GPU Hardware
CS 380P

Paul A. Navrátil

Manager – Scalable Visualization Technologies

Texas Advanced Computing Center

with thanks to Don Fussell for slides 15-28
and Bill Barth for slides 36-55

CPU vs. GPU characteristics

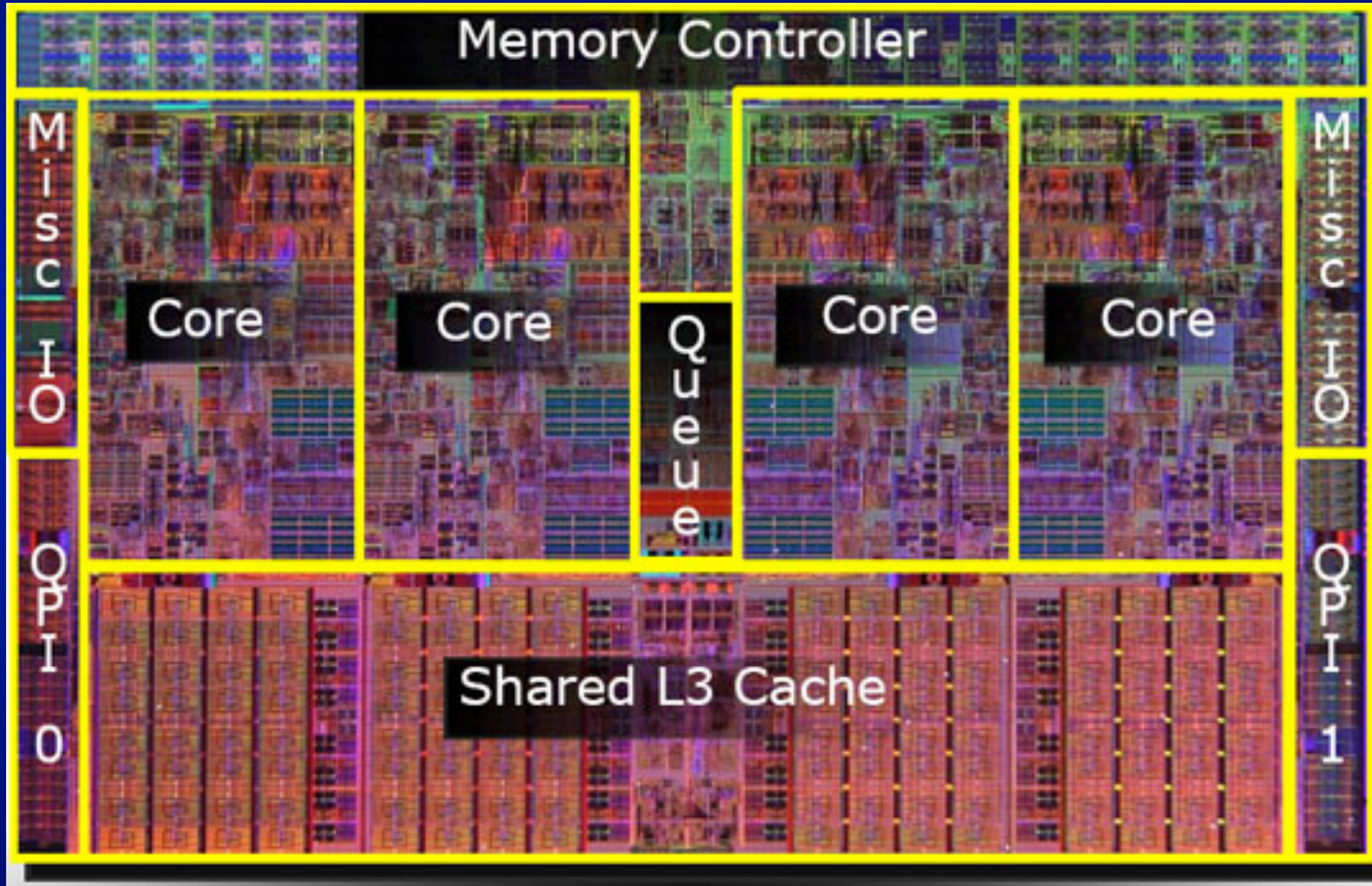
CPU

- Few computation cores
  - Supports many instruction streams, but keep few for performance
- More complex pipeline
  - Out-of-order processing
  - Deep (tens of stages)
  - Became simpler (Pentium 4 was complexity peak)
- Optimized for serial execution
  - SIMD units less so, but lower penalty for branching than GPU

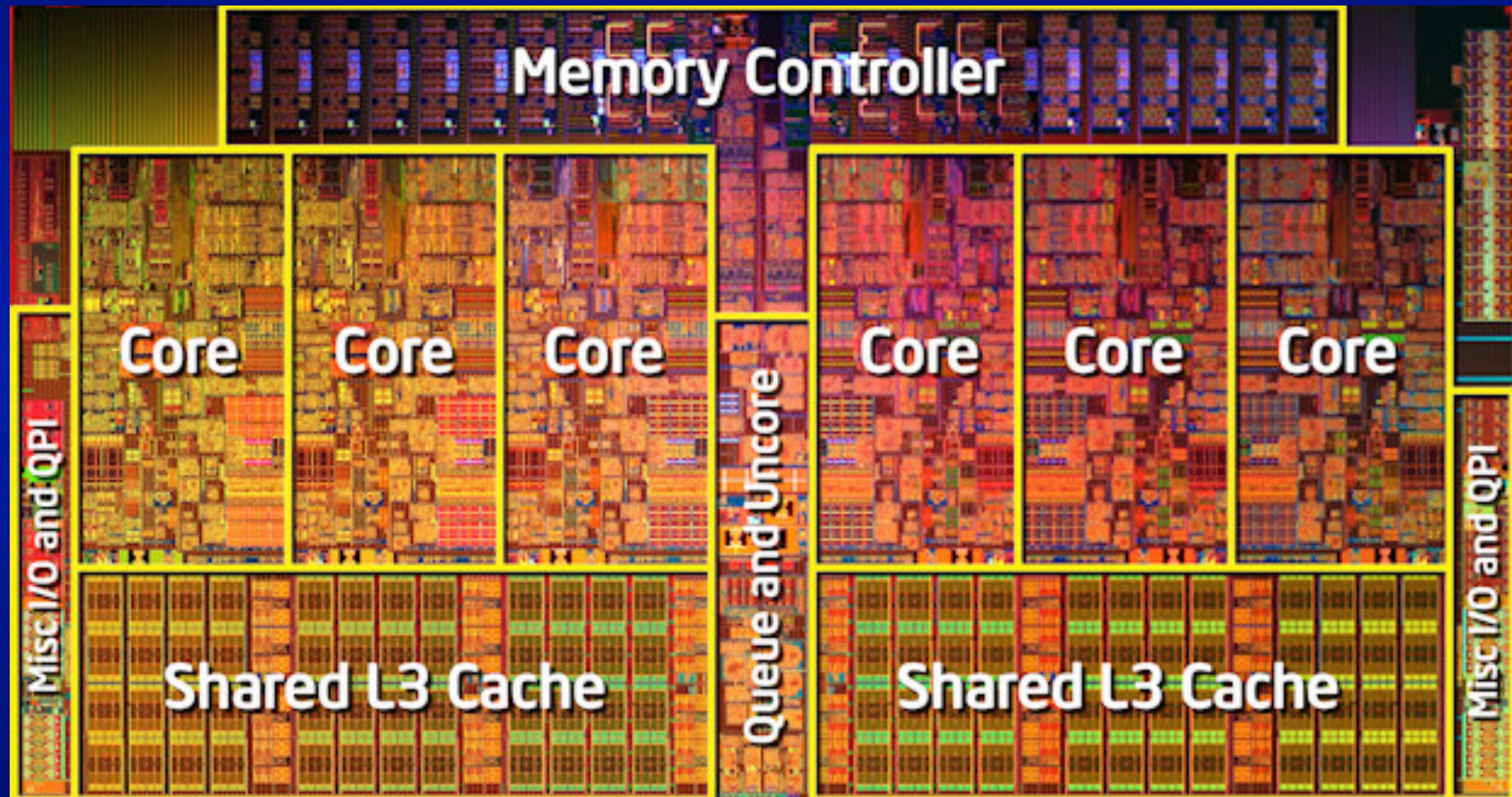
GPU

- Many of computation cores
  - Few instruction streams
- Simple pipeline
  - In-order processing
  - Shallow (< 10 stages)
  - Became more complex
- Optimized for parallel execution
  - Potentially heavy penalty for branching

Intel Nehalem
(Longhorn nodes)

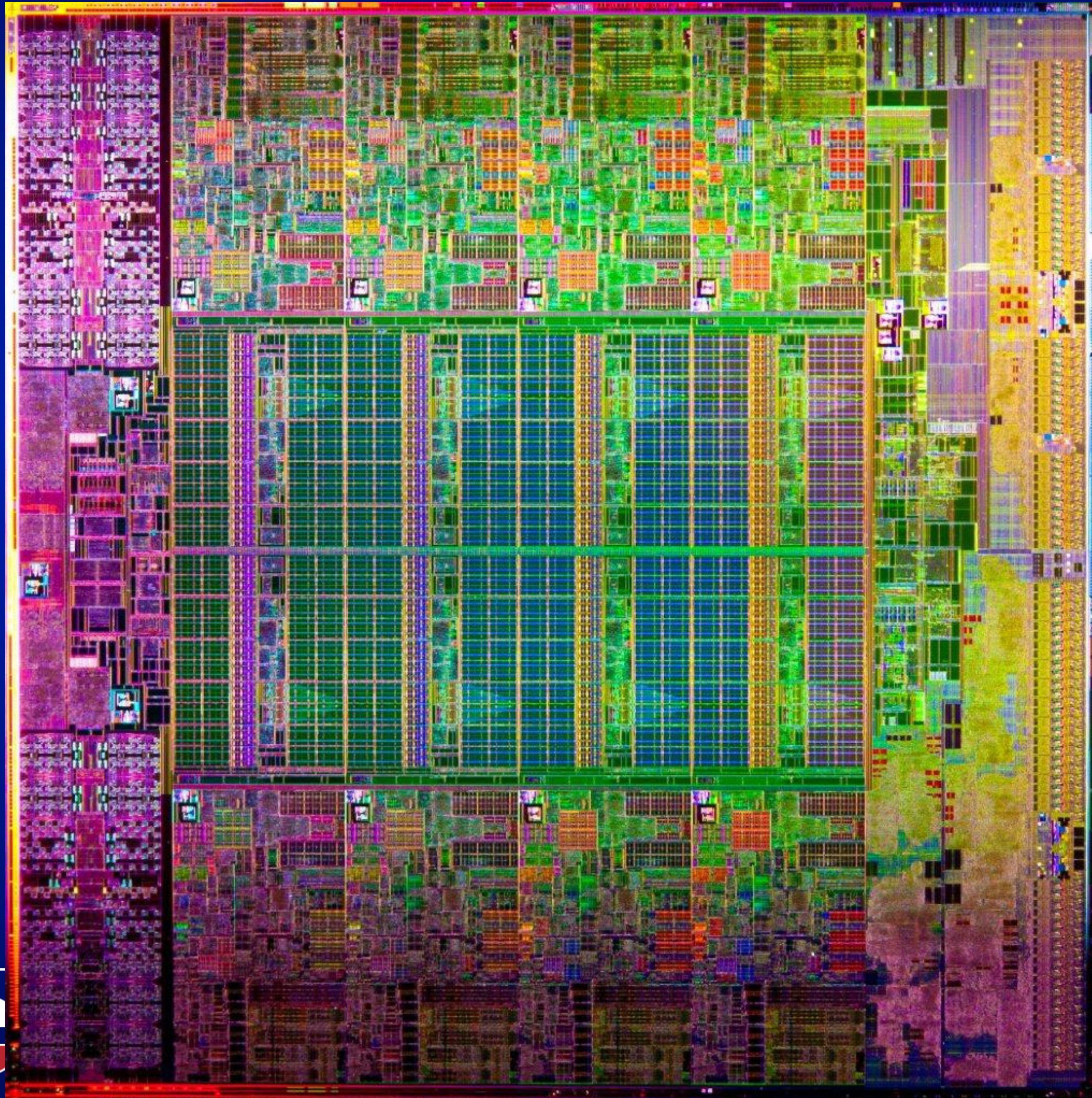


Intel Westmere
(Lonestar nodes)



TACC

Intel Sandy Bridge
(Stampede nodes)

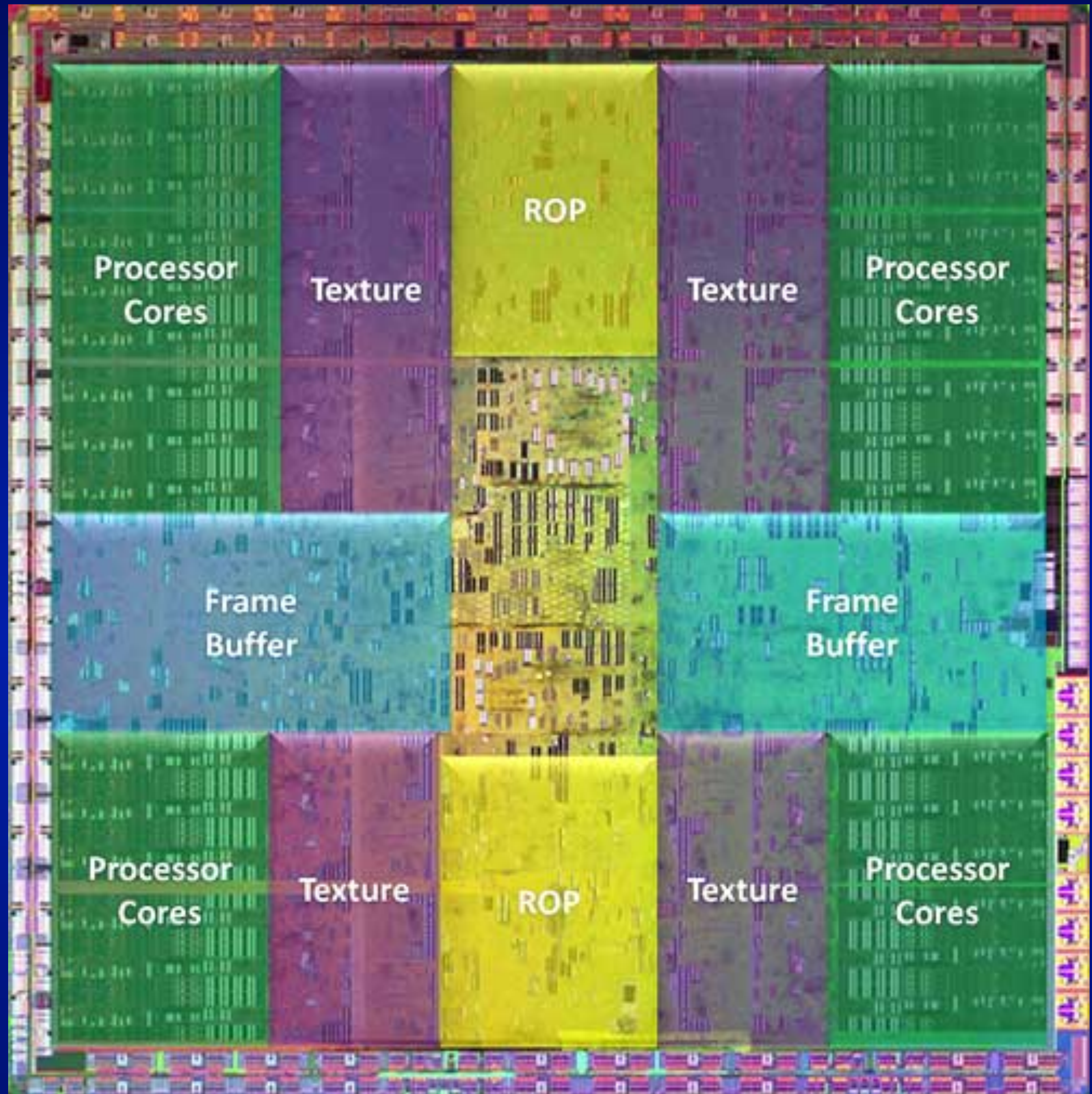


TACC

Intel Sandy Bridge (Stampede nodes)

TACC

NVIDIA GT200 (Longhorn nodes)

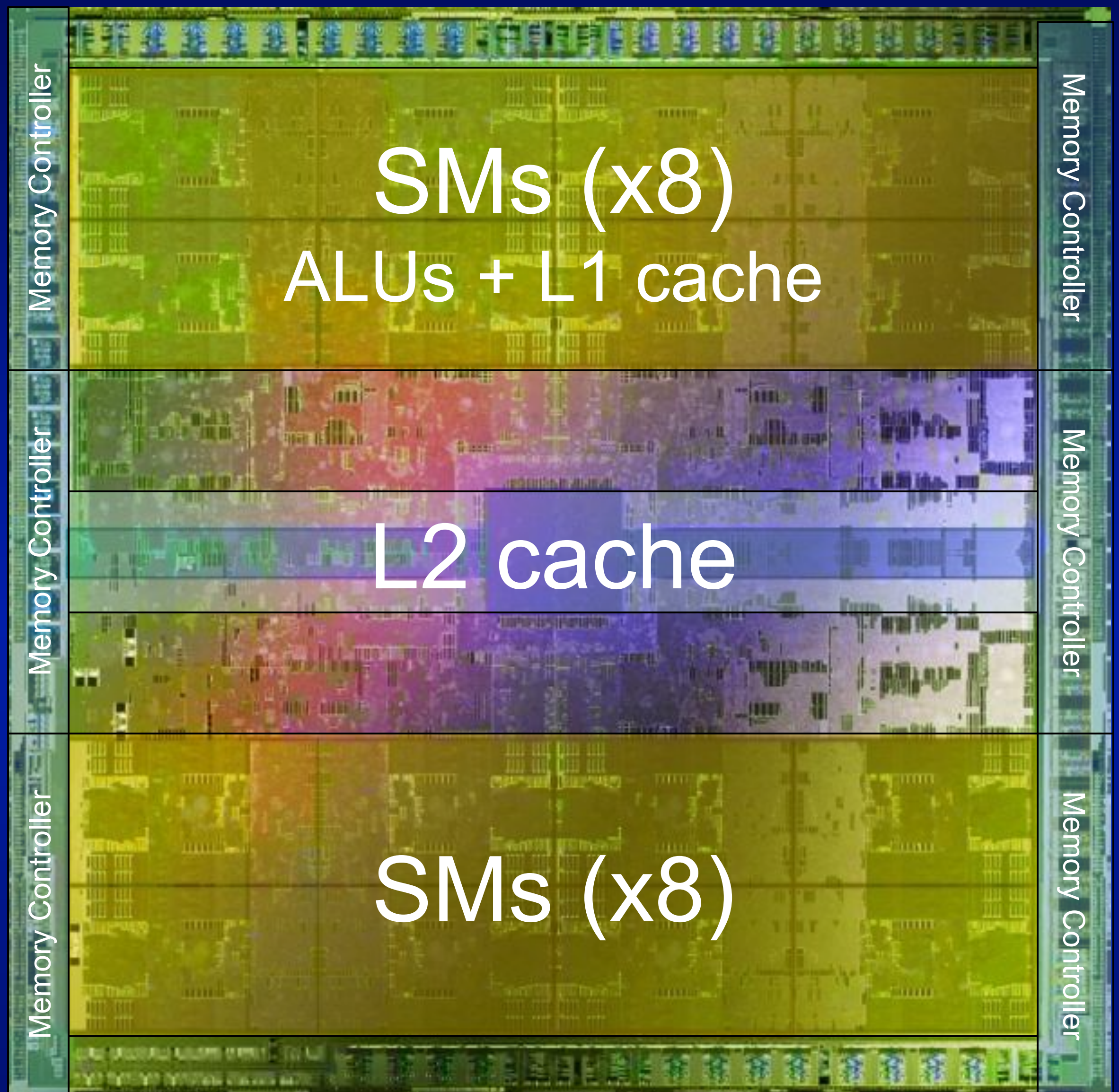


TACC

NVIDIA GF100

Fermi

(Lonestar nodes)

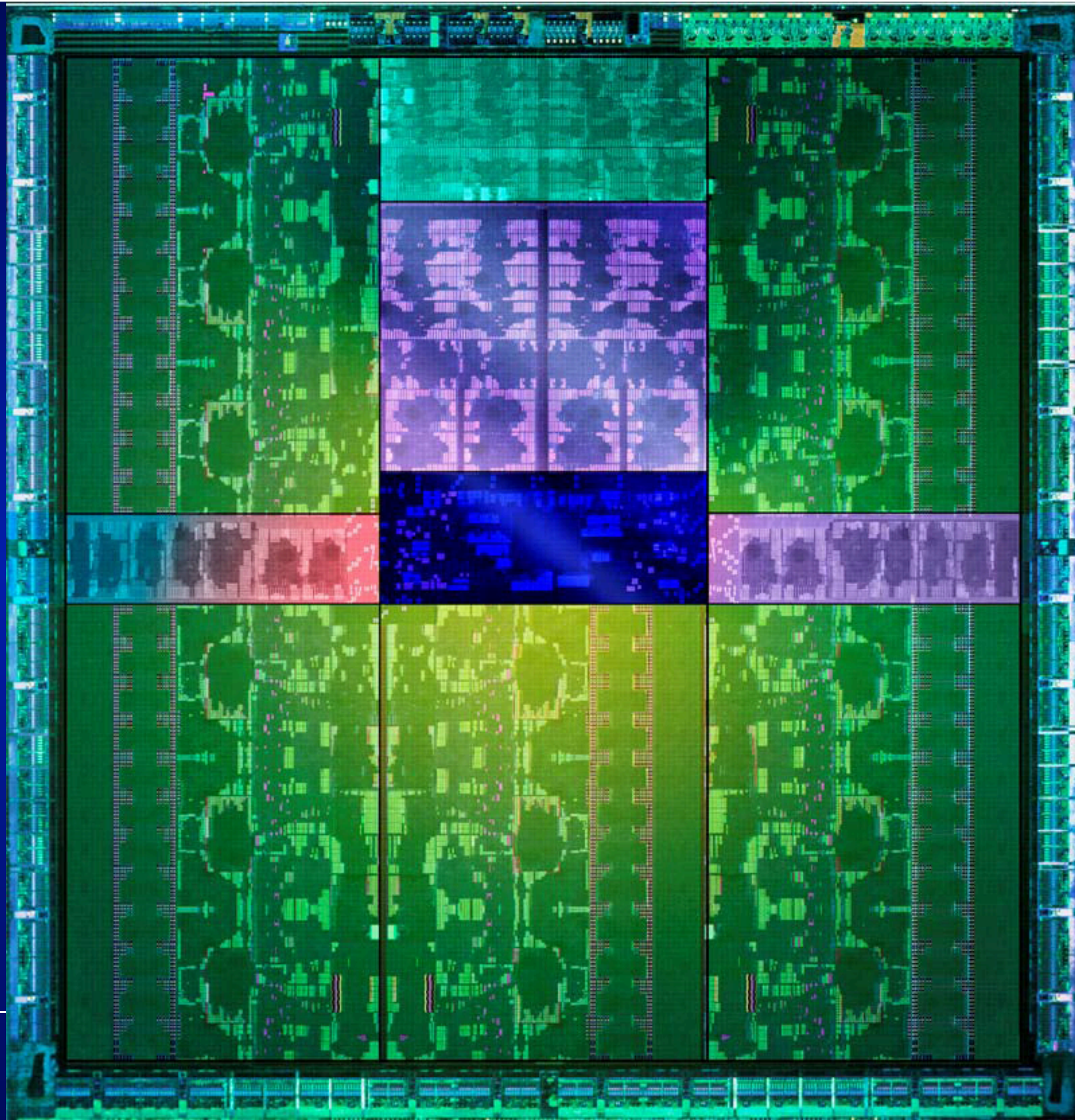


TACC

NVIDIA GK110

Kepler

(Stampede nodes)



TACC

PCI Express 3.0 Host Interface
GigaThread Engine
SMX
L2 Cache
Memory Controller

SMX

Instruction Cache

Warp Scheduler | Warp Scheduler | Warp Scheduler | Warp Scheduler

Dispatch | Dispatch | Dispatch | Dispatch | Dispatch | Dispatch | Dispatch | Dispatch

Register File (65,536 x 32-bit)

Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU

Interconnect Network

64 KB Shared Memory / L1 Cache

48 KB Read-Only Data Cache

Tex	Tex	Tex	Tex	Tex	Tex	Tex	Tex
Tex	Tex	Tex	Tex	Tex	Tex	Tex	Tex

Hardware Comparison
(Longhorn- and Lonestar-deployed versions)

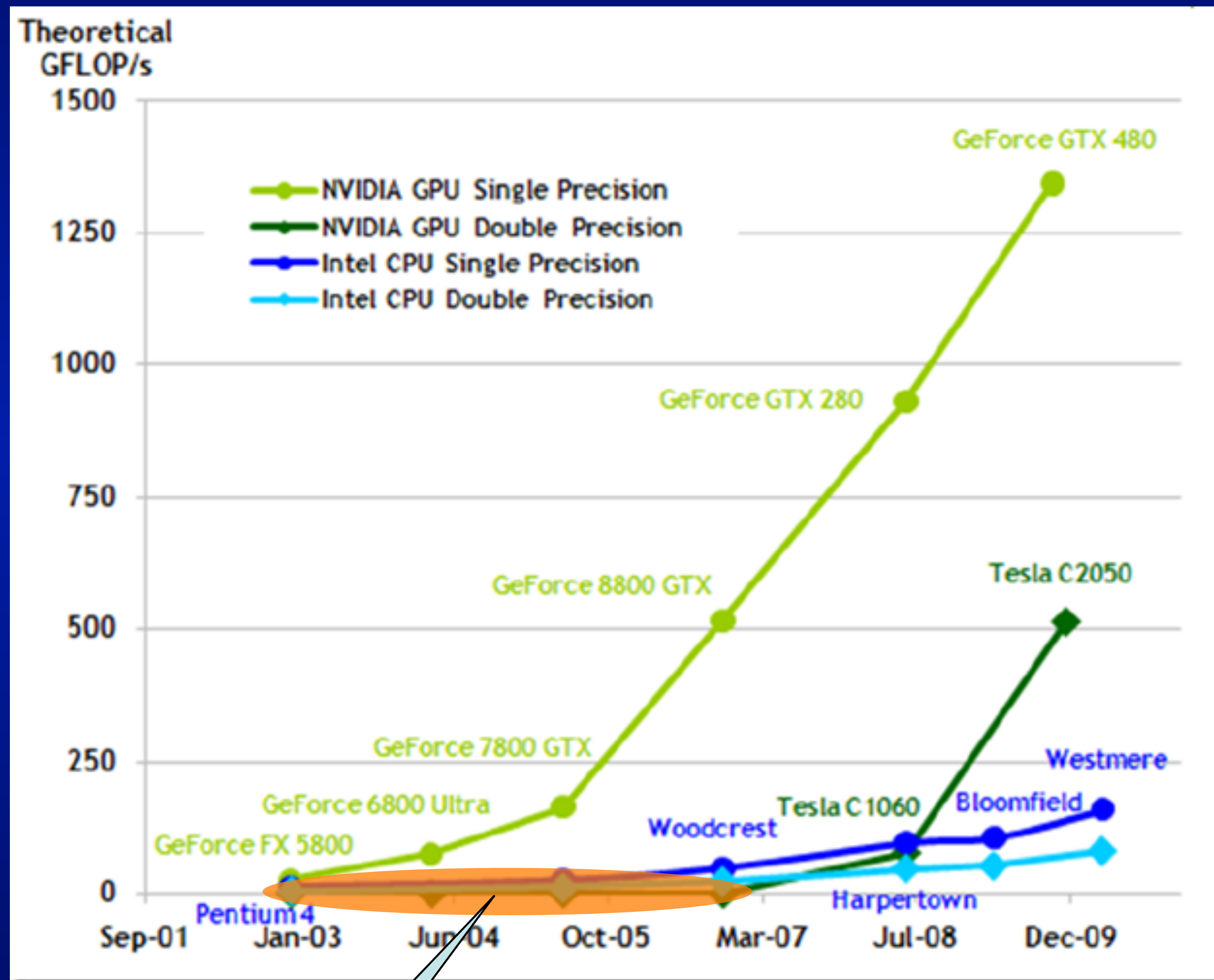
	Nehalem E5540	Westmere X5680	Sandy Bridge E5-2680	Tesla Quadro FX 5800	Fermi Tesla M2070	Kepler Tesla K20
Functional Units	4	6	8	30	14	13
Speed (GHz)	2.53	3.33	2.7	1.30	1.15	.706
SIMD / SIMT width	4	4	4	8	32	32
Instruction Streams	16	24	32	240	448	2496
Peak Bandwidth DRAM->Chip (GB/s)	35	35	51.2	102	150	208

A Word about FLOPS

- Yesterday's slides calculated Longhorn's GPUs (NVIDIA Quadro FX 5800) at 624 peak GFLOPS…
- … but NVIDIA marketing literature lists peak performance at 936 GFLOPS!?

- NVIDIA's number includes the Special Function Unit (SFU) of each SM, which handles unusual and expectional instructions (transcendentals, trigonometrics, roots, etc.)
- Fermi marketing materials do not include SFU in FLOPs measurement, more comparable to CPU metrics.

The GPU's Origins: Why They are Made This Way

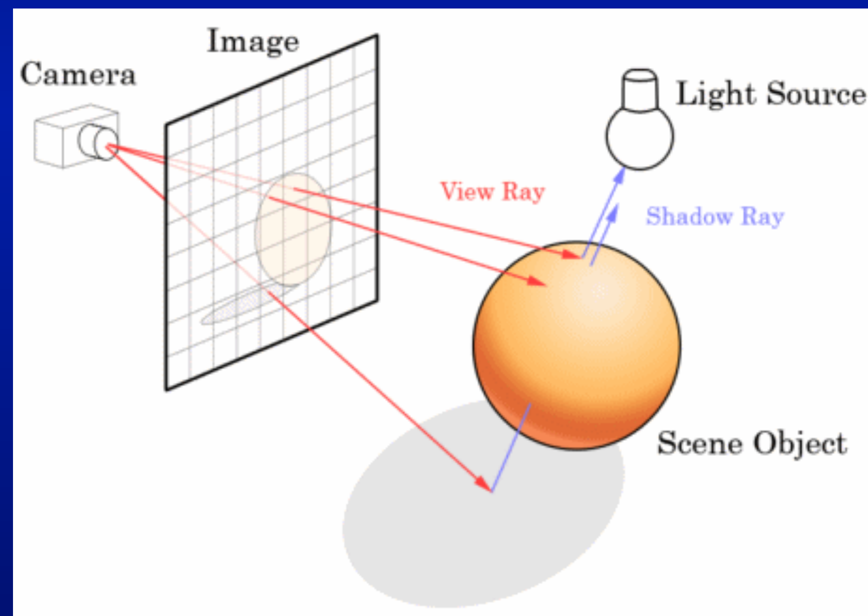


Zero DP!

TACC

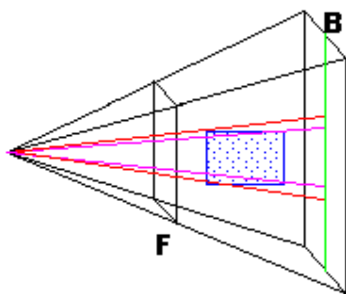
GPU Accelerates Rendering

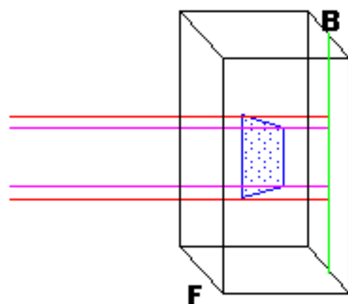
- Determining the color to be assigned to each pixel in an image by simulating the transport of light in a synthetic scene.

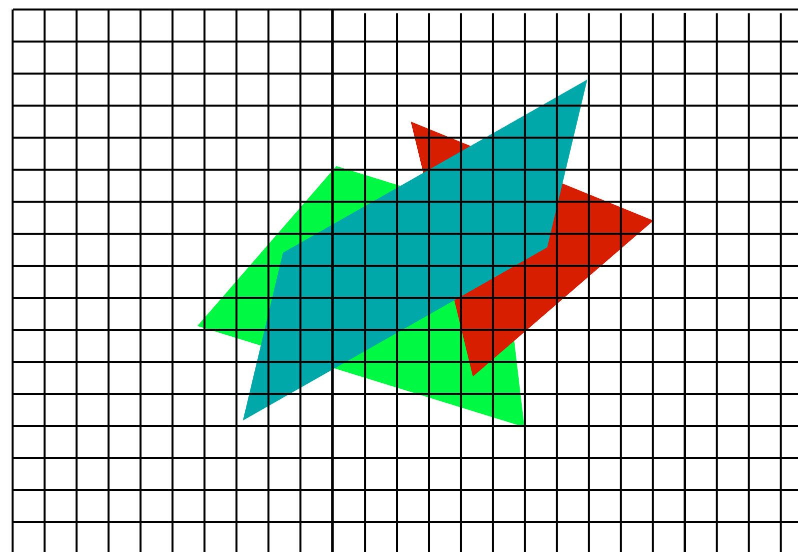


The Key Efficiency Trick

- Transform into perspective space, densely sample, and produce a large number of independent SIMD computations for shading

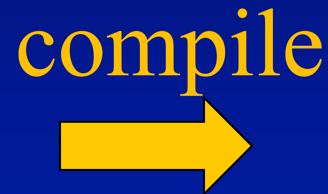






Shading a Fragment

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;
float4 diffuseShader(float3 norm, float2 uv)
{
        float3 kd;
        kd = myTex.Sample(mySamp, uv);
        kd *= clamp(dot(lightDir, norm), 0.0, 1.0);
        return float4(kd, 1.0);
}
```

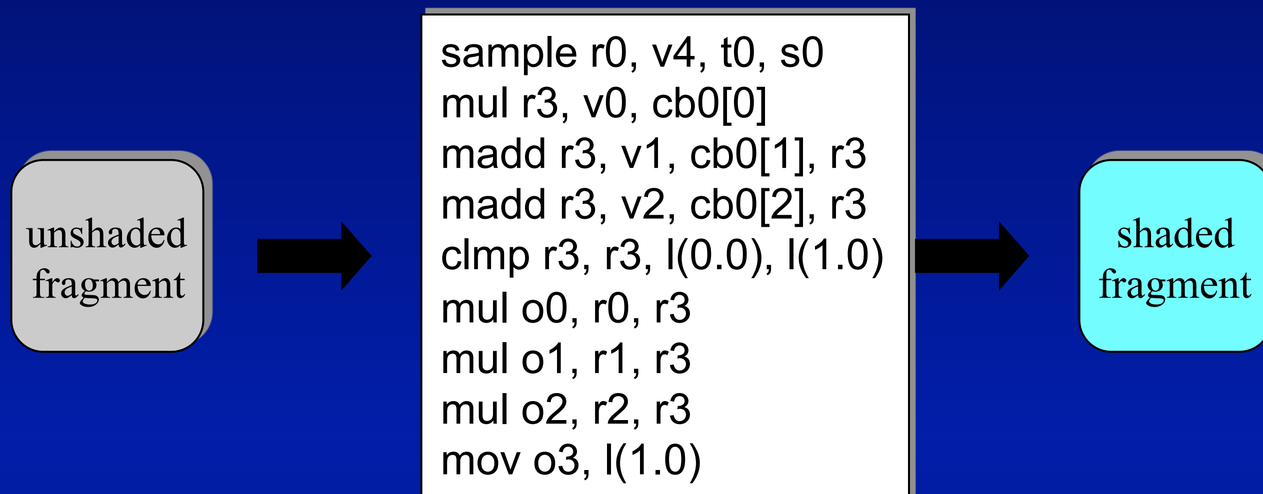



```
sample r0, v4, t0, s0
mul r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul o0, r0, r3
mul o1, r1, r3
mul o2, r2, r3
mov o3, l(1.0)
```

- Simple Lambertian shading of texture-mapped fragment.
- Sequential code
- Performed in parallel on many independent fragments
- How many is “many”?

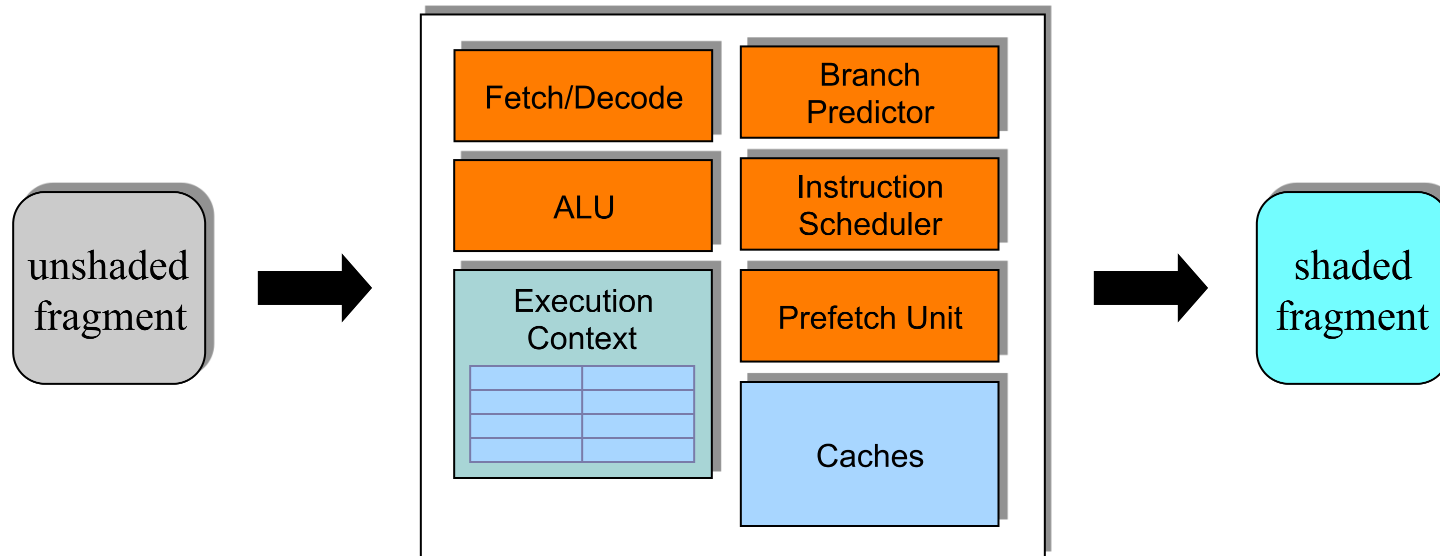
At least hundreds of thousands per frame

Work per Fragment



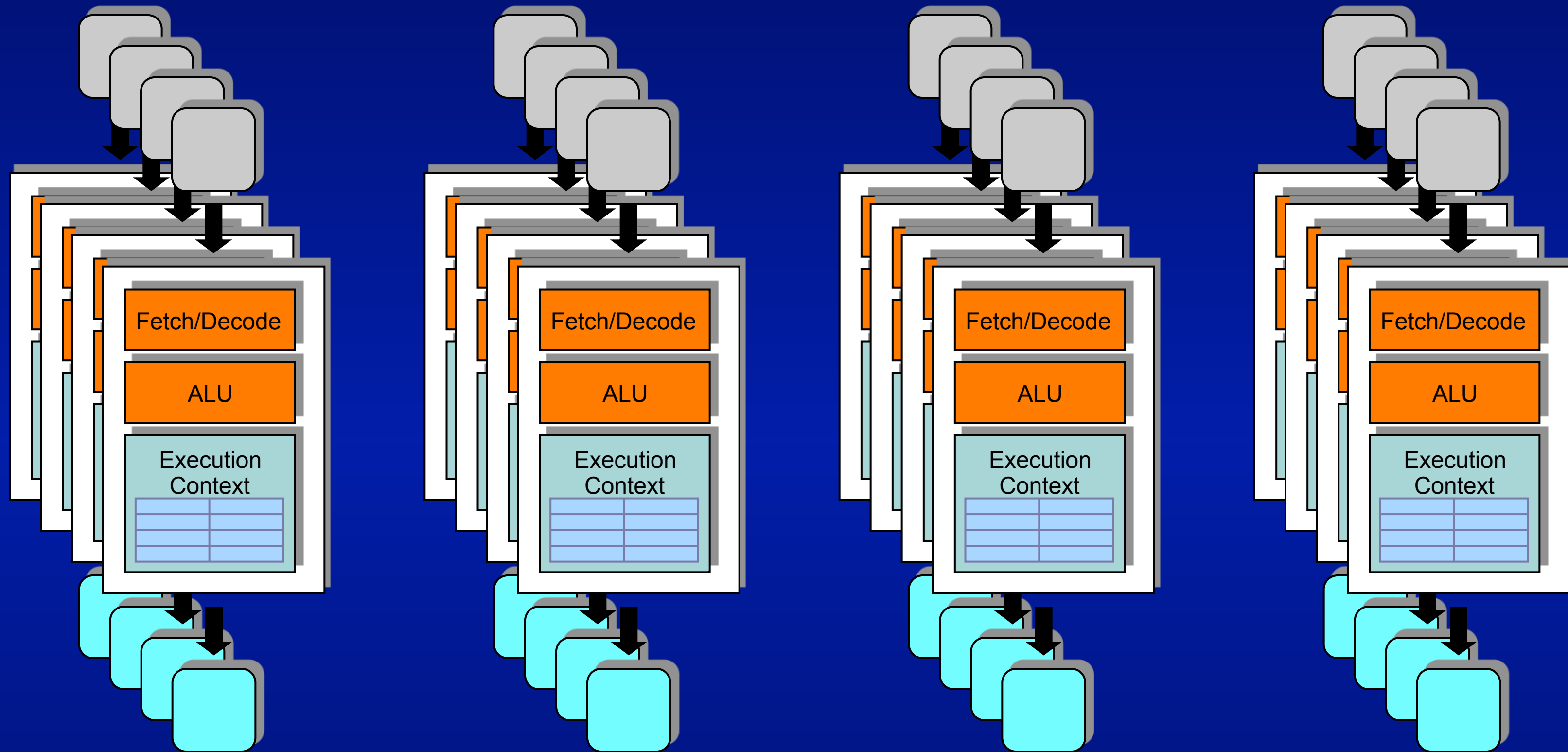
- Do a a couple hundred thousand of these @ 60 Hz or so
- How?
- We have independent threads to execute, so use multiple cores
- What kind of cores?

The CPU Way



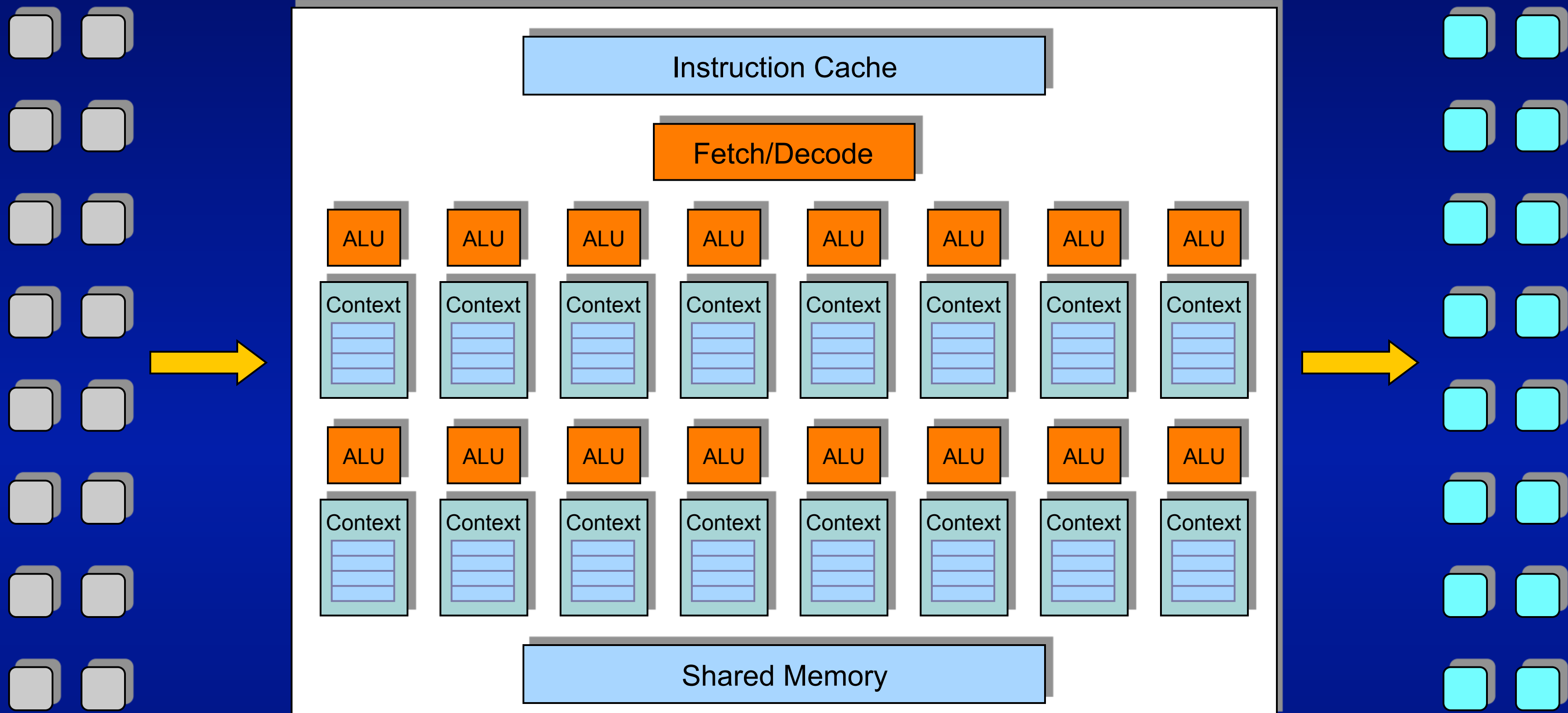
- Big, complex, but fast on a single thread
- However, each program is very short, so do not need this much complexity
- Must complete many many short programs quickly

Simplify and Parallelize



- Don't use a few CPU style cores
- Use simpler ones and many more of them.

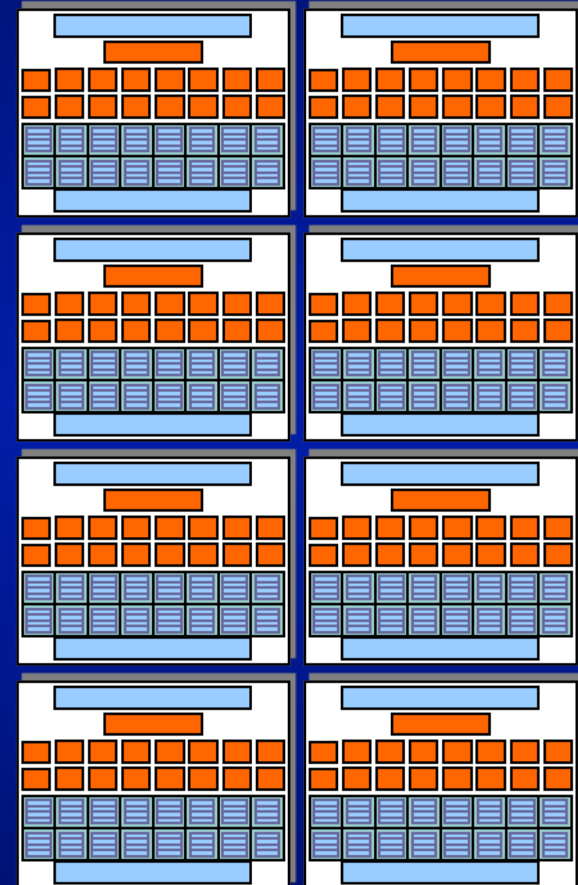
Shared Instructions



- Applying same instructions to different data… the definition of SIMD!
- Thus SIMD – amortize instruction handling over multiple ALUs

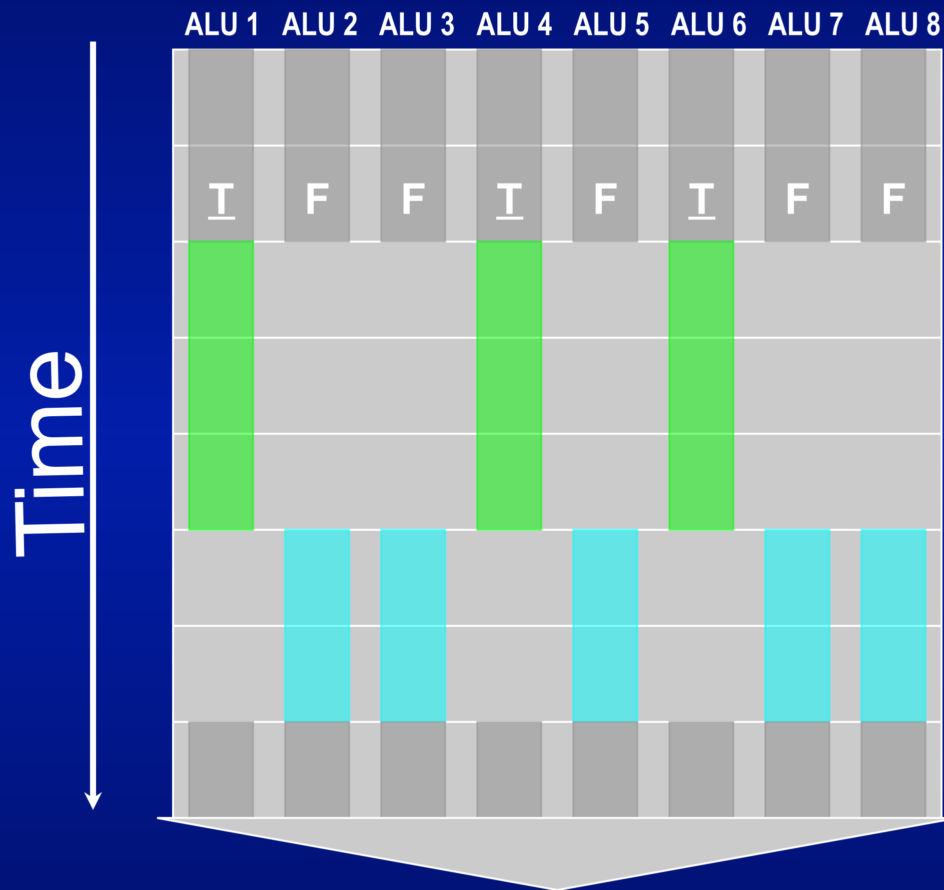
But What about the Other Processing?

- A graphics pipeline does more than shading. Other ops are done in parallel, like transforming vertices.
  So need to execute more than one program in the system simultaneously.
- If we replicate these SIMD processors, we now have the ability to do different SIMD computations in parallel in different parts of the machine.
- In this example, we can have 128 threads in parallel, but only 8 different programs simultaneously running



TACC

What about Branches?



GPUs use predication!

<unconditional shader code>

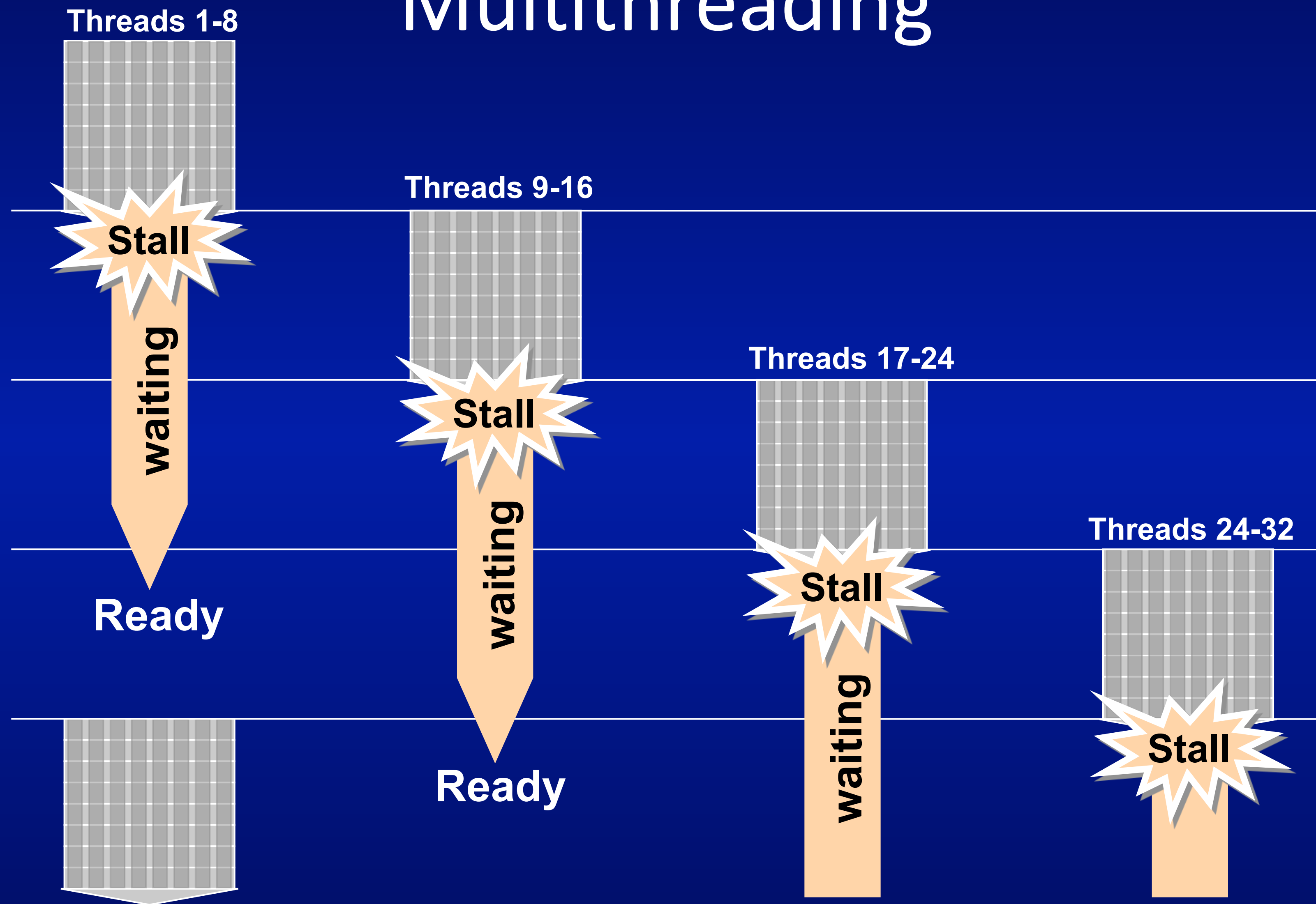
```
if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}
```

<unconditional shader code>

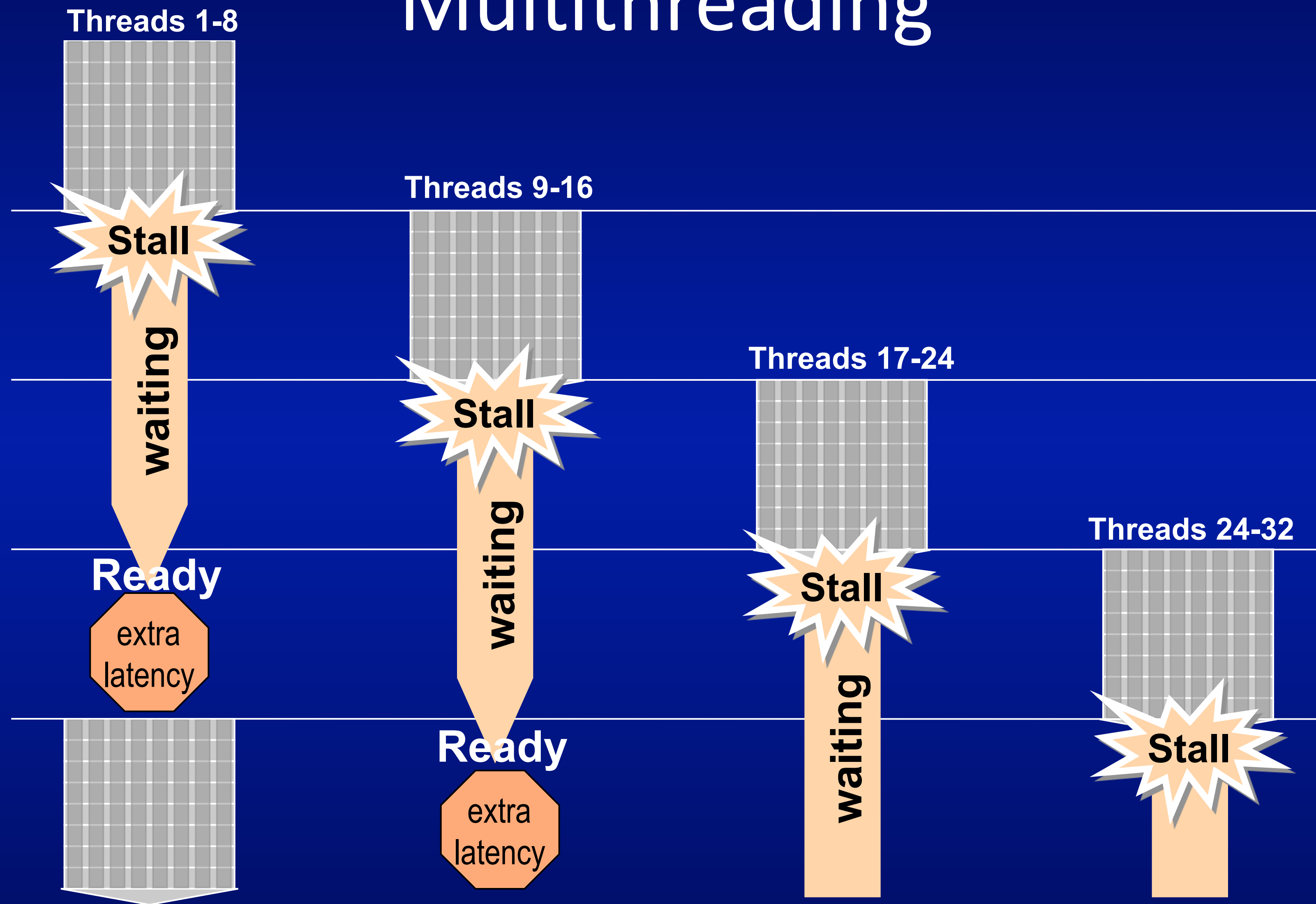
Efficiency - Dealing with Stalls

- A thread is stalled when its next instruction to be executed must await a result from a previous instruction.
  - Pipeline dependencies
  - Memory latency
- The complex CPU hardware (omitted from these machines) was effective at dealing with stalls.
- What will we do instead?
- Since we expect to have lots more threads than processors, we can interleave their execution to keep the hardware busy when a thread stalls.
- Multithreading!

TACC

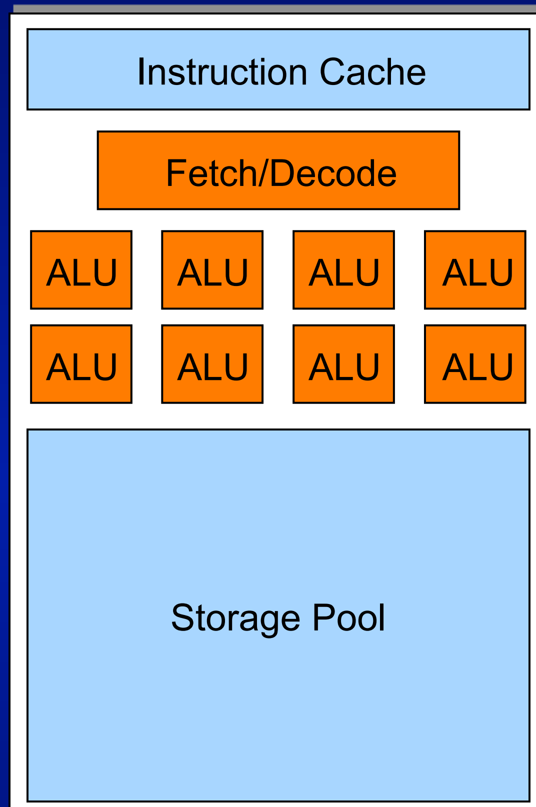

Multithreading
Threads 1-8
Stall
waiting
Ready
Threads 9-16
Stall
waiting
Ready
Threads 17-24
Stall
waiting
Threads 24-32
Stall
TACC

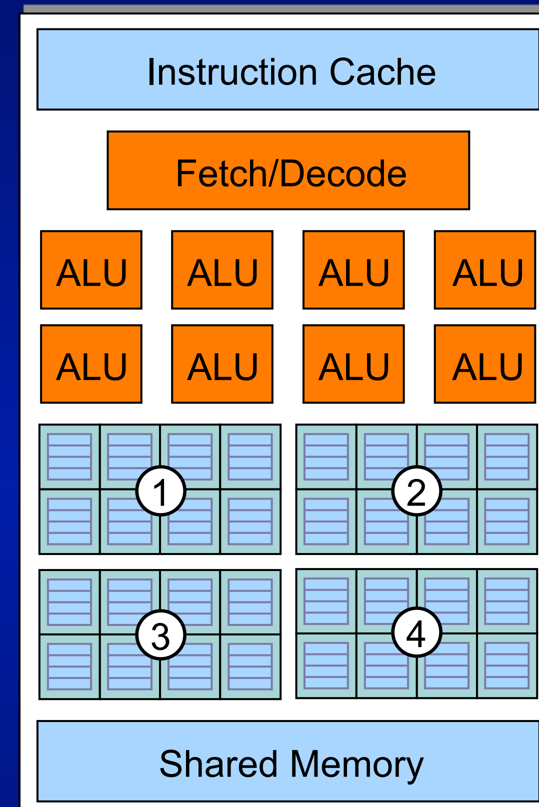
Multithreading



TACC

Costs of Multithreading





- Adds latency to individual threads
  in order to minimize time to complete all threads.
- Requires extra context storage. More contexts can mask more latency.

TACC

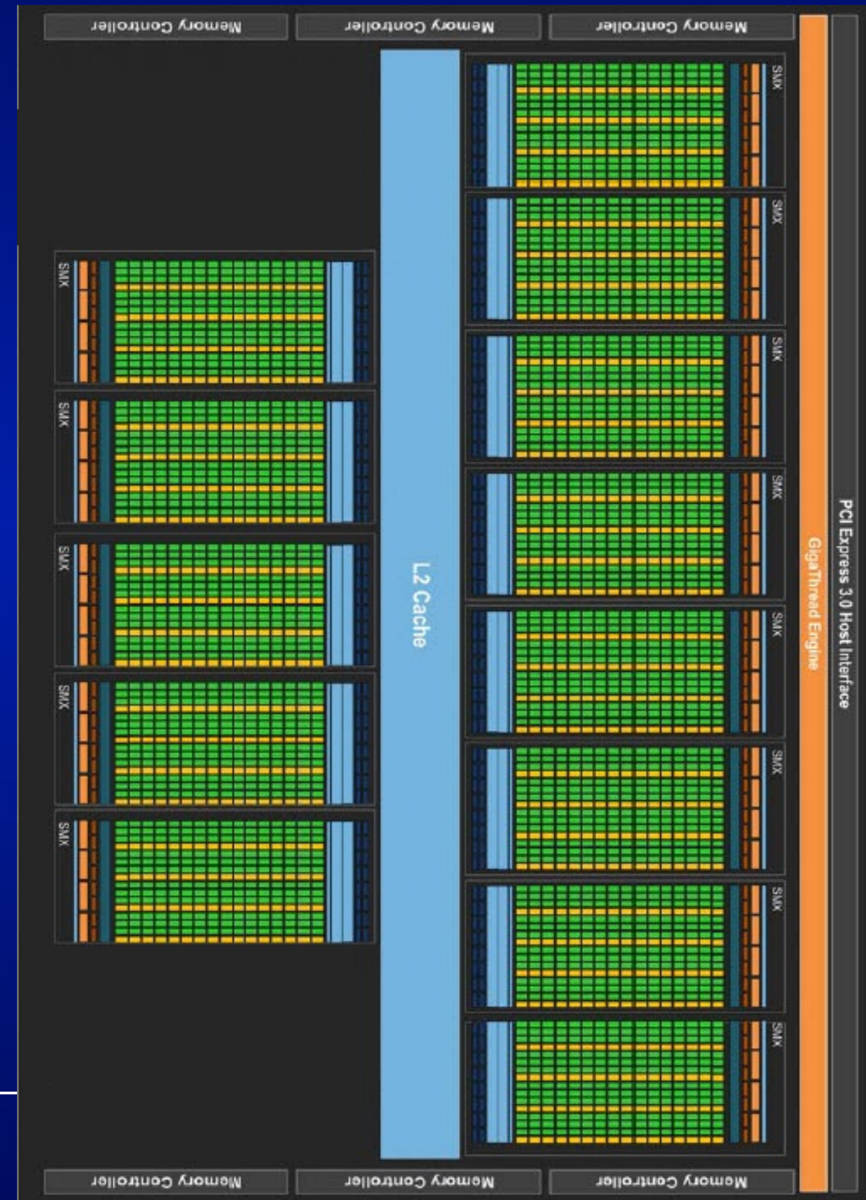
Example System

32 cores x 16 ALUs/core = 512 (madd) ALUs @ 1 GHz = 1 Teraflop

TACC

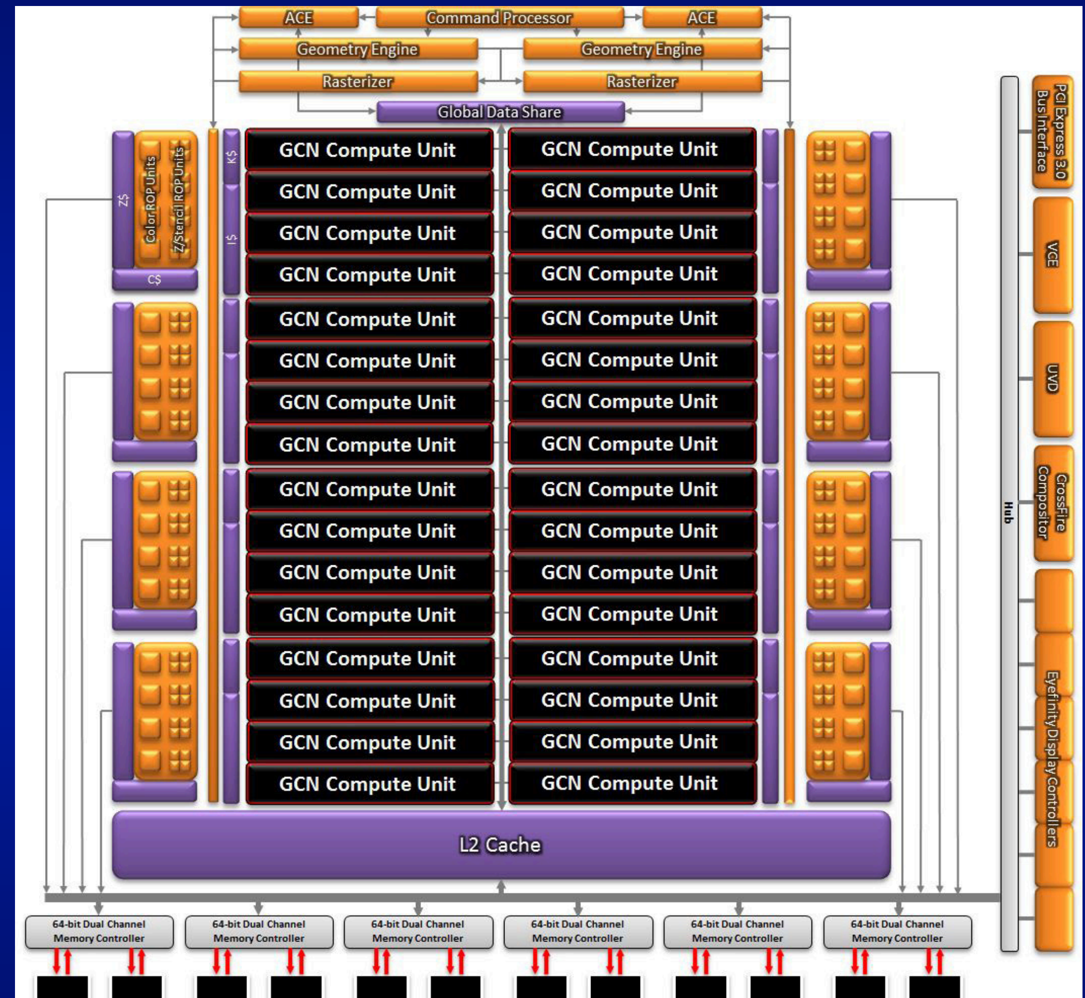
Real Example – NVIDIA Tesla K20

- 13 Cores ("Streaming Multiprocessors (SMX)")
- 192 SIMD Functional Units per Core ("CUDA Cores")
- Each FU has 1 fused multiply-add (SP and DP)
- Peak 2496 SP floating point ops per clock
- 4 warp schedulers and 8 instruction dispatch units
  - Up to 32 threads concurrently executing (called a "WARP")
  - Coarse-grained: Up to 64 WARPS interleaved per core to mask latency to memory



Real Example - AMD Radeon HD 7970

- 32 Compute Units ("Graphics Cores Next (GCN)" processors)
- 4 Cores per FU ("Stream Cores")
  - 16-wide SIMD per Stream Core
- 1 Fused Multiply-Add per ALU
- Peak 4096 SP ops per clock
- 2 level multithreading
  - Fine-grained: 8 threads interleaved into pipelined CU GCN Cores
  - Up to 256 concurrent threads (called a "Wavefront")
  - Coarse-grained: groups of about 40 wavefronts interleaved to mask memory latency
  - Up to 81,920 concurrent items



TACC

Mapping Marketing Terminology to Engineering Details

	x86	NVIDIA	AMD/ATI
Functional Unit	Core	Streaming Multiprocessor (SM)	SIMD Engines / Processor
SIMD lane		CUDA core	Stream core
Simultaneously-processed SIMD (Concurrent "threads")		Warp	Wavefront
Functional Unit instruction stream	Thread	Kernel	Kernel

TACC

Memory Architecture

- CPU style
  - Multiple levels of cache on chip
  - Takes advantage of temporal and spatial locality to reduce demand on remote slow DRAM
  - Caches provide local high bandwidth to cores on chip
  - 25GB/sec to main memory

- GPU style
  - Local execution contexts (64KB)
and a similar amount of local memory
  - Read-only texture cache
  - Traditionally no cache hierarchy
(but see NVIDIA Fermi and Intel MIC)
  - Much higher bandwidth to main memory, 150—200 GB/sec

Performance Implications of GPU Bandwidth

- GPU memory system is designed for throughput
  - Wide Bus (150 – 200 GB/sec) and high bandwidth DRAM organization (GDDR3-5)
  - Careful scheduling of memory requests to make efficient use of available bandwidth (recent architectures help with this)
- An NVIDIA Tesla K20 GPU in Stampede has 13 SMXs with 192 SIMD lanes and a 0.706 GHz clock.
  - How many peak single-precision FLOPs?
    - 3524 GFLOPs
  - Memory bandwidth is 208 GB/s. How many FLOPs per byte transferred must be performed for peak efficiency?
    - ~17 FLOPs per byte

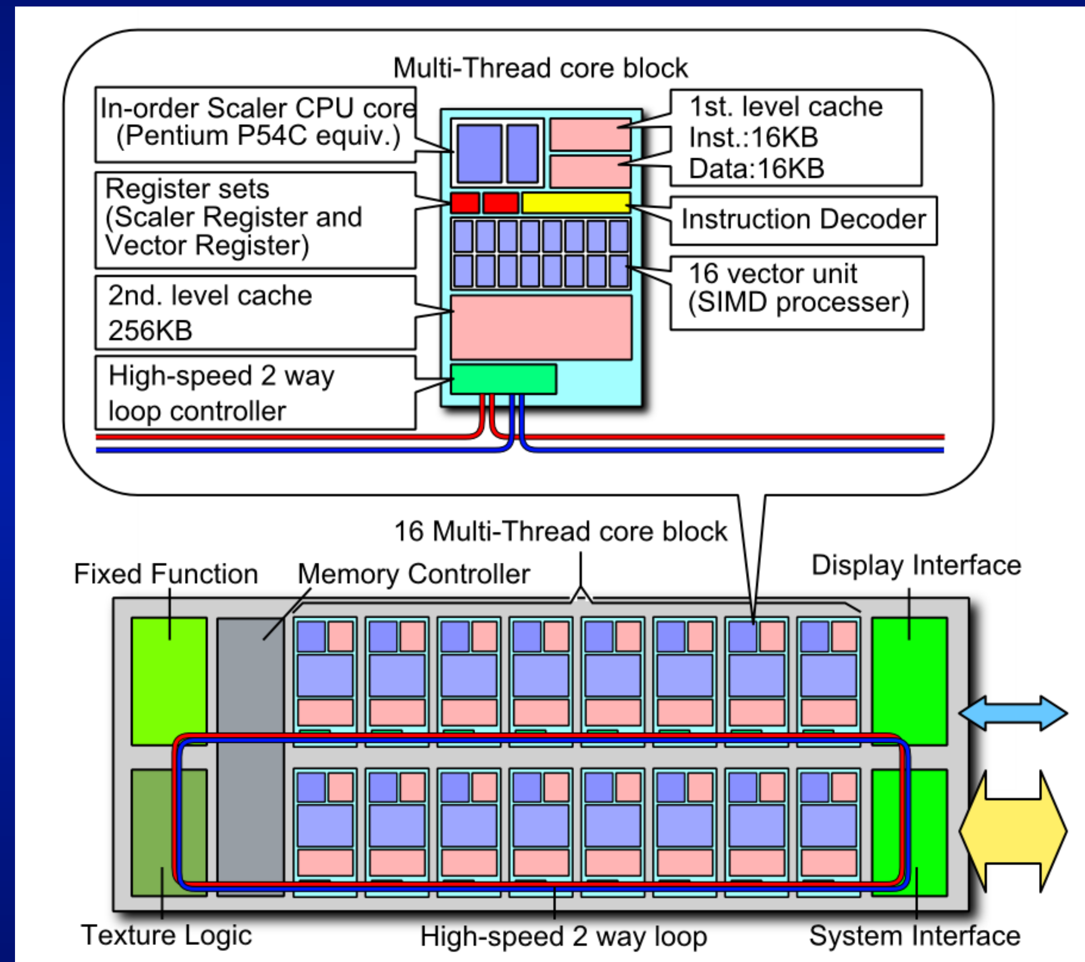
TACC

Performance Implications of GPU Bandwidth

- An AMD Radeon 7970 GHz has 32 Compute Units with 64 Stream Cores each and a 0.925 GHz clock.
  - How many peak single-precision FLOPs?
    - 3789 GFLOPs
  - Memory bandwidth is 264GB/s. How many FLOPs per byte transferred must be performed for peak efficiency?
    - ~14 FLOPs per byte
- AMD new GCN technology has closed the bandwidth gap
- Compute performance will likely continue to outpace memory bandwidth performance

"Real" Example - Intel MIC "co-processor"

- Many Integrated Cores: originally Larrabee, now Knights Ferry (dev), Knights Corner (prod)
- 32 cores (Knights Ferry) >50 cores (Knights Corner)
- Explicit 16-wide vector ISA (16-wide madder unit)
- Peak 1024 SP float ops per clock for 32 cores
- Each core interleaves four threads of x86 instructions
- Additional interleaving under software control
- Traditional x86 programming and threading model



What is it?

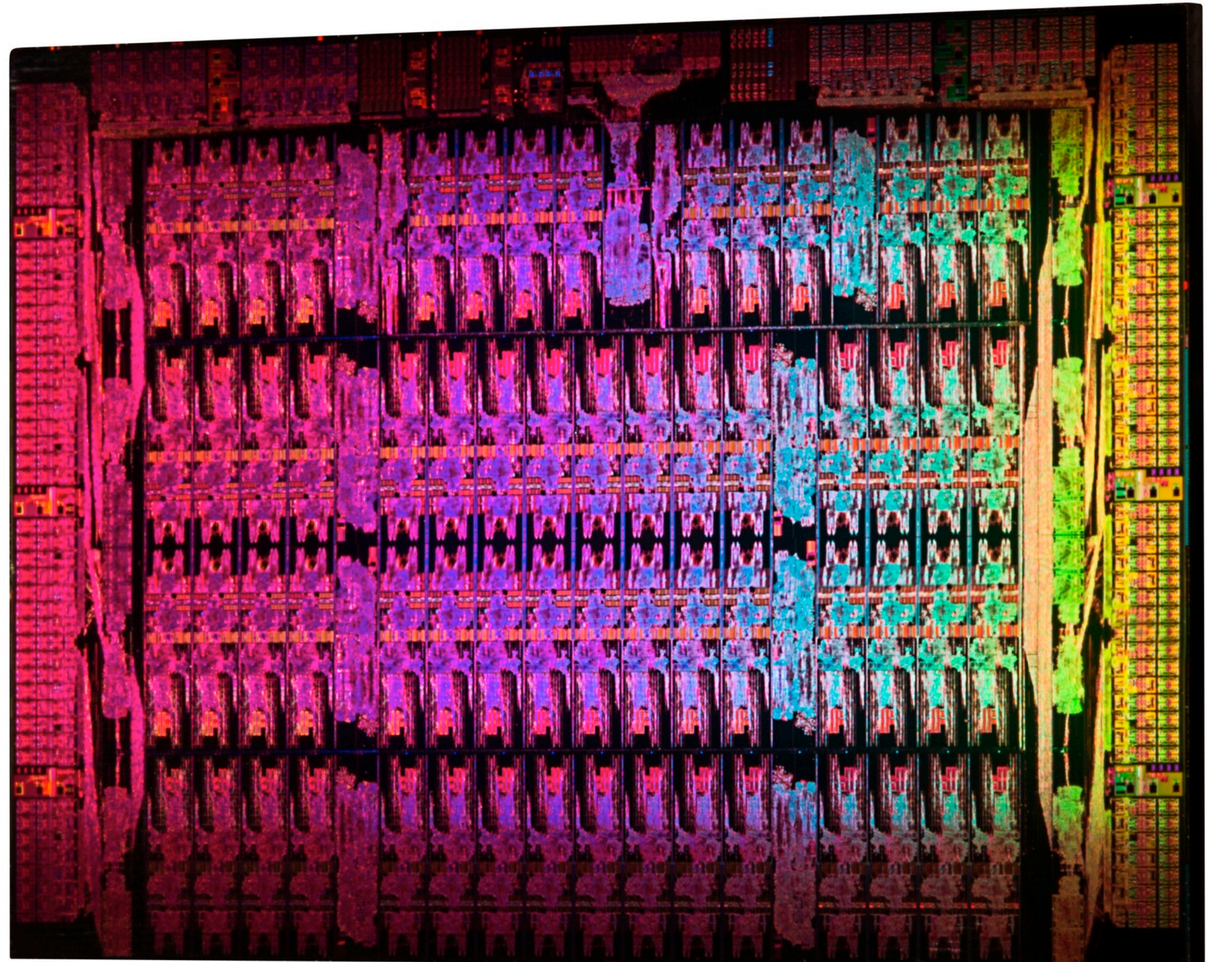
- Co-processor
  - PCI Express card
  - Stripped down Linux operating system
- Dense, simplified processor
  - Many power-hungry operations removed
  - Wider vector unit
  - Wider hardware thread count
- Lots of names
  - Many Integrated Core architecture, aka MIC
  - Knights Corner (code name)
  - Intel Xeon Phi Co-processor SE10P (product name)

What is it?

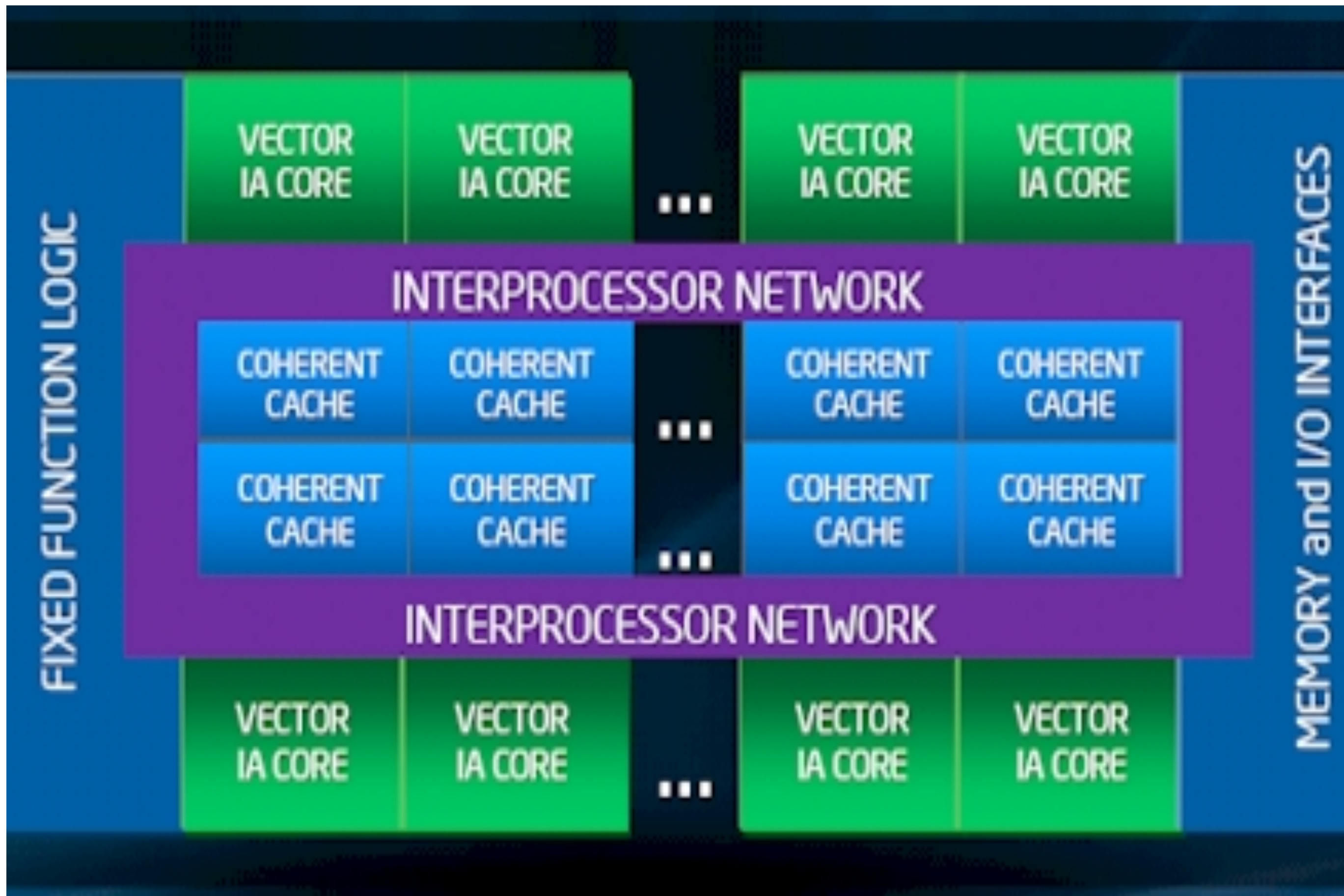
- Leverage x86 architecture (CPU with many cores)
  - x86 cores that are simpler, but allow for more compute throughput
- Leverage existing x86 programming models
- Dedicate much of the silicon to floating point ops
- Cache coherent
- Increase floating-point throughput
- Strip expensive features
  - out-of-order execution
  - branch prediction
- Widen SIMD registers for more throughput
- Fast (GDDR5) memory on card

Intel Xeon Phi Chip

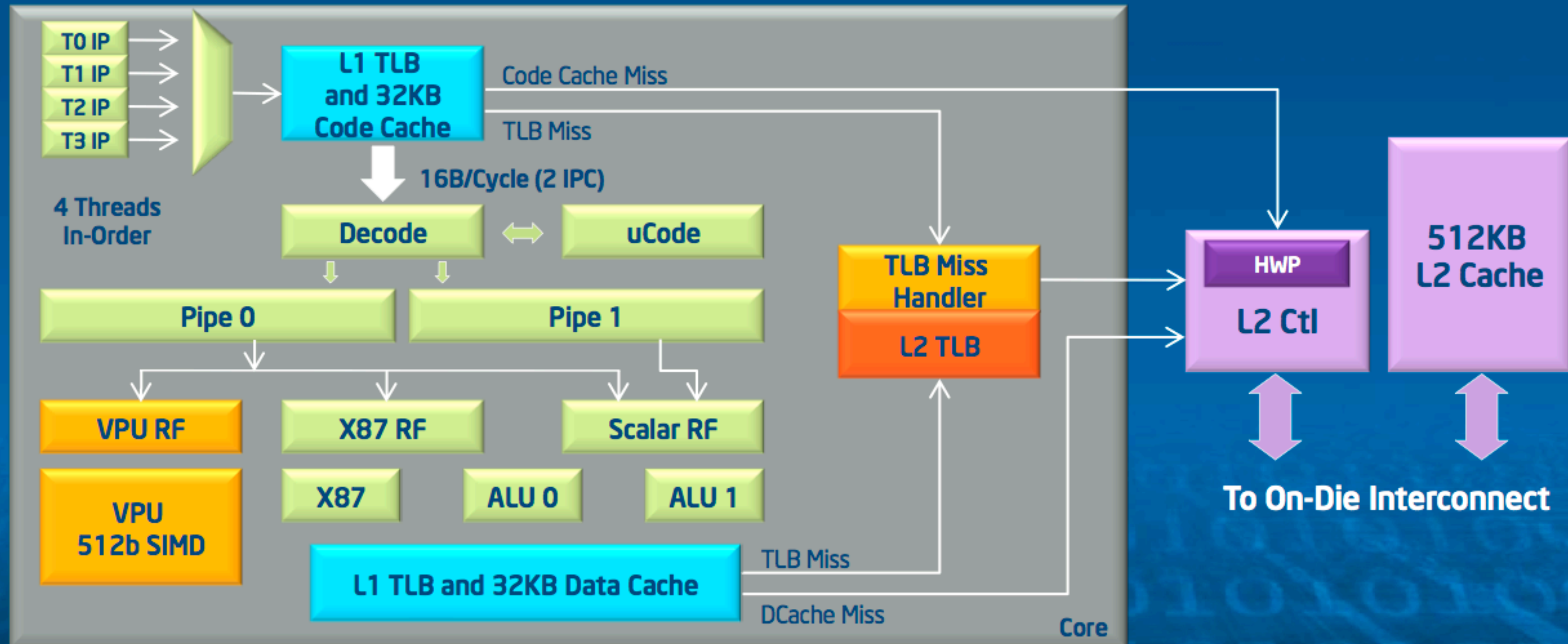
- 22 nm process
- Based on what Intel learned from
  - Larrabee
  - SCC
  - TeraFlops Research Chip



MIC Architecture



- Many cores on the die
- L1 and L2 cache
- Bidirectional ring network for L2
- Memory and PCIe connection



George Chrysos, Intel, Hot Chips 24 (2012):
http://www.slideshare.net/IntelXeon/under-the-armor-of-knights-corner-intel-mic-architecture-at-hotchips-2012

Vector Processing Unit

PPF	PF	D0	D1	D2	E	WB			
				D2	E	VC1	VC2	V1-V4	WB

D2	E	VC1	VC2	V1	V2	V3	V4

DEC

VPU
RF
3R, 1W

LD

EMU

ST

Mask
RF

Scatter
Gather

Vector ALUs

16 Wide x 32 bit
8 Wide x 64 bit

Fused Multiply Add

arallel Computing Group

Copyright © 2012 Intel Corporation. All rights reserved.

George Chrysos, Intel, Hot Chips 24 (2012):
http://www.slideshare.net/IntelXeon/under-the-armor-of-knights-corner-intel-mic-architecture-at-hotchips-2012

Speeds and Feeds

- Processor
  - ~1.1 GHz
  - 61 cores
  - 512-bit wide vector unit
  - 1.074 TF peak DP
- Data Cache
  - L1 32KB/core
  - L2 512KB/core, 30.5 MB/chip
- Memory
  - 8GB GDDR5 DRAM
  - 5.5 GT/s, 512-bit*
- PCIe
  - 5.0 GT/s, 16-bit

Advantages

- Intel's MIC is based on x86 technology
  - x86 cores w/ caches and cache coherency
  - SIMD instruction set
- Programming for MIC is similar to programming for CPUs
  - Familiar languages: C/C++ and Fortran
  - Familiar parallel programming models: OpenMP & MPI
  - MPI on host and on the coprocessor
  - Any code can run on MIC, not just kernels
- Optimizing for MIC is similar to optimizing for CPUs
  - **"Optimize once, run anywhere"**
  - Our early MIC porting efforts for codes "in the field" are frequently doubling performance on Sandy Bridge.

Stampede Programming Models

- Traditional Cluster
  - Pure MPI and MPI+X
    - X: OpenMP, TBB, Cilk+, OpenCL, …
- Native Phi
  - Use one Phi and run OpenMP or MPI programs directly
- MPI tasks on Host and Phi
  - Treat the Phi (mostly) like another host
    - Pure MPI and MPI+X
- MPI on Host, Offload to Xeon Phi
  - Targeted offload through OpenMP extensions
  - Automatically offload some library routines with MKL

Traditional Cluster

- Stampede is 2+ PF of FDR-connected Xeon E5
  - High bandwidth: 56 Gb/s (sustaining >52 Gb/s)
  - Low-latency
    - ~1 μs on leaf switch
    - ~2.5 μs across the system
- Highly scalable for existing MPI codes
- IB multicast and collective offloads for improved collective performance

Native Execution

- Build for Phi with –mmic
- Execute on host
- … or ssh to mic0 and run on the Phi
- Can safely use all 61 cores
  - Offload programs should stay away from the 61st core since the offload daemon runs here

Symmetric MPI

- Host and Phi can operate symmetrically as MPI targets
  - High code reuse
  - MPI and hybrid MPI+X
- Careful to balance workload between big cores and little cores
- Careful to create locality between local host, local Phi, remote hosts, and remote Phis
- Take advantage of topology-aware MPI interface under development in MVAPICH2
  - NSF STCI project with OSU, TACC, and SDSC

Symmetric MPI

- Typical 1-2 GB per task on the host
- Targeting 1-10 MPI tasks per Phi on Stampede
  - With 6+ threads per MPI task

MPI with Offload to Phi

- Existing codes using accelerators have already identified regions where offload works well
- Porting these to OpenMP offload should be straightforward
- Automatic offload where MKL kernel routines can be used
  - xGEMM, etc.

What we at TACC like about Phi

- Intel's MIC is based on x86 technology
  - x86 cores w/ caches and cache coherency
  - SIMD instruction set
- Programming for Phi is similar to programming for CPUs
  - Familiar languages: C/C++ and Fortran
  - Familiar parallel programming models: OpenMP & MPI
  - MPI on host and on the coprocessor
  - Any code can run on MIC, not just kernels
- Optimizing for Phiis similar to optimizing for CPUs
  - **"Optimize once, run anywhere"**
  - Our early Phi porting efforts for codes "in the field" have doubled performance on Sandy Bridge.

Will My Code Run on Xeon Phi?

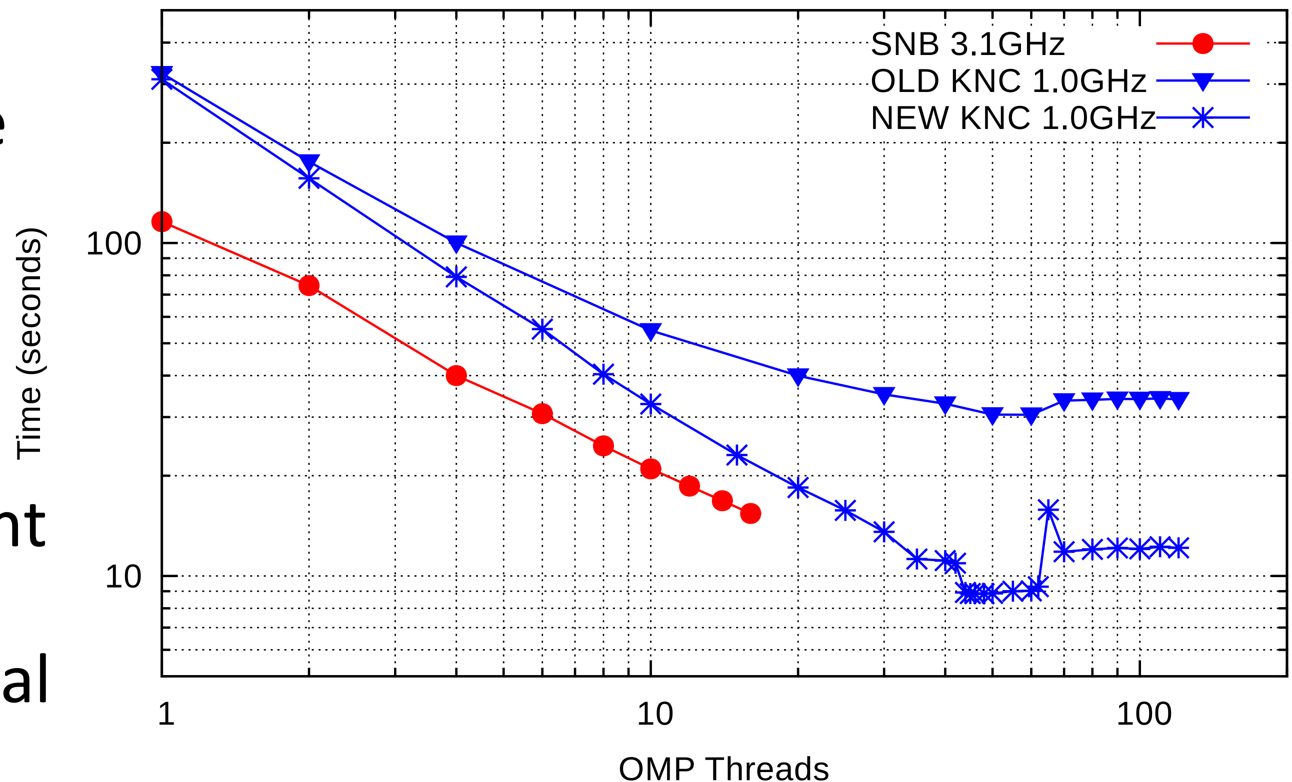
- Yes
- … but that's the wrong question
  - Will your code run *best* on Phi?, or
  - Will you get great Phi performance without additional work?

Early Phi Programming Experiences at TACC

- Codes port easily
  - Minutes to days depending mostly on library dependencies
- Performance can require real work
  - While the software environment continues to evolve
  - Getting codes to run *at all* is almost too easy; really need to put in the effort to get what you expect
- Scalability is pretty good
  - Multiple threads per core is really important
  - Getting your code to vectorize is really important

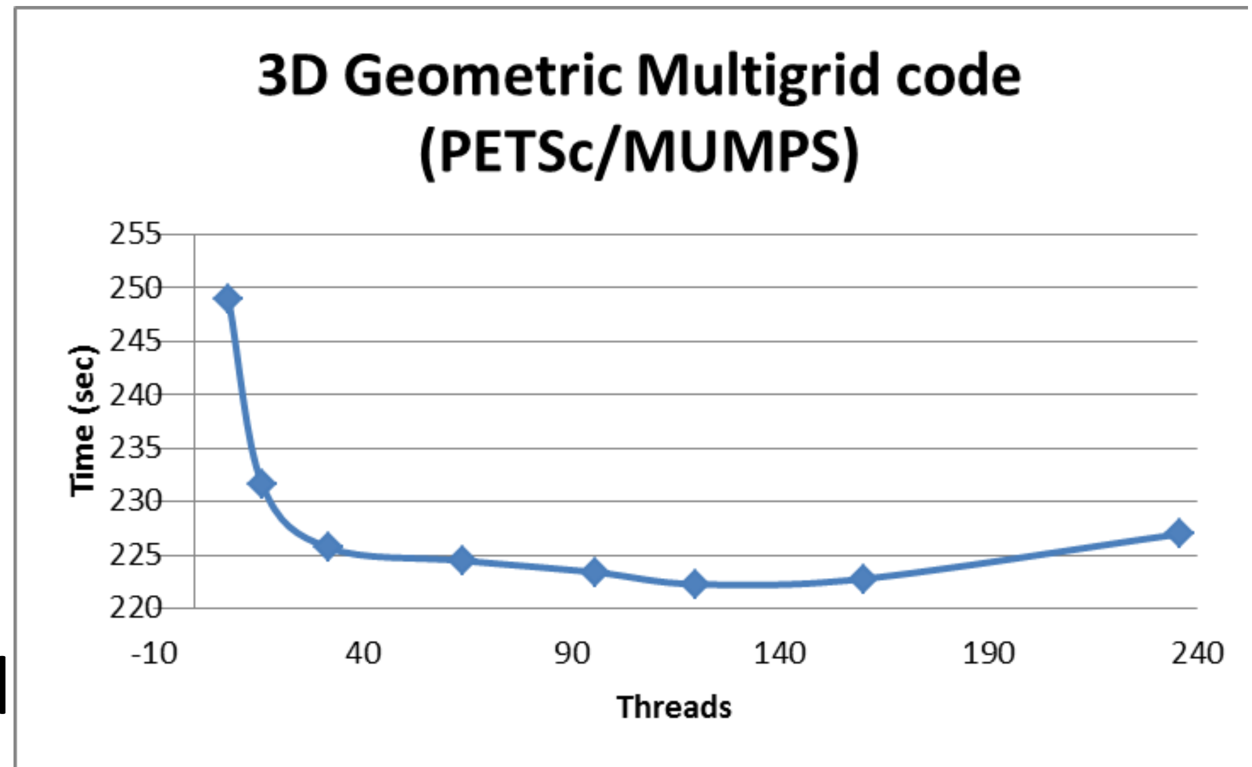
LBM Example

- Lattice Boltzmann Method CFD code
  - Carlos Rosales, TACC
  - MPI code with OpenMP
- Finding all the right routines to parallelize is critical



PETSc/MUMPS with AO

- Hydrostatic ice sheet modeling
- MUMPS solver (called through PETSC)
- BLAS calls automatically offloaded behind the scenes



Publishing Results

- Published results about Xeon Phis should include
  - Silicon stepping (B0 or B1)
  - Software stack version (Gold)
  - Intel Product SKU
    (Intel Xeon Phi Co-processor SE10P)
  - Core count: 61
  - Clock rate (1.09 GHz or 1.1 GHz)
  - DRAM Rate (5.6 GT/s)