The Future of GPU Computing

Bill Dally

Chief Scientist & Sr. VP of Research, NVIDIA

Bell Professor of Engineering, Stanford University

November 18, 2009

The Future of Computing

Bill Dally

Chief Scientist & Sr. VP of Research, NVIDIA

Bell Professor of Engineering, Stanford University

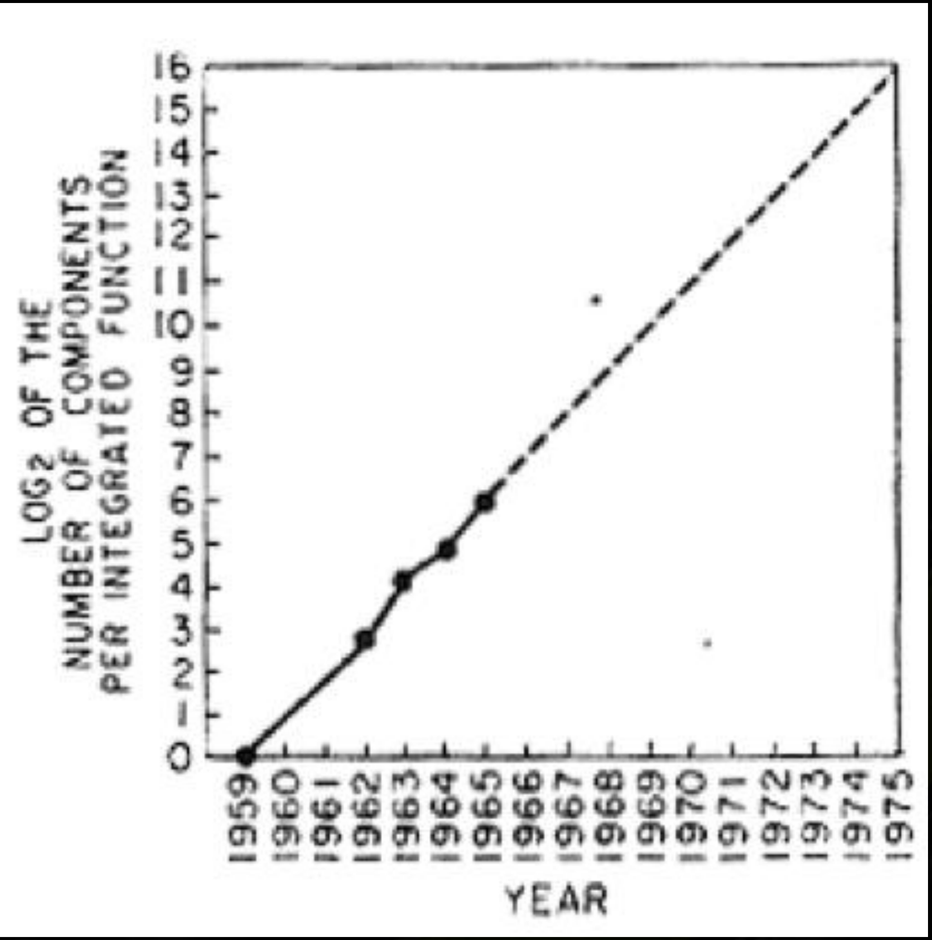
November 18, 2009

Outline

- Single-thread performance is no longer scaling
- Performance = Parallelism
- Efficiency = Locality
- Applications have lots of both
- Machines need lots of cores (parallelism) and an exposed storage hierarchy (locality)
- A programming system must abstract this
- The future is even more parallel

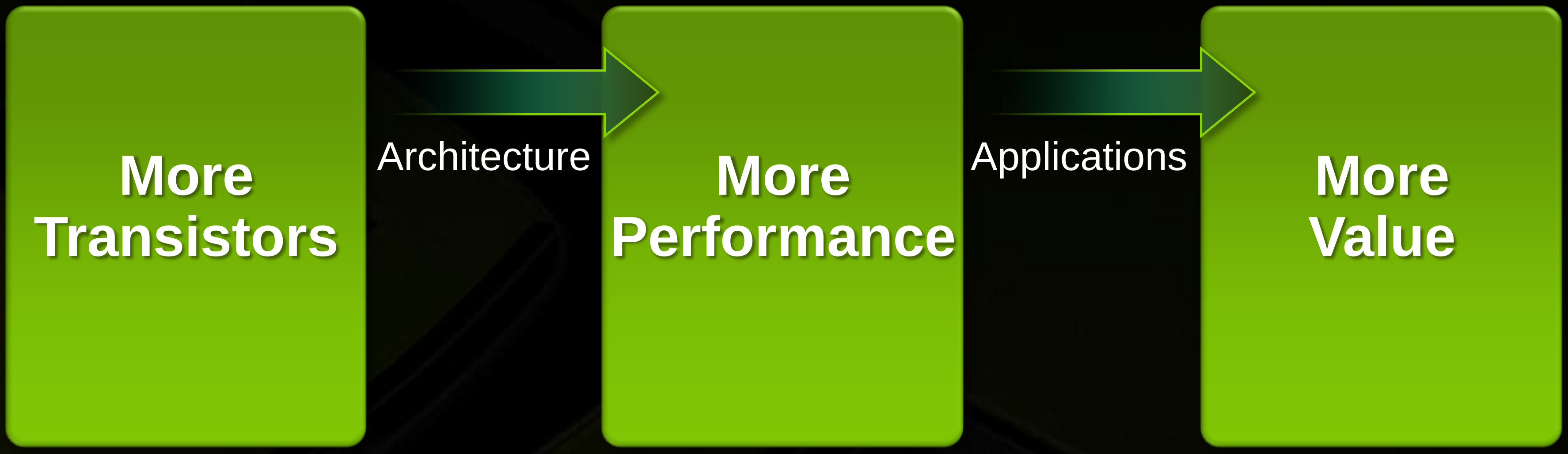
Single-threaded processor performance is no longer scaling

Moore's Law

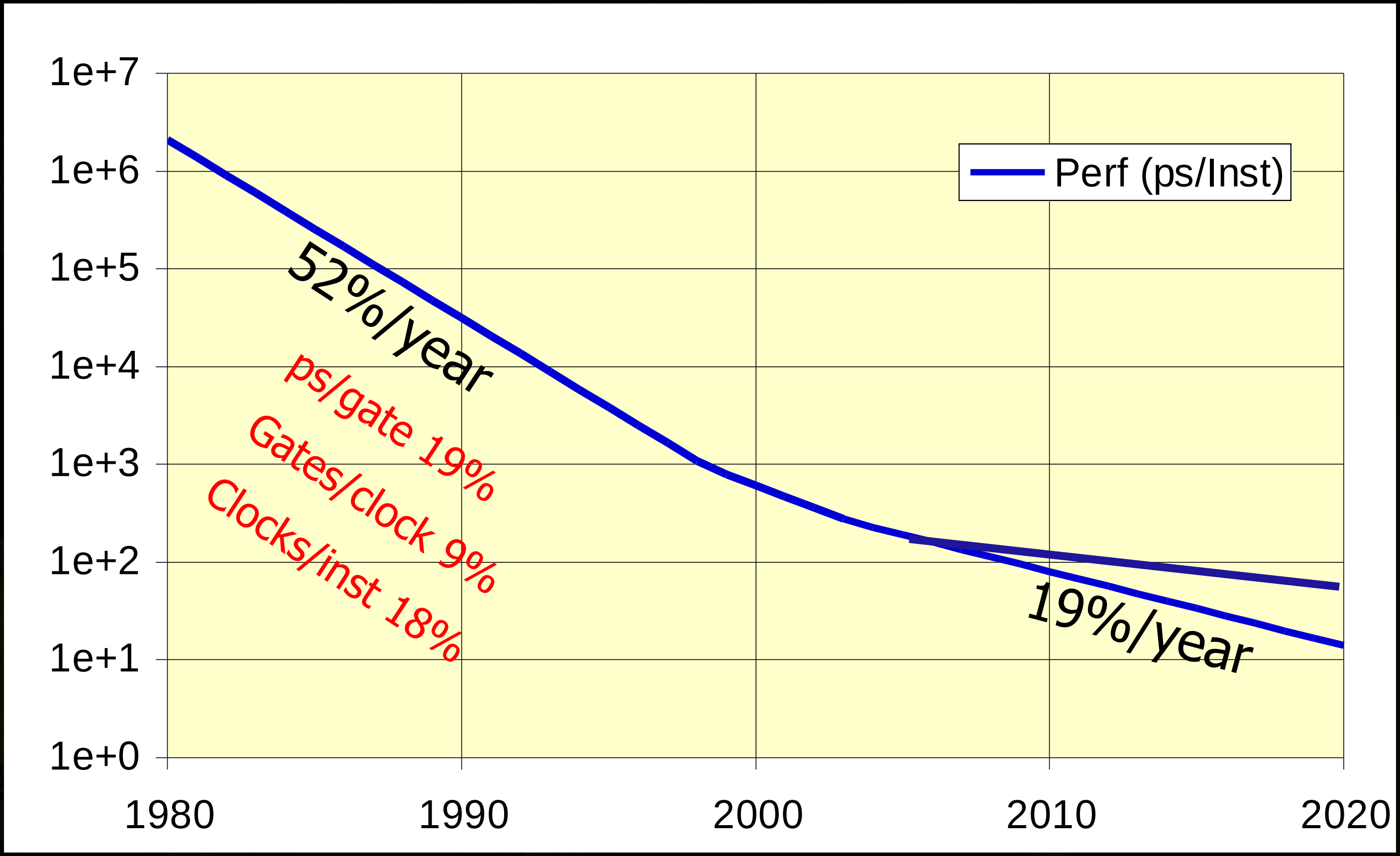


- In 1965 Gordon Moore predicted the *number of transistors* on an integrated circuit would double every year.
  - Later revised to 18 months
- Also predicted L^3 power scaling for constant function
- No prediction of processor performance

Moore, Electronics 38(8) April 19, 1965

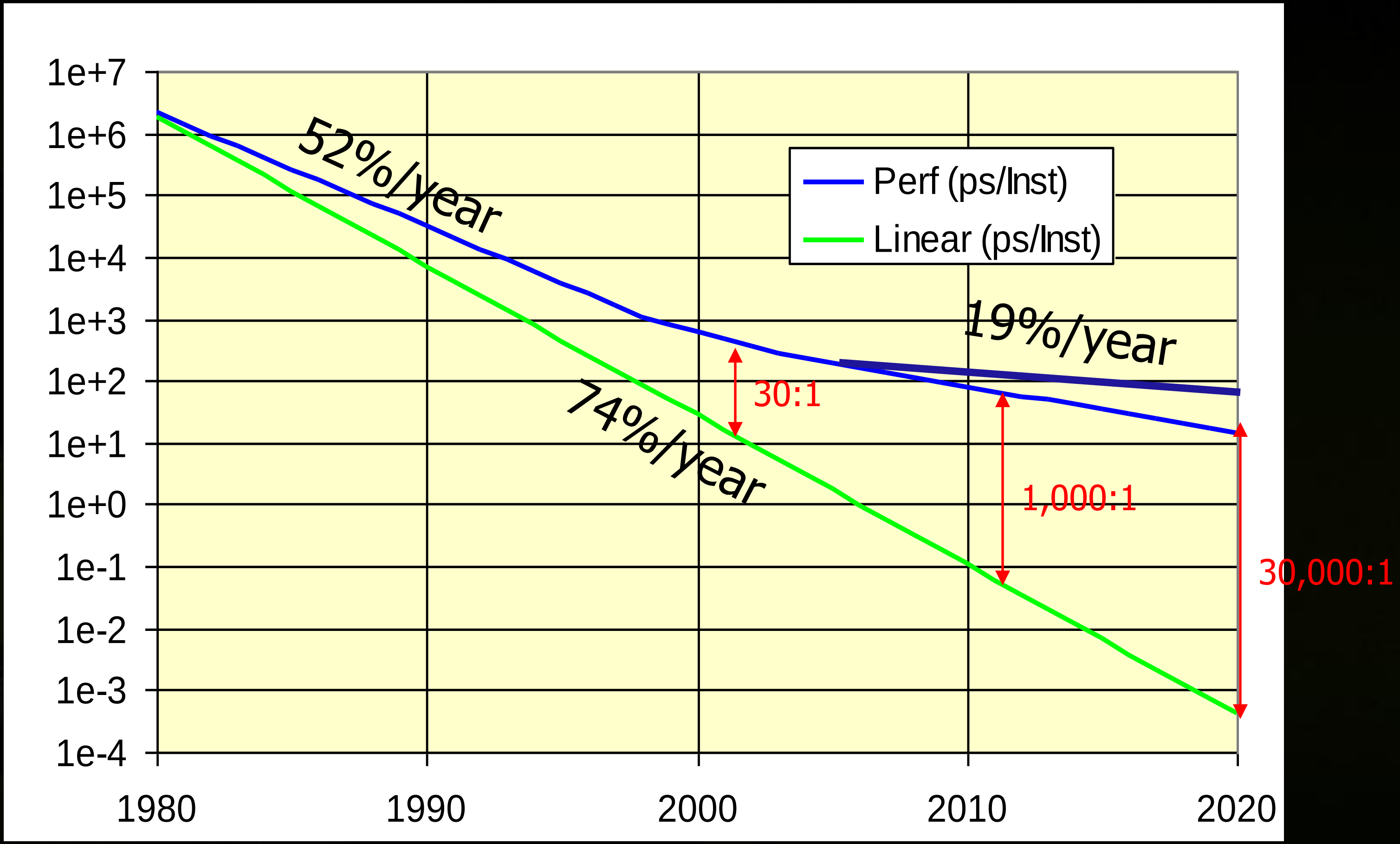

More Transistors
Architecture
More Performance
Applications
More Value

The End of ILP Scaling



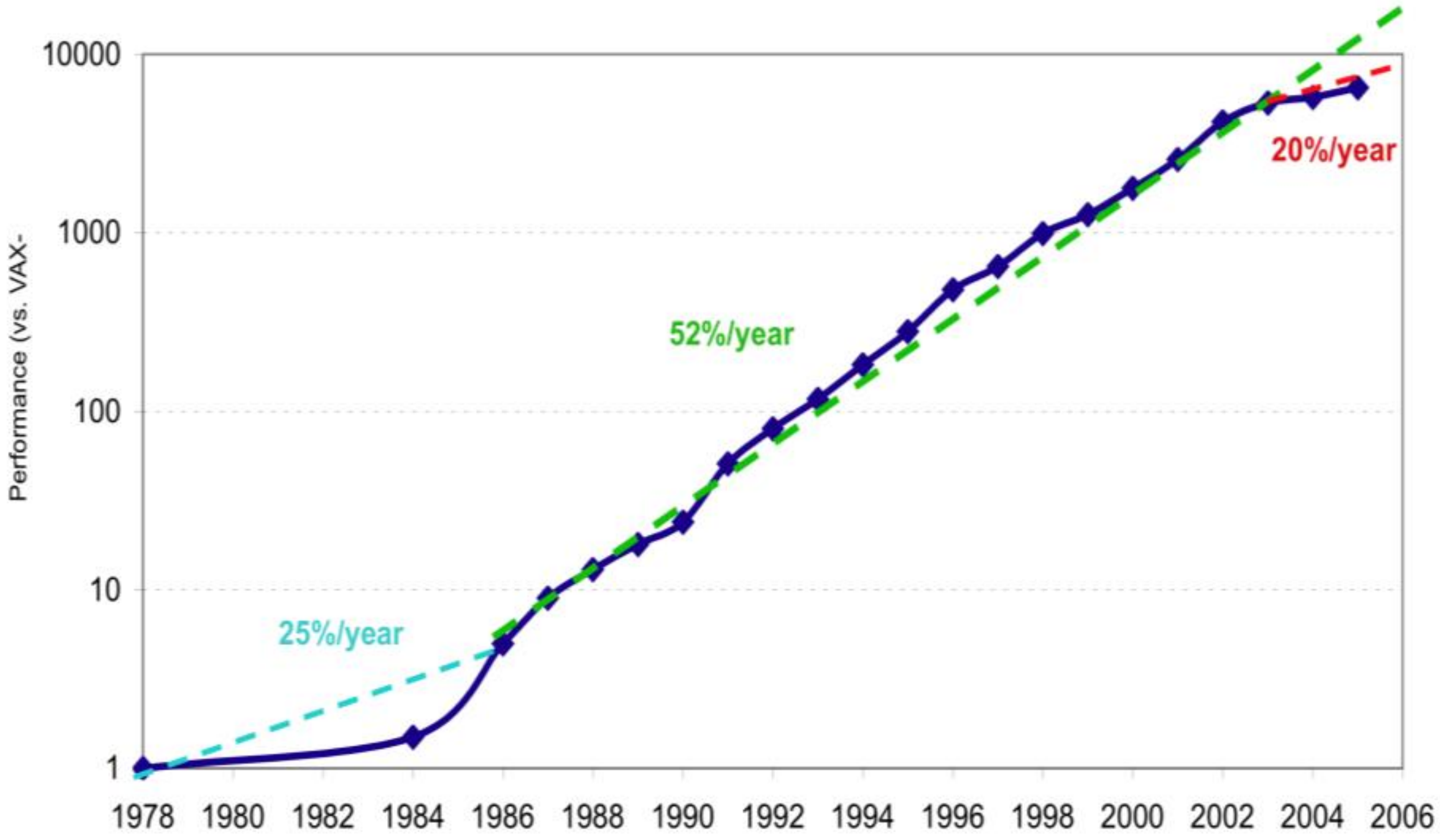
Dally et al., The Last Classical Computer, ISAT Study, 2001

Explicit Parallelism is Now Attractive



Dally et al., The Last Classical Computer, ISAT Study, 2001

Single-Thread Processor Performance vs Calendar Year



Source: Hennessy & Patterson, CAAQA, 4th Edition

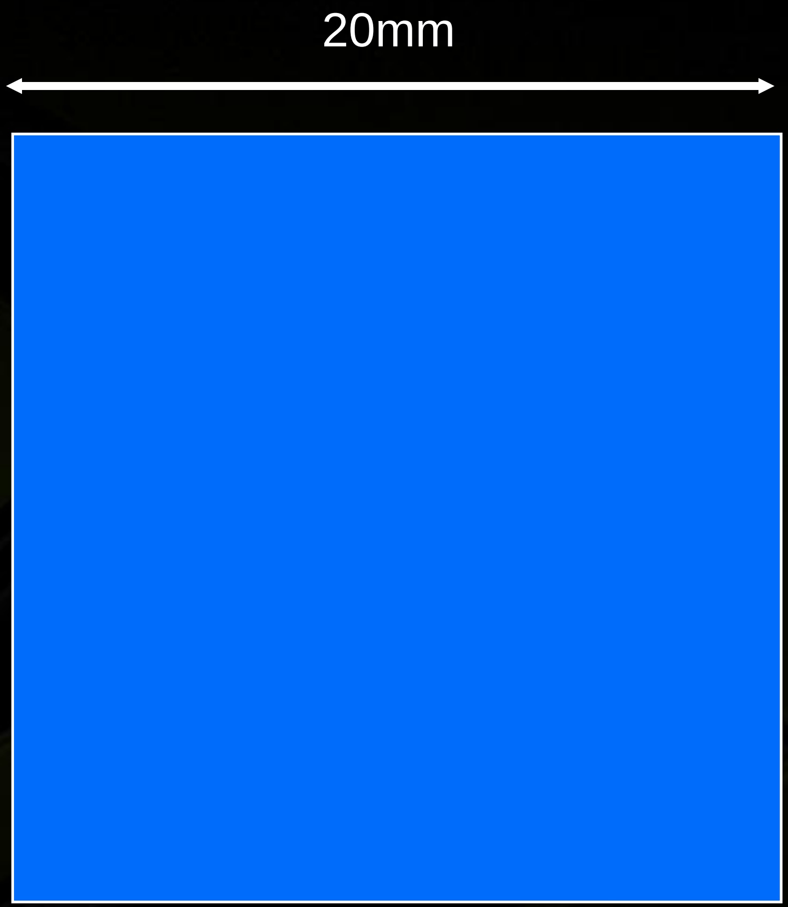
Single-threaded processor performance is no longer scaling

Performance = Parallelism

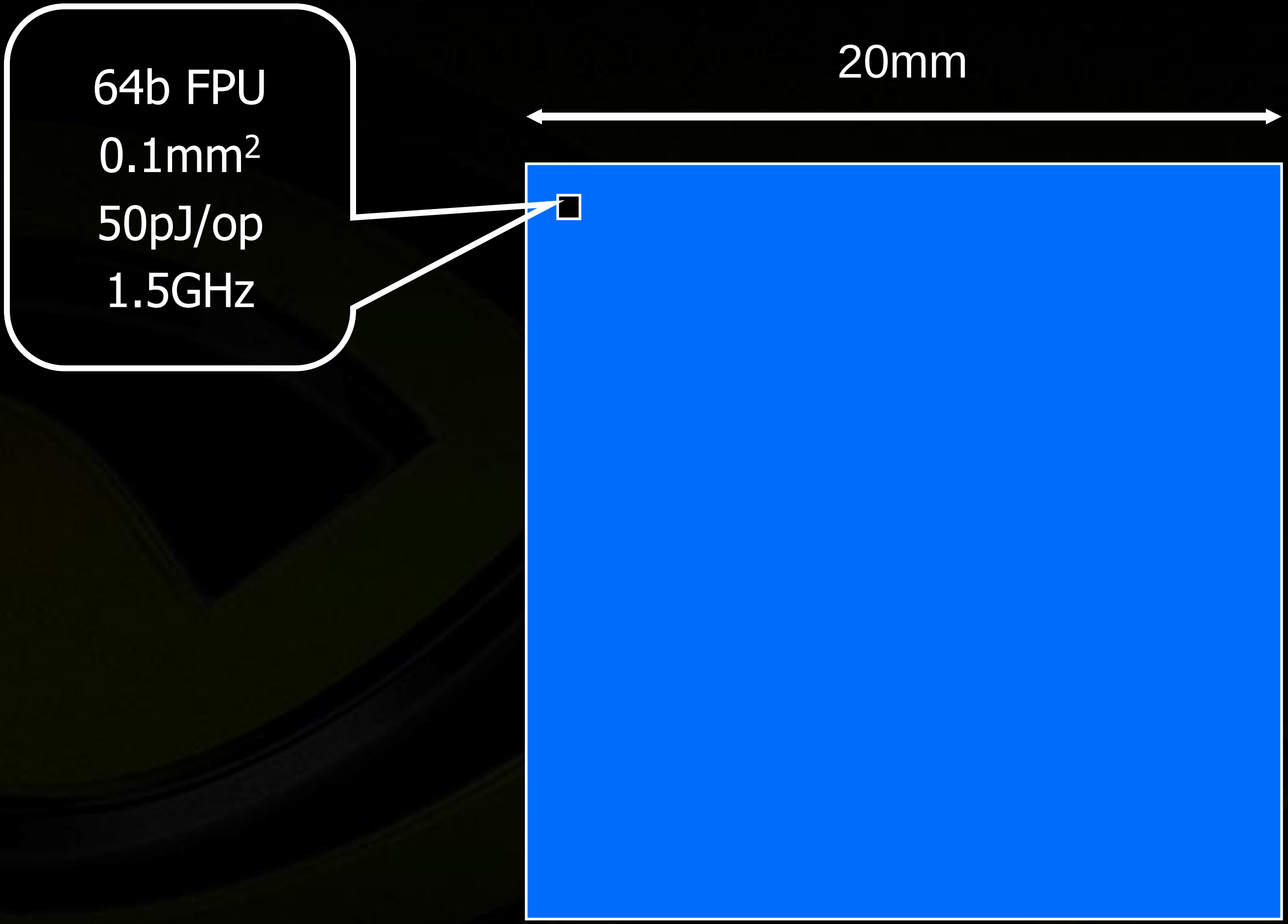
Chips are power limited

and most power is spent moving data

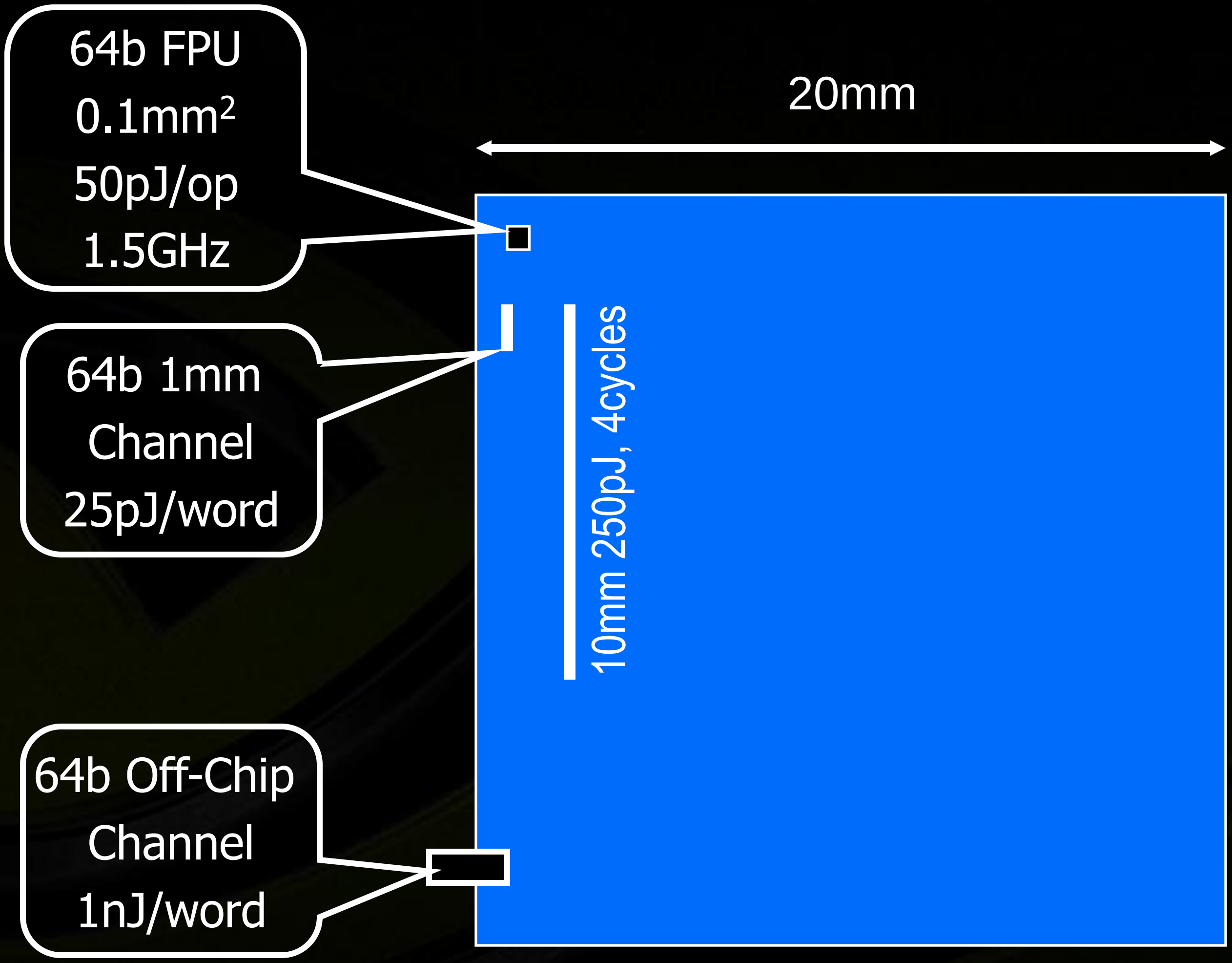
CMOS Chip is our Canvas



4,000 64b FPUs fit on a chip



Moving a word across die = 10FMAs
Moving a word off chip = 20FMAs



64b Floating Point

Chips are power limited

Most power is spent moving data

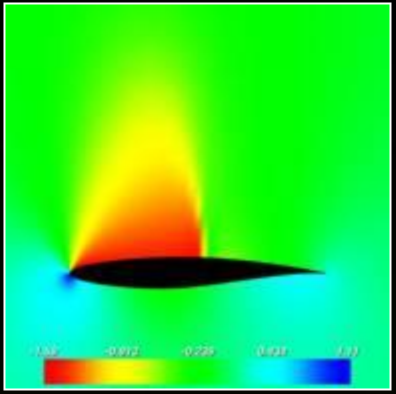
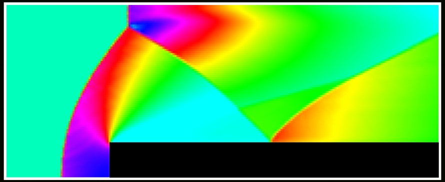
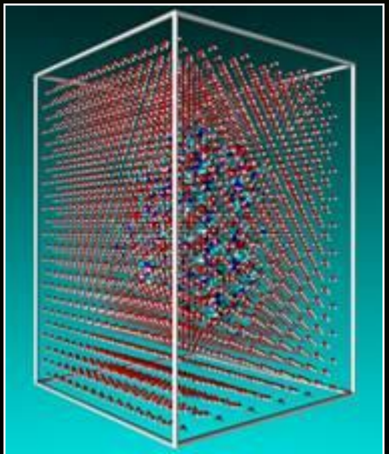
Efficiency = Locality

Performance = Parallelism

Efficiency = Locality

Scientific Applications

- Large data sets
  - Lots of parallelism
- Increasingly irregular (AMR)
  - Irregular and dynamic data structures
  - Requires efficient gather/scatter
- Increasingly complex models
  - Lots of locality
- Global solution sometimes bandwidth limited
  - Less locality in these phases

Performance = Parallelism

Efficiency = Locality

Fortunately, most applications have lots of both.

Amdahl’s *law* doesn’t apply to most future applications.

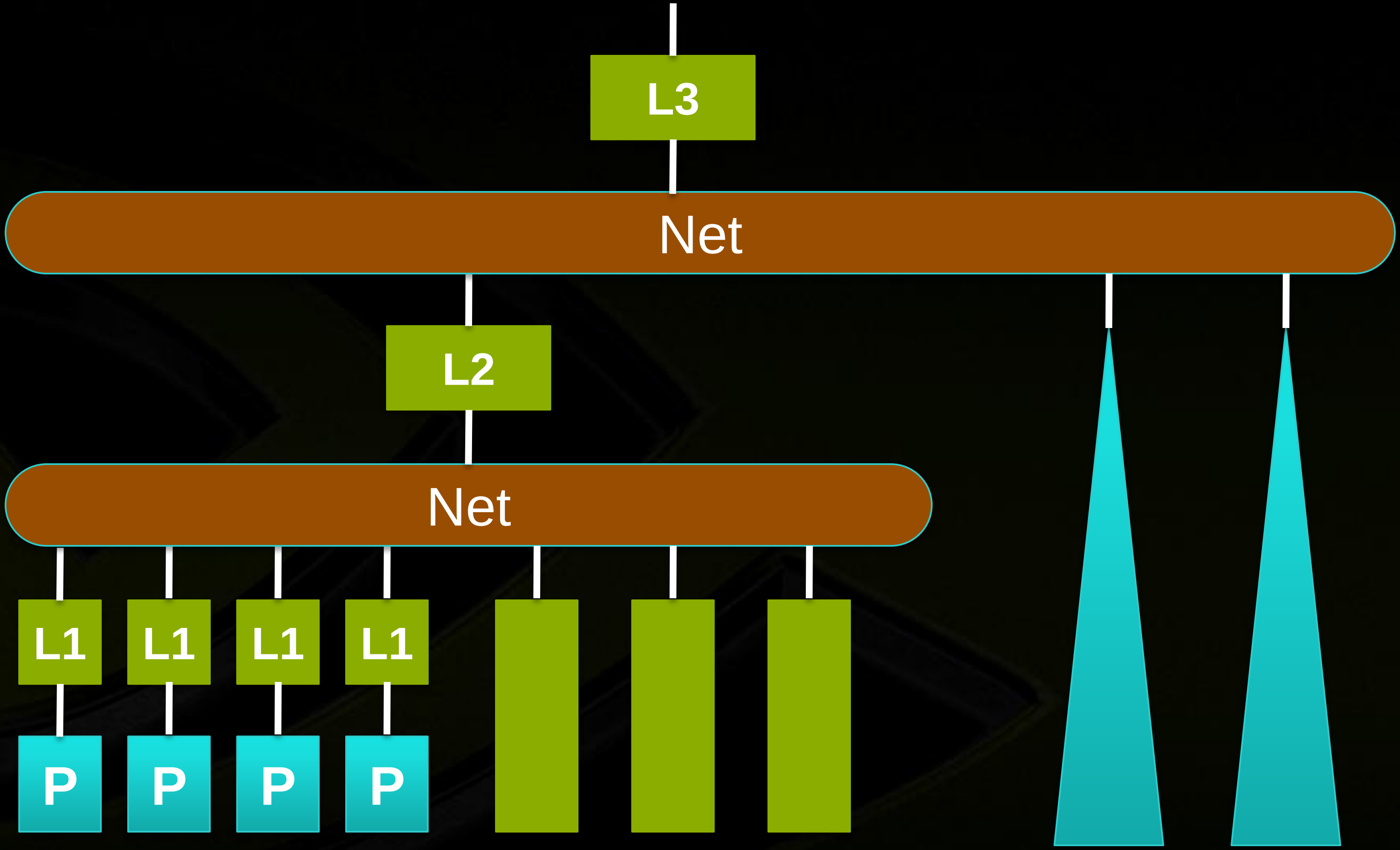
Exploiting parallelism and locality requires:

Many efficient processors
(To exploit parallelism)

An exposed storage hierarchy
(To exploit locality)

A programming system that abstracts this

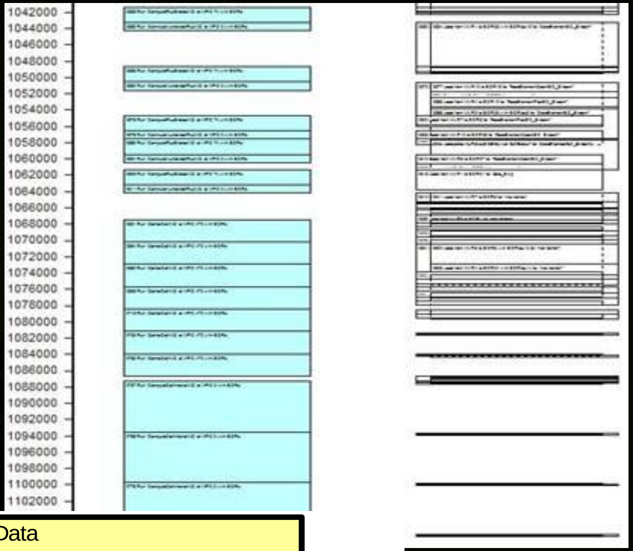
Tree-structured machines

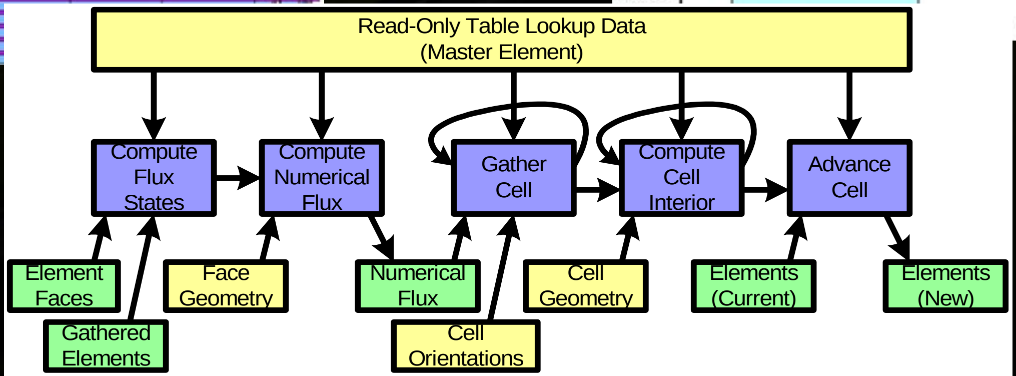


Optimize use of scarce bandwidth

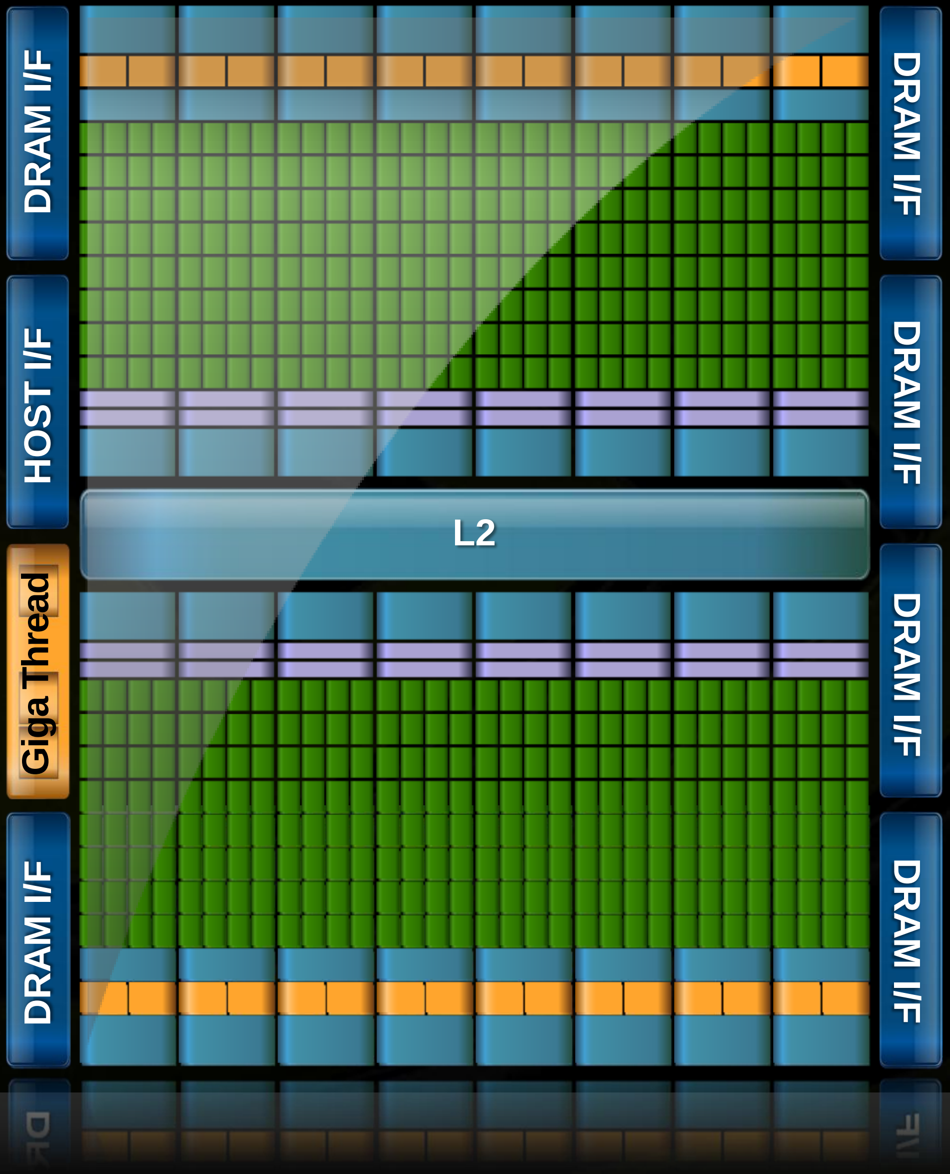
- **Provide rich, exposed storage hierarchy**
- **Explicitly manage data movement on this hierarchy**
  - **Reduces demand, increases utilization**

690
700
710
720
730
740
750
760
770
780
790
800
810



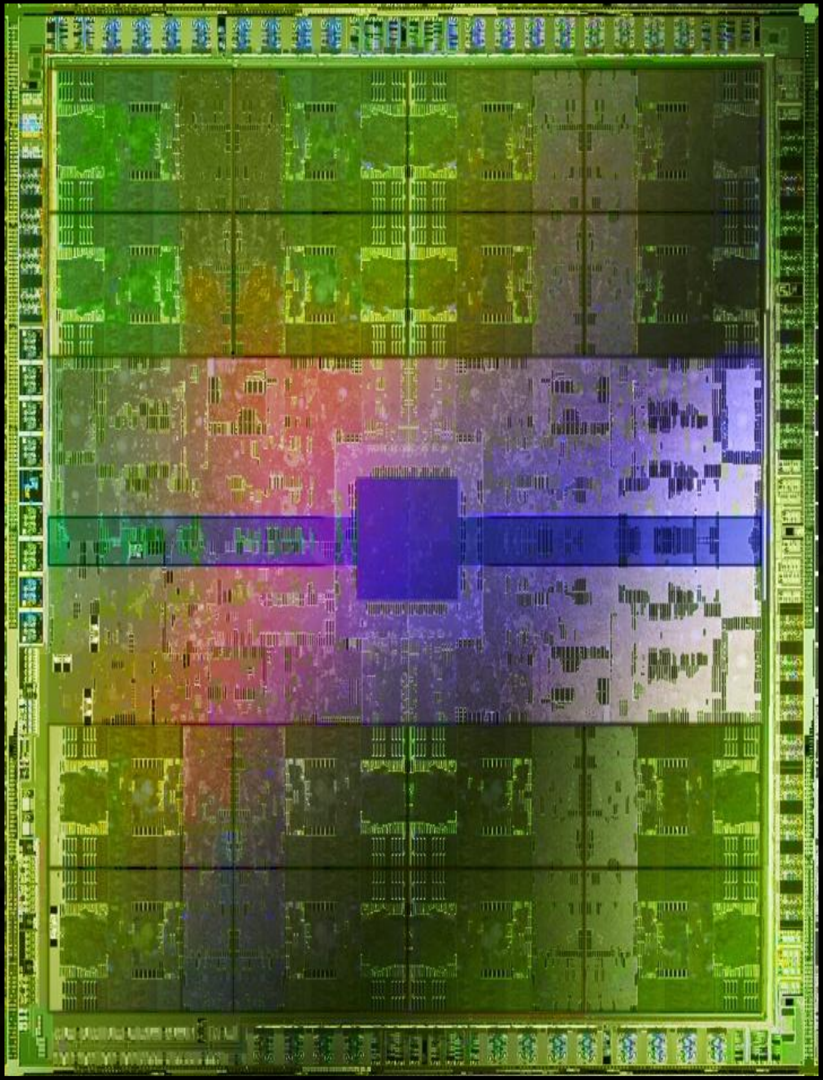


Fermi is a throughput computer



- 512 efficient cores
- Rich storage hierarchy
  - Shared memory
  - L1
  - L2
  - GDDR5 DRAM

Fermi



Avoid Denial Architecture

- Single thread processors are in denial about parallelism and locality
- They provide two illusions:
  - Serial execution - Denies parallelism
    - Tries to exploit parallelism with ILP - inefficient & limited scalability
  - Flat memory - Denies locality
    - Tries to provide illusion with caches – very inefficient when working set doesn’t fit in the cache
- These illusions inhibit performance and efficiency

CUDA Abstracts the GPU Architecture

Programmer sees many cores and exposed storage hierarchy, but is isolated from details.

CUDA as a Stream Language

- Launch a cooperative thread array

```
foo<<<nblocks, nthreads>>>(x, y, z) ;
```

- Explicit control of the memory hierarchy

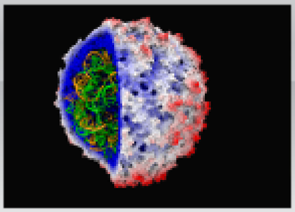
```
__shared__ float a[SIZE] ;
```

- Also enables communication between threads of a CTA
- Allows access to arbitrary data within a kernel

Examples

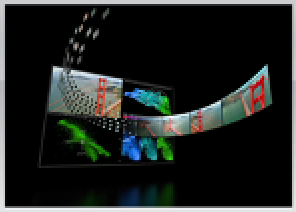
146X

Interactive visualization of volumetric white matter connectivity



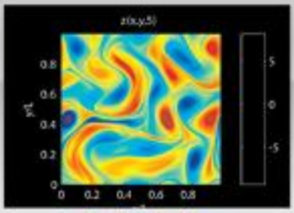
36X

Ionic placement for molecular dynamics simulation on GPU



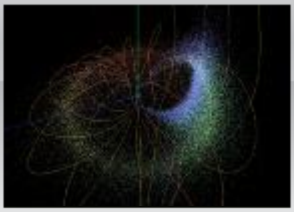
19X

Transcoding HD video stream to H.264



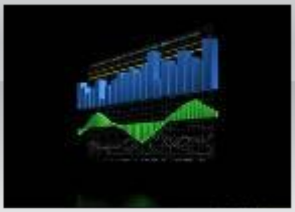
17X

Fluid mechanics in Matlab using .mex file CUDA function



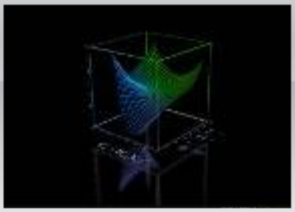
100X

Astrophysics N-body simulation



149X

Financial simulation of LIBOR model with swaptions



47X

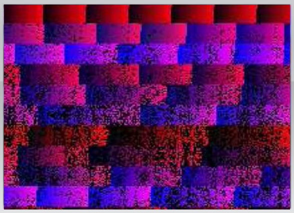
GLAME@lab: an M-script API for GPU linear algebra

20X

Ultrasound medical imaging for cancer diagnostics

24X

Highly optimized object oriented molecular dynamics



30X

Cmatch exact string matching to find similar proteins and gene sequences

Current CUDA Ecosystem

NVIDIA

Over 200 Universities Teaching CUDA

UIUC	IIT Delhi
MIT	Tsinghua
Harvard	Dortmundt
Berkeley	ETH Zurich
Cambridge	Moscow
Oxford	NTU
…	…

Languages

C, C++
DirectX
Fortran
OpenCL
Python

Compilers

PGI Fortran
CAPs HMPP
MCUDA
MPI
NOAA Fortran2C
OpenMP

Applications

Libraries

FFT
BLAS
LAPACK
Image processing
Video processing
Signal processing
Vision

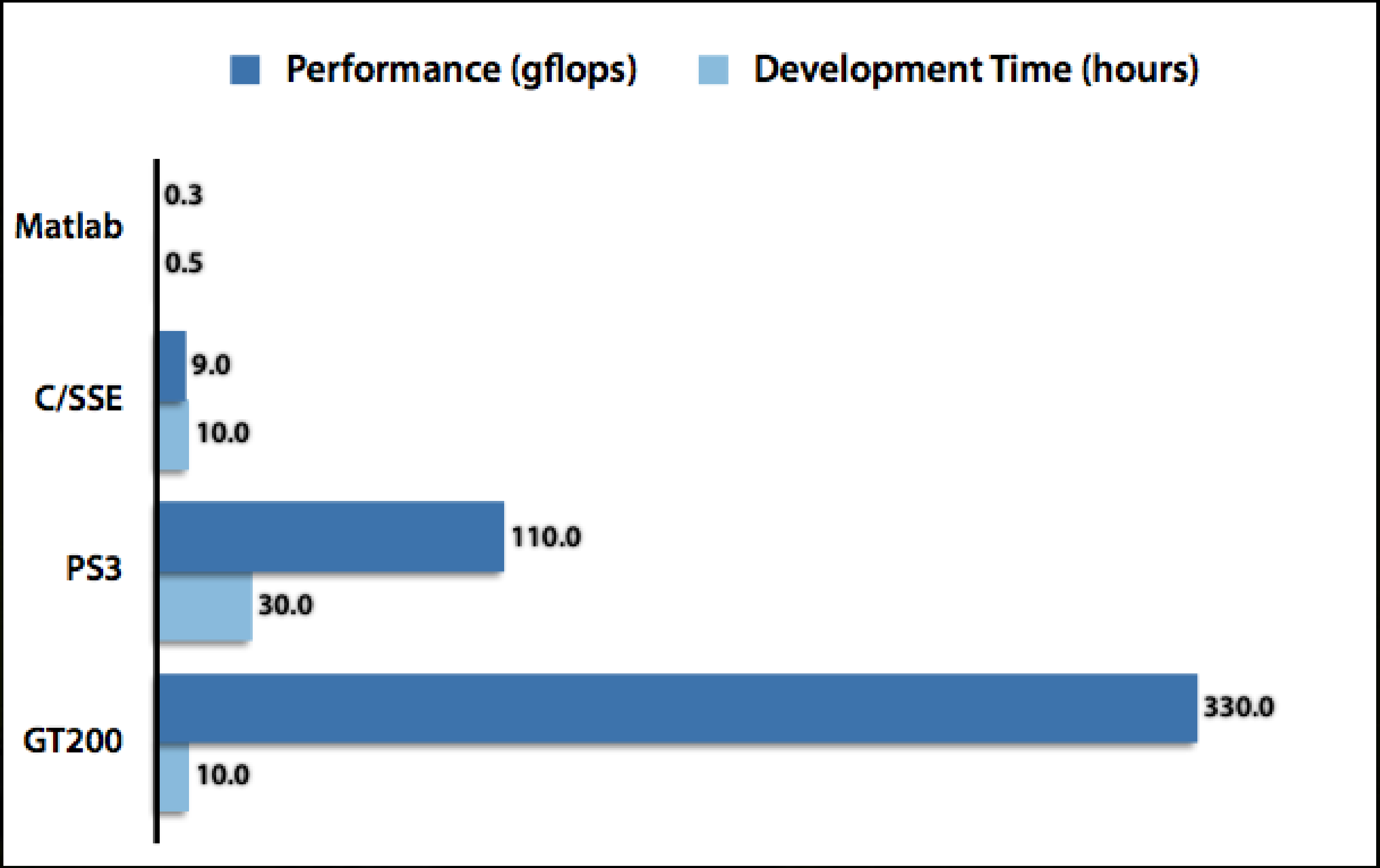
Consultants

acceleware
HPC
ANEO
SCALABLE GRAPHICS
CAPS
STONE RIDGE TECHNOLOGY
EMPhotonics
TECH-X
GASS
GPU Tech
WIPRO

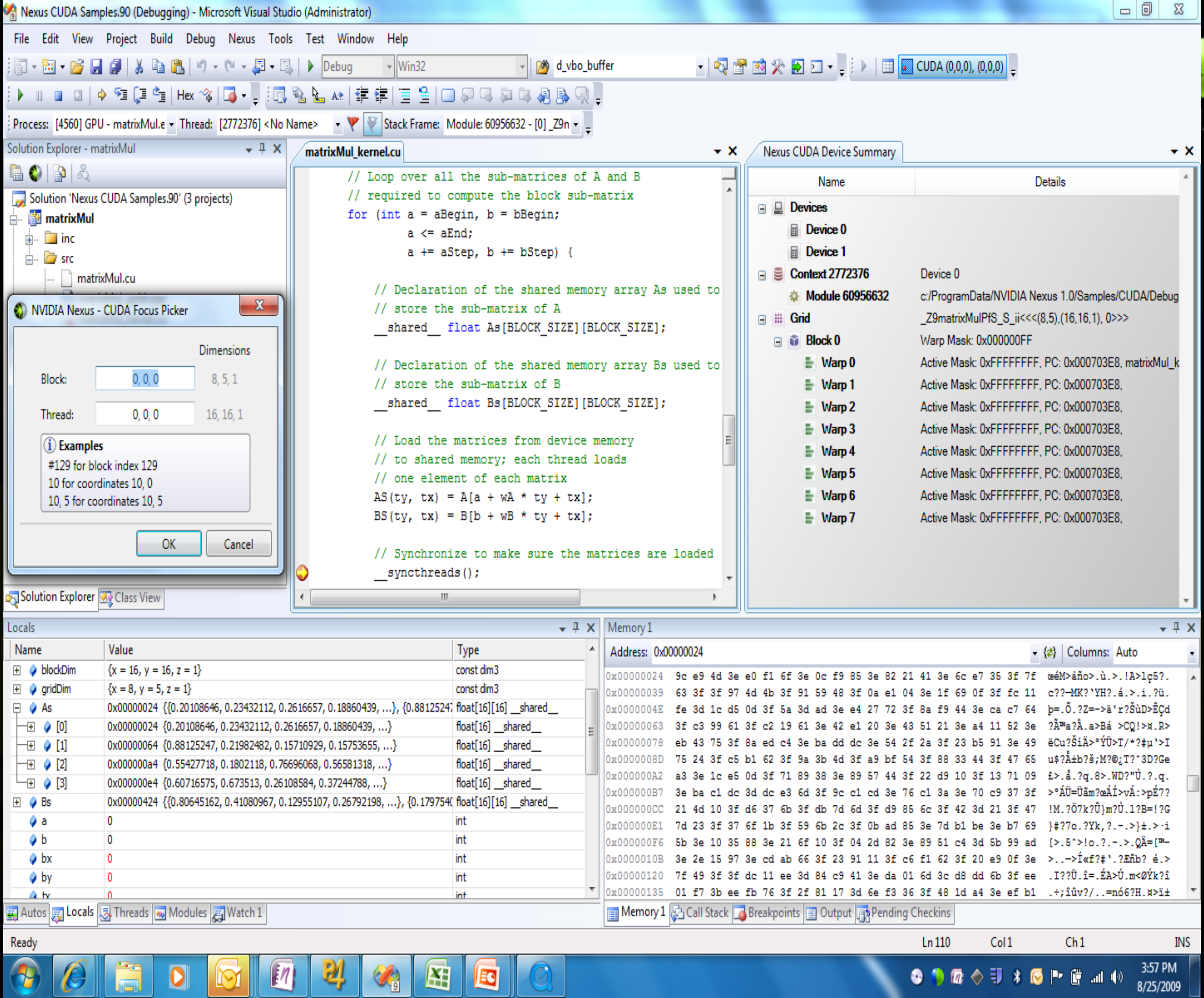
OEMs

DELL
hp
PENGUIN COMPUTING
CRAY
Sun
SUPERMICRO
sgi
FUJITSU
BULL
APPRO
lenovo联想
NEC

Ease of Programming



Source: Nicolas Pinto, MIT


Nexus CUDA Samples.90 (Debugging) - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Nexus Tools Test Window Help
Debug Win32 d_vbo_buffer CUDA (0,0,0), (0,0,0)
Hex
Process: [4560] GPU - matrixMul.e Thread: [2772376] <No Name> Stack Frame: Module: 60956632 - [0] _Z9n
Solution Explorer - matrixMul
Solution 'Nexus CUDA Samples.90' (3 projects)
matrixMul
inc
src
matrixMul.cu
Solution Explorer Class View
matrixMul_kernel.cu
Nexus CUDA Device Summary
Locals
Memory 1
Autos Locals Threads Modules Watch 1
Memory 1 Call Stack Breakpoints Output Pending Checkins
Ready Ln 110 Col 1 Ch 1 INS
3:57 PM 8/25/2009

NVIDIA Nexus - CUDA Focus Picker

		Dimensions
Block:	0, 0, 0	8, 5, 1
Thread:	0, 0, 0	16, 16, 1

Examples
#129 for block index 129
10 for coordinates 10, 0
10, 5 for coordinates 10, 5

OK Cancel

```
    // Loop over all the sub-matrices of A and B
    // required to compute the block sub-matrix
    for (int a = aBegin, b = bBegin;
             a <= aEnd;
             a += aStep, b += bStep) {

        // Declaration of the shared memory array As used to
        // store the sub-matrix of A
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

        // Declaration of the shared memory array Bs used to
        // store the sub-matrix of B
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        // Load the matrices from device memory
        // to shared memory; each thread loads
        // one element of each matrix
        AS(ty, tx) = A[a + wA * ty + tx];
        BS(ty, tx) = B[b + wB * ty + tx];

        // Synchronize to make sure the matrices are loaded
        __syncthreads();
```

Name	Details
Devices	
Device 0	
Device 1	
Context 2772376	Device 0
Module 60956632	c:/ProgramData/NVIDIA Nexus 1.0/Samples/CUDA/Debug
Grid	_Z9matrixMulPfS_S_ii<<<(8,5),(16,16,1), 0>>>
Block 0	Warp Mask: 0x000000FF
Warp 0	Active Mask: 0xFFFFFFFF, PC: 0x000703E8, matrixMul_k
Warp 1	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 2	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 3	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 4	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 5	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 6	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 7	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,

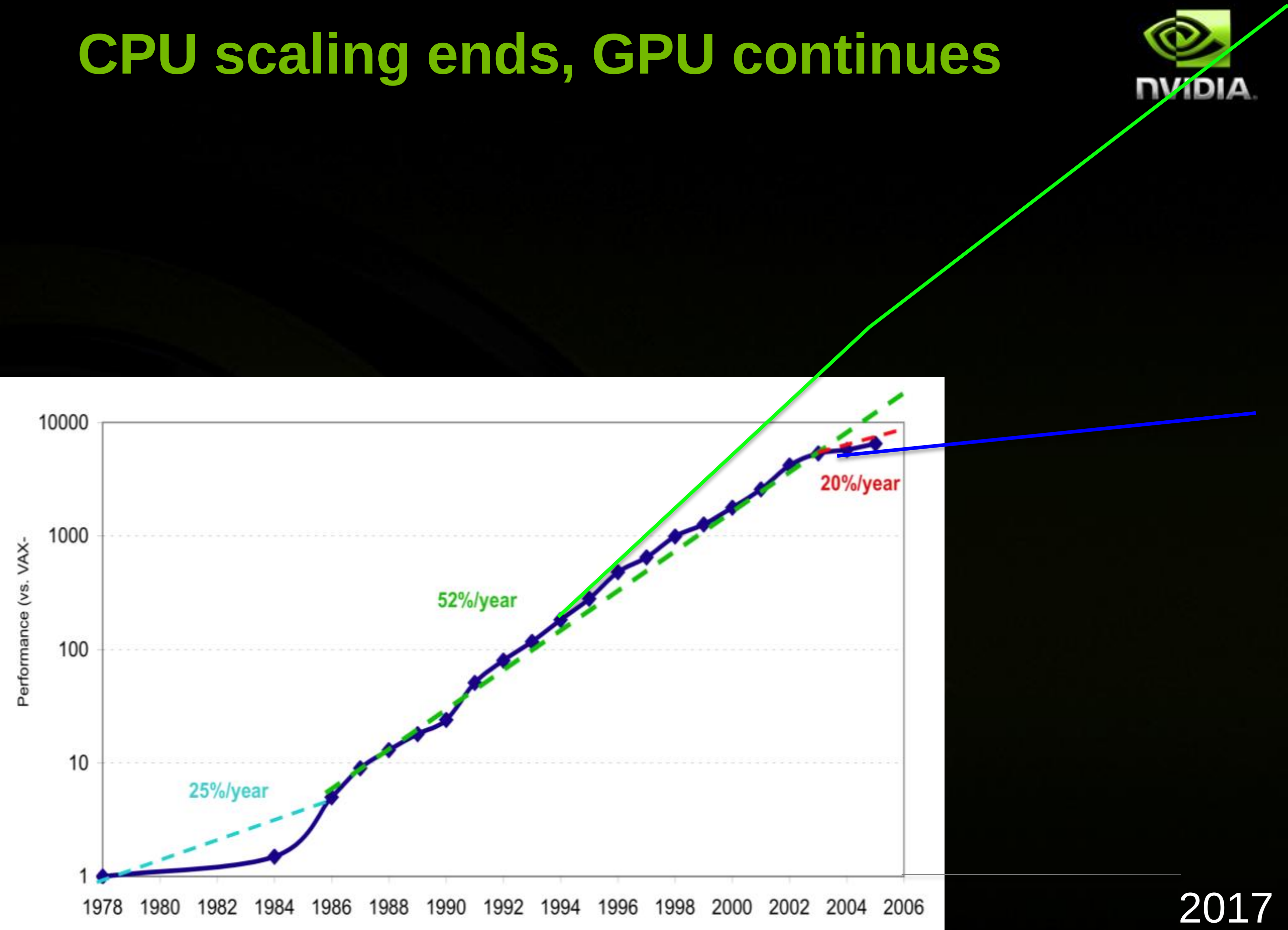
Name	Value	Type
blockDim	{x = 16, y = 16, z = 1}	const dim3
gridDim	{x = 8, y = 5, z = 1}	const dim3
As	0x00000024 {{0.20108646, 0.23432112, 0.2616657, 0.18860439, ...}, {0.8812524	float[16][16] __shared__
[0]	0x00000024 {0.20108646, 0.23432112, 0.2616657, 0.18860439, ...}	float[16] __shared__
[1]	0x00000064 {0.88125247, 0.21982482, 0.15710929, 0.15753655, ...}	float[16] __shared__
[2]	0x000000a4 {0.55427718, 0.1802118, 0.76696068, 0.56581318, ...}	float[16] __shared__
[3]	0x000000e4 {0.60716575, 0.673513, 0.26108584, 0.37244788, ...}	float[16] __shared__
Bs	0x00000424 {{0.80645162, 0.41080967, 0.12955107, 0.26792198, ...}, {0.179754	float[16][16] __shared__
a	0	int
b	0	int
bx	0	int
by	0	int
tx	0	int

Address: 0x00000024 Columns: Auto

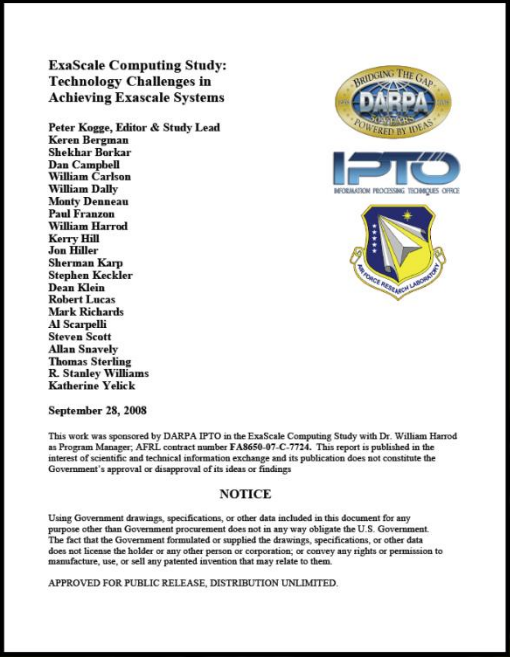
```
0x00000024  9c e9 4d 3e e0 f1 6f 3e 0c f9 85 3e 82 21 41 3e 6c e7 35 3f 7f
0x00000039  63 3f 3f 97 4d 4b 3f 91 59 48 3f 0a e1 04 3e 1f 69 0f 3f fc 11
0x0000004E  fe 3d 1c d5 0d 3f 5a 3d ad 3e e4 27 72 3f 8a f9 44 3e ca c7 64
0x00000063  3f c3 99 61 3f c2 19 61 3e 42 e1 20 3e 43 51 21 3e a4 11 52 3e
0x00000078  eb 43 75 3f 8a ed c4 3e ba dd dc 3e 54 2f 2a 3f 23 b5 91 3e 49
0x0000008D  75 24 3f c5 b1 62 3f 9a 3b 4d 3f a9 bf 54 3f 88 33 44 3f 47 65
0x000000A2  a3 3e 1c e5 0d 3f 71 89 38 3e 89 57 44 3f 22 d9 10 3f 13 71 09
0x000000B7  3e ba c1 dc 3d dc e3 6d 3f 9c c1 cd 3e 76 c1 3a 3e 70 c9 37 3f
0x000000CC  21 4d 10 3f d6 37 6b 3f db 7d 6d 3f d9 85 6c 3f 42 3d 21 3f 47
0x000000E1  7d 23 3f 37 6f 1b 3f 59 6b 2c 3f 0b ad 85 3e 7d b1 be 3e b7 69
0x000000F6  5b 3e 10 35 88 3e 21 6f 10 3f 04 2d 82 3e 89 51 c4 3d 5b 99 ad
0x0000010B  3e 2e 15 97 3e cd ab 66 3f 23 91 11 3f c6 f1 62 3f 20 e9 0f 3e
0x00000120  7f 49 3f 3f dc 11 ee 3d 84 c9 41 3e da 01 6d 3c d8 dd 6b 3f ee
0x00000135  01 f7 3b ee fb 76 3f 2f 81 17 3d 6e f3 36 3f 48 1d a4 3e ef b1
```

The future is even more parallel

CPU scaling ends, GPU continues



DARPA Study Indentifies four challenges for ExaScale Computing



ExaScale Computing Study:
Technology Challenges in
Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

BRIDGING THE GAP
DARPA
POWERED BY IDEAS
IPTO
INFORMATION PROCESSING TECHNIQUES OFFICE
AIR FORCE RESEARCH LABORATORY

Report published September 28, 2008:

- Four Major Challenges
  - Energy and Power challenge
  - Memory and Storage challenge
  - Concurrency and Locality challenge
  - Resiliency challenge
- Number one issue is *power*
  - Extrapolations of current architectures and technology indicate over 100MW for an Exaflop!
  - Power also constrains what we can put on a chip

Available at
www.darpa.mil/ipto/personnel/docs/ExaScale_Study_Initial.pdf

Energy and Power Challenge

- Heterogeneous architecture
  - A few latency-optimized processors
  - Many (100s-1,000s) throughput-optimized processors
    - Which are optimized for ops/J
- Efficient processor architecture
  - Simple control – in-order multi-threaded
  - SIMT execution to amortize overhead
- Agile memory system to capture locality
  - Keeps data and instruction access local
- Optimized circuit design
  - Minimize energy/op
  - Minimize cost of data movement

* This section is a projection based on Moore's law and does not represent a committed roadmap

An NVIDIA ExaScale Machine in 2017

- 2017 GPU Node – 300W (GPU + memory + supply)
  - 2,400 throughput cores (7,200 FPUs), 16 CPUs – single chip
  - 40TFLOPS (SP) 13TFLOPS (DP)
  - Deep, explicit on-chip storage hierarchy
  - Fast communication and synchronization
- Node Memory
  - 128GB DRAM, 2TB/s bandwidth
  - 512GB Phase-change/Flash for checkpoint and scratch
- Cabinet – ~100kW
  - 384 Nodes – 15.7PFLOPS (SP), 50TB DRAM
  - Dragonfly network – 1TB/s per node bandwidth
- System – ~10MW
  - 128 Cabinets – 2 EFLOPS (SP), 6.4 PB DRAM
  - Distributed EB disk array for file system
  - Dragonfly network with active optical links
- RAS
  - ECC on all memory and links
  - Option to pair cores for self-checking (or use application-level checking)
  - Fast local checkpoint

* This section is a projection based on Moore's law and does not represent a committed roadmap

Conclusion

Performance = Parallelism

Efficiency = Locality

Applications have lots of both.

GPUs have lots of cores (parallelism) and an exposed storage hierarchy (locality)