the SILK ROAD PROJECT at NAOC/
丝绸之路计划

ZAH
ARI ITA LSW

ZENTRUM FÜR
ASTRONOMIE
Univ. Heidelberg

UNIVERSITÄT
HEIDELBERG
Zukunft. Seit 1386.

# Introduction to GPU Accelerated Computing

Rainer Spurzem
Astronomisches Rechen-Inst., ZAH, Univ. of Heidelberg, Germany
National Astronomical Observatories (NAOC), Chinese Academy of Sciences
Kavli Institute for Astronomy and Astrophysics (KIAA), Peking University
https://astro-silkroad.eu
https://wwwstaff.ari.uni-heidelberg.de/spurzem/

VolkswagenStiftung

spurzem@ari.uni-heidelberg.de
spurzem@nao.cas.cn

Picture: Xishuangbanna,
Yunnan, China by R.Sp.

中国科学院国家天文台
National Astronomical Observatories, CAS

北京大学
PEKING UNIVERSITY

# Introduction to GPU Accelerated Computing
# Feb. 17-21, 2025

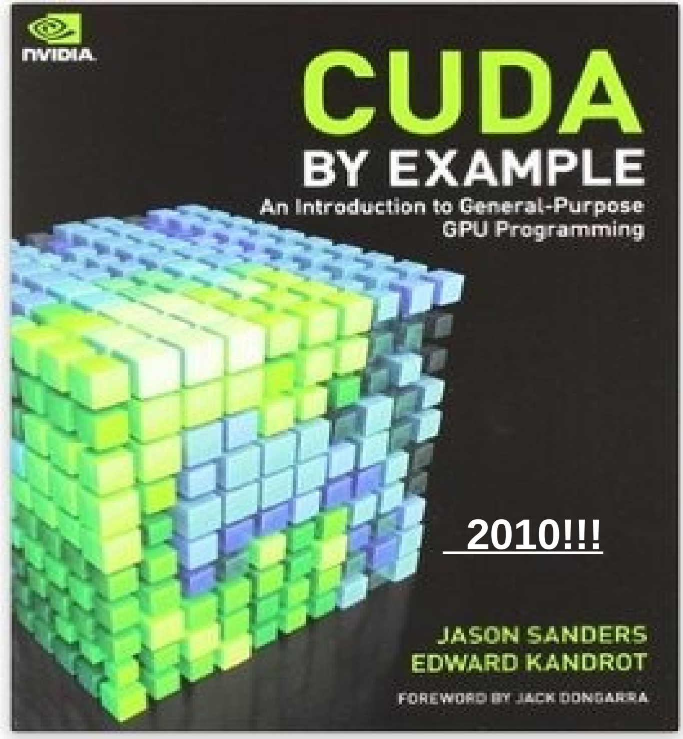
**Table of Contents (subject to adjustment/change):**

1. Monday morning 1: General Introduction Computer Architecture, Many-Core, GPU and others…, Access...
2. Monday morning 2/afternoon: Access to bwUniCluster, CUDA Hello, GPU Properties, First CUDA Scalar, Simple Vector Add
3. Tuesday morning 1: More on GPU Software and Hardware
4. Tuesday morning 2/afternoon: CUDA Vector Add, Scalar Products, Using Blocks and Threads
5. Wednesday morning: Parallelization and Amdahl's Law, GPU Acceleration, Future Architecture
6. Wednesday morning 2/afternoon: CUDA Scalar Products cont’d Events, Histograms, Matrix Multiplication
7. Thursday Morning: Astrophysical N-Body Code
8. Thursday Afternoon: Astrophysical Parallel N-Body Code Using MPI and GPU
9. Friday Morning: CUDA Matrix Mult., Histograms, Wrap-Up, Q+A, Other Lectures (Wen-Mei Hwu)

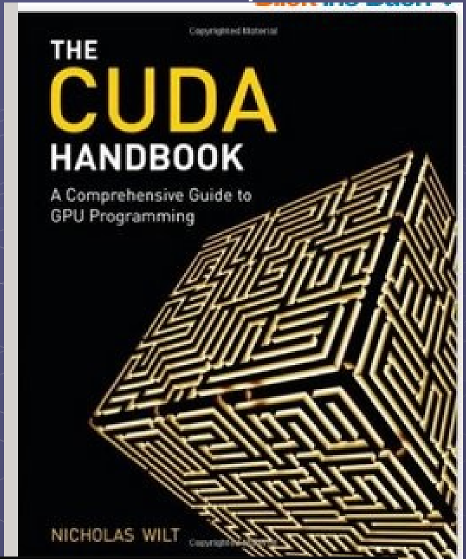
**Literature:** why NVIDIA? CUDA … ?

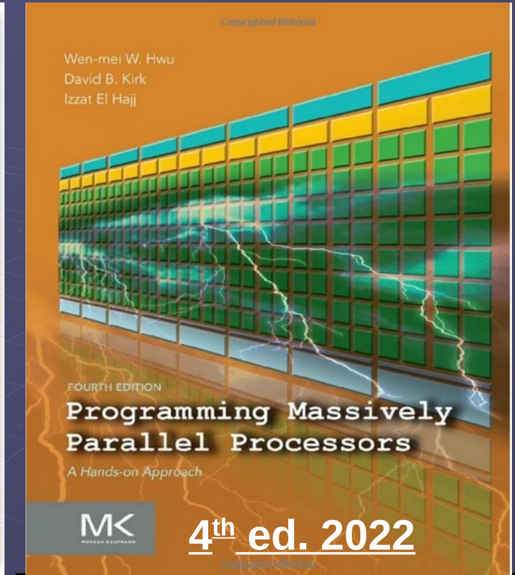
easy to learn!! runs on our training system kepler

future? SYCL/openCL? HIP / HIPIFY ?

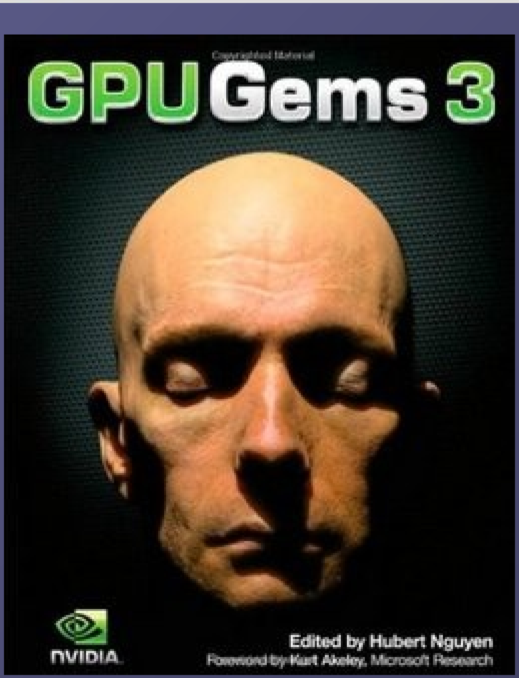

NVIDIA
CUDA
BY EXAMPLE
An Introduction to General-Purpose GPU Programming
JASON SANDERS
EDWARD KANDROT
FOREWORD BY JACK DONGARRA

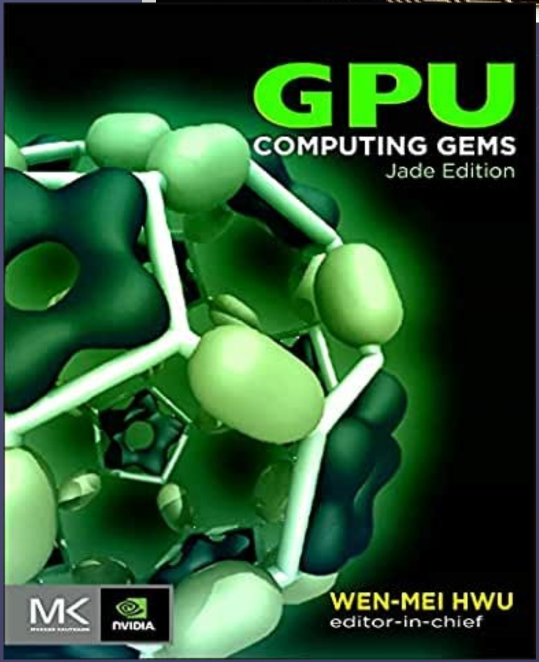
2010!!!

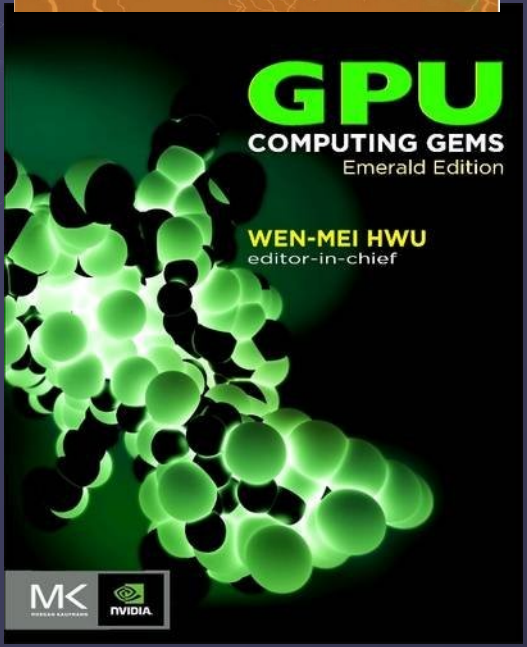




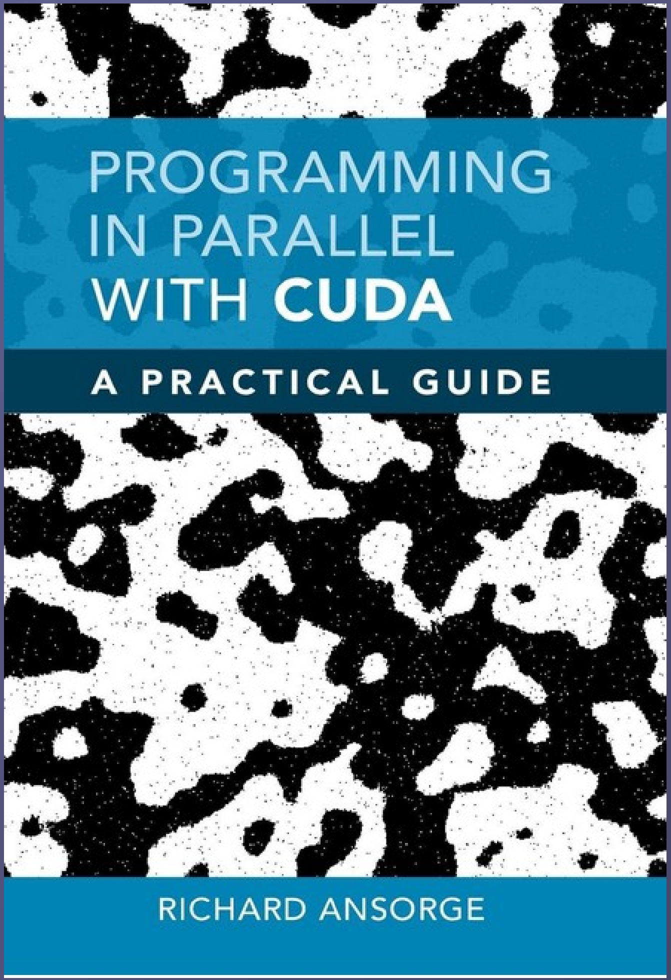
4th ed. 2022







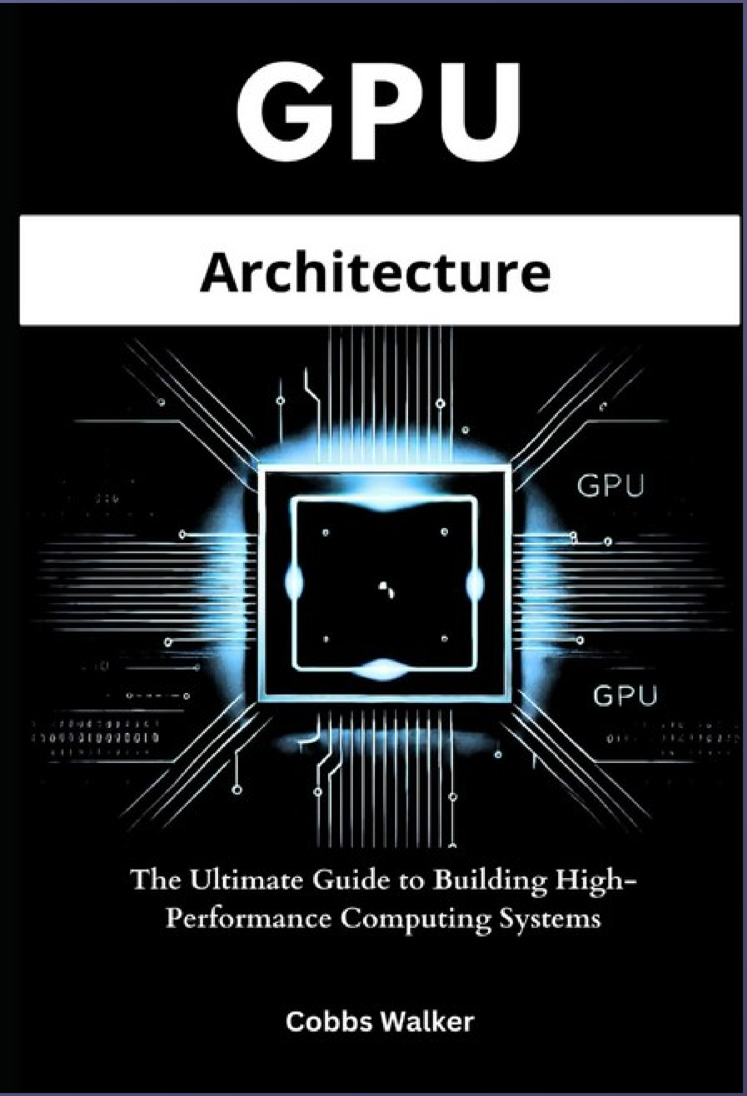
# Literature continued:



"CUDA is now the dominant language used for programming GPUs, one of the most exciting hardware developments of recent decades. With CUDA, you can use a desktop PC for work that would have previously required a large cluster of PCs or access to a HPC facility. As a result, CUDA is increasingly important in scientific and technical computing across the whole STEM community, from medical physics and financial modelling to big data applications and beyond. This unique book on CUDA draws on the author's passion for and long experience of developing and using computers to acquire and analyse scientific data. The result is an innovative text featuring a much richer set of examples than found in any other comparable book on GPU computing. Much attention has been paid to the C++ coding style, which is compact, elegant and efficient. A code base of examples and supporting material is available online, which readers can build on for their own projects"--

New Book of 2022, Text from Book advertisement in amazon.

# Literature continued:


GPU
Architecture
GPU
GPU
The Ultimate Guide to Building High-Performance Computing Systems
Cobbs Walker

“GPU systems have revolutionized the fields of artificial intelligence, data science, and high-performance computing. Their unparalleled ability to handle massive parallel processing tasks has made them indispensable for industries that rely on cutting-edge computational power. From AI model training to scientific simulations and beyond, understanding how to design and optimize GPU architectures is key to maximizing performance and staying ahead in a rapidly evolving tech landscape.”

“Authored by a high-performance computing expert, (Cobbs Walker) provides the most up-to-date, actionable insights on GPU system design. This book is based on years of hands-on experience building and optimizing GPU infrastructures, paired with real-world case studies that demonstrate successful implementations. Whether you're designing AI systems, working on complex simulations, or building GPU-driven applications, the expertise shared here is reliable and practical.”

For deeper interest in GPU hardware.

Text from Book advertisement in amazon.

# Introduction to GPU Accelerated Computing
# Feb. 17-21, 2025

**“Table of Contents” what we will NOT cover:**

- Artificial Intelligence / Machine Learning
- Graphics Rendering / Ray Tracing with GPU
- Using Tensor Cores for 3D simulations
- Other Languages such as HIP (AMD), OpenCL…

We will solely use CUDA for High Performance Computing on GPU (many simple examples, one real Application).

What you learn here will give you a good start for all the applications not covered!

# GPU Computing

# History

# History

Erik Holmberg (1908-2000)

Dissertation Univ. Lund (Schweden) (1937):

``A study of double and multiple galaxies´´

Galaxies  often in Groups and Pairs

Irregular Distribution of Satellite Galaxies

(Holmberg-Effect)

**Father of numerical astrophysics?**

» **...with 200 light bulbs**

# History

http://cdsads.u-strasbg.fr/abs/1941ApJ....94..385H

The Astrophysical Journal, Nov. 1941

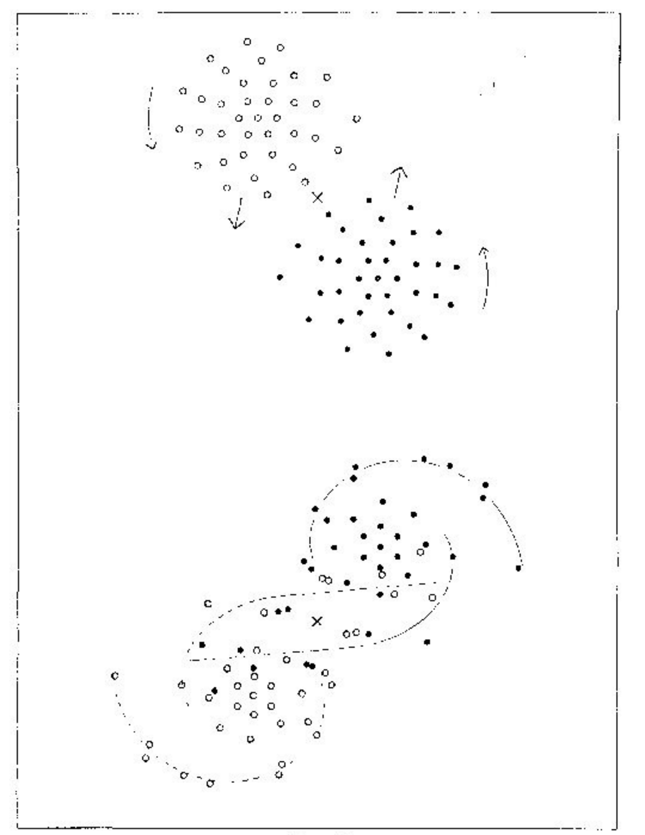


FIG. 4b

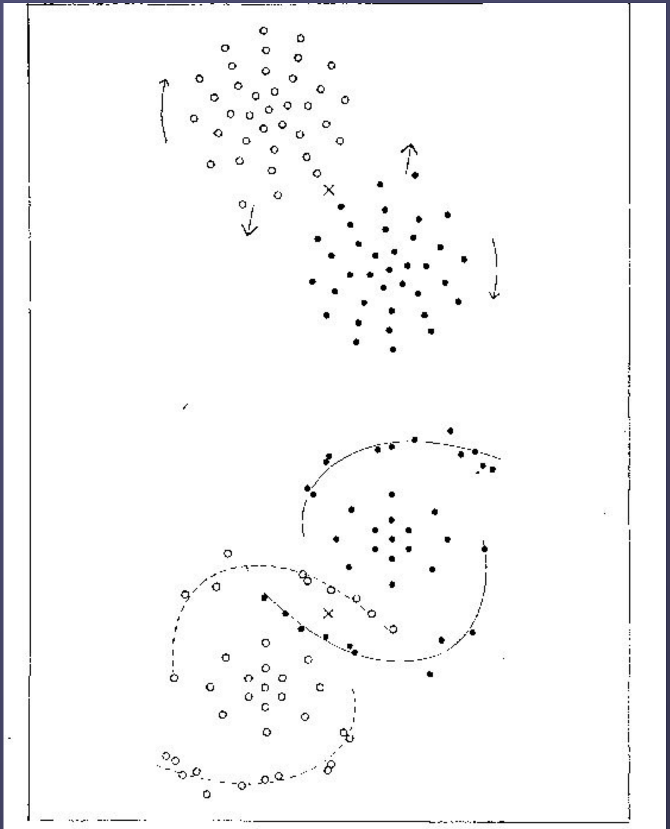


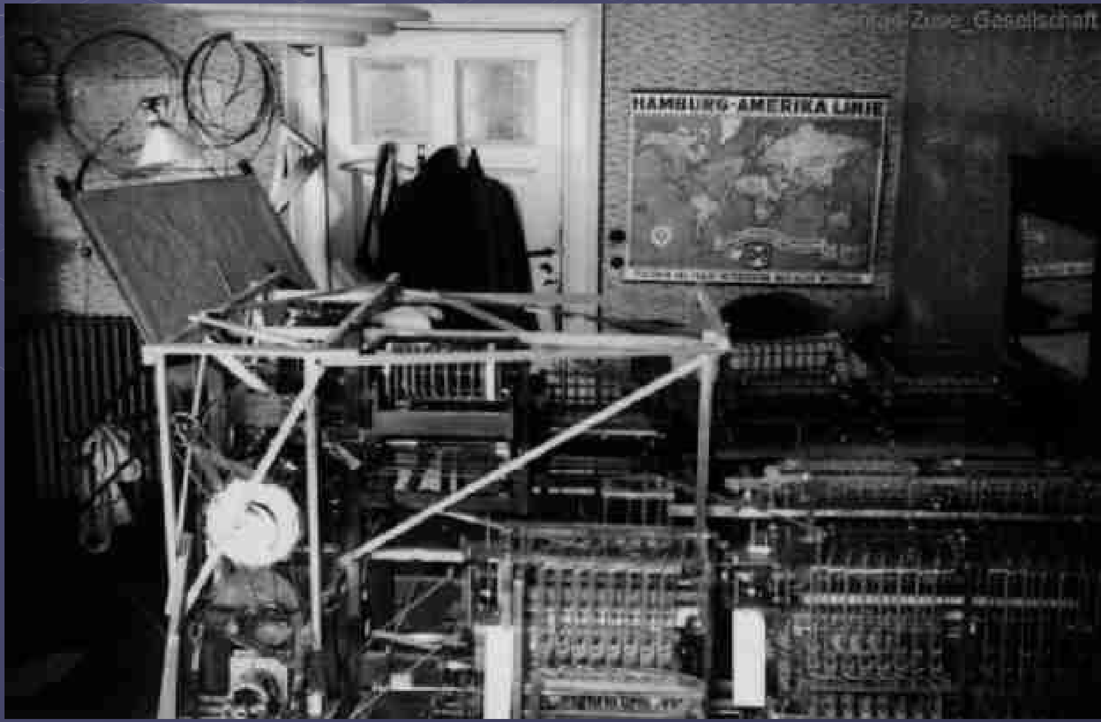
FIG. 4a

# HARDWARE

...before von Neumann...

- Konrad Zuse (1910-1995) Berlin

Invented freely programmable Computer

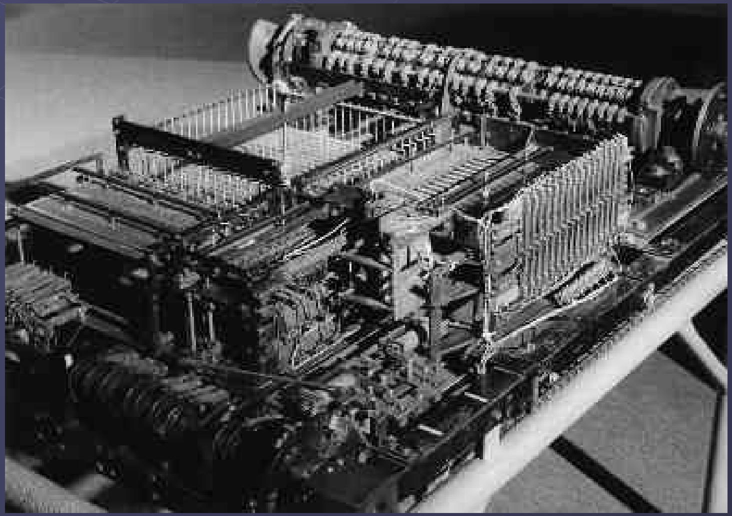


Z1 in parental flat 1936

# History

**Zuse Z4: 1944 Berlin, 1950 Zürich, 1954 Frankreich**

**1959 Deutsches Museum München**



**Computing Speed 0.03 MHz**

**Memory 256 byte**

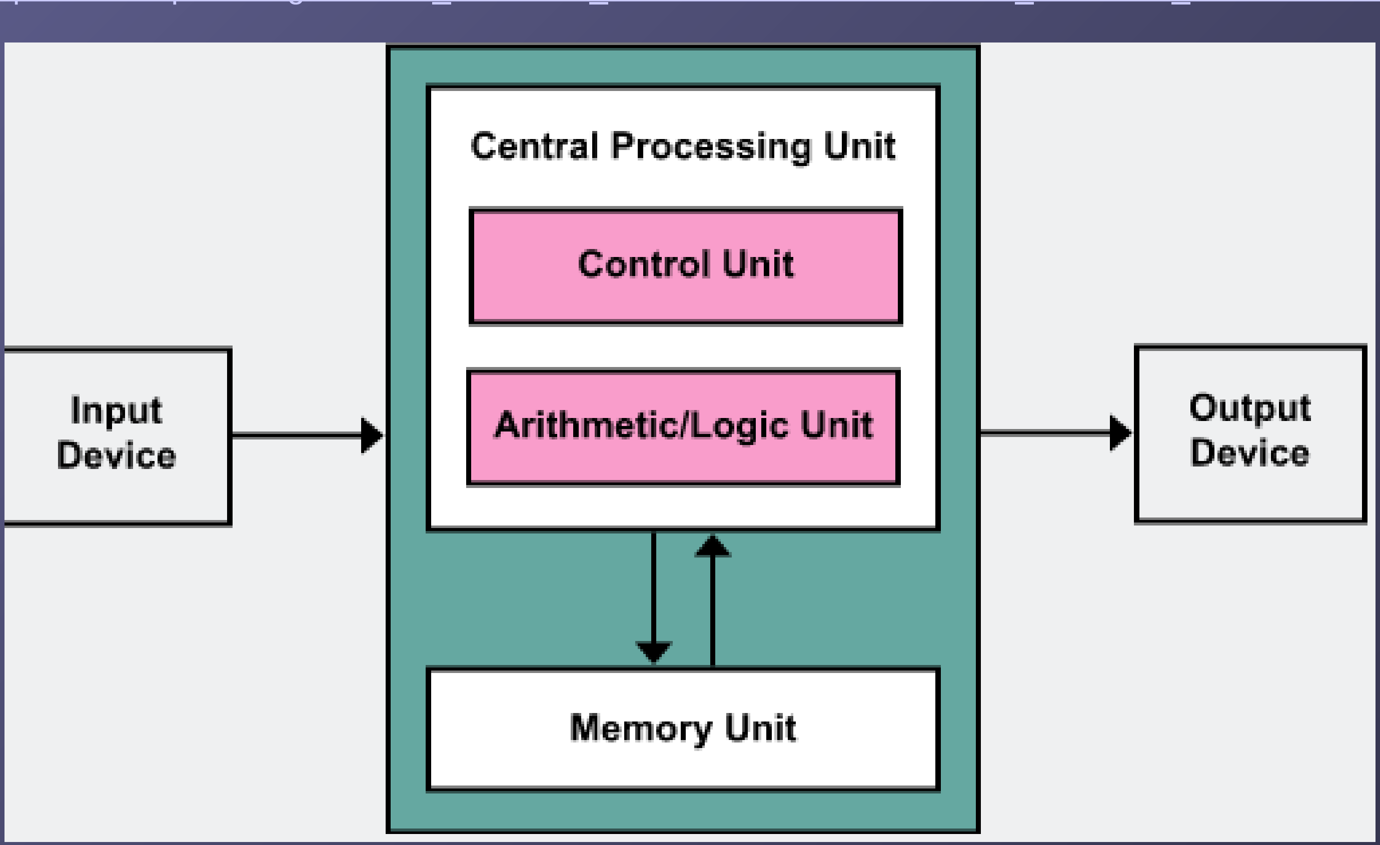
# HARDWARE

- John von Neumann (1903-1957)

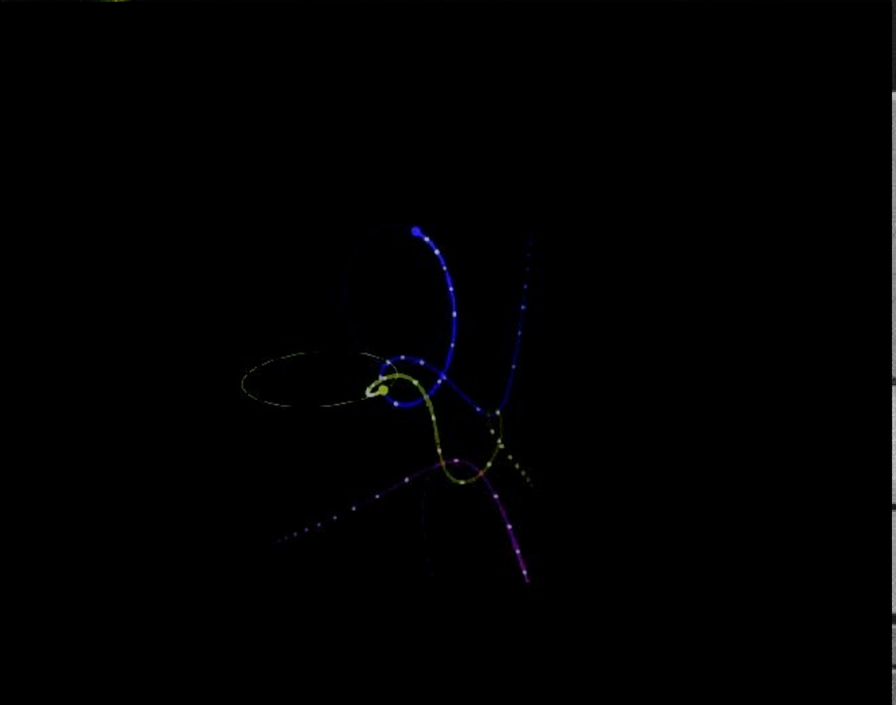
Born Budapest, Lecturer Berlin, since 1930 Princeton Univ.

Fundamental Architecture of an electronic computing device(1946)

Source: https://en.wikipedia.org/wiki/Von_Neumann_architecture#/media/File:Von_Neumann_Architecture.svg



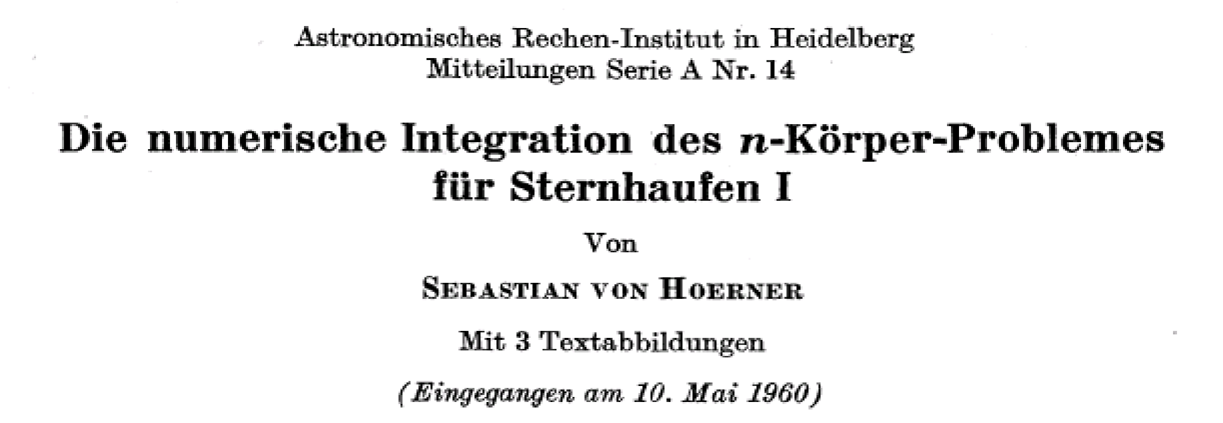
Astronomisches Rechen-Institut (ARI) at Univ. of Heidelberg, Germany



**Siemens 2002**
**Computer in 1964**
**At ARI**

# History

http://cdsads.u-strasbg.fr/abs/1960ZA.....50..184V



Astronomisches Rechen-Institut in Heidelberg
Mitteilungen Serie A Nr. 14

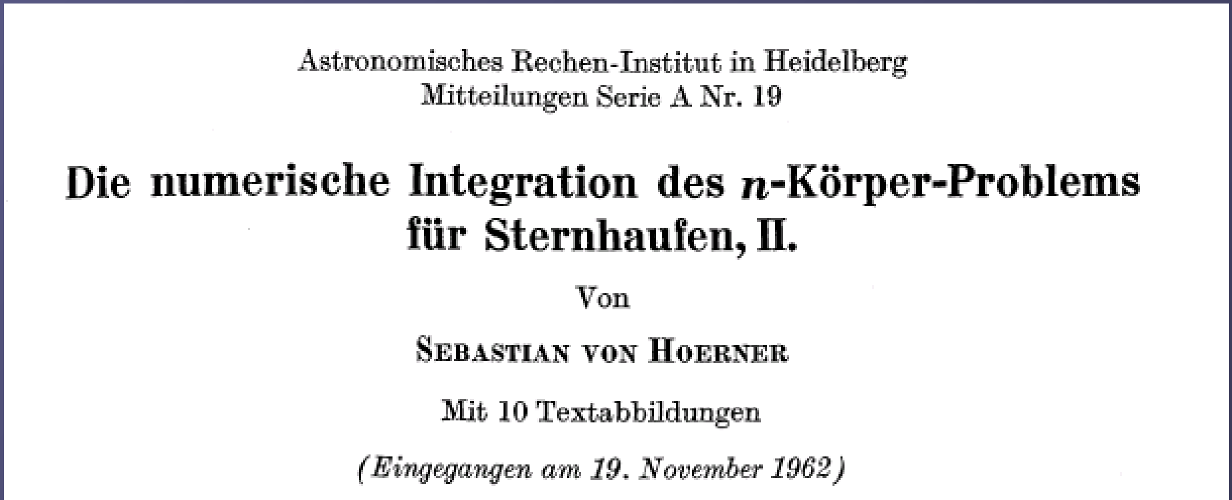
**Die numerische Integration des *n*-Körper-Problemes für Sternhaufen I**

Von

**Sebastian von Hoerner**

Mit 3 Textabbildungen

*(Eingegangen am 10. Mai 1960)*



Astronomisches Rechen-Institut in Heidelberg
Mitteilungen Serie A Nr. 19

**Die numerische Integration des *n*-Körper-Problems für Sternhaufen, II.**

Von

**Sebastian von Hoerner**

Mit 10 Textabbildungen

*(Eingegangen am 19. November 1962)*

http://cdsads.u-strasbg.fr/abs/1963ZA.....57...47V

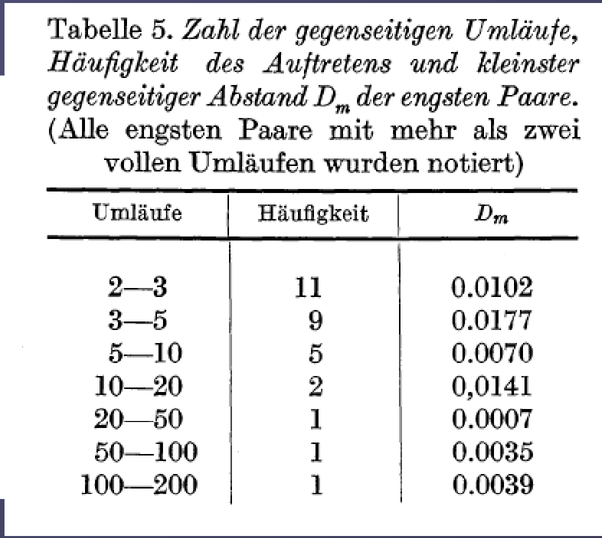


Tabelle 5. *Zahl der gegenseitigen Umläufe, Häufigkeit des Auftretens und kleinster gegenseitiger Abstand $D_m$ der engsten Paare.* (Alle engsten Paare mit mehr als zwei vollen Umläufen wurden notiert)

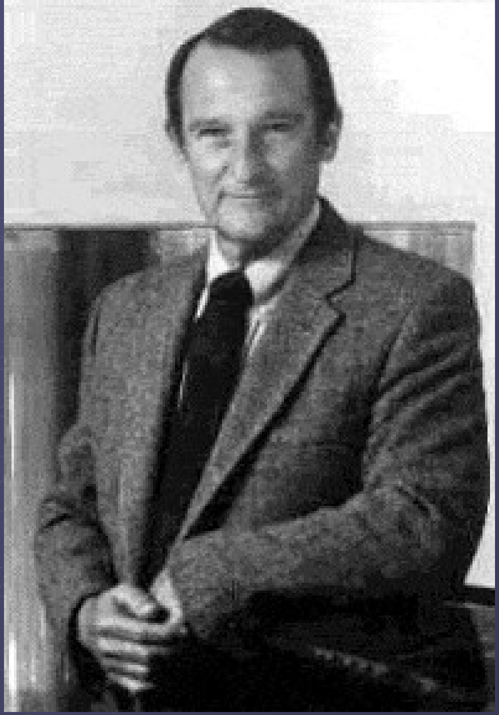
| Umläufe | Häufigkeit | $D_m$ |
|---|---|---|
| 2—3 | 11 | 0.0102 |
| 3—5 | 9 | 0.0177 |
| 5—10 | 5 | 0.0070 |
| 10—20 | 2 | 0,0141 |
| 20—50 | 1 | 0.0007 |
| 50—100 | 1 | 0.0035 |
| 100—200 | 1 | 0.0039 |

S.v. Hoerner,
Z.f.Astroph. 1960, 63

Siemens 2002
N=4,8,12,16 (4 Trx)

N=16,25 (40 Trx)

# History



- Seymour Cray (1925-1996)

“father of supercomputing”

https://en.wikipedia.org/wiki/Women_in_computing

**CRAY1: Vectorregisters (1976)**

**160 Mflop, 80 MHz, 8 MByte RAM**

**CRAY2: (1984)**

**1Gflop, 120MHz, 2GByte RAM**

# History

*Supercomputer*

*JUGENE*

*IBM Blue Gene*

*At FZ Jülich,*

*Germany*

*Opening Ceremony June 2008*

# Computational Science...

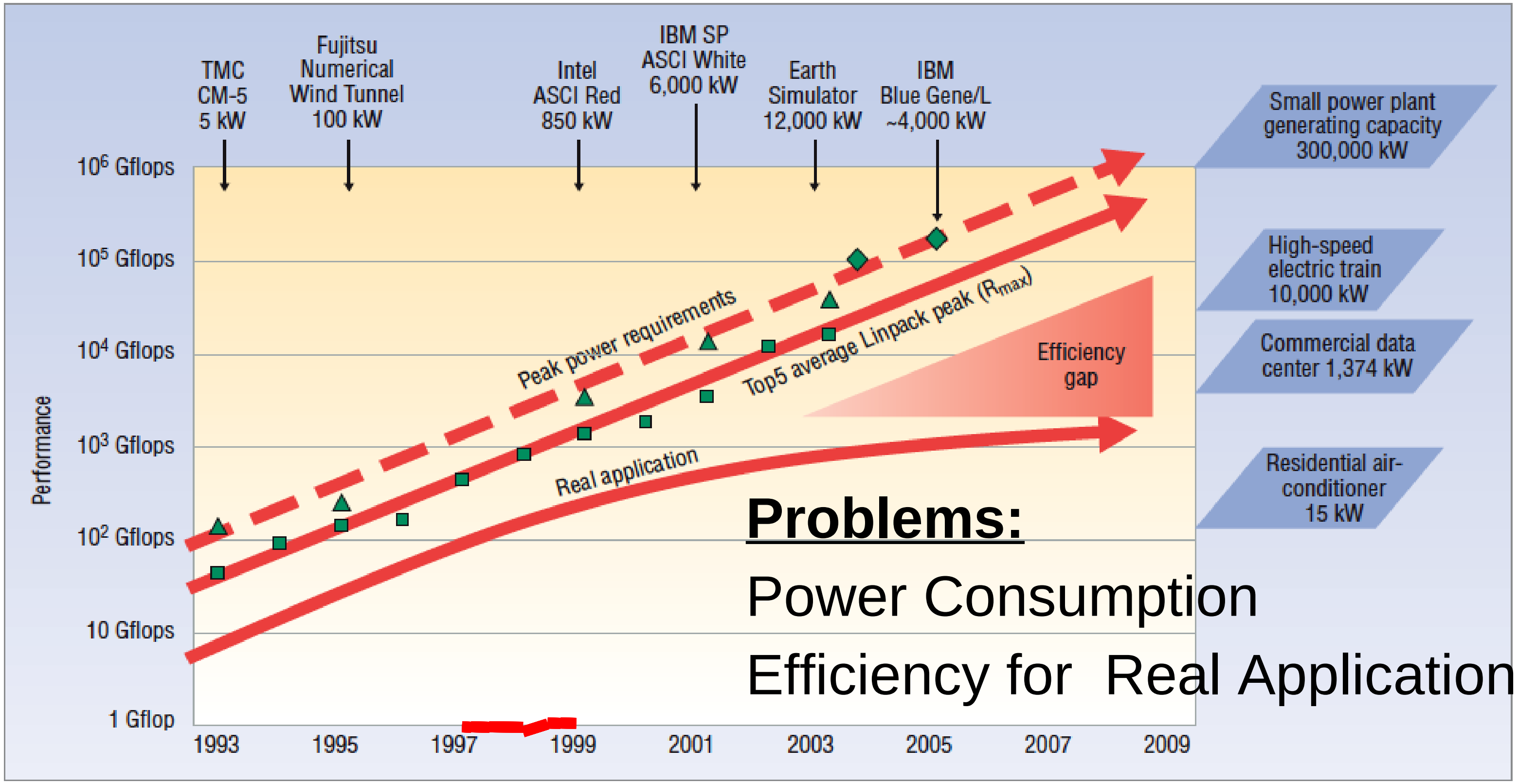
...after von Neumann...

Exaflop/s

Petaflop/s

Teraflop/s

Gigaflop/s



**Problems:**

Power Consumption

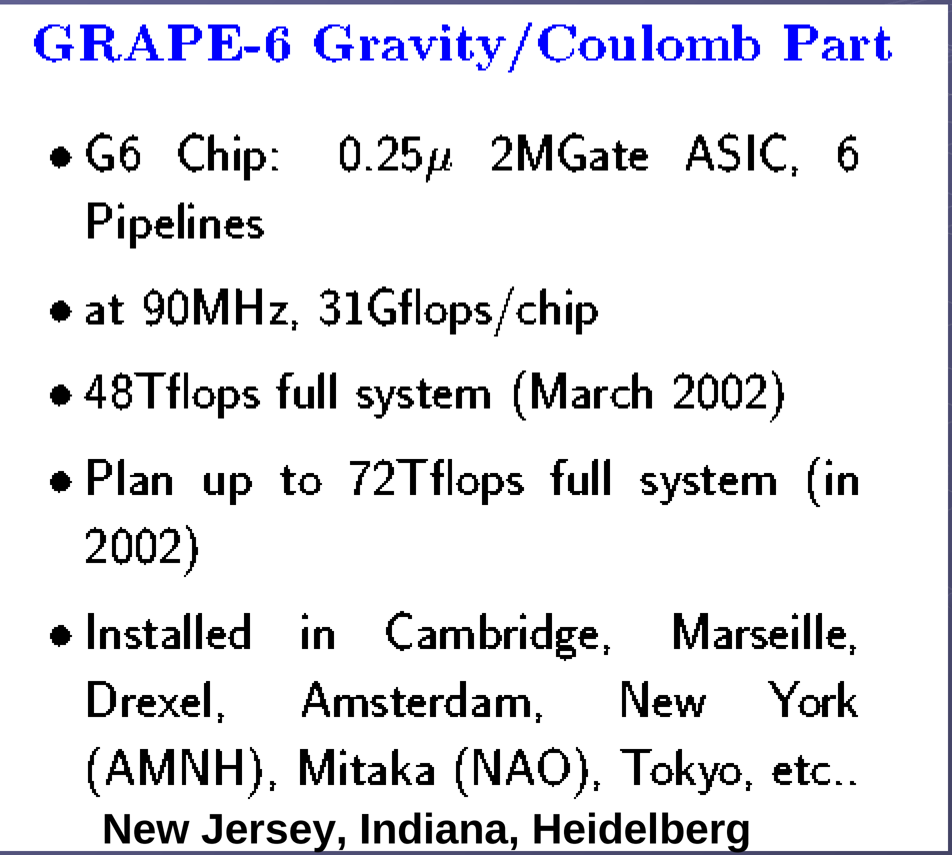
Efficiency for  Real Applications

***Figure 1. Rising power requirements. Peak power consumption of the top supercomputers has steadily increased over the past 15 years.***

Thanks to Horst Simon, LBNL/NERSC for this diagram.

# Special Hardware Accelerators

# HARDWARE



## GRAPE-6 Gravity/Coulomb Part

- G6 Chip: 0.25$\mu$ 2MGate ASIC, 6 Pipelines
- at 90MHz, 31Gflops/chip
- 48Tflops full system (March 2002)
- Plan up to 72Tflops full system (in 2002)
- Installed in Cambridge, Marseille, Drexel, Amsterdam, New York (AMNH), Mitaka (NAO), Tokyo, etc..

  **New Jersey, Indiana, Heidelberg**

# GPU: NAOC laohu cluster Beijing, China

2010

Peter Berczik

*Silk Road Team*

# BwUniCluster 2.0

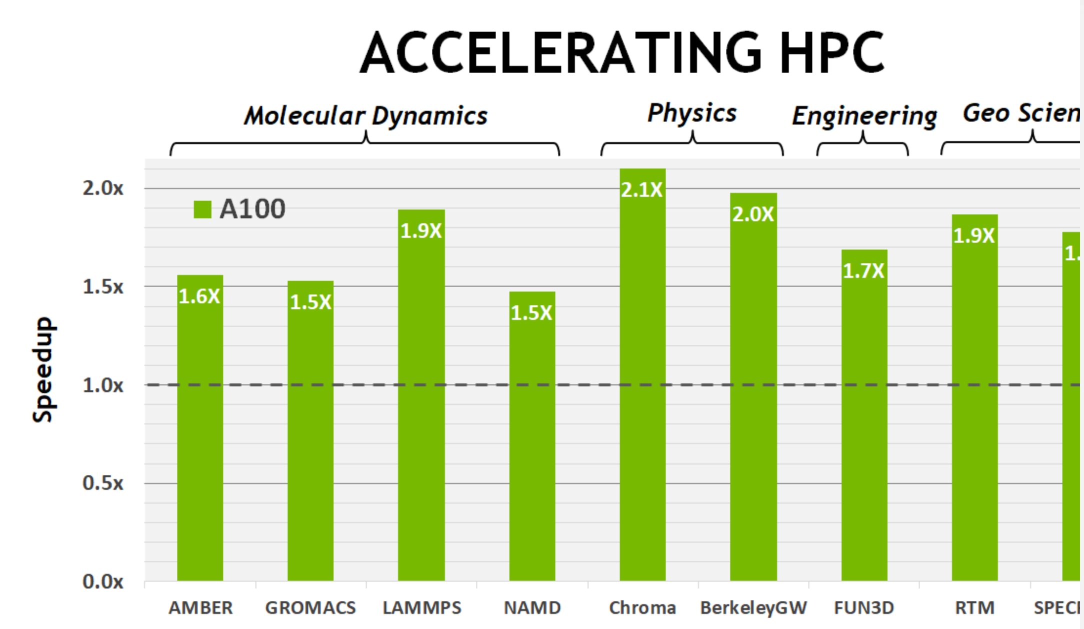
The bwUniCluster 2.0 is the joint high-performance computer system of Baden-Württemberg's Universities and Universities of Applied Sciences for general purpose and teaching and located at the Scientific Computing Center (SCC) at Karlsruhe Institute of Technology (KIT). The bwUniCluster 2.0 complements the four bwForClusters and their dedicated scientific areas.

https://wiki.bwhpc.de/e/BwUniCluster2.0

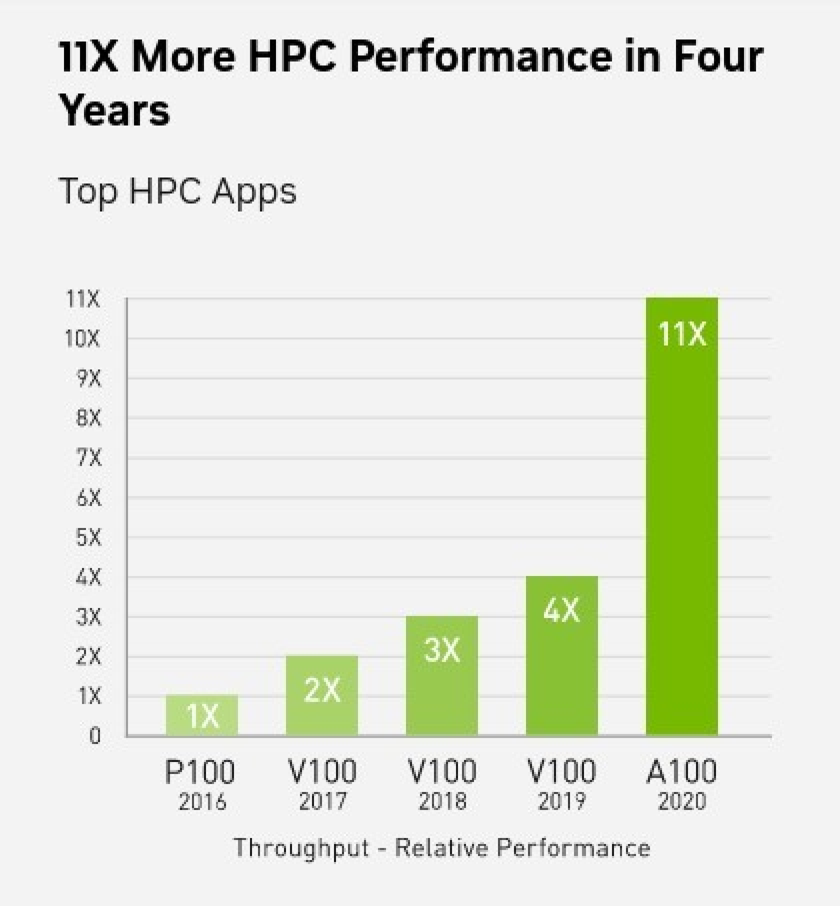
Total Number of Nodes: 848

GPU Nodes: 39 (NVIDIA Ampére A100, Volta V100)

# *NVIDIA Ampere A100 GPU, 54 billion transistors, 6920 cores*

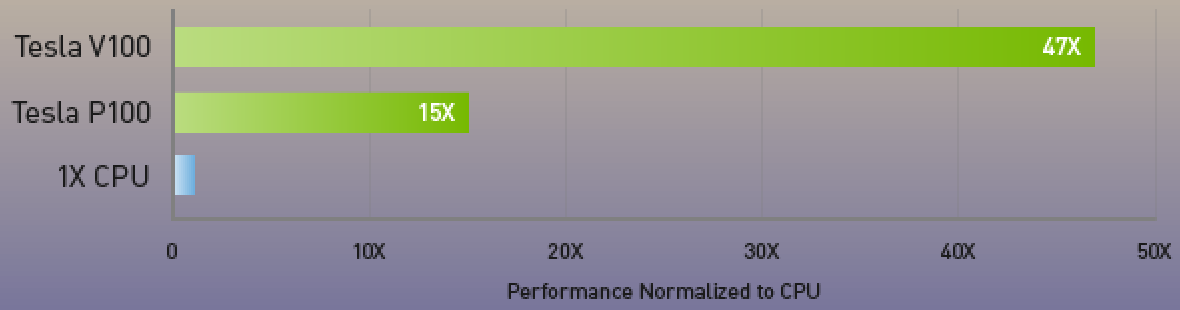


All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE
Chroma with szscl21_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model
BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 in DGX A100



# *NVIDIA Volta V100 GPU, 21 billion transistors, 5120 cores*

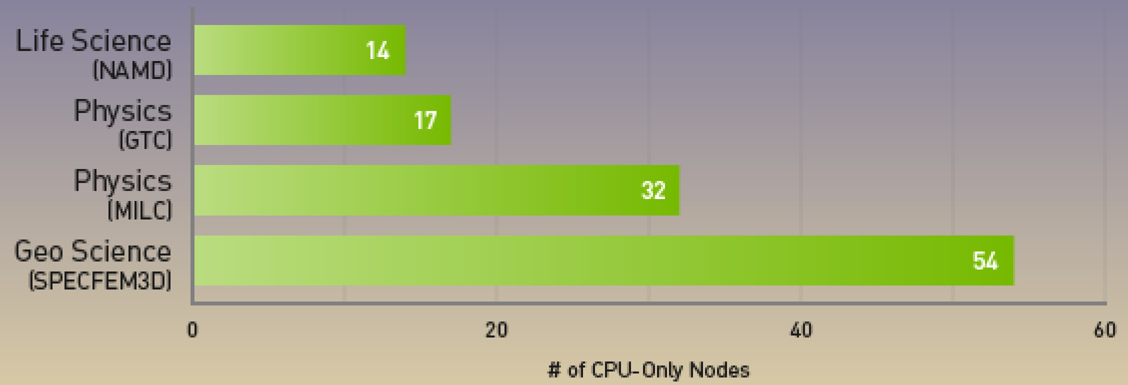
## 47X Higher Throughput Than CPU Server on Deep Learning Inference



Workload: ResNet-50 | CPU: 1X Xeon E5-2690v4 @ 2.6 GHz | GPU: Add 1X Tesla P100 or V100

## 1 GPU Node Replaces Up To 54 CPU Nodes

Node Replacement: HPC Mixed Workload

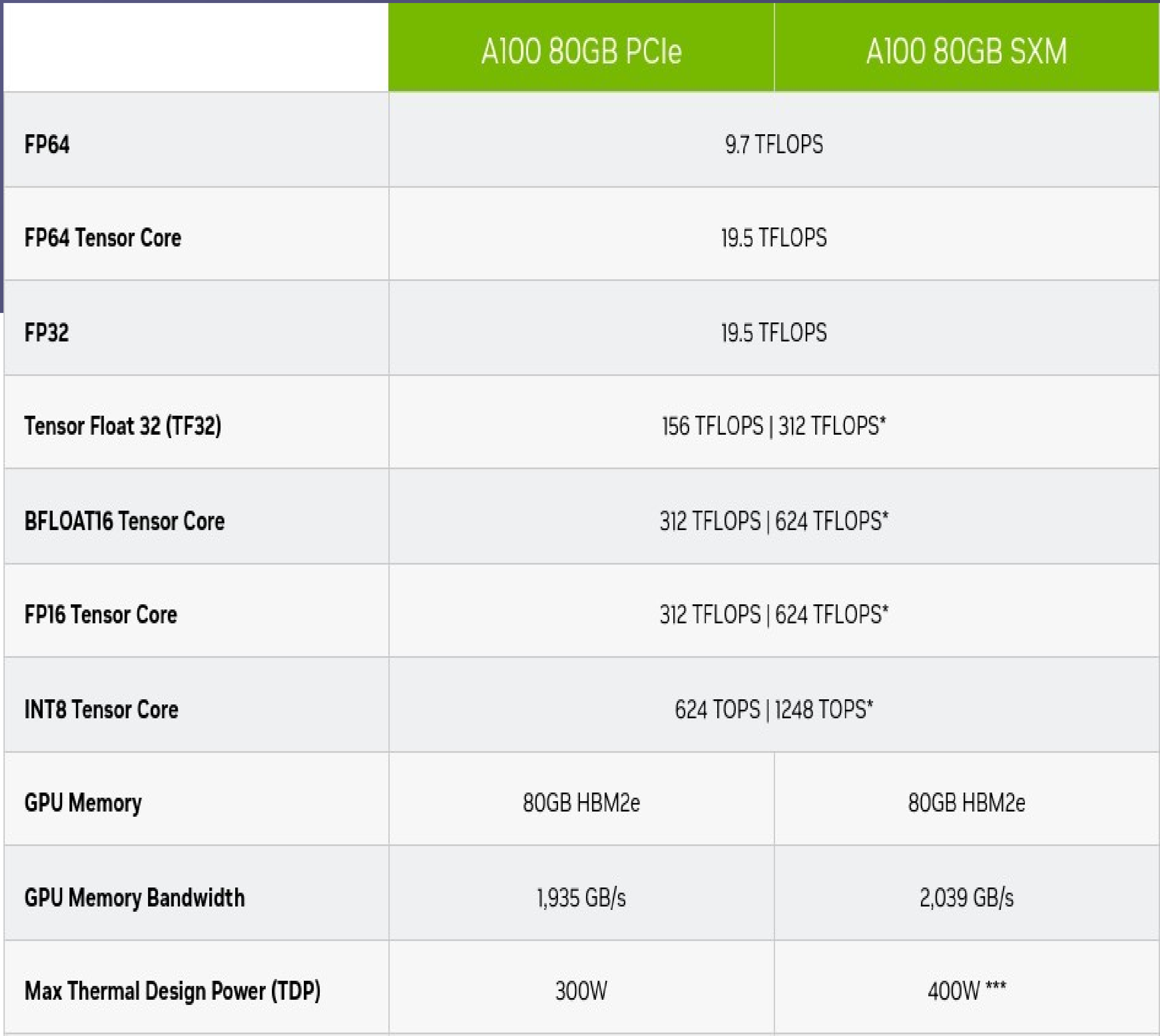


CPU Server: Dual Xeon Gold 6140@2.30GHz, GPU Servers: same CPU server w/ 4x V100 PCIe | CUDA Version: CUDA 9.x| Dataset: NAMD (STMV), GTC (mpi#proc.in), MILC (APEX Medium), SPECFEM3D (four_material_simple_model) | To arrive at CPU node equivalence, we use measured benchmark with up to 8 CPU nodes. Then we use linear scaling to scale beyond 8 nodes.

# *NVIDIA Ampere A100 GPU, 54 billion transistors, 6920 cores (Hopper H100, ...)*

With NVLINK

Without NVLINK

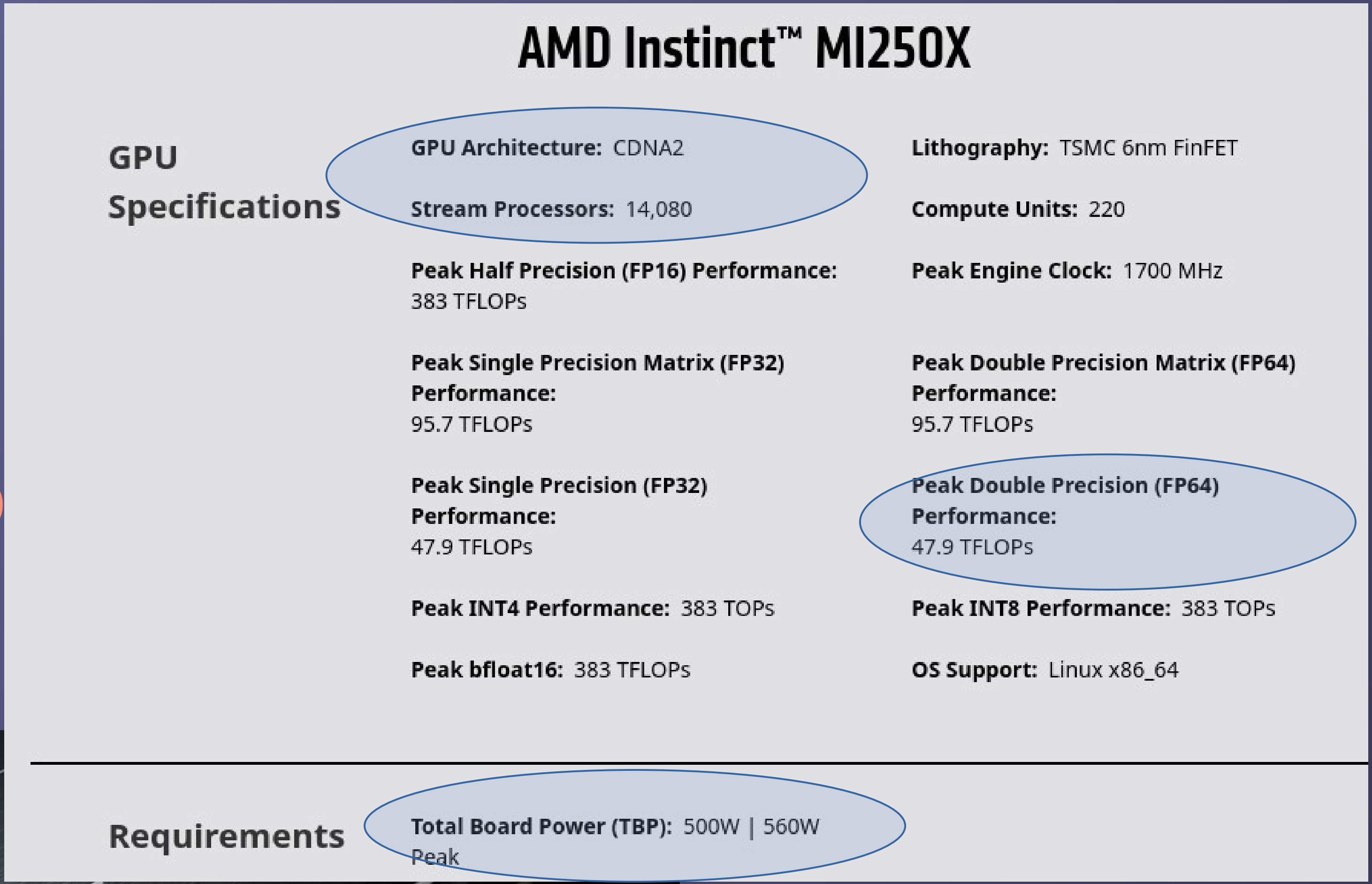


| | A100 80GB PCIe | A100 80GB SXM |
|---|---|---|
| **FP64** | 9.7 TFLOPS | |
| **FP64 Tensor Core** | 19.5 TFLOPS | |
| **FP32** | 19.5 TFLOPS | |
| **Tensor Float 32 (TF32)** | 156 TFLOPS \| 312 TFLOPS* | |
| **BFLOAT16 Tensor Core** | 312 TFLOPS \| 624 TFLOPS* | |
| **FP16 Tensor Core** | 312 TFLOPS \| 624 TFLOPS* | |
| **INT8 Tensor Core** | 624 TOPS \| 1248 TOPS* | |
| **GPU Memory** | 80GB HBM2e | 80GB HBM2e |
| **GPU Memory Bandwidth** | 1,935 GB/s | 2,039 GB/s |
| **Max Thermal Design Power (TDP)** | 300W | 400W *** |

AMD Instinct MI250X GPU

Nov.2023 Lists:
Used in:

Frontier (#1 US)
And LUMI (#5 FIN)



# AMD Instinct™ MI250X

## GPU Specifications

**GPU Architecture:** CDNA2

**Stream Processors:** 14,080

**Peak Half Precision (FP16) Performance:** 383 TFLOPs

**Peak Single Precision Matrix (FP32) Performance:** 95.7 TFLOPs

**Peak Single Precision (FP32) Performance:** 47.9 TFLOPs

**Peak INT4 Performance:** 383 TOPs

**Peak bfloat16:** 383 TFLOPs

**Lithography:** TSMC 6nm FinFET

**Compute Units:** 220

**Peak Engine Clock:** 1700 MHz

**Peak Double Precision Matrix (FP64) Performance:** 95.7 TFLOPs

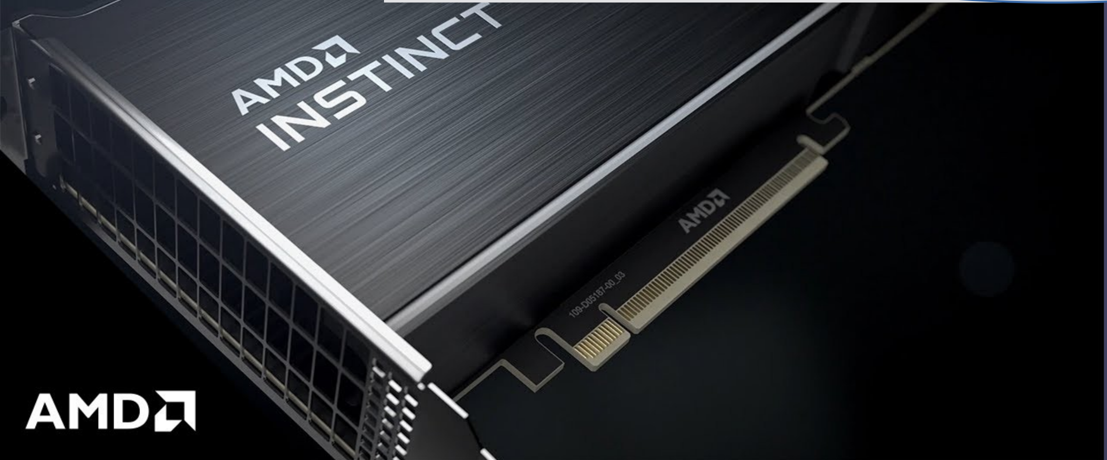
**Peak Double Precision (FP64) Performance:** 47.9 TFLOPs

**Peak INT8 Performance:** 383 TOPs

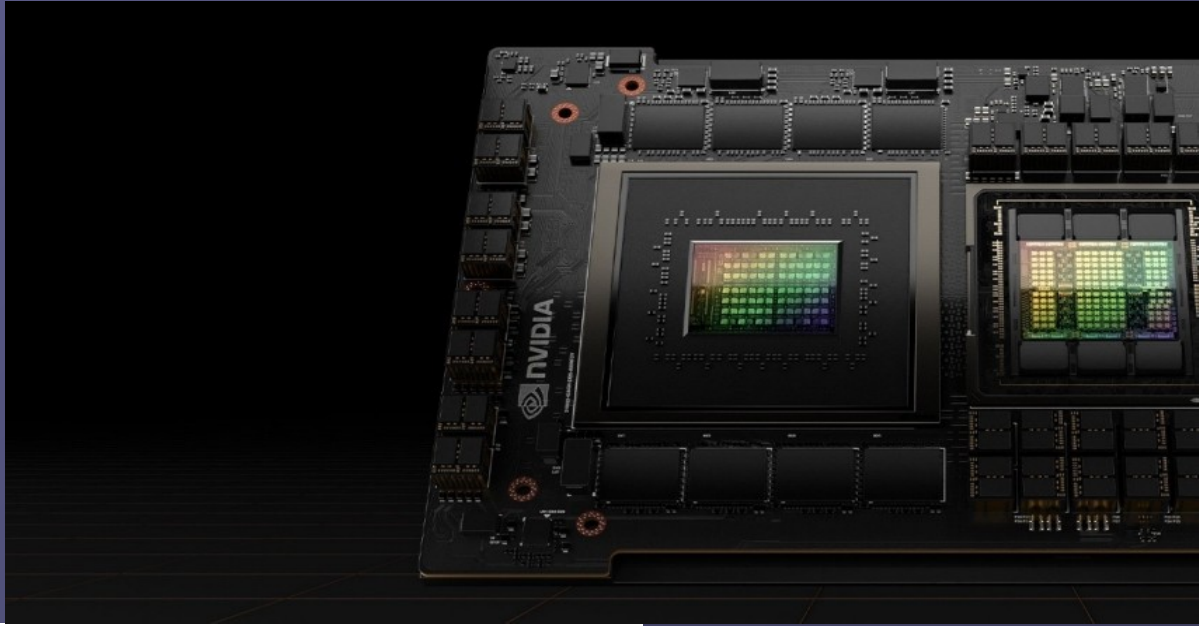
**OS Support:** Linux x86_64

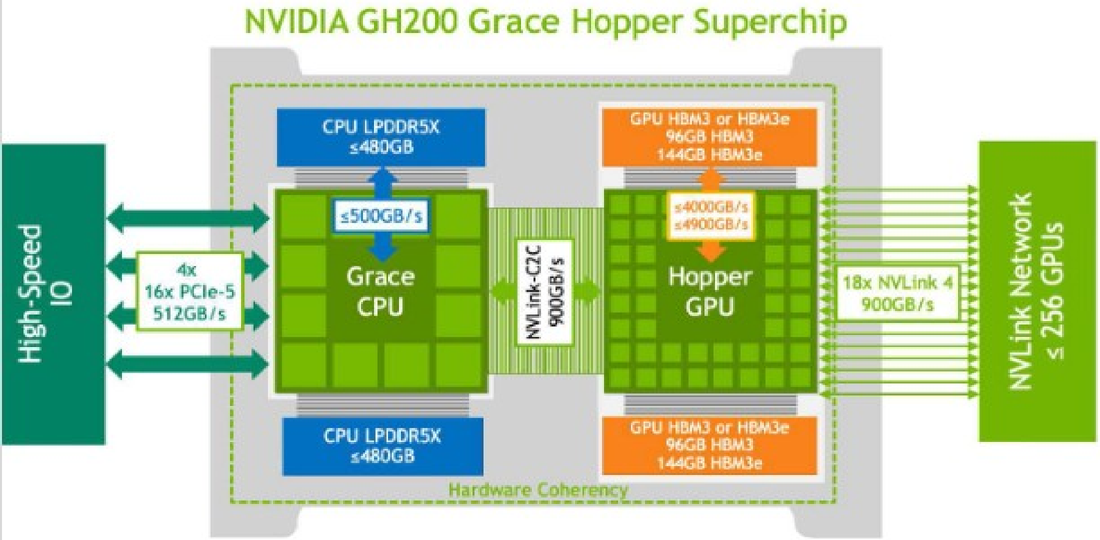
## Requirements

**Total Board Power (TBP):** 500W | 560W Peak



New “Grace Hopper GH200 superchip” ; GPU + CPU on one platform; used in new Jupiter supercomputer at JSC Jülich.





https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/

NVIDIA GH200 Grace Hopper S

Hopper GPU
16896 CUDA cores
528 tensor cores
34 Tflop/s double prec.
67 Tflop/s single prec.
67 Tflop/s tensor core double prec.

72 Armv9 CPU cores
480 GB memory

From https://www.top500.org/

Nov. 2023 List

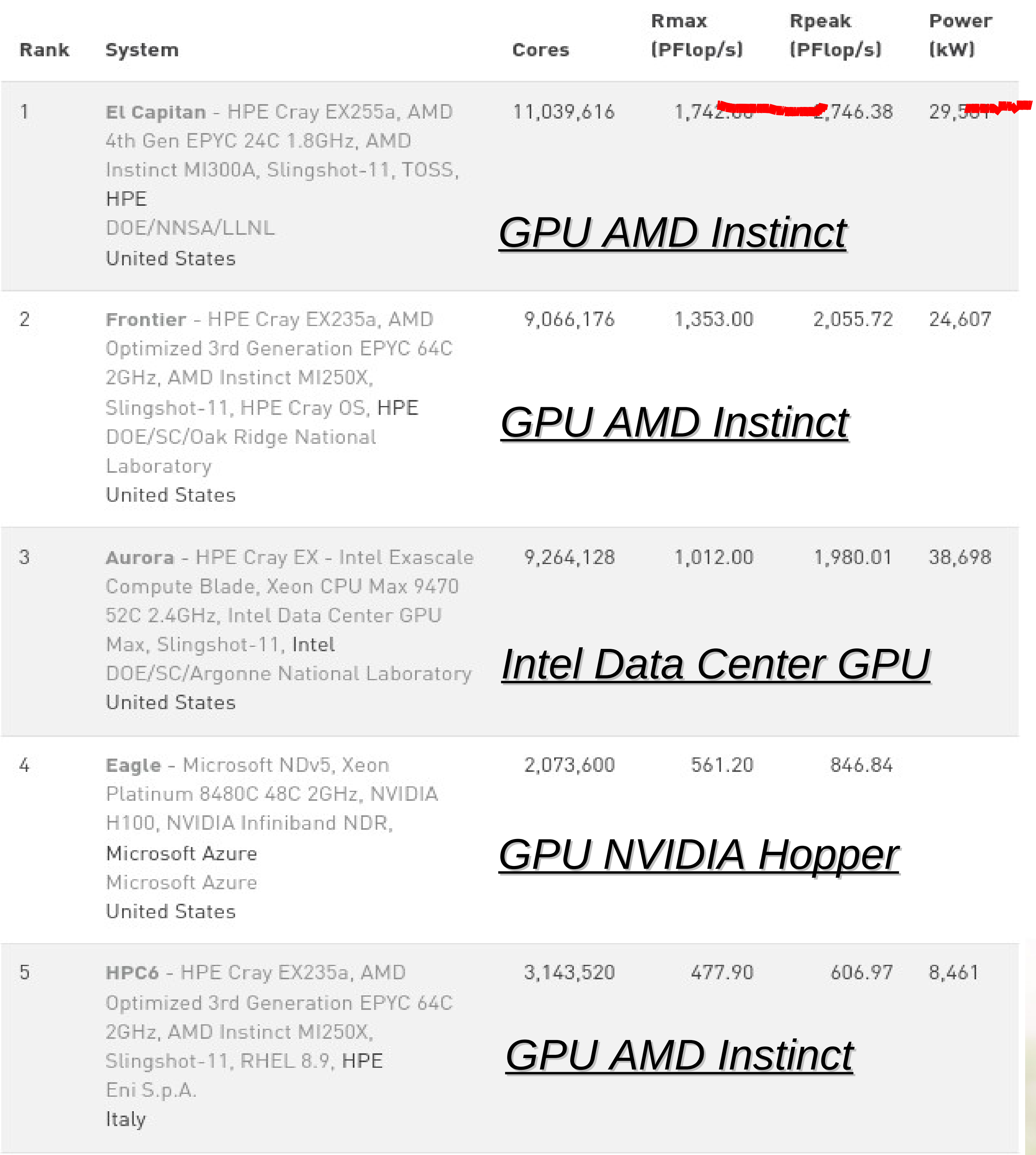
USA

USA

USA

USA

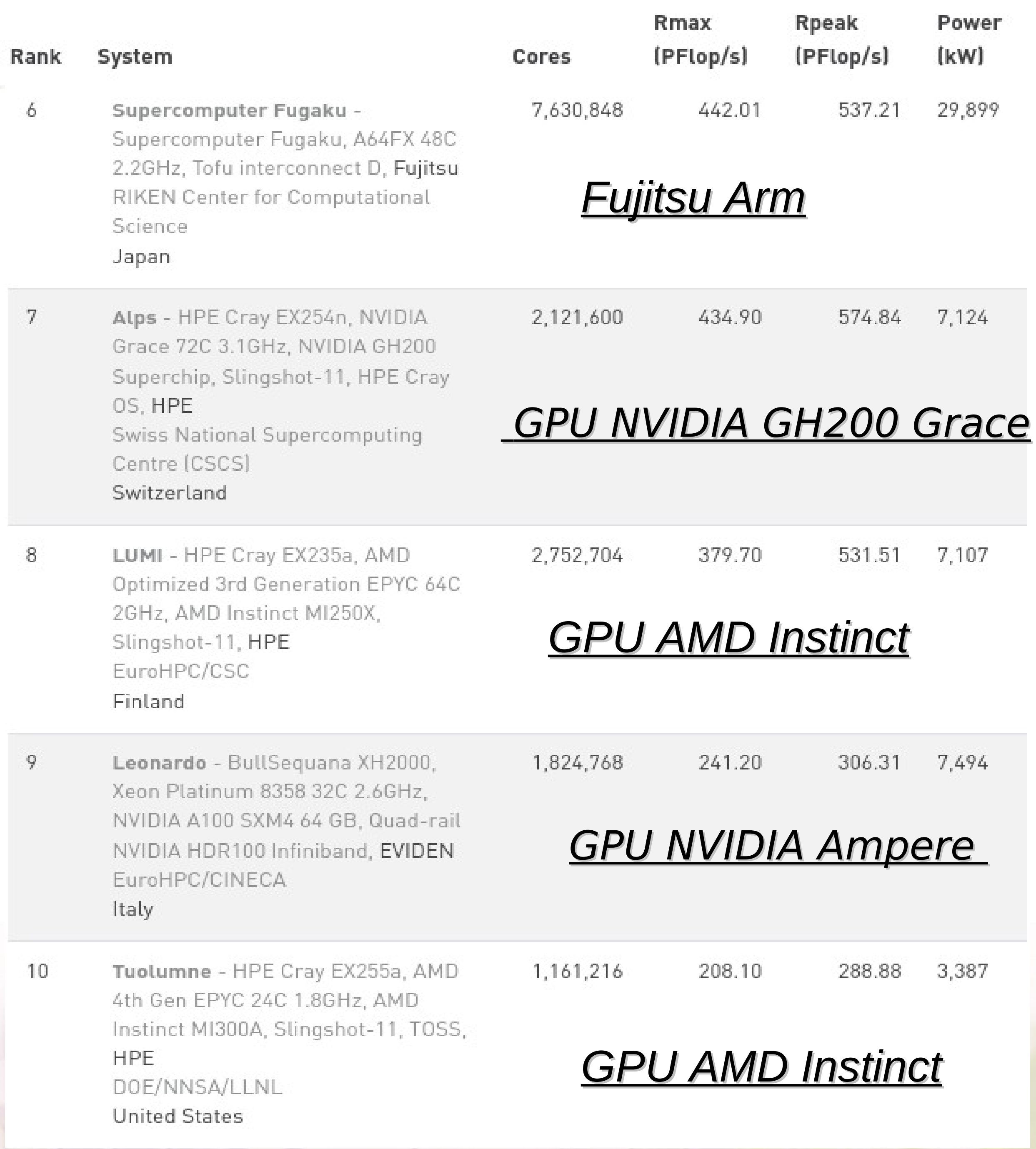
Italy



| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States<br>***GPU AMD Instinct*** | 11,039,616 | 1,742[redacted] | [redacted],746.38 | 29,5[redacted] |
| 2 | **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States<br>***GPU AMD Instinct*** | 9,066,176 | 1,353.00 | 2,055.72 | 24,607 |
| 3 | **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States<br>***Intel Data Center GPU*** | 9,264,128 | 1,012.00 | 1,980.01 | 38,698 |
| 4 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure<br>Microsoft Azure<br>United States<br>***GPU NVIDIA Hopper*** | 2,073,600 | 561.20 | 846.84 | |
| 5 | **HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE<br>Eni S.p.A.<br>Italy<br>***GPU AMD Instinct*** | 3,143,520 | 477.90 | 606.97 | 8,461 |

From https://www.top500.org/

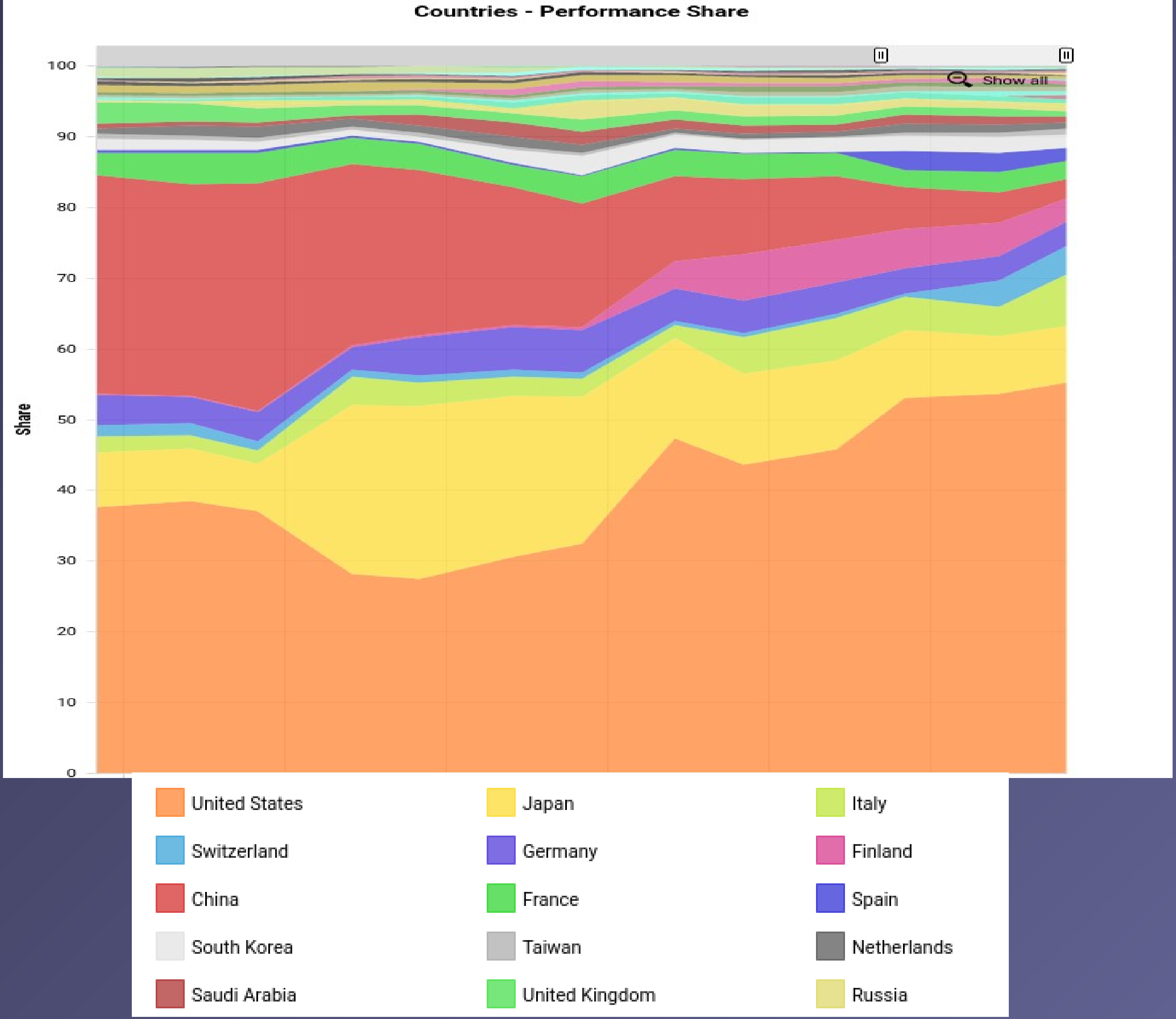
Nov. 2024 List



| | Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) | |
|---|---|---|---|---|---|---|---|
| **Japan** | 6 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442.01 | 537.21 | 29,899 | ***Fujitsu Arm*** |
| **USA** | 7 | **Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE<br>Swiss National Supercomputing Centre (CSCS)<br>Switzerland | 2,121,600 | 434.90 | 574.84 | 7,124 | ***GPU NVIDIA GH200 Grace*** |
| **Finland (EuroHPC)** | 8 | **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>EuroHPC/CSC<br>Finland | 2,752,704 | 379.70 | 531.51 | 7,107 | ***GPU AMD Instinct*** |
| **USA** | 9 | **Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN<br>EuroHPC/CINECA<br>Italy | 1,824,768 | 241.20 | 306.31 | 7,494 | ***GPU NVIDIA Ampere*** |
| **USA** | 10 | **Tuolumne** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States | 1,161,216 | 208.10 | 288.88 | 3,387 | ***GPU AMD Instinct*** |

# Top 500 List November 2023 – Performance Share of Countries

From https://www.top500.org



LUMI

# Supercomputer, Kajaani, Finland

Using only Hydroelectric Power and its Heat used for heating buildings.

No. 5 in top500
No. 7 in green500

2.2 million cores
~12.000 AMD GPUs

EuroHPC and LUMI consortium:

Finland, Belgium, Czech Republic, Denmark, Estonia, Iceland, Norway, Poland, Sweden, and Switzerland.

RIKEN, Kobe, JAPAN

# FUGAKU

Nature's Secrets

富岳

Mt. Fuji

## The world's fastest Super Computer 2020 /2021

7.6 million cores, 442 Pflop/s

source :nytimes

Fugaku extends its reign as champion of supercomputers

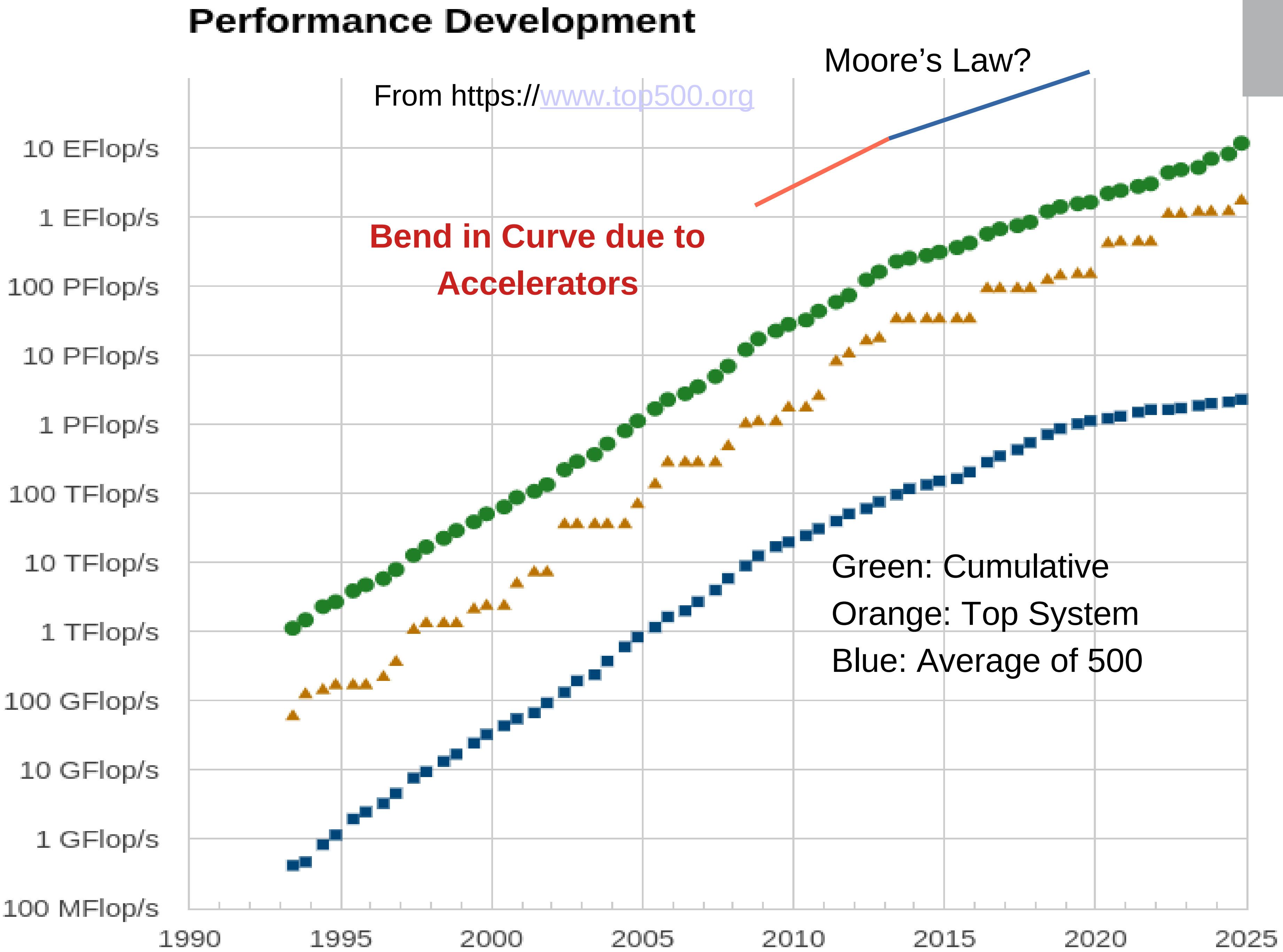
GCS
Gauss Centre for Supercomputing

JUWELS Booster 936 nodes (AMD CPU, 4x Ampere GPU)
~450.000 AMD cores, 25 million NVIDIA Ampere GPU cores
~ 70 Pflop/s SP ~ 44 Pflop/s DP
No. 18 in top500 list, No. 3 in green500 list

Jülich Wizard for European Leadership Science

**Watch out for new Exascale System at Jülich (JSC): JEDI / JUPITER !**

*Copyright:*
*— Forschungszentrum Jülich*



Moore’s Law?

From https://www.top500.org

**Bend in Curve due to Accelerators**

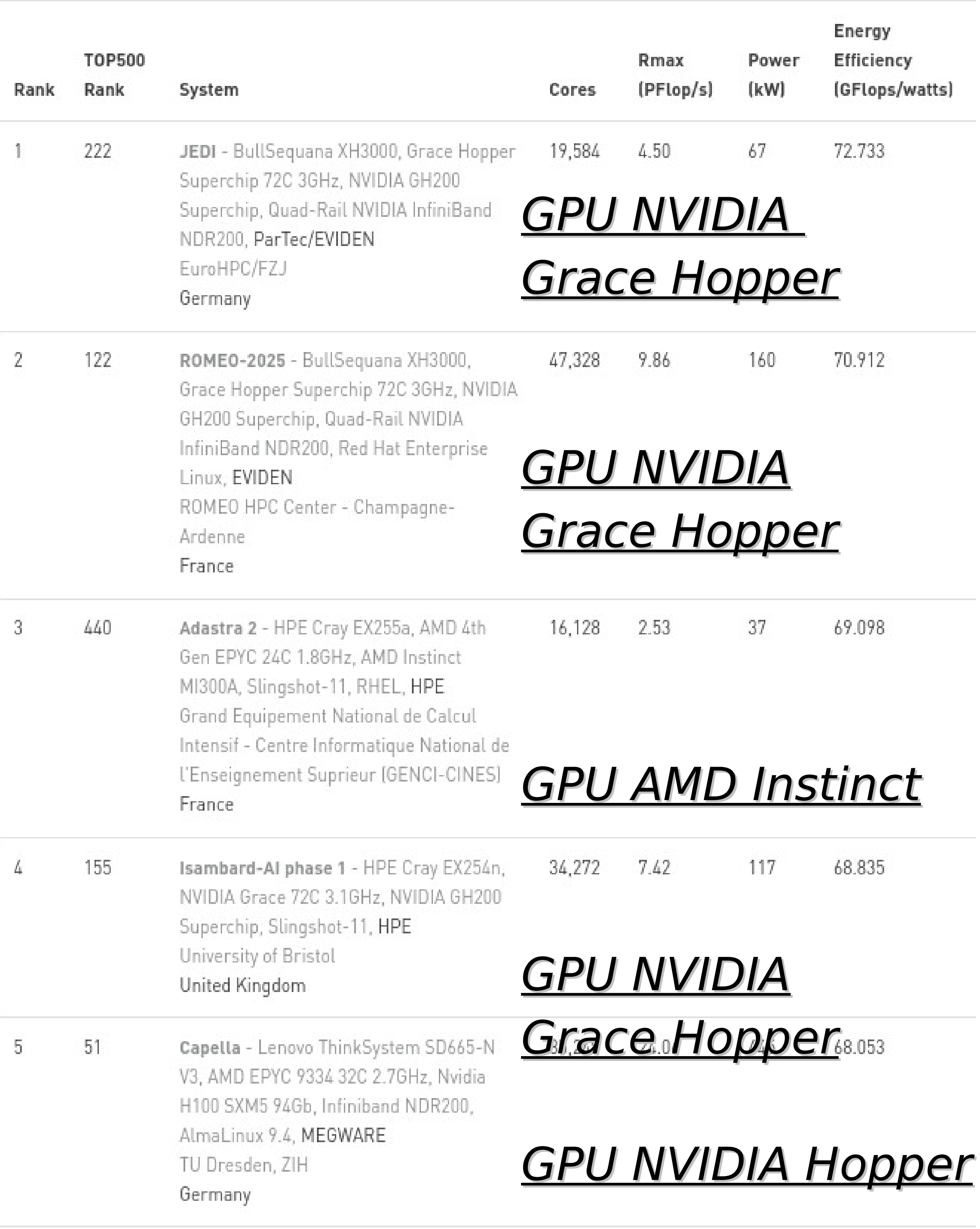
Green: Cumulative

Orange: Top System

Blue: Average of 500

## GREEN 500 list Nov. 2024

Power Efficiency
(Gflops/Watts),
see also top500 webpage
right: 1-5
below: 6-10



| Rank | TOP500 Rank | System | Cores | Rmax (PFlop/s) | Power (kW) | Energy Efficiency (GFlops/watts) |
|---|---|---|---|---|---|---|
| 1 | 222 | **JEDI** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN<br>EuroHPC/FZJ<br>Germany | 19,584 | 4.50 | 67 | 72.733 |
| 2 | 122 | **ROMEO-2025** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, Red Hat Enterprise Linux, EVIDEN<br>ROMEO HPC Center - Champagne-Ardenne<br>France | 47,328 | 9.86 | 160 | 70.912 |
| 3 | 440 | **Adastra 2** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, RHEL, HPE<br>Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES)<br>France | 16,128 | 2.53 | 37 | 69.098 |
| 4 | 155 | **Isambard-AI phase 1** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE<br>University of Bristol<br>United Kingdom | 34,272 | 7.42 | 117 | 68.835 |
| 5 | 51 | **Capella** - Lenovo ThinkSystem SD665-N V3, AMD EPYC 9334 32C 2.7GHz, Nvidia H100 SXM5 94Gb, Infiniband NDR200, AlmaLinux 9.4, MEGWARE<br>TU Dresden, ZIH<br>Germany | [illegible] | [illegible] | [illegible] | 68.053 |

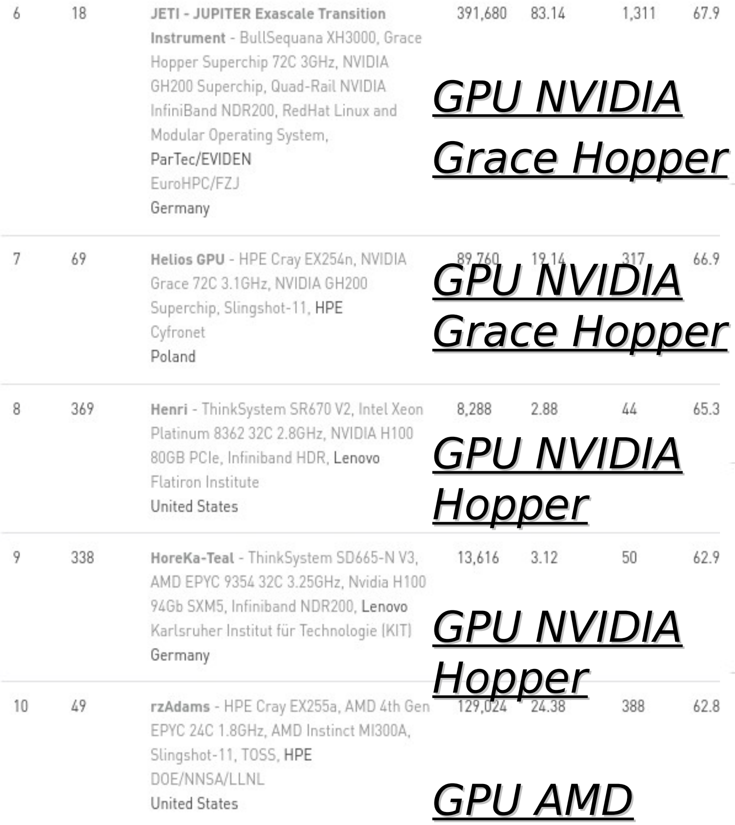
***GPU NVIDIA Grace Hopper***

***GPU NVIDIA Grace Hopper***

***GPU AMD Instinct***

***GPU NVIDIA Grace Hopper***

***GPU NVIDIA Hopper***



| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 18 | **JETI - JUPITER Exascale Transition Instrument** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Linux and Modular Operating System, ParTec/EVIDEN<br>EuroHPC/FZJ<br>Germany | 391,680 | 83.14 | 1,311 | 67.9 |
| 7 | 69 | **Helios GPU** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE<br>Cyfronet<br>Poland | 89,760 | 19.14 | 317 | 66.9 |
| 8 | 369 | **Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo<br>Flatiron Institute<br>United States | 8,288 | 2.88 | 44 | 65.3 |
| 9 | 338 | **HoreKa-Teal** - ThinkSystem SD665-N V3, AMD EPYC 9354 32C 3.25GHz, Nvidia H100 94Gb SXM5, Infiniband NDR200, Lenovo<br>Karlsruher Institut für Technologie (KIT)<br>Germany | 13,616 | 3.12 | 50 | 62.9 |
| 10 | 49 | **rzAdams** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States | 129,024 | 24.38 | 388 | 62.8 |

***GPU NVIDIA Grace Hopper***

***GPU NVIDIA Grace Hopper***

***GPU NVIDIA Hopper***

***GPU NVIDIA Hopper***

***GPU AMD Instinct***

GPU Computing

# More on GPU

# Graphics Processors (GPU) as General Purpose Supercomputers (GPGPU)

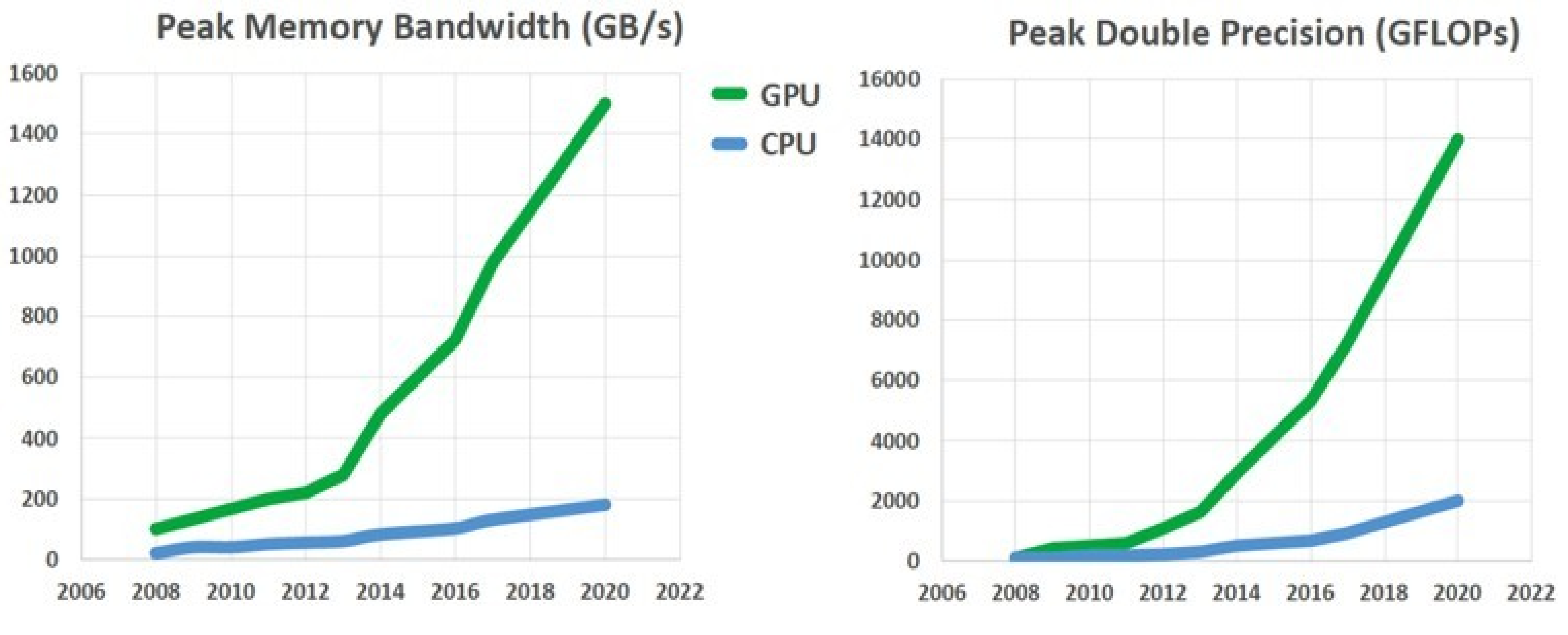
Turing 2019:
GeForce RTX 2080ti

Pascal 2016:
Quadro P6000

2010: Tesla C1070
Laohu 北京
老虎

2008…
GeForce 9800 GTX, 128 Stream Proc., 512 MB
GeForce 9800 GX2, 256 Stream Proc., 1 GB
GeForce 9800 GT, 64 Stream Proc., 512 MB
[…]
2009: Tesla ~200 Proc., 4GB
2010: Fermi ~400 Proc., 4GB
2013: Kepler K20, ~2500 Procs., 6GB
2016: Kepler K80, ~5000 Procs.
2016-18: Pascal, Volta, Turing > 5000 Procs., 40 GB
2019-25: Ampere, Hopper, … > 10000 Procs. 240 GB

# Peak Floating Point Operations per Second And Peak Memory Bandwidth for CPU and GPU

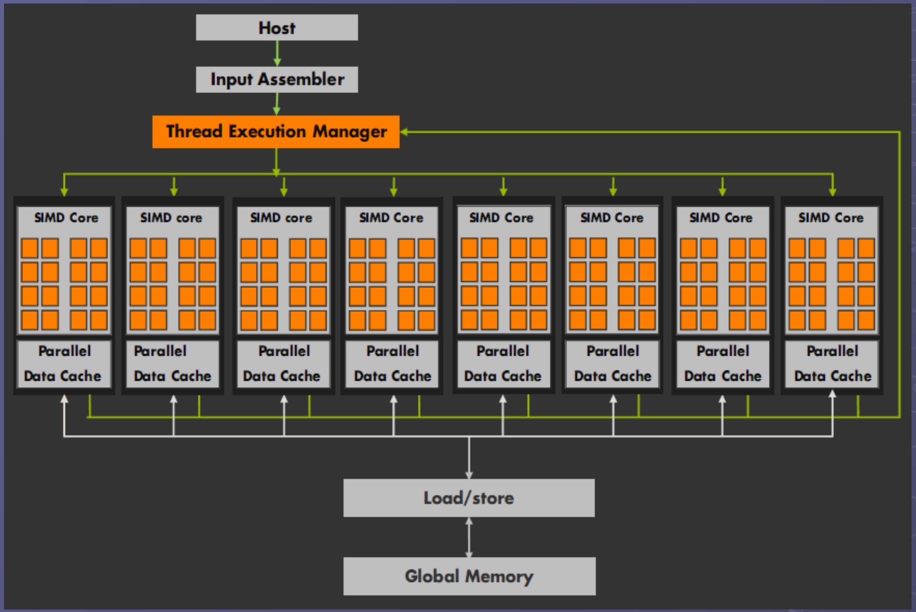


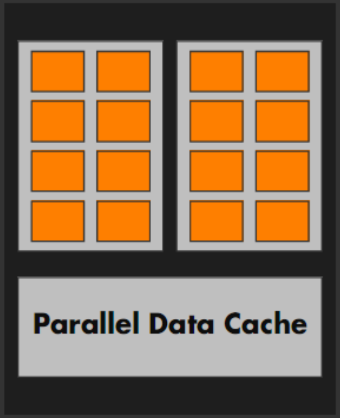
Chip to chip comparison of peak memory bandwidth in GB/s and peak double precision gigaflops for GPUs and CPUs since 2008. Data for Nvidia "Volta" V100 and Intel "Cascade Lake" Xeon SP are used for 2019 and projected into 2020. From:

https://www.nextplatform.com/2019/07/10/a-decade-of-accelerated-computing-augurs-well-for-gpus/

# Hardware around 2006

## architecture still valid – just scaled up (except: tensor cores and fast data links)





Each core

- 8 functional units
- SIMD 16/32 “warp”
- 8-10 stage pipeline
- Thread scheduler
- 128-512 threads/core
- 16 KB shared memory

Total #threads/chip

16 * 512 = 8K

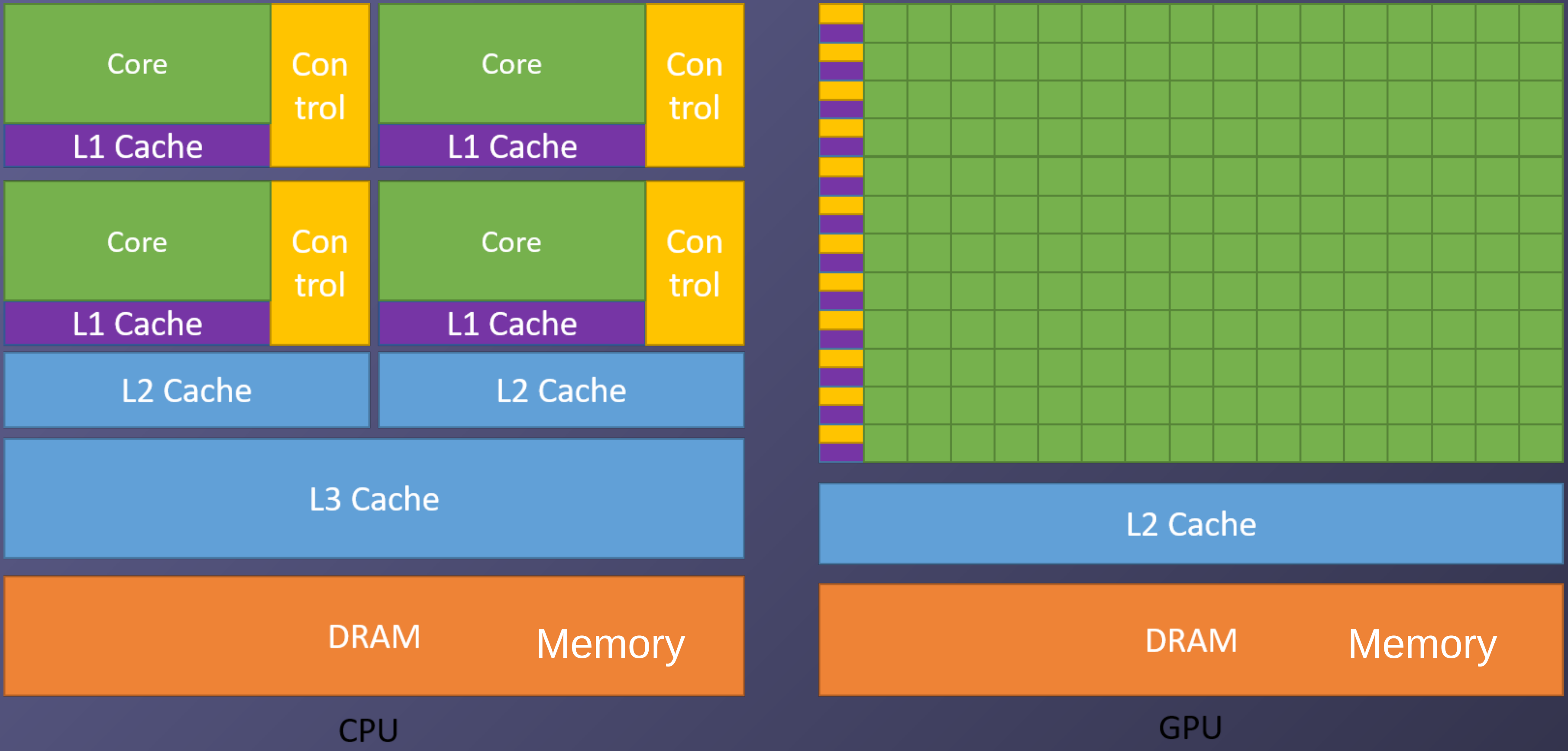
**These are the physical parameters! The software (“runtime system”) sees a “virtual GPU” which is MUCH larger!!**

**GeForce 8800 GTX:**

**575 MHz * 128 processors * 2 flop/inst * 2 inst/clock = 333 Gflops**

# CPU and GPU; from CUDA NVIDIA Developer Zone at

http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

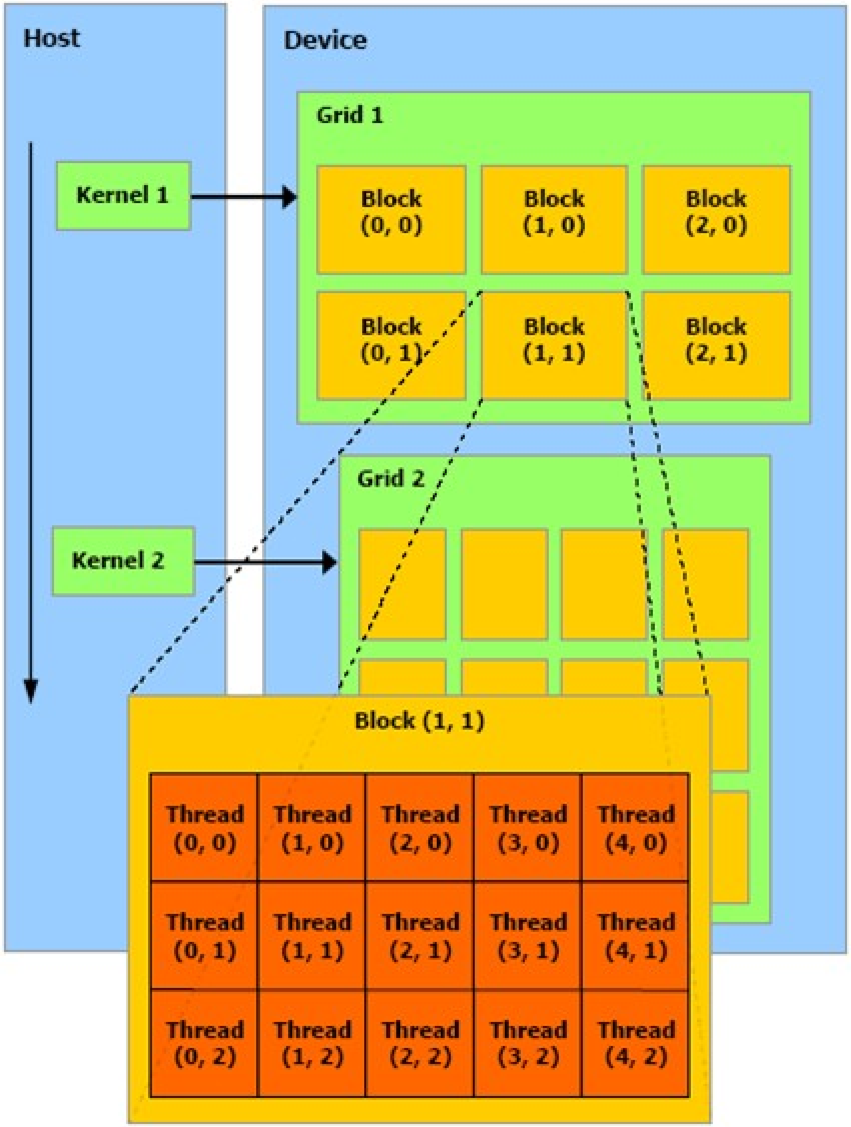


**“The GPU devotes more transistors to computing”**

**“favours data parallel operations”**

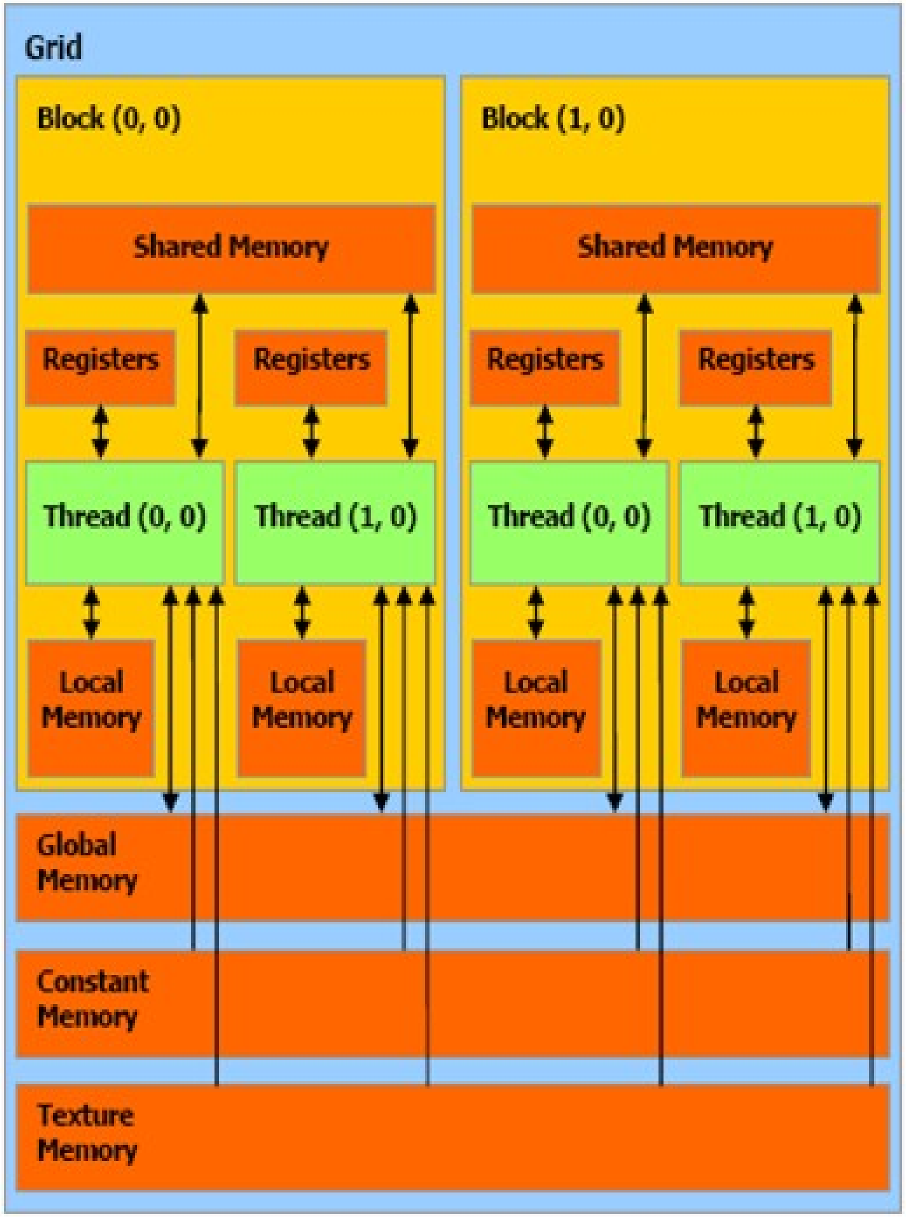
# GPU Structure

https://docs.nvidia.com/cuda/parallel-thread-execution/index.html



| | | | | |
|---|---|---|---|---|
| Thread (0, 0) | Thread (1, 0) | Thread (2, 0) | Thread (3, 0) | Thread (4, 0) |
| Thread (0, 1) | Thread (1, 1) | Thread (2, 1) | Thread (3, 1) | Thread (4, 1) |
| Thread (0, 2) | Thread (1, 2) | Thread (2, 2) | Thread (3, 2) | Thread (4, 2) |

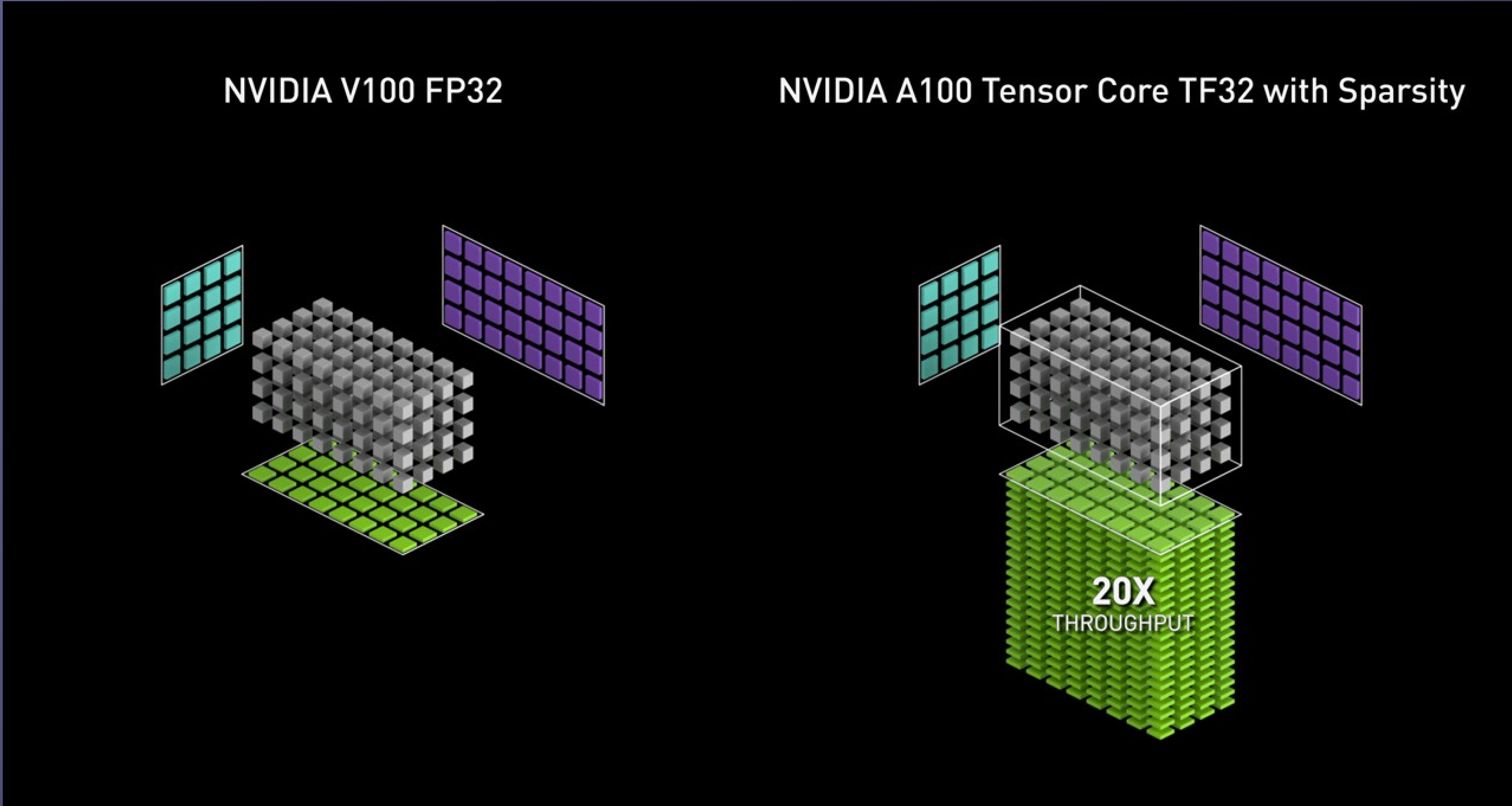
The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks



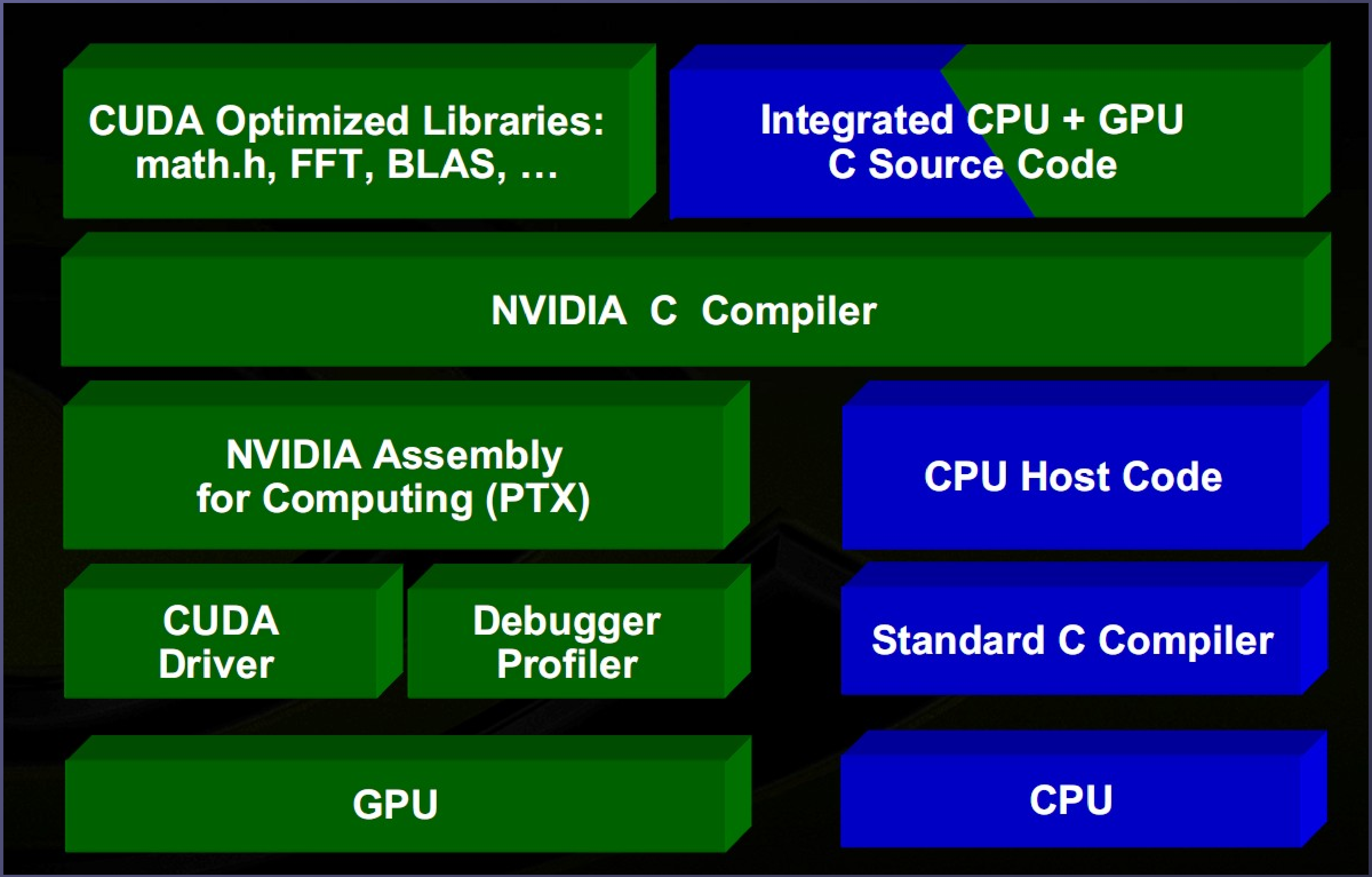
# New feature in Volta, Ampere, Turing: Tensor Cores

https://www.nvidia.com/en-us/data-center/tensor-cores/

**FP64 Tensor Cores:** “A100 brings the power of Tensor Cores to HPC, providing the biggest milestone since the introduction of double-precision GPU computing for HPC. By enabling matrix operations in FP64 precision, a whole range of HPC applications that need double-precision math can now get a 2.5X boost in performance and efficiency compared to prior generations of GPUs.” (Quote from NVIDIA webpages)



# CUDA



CUDA Optimized Libraries: math.h, FFT, BLAS, …

Integrated CPU + GPU C Source Code

NVIDIA C Compiler

NVIDIA Assembly for Computing (PTX)

CPU Host Code

CUDA Driver

Debugger Profiler

Standard C Compiler

GPU

CPU

# GPU Computing Applications

## Libraries and Middleware

| | | | | | | |
|---|---|---|---|---|---|---|
| cuDNN<br>TensorRT | cuFFT<br>cuBLAS<br>cuRAND<br>cuSPARSE | CULA<br>MAGMA | Thrust<br>NPP | VSIPL<br>SVM<br>OpenCurrent | PhysX<br>OptiX<br>iRay | MATLAB<br>Mathematica |

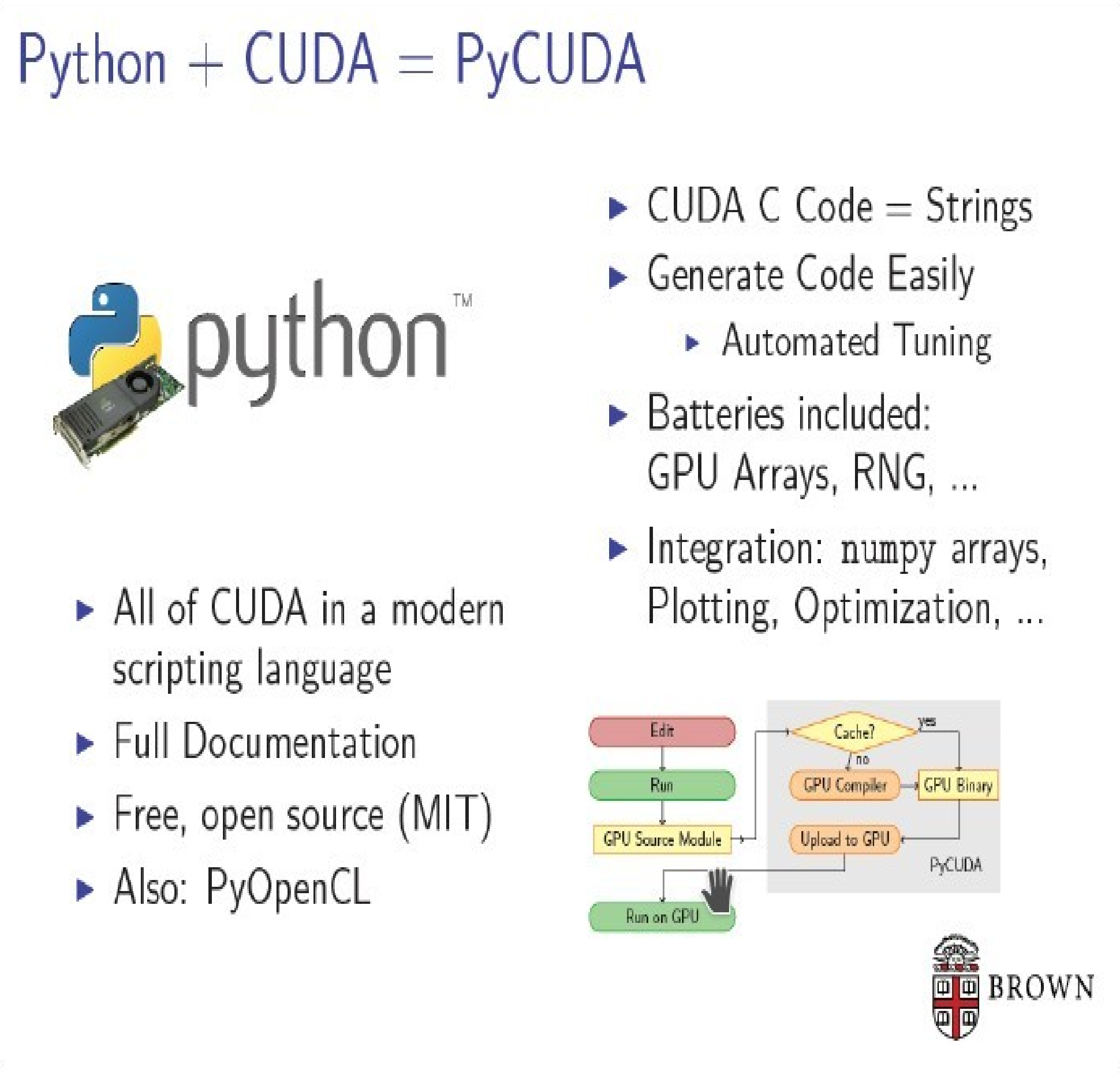
## Programming Languages

| | | | | | |
|---|---|---|---|---|---|
| C | C++ | Fortran | Java<br>Python<br>Wrappers | DirectCompute | Directives<br>(e.g. OpenACC) |

… Hopper… Blackwell … X

## CUDA-Enabled NVIDIA GPUs

| | | | | |
|---|---|---|---|---|
| NVIDIA Ampere Architecture<br>(compute capabilities 8.x) | | | | Tesla A Series |
| NVIDIA Turing Architecture<br>(compute capabilities 7.x) | | GeForce 2000 Series | Quadro RTX Series | Tesla T Series |
| NVIDIA Volta Architecture<br>(compute capabilities 7.x) | DRIVE/JETSON<br>AGX Xavier | | Quadro GV Series | Tesla V Series |
| NVIDIA Pascal Architecture<br>(compute capabilities 6.x) | Tegra X2 | GeForce 1000 Series | Quadro P Series | Tesla P Series |
| | Embedded | Consumer Desktop/Laptop | Professional Workstation | Data Center |



# Python + CUDA = PyCUDA

- All of CUDA in a modern scripting language
- Full Documentation
- Free, open source (MIT)
- Also: PyOpenCL

- CUDA C Code = Strings
- Generate Code Easily
  - Automated Tuning
- Batteries included: GPU Arrays, RNG, ...
- Integration: `numpy` arrays, Plotting, Optimization, ...

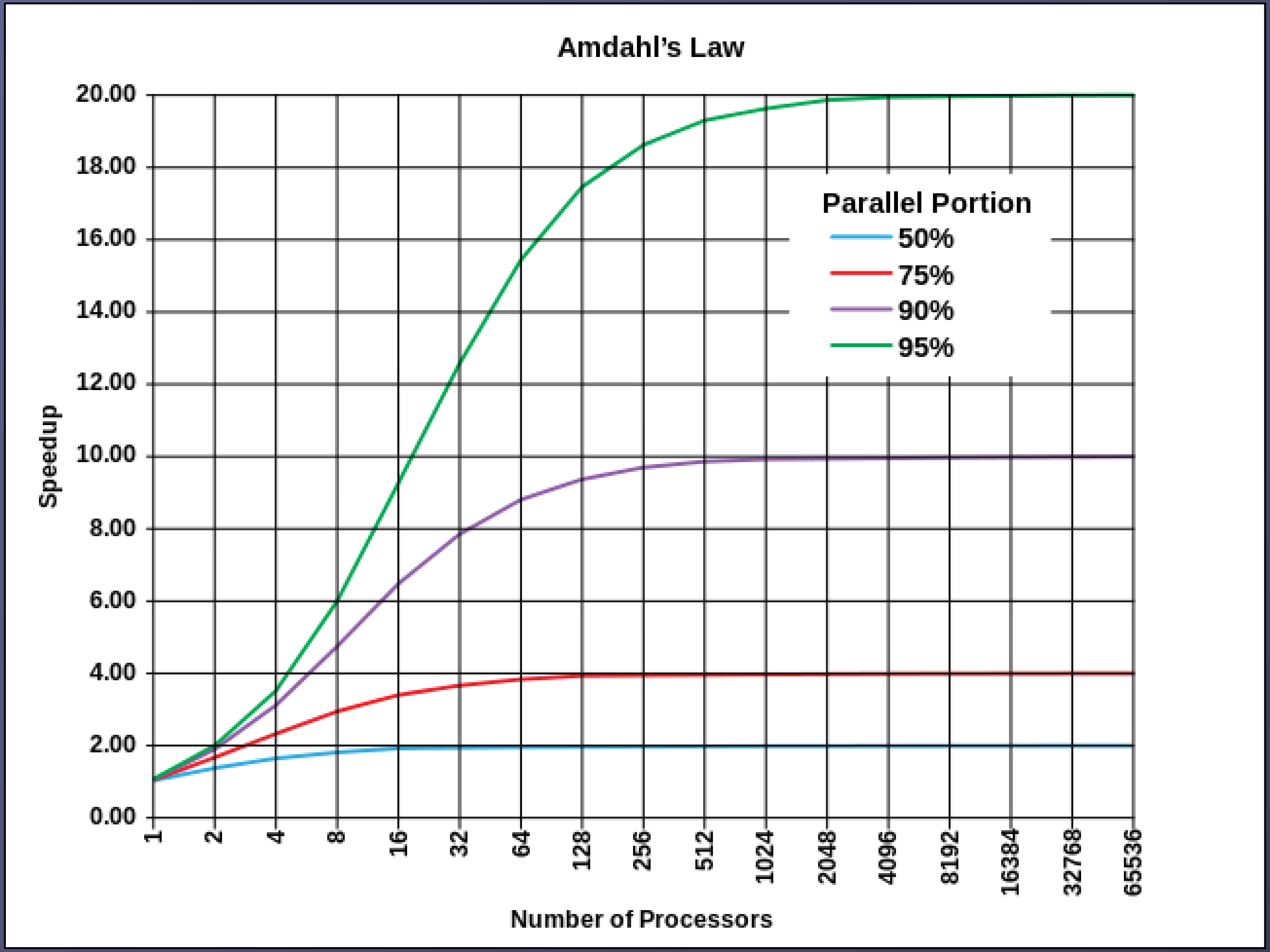
https://developer.nvidia.com/cuda-python

**http://mathema.tician.de/software/pycuda** https://documen.tician.de/pycuda/

# Parallel Computing

## Some basic ideas

# Amdahl's Law  (Gene Amdahl 1967)



Evolution according to Amdahl's law of the theoretical speedup of the execution of a program as function of the number of processors p executing it, for different values of p. The speedup is limited by the serial part of the program. For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20 times.

By Daniels220 at English Wikipedia - Own work based on: File:AmdahlsLaw.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6678551

# Calculate Amdahl's Law:

Let X be the part of my program (in terms of computing time) which can be parallelised. The sequential computing time $T_{seq}$ is normalized to unity (1), and can be expressed as:

$$T_{seq} = 1 = X + (1\text{-}X)$$

The parallel computing time Tpar under ideal conditions (ideal load balancing, ultrafast communication):

$$T_{par} = X/p + (1\text{-}X)$$

with processor number (core number)   p ;

Then the speed-up of the program $S = T_{seq} / T_{par}$ :

$$S = 1 / (1\text{-}X+X/p) \qquad ;$$

**Note: $T_{par}/T_{seq} = 1/S$  (sometimes also plotted)**

Note the limit of S for p>>1 and X~1  is very large:  S = 1/(1-X).

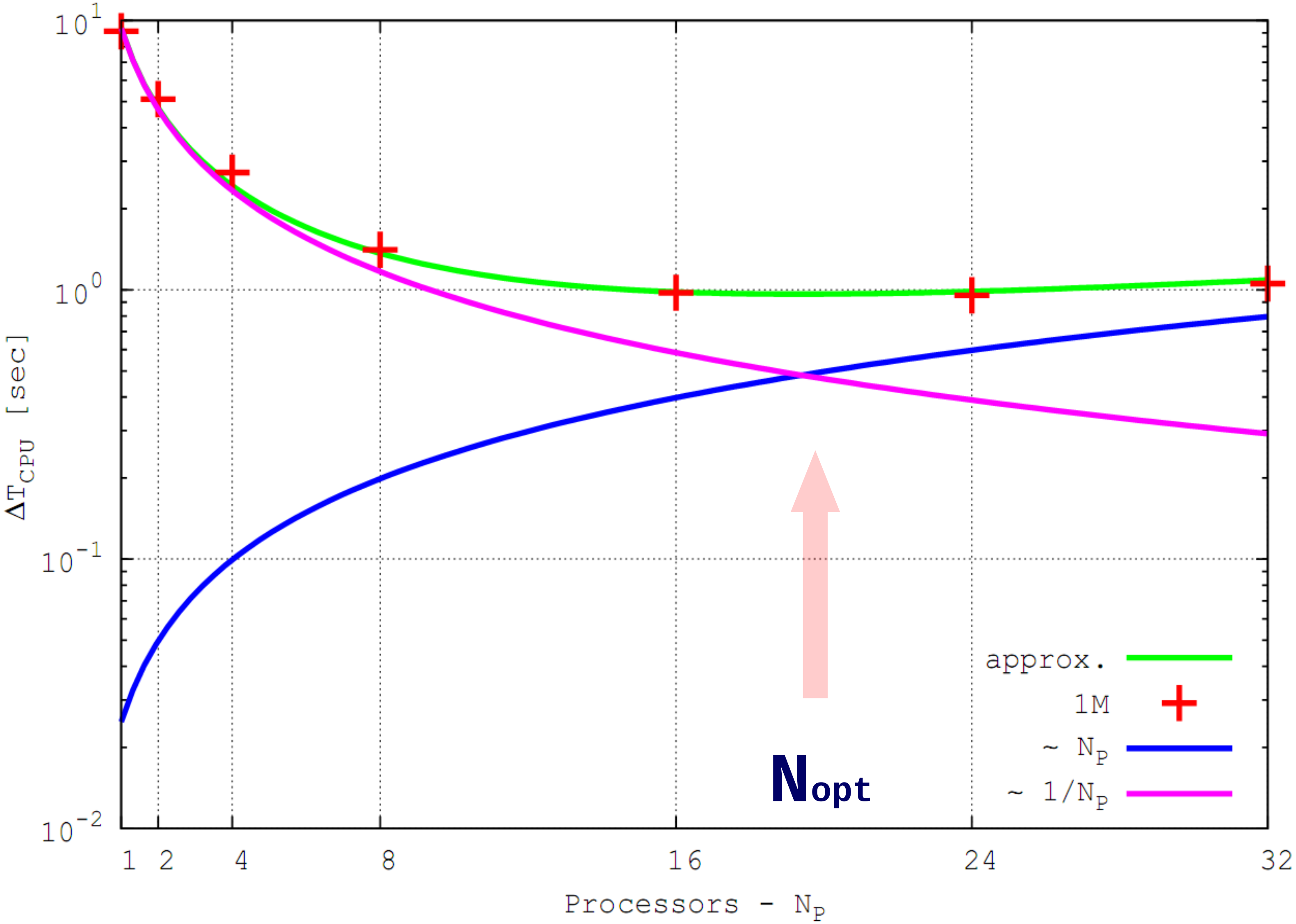
With communication overhead:

$$T_{par} = X/p + (1\text{-}X) + T_{comm} \qquad \rightarrow \qquad S = 1 / (1\text{-}X+X/p+T_{comm})$$

If $T_{comm}$ independent of p we have for large p:  $S = 1 / (1\text{-}X + T_{comm}) = \text{const.}$

**If** $T_{comm} = c\, p^k$ (k>0) we get:  $S = 1 / (1\text{-}X + c\, p^k)$  → 0 for large p!!!

# Parallel code on cluster



# Strong and Soft Scaling

➔ Strong Scaling: Fixed Problem size, increase p

➔ Soft Scaling: Increase Problem size, increase p (constant amount of work per processing element)

Ansatz for Soft Scaling (T$_{\text{comm}}$ neglected here):

➔ **$T_{seq} = p\ (X + (1\text{-}X))$**

➔ **$T_{par} = X + p\ (1\text{-}X)$**

➔ **$S = T_{seq}/T_{par} = p\ /\ (X+p\ (1\text{-}X))$**

**If $X \sim 1$: $S = p$ ; $T_{par} = X =$ const.**

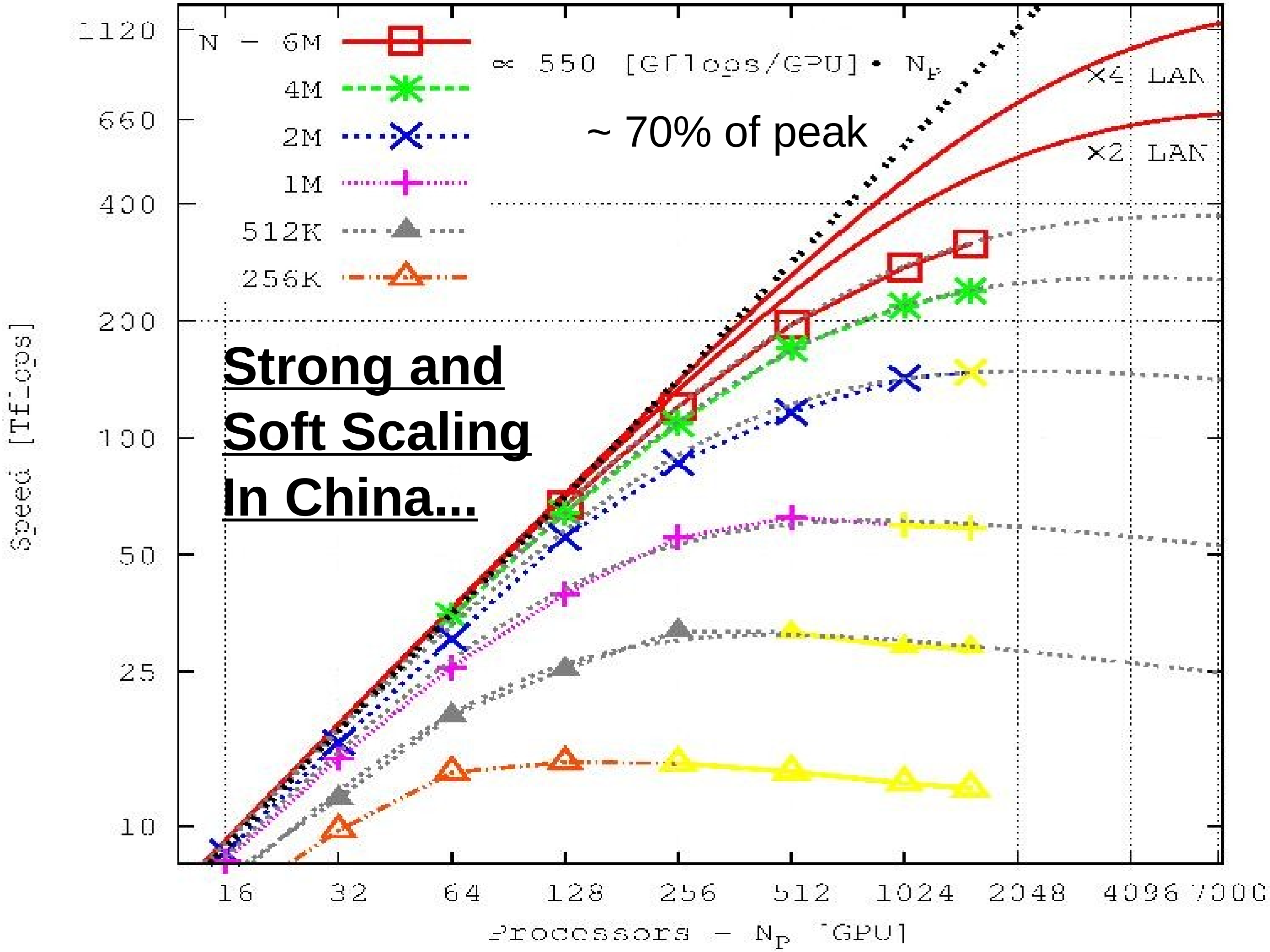
# ΦGPU – NBODY Code

中国科学院国家天文台
National Astronomical Observatories, CAS

**350 Teraflop/s**
**1600 GPUs .**
**440 cores**
**= 704.000**
**GPU-Cores**

**Using**
**Mole-8.5**
**of**
**IPE/CAS**
**Beijing**

**Berczik et al.**
**2013**



~ 70% of peak

**Strong and**
**Soft Scaling**
**In China...**

https://ui.adsabs.harvard.edu/abs/2013hpc..conf...52B/abstract

中国科学院过程工程研究所
Institute Of Process Engineering, Chinese Academy Of Sciences

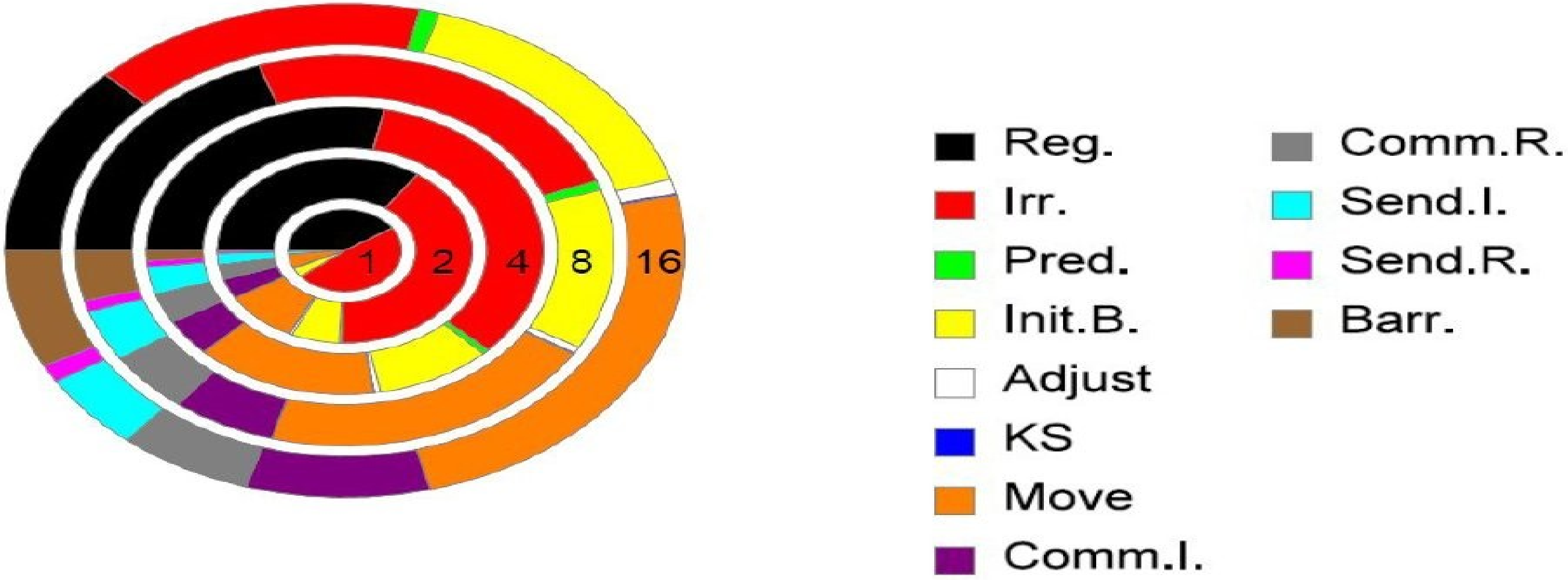
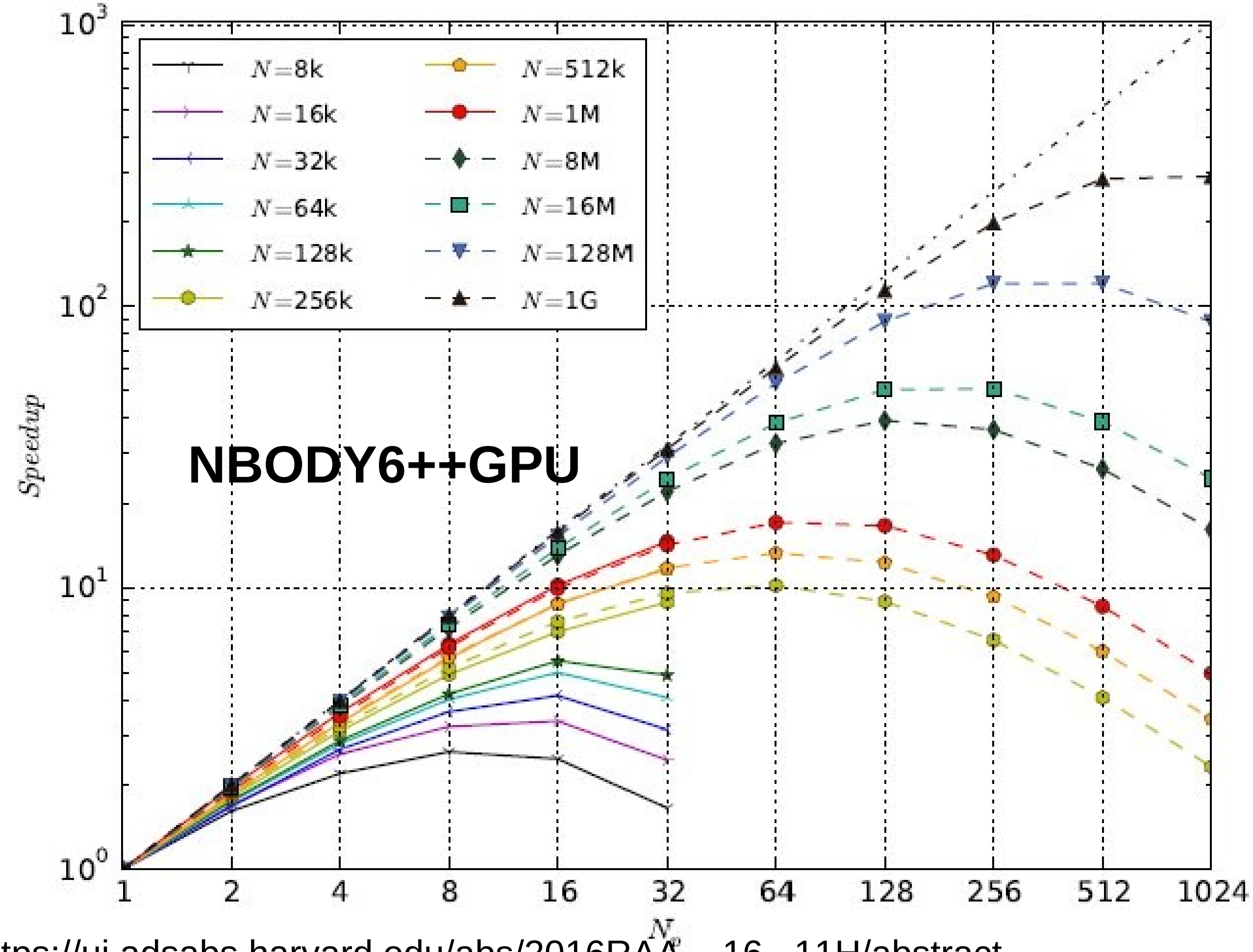


**Table 1** Main components of NBODY6++

| Description | Timing variable | Expected scaling | | Fitting value [sec] |
|---|---|---|---|---|
| | | $N$ | $N_p$ | |
| Regular force computation | $T_{\rm reg}$ | $\mathcal{O}(N_{\rm reg} \cdot N)$ | $\mathcal{O}(N_p^{-1})$ | $(2.2 \cdot 10^{-9} \cdot N^{2.11} + 10.43) \cdot N_p^{-1}$ |
| Irregular force computation | $T_{\rm irr}$ | $\mathcal{O}(N_{\rm irr} \cdot \langle N_{nb} \rangle)$ | $\mathcal{O}(N_p^{-1})$ | $(3.9 \cdot 10^{-7} \cdot N^{1.76} - 16.47) \cdot N_p^{-1}$ |
| Prediction | $T_{\rm pre}$ | $\mathcal{O}(N^{kn_p})$ | $\mathcal{O}(N_p^{-kp_p})$ | $(1.2 \cdot 10^{-6} \cdot N^{1.51} - 3.58) \cdot N_p^{-0.5}$ |
| Data moving | $T_{\rm mov}$ | $\mathcal{O}(N^{kn_{m1}})$ | $\mathcal{O}(1)$ | $2.5 \cdot 10^{-6} \cdot N^{1.29} - 0.28$ |
| MPI communication (regular) | $T_{\rm mcr}$ | $\mathcal{O}(N^{kn_{cr}})$ | $\mathcal{O}(kp_{cr} \cdot \frac{N_p - 1}{N_p})$ | $(3.3 \cdot 10^{-6} \cdot N^{1.18} + 0.12)(1.5 \cdot \frac{N_p - 1}{N_p})$ |
| MPI communication (irregular) | $T_{\rm mci}$ | $\mathcal{O}(N^{kn_{ci}})$ | $\mathcal{O}(kp_{ci} \cdot \frac{N_p - 1}{N_p})$ | $(3.6 \cdot 10^{-7} \cdot N^{1.40} + 0.56)(1.5 \cdot \frac{N_p - 1}{N_p})$ |
| Synchronization | $T_{\rm syn}$ | $\mathcal{O}(N^{kn_s})$ | $\mathcal{O}(N_p^{kp_s})$ | $(4.1 \cdot 10^{-8} \cdot N^{1.34} + 0.07) \cdot N_p$ |
| Sequential parts on host | $T_{\rm host}$ | $\mathcal{O}(N^{kn_h})$ | $\mathcal{O}(1)$ | $4.4 \cdot 10^{-7} \cdot N^{1.49} + 1.23$ |



NBODY6++GPU

https://ui.adsabs.harvard.edu/abs/2016RAA....16...11H/abstract

*Huang, Berczik, Spurzem, Res. Astron. Astroph. 2016, 16, 11.*

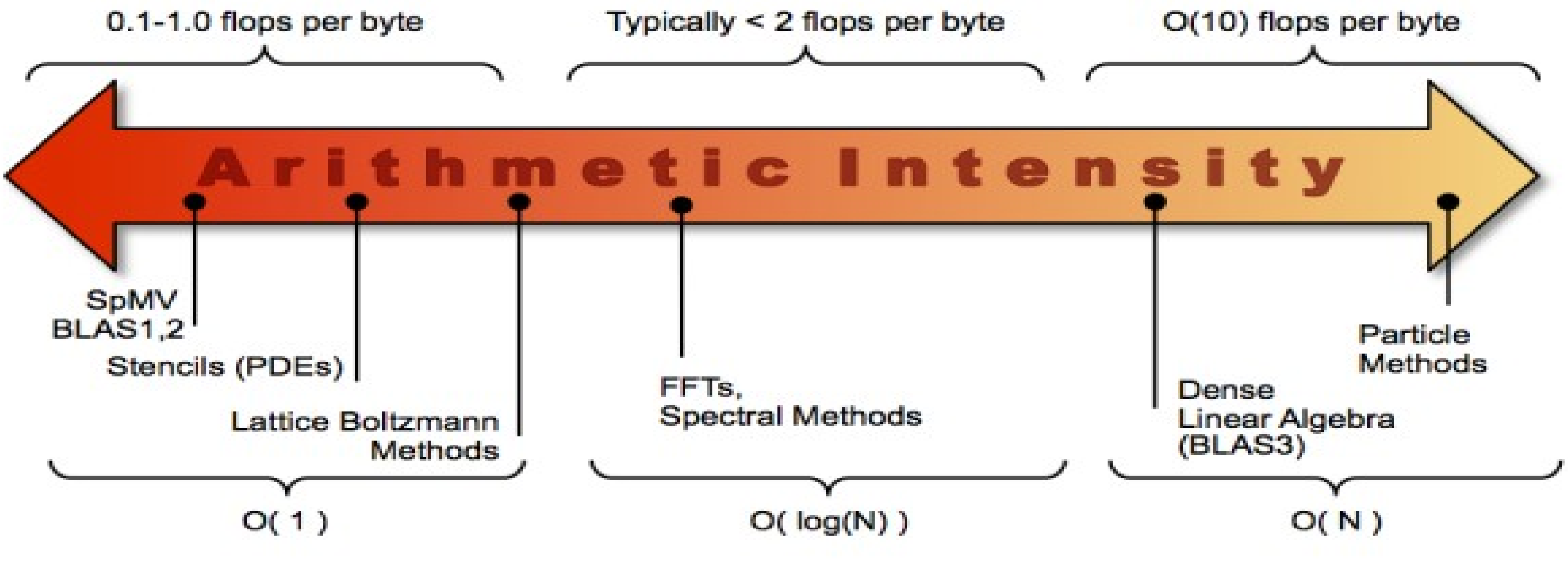
**Fig. 2** The speed-up ($S$) of NBODY6++ as a function of particle number ($N$) and processor number ($N_p$). Solid points are the measured speed-up ratio between sequential and parallel wall-clock time, dash lines predict the performance of larger scale simulations further. The symbols used in figure have the magnitudes: $1k = 1,024$, $1M = 1k^2$ and $1G = 1k^3$.

# Roofline Performance Model (LBL)

http://crd.lbl.gov/departments/computer-science/PAR/research/roofline

## Arithmetic Intensity

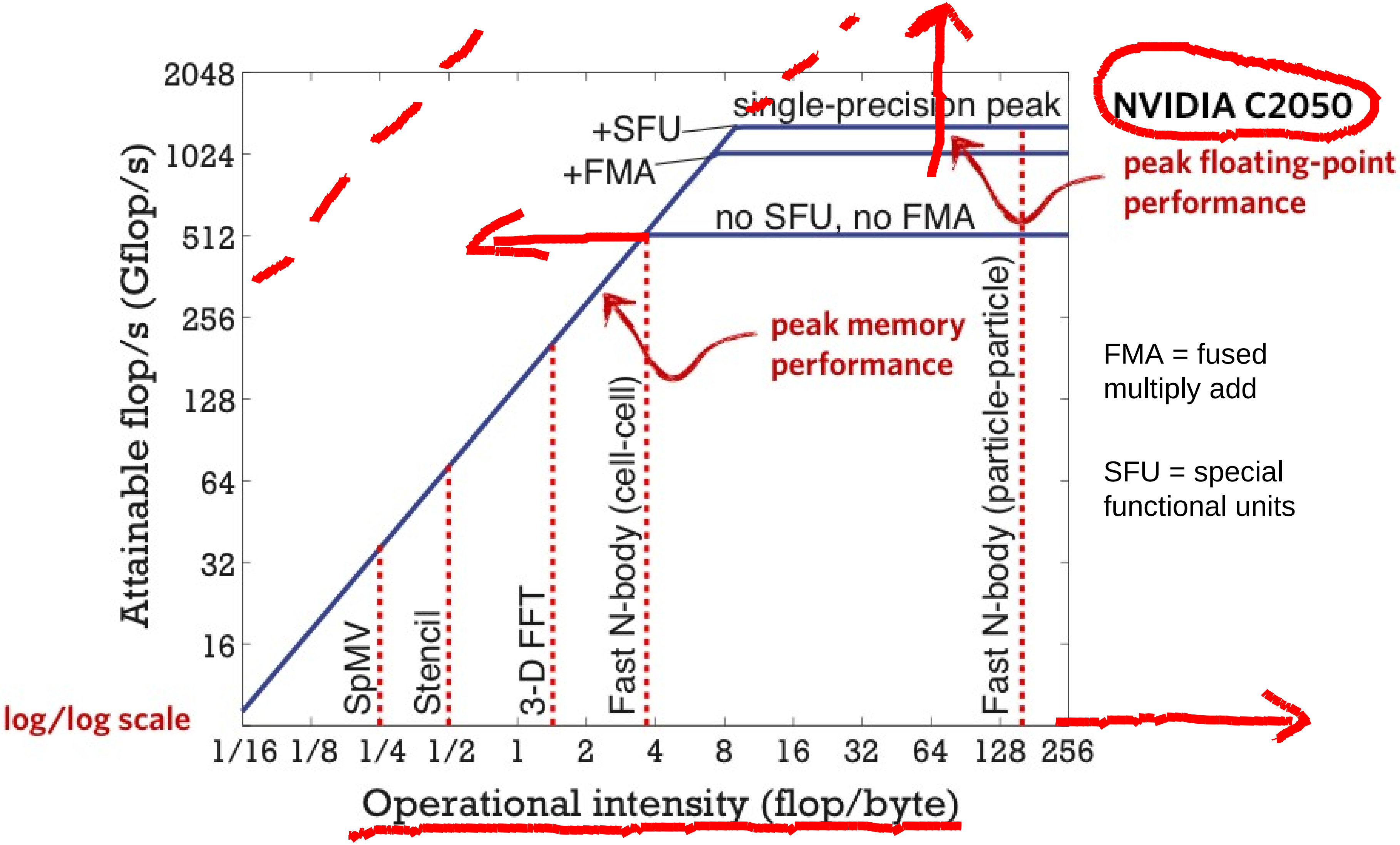
The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes).



# Roofline Performance Model (LBL): Lorena Barba Research Group at George Washington University:

https://engineering.gwu.edu/lorena-barba

http://lorenabarba.com/wp-content/uploads/2012/01/roofline_slide.png



FMA = fused multiply add

SFU = special functional units

## Friday, Feb. 21:

## Matrix Multiplication

## Histograms (from Jason Sanders' book; see our webpage link)

https://developer.nvidia.com/cuda-example

Download source code of examples in Jason Sanders' book

(Includes also HANDLE_ERROR routine)

## Timing and Debugging

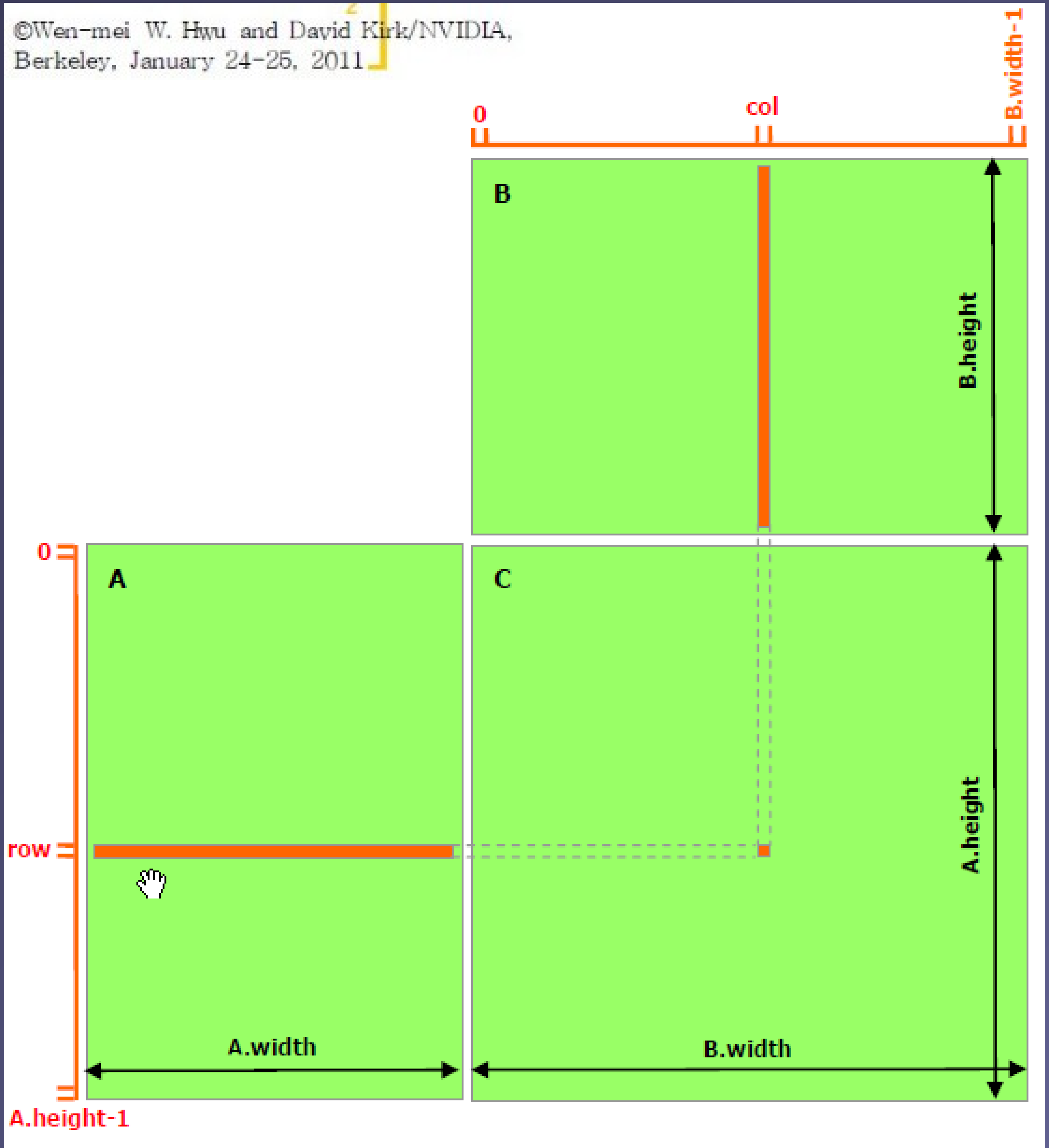
## Wrap-Up of CUDA/Outlook

# Error Handler used in Jason Sanders' book examples

```
static void HandleError( cudaError_t err,
                         const char *file,
                         int line ) {
    if (err != cudaSuccess) {
        printf( "%s in %s at line %d\n", cudaGetErrorString(
err ),
                file, line );
        exit( EXIT_FAILURE );
    }
}
#define HANDLE_ERROR( err ) (HandleError( err, __FILE__,
__LINE__ ))
```

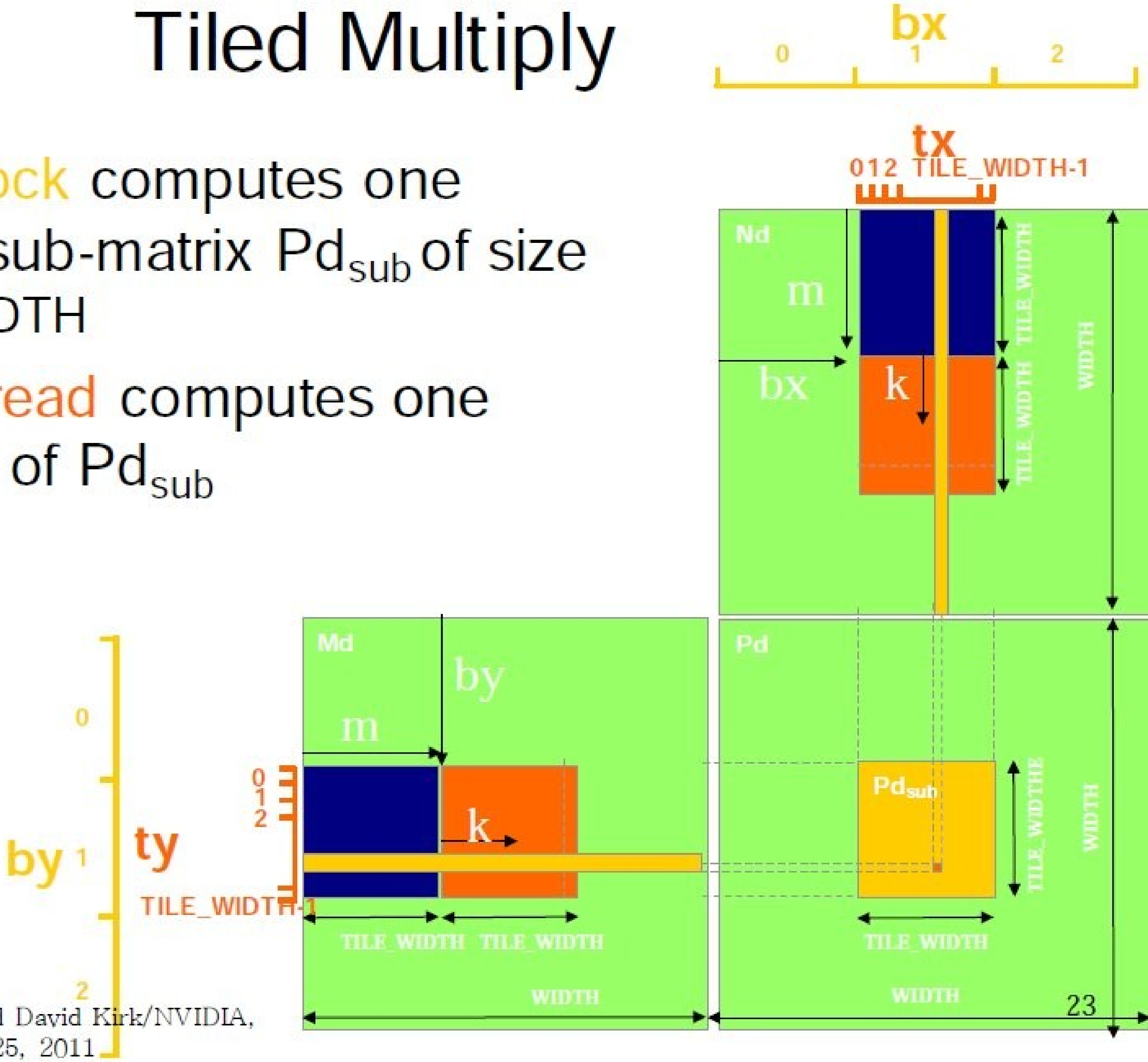
Make sure that this code appears somewhere in your source, or somewhere in a header you #include.

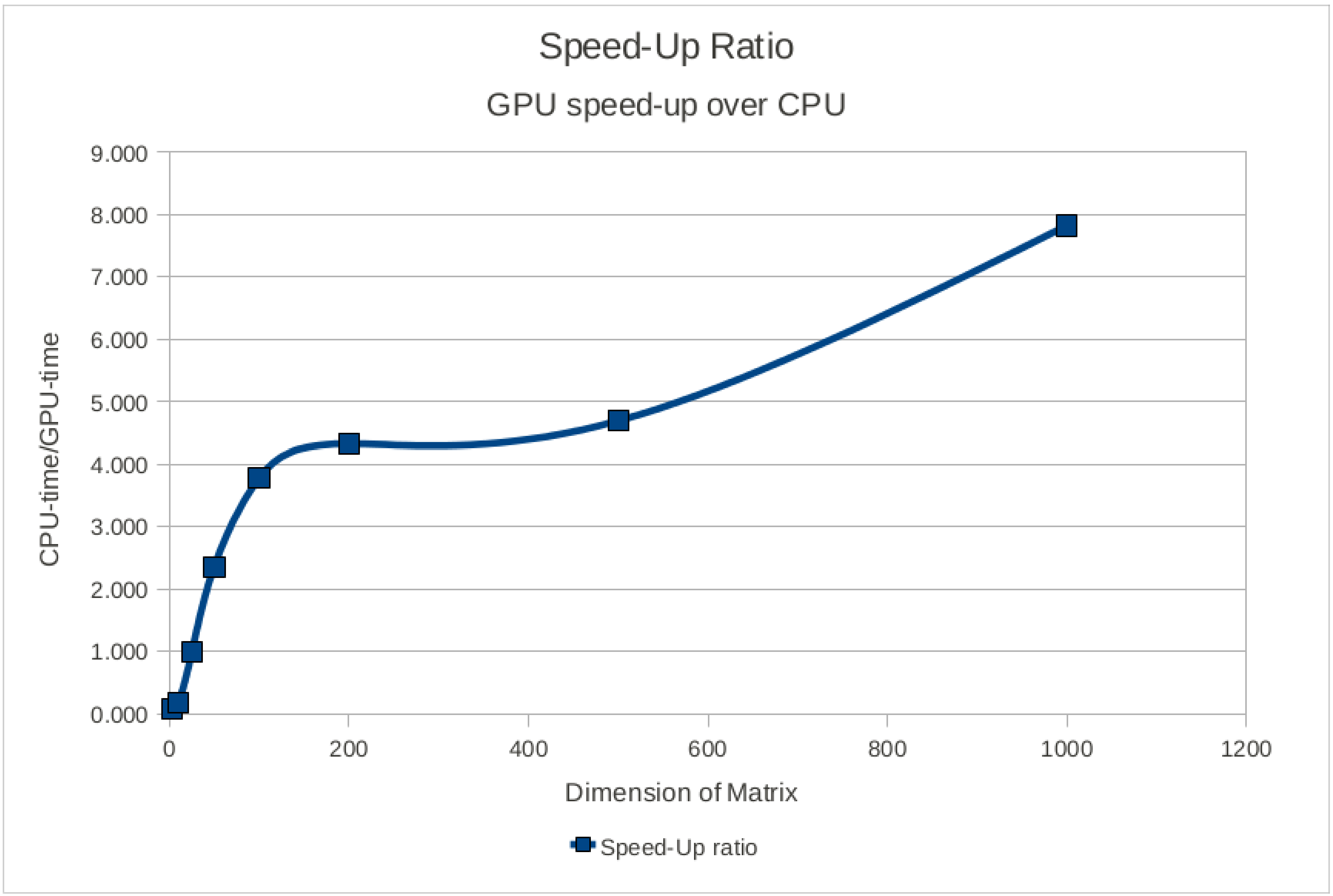
# Matrix Intuitive Multiply


©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011
0
col
B.width-1
B
B.height
0
A
C
A.height
row
A.width
B.width
A.height-1

# Tiled Multiply

- Each block computes one square sub-matrix $Pd_{sub}$ of size TILE_WIDTH
- Each thread computes one element of $Pd_{sub}$




Speed-Up Ratio
GPU speed-up over CPU
CPU-time/GPU-time
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0.000
0
200
400
600
800
1000
1200
Dimension of Matrix
Speed-Up ratio

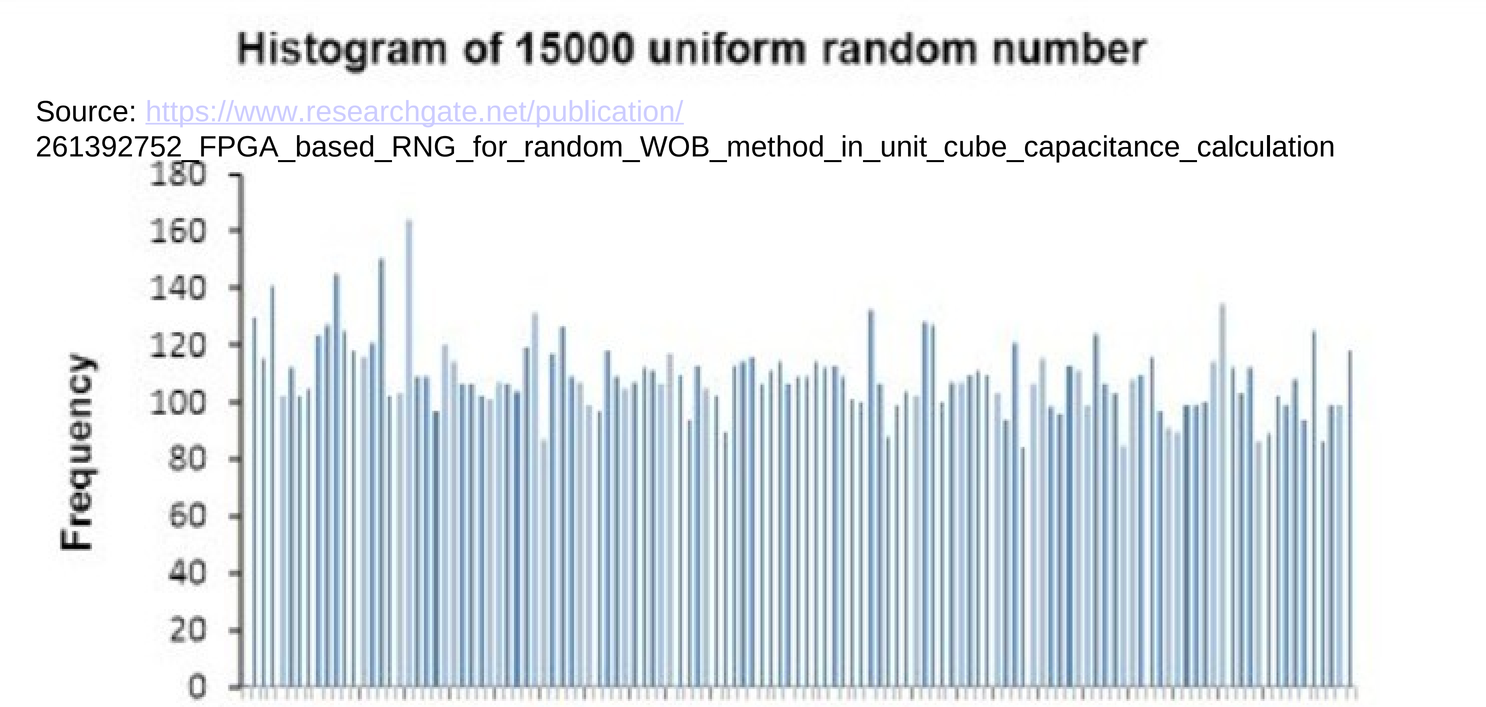
# Histogram Computation

## 8_histo

**Task:**

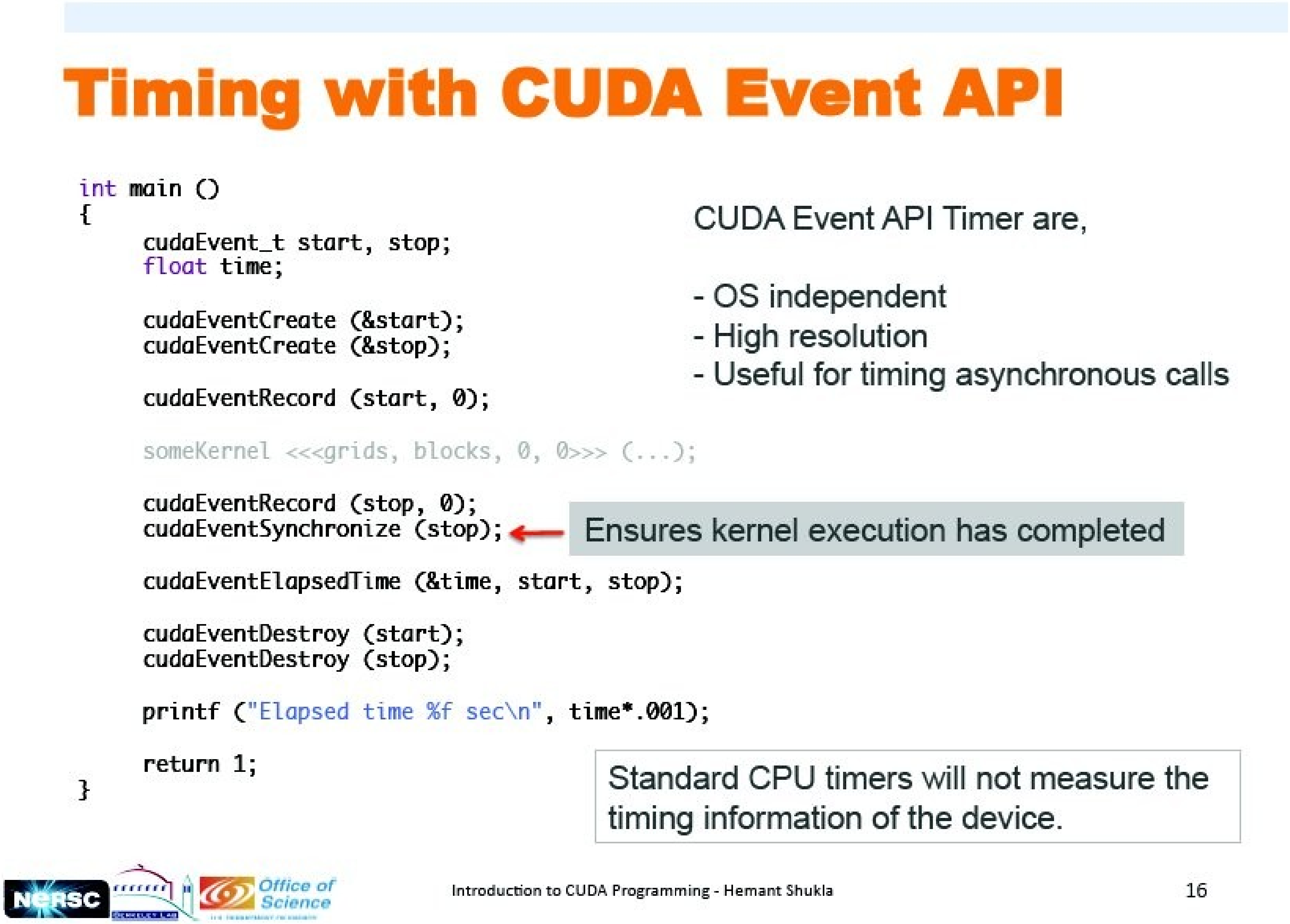
- 100 million integers $0 \le n_i < 256$;
- Randomly distributed with equal probability;
- What is the frequency (number) with which every integer occurs?

**Expect: equal distribution again, with fluctuations.**



Source: https://www.researchgate.net/publication/261392752_FPGA_based_RNG_for_random_WOB_method_in_unit_cube_capacitance_calculation

This Timing API is used in 8_histo/histo.cu !



# Timing with CUDA Event API

```
int main ()
{
      cudaEvent_t start, stop;
      float time;

      cudaEventCreate (&start);
      cudaEventCreate (&stop);

      cudaEventRecord (start, 0);

      someKernel <<<grids, blocks, 0, 0>>> (...);

      cudaEventRecord (stop, 0);
      cudaEventSynchronize (stop);

      cudaEventElapsedTime (&time, start, stop);

      cudaEventDestroy (start);
      cudaEventDestroy (stop);

      printf ("Elapsed time %f sec\n", time*.001);

      return 1;
}
```

CUDA Event API Timer are,

- OS independent
- High resolution
- Useful for timing asynchronous calls

Ensures kernel execution has completed

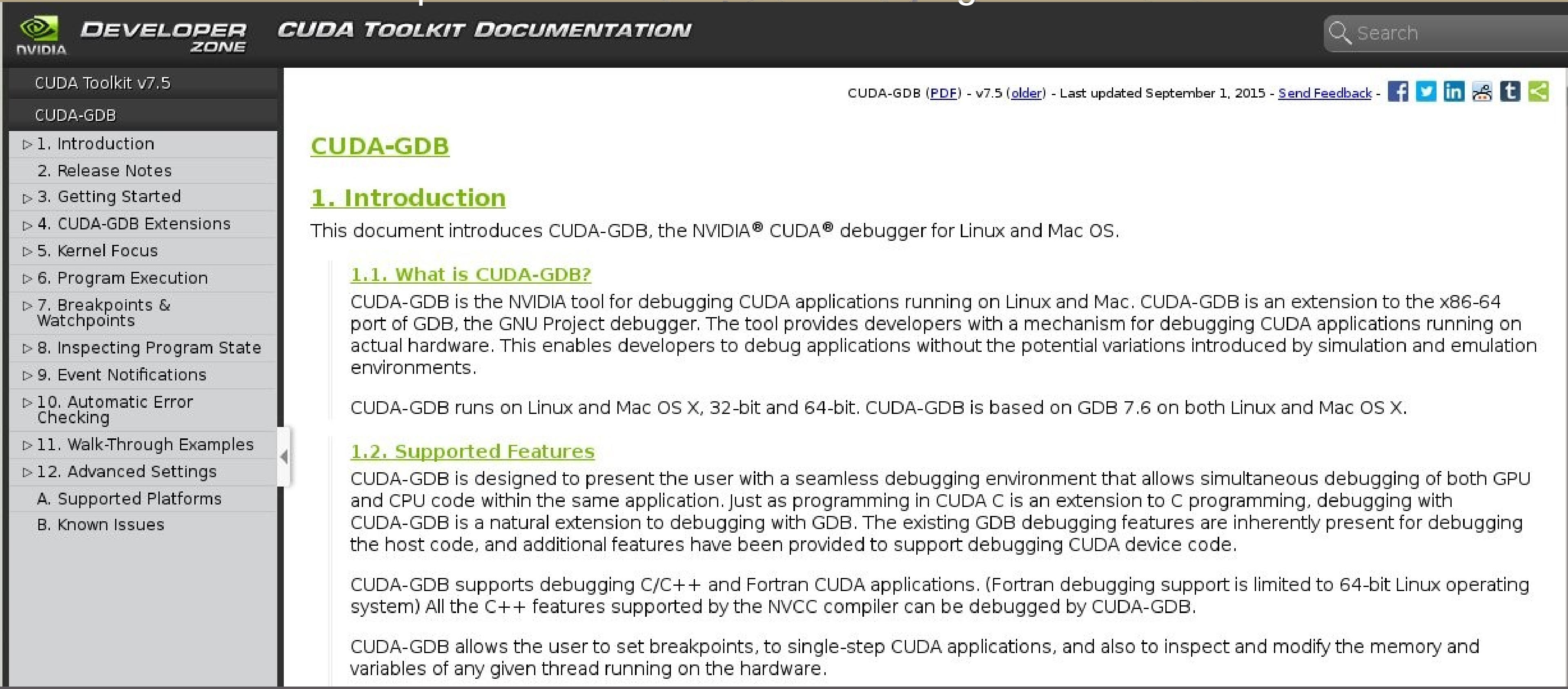
Standard CPU timers will not measure the timing information of the device.

# CUDA – GNU Debugger – CUDA-gdb

do not forget: nvcc -g -G … before running …

(not possible on kepler, login node has no GPU!)

http://docs.nvidia.com/cuda/cuda-gdb/index.html



CUDA-GDB (PDF) - v7.5 (older) - Last updated September 1, 2015 - Send Feedback -

## CUDA-GDB

## 1. Introduction

This document introduces CUDA-GDB, the NVIDIA® CUDA® debugger for Linux and Mac OS.

### 1.1. What is CUDA-GDB?

CUDA-GDB is the NVIDIA tool for debugging CUDA applications running on Linux and Mac. CUDA-GDB is an extension to the x86-64 port of GDB, the GNU Project debugger. The tool provides developers with a mechanism for debugging CUDA applications running on actual hardware. This enables developers to debug applications without the potential variations introduced by simulation and emulation environments.

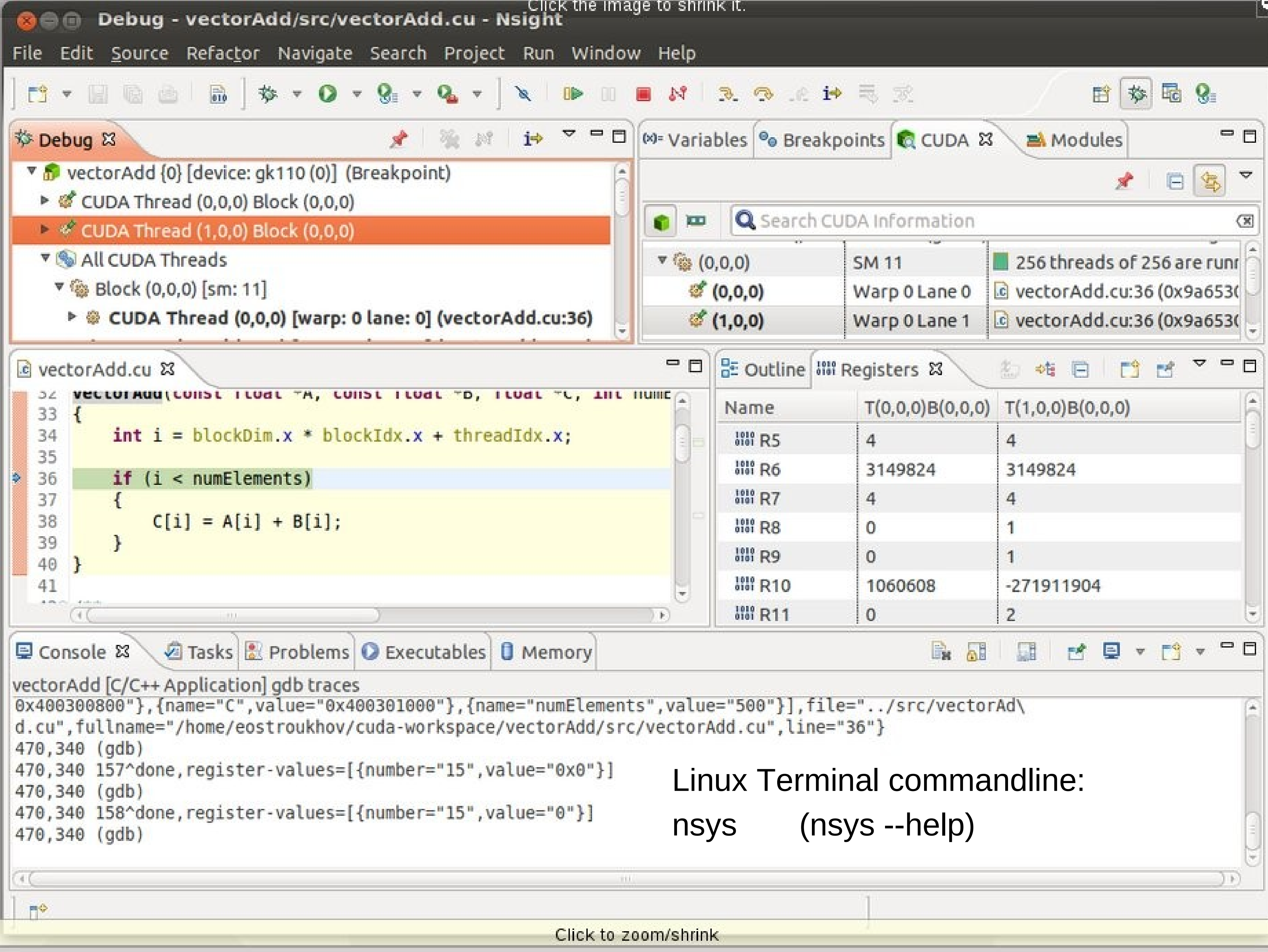
CUDA-GDB runs on Linux and Mac OS X, 32-bit and 64-bit. CUDA-GDB is based on GDB 7.6 on both Linux and Mac OS X.

### 1.2. Supported Features

CUDA-GDB is designed to present the user with a seamless debugging environment that allows simultaneous debugging of both GPU and CPU code within the same application. Just as programming in CUDA C is an extension to C programming, debugging with CUDA-GDB is a natural extension to debugging with GDB. The existing GDB debugging features are inherently present for debugging the host code, and additional features have been provided to support debugging CUDA device code.

CUDA-GDB supports debugging C/C++ and Fortran CUDA applications. (Fortran debugging support is limited to 64-bit Linux operating system) All the C++ features supported by the NVCC compiler can be debugged by CUDA-GDB.

CUDA-GDB allows the user to set breakpoints, to single-step CUDA applications, and also to inspect and modify the memory and variables of any given thread running on the hardware.



- vectorAdd {0} [device: gk110 (0)] (Breakpoint)
  - CUDA Thread (0,0,0) Block (0,0,0)
  - CUDA Thread (1,0,0) Block (0,0,0)
  - All CUDA Threads
    - Block (0,0,0) [sm: 11]
      - CUDA Thread (0,0,0) [warp: 0 lane: 0] (vectorAdd.cu:36)

|  |  |  |
|---|---|---|
| (0,0,0) | SM 11 | 🟩 256 threads of 256 are runr |
| (0,0,0) | Warp 0 Lane 0 | vectorAdd.cu:36 (0x9a653( |
| (1,0,0) | Warp 0 Lane 1 | vectorAdd.cu:36 (0x9a653( |

```
vectorAdd(const float *A, const float *B, float *C, int nume
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}

```

| Name | T(0,0,0)B(0,0,0) | T(1,0,0)B(0,0,0) |
|---|---|---|
| R5 | 4 | 4 |
| R6 | 3149824 | 3149824 |
| R7 | 4 | 4 |
| R8 | 0 | 1 |
| R9 | 0 | 1 |
| R10 | 1060608 | -271911904 |
| R11 | 0 | 2 |

vectorAdd [C/C++ Application] gdb traces

```
0x400300800"},{name="C",value="0x400301000"},{name="numElements",value="500"}],file="../src/vectorAd\
d.cu",fullname="/home/eostroukhov/cuda-workspace/vectorAdd/src/vectorAdd.cu",line="36"}
470,340 (gdb)
470,340 157^done,register-values=[{number="15",value="0x0"}]
470,340 (gdb)
470,340 158^done,register-values=[{number="15",value="0"}]
470,340 (gdb)
```

Linux Terminal commandline:

nsys (nsys --help)

# Wrapping Up 1

## Exercises afternoons

0. hello, device- first kernel call, hello world, GPU properties
1. add - vector addition using one thread in one block only
2. add-index - vector addition using blocks in parallel,
one thread per block only.
3. add-parallel - vector addition using all blocks and threads in parallel
4. dot - scalar product using shared memory of one block
only for reduction
5. dot-full - scalar product using shared memory and
atomic add across blocks
6. dot-perfect - scalar product; fat threads and final reduction on host.
**7.** matmul - matrix multiplication with tiled access shared memory.
**8.** histo - histogram using fat threads and atomic add
on shared and global memory, timing

# Wrapping Up 2

## Elements of CUDA C learnt:

| | |
|---|---|
| threadId.x , blockId.x, blockDim.x, gridDim.x | Threads, Blocks |
| (threadId.y, blockId.y, blockdim.y, gridDim.y | (matmul coming with 2D grids) |
| kernel<<<n,m>>> (...) | kernel calls |
| Kernel<<n,m,size>>(…) | kernel call with dyn. alloc. size |
| kernel<<<dimBlock,dimGrid>>>(…) | dim3 variable type (matmul) |
| __global__ | device code |
| __shared__ | shared memory on GPU |
| cudaMalloc / cudaFree | manage global memory of GPU |
| cudaMemcpy / cudaMemset | copy/set to or from memory |
| cudaGetDeviceProperties | get device properties in program |
| cudaEventCreate, cudaEventRecord,<br>cudaEventSynchronize, cudaEventElapsedTime,<br>cudaEventDestroy | CUDA profiling |
| AtomicAdd (on global or shared mem.) | atomic functions |

# Wrapping Up 3

## What we have not yet learnt...

| | |
|---|---|
| __constant__ | constant memory on GPU |
| __device__ | functions device to device |
| Intrinsic Functions ( __device__ type) | |
| https://docs.nvidia.com/cuda/cuda-math-api/group__CUDA__MATH__SINGLE.html#group__CUDA__MATH__SINGLE | |
| __host__ | functions host to host |
| More atomic functions | |
| cudaBindTexture | using texture memory |
| fat threads for 2D and 3D stencils | thread coalescence opt. |
| cudaStreamCreate, cudaStreamDestroy | working with CUDA streams |
| <<<n,m,size,s>>> | kernel call with streams s |
| using Tensor Cores | |
| ... | |