HPC Directions at NVIDIA

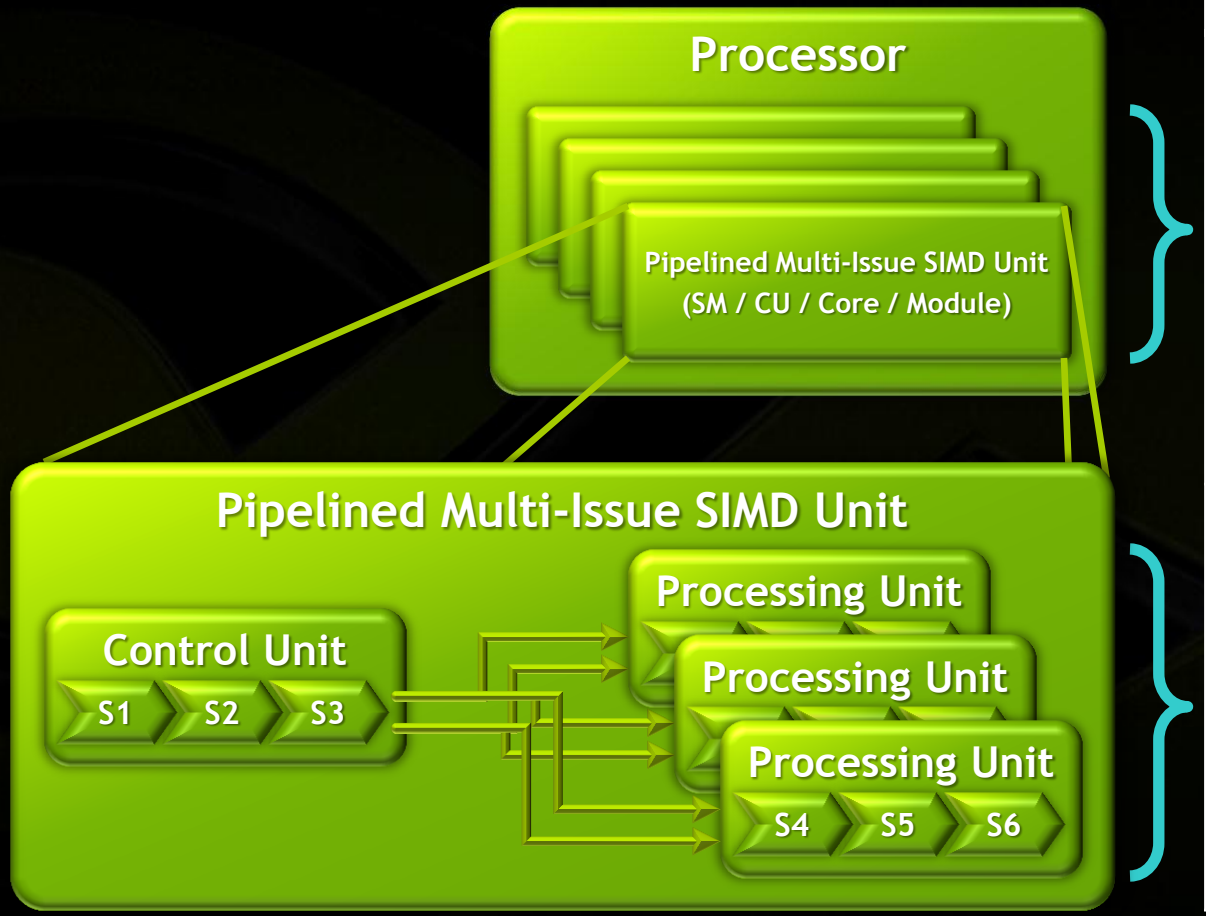
49th HPC User Forum, 2013, Tucson, Arizona

Cyril Zeller
Senior Manager, Developer Technology, NVIDIA

The Future is Parallel

- Single-thread performance per watt has stalled years ago
  - Since the end of processor frequency scaling
- Massive multi-threading is the only way forward to significantly more performance per watt
- Processors are getting wider, not faster

Architectural Features Common to All Processors



NVIDIA Kepler	AMD Southern Islands	Intel Xeon Phi	Intel Sandy Bridge	AMD Bulldozer
15 Streaming Multiprocessors	32 Compute Units	60 Cores	8 Cores	8 Modules
32 threads (Warp)	64 threads (Wavefront)	16-SIMD (512-bit vector)	8-SIMD (256-bit vector)	8-SIMD (256-bit vector)

Application Requirements for High Performance on Any Processor

- High level of parallelism
  - Enough instructions in flight to utilize all cores/SMs and to saturate instruction pipelines
  - Enough bytes in flight to saturate DRAM bandwidth
- Data locality
  - Memory access within a vector/warp should be coalesced
    - To minimize number of cache lines requested by vector/warp
    - To maximize bytes used from each cache line
- Execution locality
  - Control flow within a vector/warp should not diverge
    - To minimize number of instructions executed (divergent code paths are executed for all elements/threads in a vector/warp)

GPU = More Parallelism, More Perf/Watt

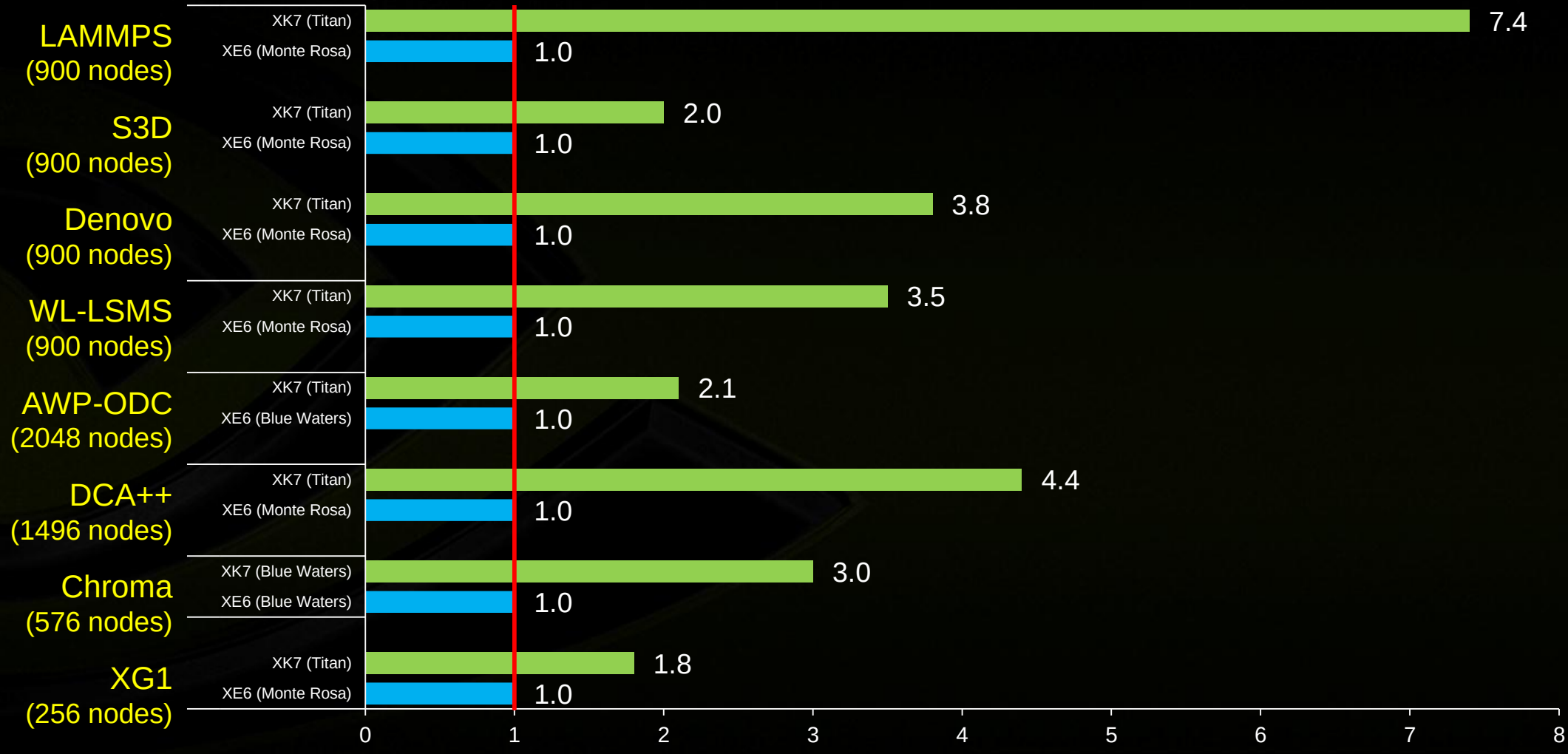
- GPUs require higher level of parallelism than CPUs
  - E.g., number of instructions in flight to saturate fp32 mul instruction pipelines:
    - ~24000 for NVIDIA Kepler K20X
    - ~600 for Intel 8-core Sandy Bridge
- GPUs achieve higher performance per watt than CPUs
  - For massively parallel workloads
  - E.g., peak fp32:
    - 3.95 TFlops for NVIDIA Kepler K20X
    - 0.42 Tflops for Intel 8-core Sandy Bridge

Single-Node Performance

NVIDIA K20X vs Intel dual-socket Sandy Bridge

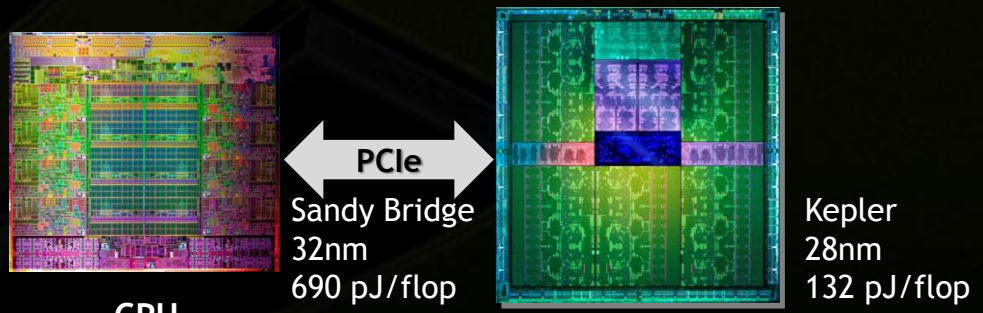
Multi-Node Performance

XK7 (NVIDIA K20X + AMD Opteron 6300) vs XE6 (2 x AMD Opteron 6300)



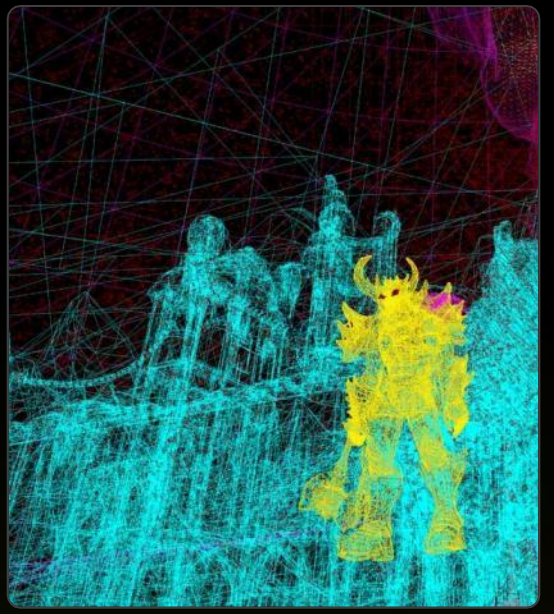
What the Future Holds

- Future processors will require even higher levels of parallelism
  - Since throughputs will increase faster than latencies will decrease
  - Targeting today's GPUs prepares you for future processors (CPUs and GPUs)
    - Often, porting code for GPUs exposes parallelism in a way that helps CPUs too
- The future is hybrid:
  - Do most of the work on cores optimized for extreme energy efficiency
  - Still need a few cores optimized for fast serial work

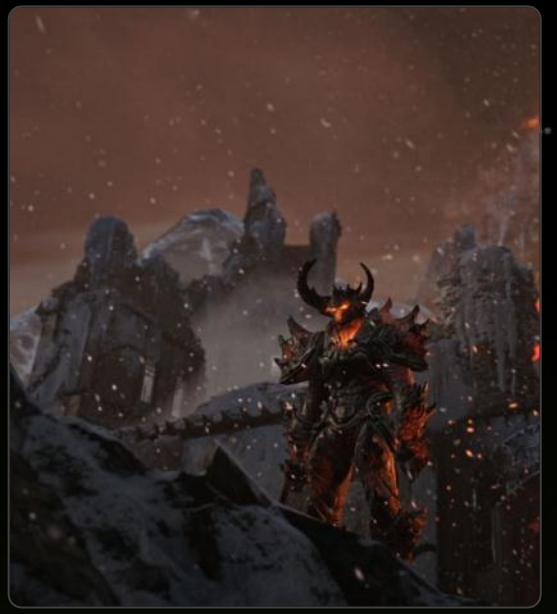


GPU = Many Years of R&D in Massively Parallel Computing:

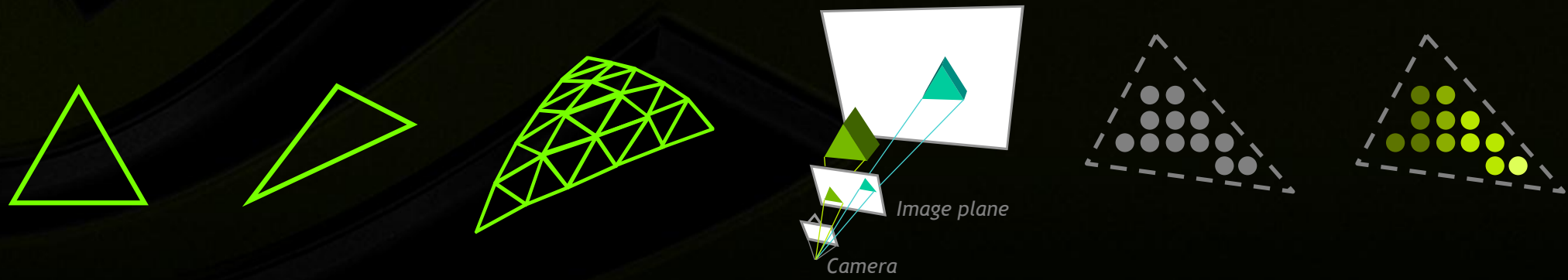
Real-Time 3D Graphics Rendering



Millions of triangles



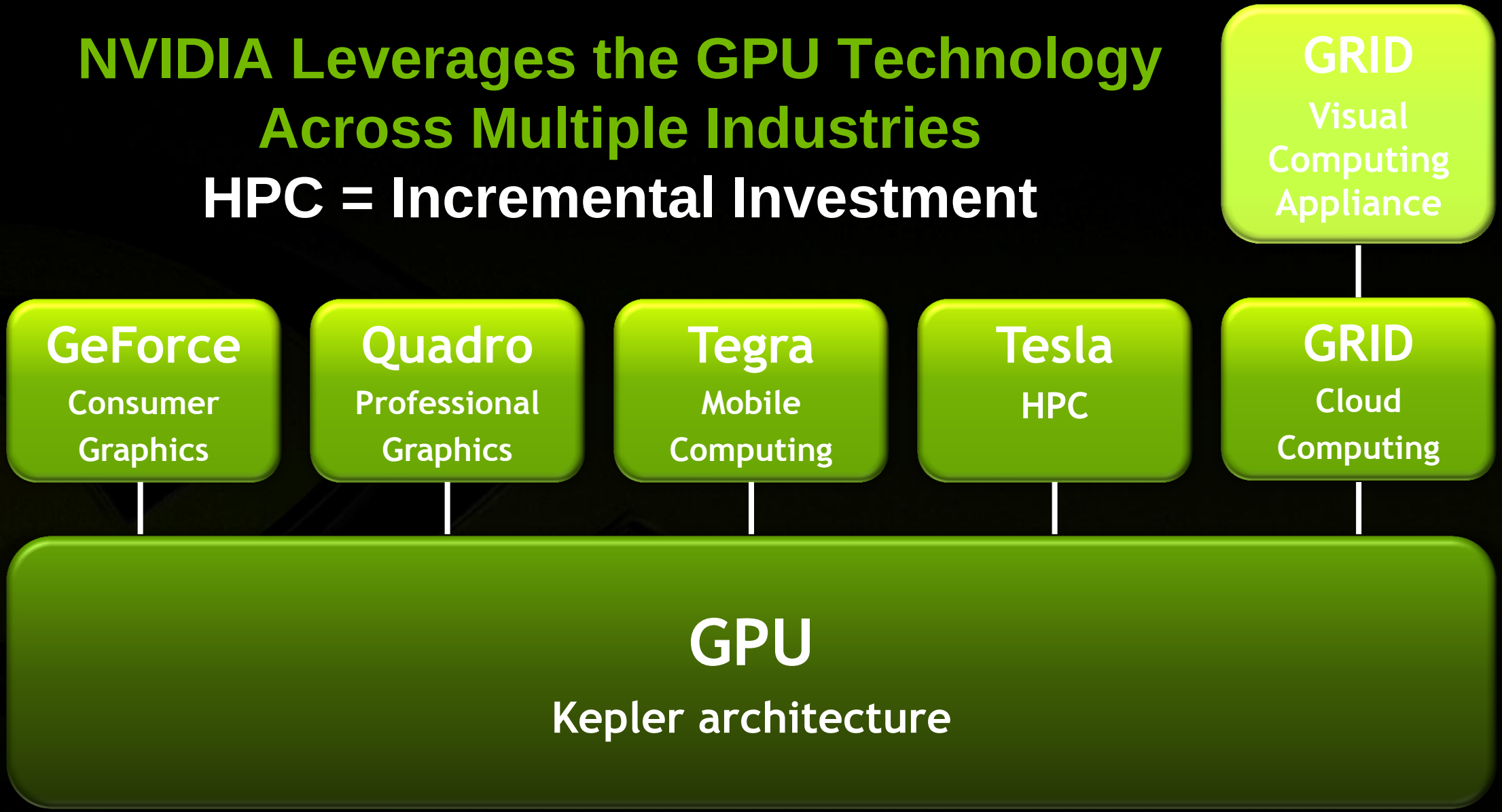
Millions of pixels



Input triangle *Transform vertices* *Tessellate* *Projection* *Rasterize* *Shade*

NVIDIA Leverages the GPU Technology Across Multiple Industries

HPC = Incremental Investment



Investing in the Future

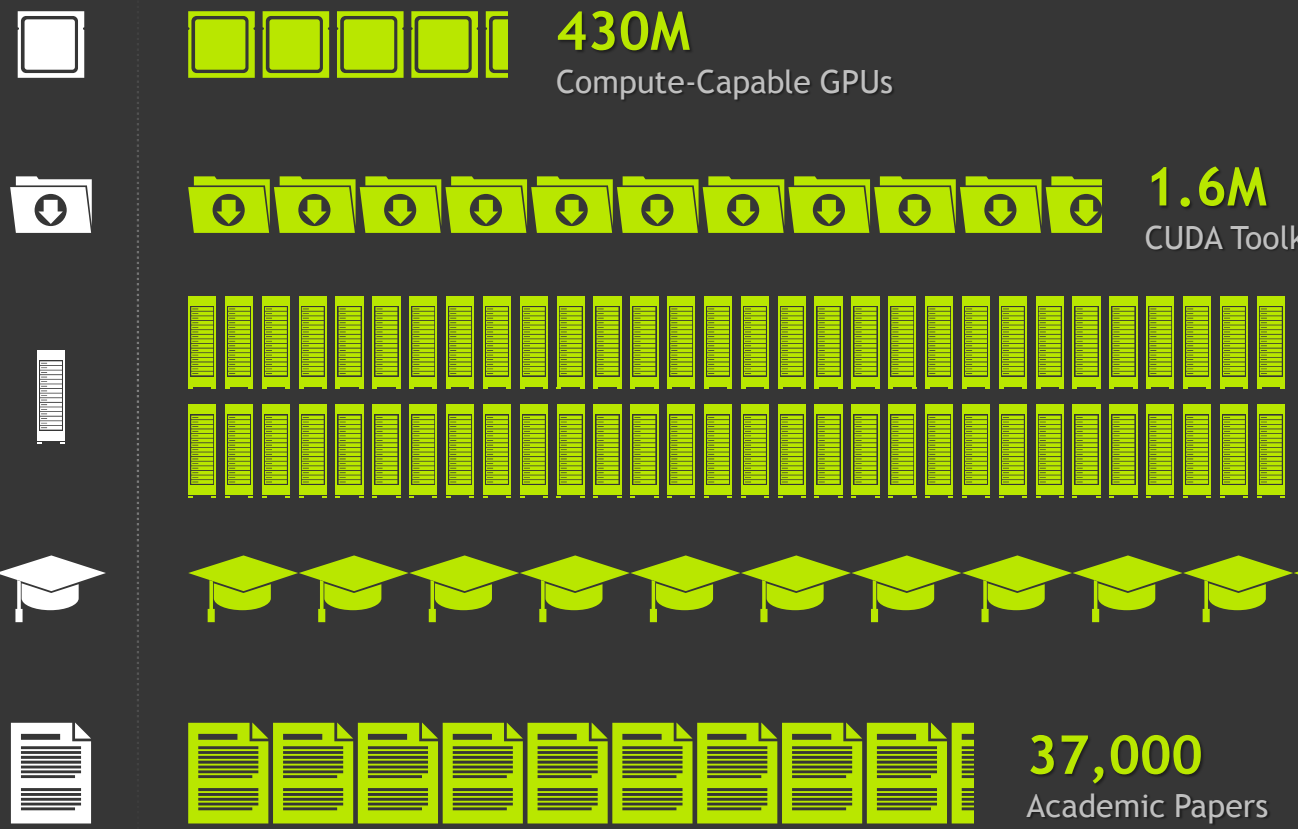
Enable More Developers

More Performance per Watt

Future Computing Platforms

GPU Computing Momentum

	2008	2013
Compute-Capable GPUs	100M	430M
CUDA Toolkit Downloads	150K	1.6M
Supercomputer	1	50
University Courses	60	640
Academic Papers	4,000	37,000



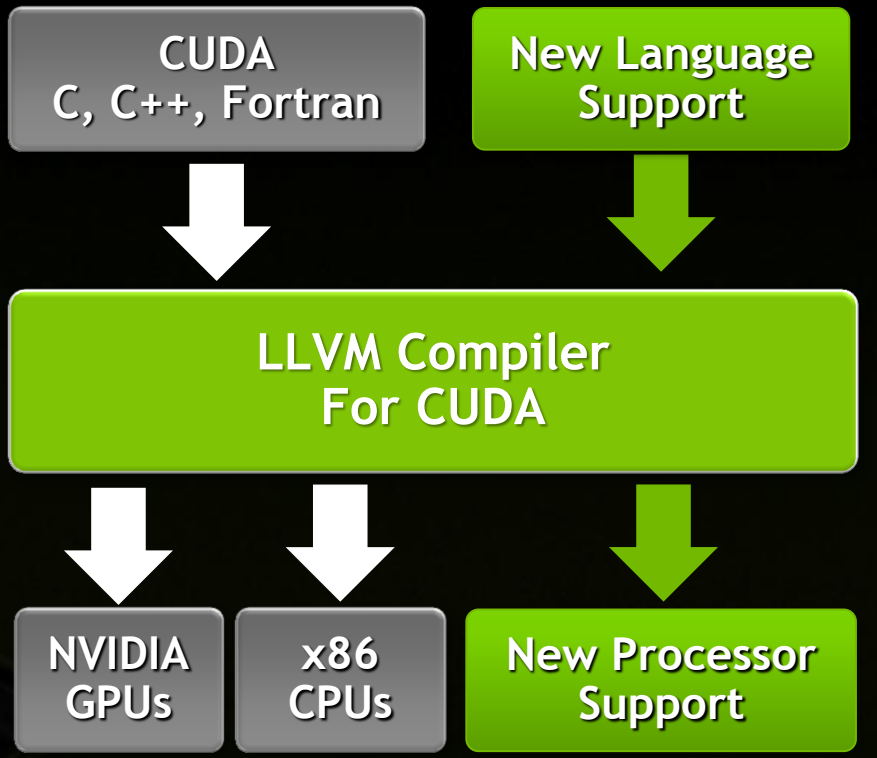
Enabling More Programming Languages

Developers want to build front-ends for

Python, Java, R, DSLs …

Target other processors like

ARM, FPGAs, GPUs, x86 …

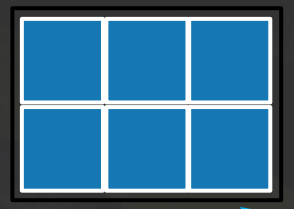


LLVM COMPILER INFRASTRUCTURE

OpenACC Directives

OpenMP

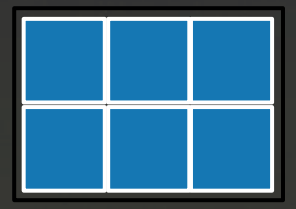
CPU



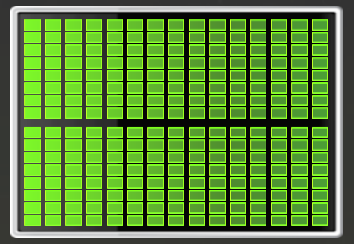
```
main() {
  double pi = 0.0; long i;

  #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

CPU



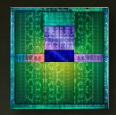
GPU



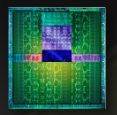
```
main() {
  double pi = 0.0; long i;

  #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);
}
```

Unified Virtual Memory

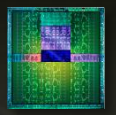
- Software prototype working on Kepler in a future CUDA release
- Hardware support in Maxwell



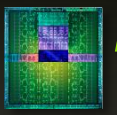
Tesla
CUDA



Fermi
FP64



Kepler
Dynamic Parallelism



Maxwell
Unified Virtual Memory

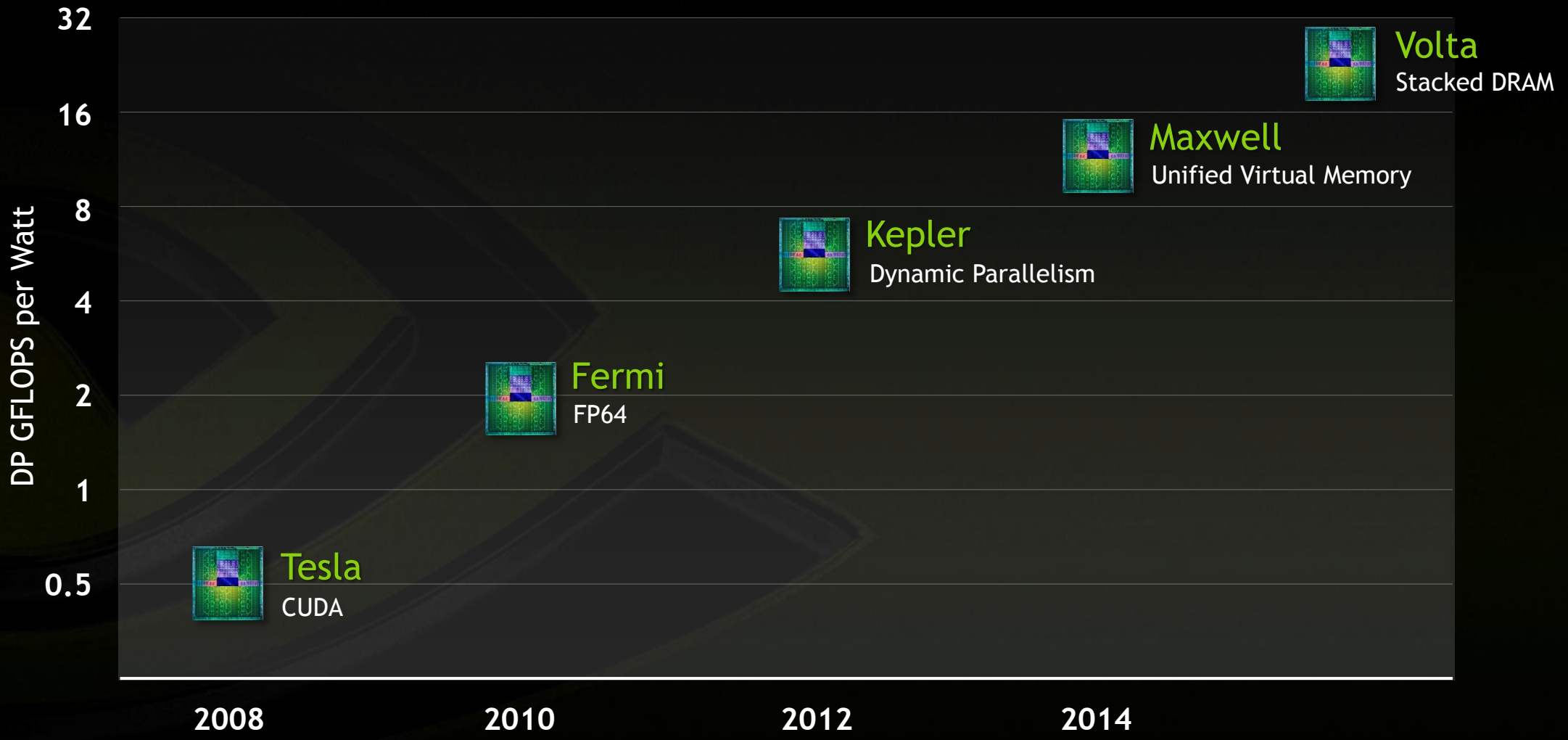
2008 2010 2012 2014

UVM = Explicit Memory Copies No Longer Required

```
void sortfile(FILE *fp, int N) {
  char *data = (char*)malloc(N);
  char *sorted = (char*)malloc(N);
  fread(data, 1, N, fp);

  char *d_data, *d_sorted;
  cudaMalloc(&d_data, N);
  cudaMalloc(&d_sorted, N);
  cudaMemcpy(d_data, data, N, ...);

  parallel_sort<<< ... >>>(d_sorted, d_data, N);

  cudaMemcpy(sorted, d_sorted, N, ...);
  cudaFree(d_data);
  cudaFree(d_sorted);

  use_data(sorted);
  free(data); free(sorted);
}
```

```
void sortfile(FILE *fp, int N) {
  char *data = (char*)malloc(N);
  char *sorted = (char*)malloc(N);
  fread(data, 1, N, fp);

  parallel_sort<<< ... >>>(  sorted,   data, N);

  use_data(sorted);
  free(data); free(sorted);
}
```

More Performance per Watt

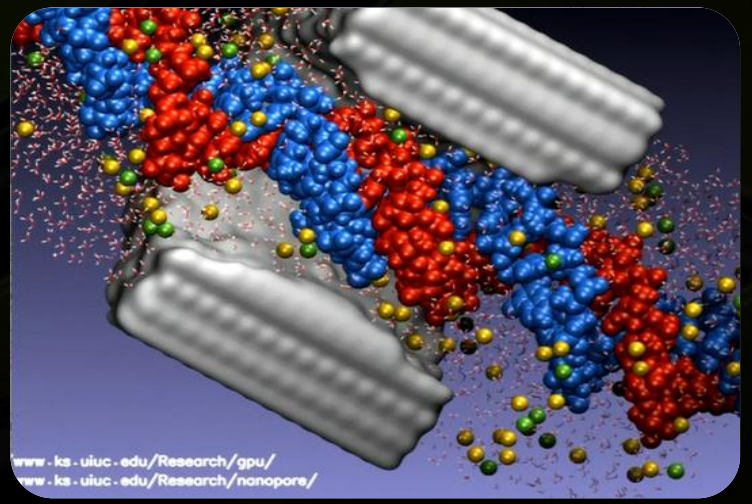


Kayla Development Platform

CUDA 5 | OpenGL 4.3

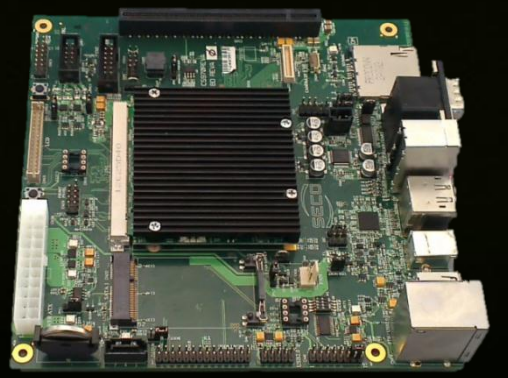
Kick starts ARM + CUDA Ecosystem

NAMD Ported in 2 Days



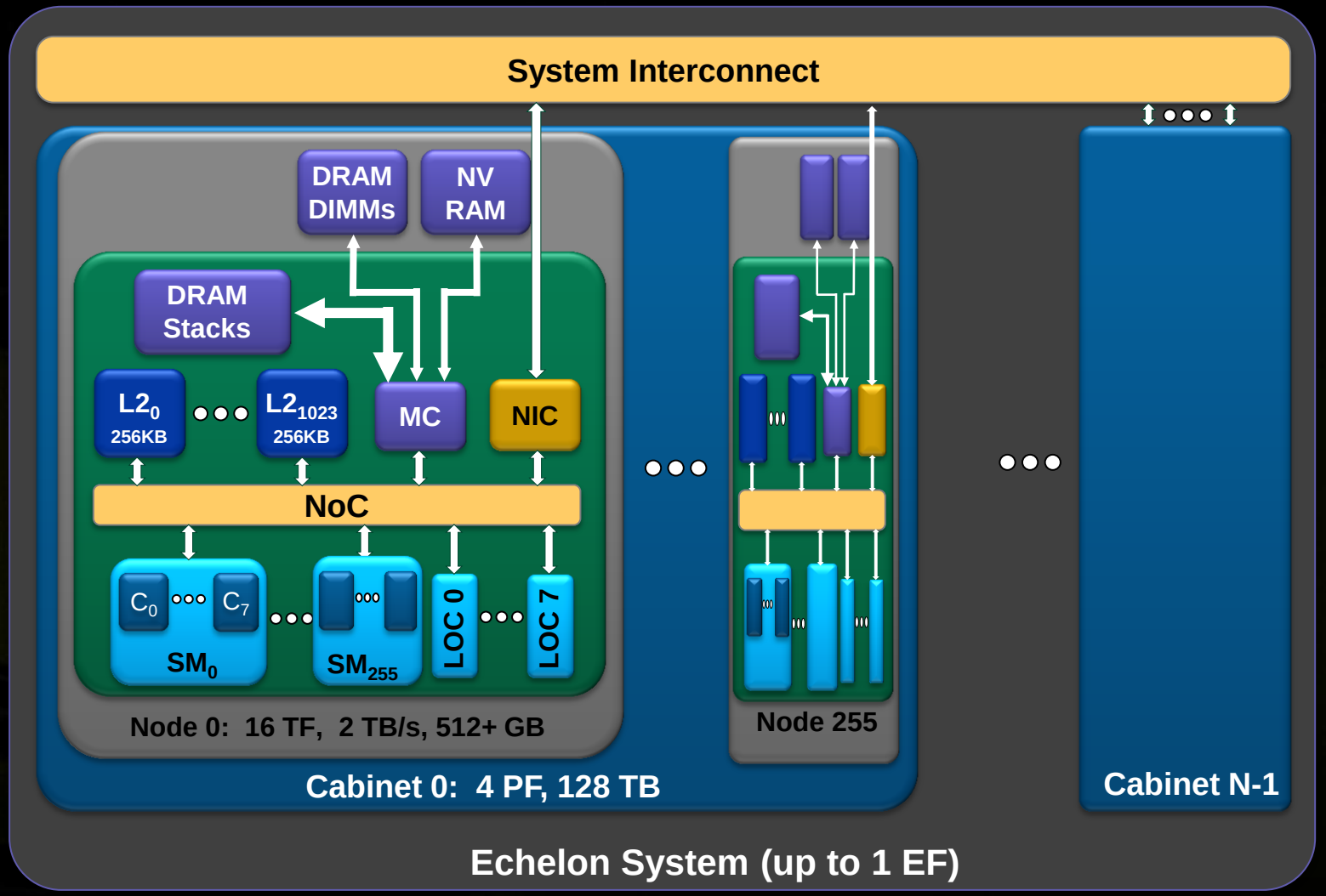
https://developer.nvidia.com/kayla-platform

Quad ARM + Kepler GPU



Quad ARM + Any CUDA GPU

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