ECE408/CS483/CSE408 Spring 2023

Applied Parallel Programming

Lecture 1: Introduction

Before We Get Started

- Welcome to the course!
- The course is taught in mixed-mode (in-person and on-line)
  - Lectures are in-class and are streamed in real-time via zoom
  - They also will be recorded and posted on-line
  - Labs (MPs) & Projects are on your own out of class activities
  - Midterm exams are on-line
- Lecture slides will be posted on-line prior to the lecture on the course's wiki page
  - https://wiki.illinois.edu/wiki/display/ECE408

People

Instructor: **Prof. Volodymyr Kindratenko (kindrtnk@illinois.edu)**

TAs:	**Xiaoyu Ma (xiaoyum2@illinois.edu)**	**(Labs TA)**
	Xiyue Zhu (xiyuez2@illinois.edu)	**(Project TA)**
	Huili Tao (huilit2@illinois.edu)	

RAI administrator: **Andrew Schuh (aschuh@illinois.edu)**

About Prof. V. Kindratenko

- Ph.D. from University of Antwerp, Belgium, 1997
- At NCSA since 1997
  - Past: Director of Innovative Systems Lab
  - Current: Director of the Center for AI Innovation
- Research: Computing Systems, HPC, Computational Accelerators (FPGAs, GPUs), ML systems & applications

AC – first GPU HPC cluster built in 2008 (used to teach this course too)

- 32 S1070 GPUs

HAL – first AI-oriented cluster built in 2018

- 64 V100 GPUs

Course Goals

- 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

Web Resources

- wiki space
  - **https://wiki.illinois.edu/wiki/display/ECE408**
  - Links to lecture slides/recordings
  - Links to labs/projects
- web board discussions in Campuswire
  - Channel for electronic announcements
  - Forum for Q&A – staff will read the board, and your classmates often have answers
- Canvas – grades & exams & lab quizzes & project reports

Grading

- Exams: 40%
  - Midterm 1: 20% -- ~ first 10+ lectures
  - Midterm 2: 20% -- ~ the remaining lectures
- Labs (Machine Problems): 35%
  - Passing Datasets 90%
  - Correct answers to questions
  - Lowest graded lab will be dropped
- Project: 25%
  - Demo/Functionality/Coding Style: ~50%
  - Performance with full functionality: ~50%
  - Detailed Rubric will be posted

Academic Honesty

- You are allowed and encouraged to discuss assignments with other students in the class. Getting verbal advice/help from people who've already taken the course is also fine.
- Any reference to assignments from previous terms or web postings is unacceptable.
- 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.

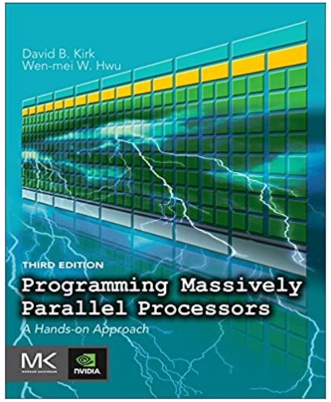
Academic Honesty (cont'd)

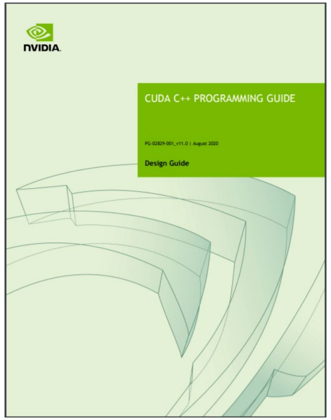
- Giving/receiving help on an exam is unacceptable.
- Deliberately sidestepping the lab requirements is unacceptable.
- Penalties for academic dishonesty:
  - Zero on the assignment/exam for the first occasion
  - Automatic failure of the course for repeat offenses

Text/Notes

1. D. Kirk and W. Hwu, “Programming Massively Parallel Processors – A Hands-on Approach,” Morgan Kaufman Publisher, 3rd edition, 2016, ISBN 978-0123814722

2. NVIDIA, *NVidia CUDA C Programming Guide*, version 7.5 or later (reference book) https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html





Tentative Schedule

- "Week 1":
  - Tuesday: Lecture 1: Introduction
  - Thursday: Lecture 2: CUDA Intro
  - Release: Lab 0, Installation, Test account
- "Week 2":
  - Tuesday: Lecture 3: Data Parallelism Model
  - Thursday: Lecture 4: CUDA Memory Model
  - Due: Lab 0, Installation, Test account
  - Release: Lab 1, Vector Addition
- Week 3:
  - Tuesday: Lecture 5: CUDA Memory Model
  - Thursday: Lecture 6: Performance Considerations
  - Due: Lab 1, Vector Addition
  - Release: Lab 2, Simple Matrix Multiply

all *Labs (MPs)* and *Project Milestones* (PMs) are due on Fridays at 8:00pm US Central Time

RAI

- Framework for submitting labs (MPs) and projects and grading them
  - You will receive an email with your RAI account instructions
  - Lab 0 will include all the details about deploying and using RAI client
- All Labs (MPs) and Project Milestones (PMs) are due on
  - Fridays at 8:00pm US Central Time
- Lab workflow:
  - #1: Get base code from GitHub → write code & compile & run (using RAI) → submit final code for grading (via RAI)
  - #2: Answer questions on Canvas AFTER #1 is completed
- **Lab 0 will have all the instructions to get started with RAI**

A major paradigm shift

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

A major paradigm shift

- **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

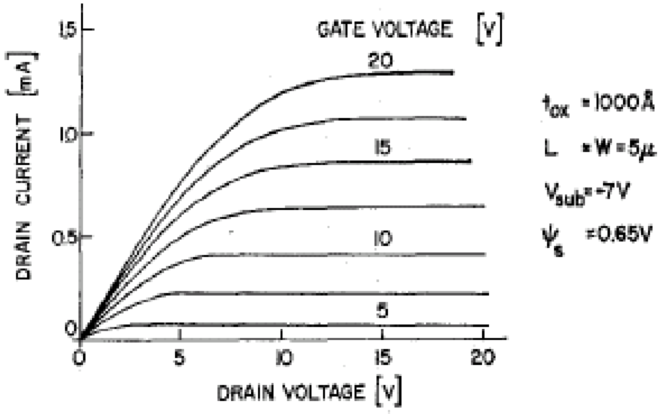
20th Century

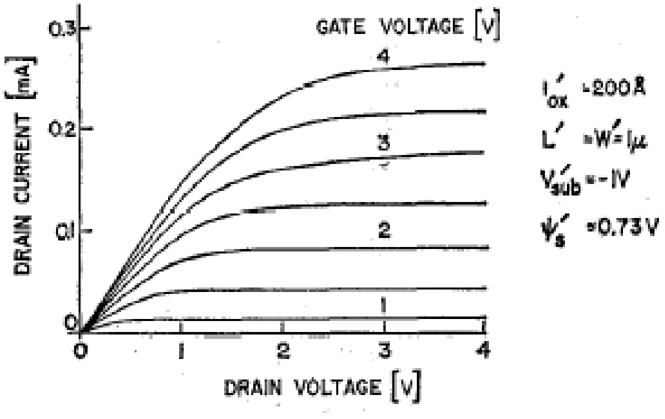
- Small mask patterns
- Electronic microscope and Crystallography with computational image processing
- Anatomic imaging with computational image processing
- Teleconference
- GPS

21st Century

- Optical proximity correction
- Computational microscope with initial conditions from Crystallography
- Metabolic imaging sees disease before visible anatomic change
- Tele-immersion
- Self-driving cars

Dennard Scaling of MOS Devices

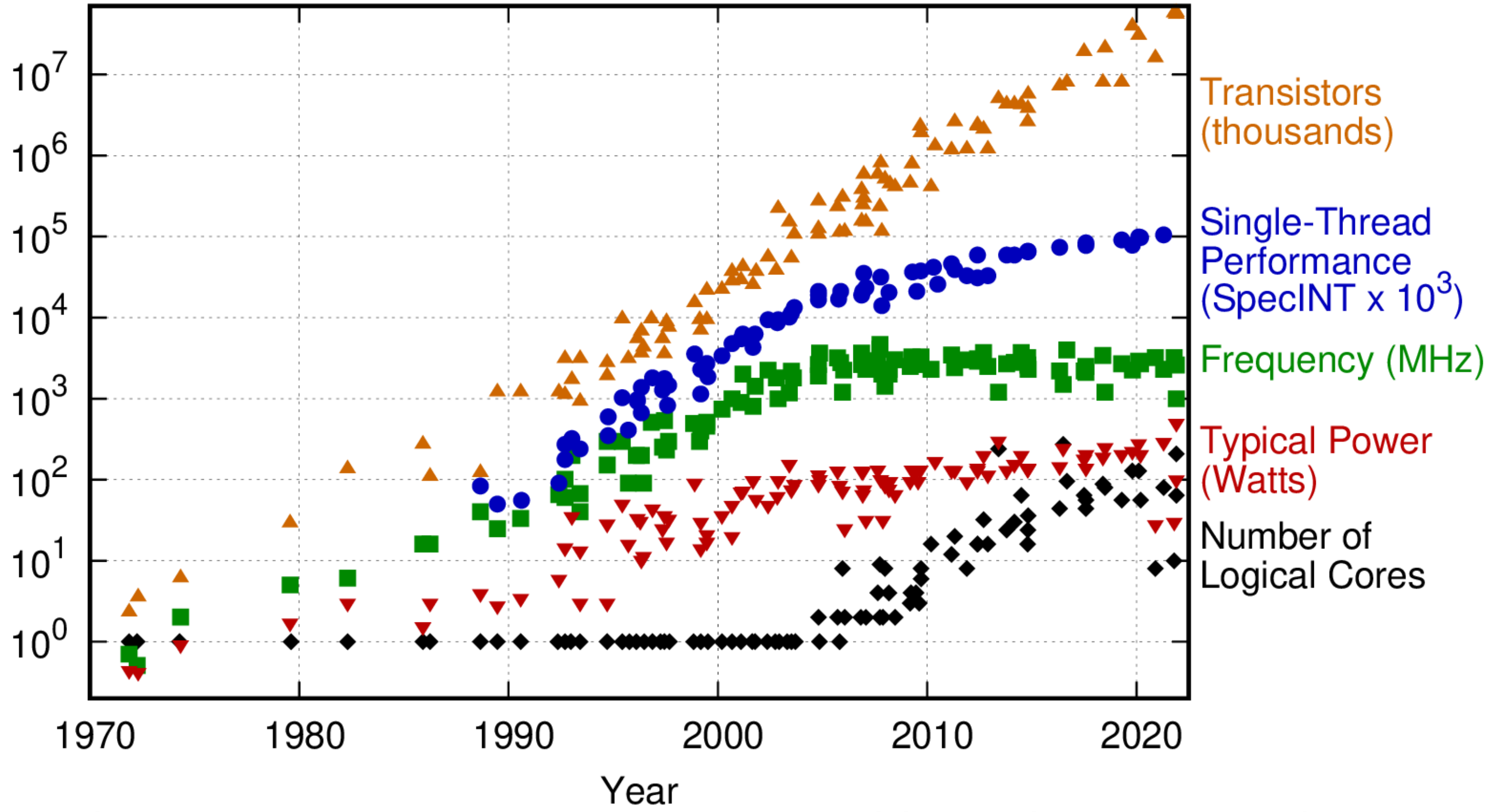




JSSC Oct **1974**, page 256

- In this ideal scaling, as L → α*L
  - V_{DD} → α*V_{DD}, C → α*C, I → α*I
  - Delay = CV_{DD}/I scales by α, so f → 1/α
  - Power for each transistor is CV^2*f and scales by α^2
    - keeping total power constant for same chip area

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

Post-Dennard Pivoting

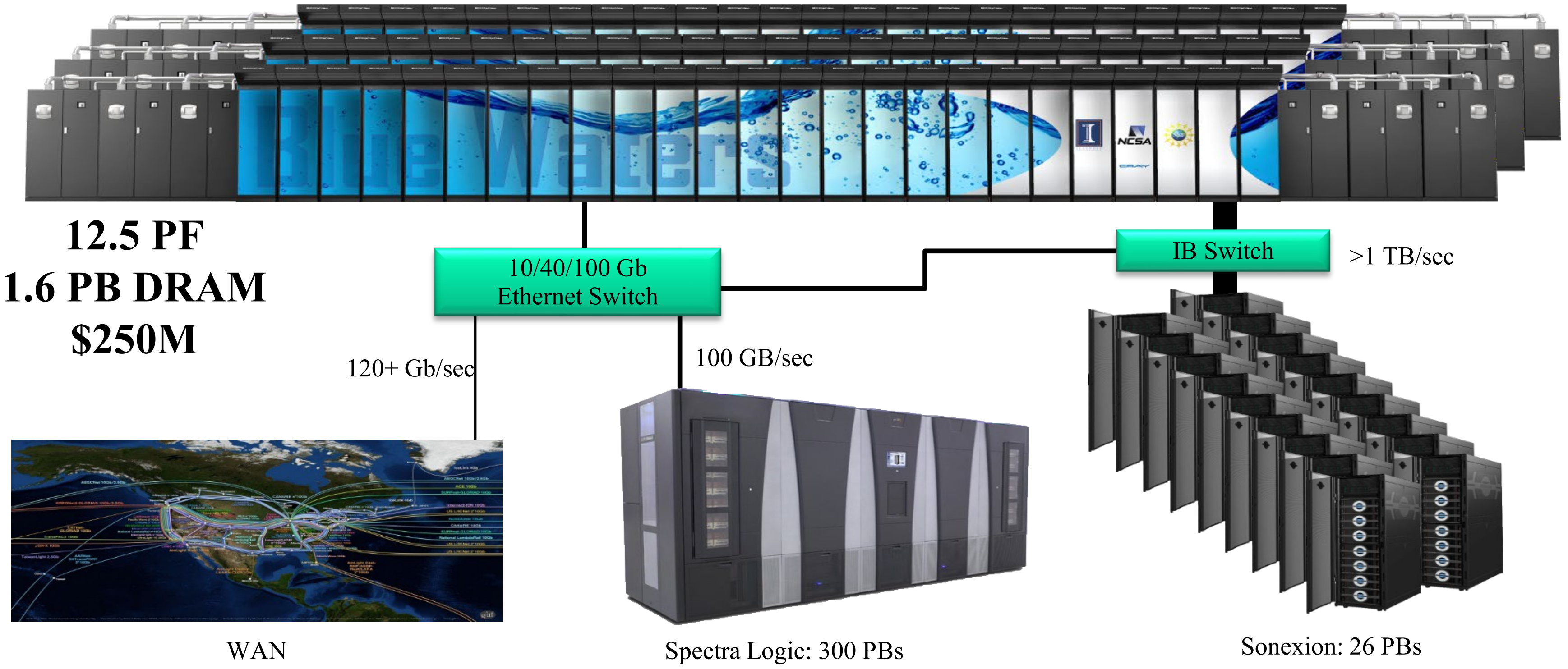
- Multiple cores with more moderate clock frequencies
- Heavy use of vector execution
- Employ both latency-oriented and throughput-oriented cores
- 3D packaging for more memory bandwidth

Blue Waters Computing System

Operational at Illinois since 3/2013

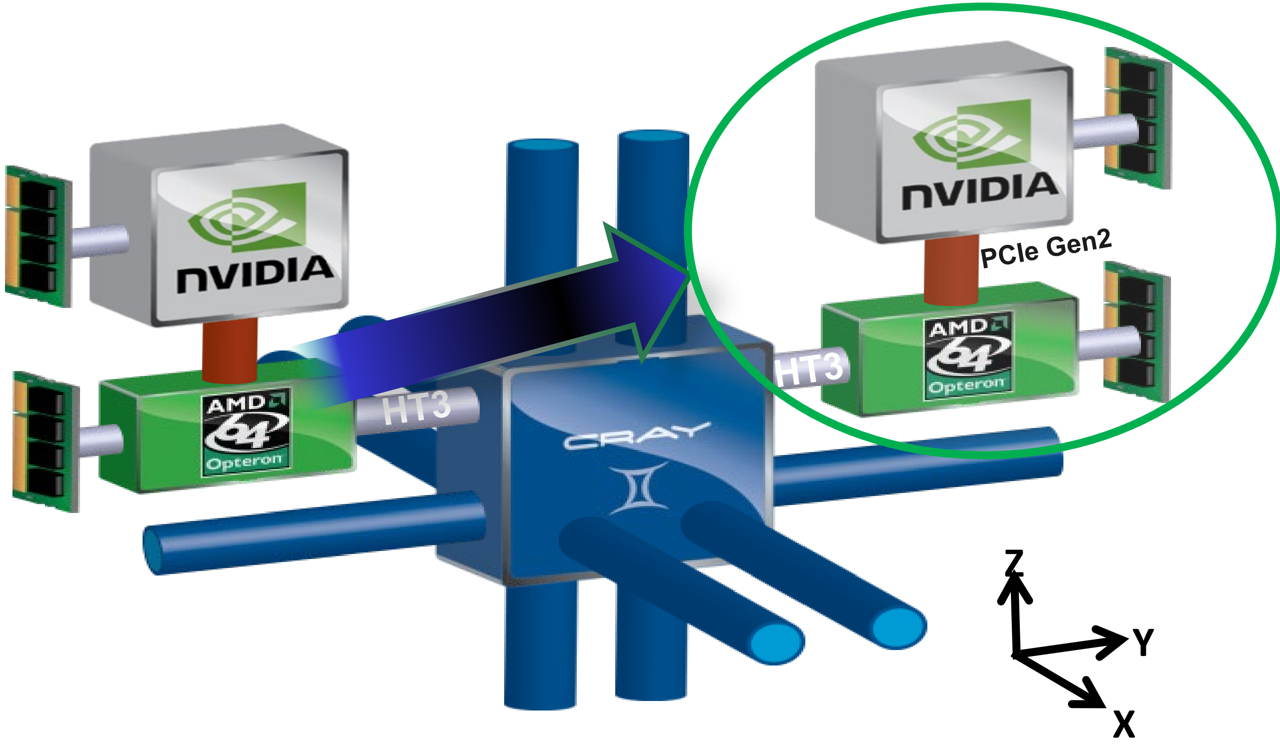
49,504 CPUs -- 4,224 GPUs

12.5 PF
1.6 PB DRAM
$250M

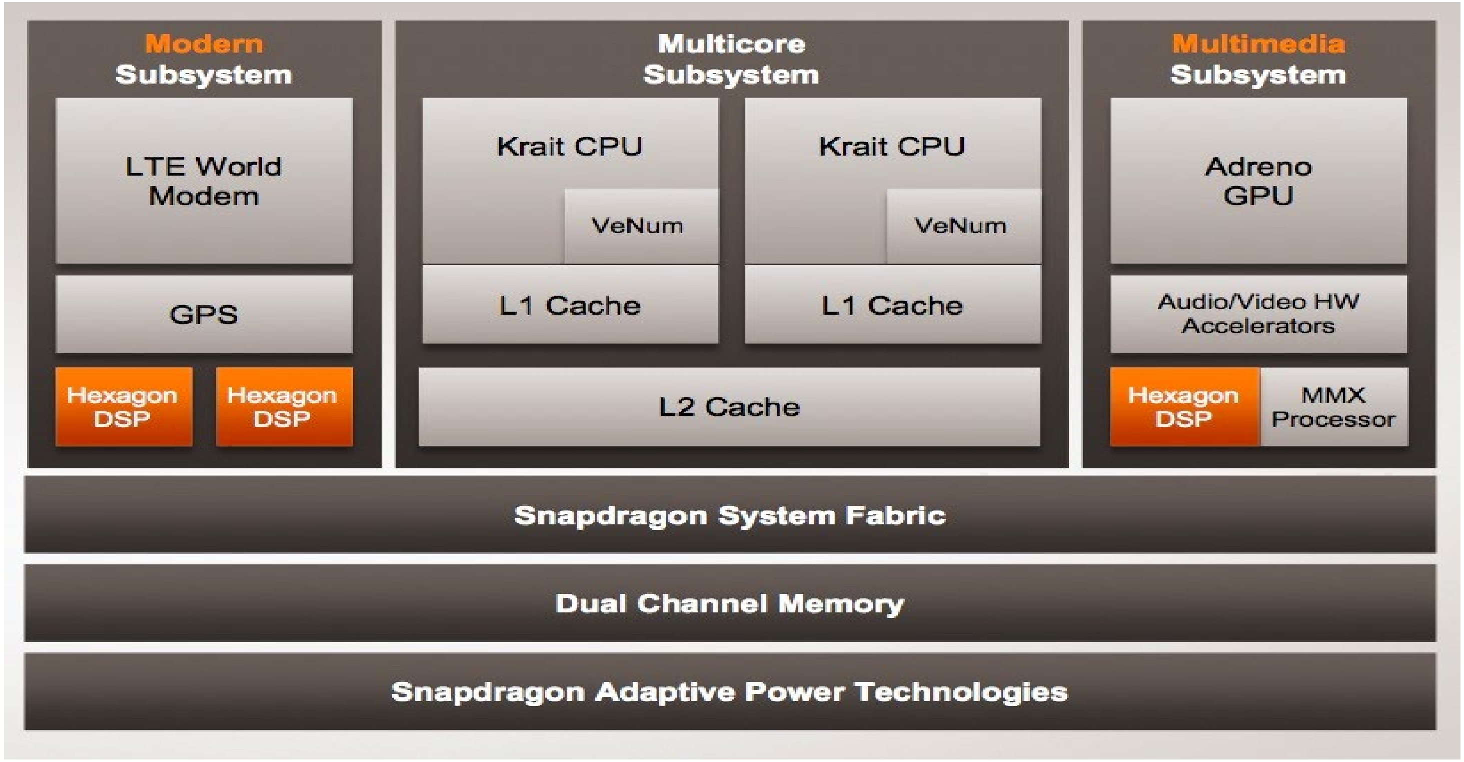


Cray XK7 Compute Node

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

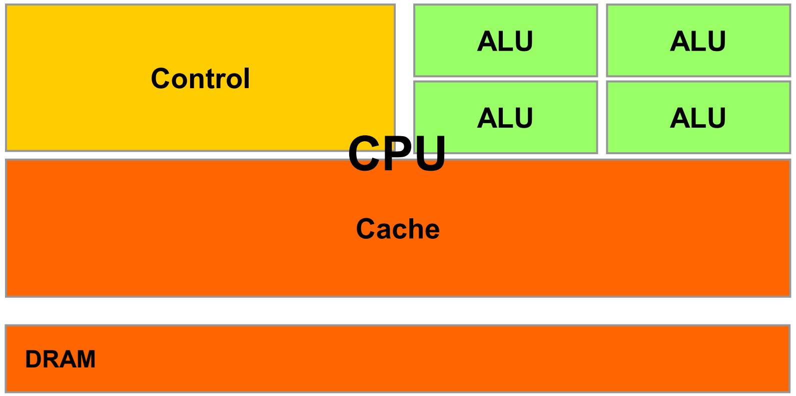


Qualcomm SoC for Mobile



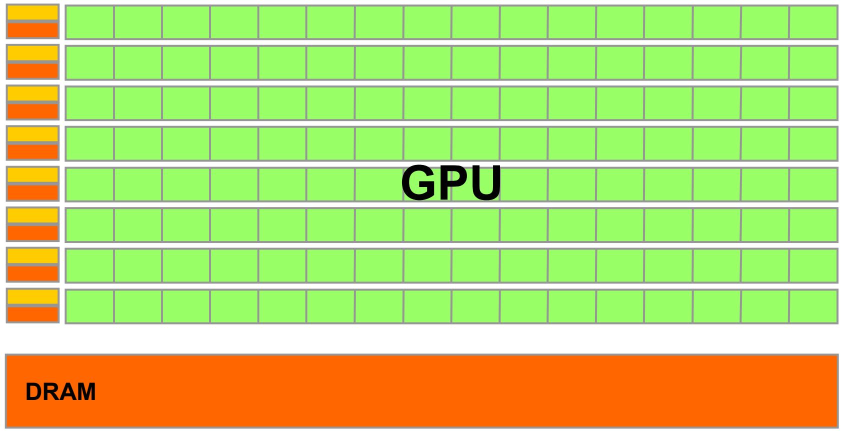
CPUs: Latency Oriented Design

- 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



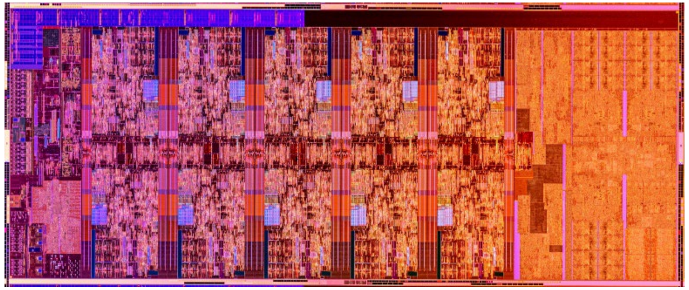
GPUs: Throughput Oriented Design

- 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

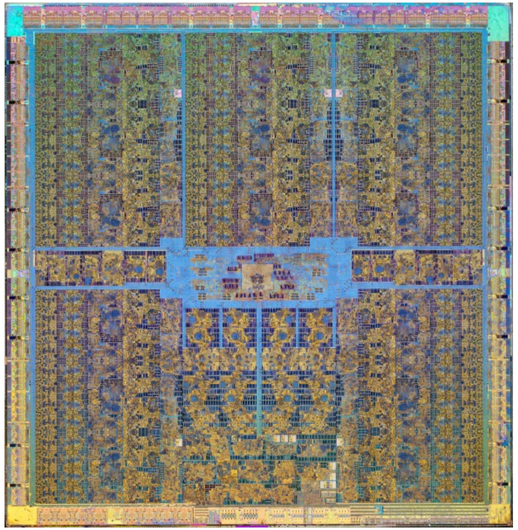


CPU vs GPU

- 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

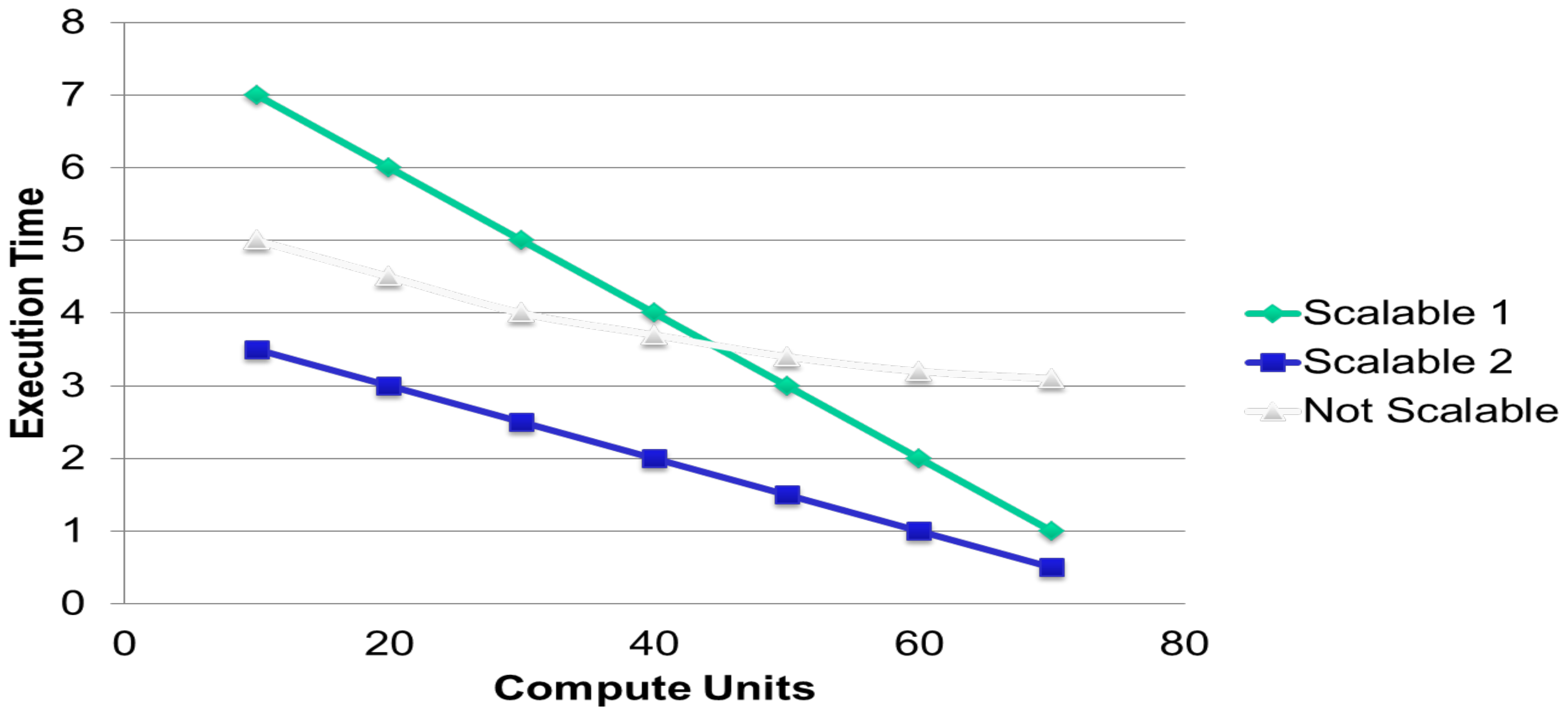
Interactive Physics

Numerical Methods

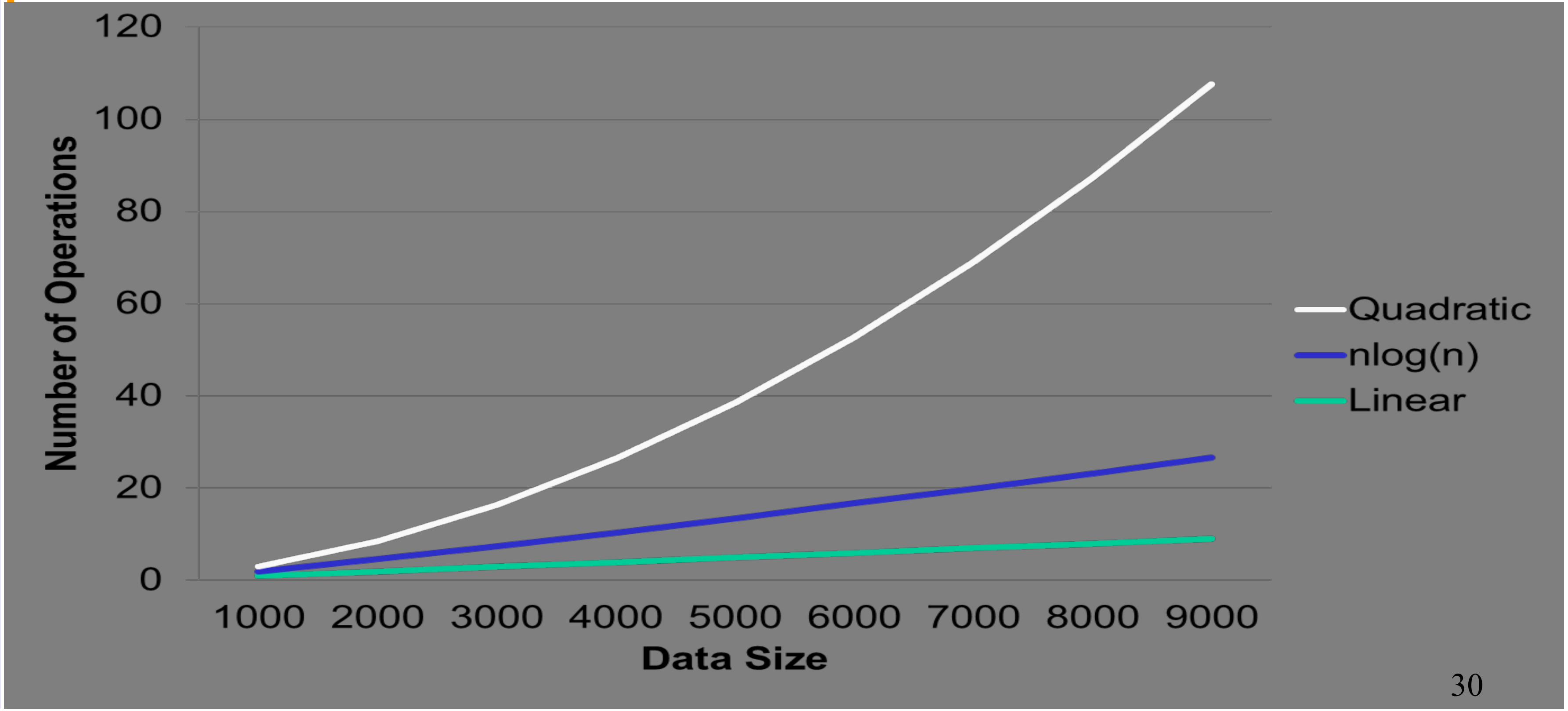
Parallel Programming Workflow

- 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**

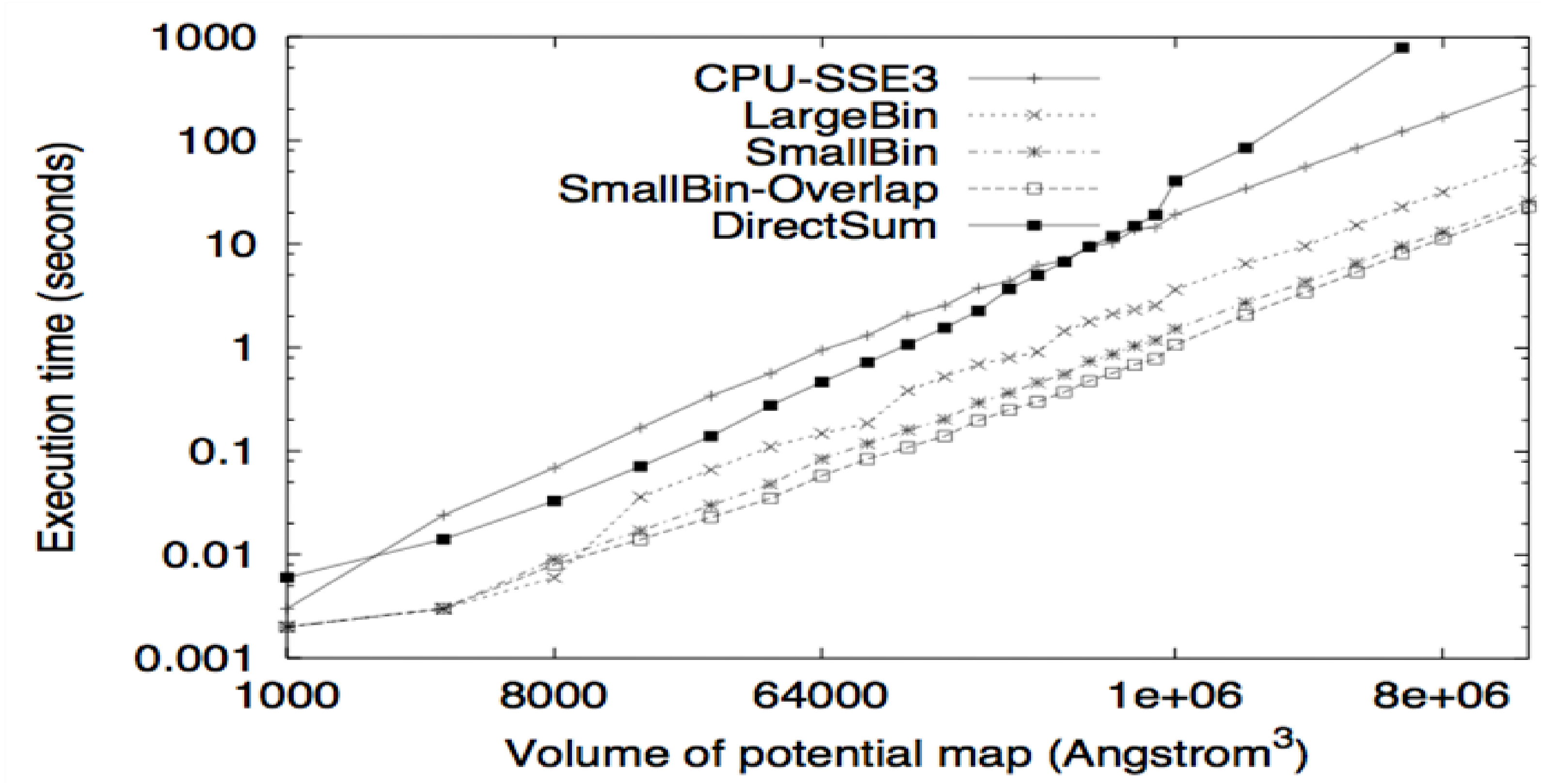
Parallelism Scalability



Algorithm Complexity and Data Scalability



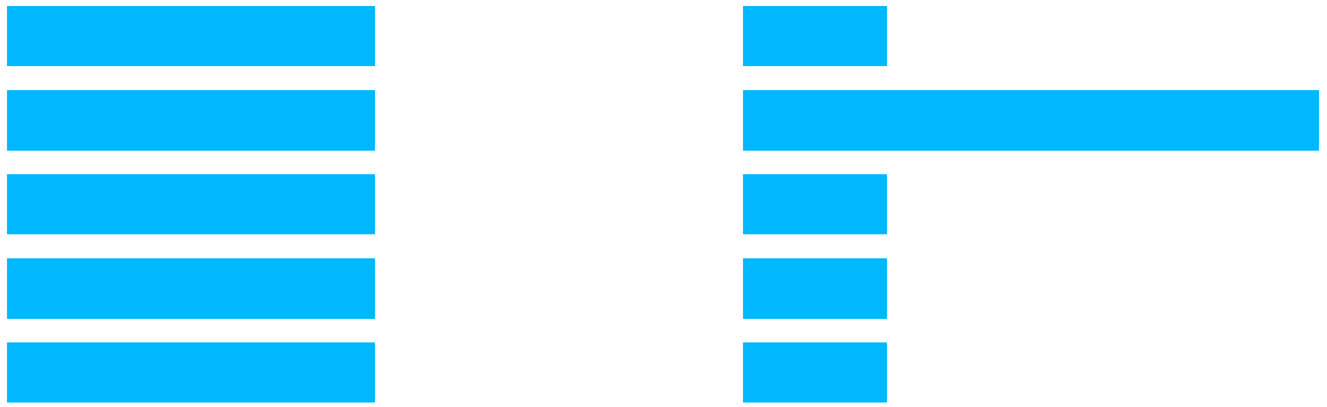
A Real Example of Data Scalability Particle-Mesh Algorithms



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



good

bad!

Global Memory Bandwidth

Ideal

Reality

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?