Introduction

Parallel Computing – COSC 3P93

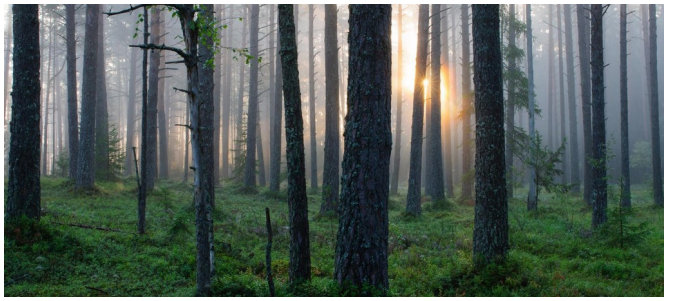
Course Outline

- **Introduction**
- **Parallel Hardware**
- **Parallel Software**
- **Performance**
- **Parallel Programming Models**
- **Patterns**
- **SMP with Threads and OpenMP**
- **Distributed-Memory Programming with MPI**
- **Algorithms**
- **Tools**
- **Parallel Program Design**

Today Class

- Overview
  - What, Why, Who
  - Why Parallel Computing?
  - Concepts and Terminology
  - Single-processor Computing
  - Some Background

Parallel Computing

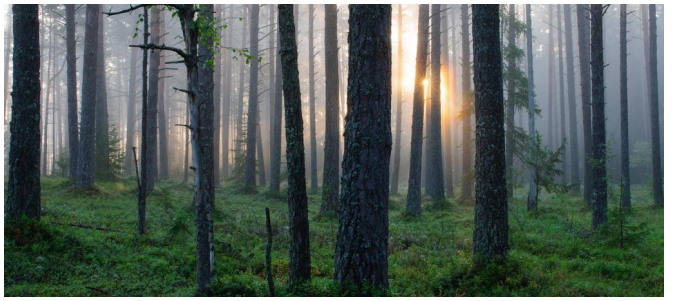


- Let’s start with a cliché
  - “They cannot see the forest for the trees”

- Can we learn Parallel Programming without first knowing computer architecture?

TREES	FOREST
Cores	Parallelism, Locality
HW threads	Parallelism, Locality
Vectors	Parallelism, Locality
Offload	Parallelism, Locality
Heterogeneity	Parallelism, Locality
Cloud	Parallelism, Locality
Caches	Parallelism, Locality
NUMA	Parallelism, Locality

Learn the Forest

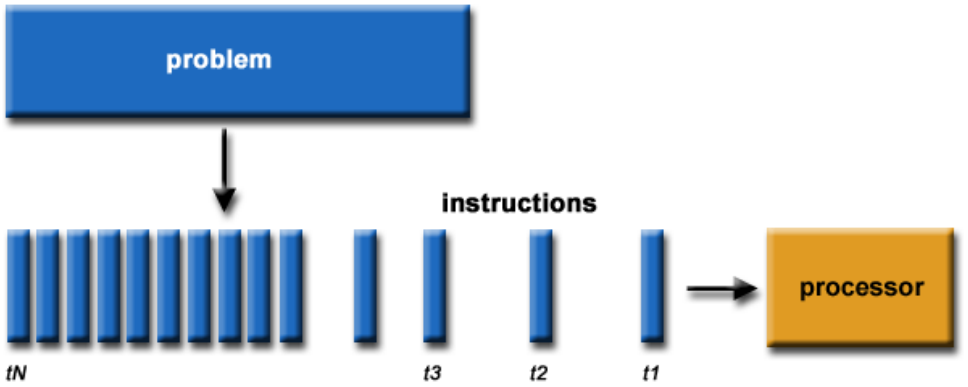


- Increase exposing parallelism
- Increase locality of reference

- **Why?**
  - Because it’s programming that addresses the universal needs of computers today and in the future future.

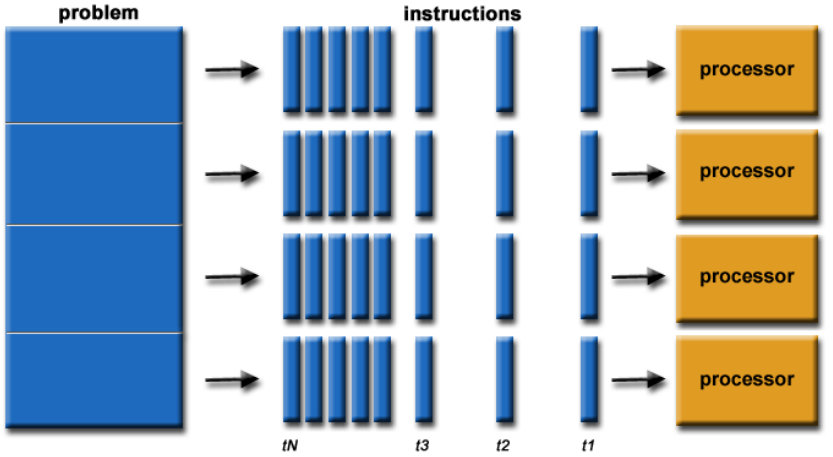
Serial Computing

- **Traditionally** → software has been written for serial computation
  - A problem is broken into a discrete series of instructions
  - Instructions are executed sequentially one after another
  - Executed on a single processor
  - Only one instruction may execute at any moment in time



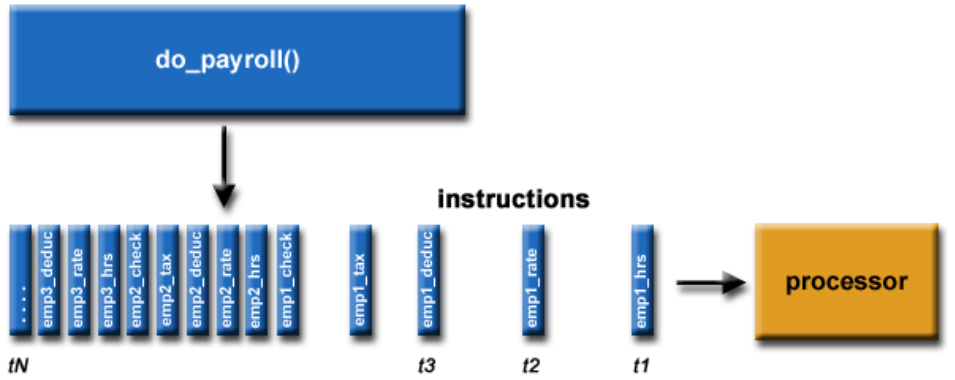
Parallel Computing

- In the simplest sense, parallel computing is the **simultaneous** use of multiple compute resources to solve a computational problem
  - A problem is broken into **discrete parts** that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different processors
  - An overall control/coordination mechanism is employed

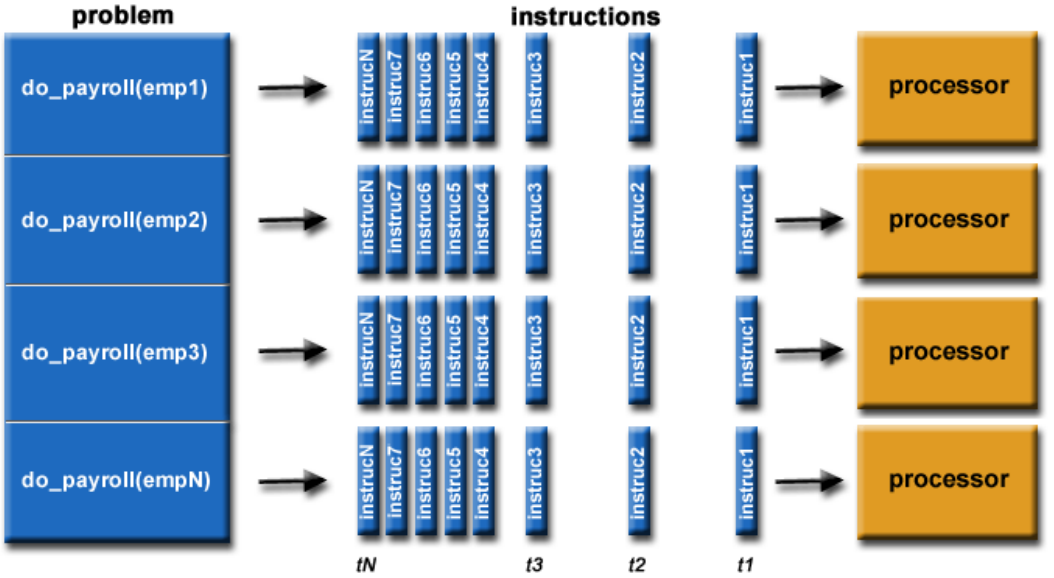


Pay Roll Example

- Serial



- Parallel

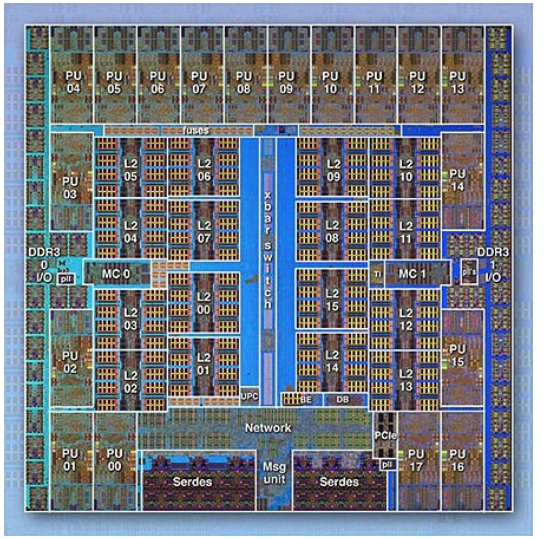


Considerations

- **The computational problem should be able to**
  - Be broken apart into discrete pieces of work that can be solved simultaneously
  - Execute multiple program instructions at any moment in time
  - Be solved in less time with multiple compute resources than with a single compute resource
- **The compute resources are typically**
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network

Parallel Computers

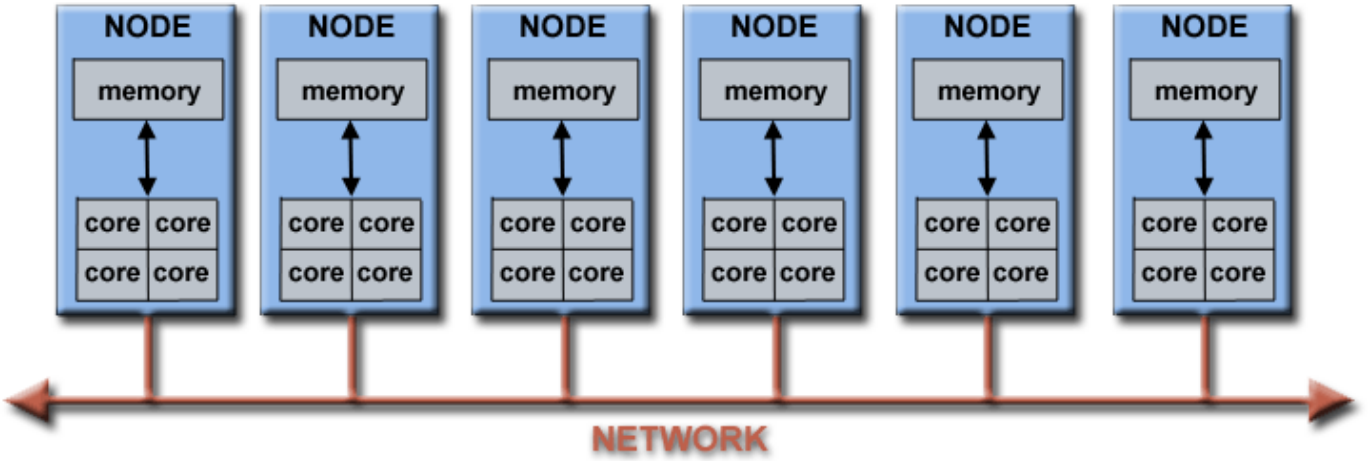
- Virtually all stand-alone computers today are parallel from a hardware perspective
  - Multiple functional units
    - L1 cache, L2 cache, branch, prefetch, decode, floating-point, graphics processing (GPU), integer, etc
  - Multiple execution units/cores
  - Multiple hardware threads



IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

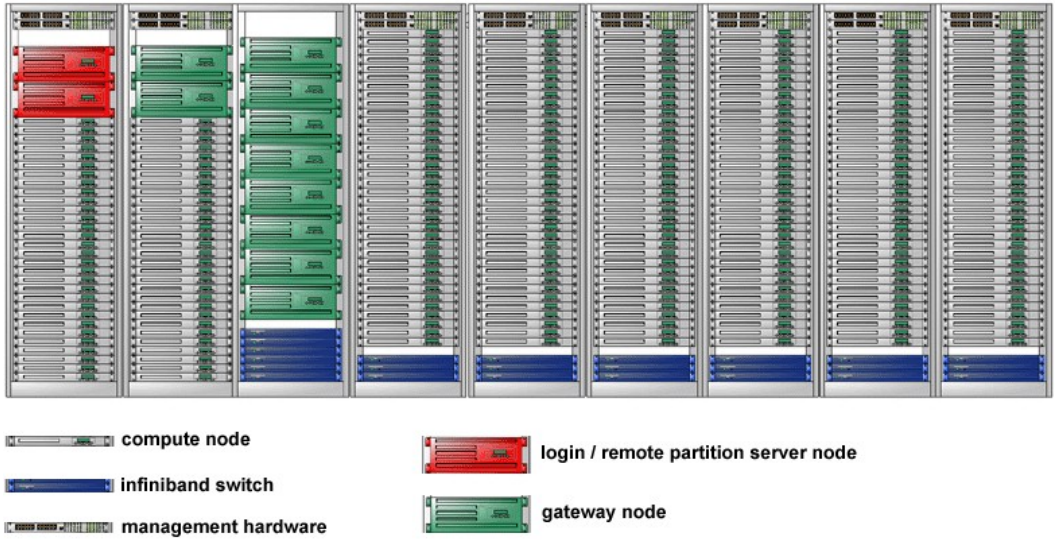
Networking

- Networks connect multiple stand-alone computers (nodes)
  - Allowing to make larger parallel computer clusters



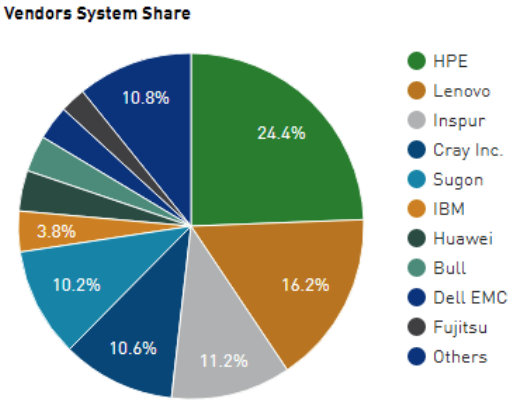
A Parallel Computer Cluster

- A typical parallel computer cluster
  - Each compute node is a multi-processor parallel computer in itself
  - Multiple compute nodes are networked together
    - An Infiniband network, Ethernet network, optical fibers , etc
  - Special purpose nodes, also multi-processor
    - Used for other purposes



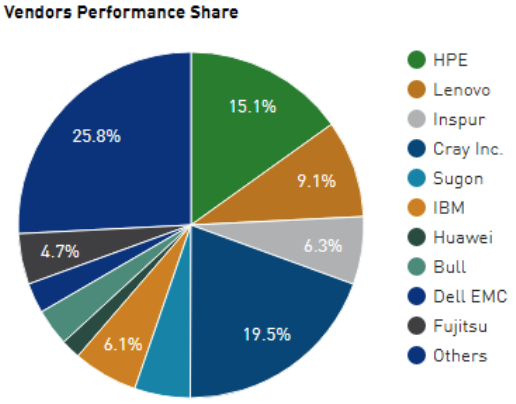
Cluster Vendors

- The majority of the world's large parallel computers (supercomputers)
  - Clusters of hardware produced by a handful of (mostly) well known vendors



Vendors	Count	System Share (%)
HPE	122	24.4
Lenovo	81	16.2
Inspur	56	11.2
Cray Inc.	53	10.6
Sugon	51	10.2
IBM	19	3.8
Huawei	19	3.8
Bull	17	3.4
Dell EMC	16	3.2
Fujitsu	12	2.4
Penguin Computing	10	2
NUDT	4	0.8
PEZY Computing / Exascaler Inc.	4	0.8
NEC	4	0.8
Atipa	3	0.6
Lenovo/IBM	3	0.6
Nvidia	2	0.4
Dell EMC / IBM-GBS	2	0.4
T-Platforms	2	0.4
IBM/Lenovo	2	0.4
Self-made	1	0.2
T-Platforms, Intel, Dell	1	0.2
ClusterVision	1	0.2
Supermicro	1	0.2
ExaScaler	1	0.2
E4 Computer Engineering S.p.A.	1	0.2
RSC Group	1	0.2

Source: top500.org



Why Parallel Computing

- **Real World is Massively Parallel**
  - In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence
  - Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena
  - For example, imagine modeling these serially

Rush Hour Traffic


Plate Tectonics


Weather

Why Parallel Computing

- **Real World is Massively Parallel**
  - In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence
  - Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena
  - For example, imagine modeling these serially

Galaxy Formation | Planetary Movments | Climate Change

Why Parallel Computing

- **Real World is Massively Parallel**
  - In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence
  - Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena
  - For example, imagine modeling these serially

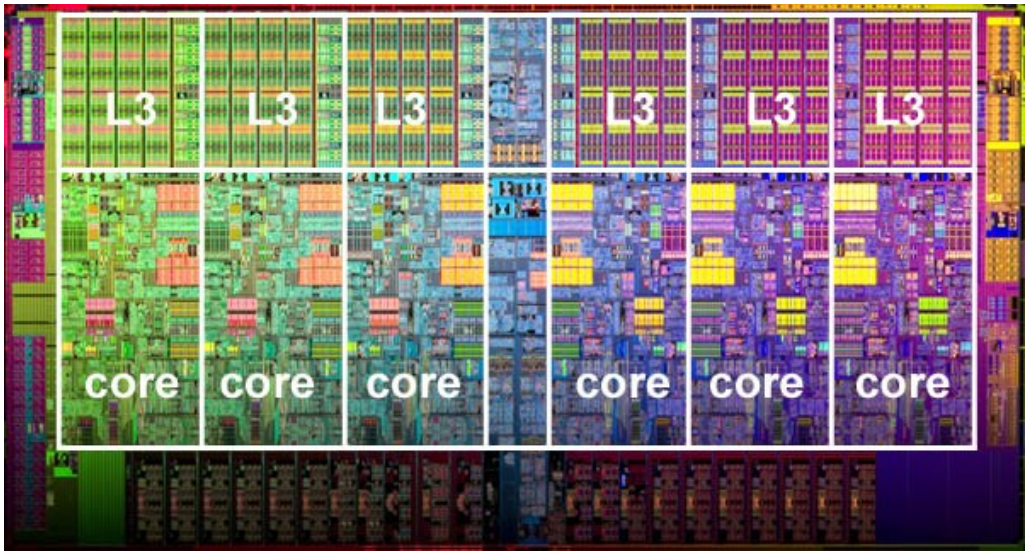
Auto Assembly | Jet Construction | Drive-thru Lunch

Why Parallel Computing

- **To save time and money**
  - In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings
  - Parallel computers can be built from cheap, commodity components

Why Parallel Computing

- **To make better use of underlying parallel hardware**
  - Modern computers, even laptops, are parallel in architecture with multiple processors/cores
  - Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc
  - In most cases, serial programs run on modern computers **waste** potential computing power



Intel Xeon processor

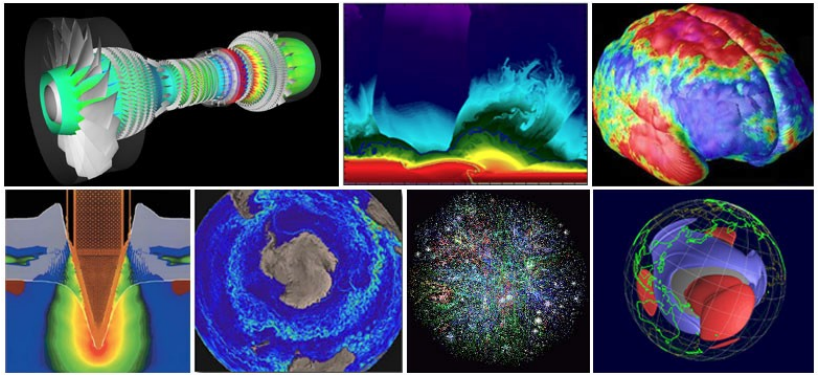
6 cores and 6 L3 cache units

Why Parallel Computing

- 50%/year performance increase of microprocessors from 1986 to 2002
- After 2002 → performance increase slowed down
  - 20%/year
  - Dramatic deceleration
  - In 10-year period
    - 60x faster with 50% and 6x faster with 20%
- **Manufacturers rapid performance increase, but**
  - Not anymore in monolitic processors
  - But in the direction of parallelism → multiple processors on a single integrated circuit
    - Change for software design
      - Serial programs → parallel programs

Parallel Computing Projects

- **Science and Engineering**
  - Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering
    - Atmosphere, Earth, Environment
    - Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
    - Bioscience, Biotechnology, Genetics
    - Chemistry, Molecular Sciences
    - Geology, Seismology
    - Mechanical Engineering - from prosthetics to spacecraft
    - Electrical Engineering, Circuit Design, Microelectronics
    - Computer Science, Mathematics
    - Defense, Weapons



- Mandelbrot

Why the Need for Ever-Increasing Performance?

- **Growing complexity of problems**
  - As computational power increases
- For instance
  - Climate modeling
  - Protein folding
  - Drug discovery
  - Energy research
  - Data analysis

Why Parallel Computing

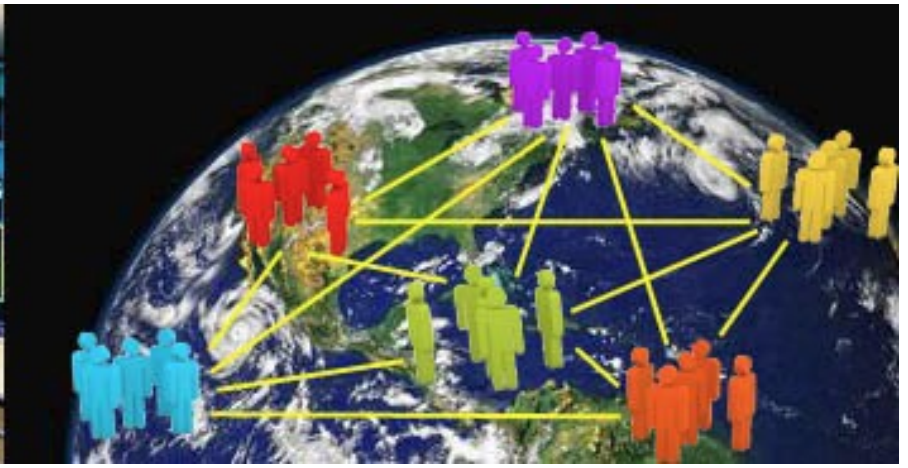
- **To solve larger, more complex, problems**
  - Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer
    - Especially given limited computer memory
  - Example
    - Grand Challenge Problems
      - Requiring PetaFLOPS and PetaBytes of computing resources
  - Web search engines/databases
    - Processing millions of transactions every second

Why Building Parallel Systems?

- **Integrate circuits reaching their limits**
  - Transistor density
  - However → need for computer power increase
- **Exploring different paradigms**
  - Multicore processors

Why Parallel Computing

- **To provide concurrency**
  - A single compute resource can only do one thing at a time
    - Multiple compute resources can do many things simultaneously
  - Example
    - Collaborative Networks provide a global venue where people from around the world can meet and conduct work **virtually**

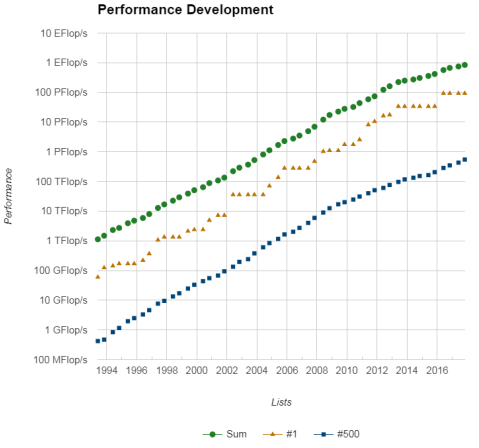


Why Parallel Computing

- **To take advantage of non-local resources**
  - Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient
  - Examples → over 1.7 million contributors globally (May 2018)
    - SETI@home (setiathome.berkeley.edu)
    - Folding@home (folding.stanford.edu)

Trends

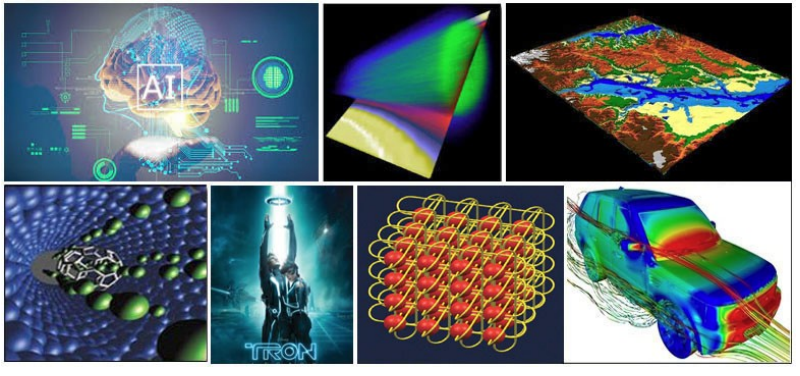
- **During the past 20+ years**
  - Trends indicated
    - Ever faster networks, distributed systems, and multi-processor computer architectures → even at the desktop level
      - Clearly show that parallelism is the future of computing
- In this same time period, there has been a greater than 500,000x increase in supercomputer performance
  - No end currently in sight
- The race is already on for **Exascale** Computing!
  - Exaflop = 10^{18} calculations per second



Source: Top500.org

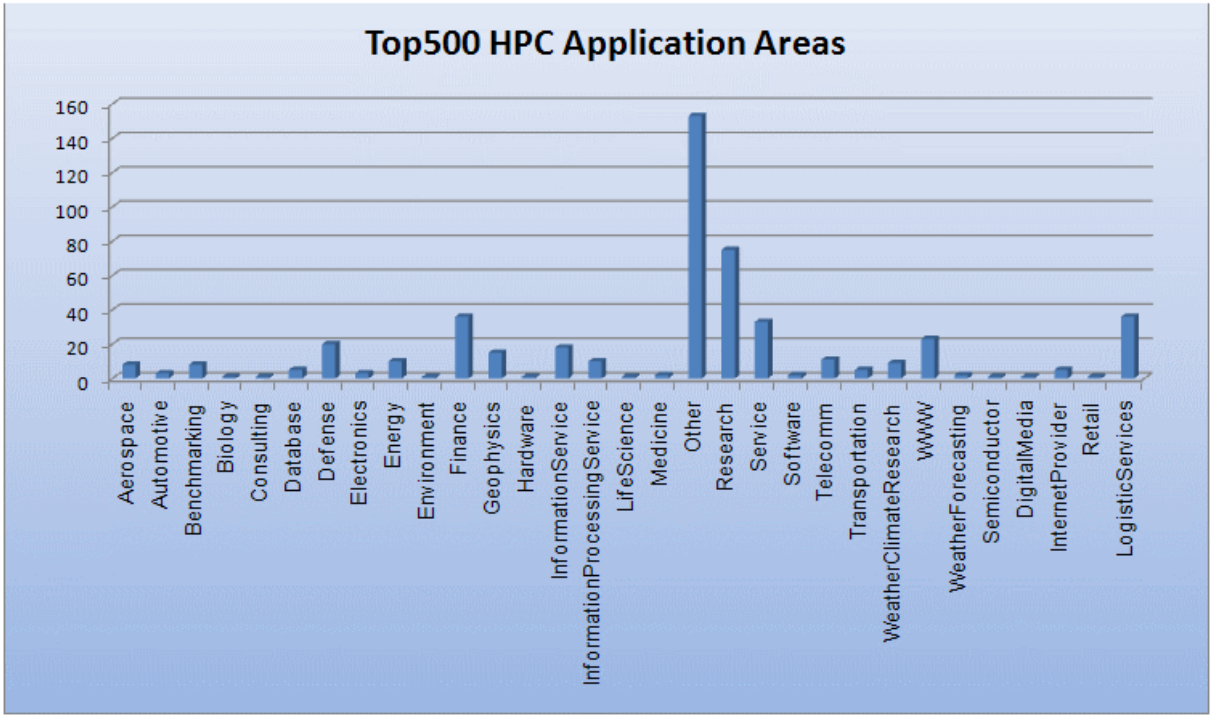
Parallel Computing Projects

- **Industrial and Commercial**
  - Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways
    - Big Data, databases, data mining
    - Web search engines, web based business services
    - Medical imaging and diagnosis
    - Financial and economic modeling
    - Advanced graphics and virtual reality, particularly in the entertainment industry
    - Networked video and multi-media technologies
    - Artificial Intelligence (AI)
    - Pharmaceutical design
    - Oil exploration



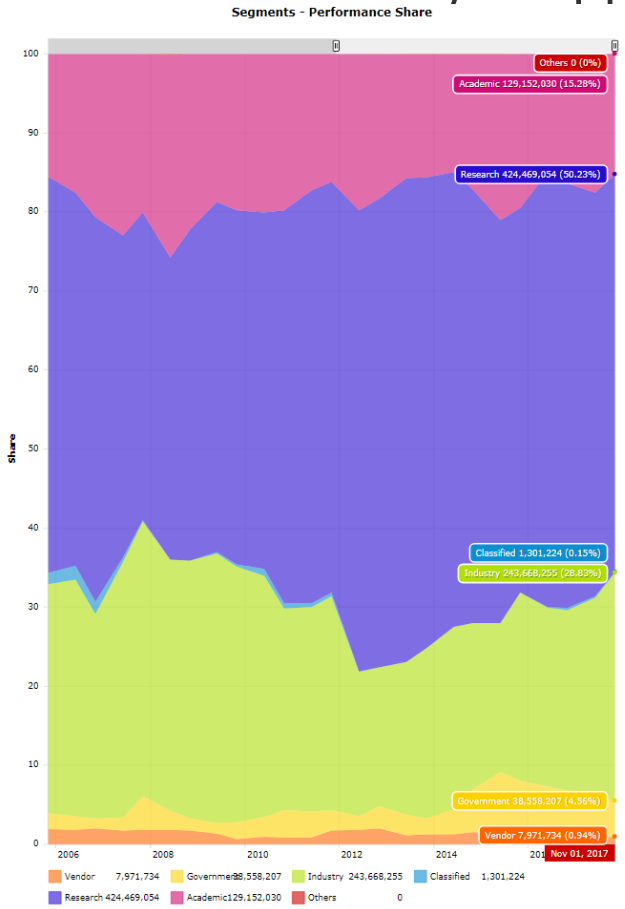
Global Applications

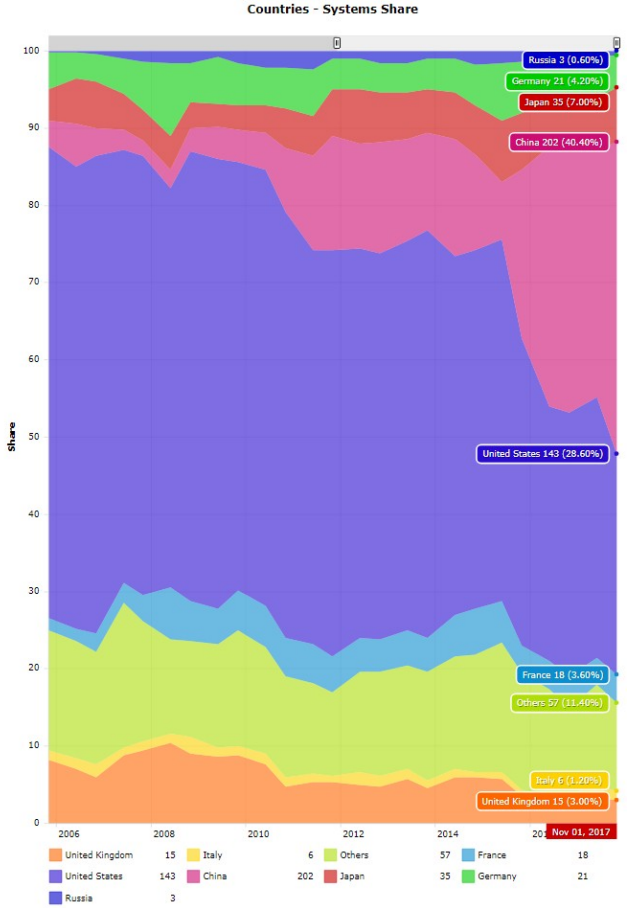
- Parallel computing is now being used extensively around the world
  - In a wide variety of applications (Source: Top500.org)



Global Applications

- Parallel computing is now being used extensively around the world
  - In a wide variety of applications (Source: Top500.org)





Why the Need to Write Parallel Programs?

- Running multiple instances of the same program
  - Independent processes → high parallelism
  - However → **not the common desired case**

Why the Need to Write Parallel Programs?

- Running a program faster
  - **Converting it from serial to parallel**
    - Multiple cores
  - **Using translators**
    - Automatic conversion
      - Really bad efficiency
    - Existing converters written in C and C++

Translation is not Simple and Efficient

- Translating serial programs into parallel programs is not straightforward
- For instance
  - Converting multiplication of n x n matrices
    - Originally → sequence of dot products
    - Parallel → sequence of parallel dot products
      - Very slow on many systems → Why?
      - Efficient parallelization
        - Not finding efficient parallelization of each step
        - But stepping back and devising a totally new algorithm

Example

- Calculate ***n*** values and add them together
  - `sum = 0;`
  - `for (i = 0; i < n; i++) {`
    - `x = Compute_next_value(. . .);`
    - `sum += x;`
  - `}`

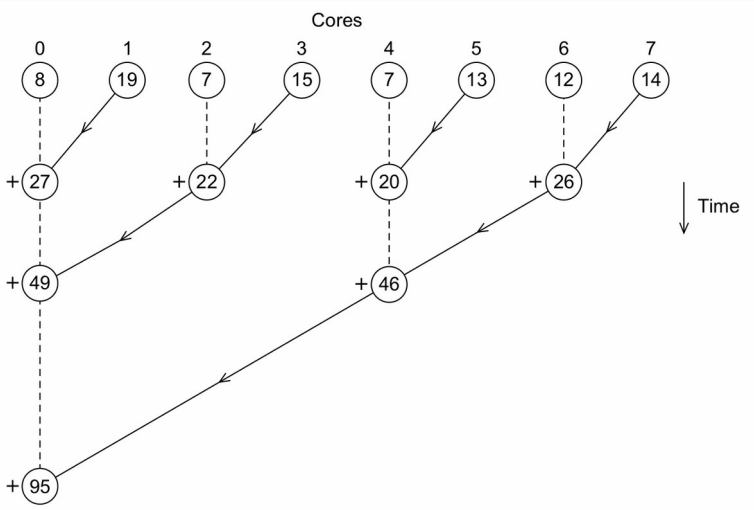
- Suppose there are ***p*** cores (***p*** << ***n***)
  - `my_sum = 0;`
  - `my_first_i = . . . ;`
  - `my_last_i = . . . ;`
  - `for (my_i = my_first_i; my_i < my_last i; my_i++) {`
    - `my_x = Compute_next_value(. . .);`
    - `my_sum += my_x;`
  - `}`
  - Where **my_** means local variables in a core
    - There will be ***p*** **my_sum**'s

Example

- Master core can sum up all these local sums
  - `if ( I'm the master core) {`
    - `sum = my_x;`
    - `for each core other than myself {`
      - `receive value from core;`
      - `sum += value;`
    - `}`
  - `} else {`
    - `send my x to the master;`
  - `}`
- Not the best parallelization
  - Large number of cores → high **p**
    - Levels of grouping

Example

- Master core can sum up all these local sums
  - `if ( I'm the master core) {`
    - `sum = my_x;`
    - `for each core other than myself {`
      - `receive value from core;`
      - `sum += value;`
    - `}`
  - `} else {`
    - `send my x to the master;`
  - `}`
- Not the best parallelization
  - Large number of cores → high **p**
    - Levels of grouping



Example

- Translation
  - Unlikely that it would produce an efficient solution
    - As the 2nd
- Higher the complexity
  - Less likely to have a precoded parallelization
- Thus
  - Parallel programs must be written and not translated

How to Write Parallel Programs?

- Many possible ways
  - Share the same basic concept → **partitioning**
  - Two approaches
    - **Task-parallelism**
      - Partitioning problem in several tasks
        - Assigned to many cores
    - **Data-parallelism**
      - Splitting data in many cores
        - Cores execute similar operations on data portions

Example 1

- A course with 120 students
  - 4 TAs: A, B, C, and D
  - Final exam with 5 questions
  - Correction options
    - Prof + TAs → grade 120 exams each
      - Just a question for each
    - 120 exams split in 5 piles
      - Each corrector has 24 exams → 5 questions each

Example 1

- A course with 120 students
  - 4 TAs: A, B, C, and D
  - Final exam with 5 questions
  - Correction options
    - Prof + TAs → grade 120 exams each
      - Just a question for each
    - 120 exams split in 5 piles
      - Each corrector has 24 exams → 5 questions each
  - Analogy
    - Prof and TAs = cores
    - Grading a question = task (over multiple answers)
    - Student answer = data
  - 1st correction layout → task-parallelism
    - Each question (task) for each processor (corrector)
  - 2nd correction layout → data-parallelism
    - Pile of student exams (data) for each processor (corrector)
      - All correctors do the same 5 tasks (questions)

Example 2

- Previous example of sums → global sum
  - 1st part → Data-parallelism
    - Data = values calculated by Compute_next_value
    - All cores fairly do the same operations
  - 2nd part → task-parallelism
    - 2 tasks: receiving and adding partial sums (master core)

Parallelism Complexity

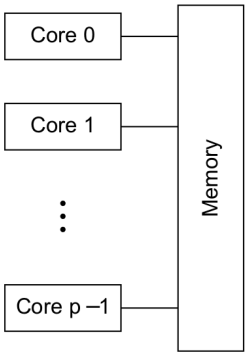
- Independent operations/tasks
  - Parallel design → as simple as sequential
- Coordinated work → higher complexity
  - 2nd part of global sum → use of a simple tree structure
    - Implementing the code is rather complex
- Coordination
  - Communication → global sums, for ex
  - Load balancing → same computing load (efficiency)
  - Synchronization → waiting before moving on (dependency)
    - Master
      - `if ( I 'm the master core)`
        - `for (my i = 0; my i < n; my i++)`
          - `scanf("%lf", &x[my i]);`
    - Slaves
      - `for ( my i = my first i ; my i < my last i ; my i ++)`
        - `my sum += x [ my i ];`
    - Synchronization
      - `Synchronize cores ();`

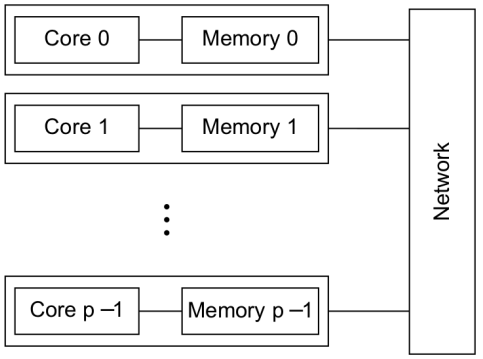
Parallel Constructs

- Most powerful parallel constructs
  - Explicit
    - Extensions to languages (C and C++)
    - Explicit instructions
      - core 0 runs task 0
      - Synchronize cores
      - …
- Higher level languages
  - More transparent (“simpler”)
  - However → sacrifice performance

What to Be Done in This Course

- When it comes to design
  - Building programs through explicit parallelism
- C language, for ex
  - Message-Passing Interface → MPI
  - POSIX threads → Pthreads / C++11 threads
  - OpenMP
- Different languages/extensions → different parallel systems
  - Two major parallel systems
    - Shared-memory
    - Distributed-memory



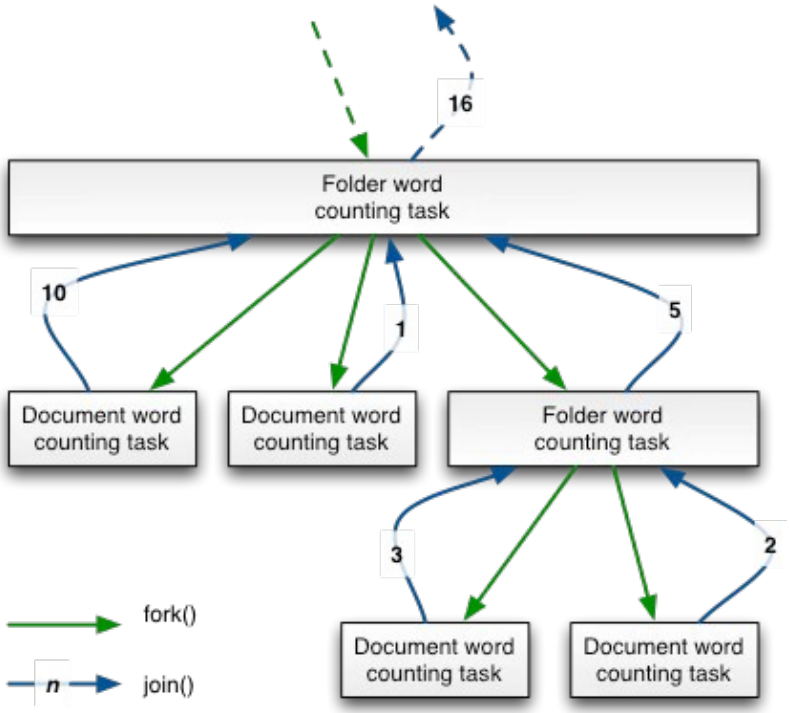


Concurrent, Parallel, and Distributed

- No complete agreement on their distinction
  - However
    - **Concurrent computing**
      - Multiple tasks might be in progress at any instant
    - **Parallel computing**
      - A program with multiple tasks closely cooperating to solve a problem
    - **Distributed computing**
      - A program cooperating with other programs to solve a problem
  - Blurry distinction → parallel vs distributed
    - Coupling → distributed is more loosely coupled
    - Memory → shared-memory programs are parallel

Example of Parallel Program

- Searching words in file tree



- java WordCounter folder house 8
- Mandelbrot

Class Recap

- Introduction
  - Parallel computing
  - Parallel systems
- Importance
- Overview
  - Complexity
  - Task and data parallelism

Next Class

- Von Neumann Architecture
- Flynn's Taxonomy
  - SISD
  - SIMD
  - MISD
  - MIMD
- General terminology and assumptions