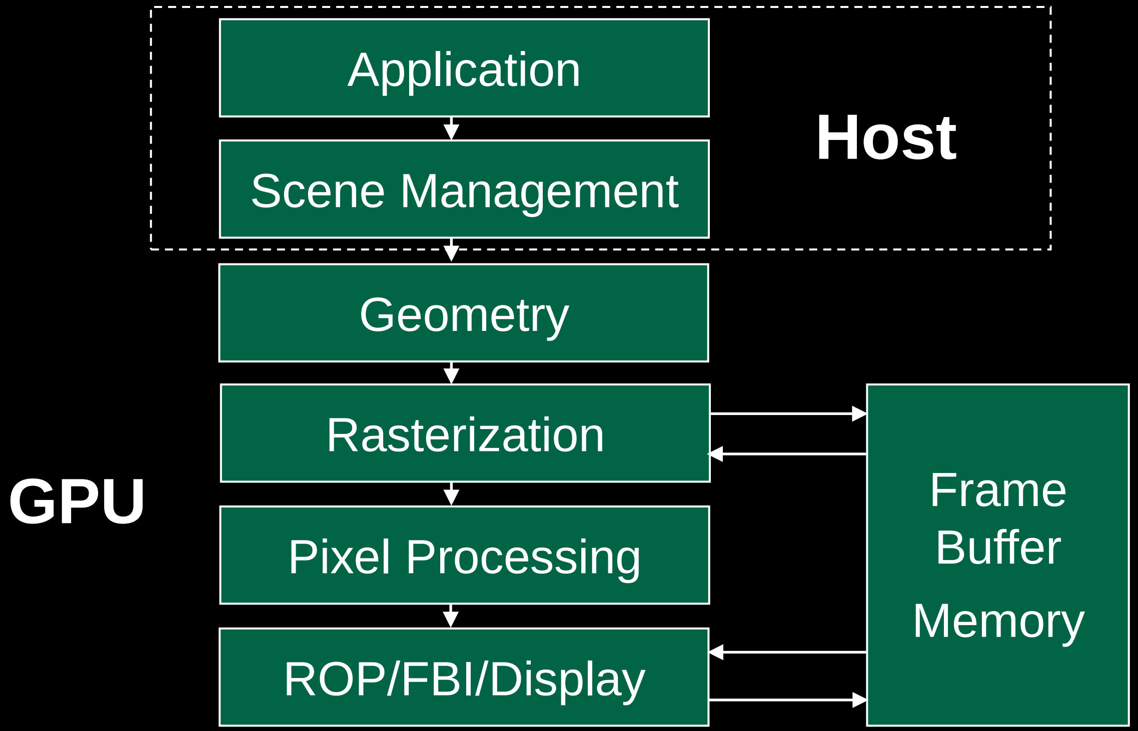
The Evolution of GPU Computing

Steve Scott
CTO Tesla, NVIDIA
SC12, Salt Lake City

Early 3D Graphics Pipeline



Moving Toward Programmability

- DirectX 5 — Riva 128
- 1998
- DirectX 6 — Multitexturing — Riva TNT
- 1999
- DirectX 7 — T&L TextureStageStage — GeForce 256
- 2000
- 2001
- DirectX 8 — SM 1.x — GeForce 3
- 2002
- Cg
- 2003
- DirectX 9 — SM 2.0 — GeForceFX
- DirectX 9.0c — SM 3.0 — GeForce 6
- 2004


Half-Life


Quake 3


Giants


Halo

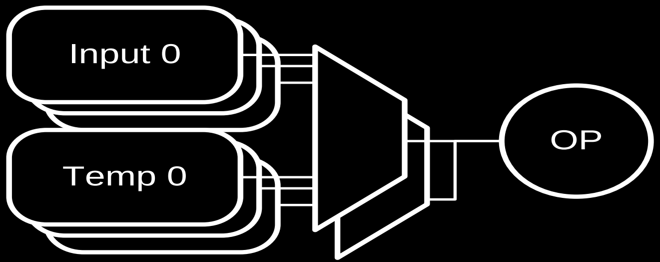

Far Cry


UE3

No Lighting | **Per-Vertex Lighting** | **Per-Pixel Lighting**

Programmable Shaders: GeForceFX (2002)

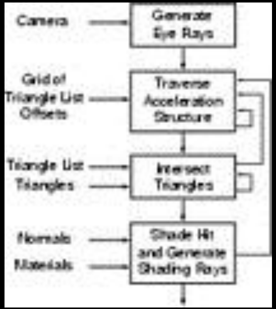
- Vertex and fragment operations specified in small (macro) assembly language (separate processors)
- User-specified mapping of input data to operations
- Limited ability to use intermediate computed values to index input data (textures and vertex uniforms)




```
ADDR R0.xyz, eyePosition.xyzx, -f[TEX0].xyzx;
DP3R R0.w, R0.xyzx, R0.xyzx;
RSQR R0.w, R0.w;
MULR R0.xyz, R0.w, R0.xyzx;
ADDR R1.xyz, lightPosition.xyzx, -f[TEX0].xyzx;
DP3R R0.w, R1.xyzx, R1.xyzx;
RSQR R0.w, R0.w;
MADR R0.xyz, R0.w, R1.xyzx, R0.xyzx;
MULR R1.xyz, R0.w, R1.xyzx;
DP3R R0.w, R1.xyzx, f[TEX1].xyzx;
MAXR R0.w, R0.w, {0}.x;
```

The Pioneers: Early GPGPU (2002)

www.gpgpu.org



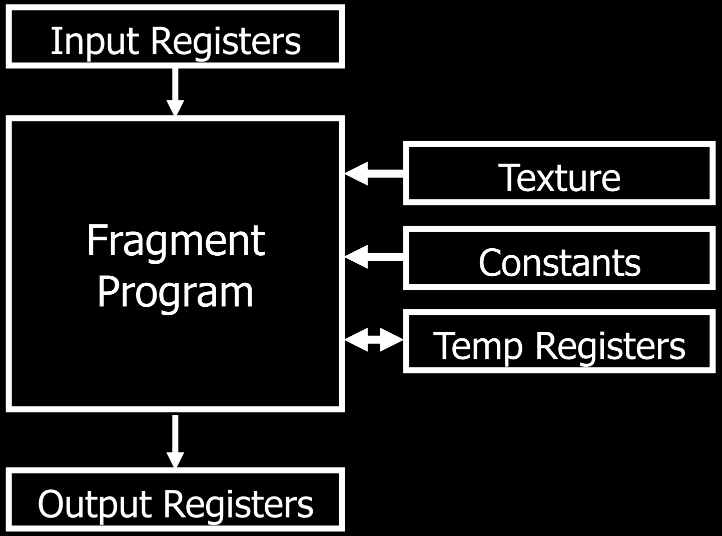
Early Raytracing

- **Ray Tracing on Programmable Graphics Hardware**, Purcell *et al.*
- **PDEs in Graphics Hardware**, Strzodka, Rumpf
- **Fast Matrix Multiplies using Graphics Hardware**, Larsen, McAllister
- **Using Modern Graphics Architectures for General-Purpose Computing: A Frameworks and Analysis**, Thompson *et al.*

This was not easy…

Challenges with Early GPGPU Programming

- HW challenges
  - Limited addressing modes
  - Limited communication: inter-pixel, scatter
  - Lack of integer & bit ops
  - No branching
- SW challenges
  - Graphics API (DirectX, OpenGL)
  - Very limited GPU computing ecosystem
  - Distinct vertex and fragment procs

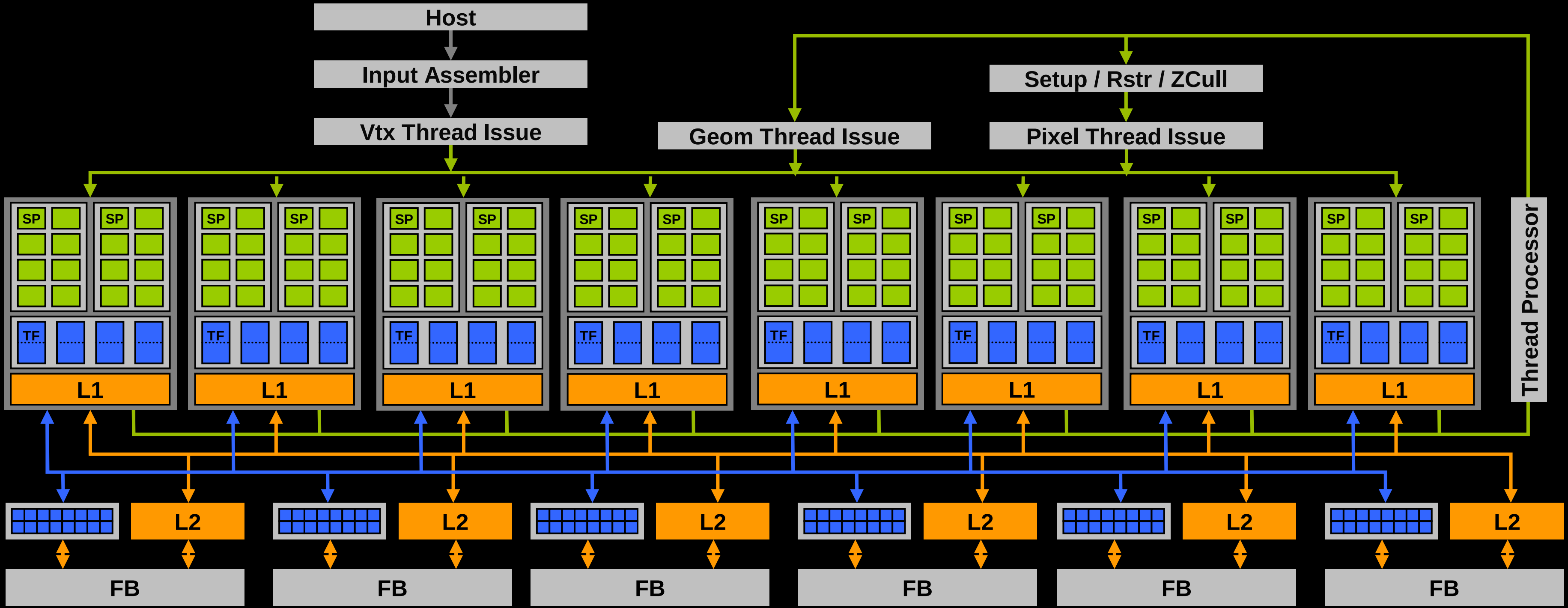


Software (DirectX, Open GL) and hardware slowly became more general purpose…

GeForce 8800 (G80)

Unified Compute Architecture

- Unified processor types
- Unified access to mem structures
- DirectX 10 & SM 4.0



CUDA C Programming

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

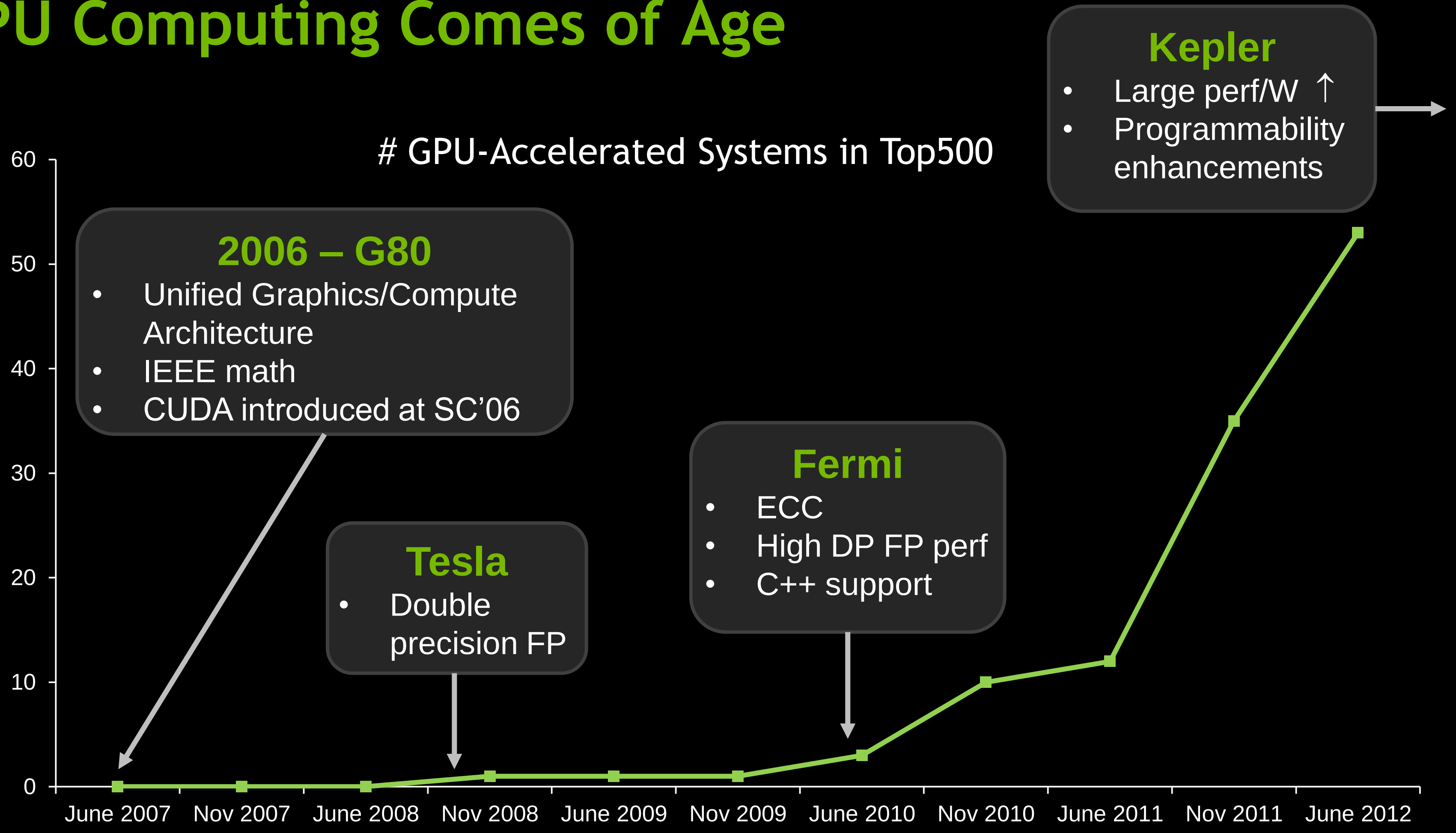
Serial C Code

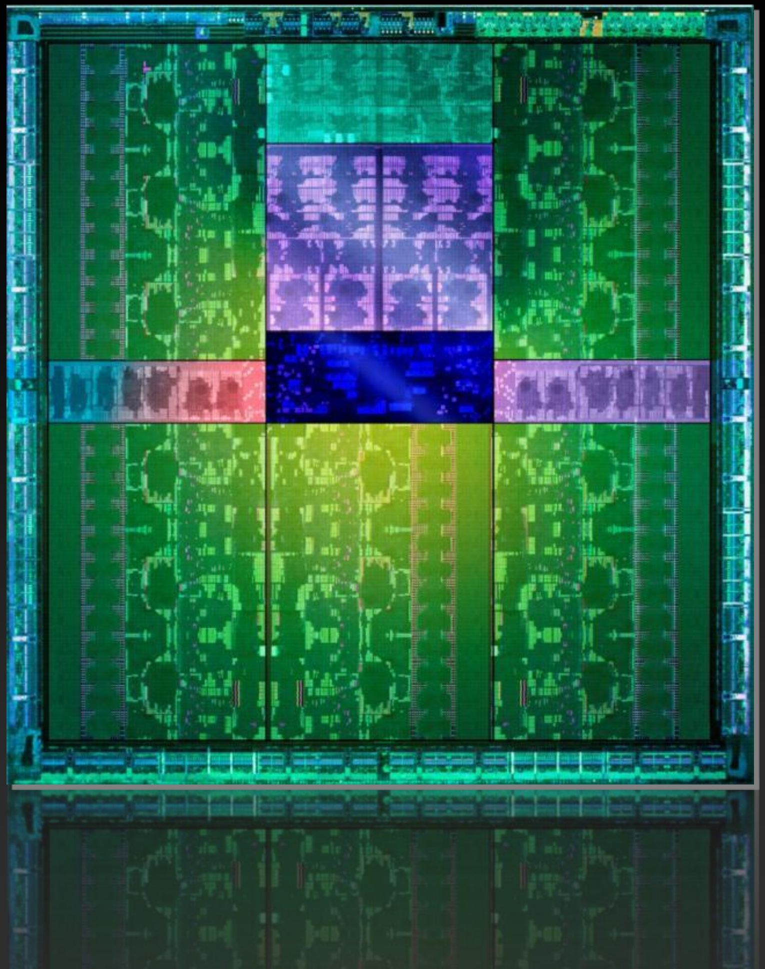
```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}

// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

GPU Computing Comes of Age





KEPLER

SMX *(3x power efficiency)*

Hyper-Q

Dynamic Parallelism

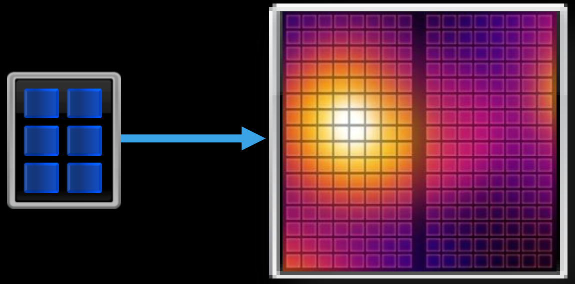
(programmability and application coverage)

Hyper-Q

Easier Speedup of Legacy MPI Apps

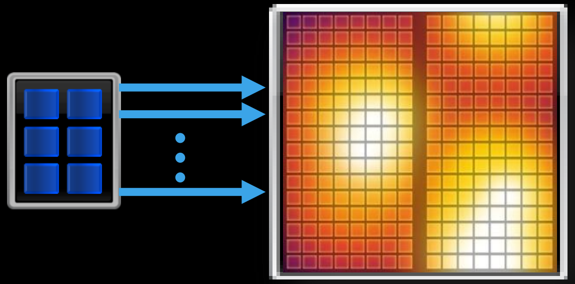
FERMI

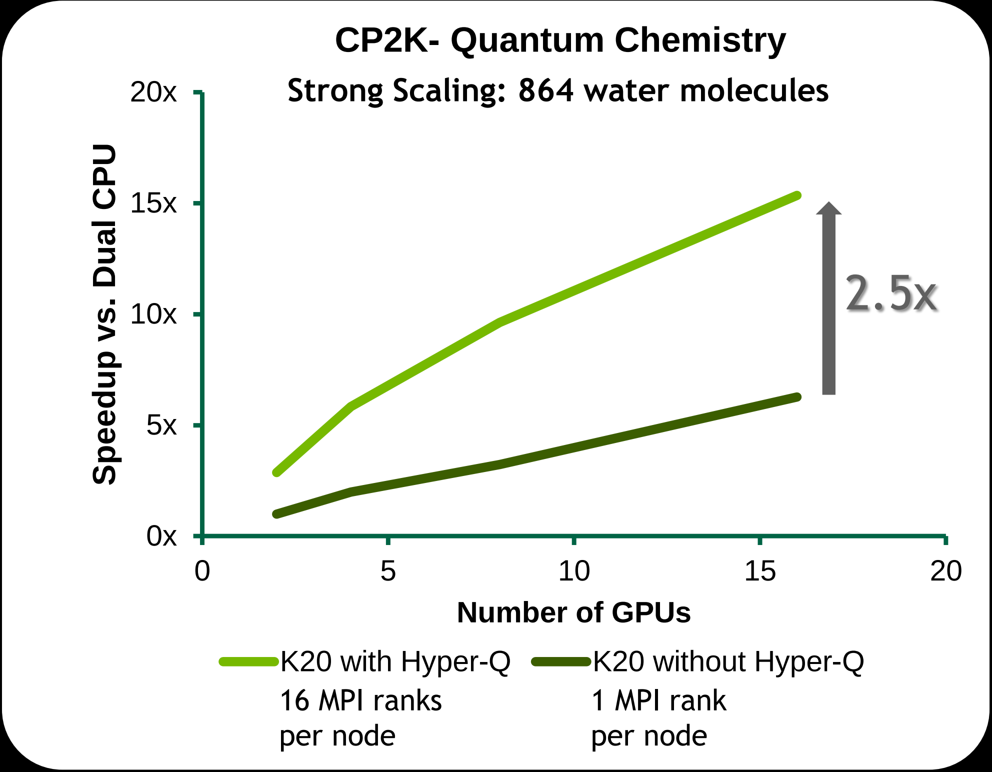
1 Work Queue



KEPLER

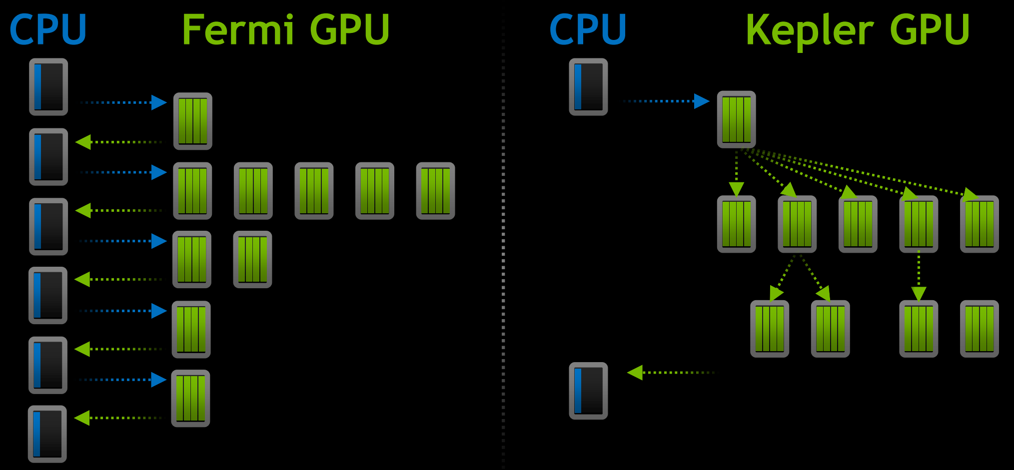
32 Concurrent Work Queues

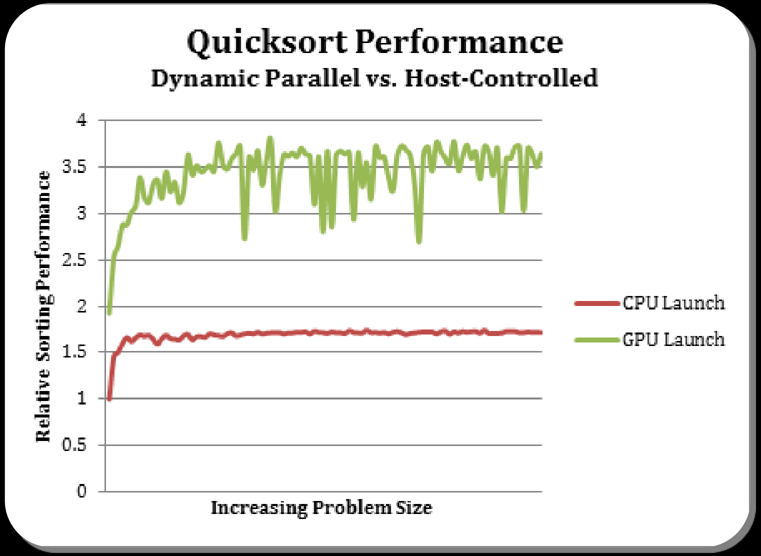




Dynamic Parallelism

Simpler Code, More General, Higher Performance





Code Size Cut by 2x

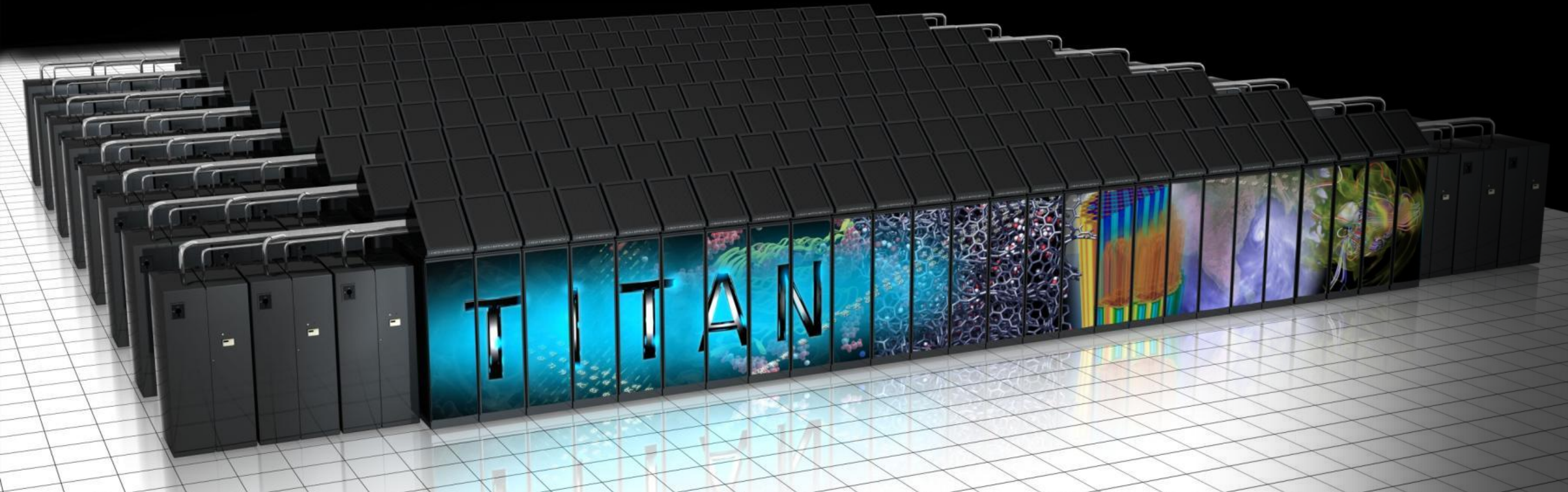
2x Performance

Titan: World’s #1 Supercomputer

18,688 Tesla K20X GPUs

World leading efficiency

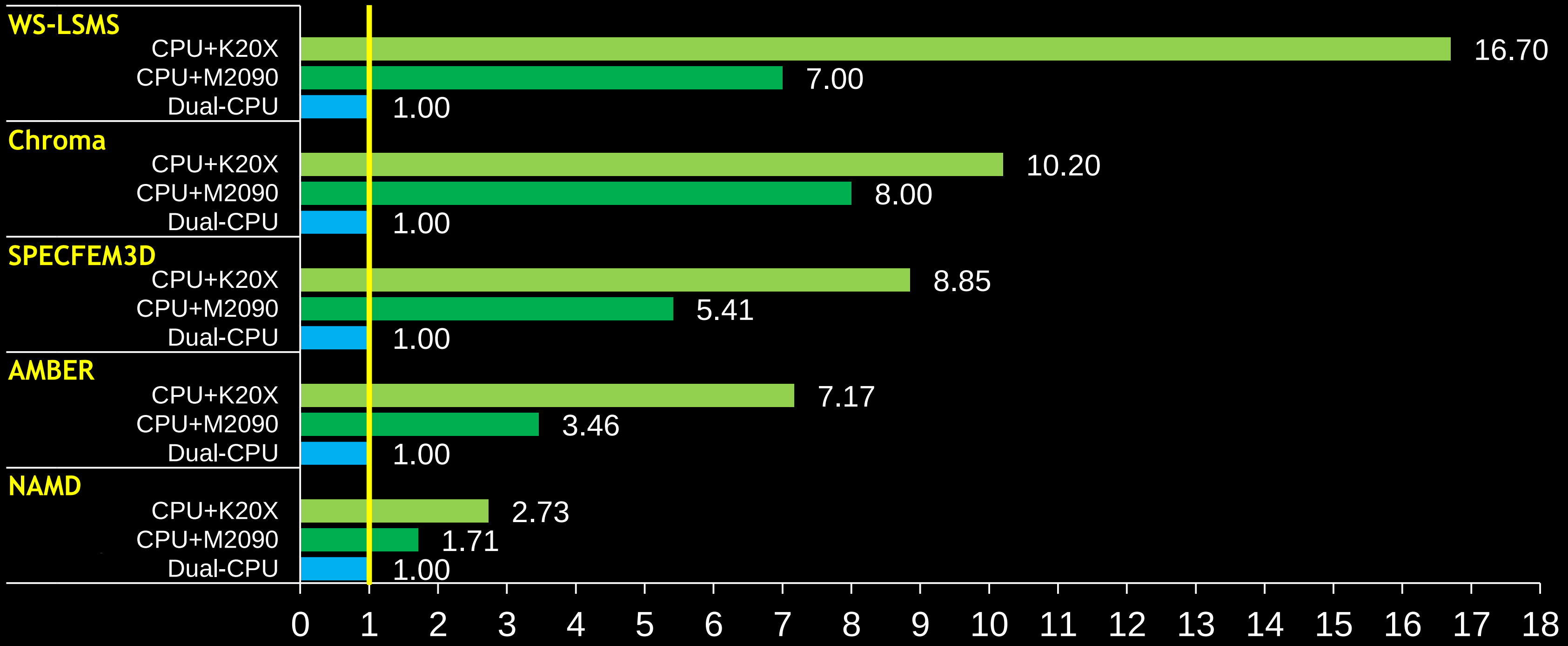
27 Petaflops



Kepler GPU Performance Results

Dual-socket comparison: CPU/GPU node vs. Dual-CPU node

CPU = 8 core SandyBridge E5-2687w 3.10 GHz



The Future

The Future of HPC Programming

Computers are not getting faster… just **wider**

⇒ *Need to structure **all** HPC apps as throughput problems*

Locality *within* nodes much more important

⇒ *Need to expose locality (programming model)*

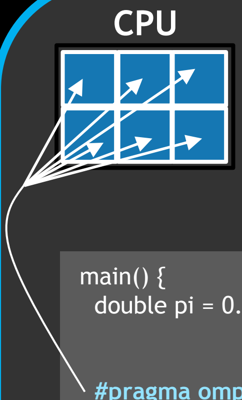
& explicitly manage memory hierarchy (compiler, runtime, autotuner)

How can we enable programmers to code for future processors in a portable way?

OpenACC Directives

Portable Parallelism

OpenMP

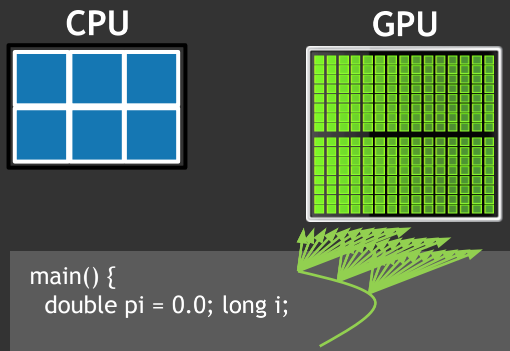



```
main() {
  double pi = 0.0; long i;

  #pragma omp parallel for reduction(+:pi)
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }

  printf(“pi = %f\n”, pi/N);
}
```

OpenACC Directives




```
main() {
  double pi = 0.0; long i;

  #pragma acc parallel loop reduction(+:pi)
  for (i=0; i<N; i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }
  printf(“pi = %f\n”, pi/N);
}
```

How Are GPUs Likely to Evolve Over This Decade?

- Integration
- Further concentration on locality (both HW and SW)
- Reducing overheads
- Continued convergence with consumer technology

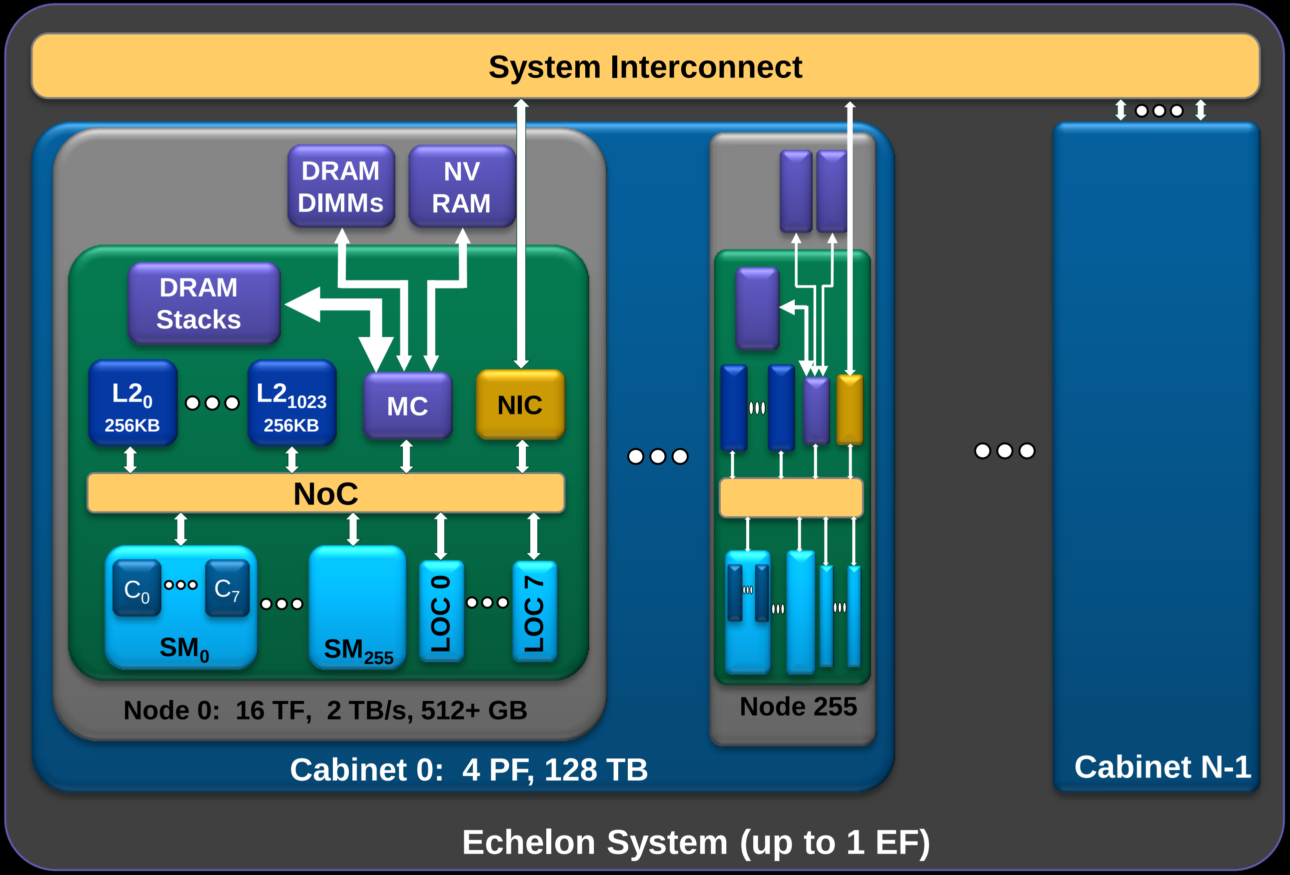


Echelon

NVIDIA's Extreme-Scale Computing Project
DARPA UHPC Program
Targeting 2018

Fast Forward Program

2018 Vision: Echelon Compute Node & System



Key architectural features:

- Malleable memory hierarchy
- Hierarchical register files
- Hierarchical thread scheduling
- Place coherency/consistency
- Temporal SIMT & scalarization
- PGAS memory
- HW accelerated queues
- Active messages
- AMOs everywhere
- Collective engines
- Streamlined LOC/TOC interaction

Summary

- GPU accelerated computing has come a *long* way in a very *short* time
- Has become much easier to program and more general purpose
- Aligned with technology trends, supported by consumer markets
- Future evolution is about:
  - Integration
  - Increased generality – efficient on any code with high parallelism
  - Reducing energy, reducing overheads

- This is simply how computers will be built

Thank You

NVIDIA