Parallel Computing: Theory - Hardware - Software with special focus on Multi-core Programming

Erwin Laure
Director PDC-HPC, Guest Professor KTH

Jarmo Rantakokko
Senior Lecturer IT/UU

Erik Hagersten
Professor IT/UU

David Black-Shaffer
Lecturer IT/UU

1

About the course

- The overall goal of the course is to give basic knowledge of the theory, hardware, and software approaches to parallel computing. Especially, hardware and software challenges and the interactions between them, as well as exposure to research challenges in this field will be emphasized.
- After the course you will be able to:
  - Understand the properties of different parallel architectures
  - Reason about the performance of a system
  - Assess the potential and limitations of parallel processing
  - Chose between different parallelization techniques
  - Write parallel programs on multi-core machines

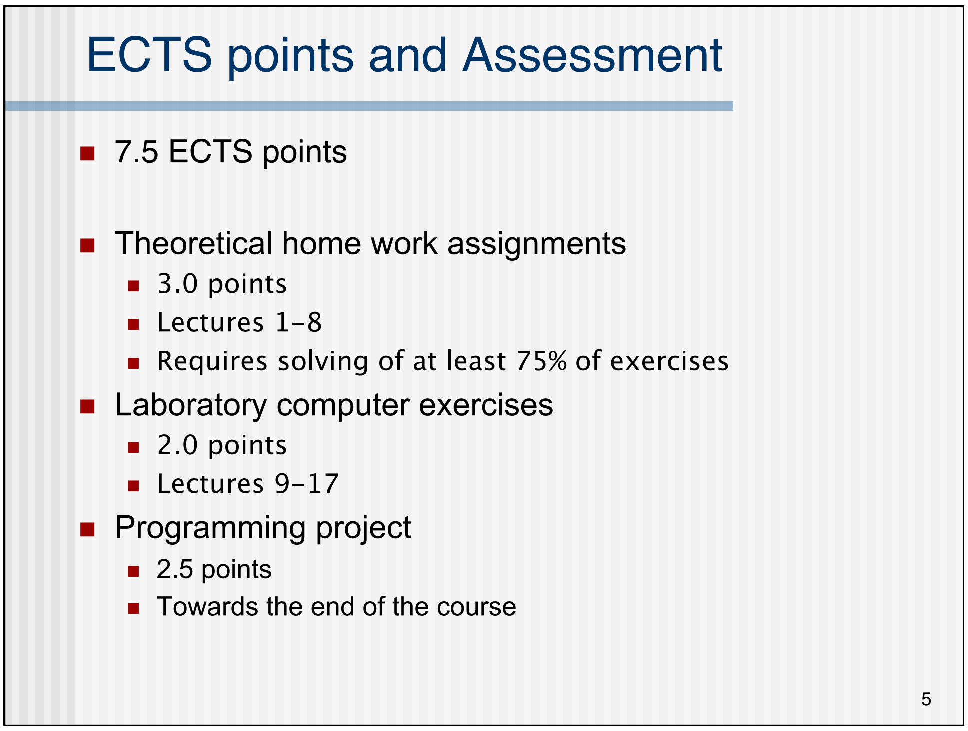
2

Content (preliminary)

- L1: Introduction
- L2: Understanding parallel computers
- L3: Reasoning about performance
- L4&5: Different kinds of parallelism
- L6&7: The impact of data
- L8: Approaches to program parallel machines
- L9: From single core to many core
- L10&11: A deeper look at memory hierarchies
- L12: Reserve
- L13-16: Programming with pthreads and OpenMP
- L17: Future trends: Cell BA, FPGA, GPU, OpenCL

Outline

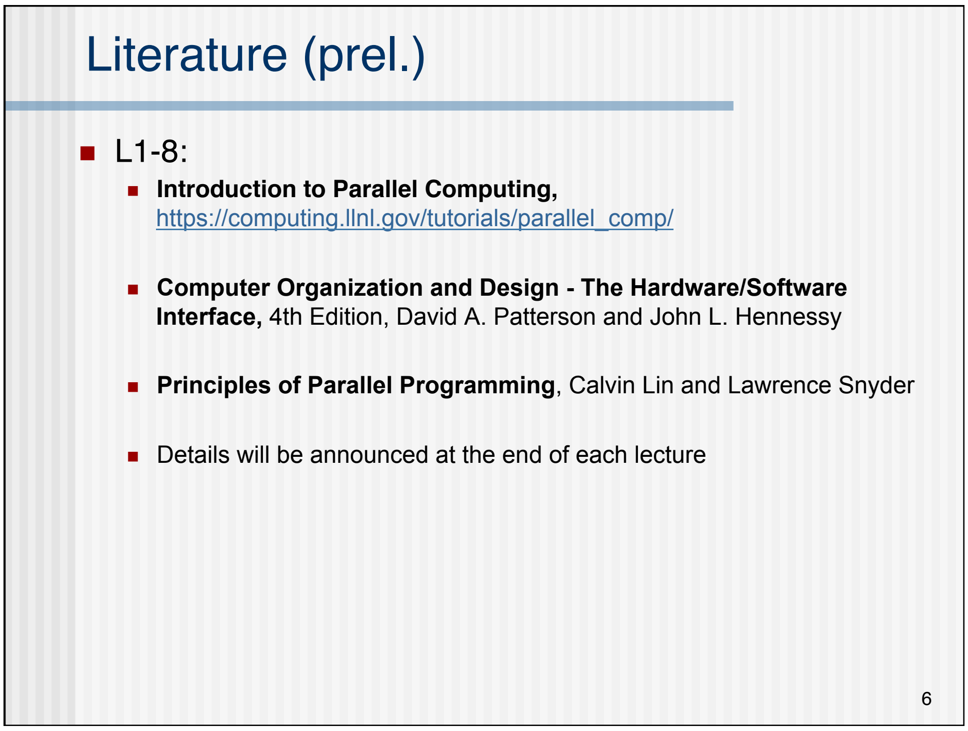
- Two lectures (1.5 hours each) per day
- Lecture 1-8 at KTH
  - Jan. 16 - Feb. 6
  - Background on parallel computing, theory
- Lecture 9-17 at UU
  - Feb. 13 – March 26
  - Focus on multicore, hands-on and project



ECTS points and Assessment

- 7.5 ECTS points
- Theoretical home work assignments
  - 3.0 points
  - Lectures 1–8
  - Requires solving of at least 75% of exercises
- Laboratory computer exercises
  - 2.0 points
  - Lectures 9–17
- Programming project
  - 2.5 points
  - Towards the end of the course

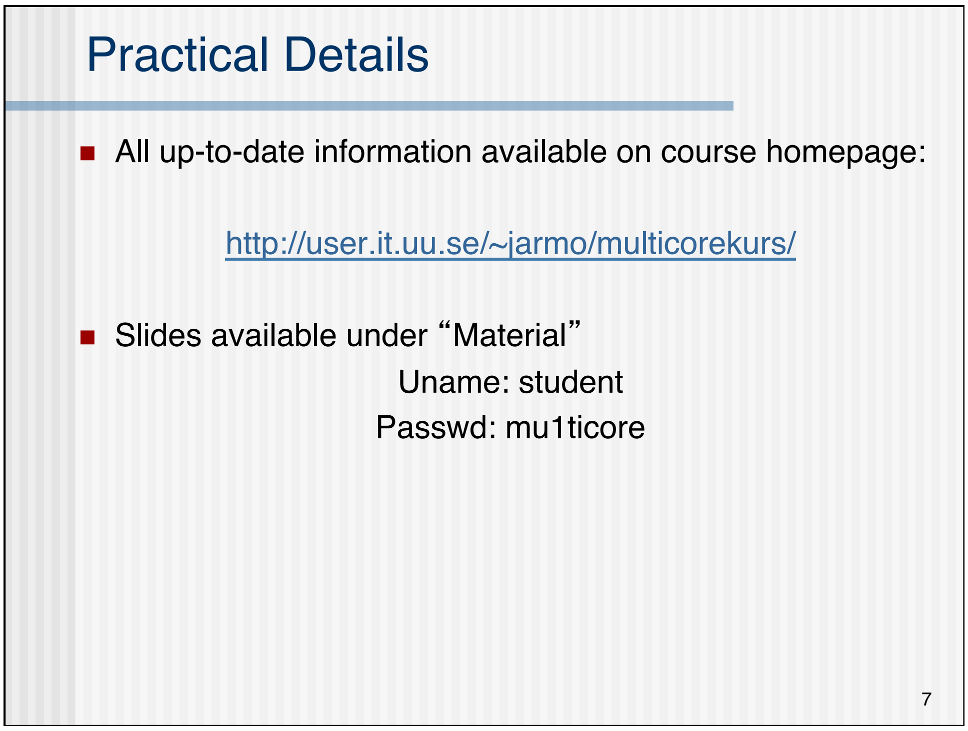
5



Literature (prel.)

- L1-8:
  - **Introduction to Parallel Computing,** https://computing.llnl.gov/tutorials/parallel_comp/
  - **Computer Organization and Design - The Hardware/Software Interface,** 4th Edition, David A. Patterson and John L. Hennessy
  - **Principles of Parallel Programming**, Calvin Lin and Lawrence Snyder
  - Details will be announced at the end of each lecture

6



Practical Details

- All up-to-date information available on course homepage:

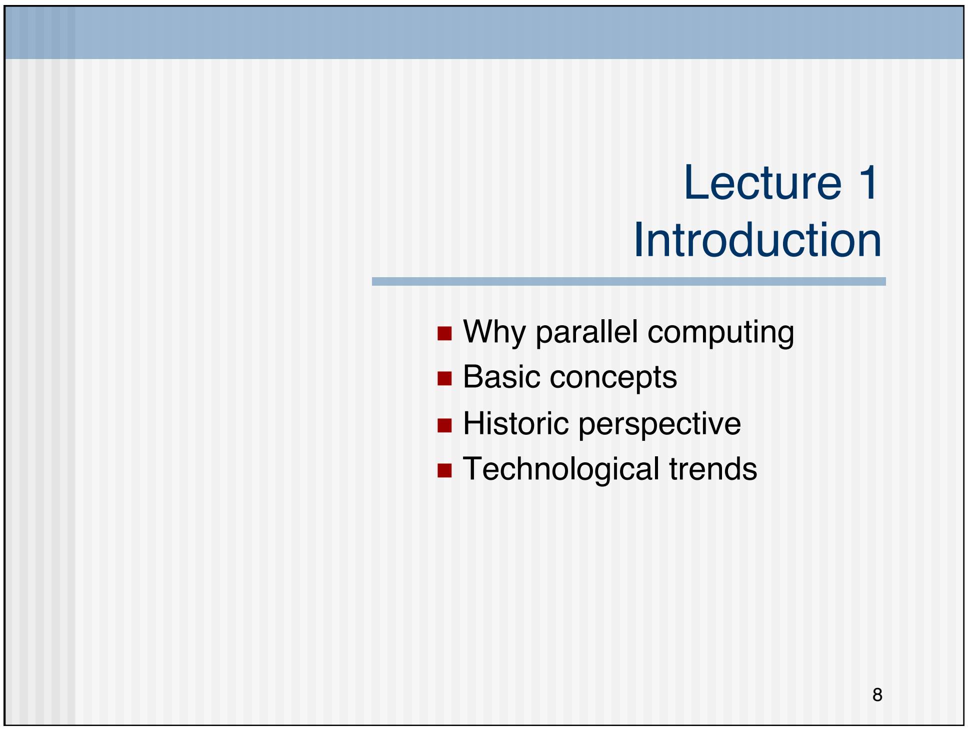
  http://user.it.uu.se/~jarmo/multicorekurs/

- Slides available under “Material”

  Uname: student

  Passwd: mu1ticore

7



Lecture 1 Introduction

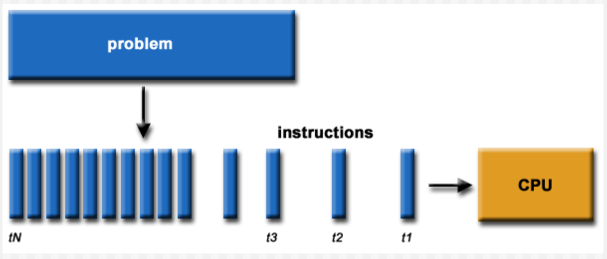
- Why parallel computing
- Basic concepts
- Historic perspective
- Technological trends

8

Why Parallel Computing

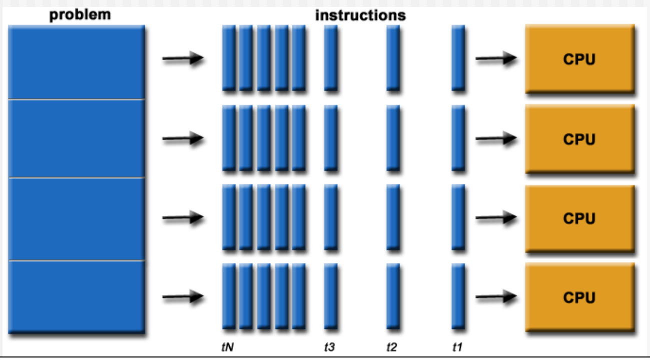
What is Parallel Computing

- Traditionally, software has been written for ***serial*** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - 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



11

Parallelism on different levels

- CPU
  - Instruction level parallelism, pipelining
  - Vector unit
  - Multiple cores
    - Multiple threads or processes
- Computer
  - Multiple CPUs
  - Co-processors (GPUs, FPGAs, …)
- Network
  - Tightly integrated network of computers (supercomputer)
  - Loosely integrated network of computers (distributed computing)

12

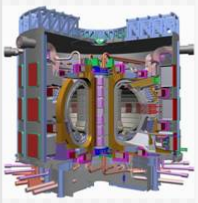
Solve a given problem in less time

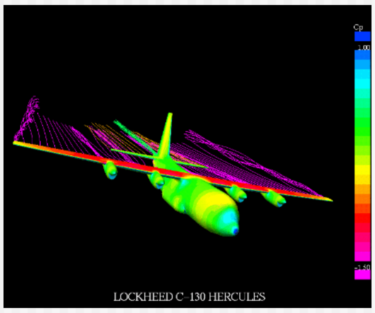
- Increase the productivity, save money

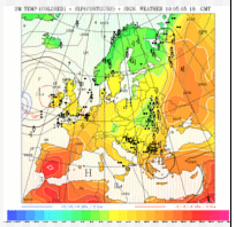
13

Solve a larger problem

- Many problems require more resources than available on a single computer (particularly memory)
  - “Grand Challenge" (en.wikipedia.org/wiki/Grand_Challenge) problems requiring PetaFLOPS and PetaBytes of computing resources.
  - Web search engines/databases processing millions of transactions per second







14

Provide concurrency

- A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously. For example, the Access Grid (www.accessgrid.org) provides a global collaboration network where people from around the world can meet and conduct work "virtually".

15

Use of external resources

- Using compute resources on a wide area network, or even the Internet when local compute resources are scarce. For example:
  - SETI@home (setiathome.berkeley.edu) uses over 2,500,000 computers (about 280.000 active for a compute power over 769 TeraFLOPS (as of November 14, 2009)
  - Folding@home (folding.stanford.edu) uses over 400,000 computers for a compute power of over 5 PetaFLOPS (as of April 9, 2009)

16

Limits to serial computing

- Both physical and practical reasons pose significant constraints to simply building ever faster serial computers:
  - Transmission speeds
    the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
  - Limits to miniaturization
    processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
  - Economic limitations
    it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

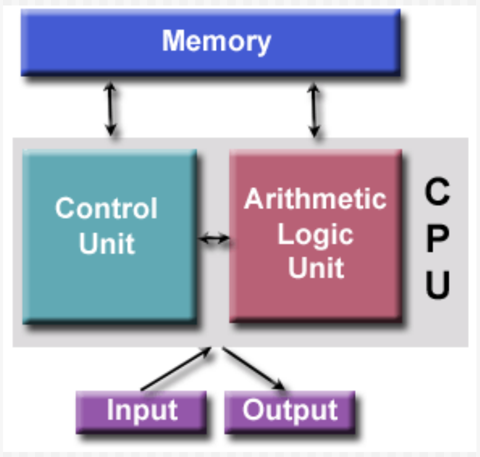
17

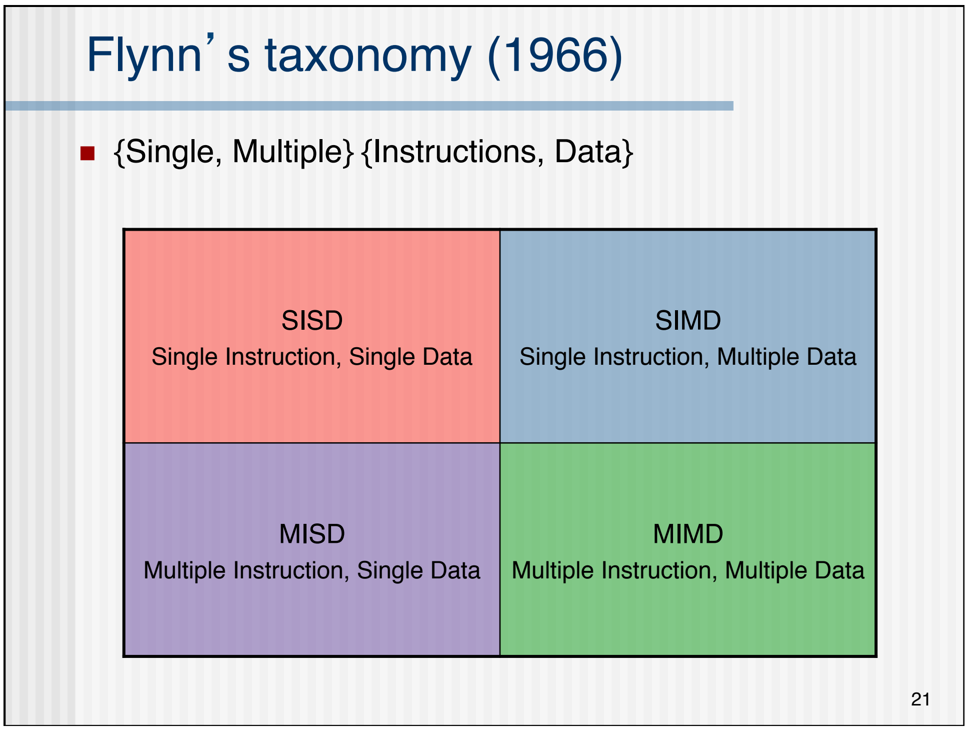
The Universe is Parallel

18

Basic Concepts

Von Neumann Architecture



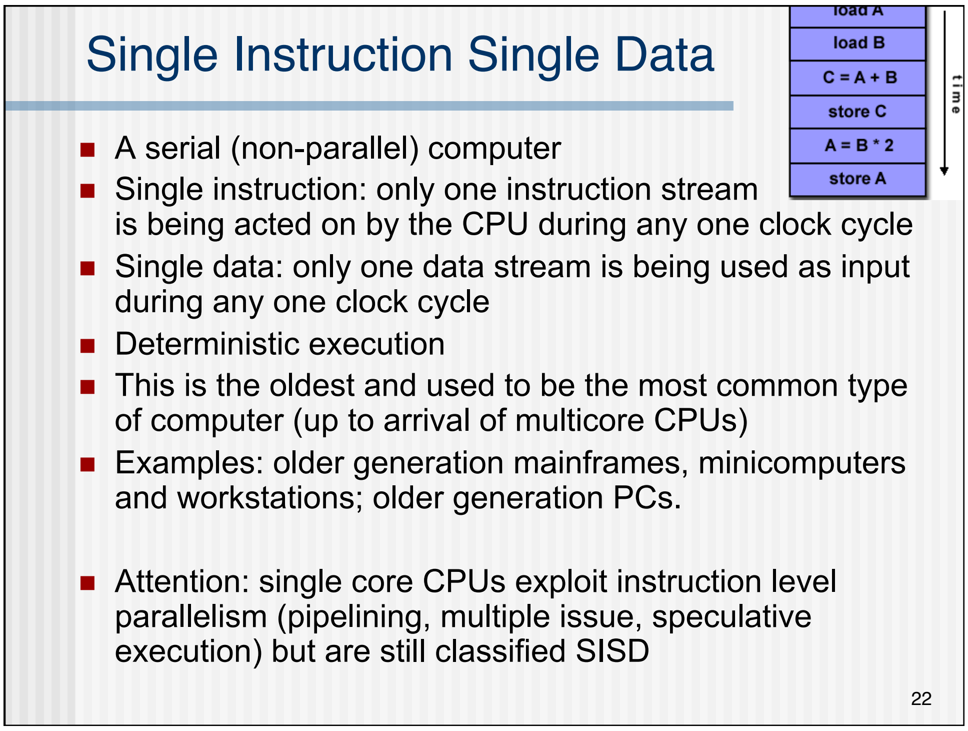


Flynn' s taxonomy (1966)

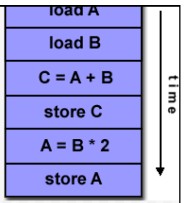
- {Single, Multiple} {Instructions, Data}

SISD Single Instruction, Single Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

21



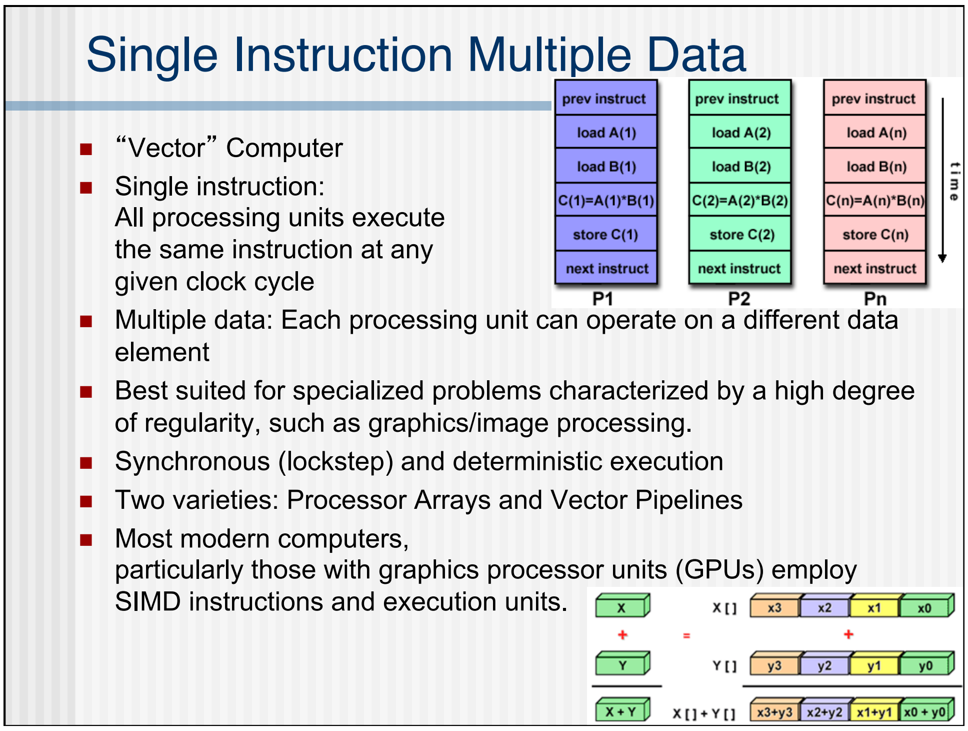
Single Instruction Single Data



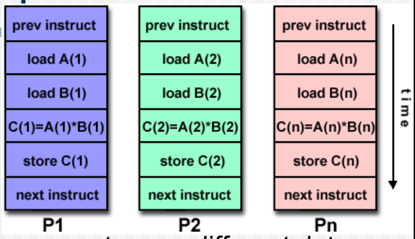
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and used to be the most common type of computer (up to arrival of multicore CPUs)
- Examples: older generation mainframes, minicomputers and workstations; older generation PCs.

- Attention: single core CPUs exploit instruction level parallelism (pipelining, multiple issue, speculative execution) but are still classified SISD

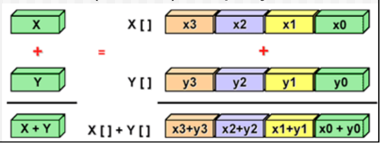
22

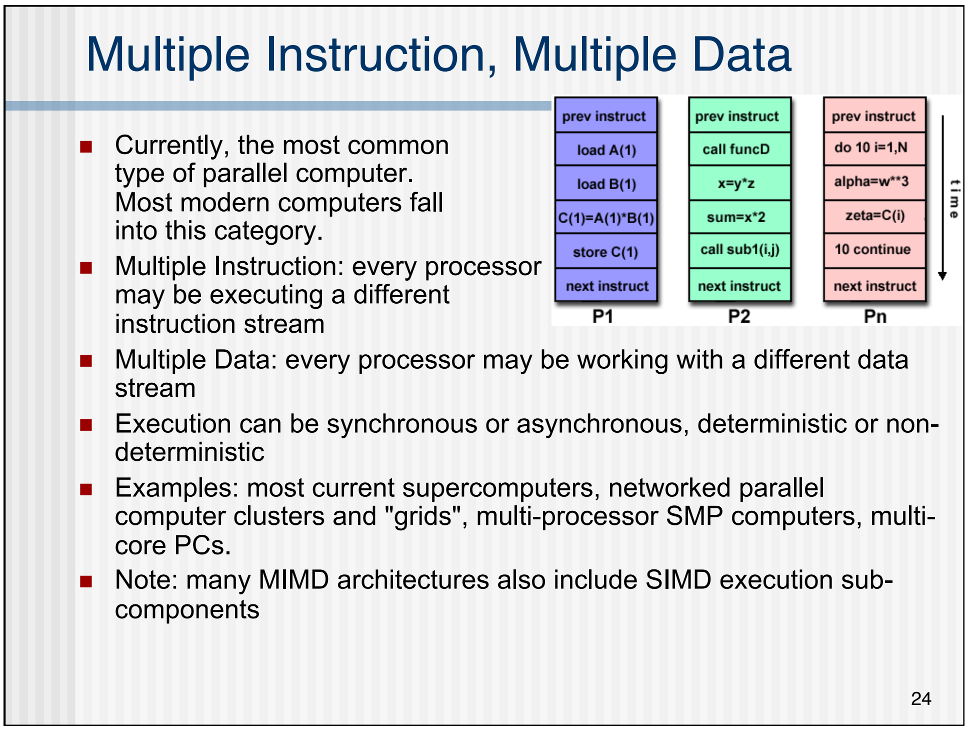


Single Instruction Multiple Data

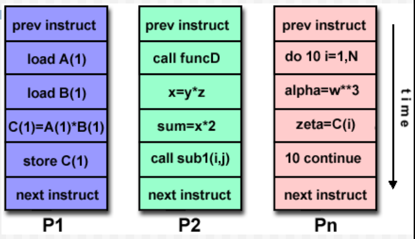


- "Vector" Computer
- Single instruction:
  All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Most modern computers,
  particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.





Multiple Instruction, Multiple Data



- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components

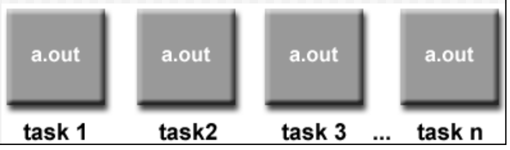
24

Multiple Instruction, Single Data

- No examples exist today
- Potential uses might be:
  - Multiple cryptography algorithms attempting to crack a single coded message
  - Multiple frequency filters operating on a single signal

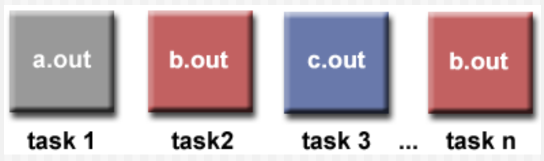
Single Program Multiple Data (SPMD)

- MIMDs are typically programmed following the SPMD model
- A single program is executed by all tasks simultaneously.
- At any moment in time, tasks can be executing the same or different instructions within the same program. All tasks may use different data. (MIMD)
- SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program - perhaps only a portion of it.



Multiple Program Multiple Data (MPMD)

- MPMD applications typically have multiple executable object files (programs). While the application is being run in parallel, each task can be executing the same or different program as other tasks.
- All tasks may use different data
- Workflow applications, multidisciplinary optimization, combination of different models



27

FLOPS

- FLoating Point Operations per Second
- Most commonly used performance indicator for parallel computers
- Typically measured using the Linpack benchmark
- Most useful for scientific applications
- Other benchmarks include SPEC, NAS, stream (memory)

Name	Flops
Yotta	10^{24}
Zetta	10^{21}
Exa	10^{18}
Peta	10^{15}
Tera	10^{12}
Giga	10^{9}
Mega	10^{6}

28

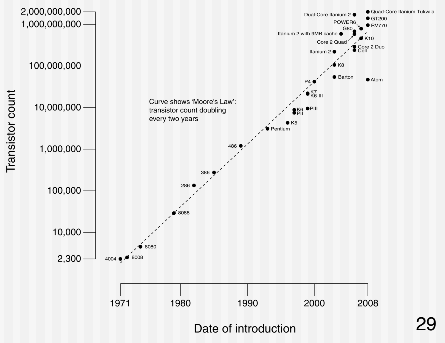
Moore’s Law

- Gordon E. Moore, "Cramming more components onto integrated circuits", *Electronics Magazine* 19 April 1965:

  “*The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.*”

- With later alterations:
  Transistor density doubles every 18 months
- So far this law holds
- It has also been interpreted as doubling performance every 18 months
  - A little inaccurate - see later

CPU Transistor Counts 1971-2008 & Moore’s Law



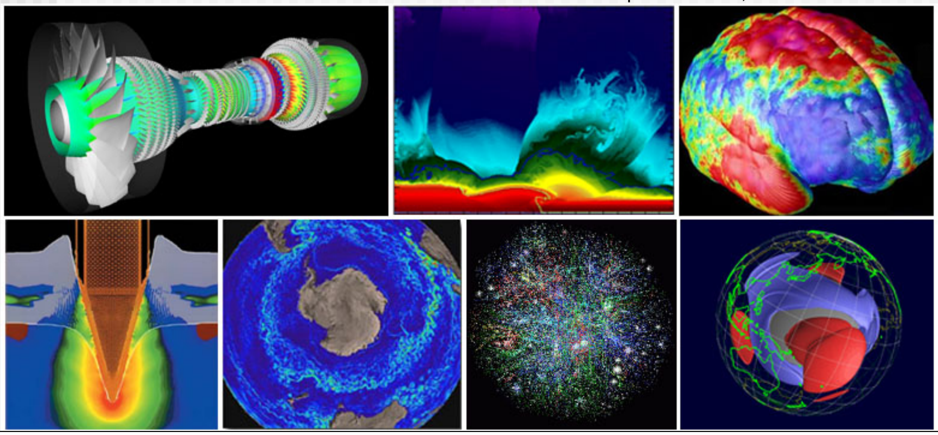
29

Historic Perspective

30

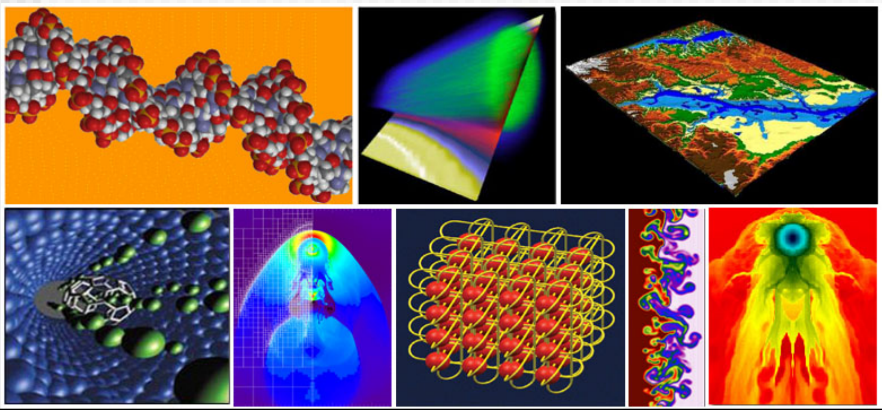
Uses for Parallel Computing

- Historically "the high end of computing”
  - 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



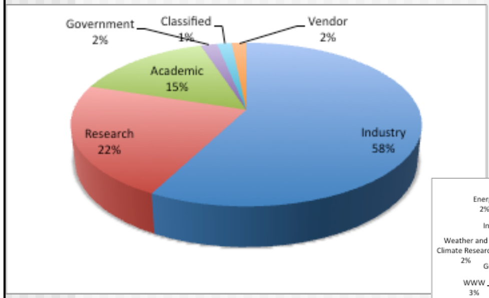
Uses for Parallel Computing 2

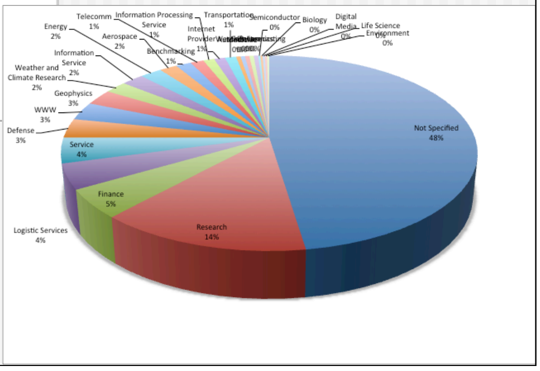
- Today, commercial applications provide an equal or greater driving force; require processing of large amounts of data in sophisticated ways
  - Databases, data mining
  - Oil exploration
  - Web search engines, web based business services
- Medical imaging and diagnosis
- Pharmaceutical design
- Management of national and multi-national corporations
- Financial and economic modeling
- Advanced graphics and virtual reality, particularly in the entertainment industry
- Networked video and multi-media technologies
- Collaborative work environments



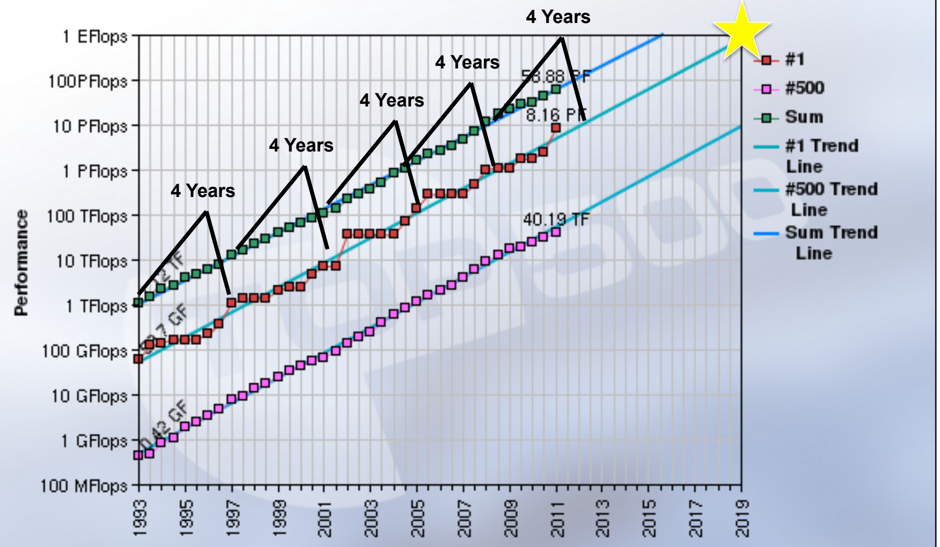
Systems per Segment/Application Area

- Top500 List, November 2011



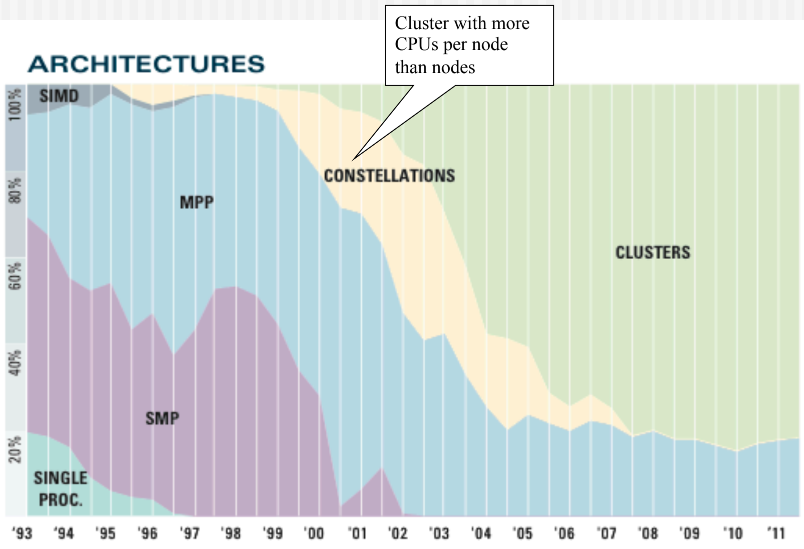


Projected Performance Development



16/06/2011 http://www.top500.org/ 34

Top 500 Architecture Development

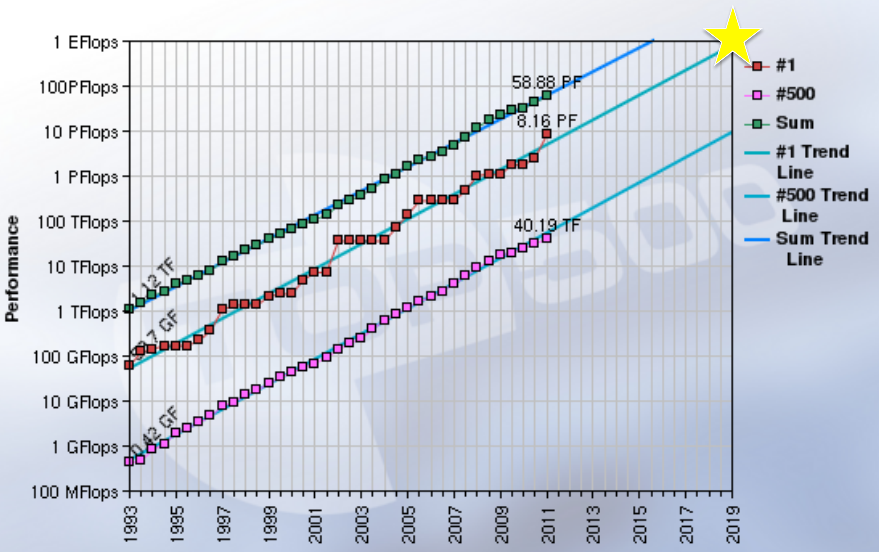


35

Top500 Nr 1: ”K Computer”

- RIKEN, Japan
- Fujitsu SPARC64, 2.0 GHz, Tofu interconnect
- 705.024 cores
- 10.5 PF Linpack (11.28 PF theoretical peak)

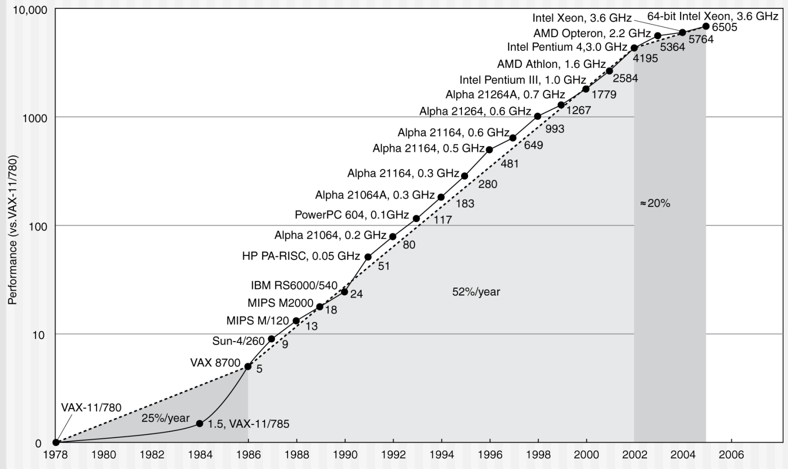
Projected Performance Development



16/06/2011 http://www.top500.org/

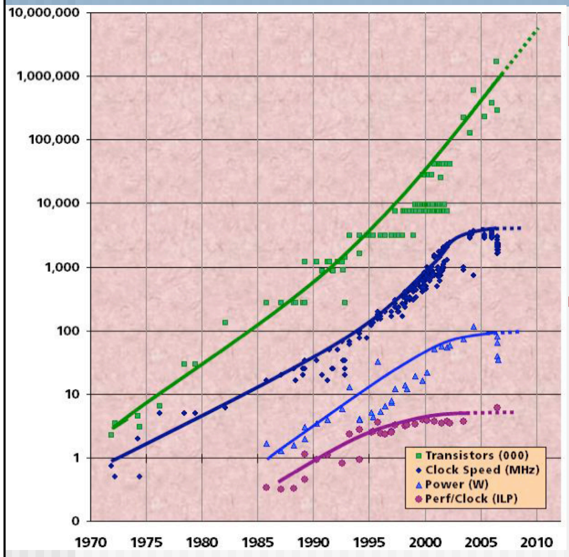
Moore's law revisited

- Doubling of transistor density every 18 months
  - Often paraphrased as doubling of performance every 18 months



38

Reinterpreting Moore's law



- **Moore's law is holding, in the number of transistors**
  - Transistors on an ASIC still doubling every 18 months at constant cost
  - 15 years of *exponential* clock rate growth has ended
- **Moore's Law reinterpreted**
  - Performance improvements are now coming from the increase in the number of cores on a processor (ASIC)
  - #cores per chip doubles every 18 months *instead* of clock

From Herb Sutter<hsutter@microsoft.com>

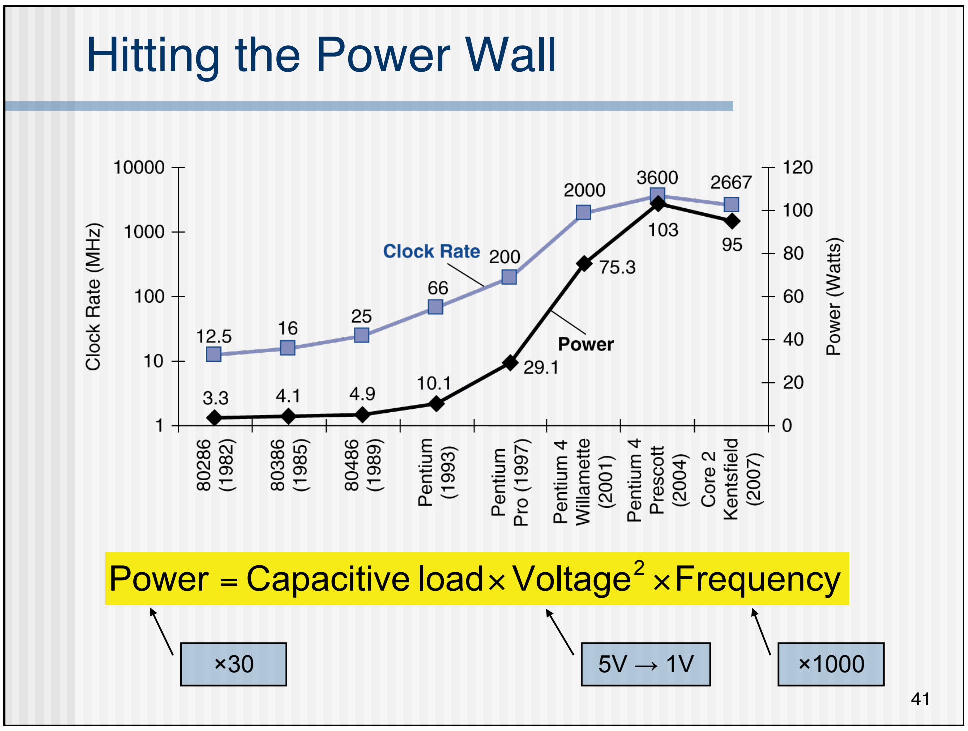
39

Computing Power Consumption

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

- Capacitive load per transistor is a function of both the number of transistors connected to an output and the technology, which determines the capacitance of both wires and transistors
- Frequency switched is a function of the clock rate

40



Hitting the Power Wall

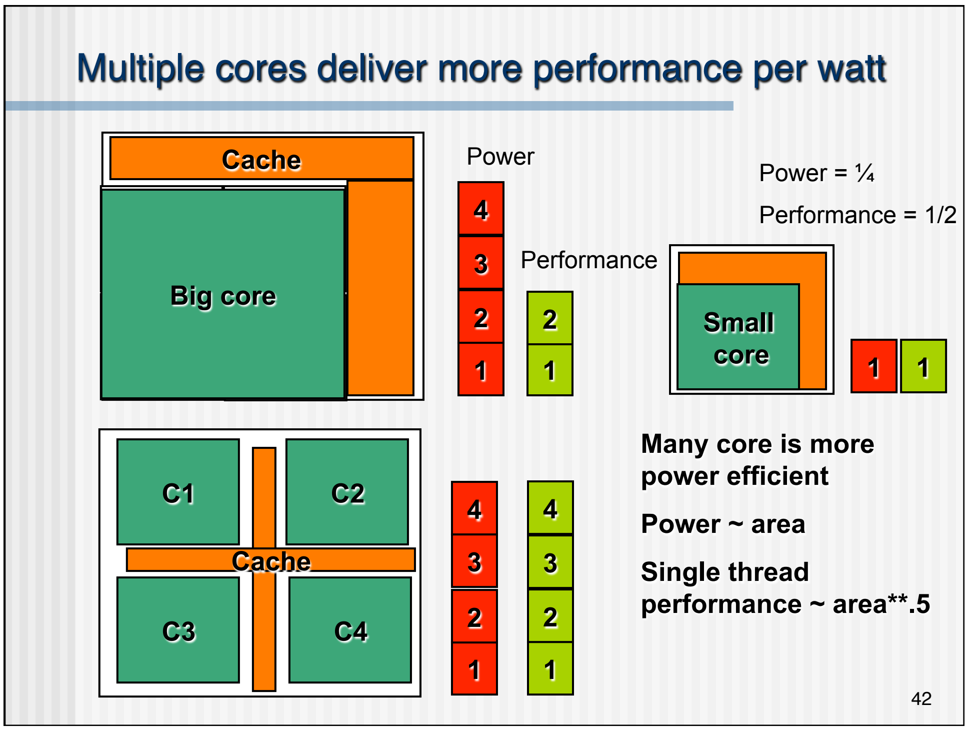
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

41



Multiple cores deliver more performance per watt

Power = ¼

Performance = 1/2

Many core is more power efficient

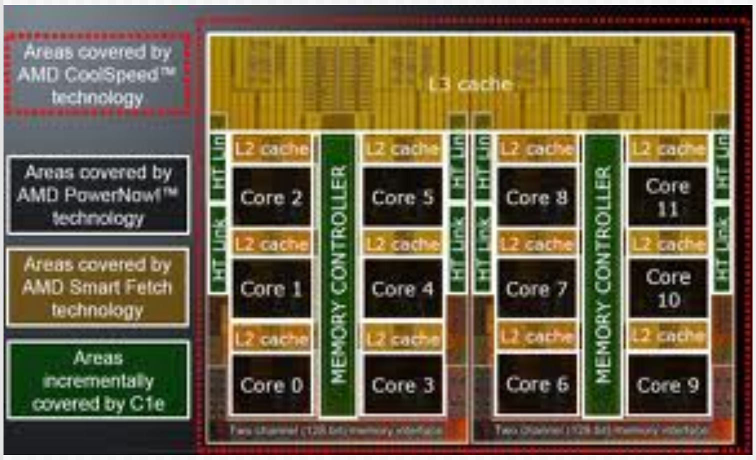
Power ~ area

Single thread performance ~ area.5**

42

Multicore CPUs

- AMD MagnyCours: 12 processor cores

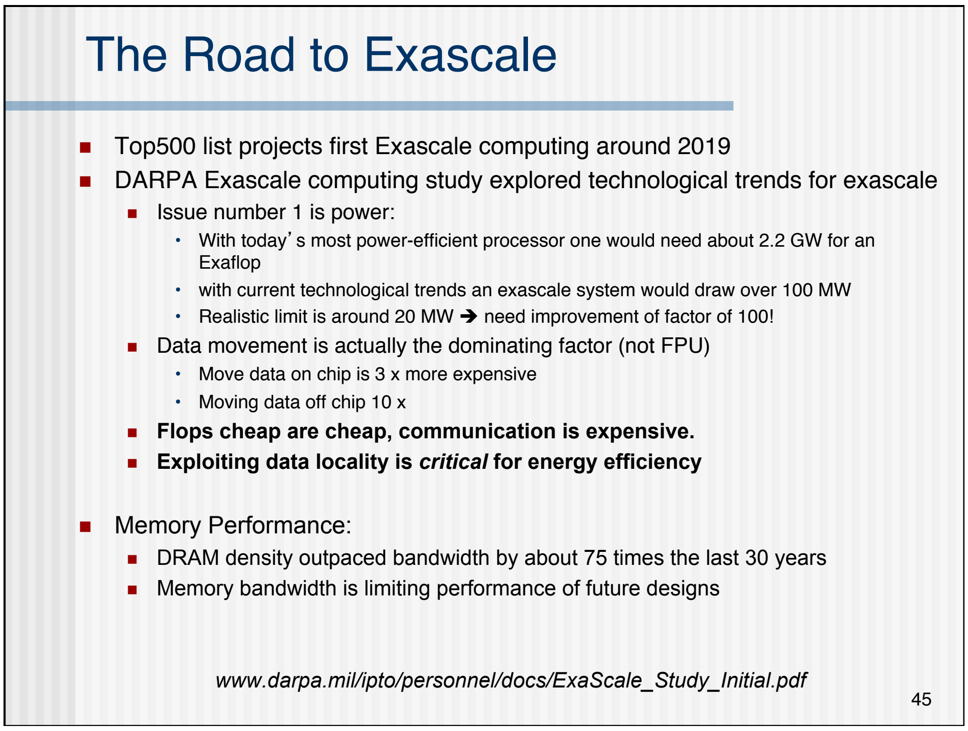

Areas covered by AMD CoolSpeed™ technology
Areas covered by AMD PowerNow!™ technology
Areas covered by AMD Smart Fetch technology
Areas incrementally covered by C1e
L3 cache
L2 cache
Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8
Core 9
Core 10
Core 11
MEMORY CONTROLLER
HT Link

43

What does this mean?

- The easy times have gone
  - Updating to the next processor generation will not automatically increase performance anymore
- Parallel computing techniques are needed to fully exploit new processor generation
- Parallel computing is going mainstream

44

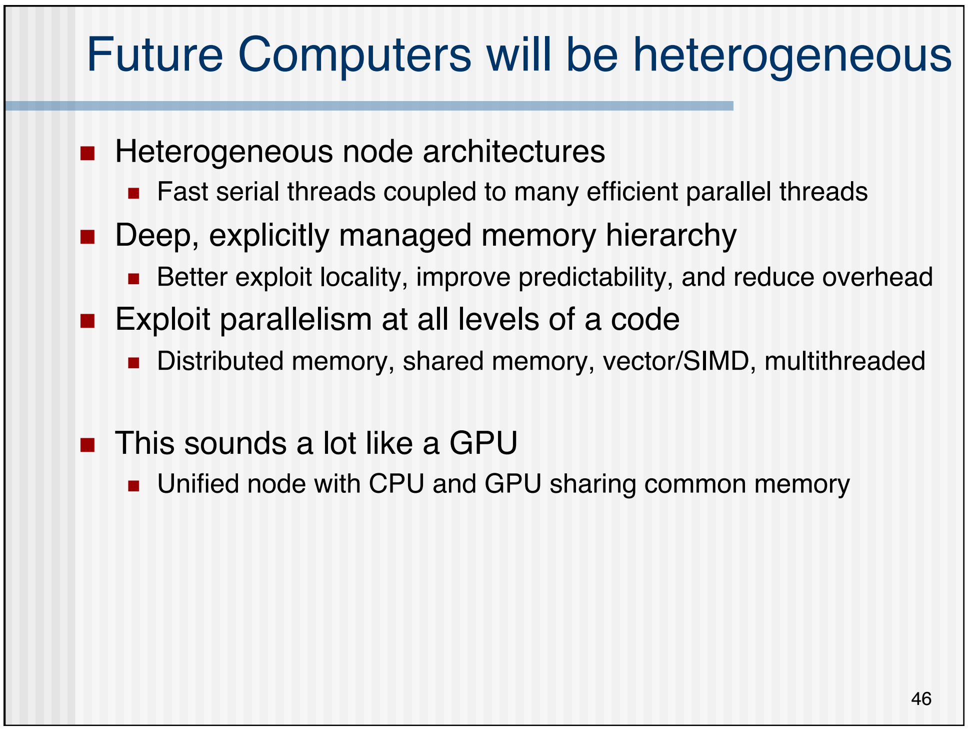


The Road to Exascale

- Top500 list projects first Exascale computing around 2019
- DARPA Exascale computing study explored technological trends for exascale
  - Issue number 1 is power:
    - With today's most power-efficient processor one would need about 2.2 GW for an Exaflop
    - with current technological trends an exascale system would draw over 100 MW
    - Realistic limit is around 20 MW ➔ need improvement of factor of 100!
  - Data movement is actually the dominating factor (not FPU)
    - Move data on chip is 3 x more expensive
    - Moving data off chip 10 x
  - **Flops cheap are cheap, communication is expensive.**
  - **Exploiting data locality is *critical* for energy efficiency**
- Memory Performance:
  - DRAM density outpaced bandwidth by about 75 times the last 30 years
  - Memory bandwidth is limiting performance of future designs

www.darpa.mil/ipto/personnel/docs/ExaScale_Study_Initial.pdf

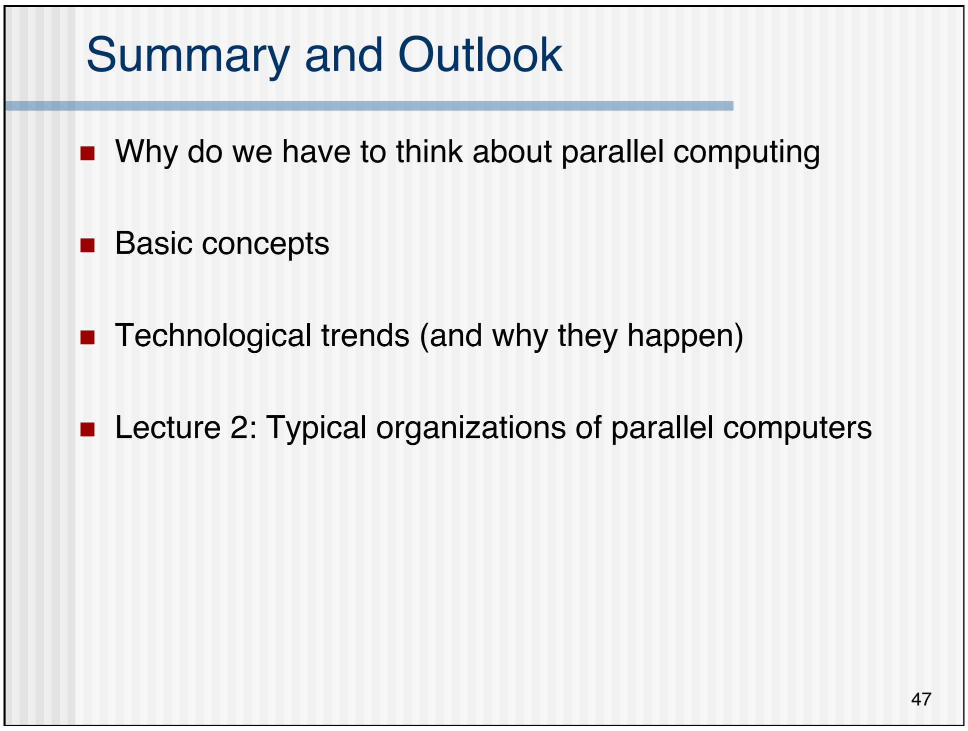
45



Future Computers will be heterogeneous

- Heterogeneous node architectures
  - Fast serial threads coupled to many efficient parallel threads
- Deep, explicitly managed memory hierarchy
  - Better exploit locality, improve predictability, and reduce overhead
- Exploit parallelism at all levels of a code
  - Distributed memory, shared memory, vector/SIMD, multithreaded
- This sounds a lot like a GPU
  - Unified node with CPU and GPU sharing common memory

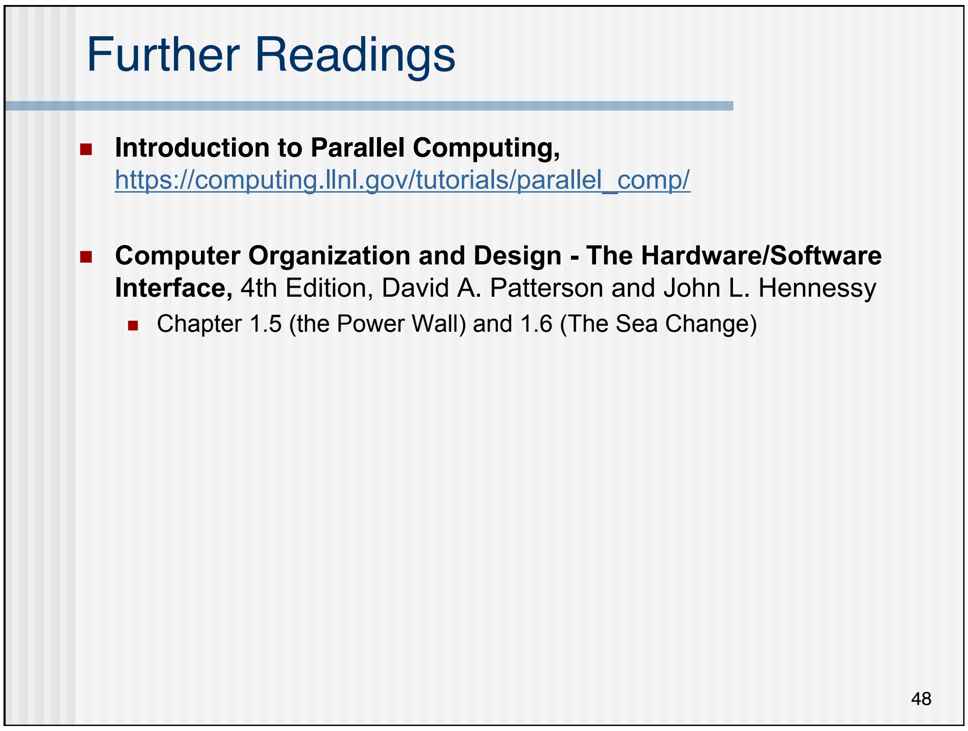
46



Summary and Outlook

- Why do we have to think about parallel computing
- Basic concepts
- Technological trends (and why they happen)
- Lecture 2: Typical organizations of parallel computers

47



Further Readings

- **Introduction to Parallel Computing,** https://computing.llnl.gov/tutorials/parallel_comp/
- **Computer Organization and Design - The Hardware/Software Interface,** 4th Edition, David A. Patterson and John L. Hennessy
  - Chapter 1.5 (the Power Wall) and 1.6 (The Sea Change)

48

Exercises

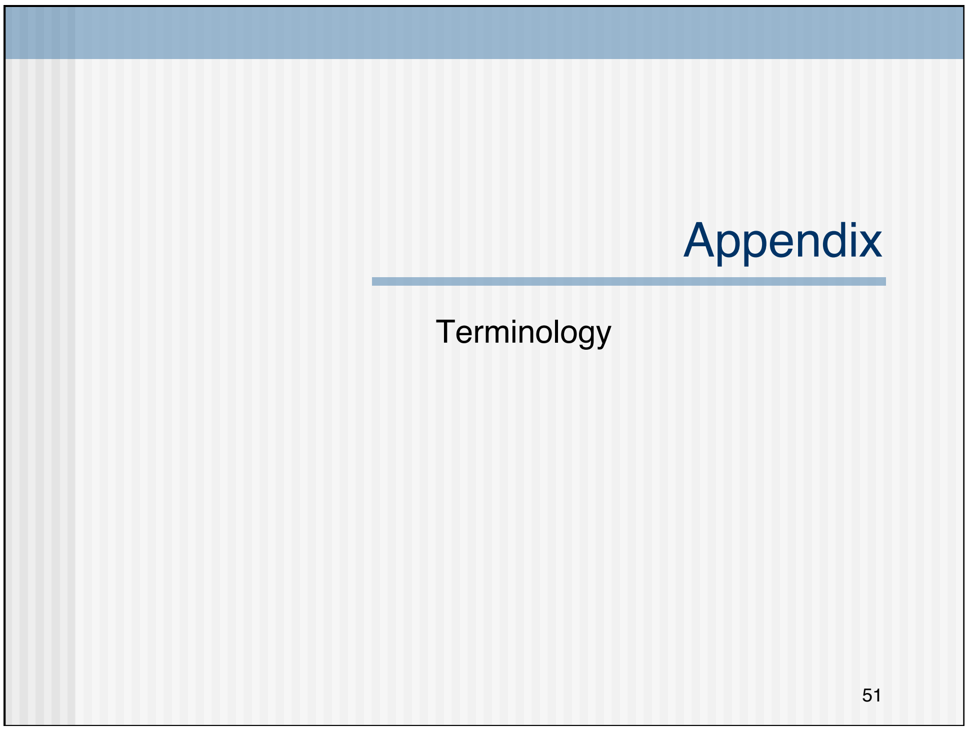
1. Start thinking parallel

2. Explain the differences/communalities between SPMD, SIMD, and MIMD

Exercises Cont' d

3. The following table shows the increase in clock rate and power of eight generations of Intel processors over 28 years. What is the largest relative change in clock rate and power between generations? How much larger is the clock rate and power of the last generation with respect to the first generation?

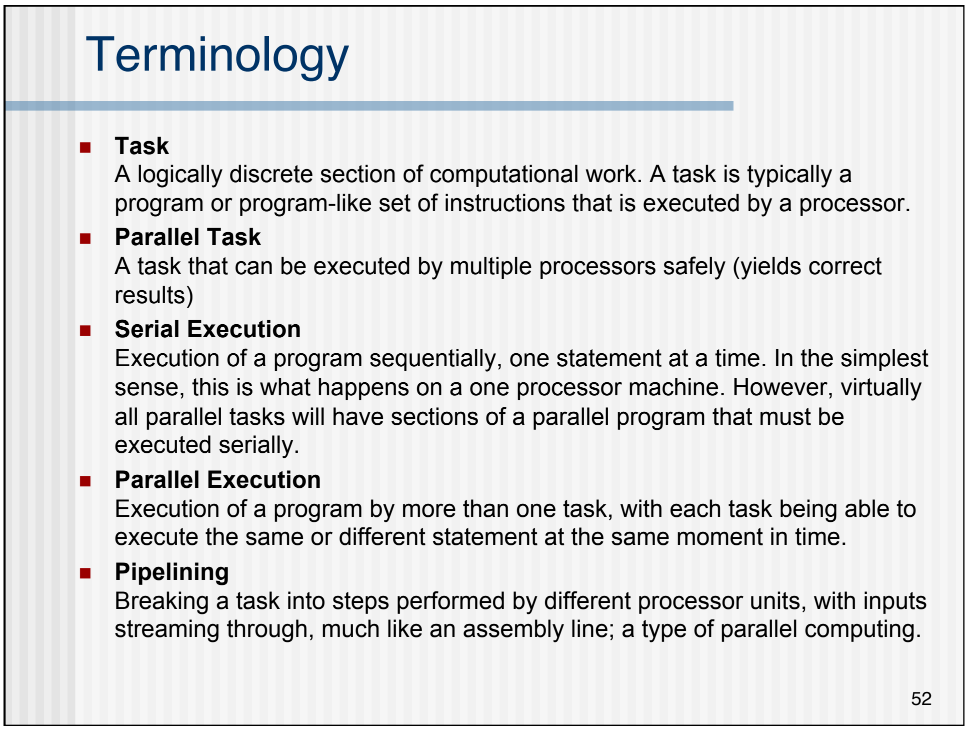
Processor	Clock rate	Power
80286 (1982)	12.5 MHz	3.3 W
80386 (1985)	16 MHz	4.1 W
80486 (1989)	25 MHz	4.9 W
Pentium (1993)	66 MHz	10.1 W
Pentium Pro (1997)	200 MHz	29.1 W
Pentium 4 Willamette (2001)	2 GHz	75.3 W
Pentium 4 Prescott (2004)	3.6 GHz	103 W
Core 2 Ketsfield (2007)	2.667 GHz	95 W



Appendix

Terminology

51



Terminology

- **Task**
  A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.
- **Parallel Task**
  A task that can be executed by multiple processors safely (yields correct results)
- **Serial Execution**
  Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.
- **Parallel Execution**
  Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.
- **Pipelining**
  Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

52

Terminology 2

- **Shared Memory**
  From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.
- **Symmetric Multi-Processor (SMP)**
  Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing.
- **Distributed Memory**
  In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.
- **Communications**
  Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

53

Terminology 3

- **Synchronization**
  The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task (s) reaches the same or logically equivalent point.Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.
- **Granularity**
  In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
  - ***Coarse:*** relatively large amounts of computational work are done between communication events
  - ***Fine:*** relatively small amounts of computational work are done between communication events
- **Observed Speedup**
  Observed speedup of a code which has been parallelized. One of the simplest and most widely used indicators for a parallel program's performance.

54

Terminology 4

- **Parallel Overhead**
  The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
  - Task termination time
- **Massively Parallel**
  Refers to the hardware that comprises a given parallel system - having many processors. The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands.
- **Embarrassingly Parallel**
  Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.

55

Terminology 5

- **Scalability**
  Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
  - Hardware - particularly memory-cpu bandwidths and network communications
  - Application algorithm
  - Parallel overhead related
  - Characteristics of your specific application and coding
- **Multi-core Processors**
  Multiple processors (cores) on a single chip.
- **Graphics Processing Unit (GPU)**
  specialized processor that offloads 3D graphics rendering from the microprocessor.
- **Cluster Computing**
  Use of a combination of commodity units (processors, networks or SMPs) to build a parallel system.
- **Supercomputing / High Performance Computing**
  Use of the world's fastest, largest machines to solve large problems.

56