



ECE408 / CS483 / CSE408
Summer 2025

Applied Parallel Programming

Lecture 1: Introduction

Welcome to the Course!

- **course is online**
 - lectures are **live via Zoom**
 - **available on UIUC MediaSpace** after some delay
 - labs (aka MPs) & project are on your own out of class activities
 - **exams** are on Zoom—you **MUST HAVE A WORKING CAMERA** to participate
(0 credit if you do not)
- lecture **slides** available **on** the class **web page**



What Can You Expect to Learn?

- Learn to **program massively parallel processors** and achieve
 - High performance
 - Functionality and maintainability
 - Scalability across future generations
- Technical subjects
 - Parallel programming basics
 - Principles and patterns of parallel algorithms
 - Programming API, tools and techniques
 - Processor architecture features and constraints
 - Killer apps

Who Will Help You?

Professor:

Steve Lumetta

lumetta@illinois.edu

OH: Tu 9 a.m. – 11 a.m.



TA: **Hanwen “Howie” Liu**

hanwenl4@illinois.edu

OH: Sat 2 p.m. – 4 p.m.





Brief Bio of Steve Lumetta

- Ph.D. in CS from University of California, 1998
(clusters of symmetric multiprocessors)
- research
 - optical network architecture and reliability
 - processor architecture
 - digital system testing / test compression
 - accelerator architectures (GPU, FPGA)
 - computational genomics
 - programming education
- teaching
 - major contributor to CompE undergrad core (120, 220, 391)
 - Director of CompE Programs for ZJUI campus
- other stuff
 - co-founded company based on packetized optics in datacenters
 - contributed to other major efforts (ex: Blue Waters)



Know Where to Find Things!

- web page: <http://lumetta.web.engr.illinois.edu/408-Sum25/>
 - announcements and handouts
 - links to lecture slides/recordings
- discussions in **Piazza**
 - channel for electronic announcements
 - forum for Q&A – staff will read the board, and your classmates often have answers
- **Canvas** – grades
- **GitHub** – lab and project assignments

How to Compute Your Grade

- **exams: 50%** (equal weight per exam)
 - Exam 1: 9 July 7:00 to 9:00 p.m.
 - Exam 2: 8 August 1:00 p.m. to 3:00 p.m.
- **labs** (Machine Problems): **25%** (equal weight per lab)
 - passing tests (90% of each lab)
 - quiz on PrairieLearn (10% of each lab except 0—100% for that)
- **project: 25%**
 - demo/functionality/coding style: ~50%
 - performance with full functionality: ~50%
 - detailed rubric will be posted

***** IGNORE ANY COMPUTATION BY CANVAS *****

Policy Allows Two Late Submissions, One Drop

- Each student
 - automatically granted **TWO EXTENSIONS (only)**
 - 48 hours each
 - **MUST BE REQUESTED BEFORE DEADLINE**
 - can also use on all but last checkpoint of project
 - **NOT ON QUIZZES / EXAMS / FINAL PROJECT**
- Once you have used both extensions, **no more extensions**.
- **Lowest lab/quiz combination dropped** from final grade.

Class Definition of Academic Honesty

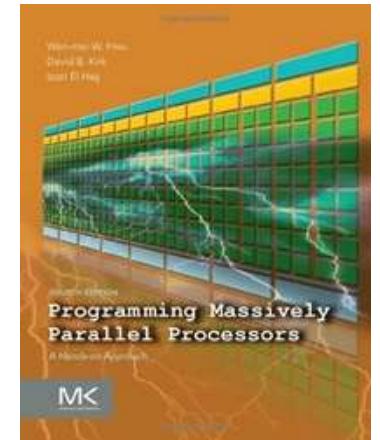
- Allowed and encouraged
 - discuss assignments with other students in the class.
 - Verbal advice/help from people who've already taken the course
- **NOT ALLOWED**
 - Any reference to assignments from previous terms or web postings
 - Any copying of non-trivial code is unacceptable
 - Non-trivial = more than a line or so
 - Copying includes reading someone else's code and then going off to write your own.
 - Those who have allowed copying will also be penalized.

Further Definition of Academic Honesty

- **Giving/receiving help on an exam** is unacceptable.
- Deliberately **sidestepping lab requirements is unacceptable**.
- Also **see AI policy** on the web page.
- Penalties for academic dishonesty:
 - **Zero** on the assignment/exam **for the first occasion**
 - Automatic **failure of the course for repeat offenses**

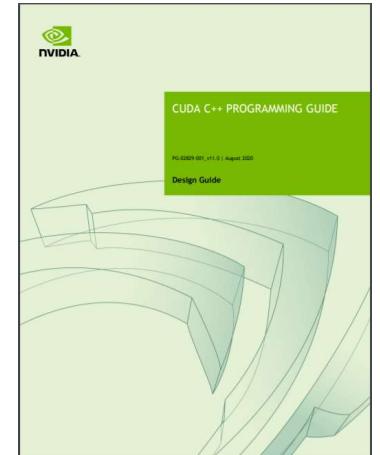
What You Should be Reading

W.W. Hwu, D.B. Kirk, I. El Hajj, “Programming Massively Parallel Processors – A Hands-on Approach,” Morgan Kaufman Publisher, 4th edition, 2022, ISBN 978-0323912310



NVIDIA, NVidia CUDA C++ Programming Guide,
(reference book)

<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>



Tentative Schedule for First Two Weeks

Week 1

- M: slide deck 1 (Introduction) & start slide deck 2 (CUDA Introduction)
- W: finish slide deck 2 & start slide deck 3 (CUDA Data Parallelism Model)

Lab 0: Fri 20 June

Week 2

- M: finish slide deck 3 & start slide deck 4 (CUDA Memory Model)
- W: finish slide deck 4, slide deck 5 (Tiled Matrix Multiply), & start slide deck 6 (Analysis of Tiling)

Lab 1: Tue 24 June
Quiz 0: Fri 27 June
Lab 2: Sat 28 June

How to Prepare for the Labs

- **Labs/project use Delta supercomputer** operated by NCSA
 - **obtain an account ASAP!** (follow link on class web page)
 - GitHub release repo includes details for setting up your environment
- Lab (MP) and project deadlines 8:00:00 p.m. US Central Time
- Lab/Quiz **workflow**:
 - #1: **Get base code** from GitHub.
 - #2: **Write code** & compile & run on Delta
 - #3: **Commit** to your repo to submit for grading.
 - #4: **Take quiz** on concepts AFTER completing lab.
- Lab 0 README includes instructions to get started

NCSA Delta Supercomputer

Delta is a dedicated, **ACCESS**-allocated resource designed by HPE and NCSA, delivering a highly capable GPU-focused compute environment for GPU and CPU workloads.



4-Way A40 GPU Compute Node Specs	
Specification	Value
Number of nodes	100
GPU	NVIDIA A40
GPUs per node	4
GPU Memory (GB)	48 DDR6
CPU	AMD Milan
CPU sockets per node	1
Cores per socket	64
Cores per node	64
Clock rate (GHz)	~ 2.45
RAM (GB)	256

Compile and Execute Labs on Delta

Let's go look at the instructions on the class web page...

Note...

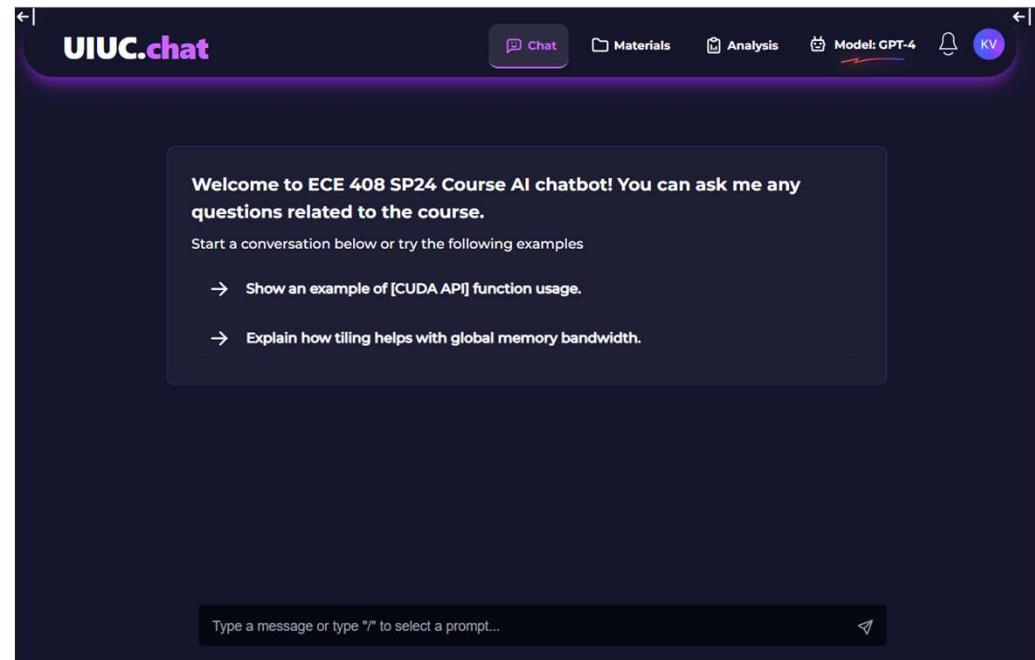
- Logging in to **Delta requires 2FA**.
- For Github,
 - we suggest a Personal Authentication Token (PAT), but
 - feel free to use an SSH key—set it up yourself.
- Delta's documentation:

<https://docs.ncsa.illinois.edu/systems/delta/en/latest/>

ChatBot Available—Use at Your Own Risk!

<https://www.uiuc.chat/ECE408SP25/chat>

- under development by CAII team at NCSA
- GPT-4 based (OpenAI API)
- Context Stuffing and Mega-Prompting (with course materials)
- **Read the course policy on outside tools on class web page before using the chatbot.**





New Millenium Witnessed a Paradigm Shift

- **In the 20th Century, we were able to understand, design, and manufacture what we could measure**
 - Physical instruments and computing systems allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes...

- 
- **In the 21st Century, we are able to understand, design, and create what we can compute**
 - Computational models are allowing us to see even farther, going back and forth in time, learn better, test hypothesis that cannot be verified any other way, create safe artificial processes...

Examples of Paradigm Shift

20th Century

- Small mask patterns
- Electronic microscope and Crystallography with computational image processing
- Anatomic imaging with computational image processing
- Audio conference call
- Broadcast advertisements
- Cars and GPS

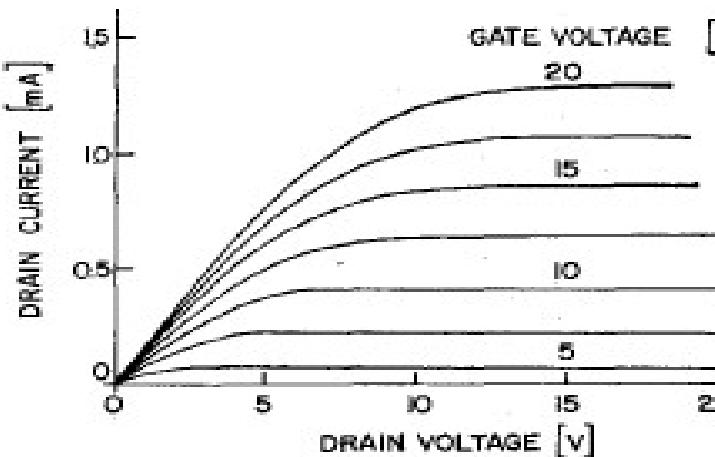
21st Century

- Optical proximity correction
- Computational microscope with initial conditions from Crystallography
- Metabolic imaging sees disease before visible anatomic change
- Tele-immersion
- AI-targeted advertisements integrated into streaming media / applications
- Self-driving electric taxis

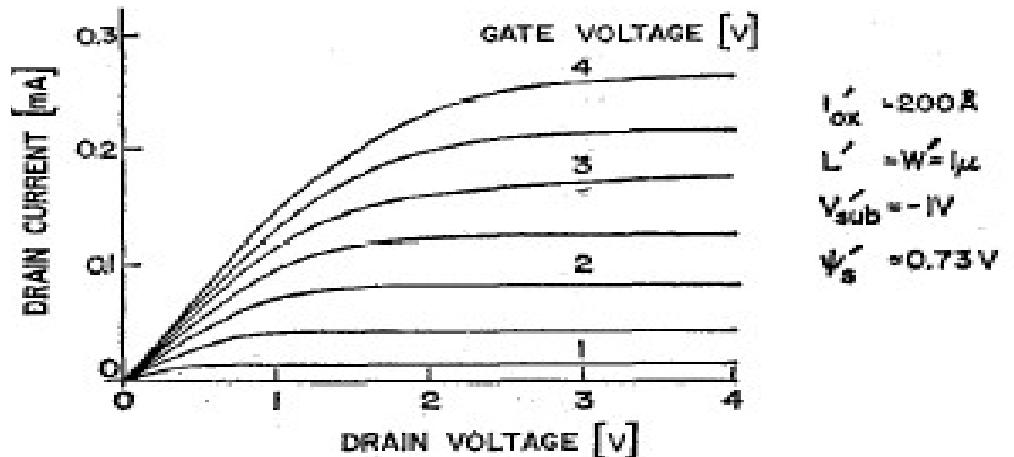


POST-DENNARD TECHNOLOGY PIVOT – PARALLELISM AND HETEROGENEITY

Dennard Scaling of MOS Devices



$t_{ox} = 1000\text{ \AA}$
 $L = W = 5\mu\text{m}$
 $V_{sub} = -7\text{ V}$
 $\psi_s = 0.65\text{ V}$



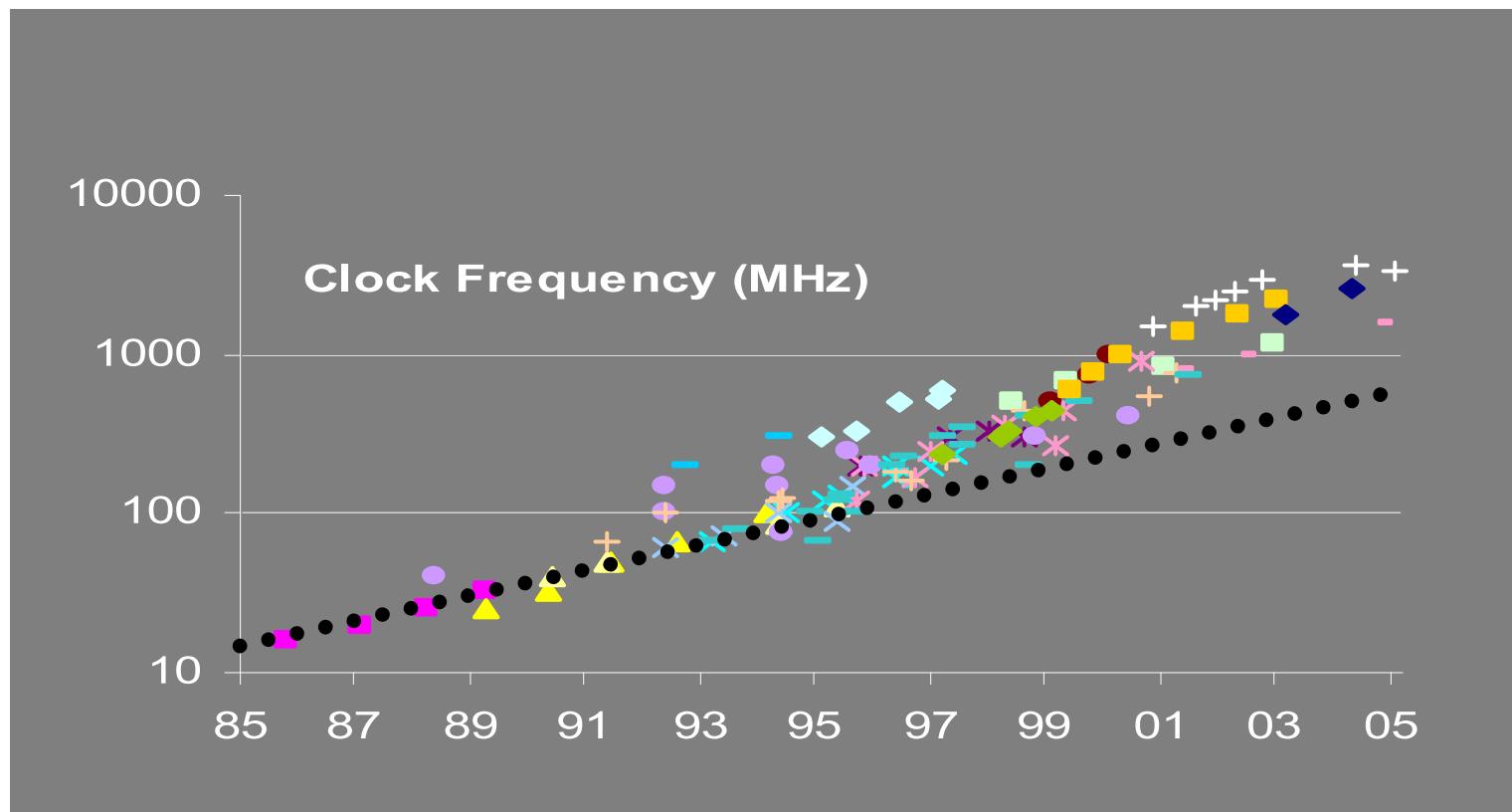
$t'_{ox} = 200\text{ \AA}$
 $L' = W' = 1\mu\text{m}$
 $V'_{sub} = -1\text{ V}$
 $\psi'_s = 0.73\text{ V}$

In this ideal scaling, as $L \rightarrow \alpha L$

- $V_{DD} \rightarrow \alpha V_{DD}$, $C \rightarrow \alpha C$, and $I \rightarrow \alpha I$
- Delay = CV_{DD}/I scales by α , so frequency $f \rightarrow 1/\alpha$
- Power for each transistor is CV^2f and scales by α^2
- keeping total power constant for same chip area

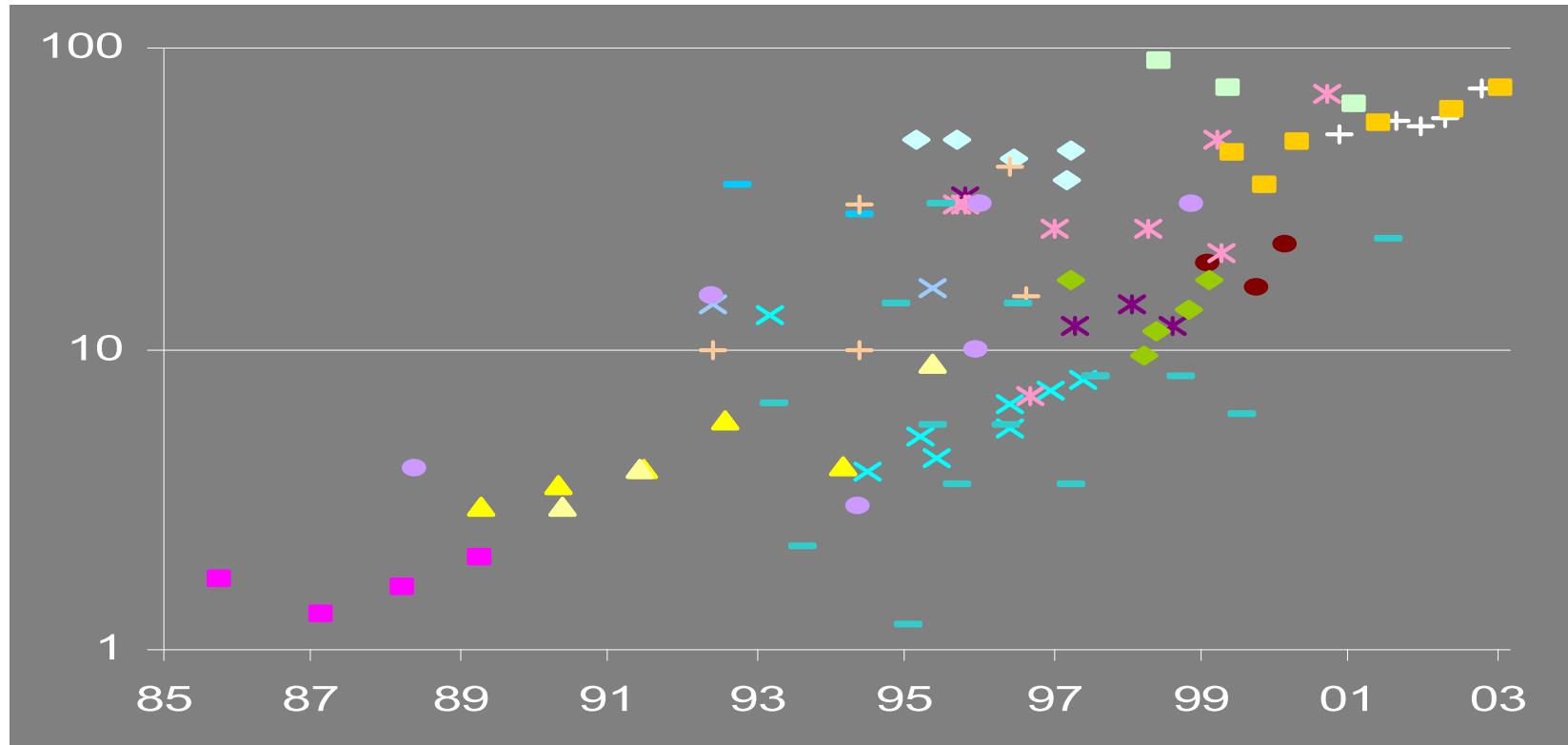
JSSC Oct 1974, page 256

Frequency Scaled Too Fast 1993-2003



Total Processor Power Increased

(super-scaling of frequency and chip size)



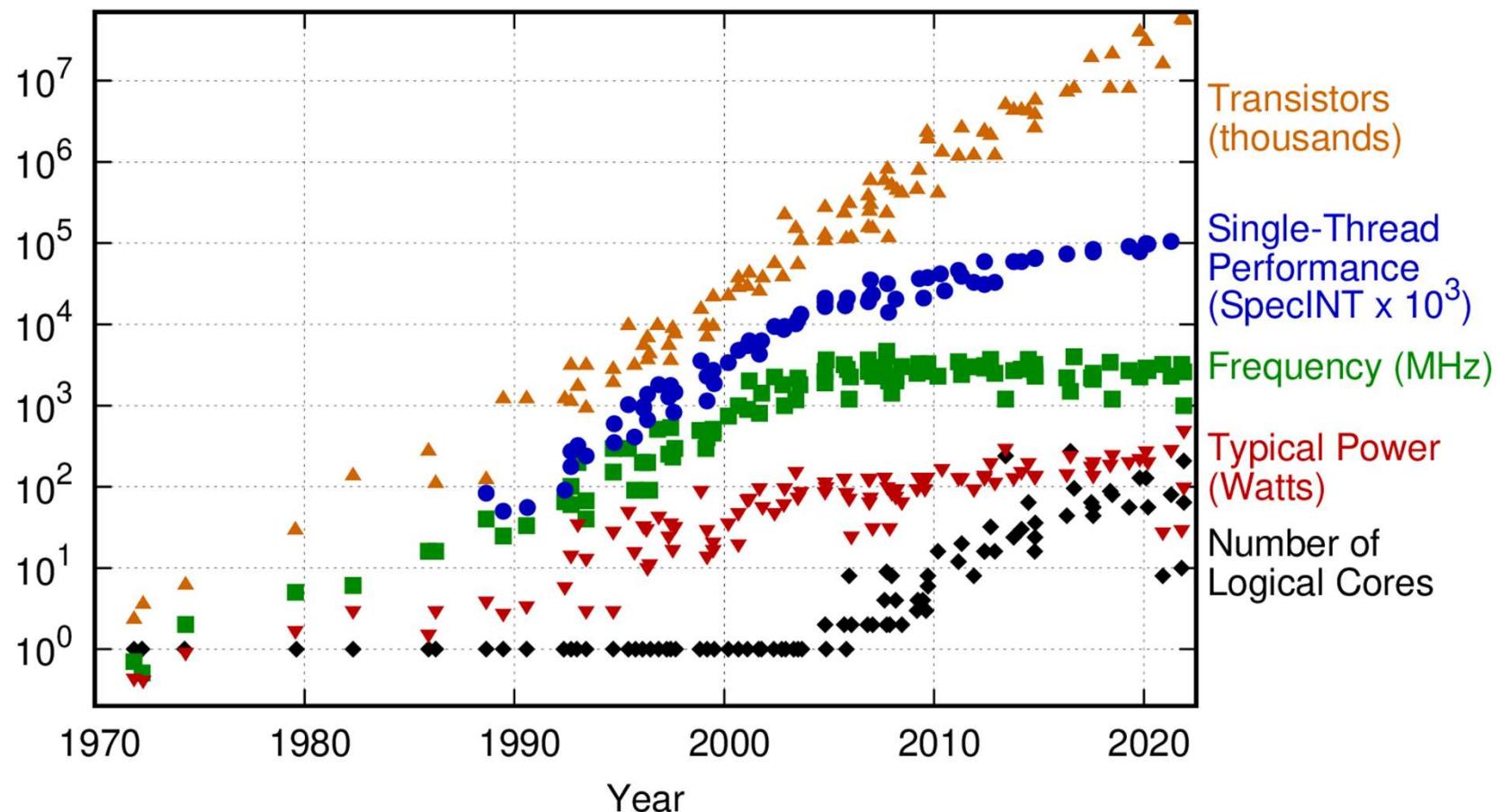


But Our Chips (Mostly) Still Don't Burn

So what changed?

- multiple cores
- more moderate clock frequencies
- heavy use of vector execution
- employ both latency-oriented and throughput-oriented cores
- 3D packaging for more memory bandwidth

Microprocessor Trends

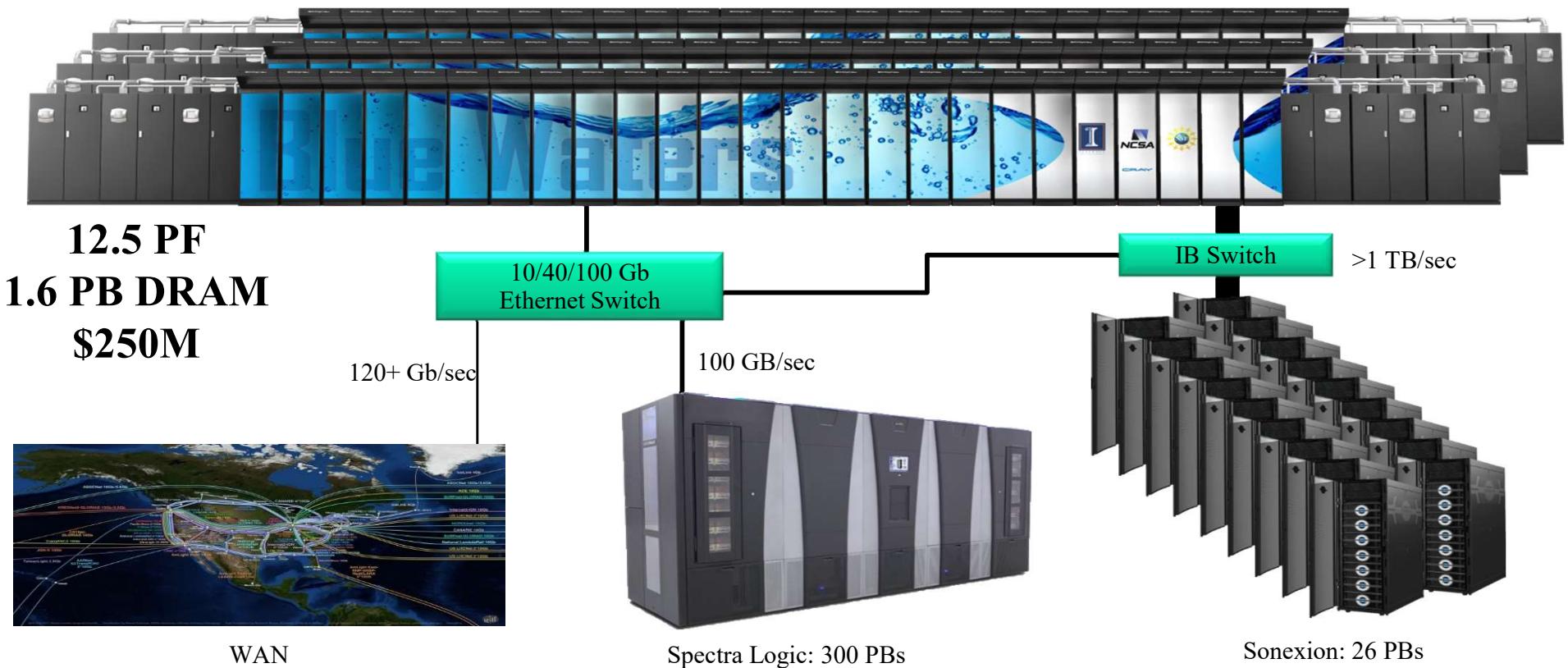


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

<https://github.com/karlrupp/microprocessor-trend-data>

Blue Waters Computing System

Operational at Illinois since 3/2013 **49,504 CPUs -- 4,224 GPUs**



Cray XK7 Compute Node

XK7 Compute Node Characteristics

AMD Series 6200 (Interlagos)

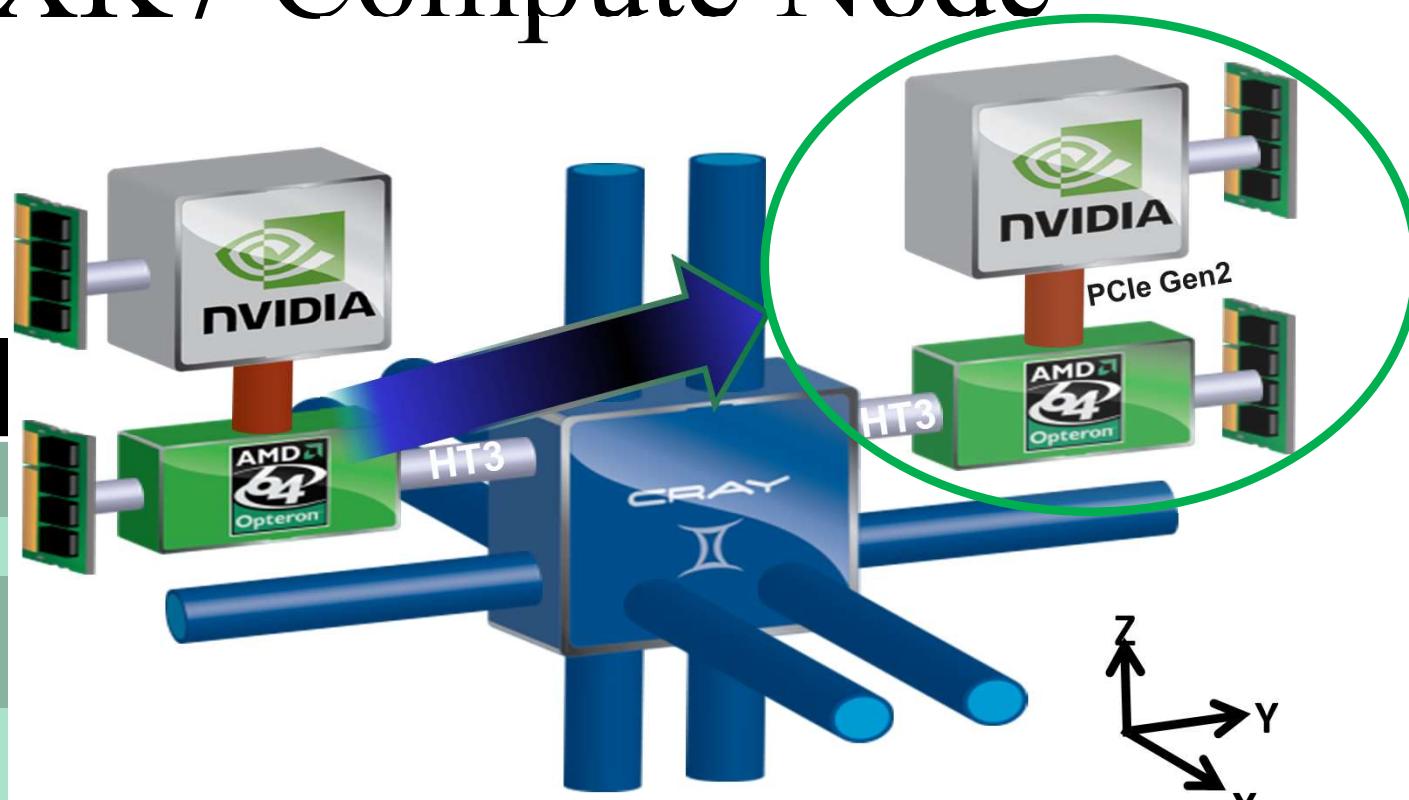
NVIDIA Kepler

Host Memory
32GB
1600 MT/s DDR3

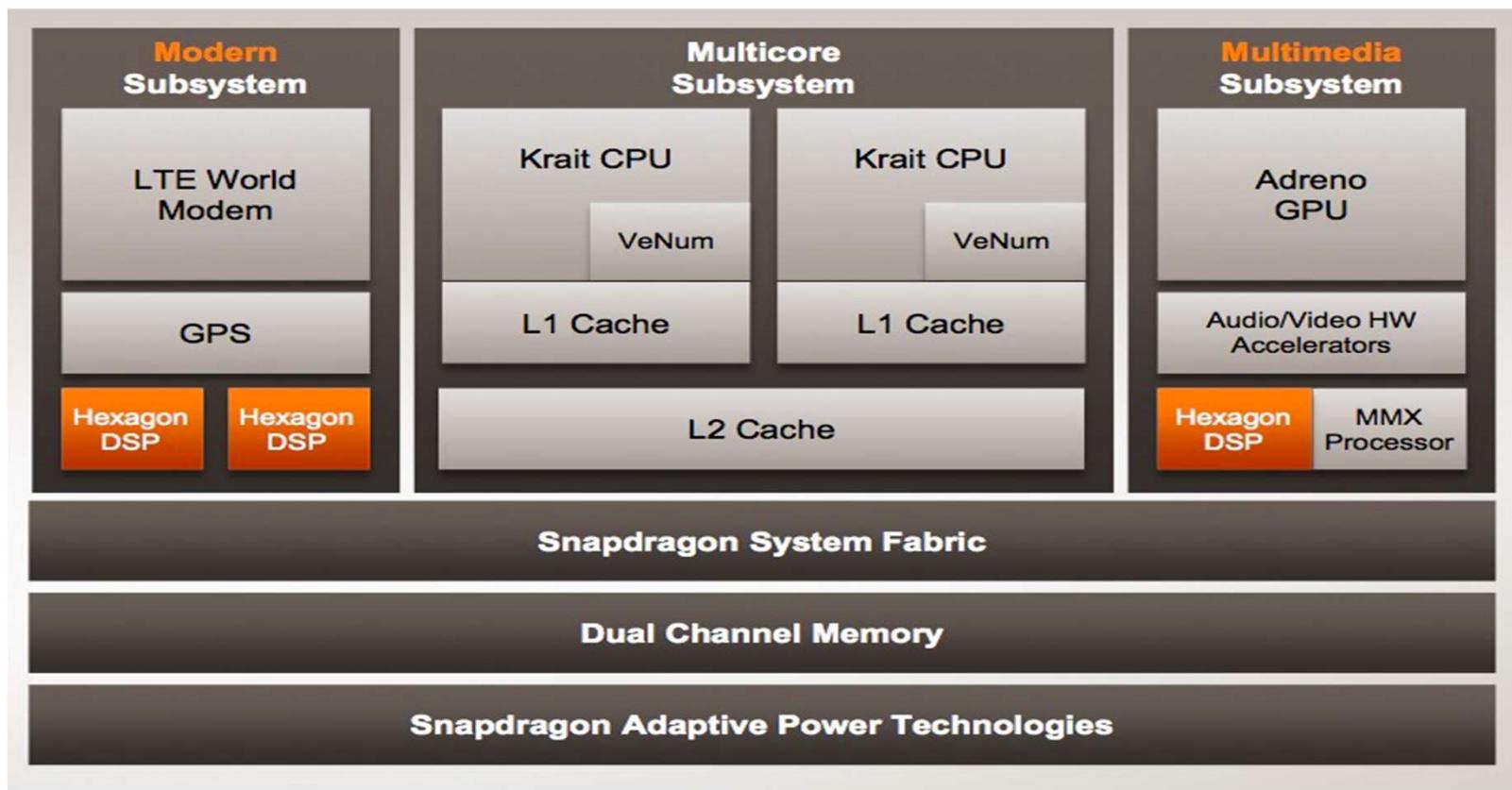
NVIDIA Tesla X2090 Memory
6GB GDDR5 capacity

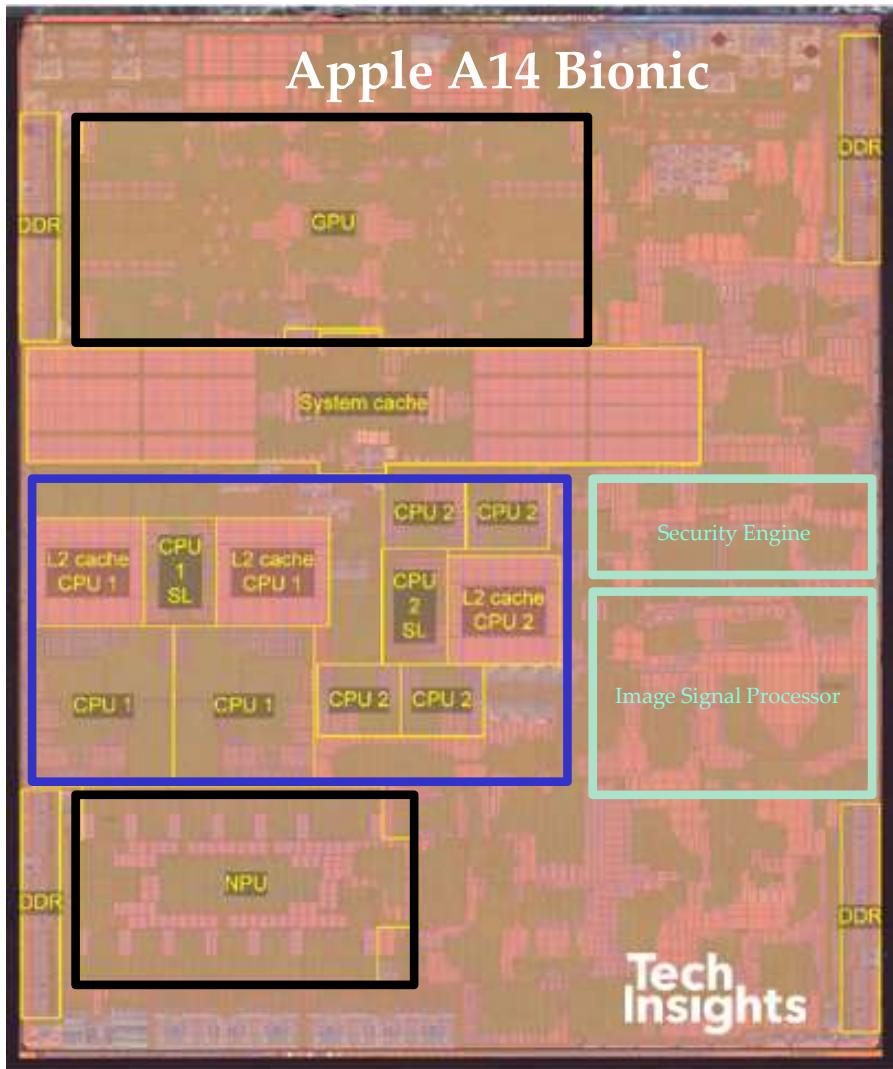
Gemini High Speed Interconnect

Keplers in final installation

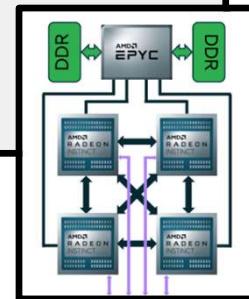


Qualcomm SoC for Mobile



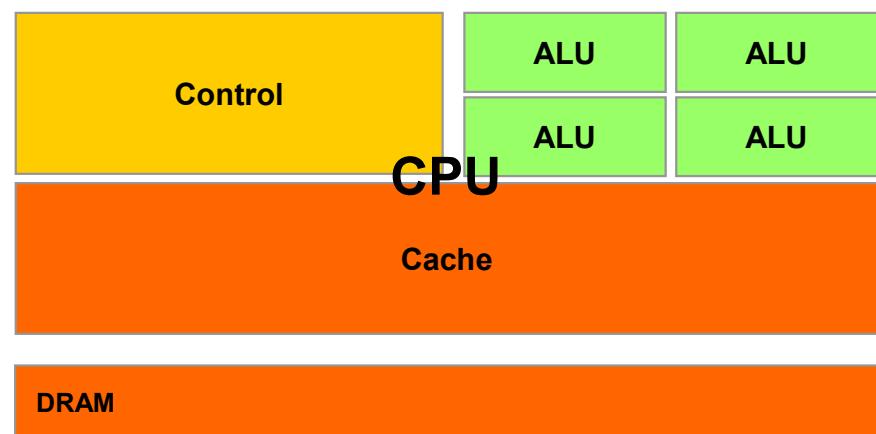


From the smallest to the largest,
computing today involves
accelerators architectures.



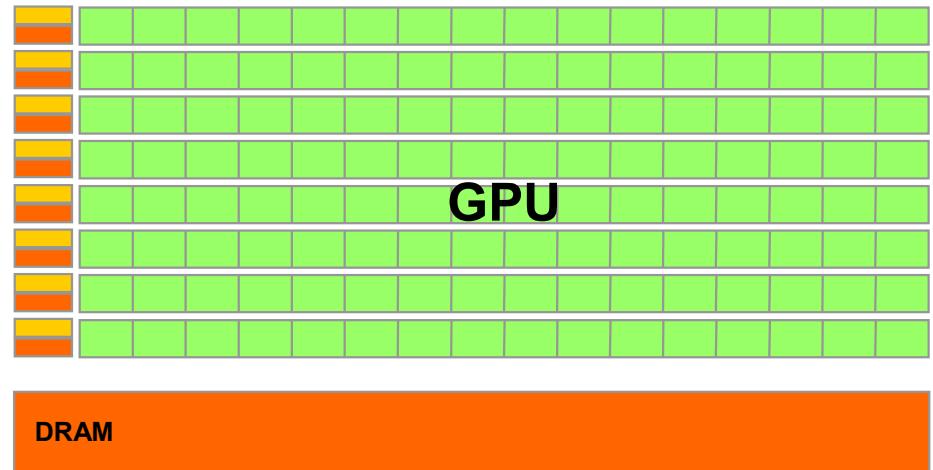
CPUs: Latency Oriented Design

- High clock frequency
- Large caches
 - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
 - Branch prediction for reduced branch latency
 - Data forwarding for reduced data latency
- Powerful ALU
 - Reduced operation latency



GPUs: Throughput Oriented Design

- Moderate clock frequency
- Small caches
 - To boost memory throughput
- Simple control
 - No branch prediction
 - No data forwarding
- Energy efficient ALUs
 - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



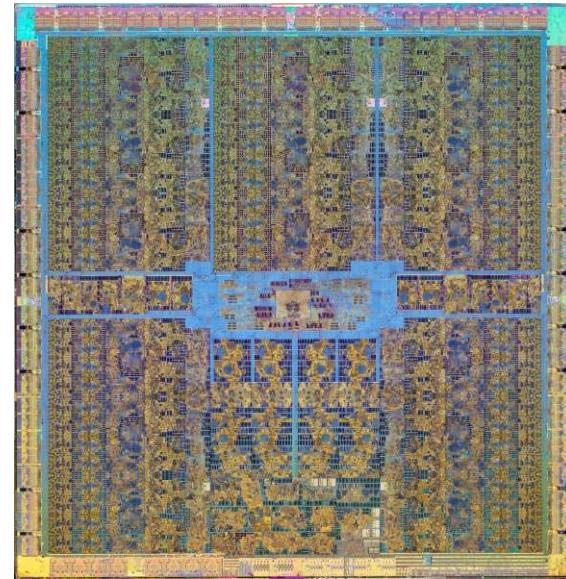
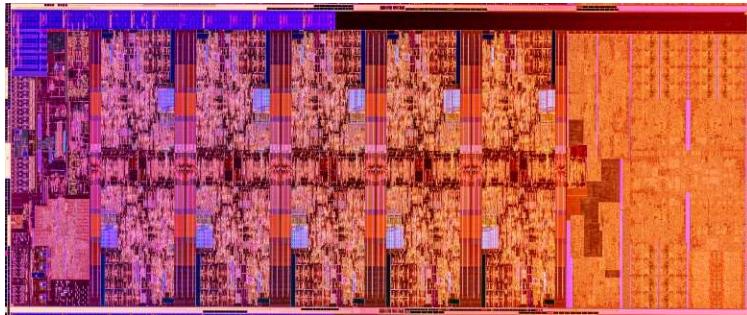


Winning Strategies Use Both CPU and GPU

- CPUs for sequential parts where latency hurts
 - CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
 - GPUs can be 10+X faster than CPUs for parallel code

CPU vs GPU

- 10th Gen Intel Core processor
 - 10 cores silicon
 - 14 nm process
- NVIDIA GK110
 - 2,880 CUDA cores
 - 28 nm process



Winning Strategies Use Both CPU & GPU

- CPUs for sequential parts where latency hurts
 - CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
 - GPUs can be 10+X faster than CPUs for parallel code



Heterogeneous Parallel Computing Applications

Financial Analysis

Scientific Simulation

Engineering Simulation

Data Intensive Analytics

Medical Imaging

Digital Audio Processing

Digital Video Processing

Computer Vision

Machine Learning

Electronic Design Automation

Biomedical Informatics

Statistical Modeling

Ray Tracing Rendering

Interactive Physics

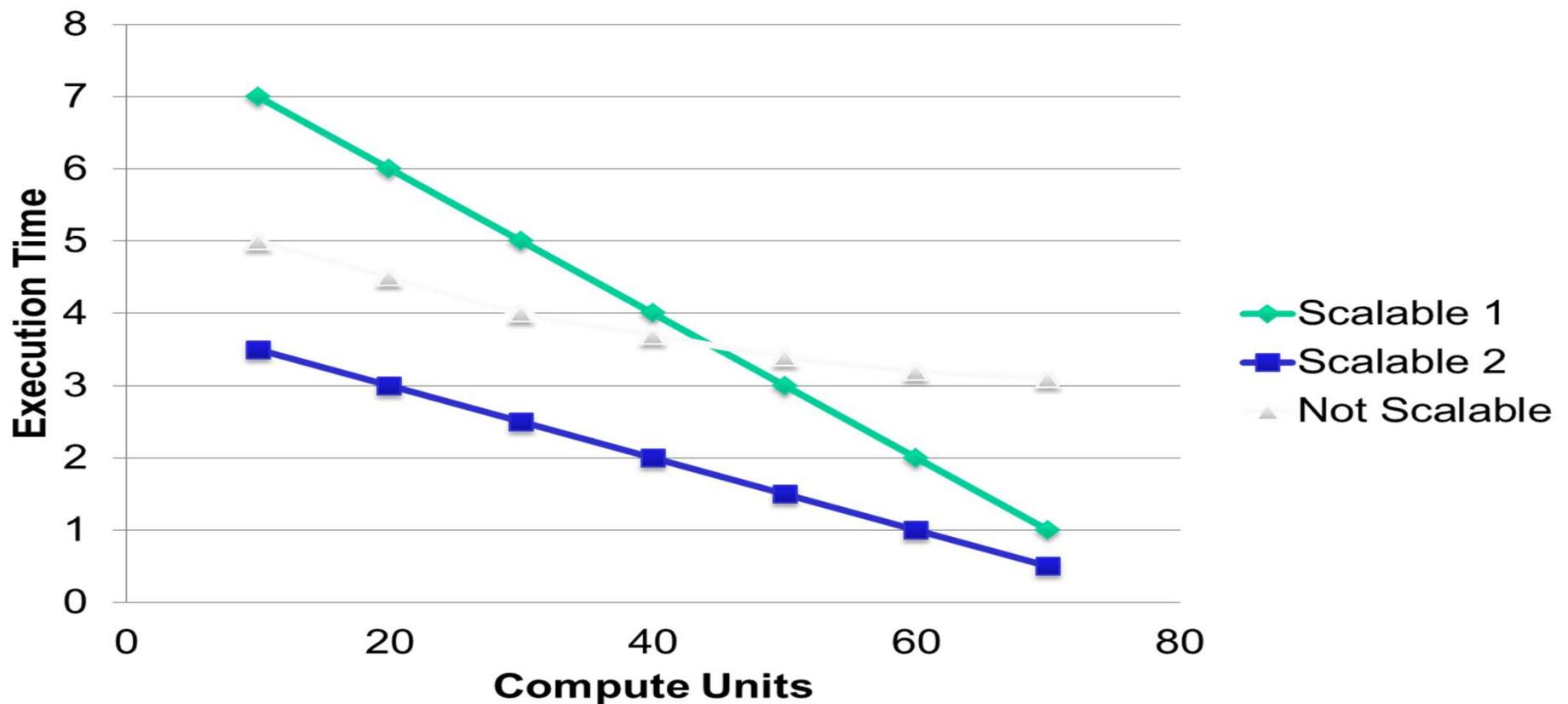
Numerical Methods



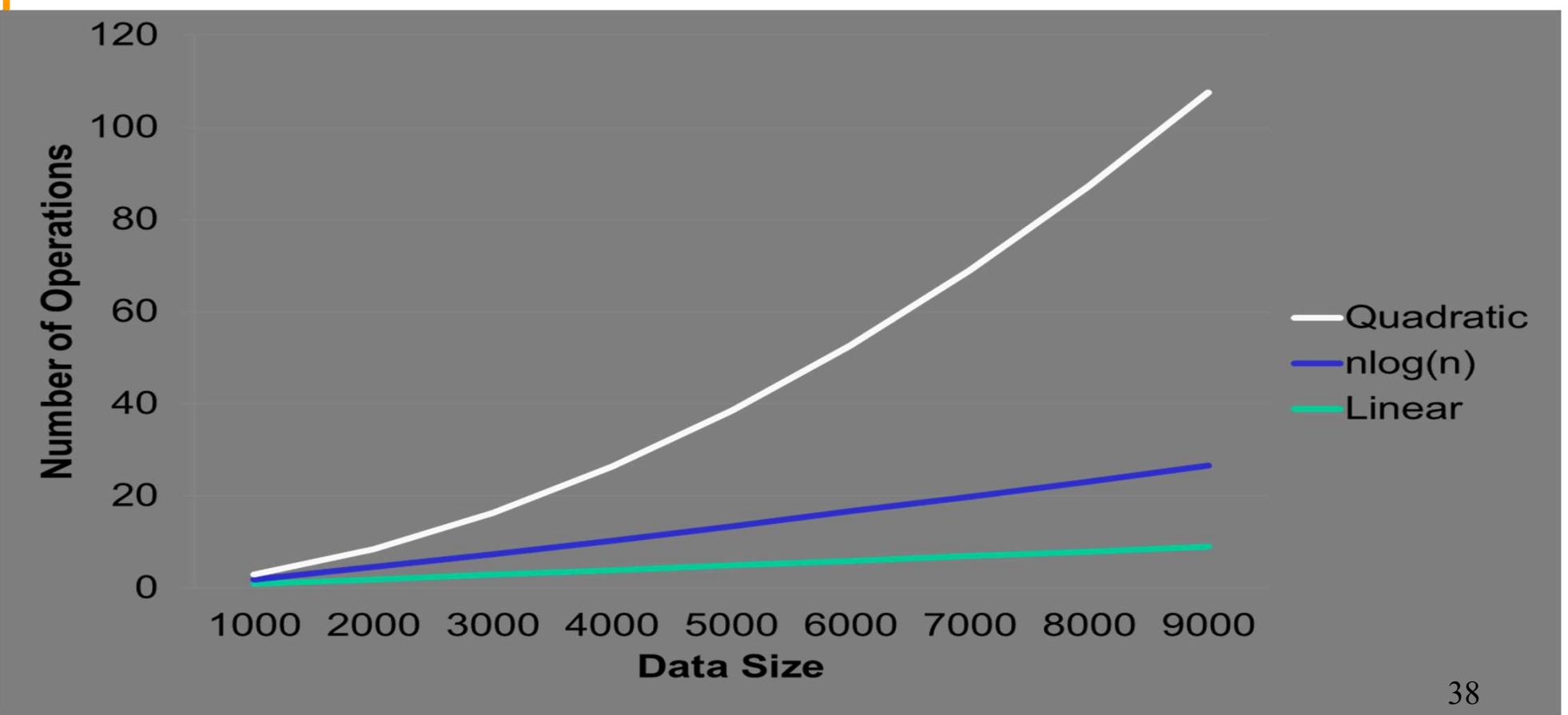
Parallel Programming Work Flow

- Identify compute intensive parts of an application
- Adopt/create scalable algorithms
- Optimize data arrangements to maximize locality
- Performance Tuning
- Pay attention to code **portability, scalability, and maintainability**

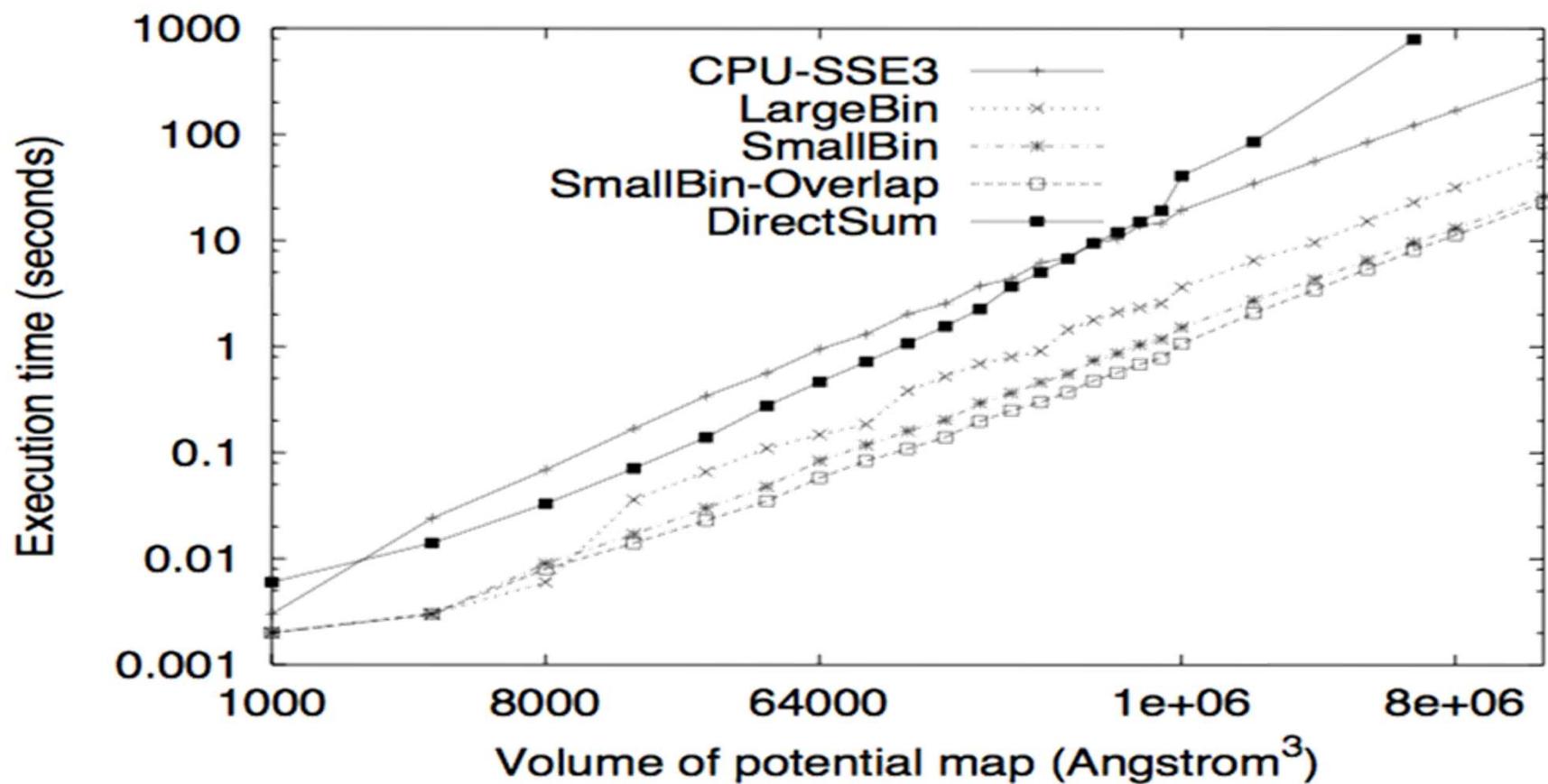
Parallelism Scalability



Algorithm Complexity and Data Scalability



Data Scalability in Particle-Mesh Algorithms





Massive Parallelism - Regularity



Load Balance

- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish



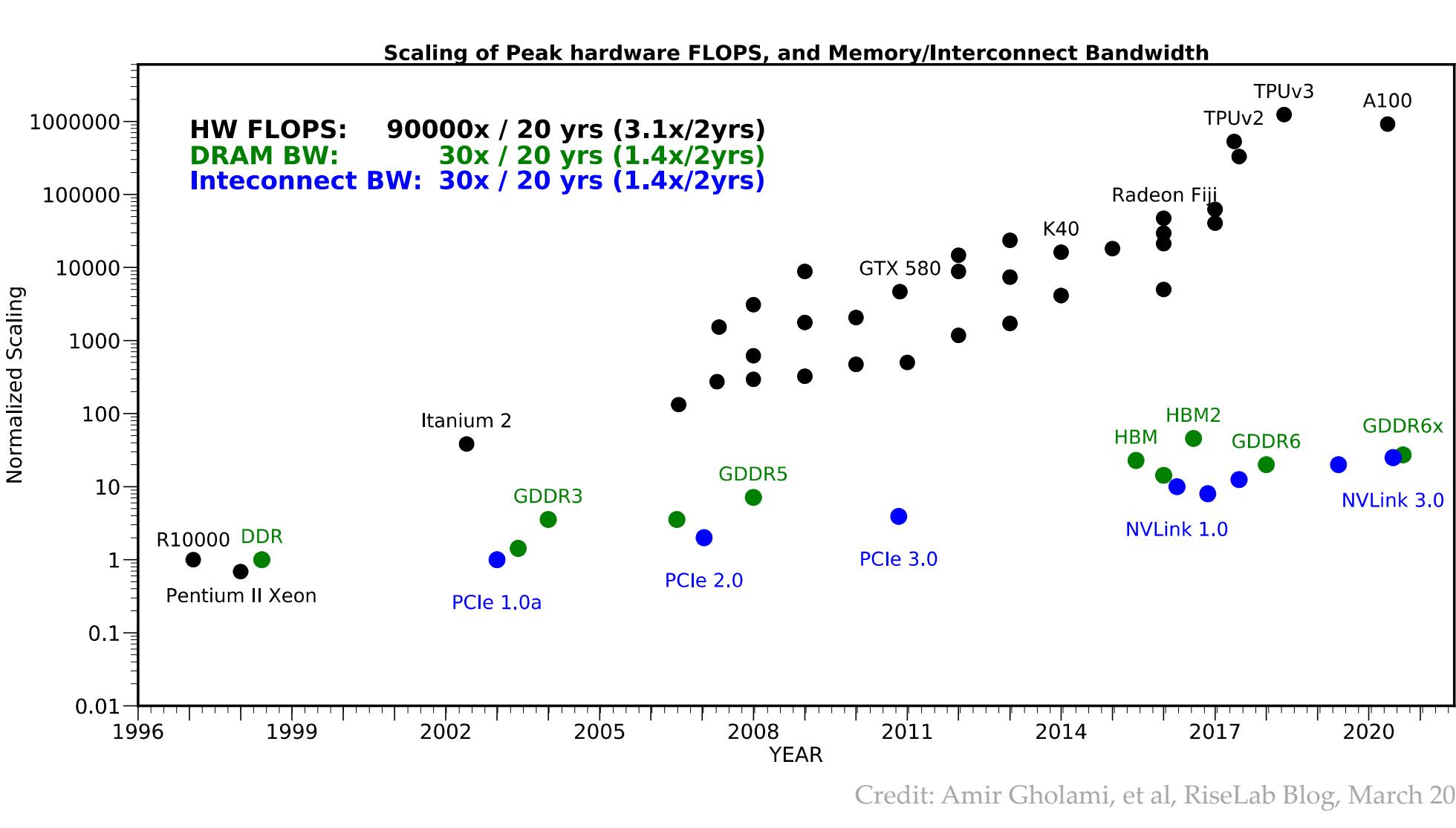
Global Memory Bandwidth

Ideal



Reality





Credit: Amir Gholami, et al, RiseLab Blog, March 2021

Conflicting Data Accesses Cause Serialization and Delays

- Massively parallel execution cannot afford serialization



- Contentions in accessing critical data causes serialization



What is the stake?

- Scalable and portable software lasts through many hardware generations

Scalable algorithms and libraries can be the best legacy we can leave behind from this era



ANY MORE QUESTIONS?