



## INTRODUCTION TO PARALLEL COMPUTING

**Class time** : MW 8:15 – 9:30 am  
**Room** : Online via Zoom  
**Instructor** : Yousef Saad

January 19, 2021

### Objectives of this course

- 1 Parallel computers, why they are needed, their architectures, various modes of parallel processing
- 2 Parallel programming including openMP, MPI, CUDA, ..
- 3 How to think parallel and design parallel programs. How to optimize parallel algorithms. Illustrations with a few applications
- 4 Novel horizons. In particular a brief introduction to quantum computing.

1-2

– Start

1-2

### Who is in this class:

- Yousef Saad - (me) Instructor
- Jim Mooney (Teaching Assistant)
- You - that is approximately 74 registered students:
  - CS graduate students ( $\approx 26$ )
  - CS Undergraduate students ( $\approx 20$ )
  - Data Science Grad. (9)
  - CE Undergrad. (4)
  - + math, Physics, economics, ...

➤ Please fill out (now if you can)

[This survey](#)

Short link url:

<https://forms.gle/k4yDw3RoPvMUyDAh8>

1-3

– Start

1-3

1-4


– Start

1-4

## Before we begin....

- Lecture notes will be posted in the cselabs class websites:

URL : [www-users.cselabs.umn.edu/classes/Spring-2021/csci5451](http://www-users.cselabs.umn.edu/classes/Spring-2021/csci5451)

- Lecture notes + some minimal information will be posted here. Everything else: Canvas.
- Links between the two (to and from Canvas) available
- Review lecture notes and try to get some understanding – if possible before class.
- Lecture note sets are grouped by topics - not by lecture.
- In the notes the symbol  indicates quick questions or suggested exercises

1-5  – Start

1-5

- Lecture notes will occasionally contain URL's. These are 'clickable' hyperlinks – For example this one:

<http://www.cs.umn.edu/~saad/teaching>

or [This one](#)

- This is an evolving document.. occasional reposts will be made
- I will often post on Canvas, code samples for demos seen in class
- Do not hesitate to contact me for any questions!

1-6  – Start

1-6

## A few resources

- Quite a few resources out there.

### For material covered in lectures:

- Book by Grama, Gupta, Karypis, & Kumar (2003, see syllabus for details)
  - Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques (104 pages) by Uzi Vishkin – October 12, 2010. see <http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf>
  - Lecture notes / Tutorials: Course from U. Cal. Berkeley see <https://sites.google.com/lbl.gov/cs267-spr2019/>
- More links will be included in lecture notes

1-7  – Start

1-7

## For parallel programming

- I would recommend referring to the books mentioned in syllabus. But there are many help sites available.

### MPI

- Tutorials: <https://mpitutorial.com/tutorials/>
- LLNL site: <https://computing.llnl.gov/tutorials/mpi/>

### CUDA

<https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/>

### openMP

<https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf>

More will be added as needed

1-8  – Start

1-8

*Schedule*

-  [PDF version of schedule](#)

Jan Feb					
01/20	02/01	02/10	02/10	02/17	
1st Day	HW1 Posted	HW1 due	Lab1 posted	Quiz 1	
March					
03/03	03/03	03/10	03/17	03/17	03/24
Lab1 Due	HW2 posted	Quiz2	HW2 Due	Lab2 posted	Quiz3
April-May					
04/02 (F)	04/11 (M)	04/14	04/28	05/03	
Lab2 Due	Lab3 Posted	Quiz 4	Lab3 due	Quiz 5	

**Notes:**

- Spring Break: Apr. 5--Apr. 9.
- There is a quiz on last day of class [May 3rd -- which is a Monday - All other quizzes are on a Wednesday]. Note also that lab2 is due on a Friday.
- No final exam
- Please read: additional information on syllabus and [Policy and general info on homeworks and exams](#)

## GENERAL INTRODUCTION

- A short history
- Why parallel computing?
- Motivation: Scientific applications
- Levels of parallelism
- Introduction to parallelism: algorithms, complexity for a simple example.
- Obstacles to efficiency of parallel programs
- Types of parallel computer organizations

1-10

*Historical perspective: The 3 stages of computing*

1. **Mechanization of arithmetic** (abacus, Blaise Pascal's adder [1623-1662], various calculators, Leibnitz [1646-1716]).
2. **Stored Programs:** Automatic looms [1801]; punched cards ; ..
3. **Merging of Mechanized Arithmetic & Stored Programs** • Babbage (1792-1871)
  - Hollerith (census) → IBM (1924)
  - Mark 1 computer [1944]– 1st electromechanical computer.
  - ENIAC : electronic computer (1946) - Upenn., and then UNIVAC (1951) first real computer [used for census]

*The four (or five?) generations:*

**First:** vacuum tubes;

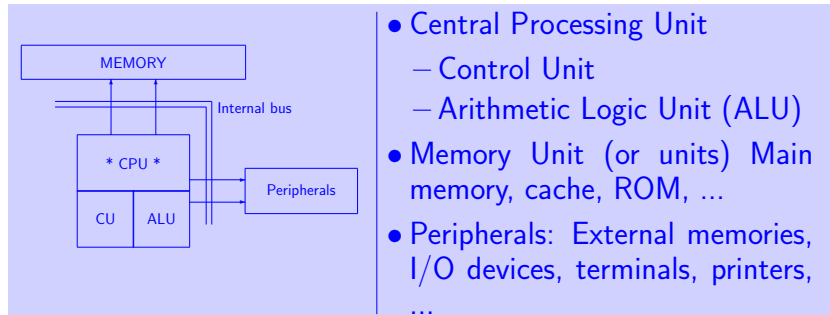
**Second:** Transistors (1959-1965)

**Third:** Integrated circuits (1965-1970's)

**Fourth:** Very Large Scale Integration. (VLSI); (1975 - ) microprocessors

**\*\*** In the 80s there was a plan [in Japan] for a *fifth generation* based on AI ideas [LISP] and massive parallelism. This generation is with us but not as envisioned.

## The Von Neumann architecture

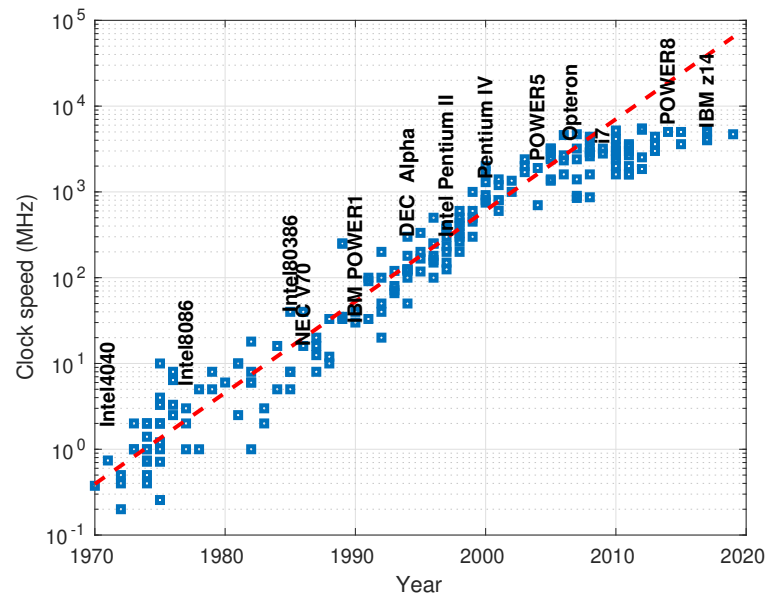


- All components linked via one internal bus


## Execution of Von Neuman programs


- Fetch next instruction from memory into instruction registers
  - Fetch operands from memory into data registers
  - Execute instruction
  - Load result into memory
  - Repeat: fetch next instruction ...
- **Parallelism** was slowly introduced into this sequential scheme from very early on
  - Became a major force starting in the late 1980s

## Why parallelism? 50 years of clock speeds



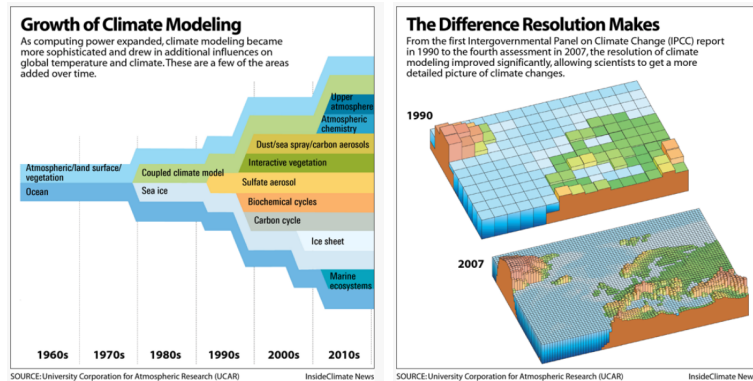
- **Main argument:** Harder and harder to increase clock speeds.
- Sustainable gains in speed, are only possible through better software, better utilization of hardware, and **parallelism**
- Parallelism is cost effective: multiplying hardware is cheap; fast components are expensive.
- Parallelism also helps from memory stand-point (multiplying memory is cheap, building a system with a single large memory is expensive).

 We want a Processor that delivers  $10^{12}$  floating points operations per second. To get one flop per cycle, assuming transfer at speed of light ( $3 \times 10^8 m/sec$ ), you need a data operand to travel no more than  $\leq l = ?$

 Let  $l$  be your answer from above. We want to pack  $10^{12}$  bytes on an  $l \times l$  chip. What (approximate) area will be occupied by 1 bit?

## Why supercomputers? Weather simulation

- What can supercomputers do that ordinary machines can't?



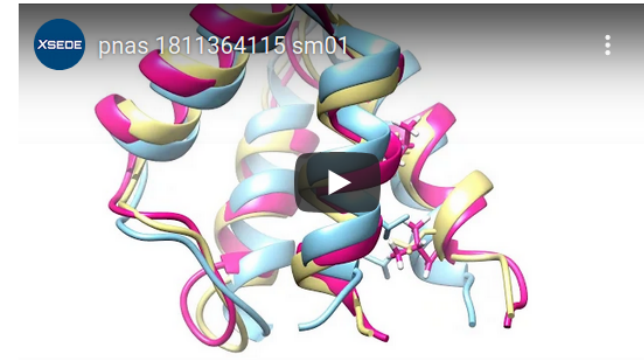
See details in original article: [40 years of the world climate research](#)

1-17

1 – Intro

1-17

## Why supercomputers? Molecular dynamics



See original article: [Supercomputer-Powered protein simulation](#)

1-18

1 – Intro

1-18

## Why supercomputers? Deep Learning



Source: BSC-CNS

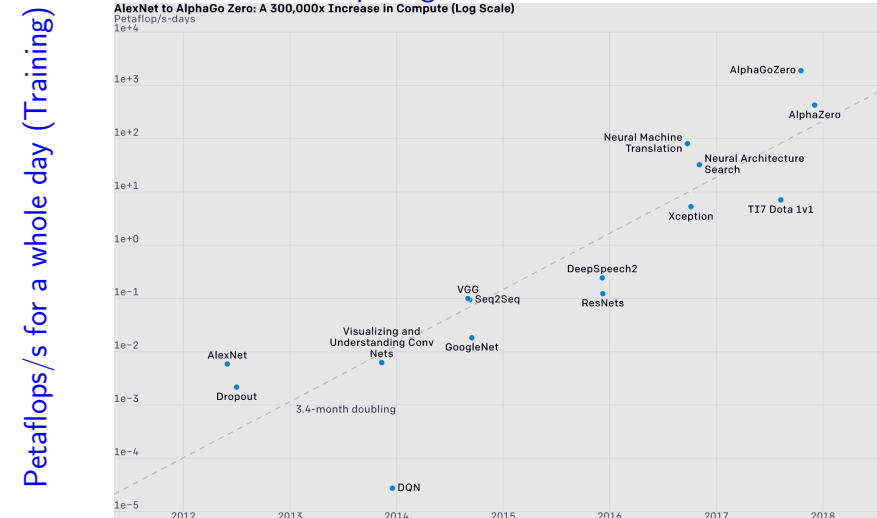
See original article: [Supercomputing=heart of Deep learning](#)

1-19

1 – Intro

1-19

## Growth of ML computing:



See: <https://openai.com/blog/ai-and-compute/>

1-20

1 – Intro

1-20

## Why supercomputers? PFLOPS=??

**Q:** What's on the vertical axis?

**A:** flop/s == floating point operations per second

Kiloflops	KFLOP/s	=	$10^3$	flop/sec
Megaflops	MFLOP/s	=	$10^6$	flop/sec
Gigaflops	GFLOP/s	=	$10^9$	flop/sec
Teraflops	TFLOP/s	=	$10^{12}$	flop/sec
Petaflops	PFLOP/s	=	$10^{15}$	flop/sec
Exaflops	EFLOP/s	=	$10^{18}$	flop/sec
Zettaflops	ZFLOP/s	=	$10^{21}$	flop/sec

➤ So 1 PFLOPs\* 1 day =  $10^{15} * 3600 * 24 \approx 86 \times 10^{18} = 86$  EFLOPS

➤ Similar terminology for memory, e.g., 1 Terabyte =  $10^{12}$  bytes.

1-21

1 - Intro

1-21

## The Top 500 list

- List of 500 fastest supercomputers in the world
- Started in 1993. Updated twice a year [once in Europe and once in the US.]

➤ Based on 'linpack' benchmark [solving a large dense linear system]

➤ It is getting gradually clear that a 'linpack' benchmark is inadequate. The computational world is mixed: sparse, dense, low precision, ...

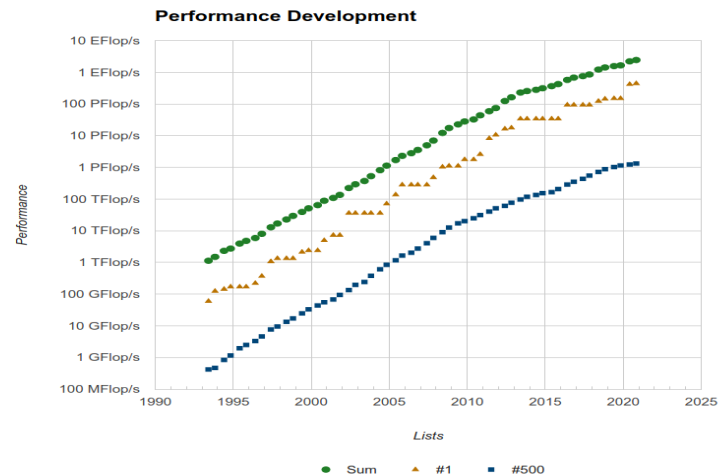
➤ Next chart shows performance of top computer (# 1 in list), bottom computer (# 500 in list) and the sum over all 500 – from 1993 to date.

1-22

1 - Intro

1-22

From the top500 site:

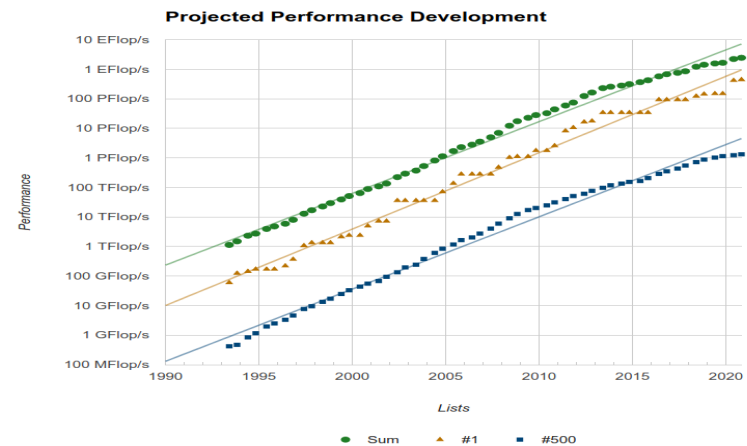


1-23

1 - Intro

1-23

Development is slowing down: ...



1-24

1 - Intro

1-24

**Q:** How are machines ranked?

**A:** A benchmark based on 'linpack' (now LAPACK) – essentially solving a dense linear system.

This is different from the **peak performance** = highest possible flop count that can **theoretically** be achieved.

 Suggested hw: visit the web site of the top 500 fastest computers:

<https://www.top500.org/lists/2020/11/>

Let us take a quick look..

1-25

**Top:** Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C. (Japan) Riken center for computational sciences. 7.6 M cores, 5 PetaB memory (5M GB) Linpack Benchmark: 442 Petaflops.



1-26

1 – Intro

1-26

Second machine:

Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband

#### General

Manufacturer: IBM  
Cores: 2,414,592  
Memory: 2,801,664 GB  
Processor: POWER9 22C  
3.07GHz  
Interconnect: Dual-rail Mellanox  
EDR Infiniband



#### Performance

Linpack Performance (Rmax) 148,600 TFlop/s

1-27

1 – Intro

1-27

#### *New things that are emphasized*

- Performance for data analytics: approximately 3 EFlop/s with Mixed precision (e.g. for deep learning)
- Energy consumption Power: 10 Megawatts. Enough to power ~ 5,000 homes.
- There is also a ranking by “greenness” the ‘Green500’ List [the number one machine is made by Fujitsu] for details see

<https://www.top500.org/lists/green500/2020/11/>

1-28

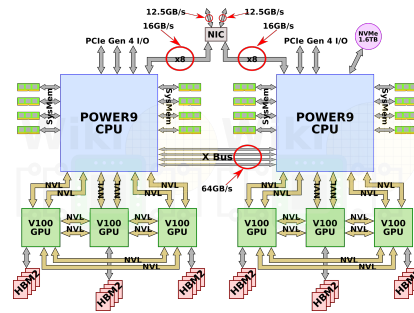
1 – Intro

1-28

## Summit's processors:

### Each node:

- 2 POWER9 proc.s
- 6 Nvidia Tesla V100 GPUs
- 608 GB fast memory
- 1.6 TB of NVMe memory



- Approximately 40 TFlop/s per node
- each node: 96 GB of HighBandwith Memory (HBM2) for use by acc's
- Each Power9 has 22 cores
- Each V100 GPU: 80 Streaming Multiprocessor [SMs]

1-29

1 - Intro

1-29

## Discussion

- 5 What are the main components that characterize a supercomputer?
- 6 For Summit, can you tell how the total number of cores (2,414,592) was obtained? [hint: there are 4,608 nodes, each with 2 Power9s (22 cores each) and 6 Nvidia V100]
- 7 Similarly how is the Peak rate (Rpeak) of 200 PFlop/s obtained? [Hint: Look at the V100 and power9 peak rates]

1-30

1 - Intro

1-30

- \* Note: they counted a streaming multiprocessor as one 'core'
- \* Below is a V100 with 84 SMs- Locate one SM on the figure



\* Here is one SM:

8 Count the total number of FP32 CPUs in a V100 board with 84 SM's

9 Same question for FP64 CPUs



1-32

1 - Intro

1-32

1-31



## Parallel computing – Motivation – Moore's Law

- Demand for computational speed is always increasing. Trend is accelerating due to new developments: demand in biology, genomics, as well as in physics/chemistry.
- It was realized around the mid 1970s that gains in raw speed (clock cycle) were becoming harder to achieve.
- From 1950 to the mid 70s: speed gained a factor of 100,000
- 3 orders of magnitude due to clock cycle. The other 2 to architecture and design.
- A factor of about 10 every 5 years.
- From 1970 to 2005: gain of 5000 – [a factor of 2 every 3 years]
- Then from 2005 on – gain in clock cycle basically stalled.


1-33

1 – Intro

1-33

- In 1975 Cray (a Minnesota company!) unveiled its first commercial supercomputer. Clock cycle: 12.5 ns. (i.e., freq. of 80MHz.)

- A 'vector' supercomputer
- Peak speed: 160 Megaflops

 10 How do we get the peak speed of 160 Megaflops?

- Your laptop is faster [and does not cost \$ millions]



- Clock-cycles are one of the consequences of large scale integration [milli, micro, nano, ..]

- Today some chips have over a few billion transistors (!)

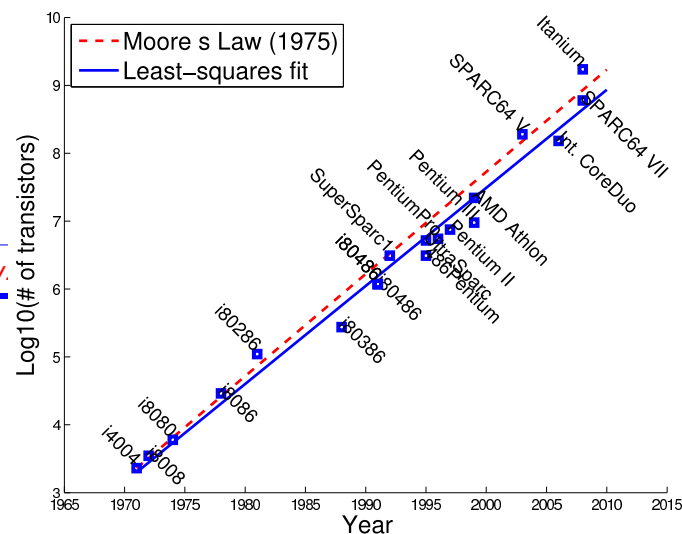
- In the early 1970s: thousands only.

1-34

1 – Intro

1-34

Number of transistors that can be placed on a chip doubles every 2 years (Moore's Law).




1-35

1 – Intro

1-35

- Moore's law has been incredibly accurate considering that it was stipulated in 1965! [actual ratio corrected in 1975.. still....]

 11 Determine what processor is in your laptop find out the number of cores, and clock speed.

 12 Continuation: see if you can determine its number of transistors [Hint: search the web!]

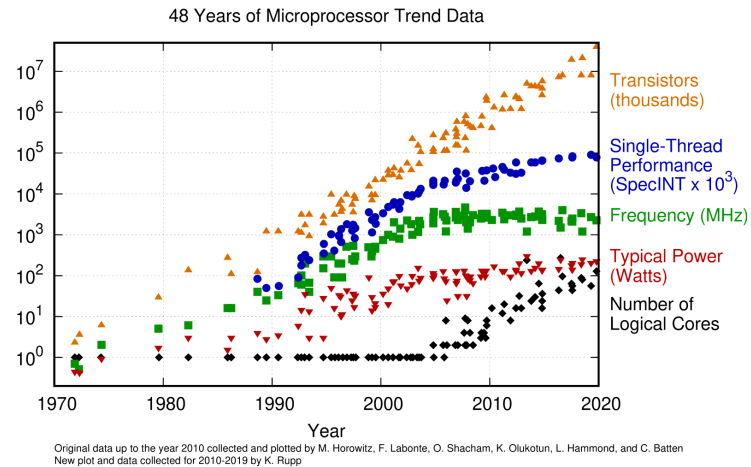
1-36

1 – Intro

1-36

## 48 years of Processor Trends

Remarkable chart [from M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, and K. Rupp]




1-37

1 – Intro

1-37

## A few computationally intensive applications

**Weather forecasting** - Partial differential equations – Time dependent problems involving several unknowns per mesh point – leads to very large nonlinear models.

 13 Assume a region of  $10,000\text{km} \times 5,000\text{km}$  and a height of  $10\text{km}$ . If cells of 1 cubic km are used: what is the total number of cells?

► Challenge: complete the calculation for tomorrow's weather before tomorrow.


**Device/ circuit simulation.** Computer chips have in the order of billions of gates. Numerical simulation of flow of electrons and holes is exceedingly complex. Simplifications are common to reduce size.

1-38

1 – Intro

1-38

**Reservoir simulation** (petroleum engineering) - Typical reservoir size: a few tens of kms, a few tens of meters deep. Problem: find concentration of oil in the ground over time. Oil companies were the first commercial buyers of supercomputers.

 14 Calculate the system size for the case of a  $10\text{km} \times 10\text{km} \times 400\text{m}$  reservoir, when a mesh-size of 2m is used (one node every 2m in each direction).

**Electronic structures calculations / quantum mechanics**

Chemists and physicists doing electronic calculations are the biggest users of supercomputers.

1-39

1 – Intro

1-39

## • The Gordon Bell awards. <https://awards.acm.org/bell>

2020

2020 ACM Gordon Bell Prize Awarded to Team for Machine Learning Method that Achieves Record Molecular Dynamics Simulation

ACM, the Association for Computing Machinery, named a nine-member team, drawn from Chinese and American institutions, recipients of the 2020 ACM Gordon Bell Prize for their project, "Pushing the limit of molecular dynamics with *ab initio* accuracy to 100 million atoms with machine learning."

2019

2019 ACM Gordon Bell Prize Awarded to ETH Zurich Team for Developing Simulation that Maps Heat in Transistors

DaCe OMEN Framework Could Help Industry Design Better and More Efficient Computer Chips

Denver, CO, November 21, 2019 – ACM, the Association for Computing Machinery, named a six-member team from the Swiss Federal Institute of Technology (ETH) Zurich recipients of the 2019 ACM Gordon Bell Prize for their project, "A Data-Centric Approach to Extreme-Scale *Ab initio* Dissipative Quantum Transport Simulations."

1-40

1 – Intro

1-40

2018

### Two Teams Honored with 2018 ACM Gordon Bell Prize for Work in Combating Opioid Addiction, Understanding Climate Change

ACM named two teams to receive the 2018 ACM Gordon Bell Prize. A seven-member team affiliated with the Oak Ridge National Laboratory is recognized for their paper "Attacking the Opioid Epidemic: Determining the Epistatic and Pleiotropic Genetic Architectures for Chronic Pain and Opioid Addiction," and a 12-member team affiliated with the Lawrence Berkeley National Laboratory is recognized for their paper "Exascale Deep Learning for Climate Analytics."

2017

### 2017 ACM Gordon Bell Prize Awarded to Chinese Team that Employs the World's Fastest Supercomputer to Simulate 20th Century's Most Devastating Earthquake

ACM named a 12-member Chinese team the recipients of the 2017 ACM Gordon Bell Prize for their research project, "18.9-Pflops Nonlinear Earthquake Simulation on Sunway TaihuLight: Enabling Depiction of 18-Hz and 8-Meter Scenarios." Using the Sunway TaihuLight, which is ranked as the world's fastest supercomputer, the team

- New: the 2018 and 2029 awards are data-centric or use DNN. More and more ML or ML+ awards made.

1-41

1 – Intro

1-41

15

What was the very first \*commercial\* computer ever built? what was its speed [in Floating Point Operations per seconds?]

16

When and where was the very first \*commercial\* 'vector' supercomputer built? [hint: it was in a cold snowy place] – what was its approximate speed?

1-42

1 – Intro

1-42

### *New Horizon: HPC for machine learning*

- Computing with Data or Infering with data has become very important.
- High-performance has played a big role in machine learning and is in part responsible for the rebirth of Deep Learning.
- The ImageNet competition (Large Scale Visual Recognition Challenge) – was a competition that evaluates algorithms for object detection and image classification held from 2010 to 2017 <http://www.image-net.org/challenges/LSVRC/>
- In 2012 a convolutional neural network (CNN) called AlexNet won the competition – achieved top-5 error of 15.3% in the ImageNet 2012 Challenge. GPUs (Graphical Processing Units) were a key ingredient

1-43

1 – Intro

1-43

### *New Horizon: Quantum computing (?)*

- Note: The question mark is deliberate

➤ Idea: Harness the unusual properties of the quantum physics world to perform large calculations

- Potential is enormous ...
- ... But so are challenges.
- Governments believe in QC as the next big thing ... [funding]
- QC - will be discussed a little at end of course

1-44

1 – Intro

1-44

## Basics of parallel computing

Simple example of calculating the sum of  $n$  numbers

$$s = \sum_{i=0}^{n-1} x_i$$

Sum is traditionally computed as :

### ALGORITHM : 1. Sum of $n$ numbers

```
s = 0;
for (i = 0; i < n; i++)
    s += x[i];
```

- $n - 1$  “sequential” operations
- How can we introduce/exploit parallelism?

1-45

1 - Intro

1-45

- Split the sum in  $p$  subsums. Assume  $n = p * m$ .

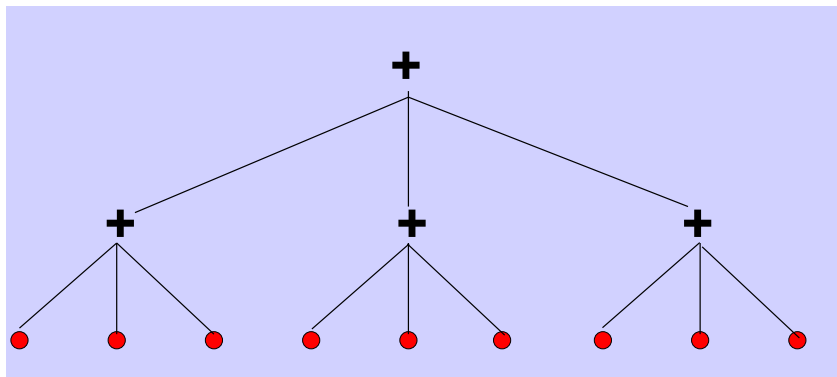
### ALGORITHM : 2. Parallel Sum of $n$ numbers

```
for (j = 0; j < p; j++) { // Parallel Loop
    tmp[j] = 0;
    for (i = j * m; i < (j + 1) * m; i++)
        tmp[j] += x[i];
}
s = 0;
for (j = 0; j < p; j++) // Sequential loop
    s += tmp[j];
```

1-46

1 - Intro

1-46



- Number of operations:  
 $p * (m - 1) + p - 1 = n - 1$  (unchanged)
- Each subsum is done independently

1-47

1 - Intro

1-47

- Assume  $n = 2^k$  and  $n/2$  arithmetic units (“processors”) are available. Then, can apply the idea recursively.
- Divide into 2 subsums – each subsum is divided in two subsums etc.. Yields the *cascade sum*:

### ALGORITHM : 3. Cascade Sum of $n$ numbers

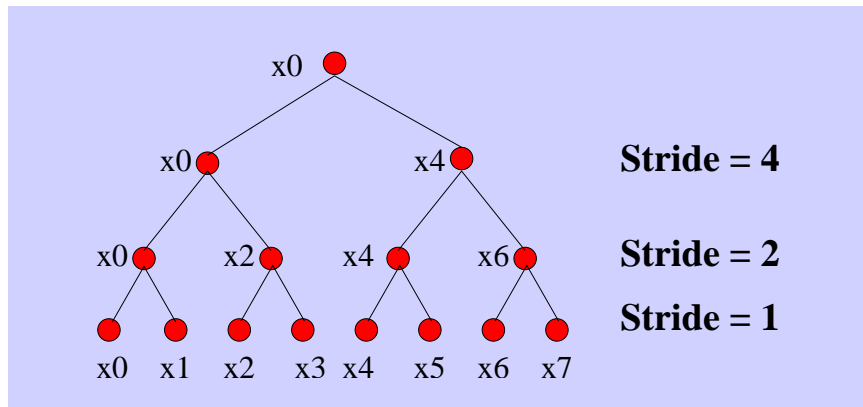
```
// Sequential loop:
for (stride = 1; stride < n; stride *= 2) {
// Parallel Loop:
    for (j = 0; j + stride < n; j += 2 * stride)
        x[j] += x[j + stride];
}
```

- Final result in  $x[0]$ . Number of  $s$  needed:  $\log_2 n$ . Can be generalized to case where  $n$  not a power of 2.

1-48

1 - Intro

1-48



- A sum of  $n$  numbers can be computed in Order  $\log(n)$  time units where a time unit is the time to perform an add.
- Similarly for the product of  $n$  numbers.
- Is it always worth doing this ?

**Example:** Assume 64 students in this class. Can compute the sum of 128 numbers by cascade algorithm.

- 17 (1) How to organize the calculations?
- 18 (2) What are the other times involved (i.e., other than the times needed for adding)?
- 19 (3) Will it be cost-effective (time-wise)?
- 20 (4) Assume now that we need to add 128 matrices of size  $10 \times 10$  each - by hand. Can the cascade algorithm be used? What has changed?

### Levels of parallelism

Five different types of parallelism are commonly exploited:

1. At **job level**, i.e.; between different running jobs.
2. At **'Macrotask' level**: execution of different parallel sections of a given program; (often termed "coarse-grain" parallelism)
3. **'Microtask' level**; for example parallelism related to different steps of a loop;
4. **Data-parallelism**; Same operation performed on similar data sets (e.g. adding two matrices, two vectors). Hardware support provided for such operations.
5. At the **arithmetic level** (pipelining, multiple functional units, etc..)

- There is a common distinction between
  - **Fine grain parallelism**: arithmetic operations or loop level, i.e., levels 3, 4 and 5 in previous list.
  - **Coarse grain parallelism** : bigger tasks ('macrotasks') are executed independently.
- In coarse-grain parallelism the user often defines the macro-tasks and programs the parallelism explicitly.
- In contrast fine-grain parallelism is often (not always) self-scheduled ('automatically' determined) or inherent to the language

**Example:** Fine grain parallelism: the SAXPY loop [BLAS1]

```
c  FORTRAN-90 CONSTRUCT:
    x(1:n) = x(1:n) + alp*d(n1:n1+n-1)
```

**Example:** Coarse grain parallelism: parallel execution of calls to a given subroutine:

```
#parallel for
  for (dom=0; dom<n_regions; dom++) {
// solve in each domain
    pde_solv(neq,domain, x, y, z, ...)
  }
```

➤ Each step solves a Partial Differential Equation on a different subregion.

## Barriers to Parallel Efficiency

➤ The example of adding  $n$  numbers can give an indication to a few of the potential barriers to efficiency:

**1. Data movement.** Data to be exchanged between processors. Data may have to move through several stages, or other processors, to reach its destination.

**2. Poor load balancing.** Ideally: all processors should perform an equal share of the work all the time. However, processors are often idle waiting data from others.

**3. Shynchronization.** Some algorithms may require synchronization. This global operation (requires participation of all processors) may be costly.

**4. Impact of Architecture.** Often, in parallel processing, processors will be competing for resources, such as access to memory, or cache..

**5. Inefficient parallel algorithm.** Some sequential algorithms do not easily parallelize. New algorithms will be required - which may be inefficient in sequential context.

## Three common parallel programming paradigms

**1. Shared global memory viewpoint** – programs will execute in parallel (parallelization with or without help from user) and access a shared address space. [example: Open MP.] → **Symmetric Multi-Processing (SMP) (!)**

**2. Message passing viewpoint** – programs to run in parallel with data exchange explicitly programmed by user. [example: MPI]

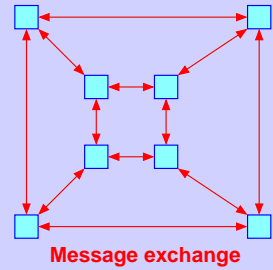
**3. SIMD/ SIMT viewpoint** programs will execute regular computations (vectors, matrices) extremely fast on specialized hardware. [Vector computers, GPUs, ..]

➤ Drawbacks with (1) : Need to make sure data being used is the latest one - need to synchronize.

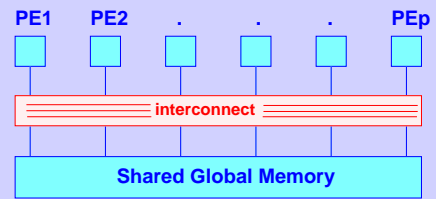
➤ Drawbacks with (2) : difficult to program.

➤ Drawbacks with (3) : what about irregular computations?

## Message passing



## Shared memory



- In the 70's and 80's (1) was very popular ["parallelizing compilers"] Then message passing gained ground with libraries such as PVM and MPI..
- Programming models for (1) and (2) are very different