lecture 2

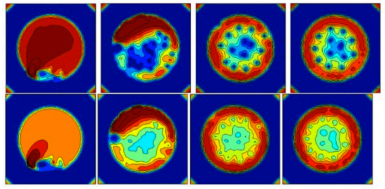
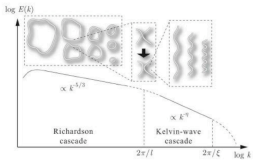
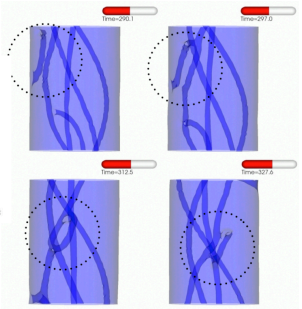
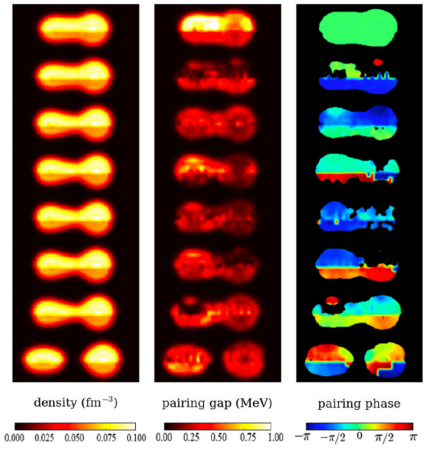
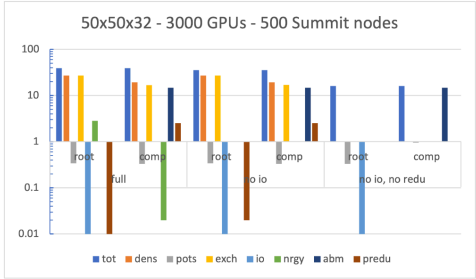


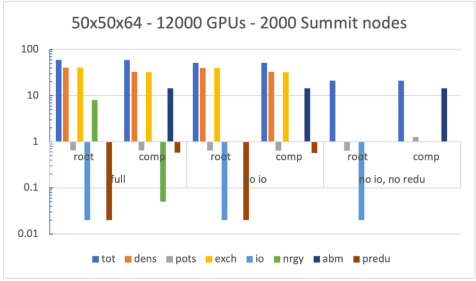
Figure 2: The magnitude of the pairing field (Δ, top row) and the corresponding density (n), bottom row) for a UFG system composed of 1800 particles in a 48^3 lattice stirred at supercritical velocity $1.216v_c$. Here thirteen vortices are formed once the stirring concludes.



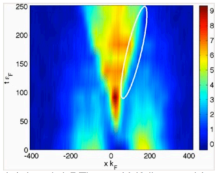






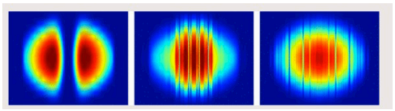


Quantum Shock Waves / Domain Walls w/in TDSLDA

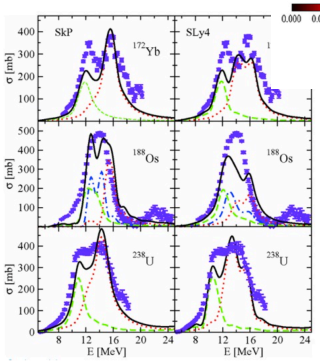


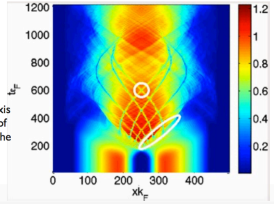
J. A. Joseph, J. E. Thomas, M. Kulkarni, and A. G. Abanov, Phys. Rev. Lett. 106, 150401 (2011).

Phys Rev Lett. 108, 150401 (2012) , A. Bulgac, Y.-L. Luo, and K.J. Roche

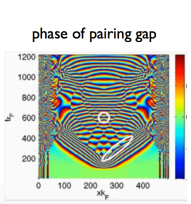


Magnitude of pairing field at t ef ~ 30, 350, 690. DWs appear as planar number density depletions with a width comparable to the diameter of a quantum vortex

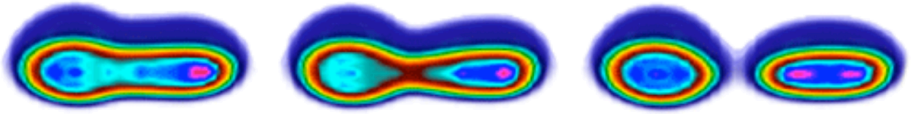


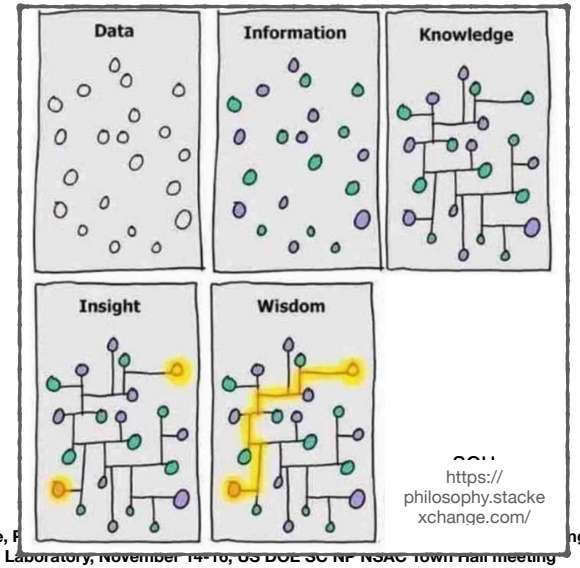


e space-time evolution of the number nsity profile $n(x,0,t)$ along the collision axis the TDSLDA simulation of the collision of o UFG clouds. Here kf and .5 kf ^2 are the tial values of the Fermi wave vector and ergy at the center of clouds.The region ere one elastic collision of two DWs curs is outlined with a white circle.



phase of pairing gap



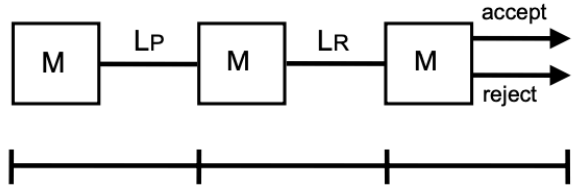


Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

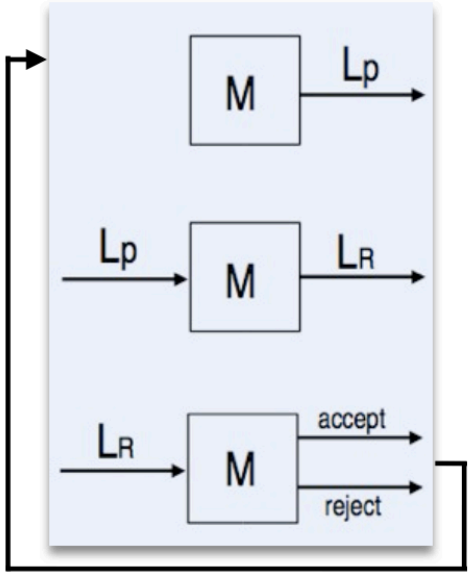
Kenneth J. Roche, P ngton, Department of Physics
Argonne National Laboratory, November 14-16, 03 DOE SC NP NSAC Town Hall meeting

Concepts: Solving Problems with Computers

- COMPLEXITY
  - PROBLEMS
  - ALGORITHMS
  - MACHINES



Measured time for machine M to generate the language of the problem plus time to generate the language of the result plus the time to accept or reject the language of the result.



Asking questions, solving problems is recursive process

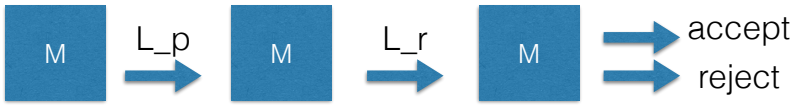
Accepting a result means a related set of conditions is satisfied

$$S = S1 \wedge S2 \wedge \ldots \wedge Sn$$

algorithm, a Turing machine that always halts
decidable problems are posed as a recursive language
undecidable problems have no algorithms that accept the language of the problem and generate / accept or reject an answer (Rice's Theorem posits that non-trivial properties of r.e. languages are undecidable. Examples are emptiness, finiteness, regularity, and context freedom.)

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Context: Classical comput(ers)ing



Programming implies control of machine state evolution

- machine can exist in finite, possibly very large, number of states
- states have representation in basis (instructions -> gates)
- transitions between states are well defined by transition function
- executable requires finite resources

software developer's challenge?

- Moore's Law persists
- sidestepped by massive increase in concurrency
- introduces challenges in all components of computing
- parallelism and concurrency are very poorly utilized in general
- programming model connected to machine design explicitly - no free lunch
  - distributed memory - message passing / remote memory operations
  - shared memory - thread control
  - hybrid (target combined CPU + GPU)
  - abstractions - PGAS (ie provide virtual global address space composed of aggregated resources); implementor pays the price
- implementation efficiency is dismal -nature is way more efficient

on supercomputers performance limited by …

1) **System power** -primary constraint (PUI, facility / total)
2) **Memory** bandwidth and capacity are not keeping pace
3) **Concurrency** 1000X increase in-node
4) **Processor** open question
5) **Programming model** compilers will not hide this
6) **Algorithms** need to minimize data movement, not flops
7) **I/O bandwidth** not on pace with machine speed
8) **Reliability and resiliency**
9) **Bisection bandwidth** limited by cost and energy

Machine construct

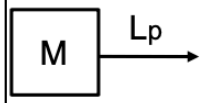
- storage and processing accomplished by switches called transistors

- calculate by using circuits composed of logic gates
  - made from a number of transistors connected together
  - operate predefined action on patterns of bits stored in temporary memories called registers
  - output is new patterns of bits

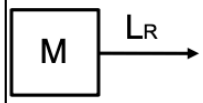
- algorithm that performs a particular calculation takes the form of an electric circuit made from a number of logic gates, with the output from one gate feeding in as the input to the next

Extended Scope of Application Software

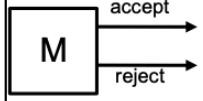
Example Problem: solving algebraically determined systems of linear equations numerically (Linpack TOP500, **FLOPs**)



• construct A, b given dim(A) dim(A)=dim(rank(1,A))=dim(rank(2,A))



• compute A=LU, solve Ly=b for y, solve Ux=y for x



• compute $|A|_{inf}$, compute $|x|_2$, compute $|b-Ax|_2$,
approximate ε (machine precision),
form $\kappa = |b-Ax|_2 / (|A|_{inf} * |x|_2 * \varepsilon)$;
if κ ~ O(dim(A)) **accept** x, else **reject** x

Ex2: BFS(Graph500,**TEPS**)

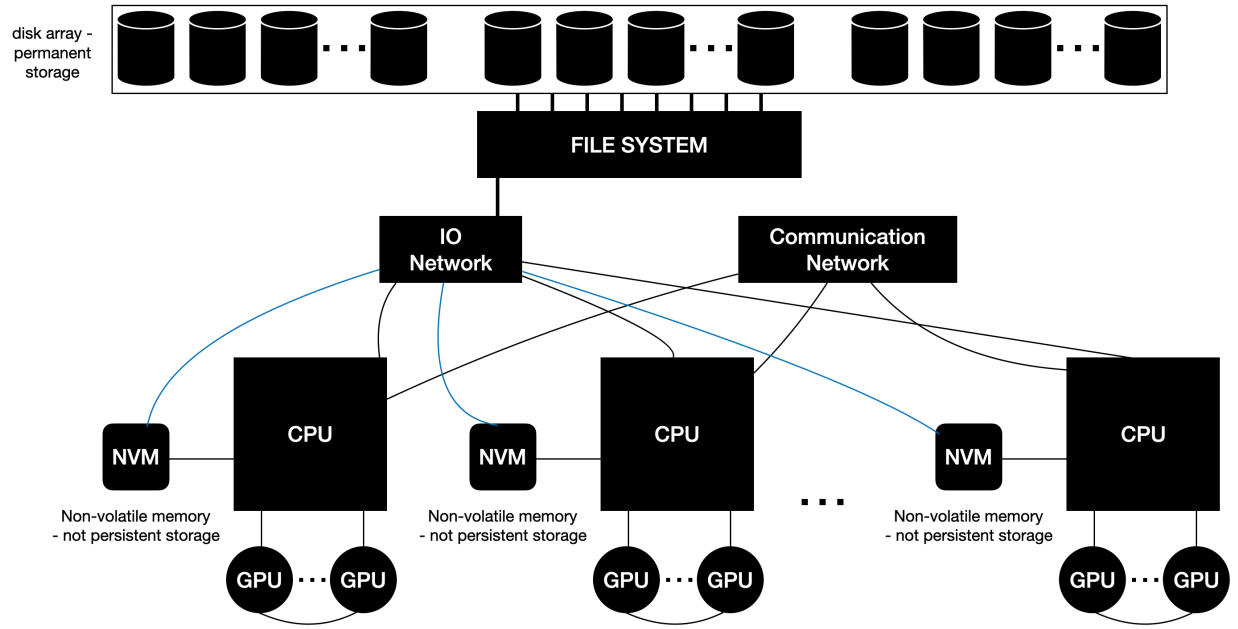
```
Input: Graph G(V,E), source node s0
Output: Distances from s0 Dist[1..|V|]
1   ∀v ∈ V do:
2       dist[v] = ∞
3   dist[s0] := 0
4   Q := ∅
5   Q.enqueue(s0)
6   while Q ≠ ∅ do
7       i := queue.dequeue()
8       for each neighbor v of i do
9           if dist[v] = ∞ then
10              dist[v] := dist[i] + 1
11              Q.enqueue(v)
12          endif
13      endfor
14  endwhile
```

Q: How do the language of the problem and the accepted result relate to reality?
Requires analysis beyond software analysis above and distinguishes ***computational science*** from system and library software development. Takes more time -needs refinement phase of **algorithms and metrics**.

Metric: the distance between two points in some topological space

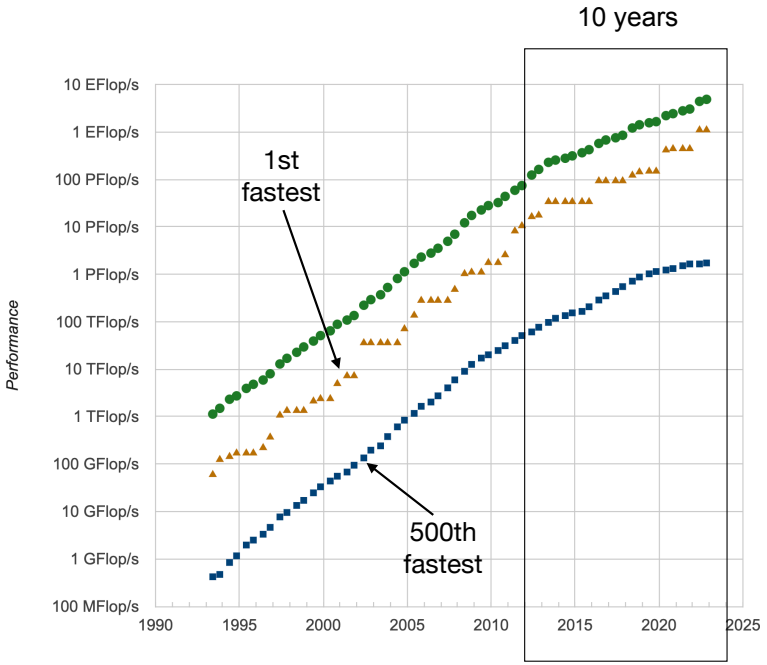
After the HW is decided , it becomes software manipulating software …
but the systems are becoming extremely complex and there are many details to be understood to achieve scalability, utilization, and efficiency

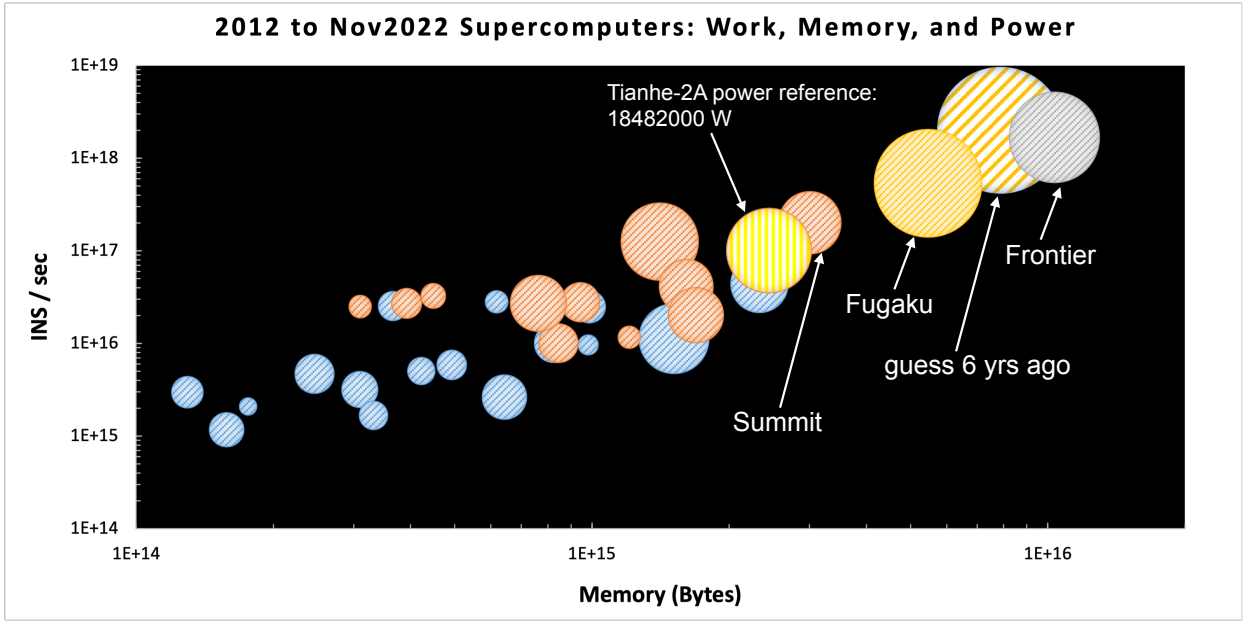
A cartoon HPC system



Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Quick look at some features of the US DOE's (and the world's) fastest supercomputers





**NB: only the top 5 machines broke the 100PF layer, 500th fastest is < 2PF as of Nov2022
**flops to byte ratio looks difficult to achieve for most applications … let's explore this further

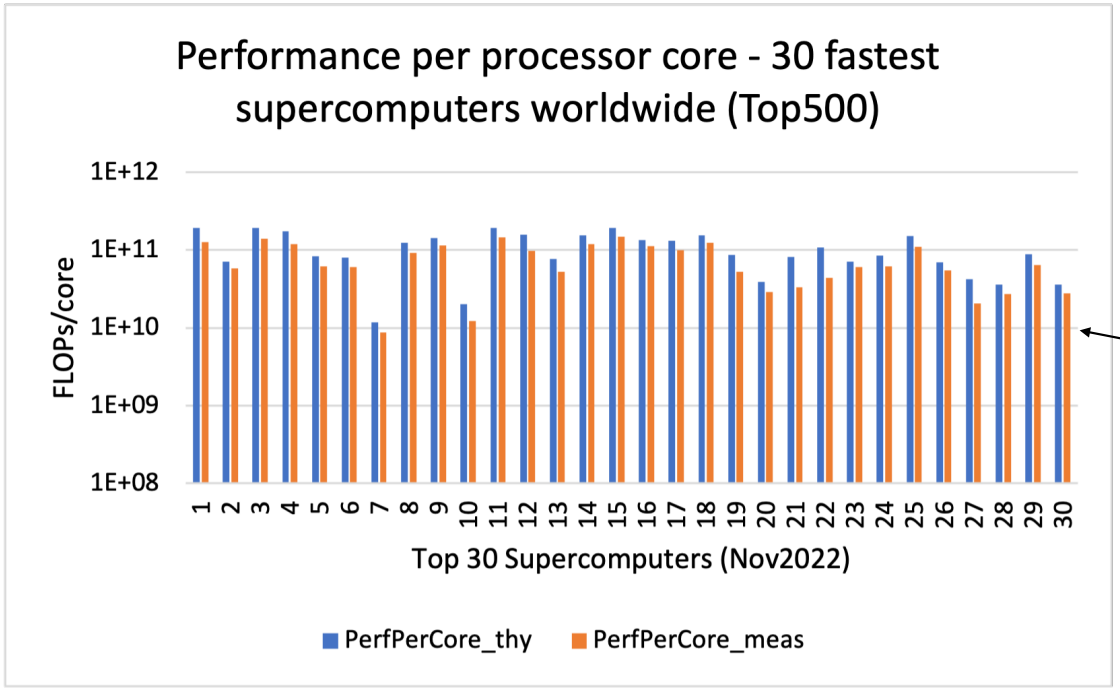
Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

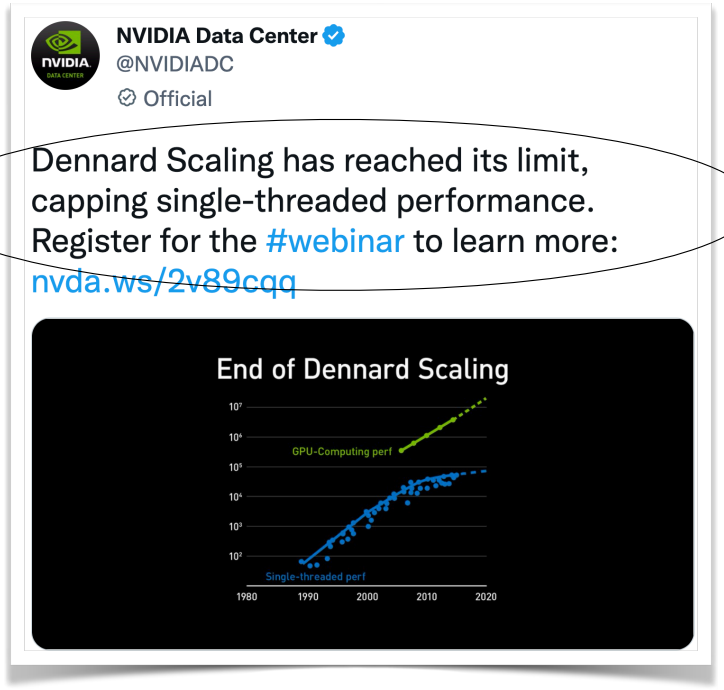
“Power Wall” has constrained practical processor frequency to around 4 GHz since 2006

Dennard scaling: one can continue to decrease the transistor feature size and voltage while keeping the power density constant

Power = a * CFV^2
- a – percent time switched
- C = capacitance
- F = frequency
- V = voltage

“leakage current” and “threshold voltage” cause practical power per transistor limit
consequence: power density increases for smaller transistors because these don’t scale with size



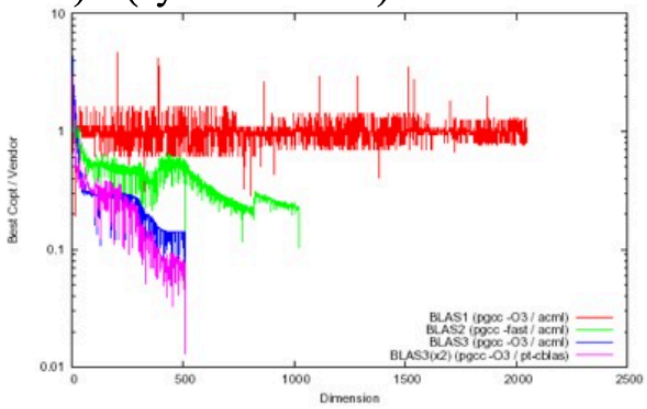


Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Memory Wall *Always* There ...

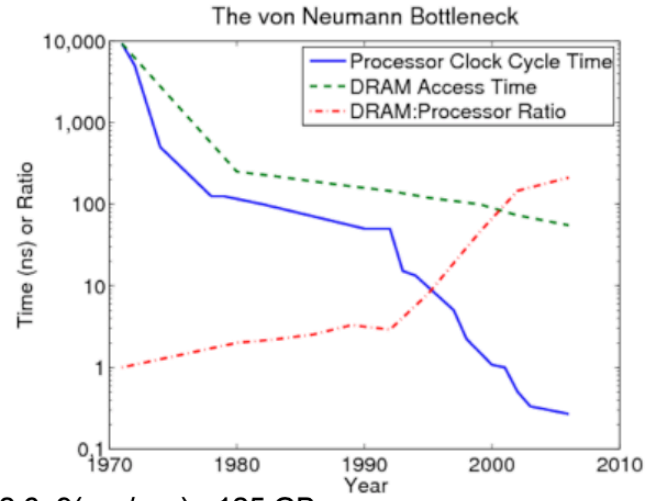
Computation: Theoretical peak: (# cpu cores) * (flops / cycle / core) * (cycles / second)

Memory: Theoretical peak: (bus width) * (bus speed)



BLAS 1:	O(n)	operations on O(n) operands
BLAS 2:	O(n**2)	operations on O(n**2) operands
BLAS 3:	O(n**3)	operations on O(n**2) operands

Aside …



$y = \alpha x + y$:

3 loads, 1 store

(more expensive than FP_OPs by a long shot)

2 floating point operations (maybe 1) on 3 operands

eg, double precision

(3 operands / 2 flop) * (8 bytes / operand) * 6 core * 4 (flop / cyc /core) * 2.6e9(cyc/sec) ~125 GBps

... We don't have this and to get it is $$$ ... how to achieve **Sustainability??**

Kenneth J. Roche
University of Washington

Let’s backup and approach this again …
Conceptualize CPU and Memory

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

CPU

- **ALU**, adds, comparisons
- **FPU**, floating point operations
- **L/S U**, data, ins loads / stores
- **Registers**, fast memory; FPR, GPR, etc.
- **PC**, program counter -address in memory of instruction that is executing (control flow, fetch / decode in CPU)
- **Memory interface**, often L1 and L2 caches

other:

clock speed
buses
ISA (Intel x86 most popular, x86-64, ...)

Consider a simple instruction like **ADD R1, R2, R3**, which adds the values in registers R2 and R3 and stores the result in R1.

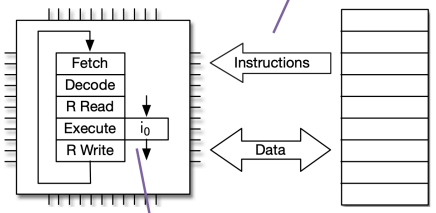
Step	Operation
Fetch	Load instruction from memory into the Instruction Register (IR).
Decode	Identify the operation (ADD) and operands (R1, R2, R3).
Execute	Compute `R1 ← R2 + R3` in the ALU and store the result in R1.

This cycle repeats for every instruction in the program.

PC technically holds address of next instruction
Instruction is fetched from memory (RAM) and stored in the Instruction Register (IR)
PC is incremented to point to the next instruction

Simplest model

CPU fetches and executes instructions



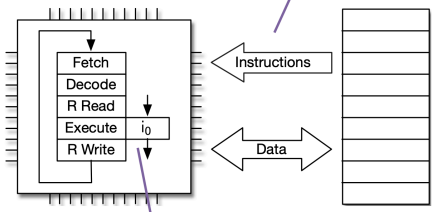
Many cycles per instruction

Technology	Paradigm	Hammer
CPU (single core)	Sequential	C compiler
SIMD/Vector (single core)	Data parallel	Intrinsics
Multicore	Threads	pthreads library
NUMA shared memory	Threads	pthreads library
GPU	GPU	CUDA
Clusters	Message passing	MPI

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Simplest model

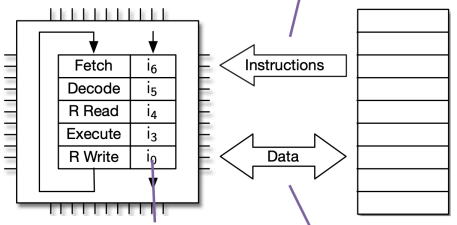
CPU fetches and executes instructions



Many cycles per instruction

Pipelining

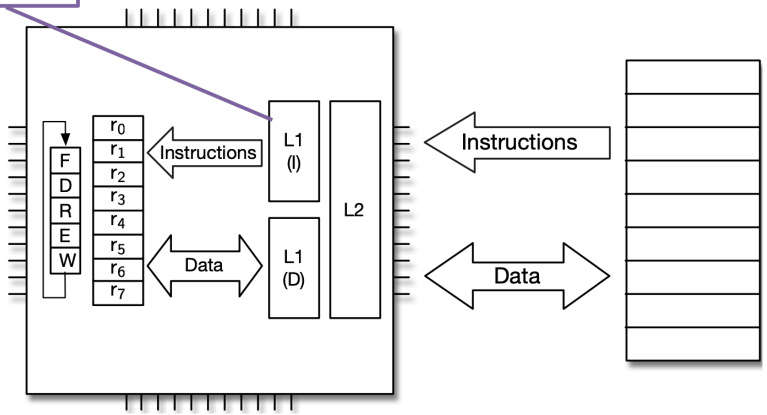
Instructions are fetched in a stream



Processed in a pipeline

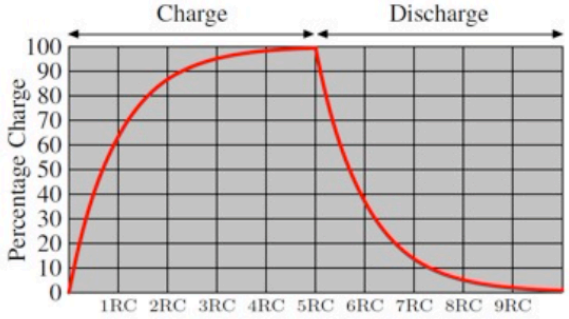
A long trip from memory

Use special, fast memory to keep data and instructions close



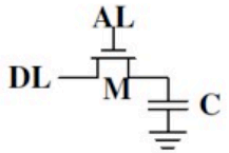
Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

DRAM COSTS: Power = Capacity * Voltage^2 * Frequency



DRAM
C, capacitor, keeps cell state

M, transistor, controls access to cell state



read the state of the cell the **access line AL** is raised
-causes a current to flow on the data line DL or not

write to the cell the **data line DL** is appropriately set and AL is raised for a time long enough to charge or drain the capacitor

Today's Memories ...

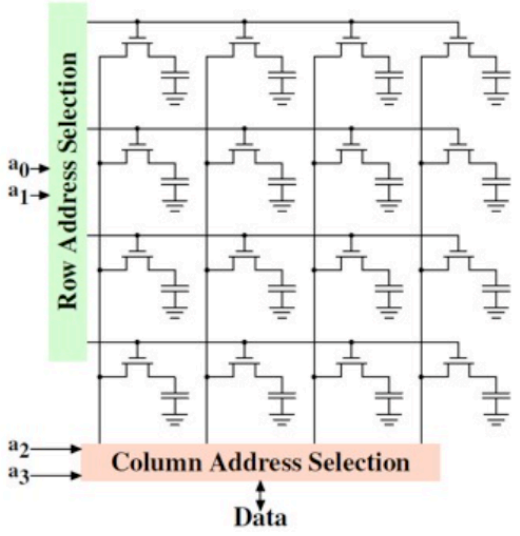
- 10^9 cells
- cell capacitance < femto-farad
- resistance O(tera-ohms)

Refresh Cycles ~ 64ms

- leakage
- reading drains the charge (read + recharge)
- stall cycle on bus > 11 cpu cycles

Faster memory

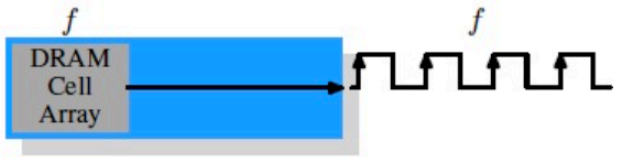
- lower voltage --> decreases stability,
- increase frequency --> $$$ as arrays get large
- (i.e. more addressable memory) and voltage is increased to assure stability



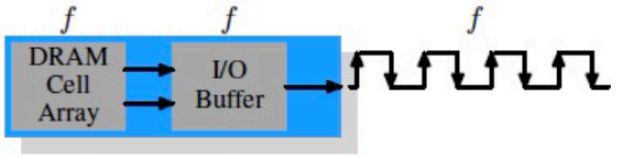
ref. Drepper, What every Programmer Should Know about Memory

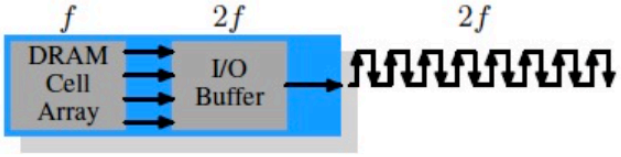
Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

SDR (PC100) ~
DRAM cell array 100MHz
data transfer rate 100Mbps



DDR (PC1600) ~ moves 2X the data / clock (leading , falling)
add "I/O" buffer (2 bits on data line) adjacent to DRAM cell array
pull two adjacent column cells per access over 2 line data bus
100 MHz X 64 bit / data bus X 2 data bus lines = 1600 MBps





DDR2 (PC6400) ~ moves 4X the data / clock
double the bus frequency --> 2X bandwidth
double "I/O" buffer speed to match the bus
4 bits / clock on 4 line data bus
200MHz array; 400MHz bus; 800MHz FSB (effective freq)
200 MHz X 64 bit / data bus X 4 data bus lines = 6400 MBps
240 PIN addressing @ 1.8V

***each stall cycle on the memory bus is > 11 cpu cycles even in the best systems**

Memory Issues

CPU waiting for memory hierarchy is bottom line of idle time

- memory latency
- miss rates
  - instruction stalls
  - branch misprediction
  - unresolved data dependencies

O/S stall times are substantial cost -not easily influenced

- misses
- coping with interference from the application
  - write references: how big should the write buffer be + queuing model

Memory Measurements

- first touches are expensive
  - misses lead to repeated first touches
  - repeated dynamic allocation / free lead to first touches
- costs can be measured

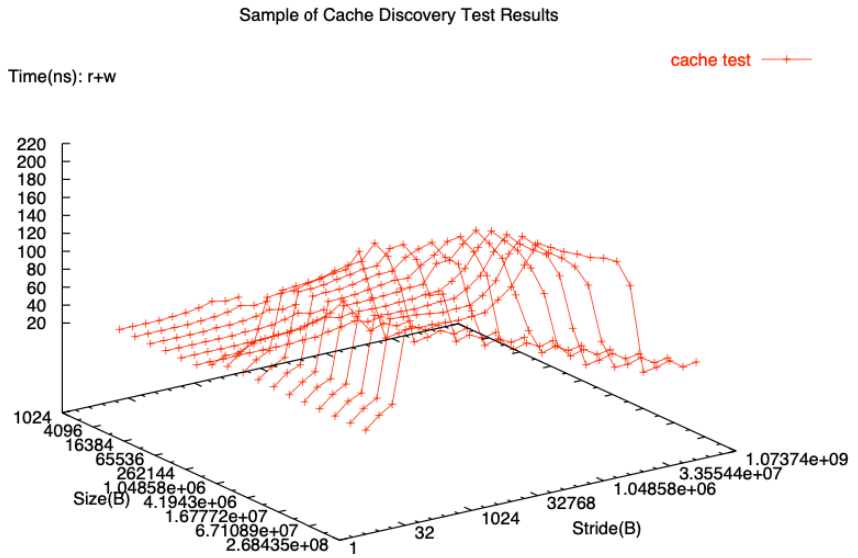
accesses / second (access rate)
x
N_{fractional miss ratio} / access
x
bytes / miss
:= bytes / second

- but, to be accurate requires average memory access times over the duration of program execution
- a program's locality behavior is not constant during execution and is basically unique

Hierarchical caches to hide memory latencies

temporal locality
when a referenced resource is referenced again sometime in the near future

spatial locality
the chance of referencing a resource is higher if a resource near it was just referenced



Cache Coherency:

write-through, if cache line is written to, the processor also writes to main memory (at all times cache and memory are synchronized)

write-back, cache line is marked dirty, write back is delayed to when cache line is being evicted

> >1 processor core is active (say in SMP) -all processors still have to see the same memory content; have to exchange CL when needed -includes the MC

write-combining (ie on graphics cards)

Use of Cache Inspired Basic Optimizations

base / node focus

- **non-temporal writes,** ie don't cache the data writes since it won't be used again soon (i.e. n-tuple initialization)
  - avoids reading cache line before write, avoids wasteful occupation of cache line and time for write (memset()); does not evict useful data
  - sfence() compiler set barriers
- **loop unrolling** , transposing matrices
- **vectorization**, 2,4,8 elements computed at the same time (SIMD) w/ multi-media extensions to ISA
- **reordering** elements so that elements that are used together are stored together -pack CL gaps w/ usable data (i.e. try to access structure elements in the order they are defined in the structure)
- **stack alignment,** as the compiler generates code it actively aligns the stack inserting gaps where needed ... is not necessarily optimal -if statically defined arrays, there are tools that can improve the alignment; separating n-tuples may increase code complexity but improve performance
- **function inlining**, may enable compiler or hand -tuned instruction pipeline optimization (ie dead code elimination or value range propagation) ; especially true if a function is called only once
- **prefetching**, hardware, tries to predict cache misses -with 4K page sizes this is a hard problem and costly penalty if not well predicted; software (void _mm_prefetch(void *p, enum _mm_hint h) --_MM_HINT_NTA -when data is evicted from L1d -don't write it to higher levels)

Loop fusion transforms multiple distinct loops into a single loop. It increases the granule size of parallel loops and exposes opportunities to reuse variables from local storage. Its dual, loop distribution, separates independent statements in a loop nest into multiple loops with the same headers.

```
PARALLEL DO I = 1, N
    A(I) = 0.0
END
                          ⟹
                        fusion
PARALLEL DO I = 1, N
    B(I) = A(I)           ⟸
END                     distribution
```

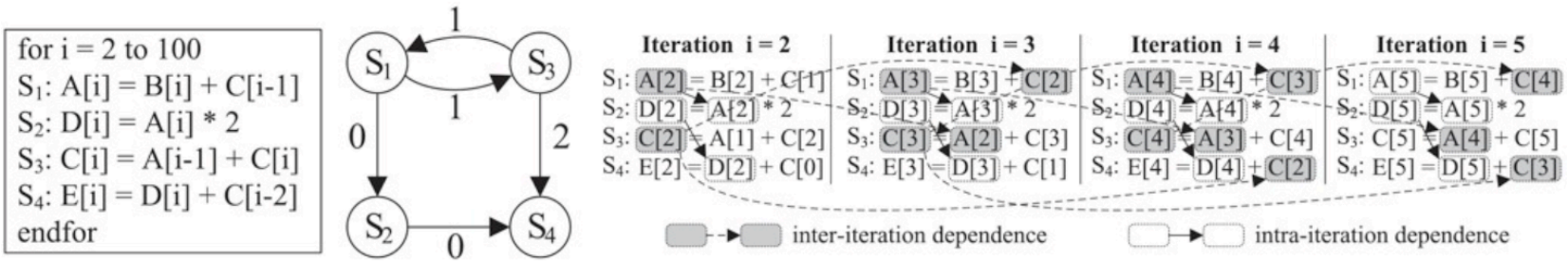
```
PARALLEL DO I = 1, N
    A(I) = 0.0
    B(I) = A(I)
END
```

In the example above, the fused version on the right experiences half the loop overhead and synchronization cost as the original version on the left. If all A(1:N) references do not fit in cache at once, the fused version at least provides reuse in cache. Because the accesses to A(I) now occur on the same loop iteration rather than N iterations apart, they could also be reused in a register. For sequential ex-

source: K. Kennedy, *Rice*

- reduce synchronization overheads in parallel loops
- improve data locality

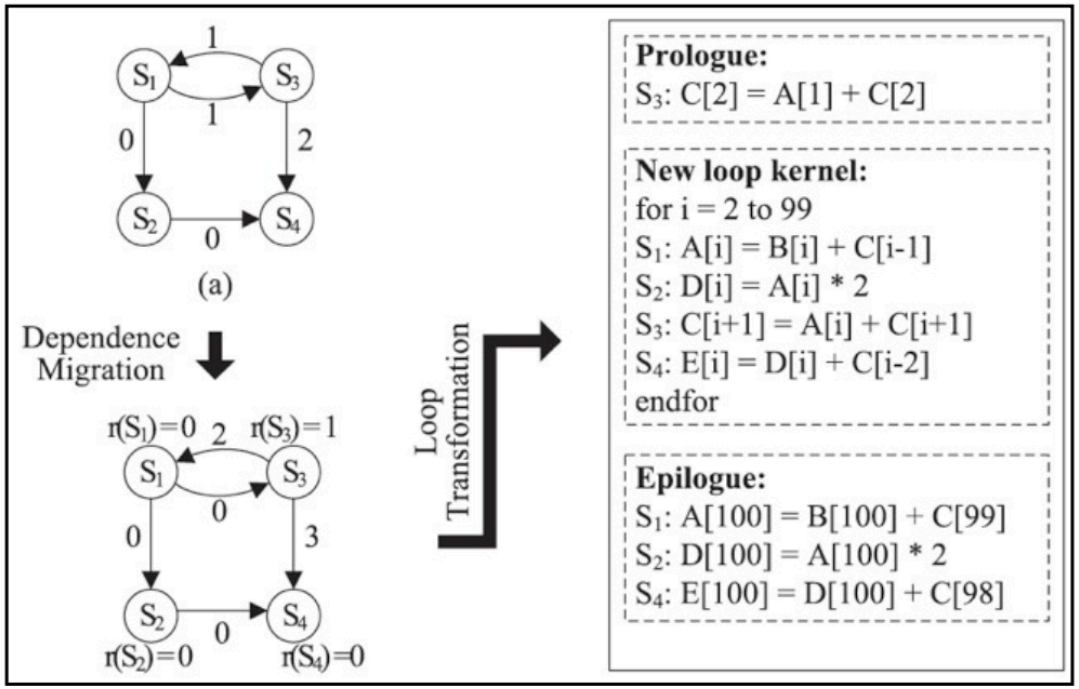
Going Beyond Instruction Level //ism to Loop Level



before, minimum nonzero edge weight = 1

after, minimum nonzero edge weight = 2

(re)moving dependencies decreases stalls, decreases time



Kenneth J. Roche
University of Washington

AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Optimally Maximizing Iteration-Level Loop Parallelism, D. Liu et al., IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, NO. 3, MARCH 2012

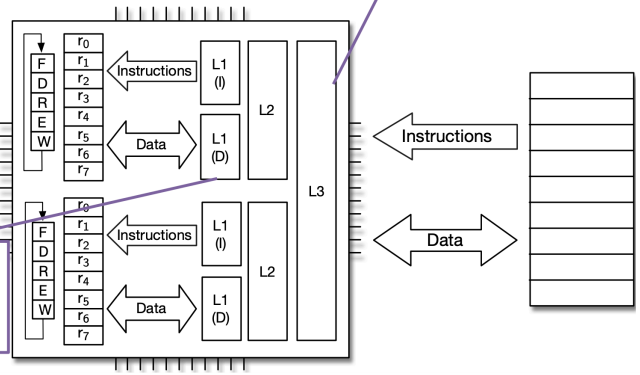
Let’s backup and approach this again … again …
Concurrency

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Multicore CPUs

Replicate 2X

Cores share slower memory



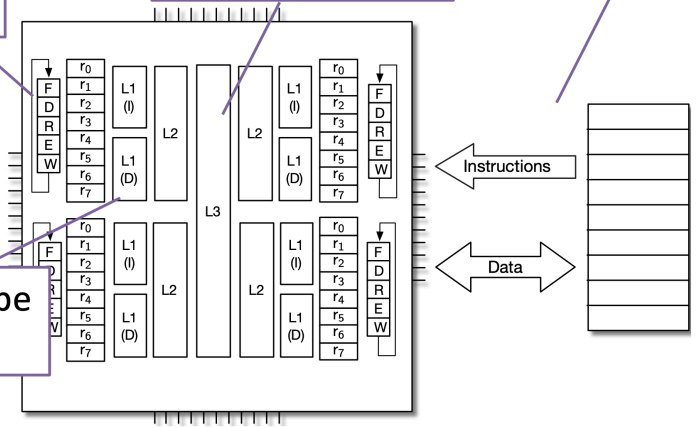
Caches need to be kept coherent

Even more cores

Replicate 4X

Cores share slower memory

Include super-slow DRAM



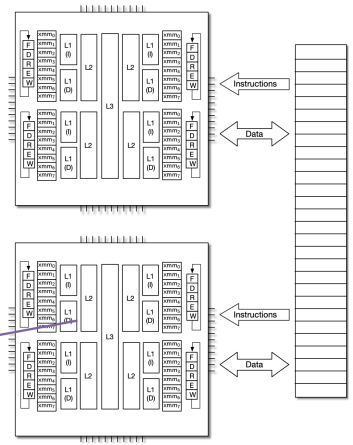
Caches need to be kept coherent

Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

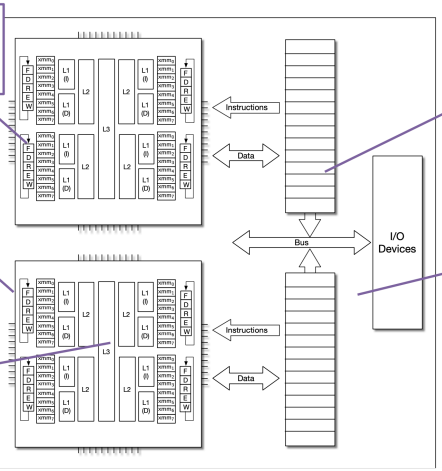
Uniform memory access (UMA)

Asymmetric

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent: CC-NUMA



Memory may be non-uniformly shared among sockets

Non-uniform memory access (NUMA – most common)

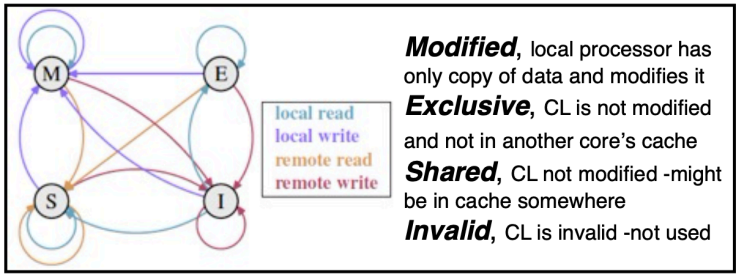
Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Use of threads means coping with complicated issues

Concurrency: sharing resources like memory and the network

threaded

- cache contention, coherency
- atomicity
- memory bandwidth
- scheduling, pinning to hardware

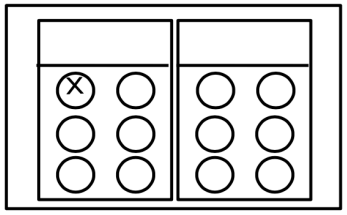


Modified, local processor has only copy of data and modifies it
Exclusive, CL is not modified and not in another core's cache
Shared, CL not modified -might be in cache somewhere
Invalid, CL is invalid -not used

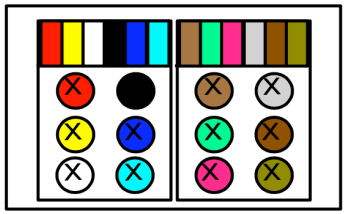
*other processor's activities are snooped on the address bus

fork (create) / join overheads

NT	Cycles	L2DCM
1	1959379	69
2	2020818	81
4	2289393	122
6	2366367	146
8	2499159	239

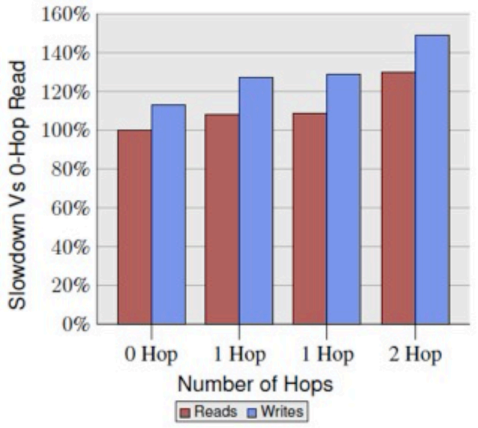


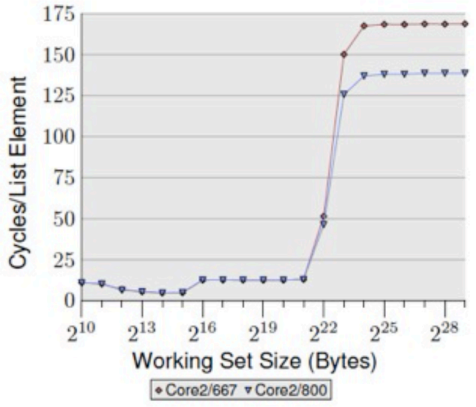
NUMA + memory affinity



no NUMA, 6 PEs/socket

make the FSB faster with increasing core count

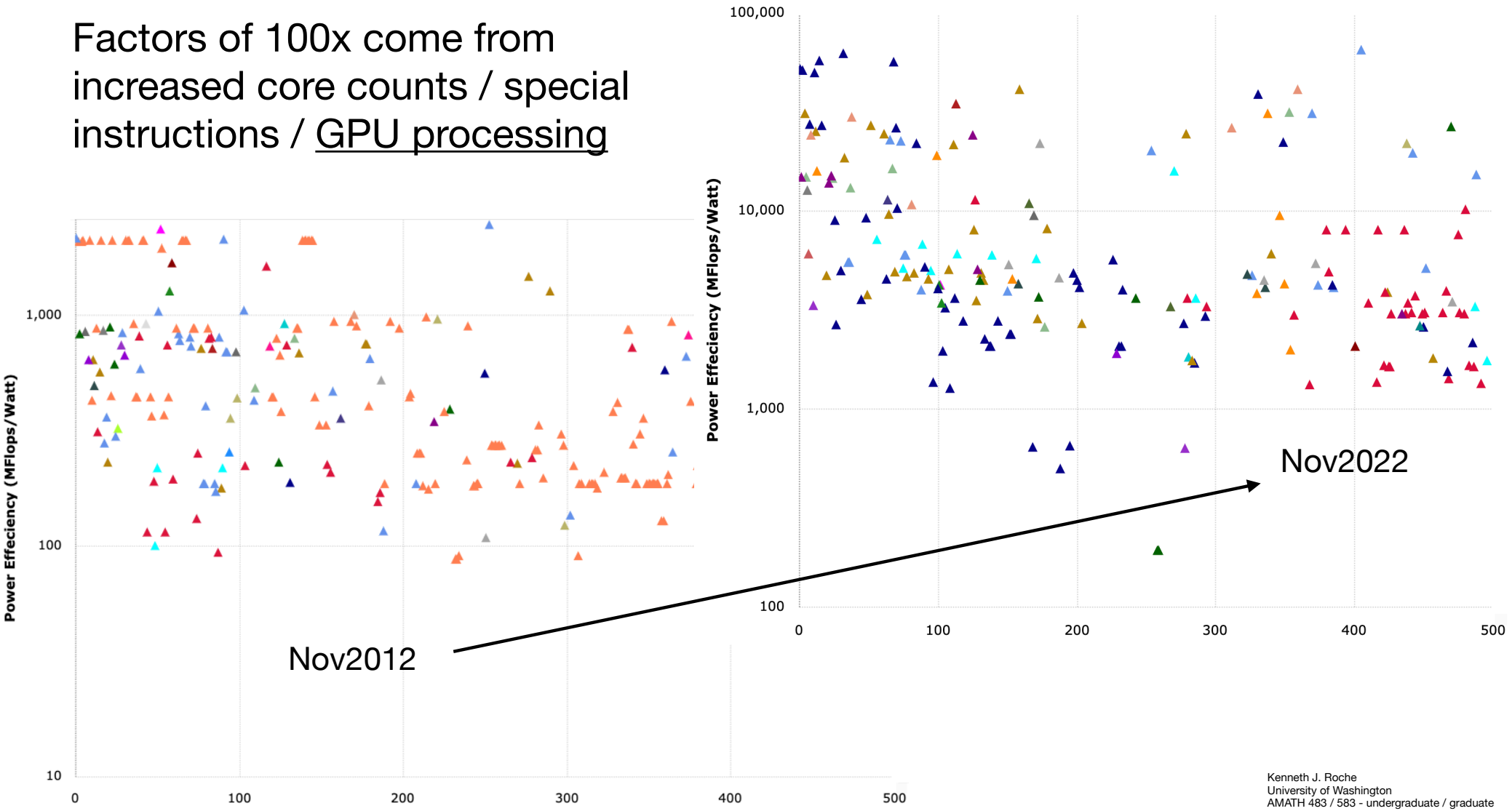




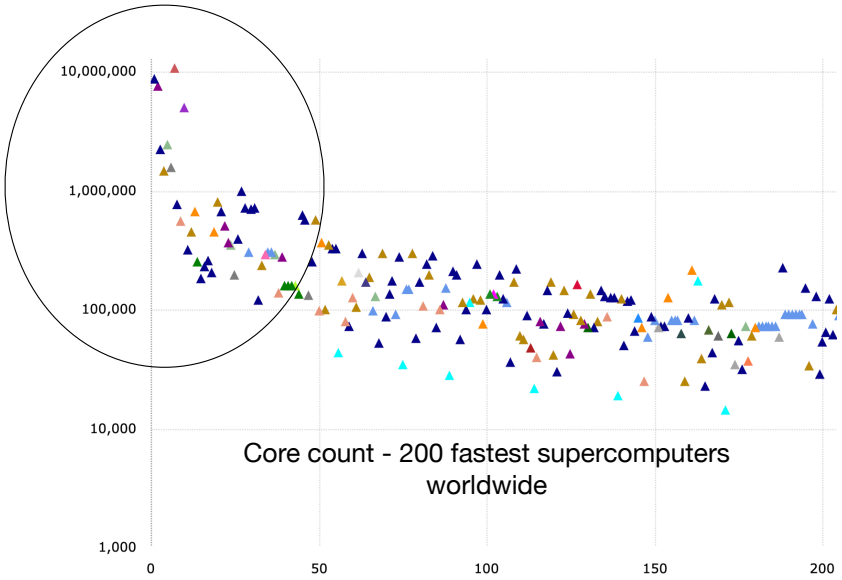
Kenneth J. Roche
University of Washington

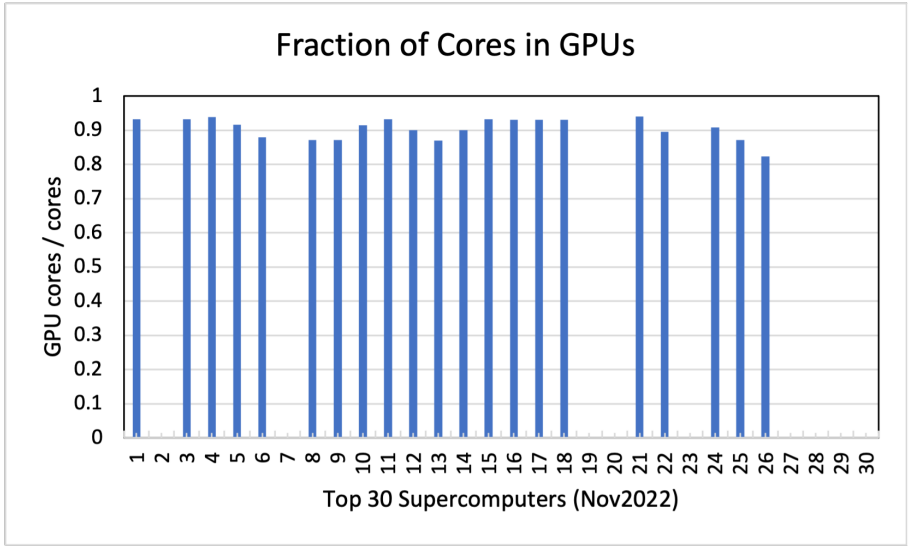
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Factors of 100x come from increased core counts / special instructions / GPU processing



Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

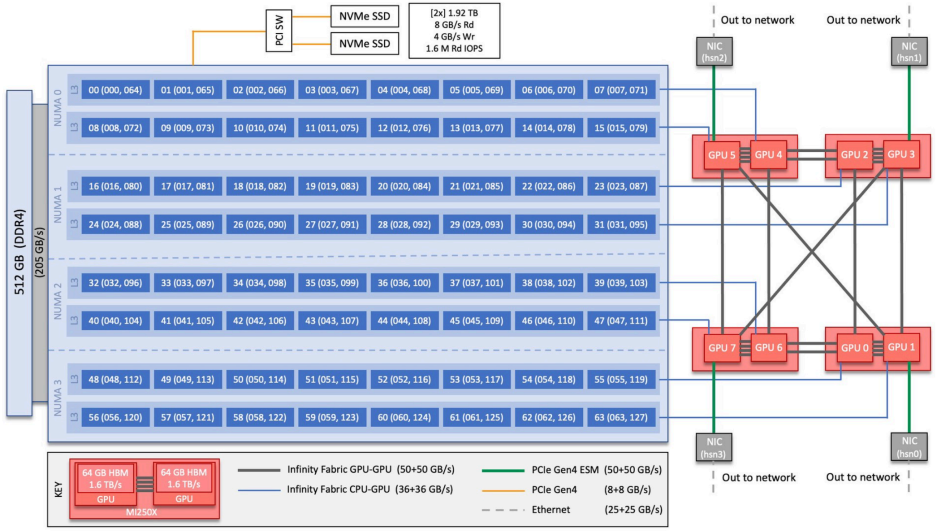


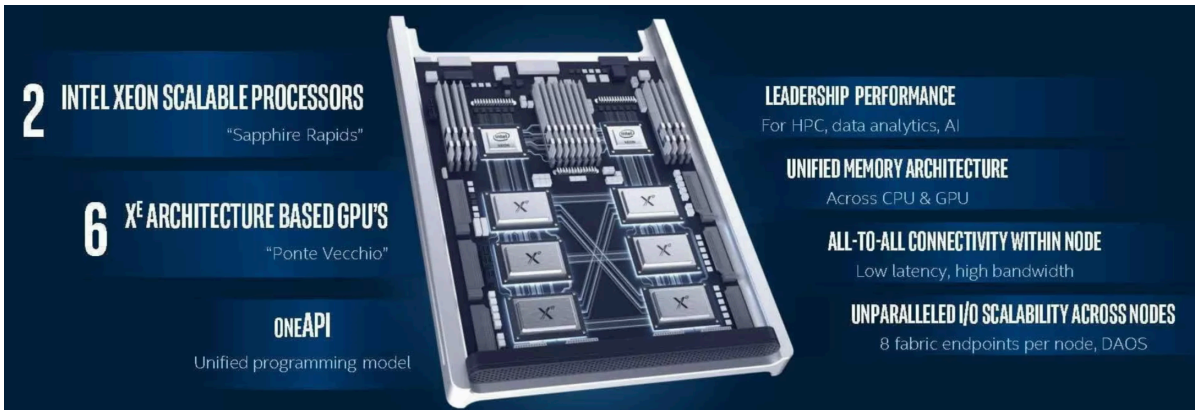


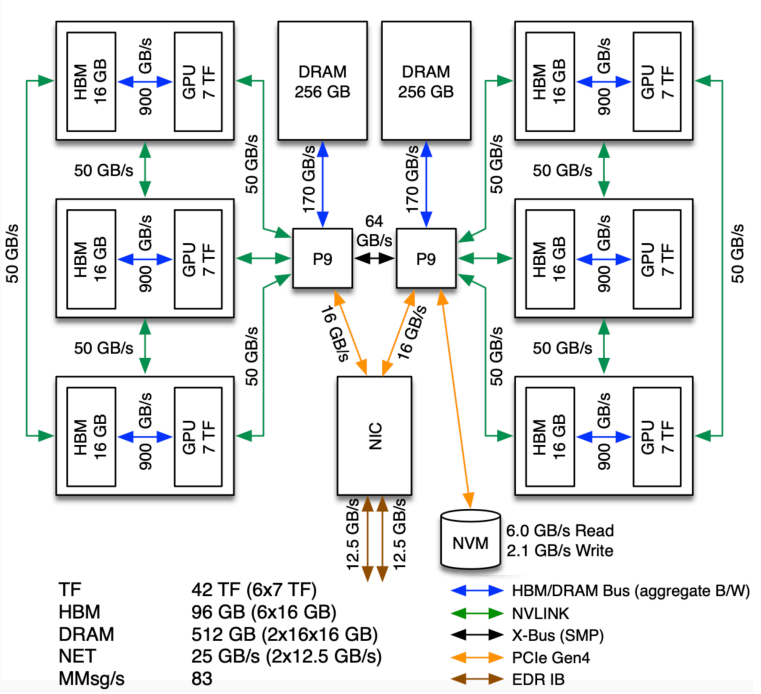
Requirement for increased concurrency to exploit performance and scaling potential of supercomputers forces programmers to design algorithms for their problems that expose instruction-level (ILP) and use thread-level parallelism (TLP)

- Problem dissection is tricky and not all problems are amenable
- Most the heavy lifting of our codes needs to execute efficiently on GPUs /accelerators
- What NP problems fit this picture? Would be useful to have a list …

Aurora, Frontier, Summit : built on 3 GPU families by 3 Vendors with 3 Programming APIs

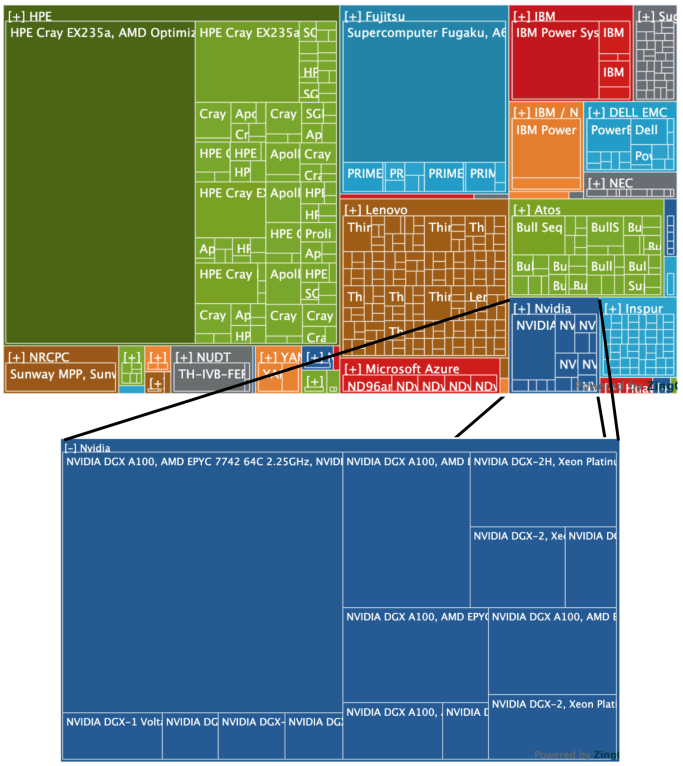






… designs look similar, but programmed differently … frustrating

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025



NVIDIA: CUDA Compute Capabilities from 3.X to 9.X

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

Software

Aurora Software Introduction

- oneAPI
  - oneAPI Overview
  - Intel oneAPI Materials
    - Intel oneAPI DevCloud
    - Intel oneAPI Documentation
    - Intel oneAPI Programming Guide
    - Intel oneAPI Specification Site

Programming Models

- SYCL/DPC++
  - SYCL and DPC++ for Aurora
  - Related Training Materials
    - DPC++ Open Source Github
    - Data Parallel C++Book Chapters
    - A Roadmap for SYCL/DPC++ on Aurora
- OpenMP
  - OpenMP Programming Model
  - Related Training Materials
    - Overview of OpenMP 4.5 and 5.0
    - Features
- Kokkos
  - Kokkos
- RAJA
  - RAJA

Data Science and Workflows

- Related Training Materials
  - Machine Learning with TensorFlow, Horovod, and PyTorch on HPC
  - Effective Use of Python

Performance Tools

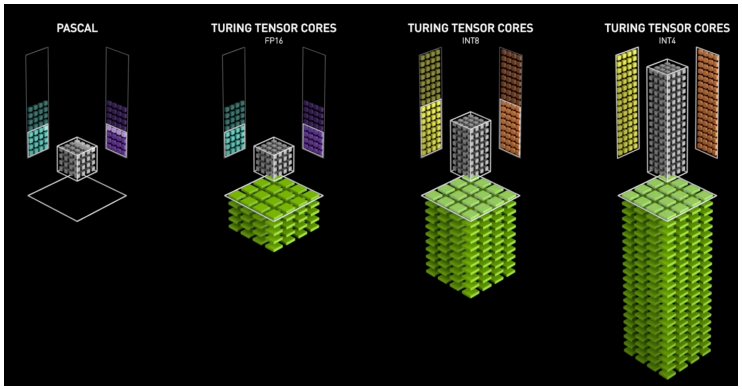
- Related Training Materials
  - Performance Tuning Using Intel Advisor and VTune Amplifier

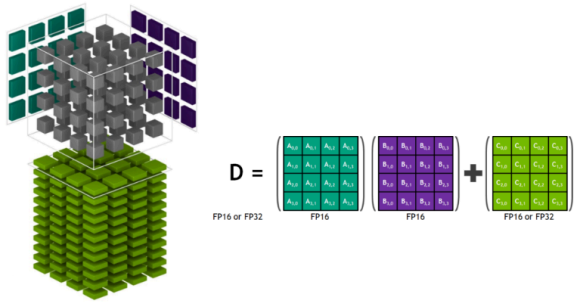
Massive programming efforts to refactor codes to target new exascale platforms

AMD	NVIDIA
Work-items or Threads	Threads
Workgroup	Block
Wavefront	Warp
Grid	Grid

Heterogeneous Interface for Portability (HIP) AMD's GPU programming environment

Disruptive special hardware





- i.e. NVIDIA Tensor Core, AMD Matrix Core Unit technologies
- under-utilized until recently in simulations due to FP8 and FP16 constraints
- reduced bit and mixed precision
- exploit on-chip memory
- HPDA and DLNNs in particular can exploit this tech (made for it)
  - leverage model sparsity by spatially mapping the neural networks to computing tiles
  - remove fetch-decode-execute overheads through dataflow and/or systolic computation
  - FAST!

But how to align our current simulation algorithms with these units?

- need to research impact of reduced and mixed-precision computations on math & physics codes
- develop methods that can deliver high precision numerical evaluations from reduced-bit operations using physics

Kenneth J. Roche
University of Washington

Developers must adjust … we will explore this together

Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

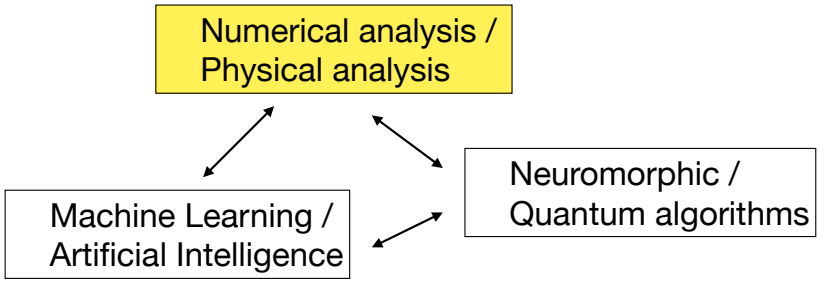
Embrace and Test Disruptive methods

- researchers need to investigate solving PDEs with machine learned techniques …
  - can these replace or improve quality AND performance of current approaches derived from operator theory and discrete numerical analysis?

- most downloaded paper in JCP for a long time introduces method that eschews numerical analysis in favor of combining the physical rules governing the PDE system with data and applying deep learning neural networks

  - PINNs have the massive advantage of industry DNN library stacks for immediate use

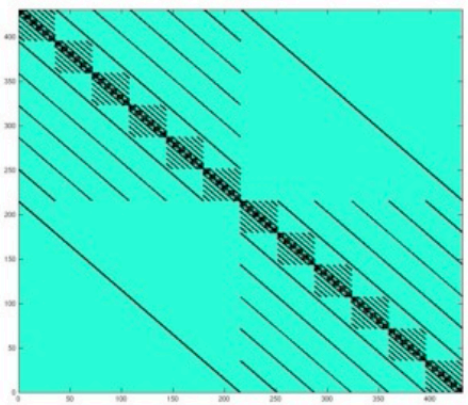
  - effectively utilize the tensor core technology previously mentioned

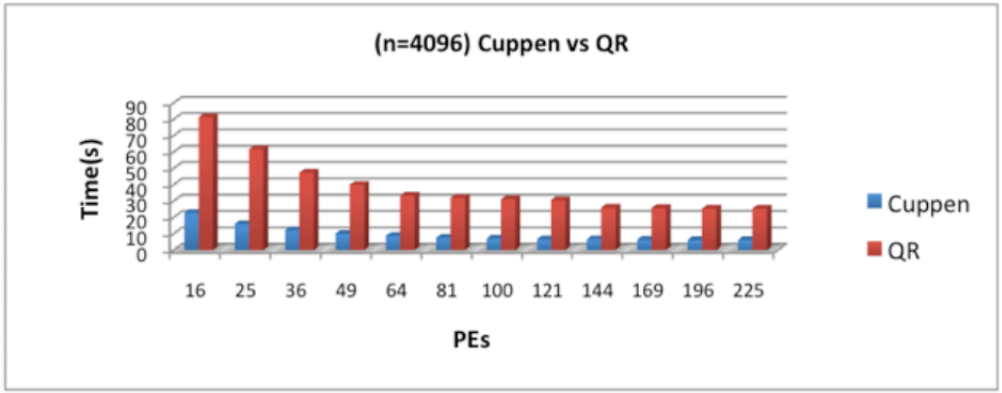


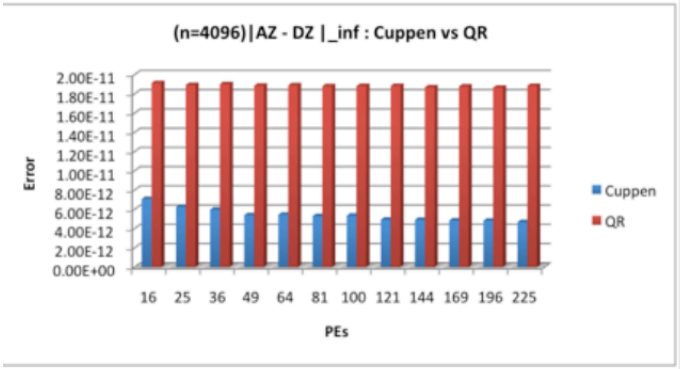
M. Raissi, P. Perdikaris, G.E. Karniadakis,
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,
Journal of Computational Physics,
Volume 378,
2019,
Pages 686-707,
ISSN 0021-9991,
https://doi.org/10.1016/j.jcp.2018.10.045.
(https://www.sciencedirect.com/science/article/pii/S0021999118307125)
Abstract: We introduce physics-informed neural networks – neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations. In this work, we present our developments in the context of solving two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. Depending on the nature and arrangement of the available data, we devise two distinct types of algorithms, namely continuous time and discrete time models. The first type of models forms a new family of data-efficient spatio-temporal function approximators, while the latter type allows the use of arbitrarily accurate implicit Runge–Kutta time stepping schemes with unlimited number of stages. The effectiveness of the proposed framework is demonstrated through a collection of classical problems in fluids, quantum mechanics, reaction–diffusion systems, and the propagation of nonlinear shallow-water waves.

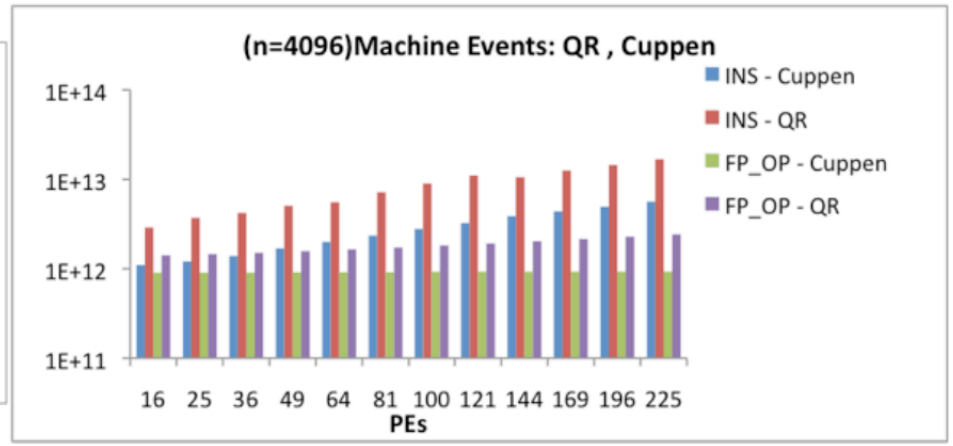
Need algorithms that Improve {ins,*flop(s)} / byte* (and don't compromise accuracy or performance)

•J.J.M. Cuppen, *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*, Numer. Math. 36, 177-195 (1981)
•F. Tisseur and J.J. Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures*, lawn132 (1998)







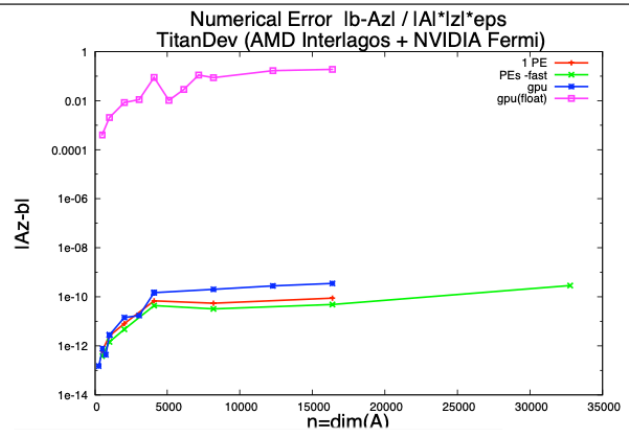


Kenneth J. Roche
University of Washington

Mixed Precision Solvers -faster, controlled accuracy

Require solver tolerance beyond limit of single precision
- DP is at least 2X slower
- Use iterative refinement

Double precision done can be done on CPU or GPU
Disadvantage is each new single precision solve is a restart



Double precision mat-vec and accumulate

```
while ( | r_k | > ε) {
    r_k = b - A x_k
    p_k = A^-1 r_k
    x_k+1 = x_k + p_k
}
```

Inner single precision solve

```
if ( | r_k | < δ | b | ) {
    r_k = b - A x_k
    b = r_k
    y = y + x_k
    x_k = 0
}
```

Reliable Updates (Sleijpen and Van der Worst 1996)
- Iterated residual diverges from true residual
- Occasionally replace iterated residual with true residual
- Also use second accumulator for solution vector

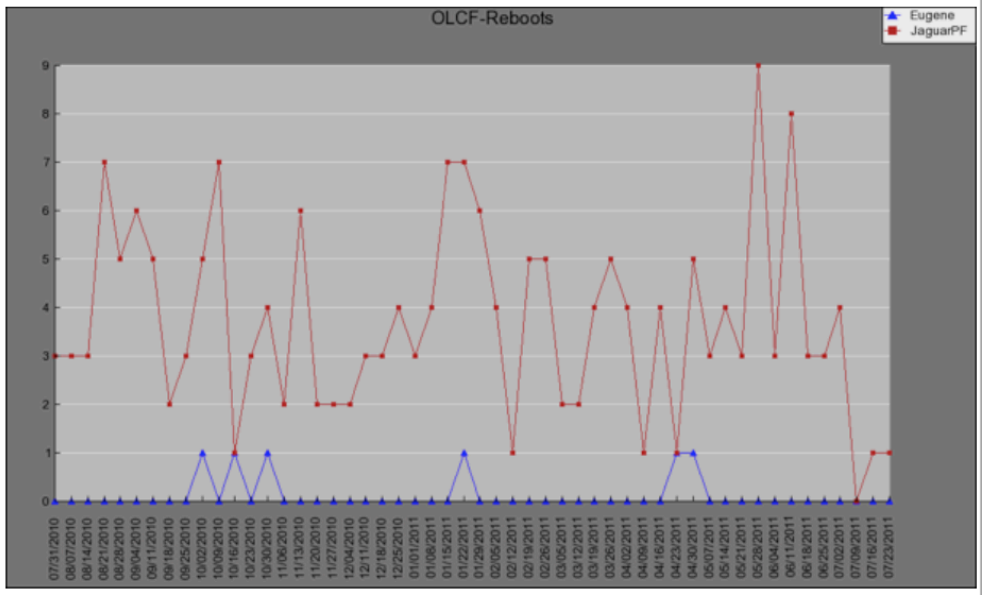
Single-precision can be used to find double-precision result
- GPU kernel is still bandwidth bound
- Use half precision for inner solve?

ref. M. Clark, NVIDIA
my PD, Saul Cohen -multigrid

Need measures for detecting, mitigating, recovering from failures

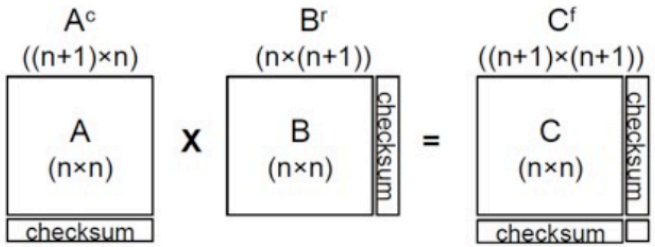
- fail / continue
- hard / soft faults

- resiliency must go beyond check point / restart
  - algorithm based fault tolerance



COSTS / TRADEOFFS ?

have to go beyond single failure



$$\mathbf{A}^c_{n+1,j} = \sum_{i=1}^{n} \mathbf{A}_{ij}$$

$$\mathbf{B}^r_{i,n+1} = \sum_{j=1}^{n} \mathbf{B}_{ij}$$

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \text{ and } \mathbf{C}^f = \mathbf{A}^c * \mathbf{B}^r$$

$$\mathbf{C}^f_{n+1,j} = \sum_{i=1}^{n} \mathbf{C}^f_{ij} \quad \mathbf{C}^f_{i,n+1} = \sum_{j=1}^{n} \mathbf{C}^f_{ij}$$

ref P. Raghavan's work

Need extensions that relate performance to power; lead to novel optimization ideas

-extension of existing metrics to reason about power and performance tradeoffs, energy driven optimizations (i.e. DVFS)

-number of floating point operations per Watt (floating point dominated)

-cost of loads or stores in bytes per Watt (data ops dominated)

-metric guided optimizations to simultaneously minimize power consumption and time to solution (IBM Zurich study)

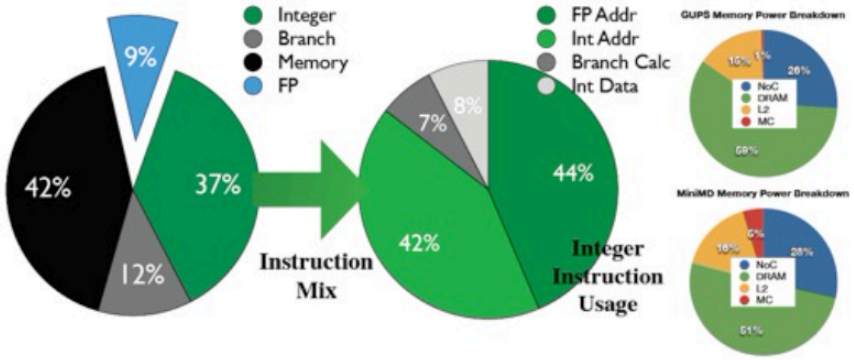
-computational cost ~ *f(time to solution) * energy*

-*f* constant, cost per execution event in Joules

-*f* linear, cost provides insight about appropriateness of hardware platform for application

-demand tools for power measurements

-memory (29%), network (29%), floating point unit (16%)) (distribution of power in HPC hardware (Kogge))



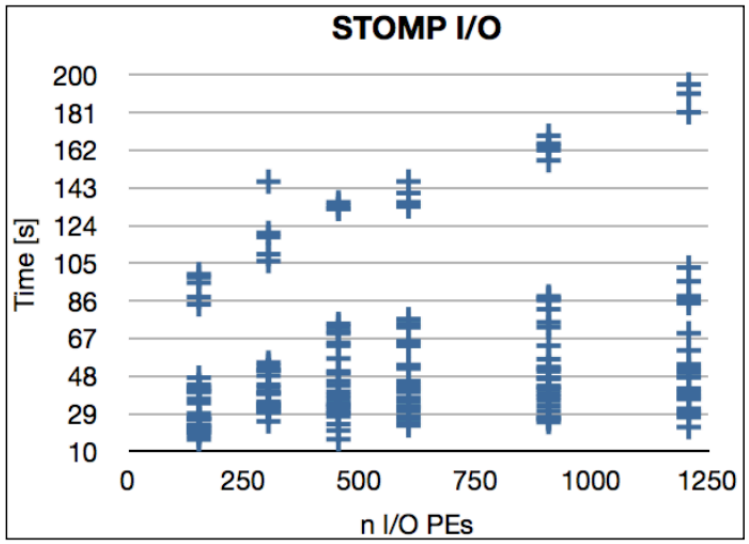
-relate cycle costs in memory refs to energy in Joules

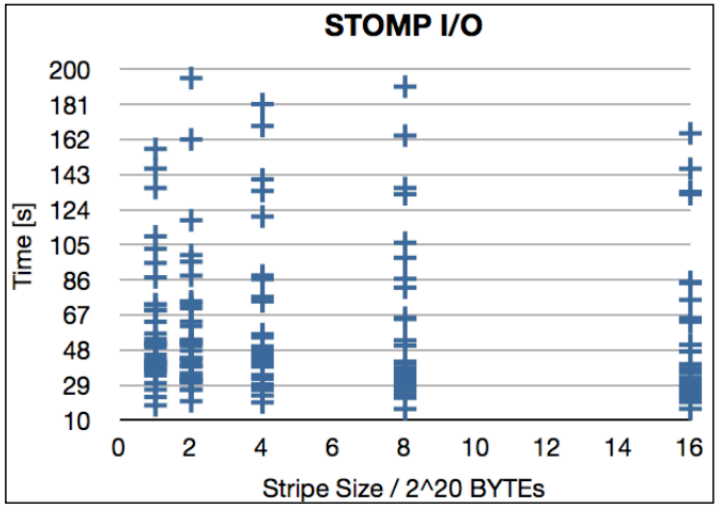
To Where	Cycles
Register	≤ 1
L1d	~ 3
L2	~ 14
Main Memory	~ 240

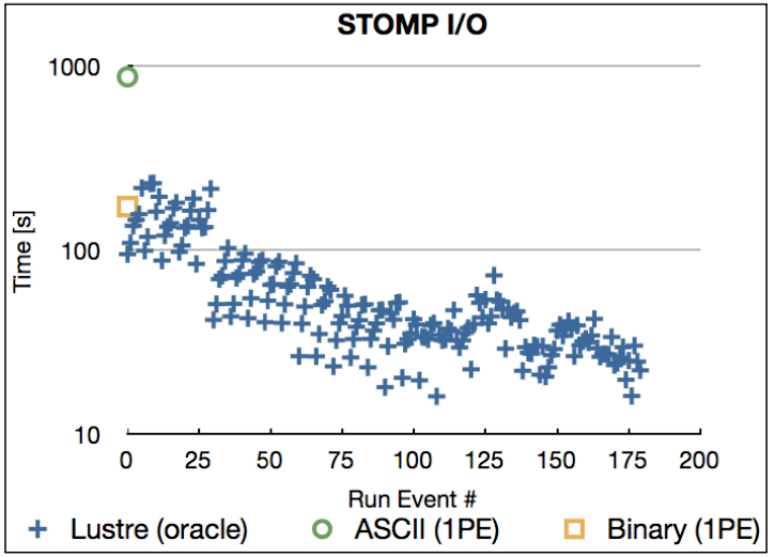
Need measures for I/O operations for applications

Parameters set in the file system related to but independent from the problem parameters:

- Number of OSTs
  1, 2, 4, 8, 16, 32
- Stripe size in BYTEs
  1 MB, 2 MB, 4MB, 8 MB, 16 MB
- access pattern (round robin)
- Number of I/O PEs for spatial decomposition
  kio ~ 1, 2, 3, 4, 6, 8
- Total number of I/O PEs is kio * nfld
  since nfld =151, 151, 302, 453, 604, 906, 1208







Kenneth J. Roche
University of Washington
AMATH 483 / 583 - undergraduate / graduate
High Performance Scientific Computing
Spring Quarter 2025

End Intro Lectures 1,2