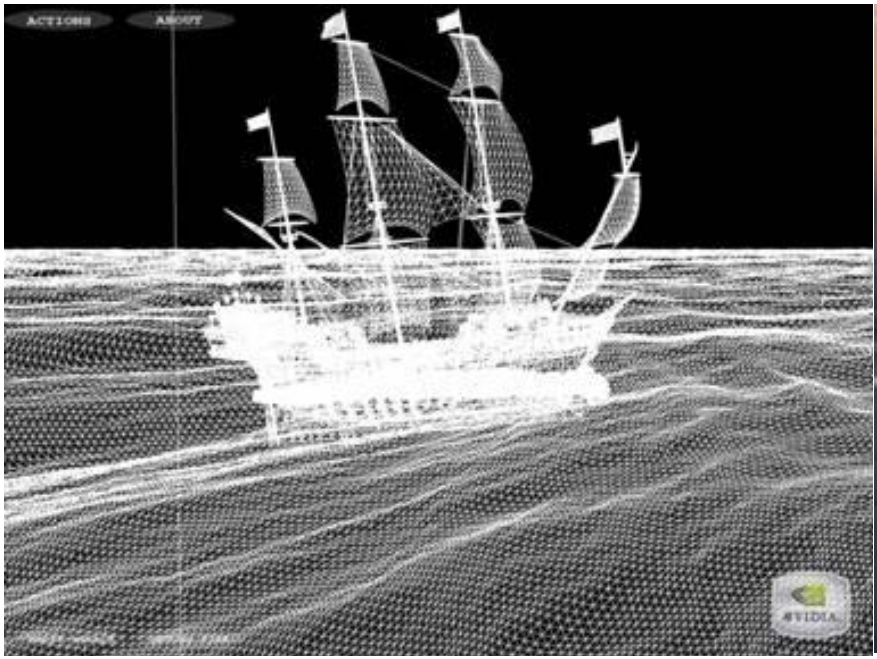
# CPUs vs GPUs

Transistor usage:




Source: NVIDIA

# Graphics pipeline



# GPUs and scientific computing

GPUs are designed to apply the
same ***shading function***
to many ***pixels*** simultaneously

# GPUs and scientific computing

GPUs are designed to apply the
same ***function***
to many ***data*** simultaneously

This is what most scientific computing needs!

# Part 3: Programming methods

# 3 Generations of GPGPU (Owens, 2008)

# 3 Generations of GPGPU (Owens, 2008)

- Making it work at all:
  - Primitive functionality and tools (graphics)
  - Comparisons with CPU not rigorous

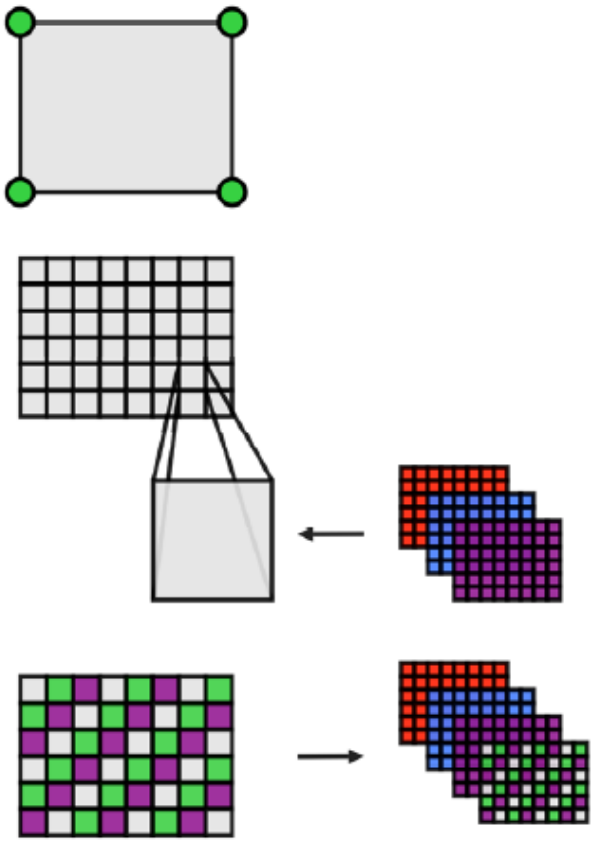
# 3 Generations of GPGPU (Owens, 2008)

- Making it work at all:
  - Primitive functionality and tools (graphics)
  - Comparisons with CPU not rigorous
- Making it work better:
  - Easier to use (higher level)
  - Understanding of how best to do it

# 3 Generations of GPGPU (Owens, 2008)

- Making it work at all:
  - Primitive functionality and tools (graphics)
  - Comparisons with CPU not rigorous
- Making it work better:
  - Easier to use (higher level)
  - Understanding of how best to do it
- Doing it right:
  - Stable, portable, modular building blocks

# GPU – Programming for graphics



Application specifies geometry – GPU rasterizes

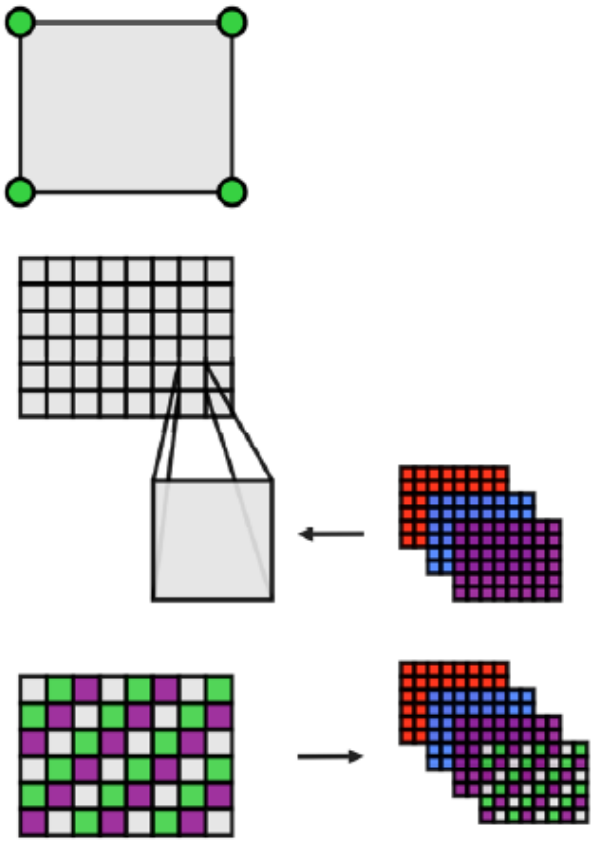
Each fragment is shaded (SIMD)

Shading can use values from memory (textures)

Image can be stored for re-use

Courtesy, John Owens, UC Davis

# GPGPU programming (“old-school”)



Draw a quad

Run a SIMD program over each fragment

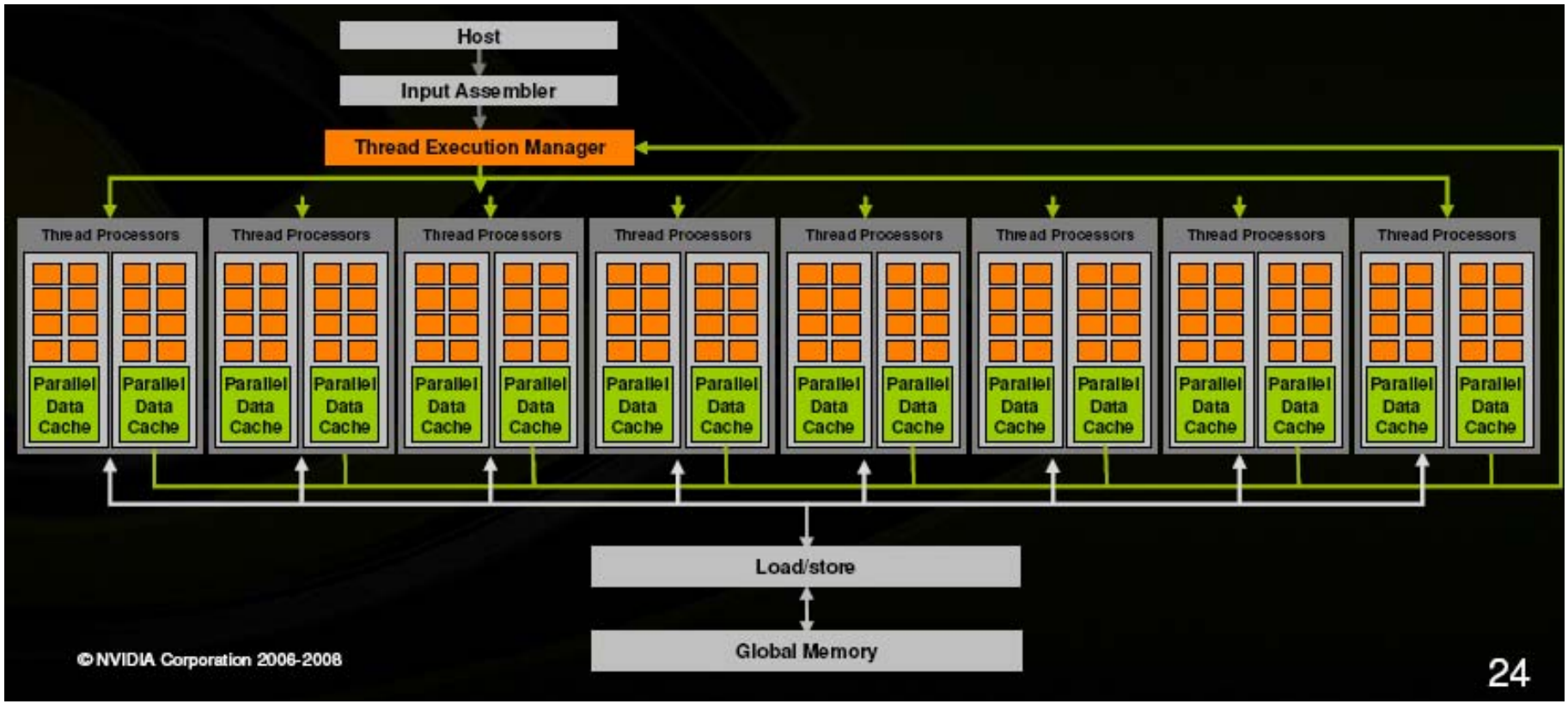
Gather is permitted from texture memory

Resulting buffer can be stored for re-use

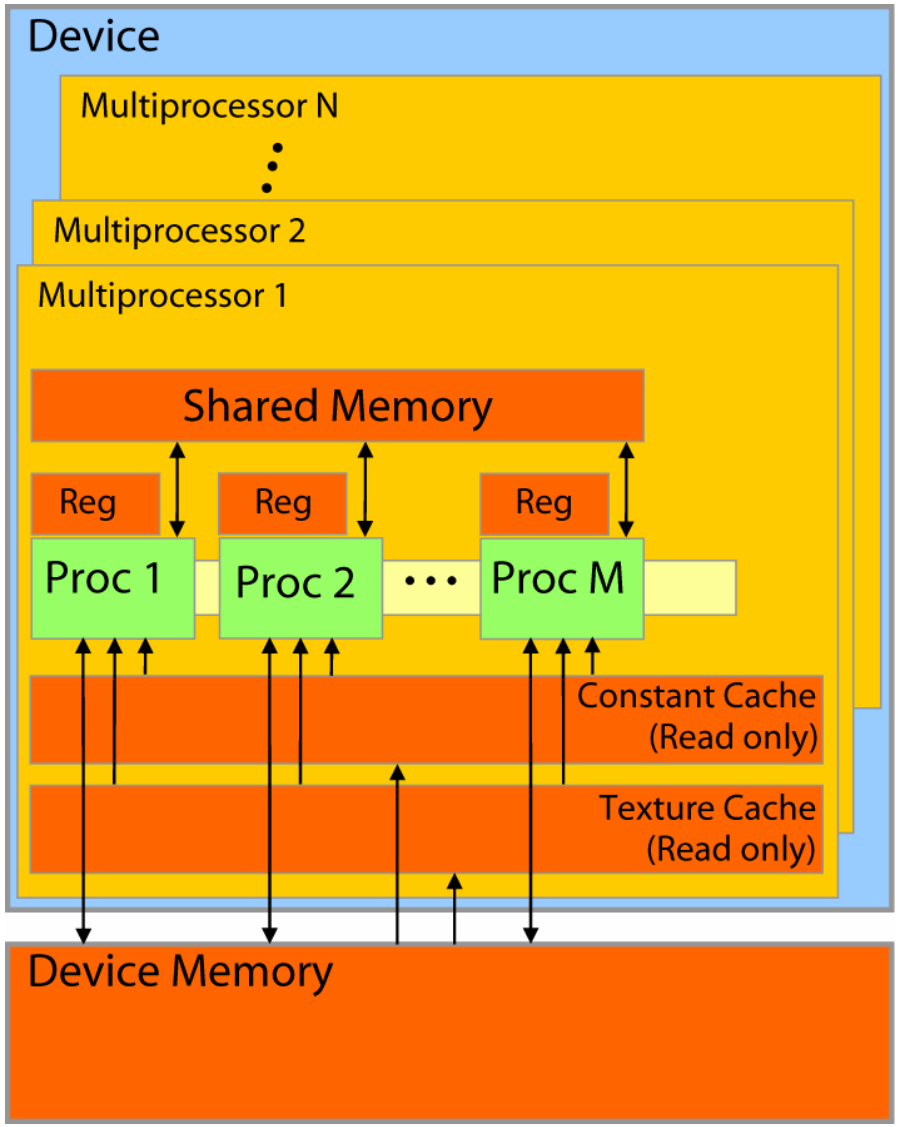
Courtesy, John Owens, UC Davis

# NVIDIA G80 hardware implementation

- Now view GPU as massively parallel co-processor
- Set of (16) SIMD MultiProcessors (8 cores)



# NVIDIA G80 hardware implementation




Divide 128 cores into

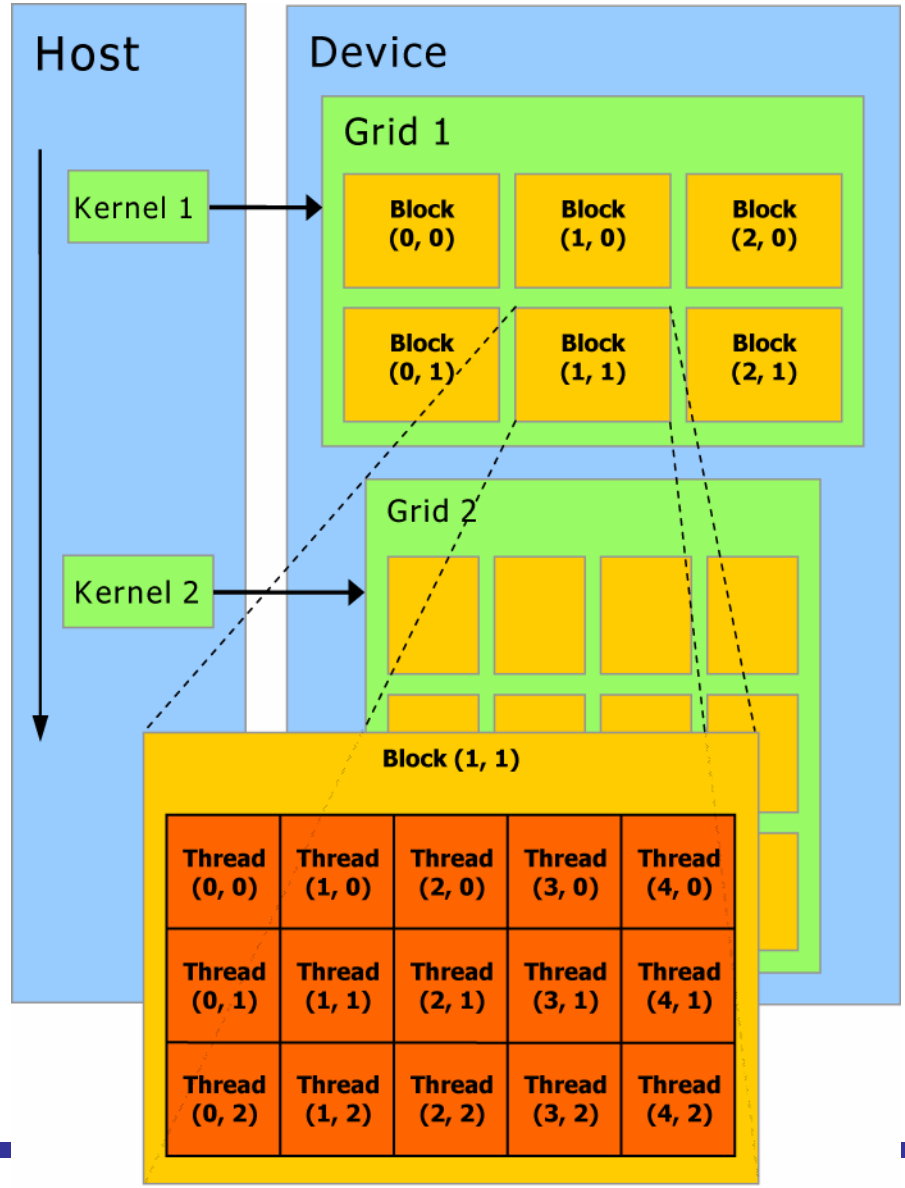
16 Multiprocessors (MPs)

- Each MP has:
  - –Registers
  - –Shared memory
  - –Read only constant cache
  - –Read only texture cache

# NVIDIA's CUDA programming model

- Hardware supports many thousands of active ***threads***
- Threads are lightweight:
  - Little creation overhead
  - "instant" switching
  - Efficiency achieved through 1000's of threads
- Threads are organised into ***blocks*** (1D, 2D, 3D)
- Blocks are further organised into a ***grid***

# Kernels, grids, blocks and threads



# Kernels, grids, blocks and threads

- Organisation of threads and blocks is key abstraction

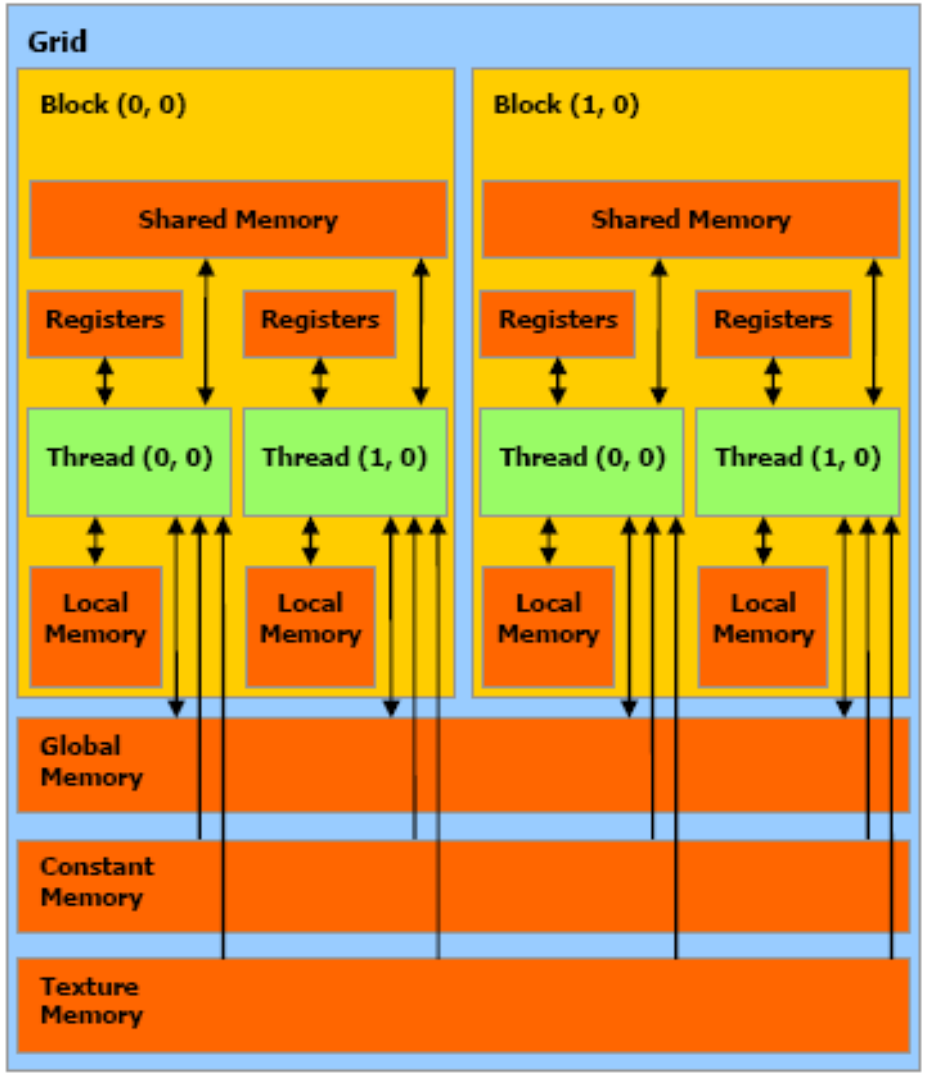
# Kernels, grids, blocks and threads

- Organisation of threads and blocks is key abstraction
- Software:
  - Threads from one block may cooperate:
    - Using data in shared memory
    - Through synchronising

# Kernels, grids, blocks and threads

- Organisation of threads and blocks is key abstraction
- Software:
  - Threads from one block may cooperate:
    - Using data in shared memory
    - Through synchronising
- Hardware:
  - A block runs on one MP
  - Hardware free to schedule any block on any MP
  - More than one block can reside on one MP

# Kernels, grids, blocks and threads



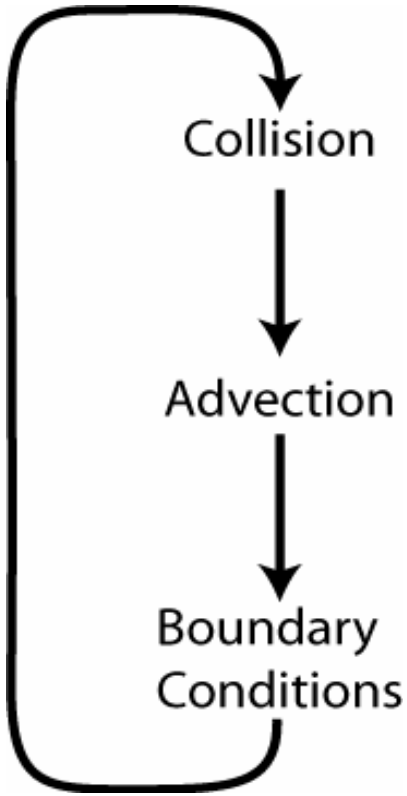
# CUDA implementation

- CUDA implemented as extensions to C

- CUDA programs:
  - explicitly manage host and device memory:
    - allocation
    - transfers
  - set thread blocks and grid
  - launch kernels
  - are compiled with the CUDA **nvcc** compiler

# Part 4: An example – Lattice Bolzmann CFD

# Solution procedure




Can be viewed as a particle method:

Collide – particles interact

Stream – particles move from site to site

BCs – inlet, exit, solid wall, etc.

# CPU code: main.c

```
/* Memory allocation */
f0 = (float *)malloc(ni*nj*sizeof(float));
  ...

/* Main loop */
Stream (...args...);
Apply_BCs (...args...);
Collide (...args...);
```

# GPU code: main.cu

```
/* allocate memory on host */
f0 = (float *)malloc(ni*nj*sizeof(float));

/* allocate memory on device */
cudaMallocPitch((void **)&f0_data, &pitch,
                sizeof(float)*ni, nj);

cudaMallocArray(&f0_array, &desc, ni, nj);

/* Main loop */
Stream (...args...);
Apply_BCs (...args...);
Collide (...args...);
```

# CPU code – collide.c

```
for (j=0; j<nj; j++) {
  for (i=0; i<ni; i++) {
      i2d = I2D(ni,i,j);
/* Flow properties */
      density = ...function of f's ...
      vel_x = ...   "
      vel_y = ...   "
/* Equilibrium f's */
      f0eq = ... function of density, vel_x, vel_y ...
      f1eq = ...     "
/* Collisions */
      f0[i2d] = rtau1 * f0[i2d] + rtau * f0eq;
      f1[i2d] = rtau1 * f1[i2d] + rtau * f1eq;
      ...
  }
}
```

# GPU code – collide.cu – kernel wrapper

```
void collide( ... args ...)
{
/* Set thread blocks and grid */
    dim3 grid = dim3(ni/TILE_I, nj/TILE_J);
    dim3 block = dim3(TILE_I, TILE_J);

/* Launch kernel */
    collide_kernel<<<grid, block>>>(... args ...);

}
```

# GPU code – collide.cu - kernel

```
/* Evaluate indices */
i = blockIdx.x*TILE_I + threadIdx.x;
j = blockIdx.y*TILE_J + threadIdx.y;
i2d = i + j*pitch/sizeof(float);
/* Read from device global memory */
f0now = f0_data[i2d];
f1now = f1_data[i2d];

/* Calc flow, feq, collide, as CPU code */

/* Write to device global memory */
f0_data[i2d] = rtau1 * f0now + rtau * f0eq;
f1_data[i2d] = rtau1 * f1now + rtau * f1eq;
```

# CPU / GPU demo

# Part 5: Alternatives to NVIDIA

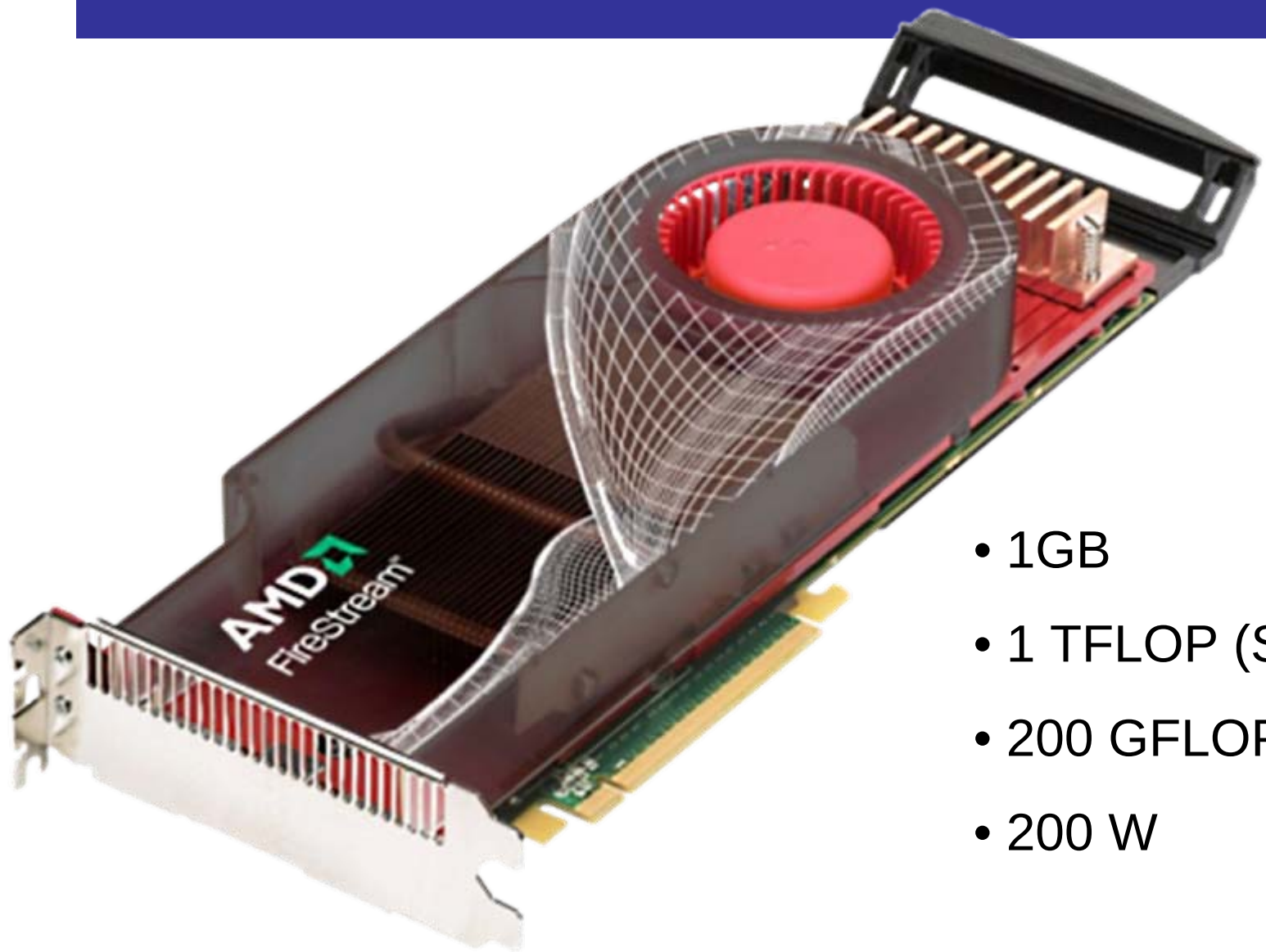
# NVIDIA Tesla S1070

- 4 Tesla C1060 GPUs

  per card:

  - 4 GB
  - 1 TFLOP (SP)
  - 90 GFLOP (DP)
  - 200 W

# AMD Firestream 9250

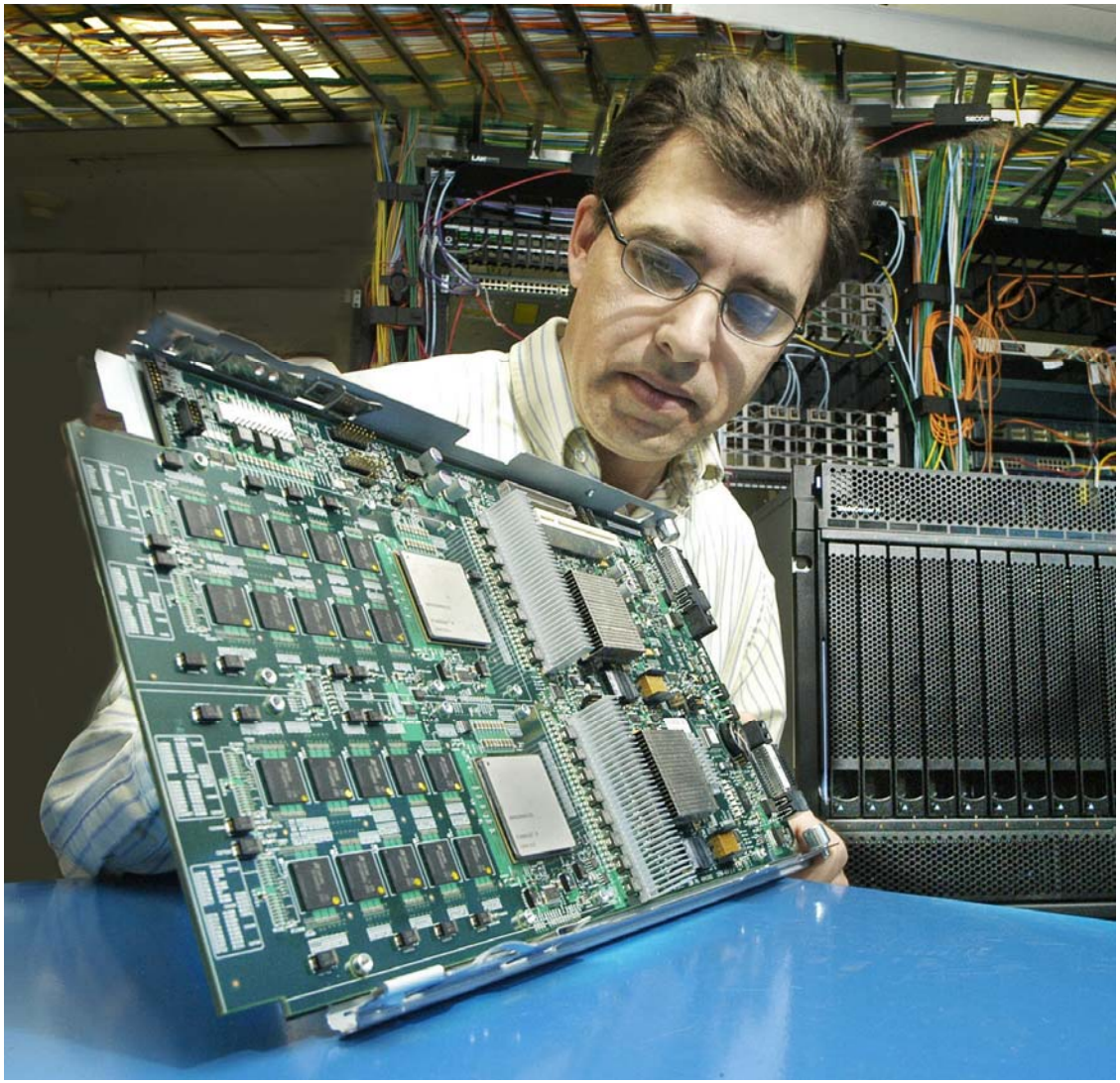



- 1GB
- 1 TFLOP (SP)
- 200 GFLOP (DP)
- 200 W

# ClearSpeed e710


- 2 GB
- 96 GFLOP (DP)
- 25 W

# IBM PowerXCell8i



- up to 32 GB
- 200 GFLOP (SP)
- 102 GFLOP (DP)
- 92 W

# Too much choice!

- Each device has
  - different hardware characteristics
  - different software (C extensions)
  - different developer tools

- Standardisation – in some form – is needed

# Part 6: Conclusions

# Conclusions

- Many science applications fit the SIMD model
- GPUs are commodity SIMD chips
- Good speedups (10x – 100x) can be achieved

# Conclusions

- Many science applications fit the SIMD model
- GPUs are commodity SIMD chips
- Good speedups (10x – 100x) can be achieved

- GPGPU is evolving:
  1. Making it work at all (graphics APIs)
  2. Doing it better (high level APIs)
  3. Doing it right (portable, modular building blocks)

# Conclusions

- Many science applications fit the SIMD model
- GPUs are commodity SIMD chips
- Good speedups (10x – 100x) can be achieved

- GPGPU is evolving:
  1. Making it work at all (graphics APIs)
  2. Doing it better (high level APIs)
  3. Doing it right (portable, modular building blocks)

# www.many-core.group.cam.ac.uk

UNIVERSITY OF CAMBRIDGE 800 YEARS 1209–2009

Search

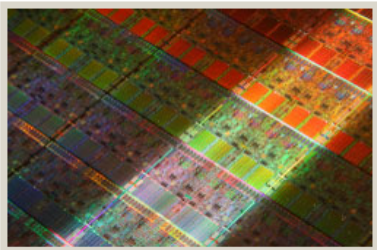
Contact us | A–Z | Email & phone search

## many-core.group

University of Cambridge › many-core.group

Many-core computing devices have large numbers of processors (cores) on a single chip. Such configurations are attractive because they can achieve a greater performance (calculations per second) for a given amount of electrical power than their single-core cousins. CPUs are heading down this route with dual-core and quad-core processors now commonplace. However, accelerator add-on cards or chips are also available today which have over 100 cores; of these, the graphics processing unit (GPU) is the most widespread.

**many-core.group** is a site where researchers at Cambridge University who are using many-core devices to accelerate their scientific applications can show their results and describe their experiences.



### Events

29 Oct 2008
**Many-core Computing Workshop, Cambridge**

15-21 Nov 2008
**Super Computing 2008, Austin, Texas**

15-16 Jan 2009
**Numerical Accuracy and Reliability workshop at Queen's University Belfast**

### On this site:

- GPGPU
- People
- Projects
- Contact