

# Lecture 18

The future of Higher Performance Computing:  
Technological Trends

# Announcements

- Final Exam Review on Thursday
- Practice Questions
- A5 due in section on Friday

# Today's lecture

- Technological Trends
- Case Studies
  - Blue Gene/L
  - STI Cell Broadband Engine
  - Nvidia Tesla

# Technological Trends

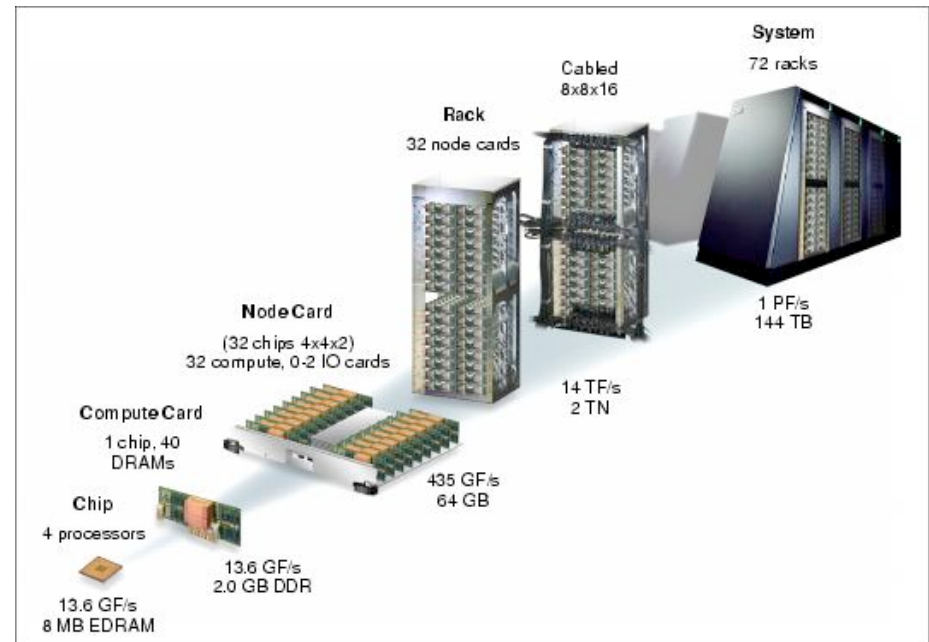
- IBM Blue Gene
- STI Cell Broadband Engine
- Nvidia Tesla

# Blue Gene

- IBM-US Dept of Energy collaboration
- First generation: Blue Gene/L
  - 64K dual processor nodes: 180 (360) TeraFlop peak
  - 1 TeraFlop = 1,000 GigaFlops
- Low power
  - Relatively slow processors; power PC 440
  - Small memory (256 MB)
- High performance interconnect

# Next Generation: Blue Gene/P

- Largest Installation at Argonne National Lab:  
163,840 cores
  - 4-way nodes
  - PowerPC 450 (850 MHz)
  - 2 GB memory per node
- Peak performance:  
13.6 Gflops/node  
= 557 TeraFlops total  
= 0.56 Petaflops



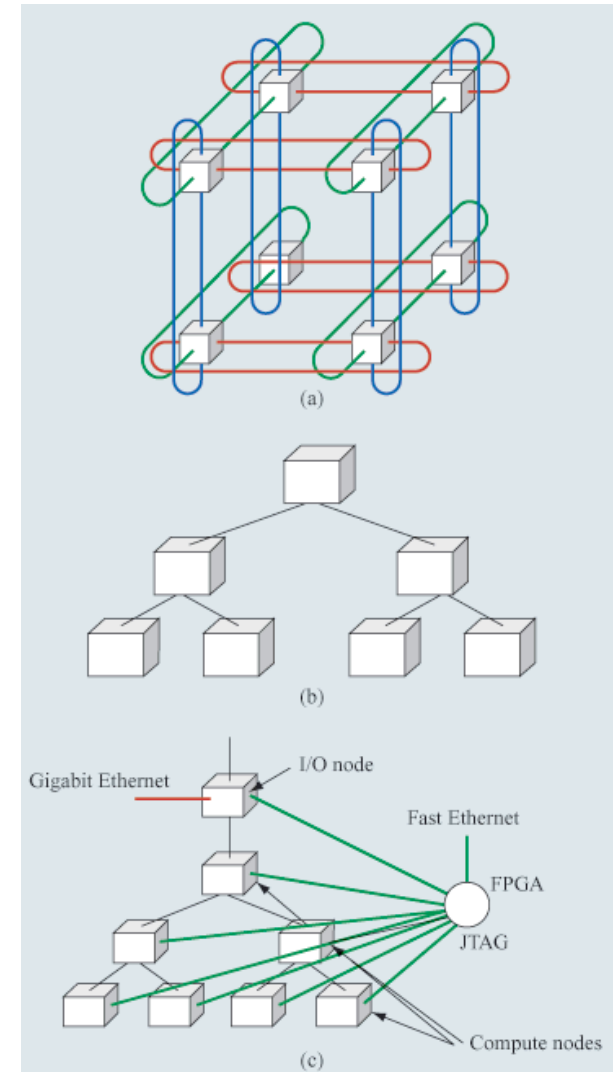
<http://www.redbooks.ibm.com/redbooks/SG247287>

# Blue Gene/P Interconnect

- 3D toroidal mesh (end around)
  - 5.1 GB/sec bidirectional bandwidth / node
  - 1.7/2.6 TB/sec bisection bandwidth (72k cores)
  - 0.5  $\mu$ s to 5  $\mu$ s latency (1 hop, farthest)
  - MPI: 3  $\mu$ s to 10  $\mu$ s
- Rapid combining-broadcast network
  - 1-way latency 1.3  $\mu$ s (5  $\mu$ s in MPI)
  - Also used to move data between I/O and compute nodes
- Low latency barrier and interrupt
  - One way: 0.65  $\mu$ s
  - 1.6  $\mu$ s in MPI

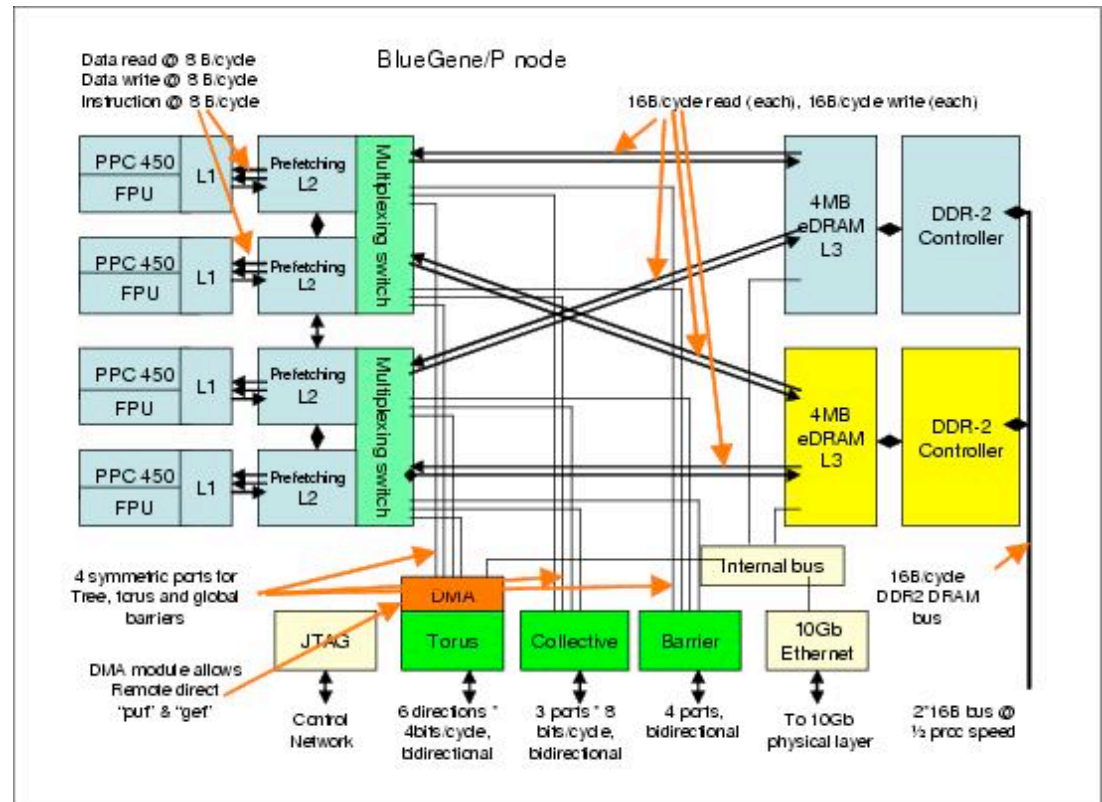
Image courtesy of IBM

<http://www.research.ibm.com/journal/rd/492/gara4.gif>



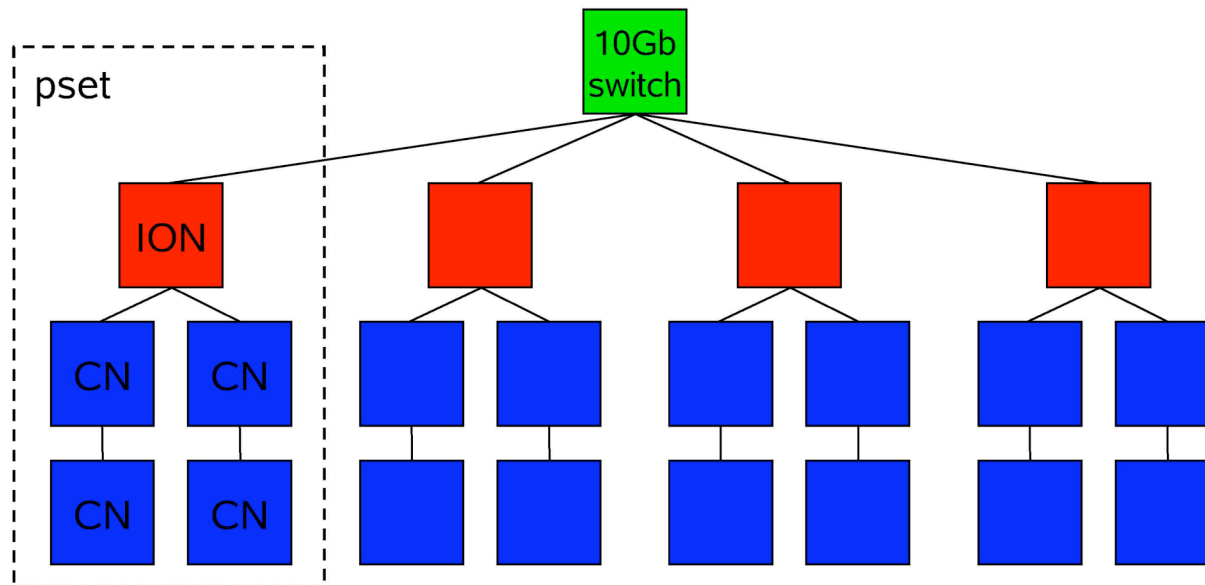
# Compute nodes

- Six connections to torus network @ 425 MB/sec/link (duplex)
- Three connections to global collective network @ 850 MB/sec/link
- Network routers are embedded within the processor



# Processor sets and I/O

- Each I/O node services a set of compute nodes



# Programming modes

- Virtual node: each node runs 4 MPI processes, 1/core
  - Memory and torus network shared by all processes
  - Shared memory is available between processes.
- Dual node: each node runs 2 MPI processes, 1 or 2 threads/process
- Symmetrical Multiprocessing:  
each node runs 1 MPI process, up to 4 threads

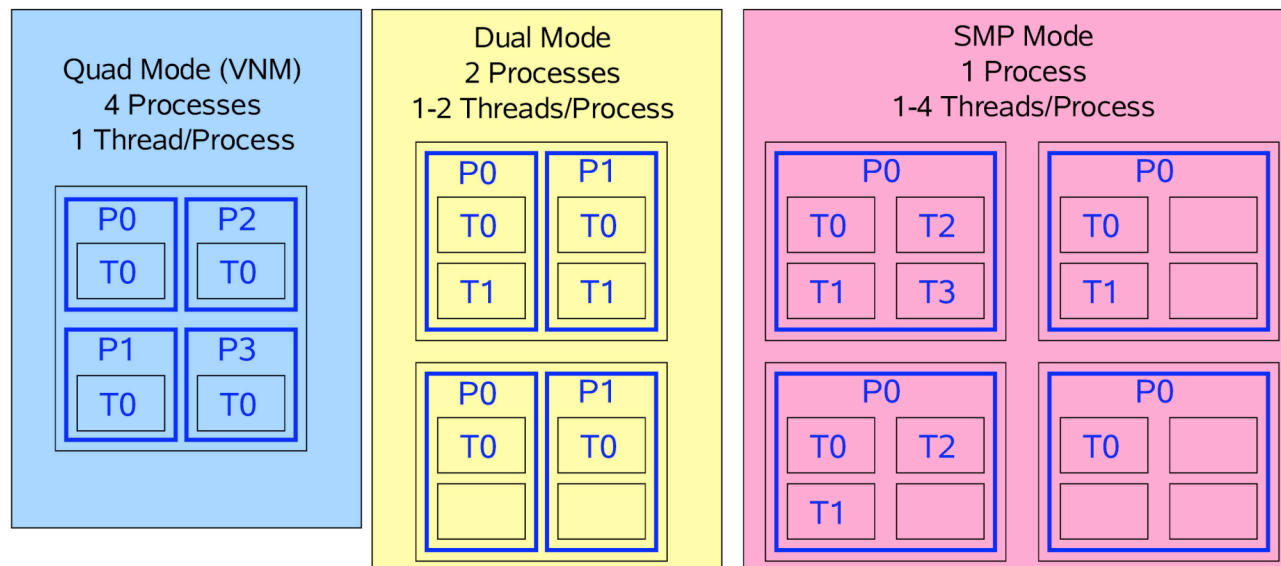


Image courtesy of IBM

# Technological Trends

- IBM Blue Gene/L
- **STI Cell Broadband Engine**
- Nvidia Tesla

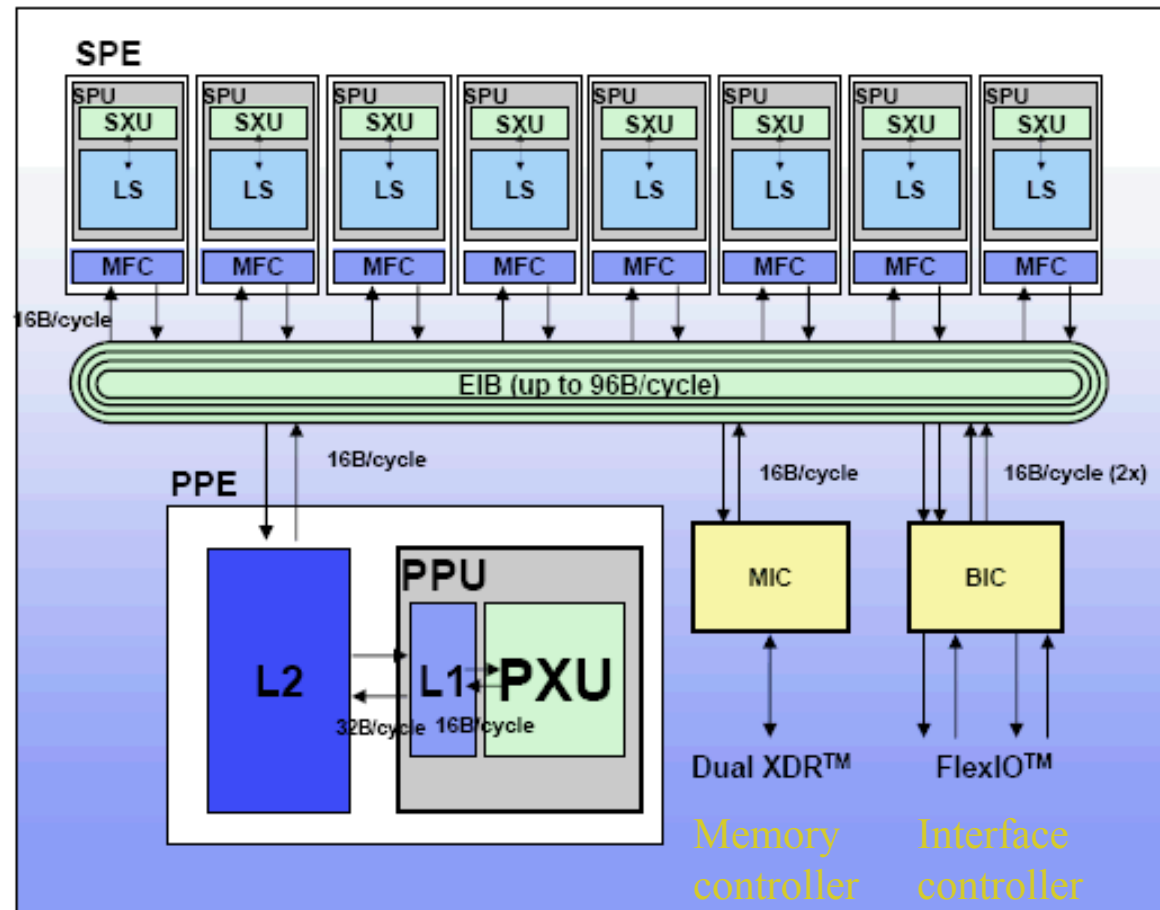
# Cell Overview

- Chip multiprocessor for multithreaded applications
- Adapted power PC: 8 SPE + 1 PPE
- Shared address space
- SIMD (VMX)
- Software managed resources on the SPE
- Optimize the common case

# Cell BE Block diagram

Coherent  
on-chip bus

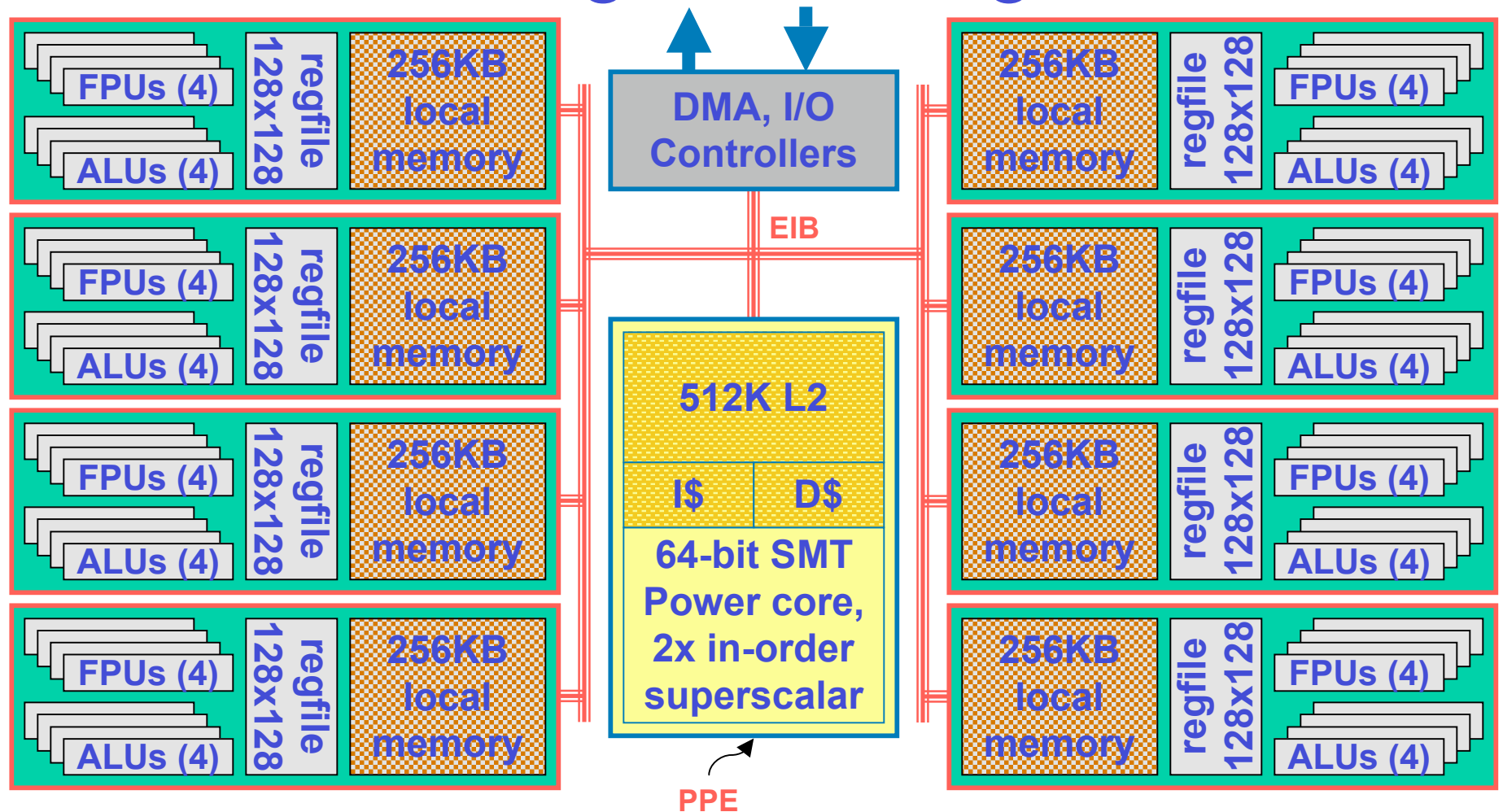
64 bit Power  
architecture  
core  
core



Source: M. Geschwind et al., Hot Chips-17, August 2005

<http://domino.watson.ibm.com/comm/research.nsf/pages/r.arch.innovation.html>

# Block diagram showing detail



Courtesy Sam Sandbote

# SPE - Synergistic processing element

- 128-bit x 128 entry register file
- 256 KB local store
- Software managed data transfers  
Rambus XDR DRAM interface
  - 3.2 GB/sec/SPE
  - 25.6 GB/s aggregate
- DMA  $\Leftrightarrow$  Local Store via EIB
- EIB: 4 x 128 bit wide
  - 200 GB/sec
  - Multiple simultaneous requests:  
16 x 16kb
  - Priority: DMA, L/S, Fetch

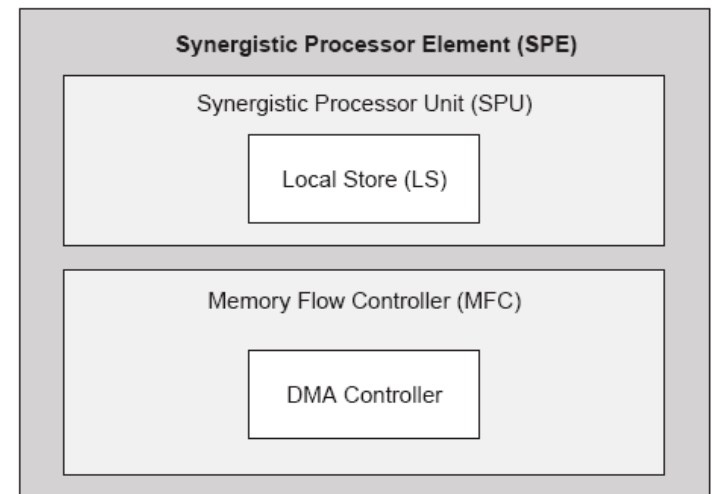


Image courtesy Keith O'Connor,  
<http://isg.cs.tcd.ie/oconork/presentations/CellBroadbandEngine.ppt>

# Usage scenarios

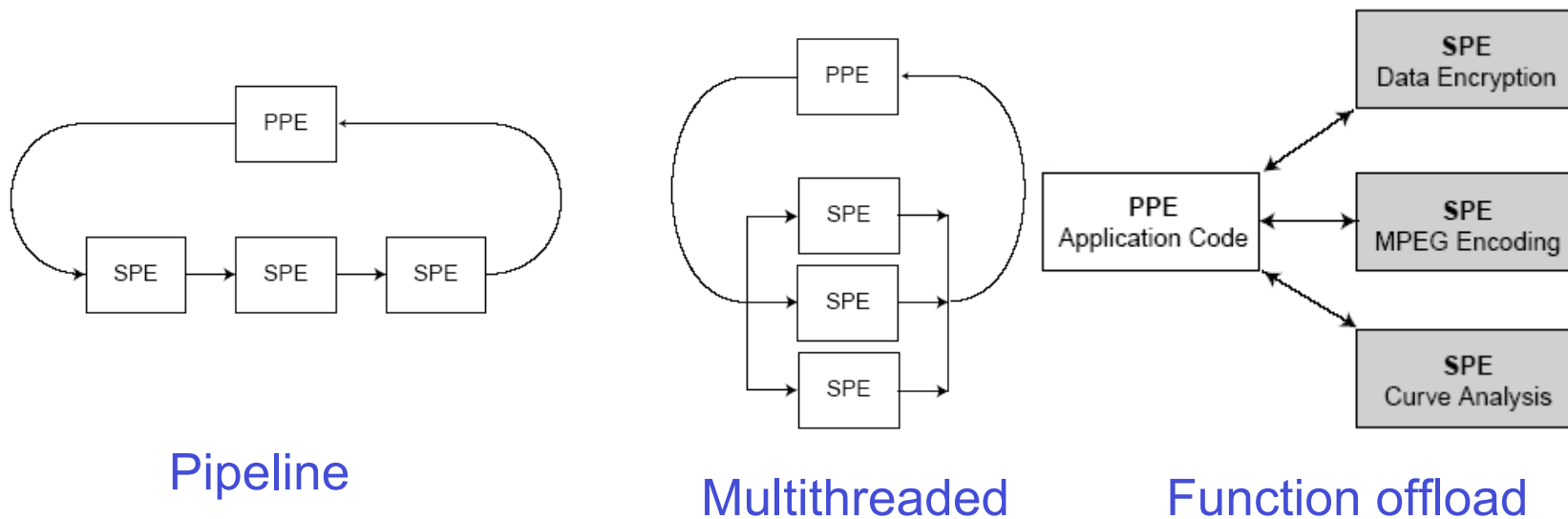


Image courtesy Keith O'Connor,  
<http://isg.cs.tcd.ie/oconork/presentations/CellBroadbandEngine.ppt>

# Technological Trends

- Blue Gene/L
- STI Cell Broadband Engine
- **Nvidia Tesla**

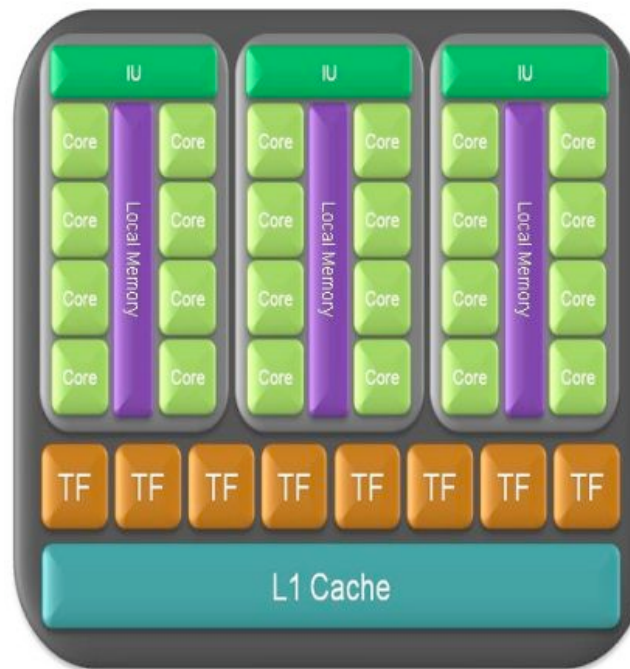
# NVIDIA GeForce GTX 280

- Hierarchically organized clusters of streaming multiprocessors
  - 240 cores @ 1.296 GHz
  - Peak performance 933.12 Gflops/s
- Parallel computing or graphic mode
- 1 GB memory (frame buffer)
- 512 bit memory interface @ 132 GB/s



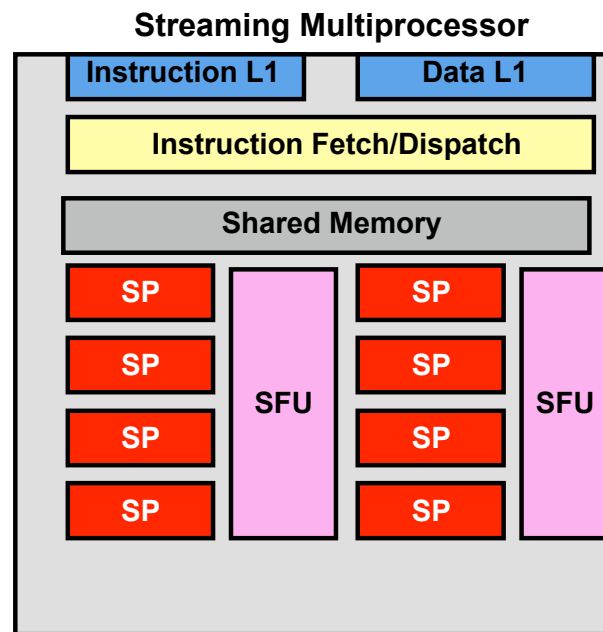
# Streaming processing cluster

- Each cluster contains 3 streaming multiprocessors
- Each multiprocessor contains 8 cores that share a local memory
- Each core supports scalar processing
- Double precision is 10× slower than floating point



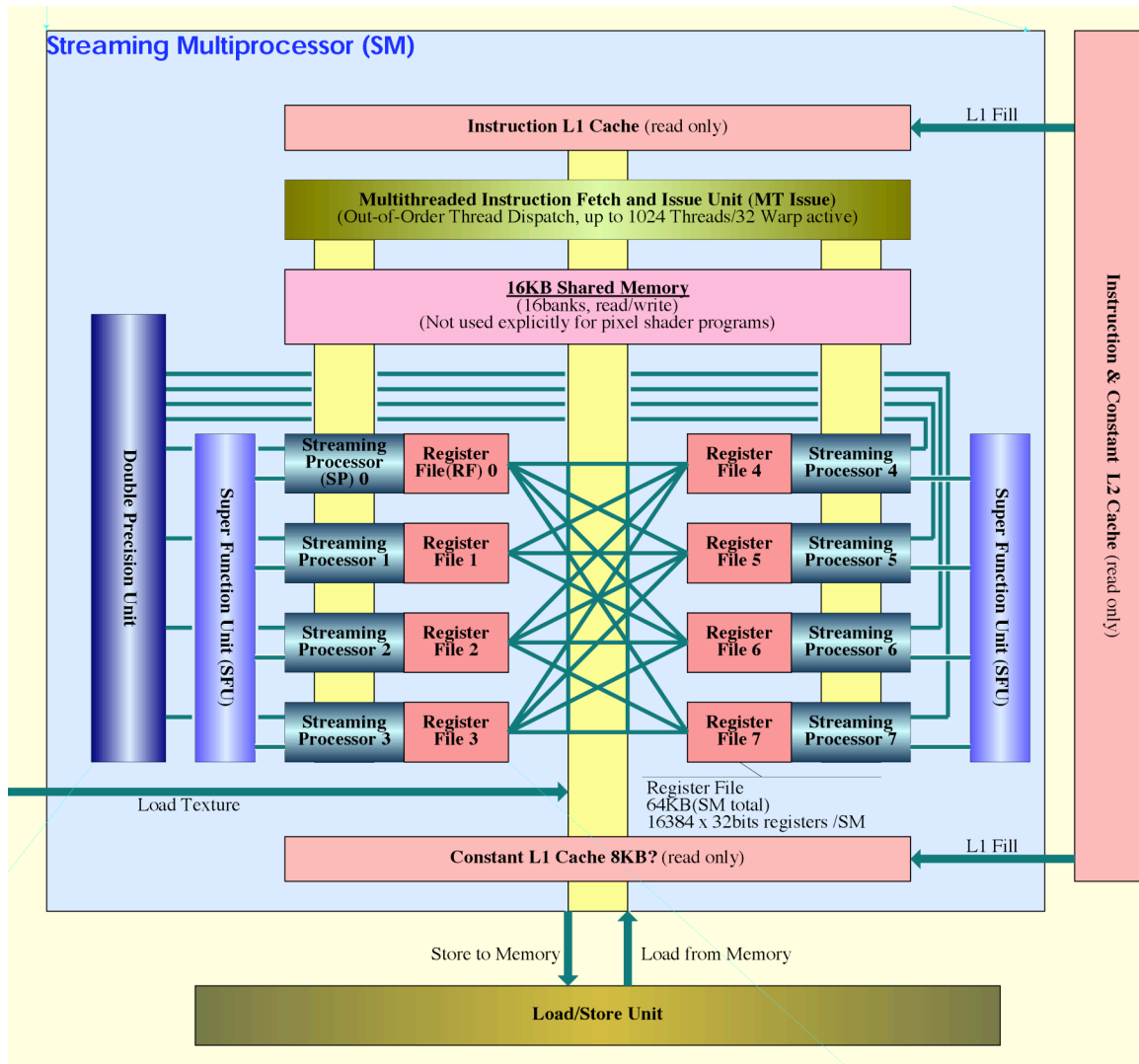
# Streaming Multiprocessor

- 8 cores (Streaming Processors)
  - Each core contains a fused multiply adder (32 bi)
- Each SM contains one 64 bit fused multiply adder
- 2 Super Function Units (SFUs), each contains 2 fused multiply-adders
- 16 KB shared memory
- 16K registers



Courtesy DavidKirk/NVIDIA  
Wen-mei Hwu/UIUC

# Detail inside the Streaming Multiprocessor

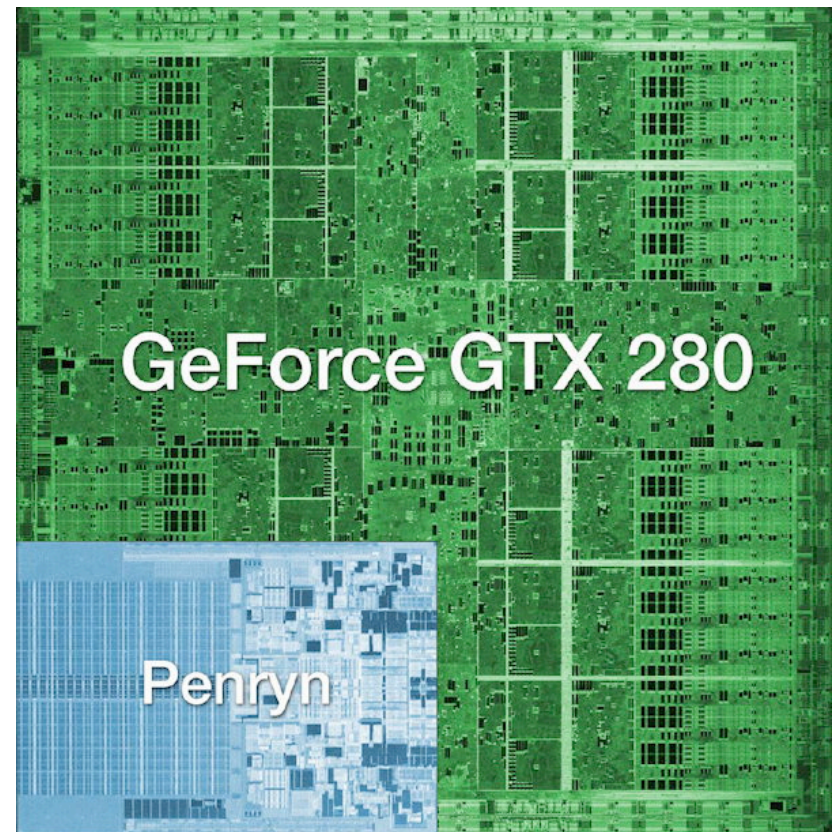
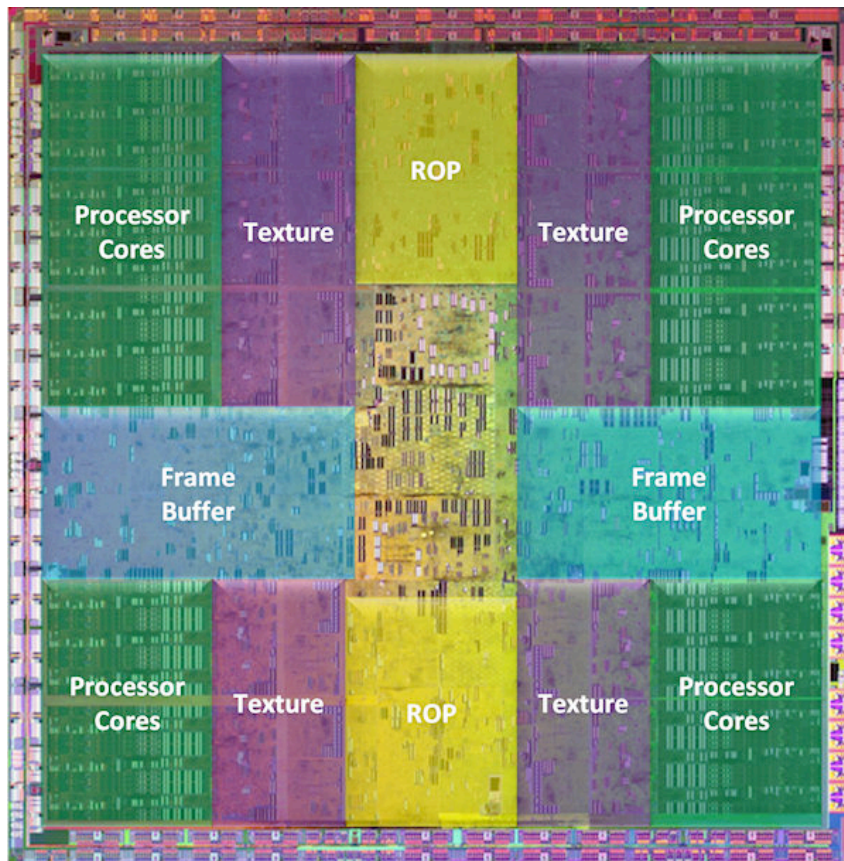


Courtesy H. Goto

# Die pictures of GTX 280

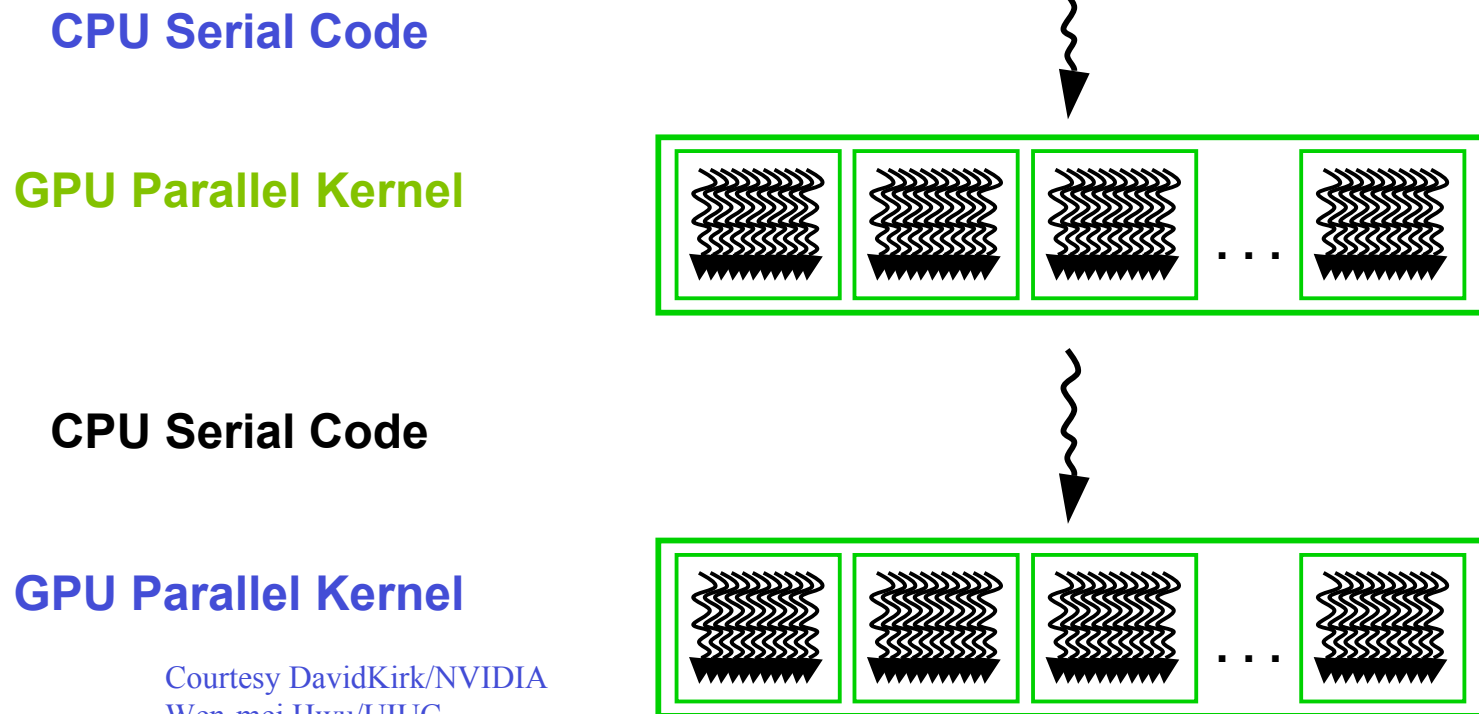
GTX 280 has 1.4B transistors

Dual Core Penryn vs GTX 280



# CUDA

- Programming environment with extensions to C
- Model: SPMD execution on the CPU, run a sequence of multi-threaded kernels on the “device”
- Threads are extremely lightweight

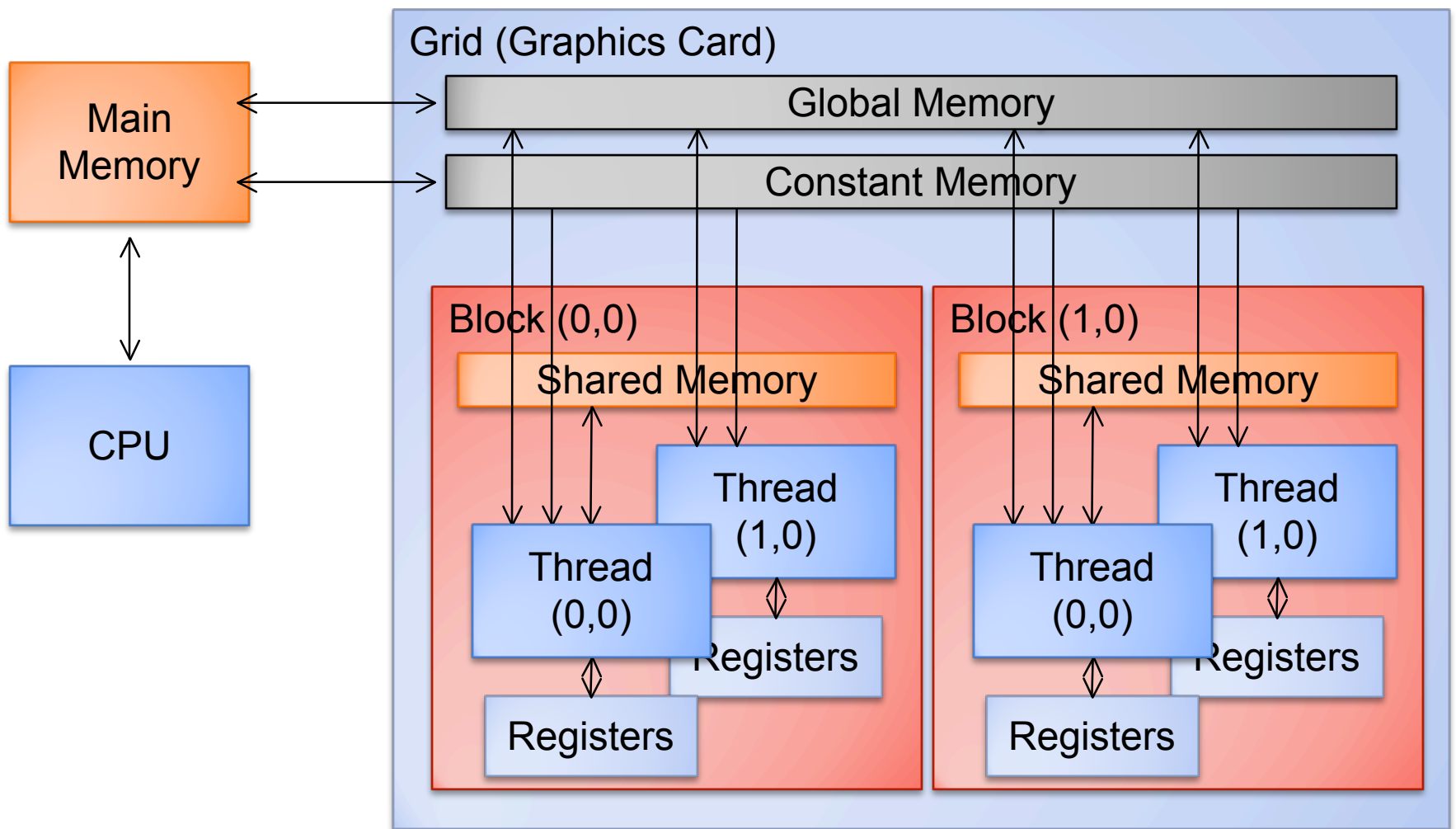


Courtesy David Kirk/NVIDIA  
Wen-mei Hwu/UIUC

# CUDA

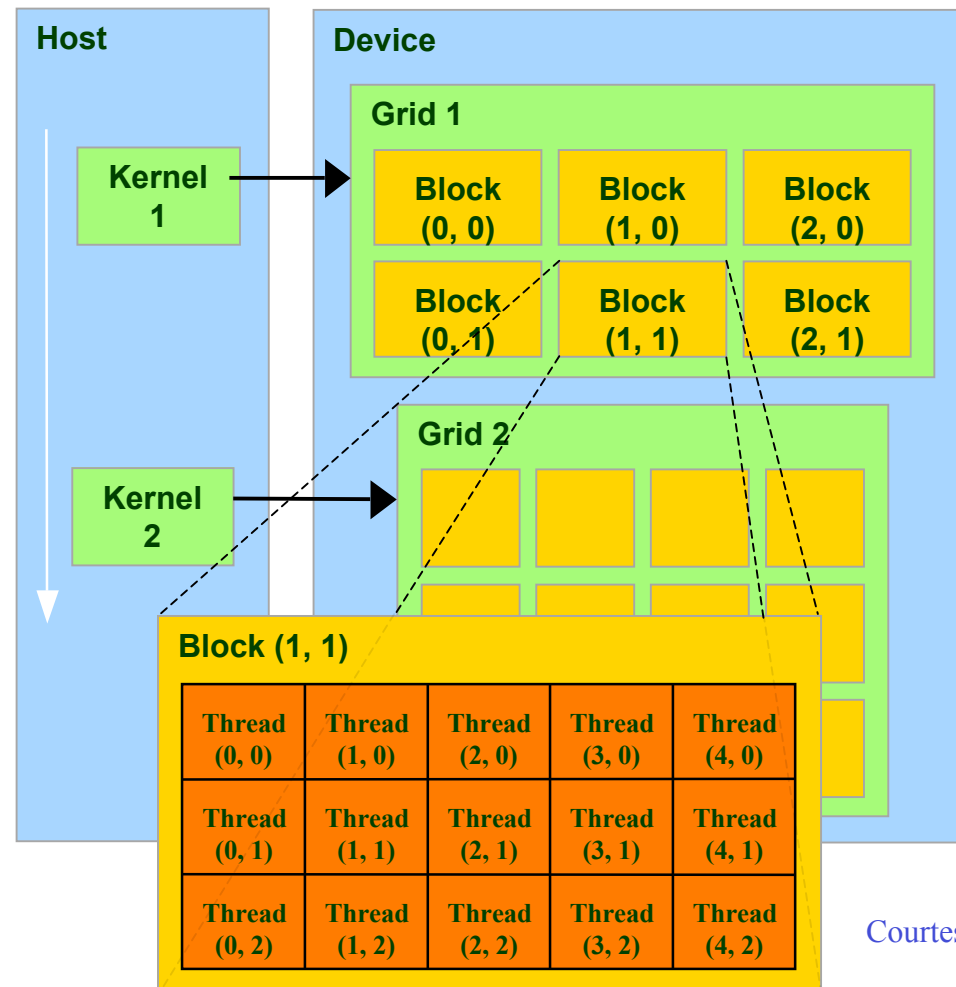
- Hierarchical thread organization
- Basic computational unit is the *grid*
- Grids composed of thread *blocks*
  - A cooperative block of threads: synchronize and share data via fast on-chip shared memory
  - Threads in different blocks cannot communicate except through slow global memory
  - Blocks within a grid are virtualized
  - May configure number of threads in the block and the number of blocks
- Parallel calls to a kernel specify the layout  
`kernelFunction <<< gridDims, blockDims >>> (args)`
- Compiler will re-arrange loads to hide latencies

# CUDA Memory Architecture



# Block scheduling (8800)

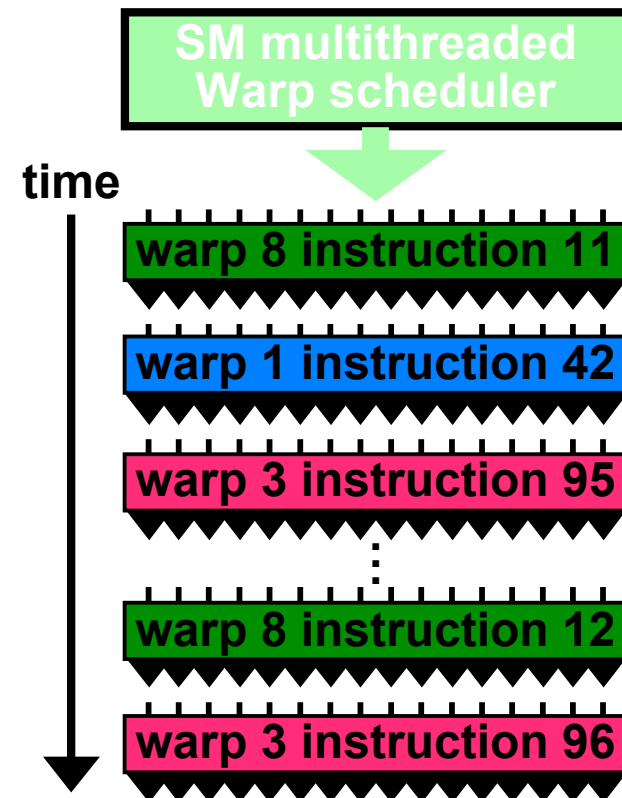
Threads assigned to SM in units of blocks, up to 8 for each SM



Courtesy of NVIDIA

# Warp scheduling (8800)

- Blocks contain multiple *warps*, a group of SIMD threads
- Half-warps are the unit of scheduling (16 threads currently)
- Hardware scheduled warps hide latency (zero overhead)
  - Scheduler looks for an eligible warp with all operands ready
  - All threads in the warp execute the same instruction
  - All branches followed: serialization, instructions can be disabled
- Many warps need to hide memory latency
- Registers shared by all threads in a block



Courtesy DavidKirk/NVIDIA  
Wen-mei Hwu/UIUC

# CUDA Coding Example

```
void addMatrix
(float *a, float *b, float *c, int N)
{
    int i, j, idx;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            idx = i + j*N;
            c[idx] = a[idx] + b[idx];
        }
    }
}
void main()
{
    . . .
    addMatrix(a, b, c, N);
}
(a)
```

```
__global__ void addMatrixG
(float *a, float *b, float *c, int N)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    int j = blockIdx.y*blockDim.y + threadIdx.y;
    int idx = i + j*N;
    if (i < N && j < N)
        c[idx] = a[idx] + b[idx];
}

void main()
{
    dim3 dimBlock (blocksize, blocksize);
    dim3 dimGrid (N/dimBlock.x, N/dimBlock.y);
    addMatrixG<<<dimGrid, dimBlock>>>(a, b, c, N);
}
(b)
```

Figure 8. Serial C (a) and CUDA C (b) examples of programs that add arrays.

## Programming issues

- Branches serialize execution within a warp
- Registers dynamically partitioned across each block in a Streaming Multiprocessor
- Tradeoff: more blocks with fewer threads or more threads with fewer blocks
  - Locality: want small blocks of data (and hence more plentiful warps) that fit into fast memory
  - Register consumption
  - Scheduling: hide latency