北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

并行程序设计

邵兵

2022年3月1日

课程目的

本课程的主要目的是让学生认识到并行计算的重要性，在硬件层面了解主要的并行计算体系结构，在软件层面掌握如何使用MPI、Pthreads和OpenMP编写并行程序，并对并行程序的性能进行评估，了解主要的并行算法。最后，还将介绍最先进的CUDA等并行计算架构。

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

- 教材
  - Peter S. Pacheco, An Introduction to Parallel Programming, 机械工业出版社,2011年1月
  - Peter S. Pacheco著 邓倩妮等译《并行程序设计导论》, 机械工业出版社, 2010年5月
- 参考书
  - 陈国良编著，《并行计算——结构 算法 编程》，机械工业出版社，2011年6月。
- 教学方法与考核方式（不及格无补考！）
  - 课堂考勤（10%）
  - 平时书面作业（10%）
  - 平时编程作业（40%）
  - 大作业天梯赛（40%）

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

主讲老师及助教

角色	姓名	邮箱
主讲老师	邵兵	**shaobing@buaa.edu.cn**
助教	王贯中	**wgz13501962114@163.com**

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Chapter 1
Parallel Computing

软件学院　邵兵

2022年3月1日

Contents

- **Overview**
- **Concepts**
- **Parallel computer memory architectures**
- **Parallel programming models**
- **Designing parallel programs**
- **Parallel examples**

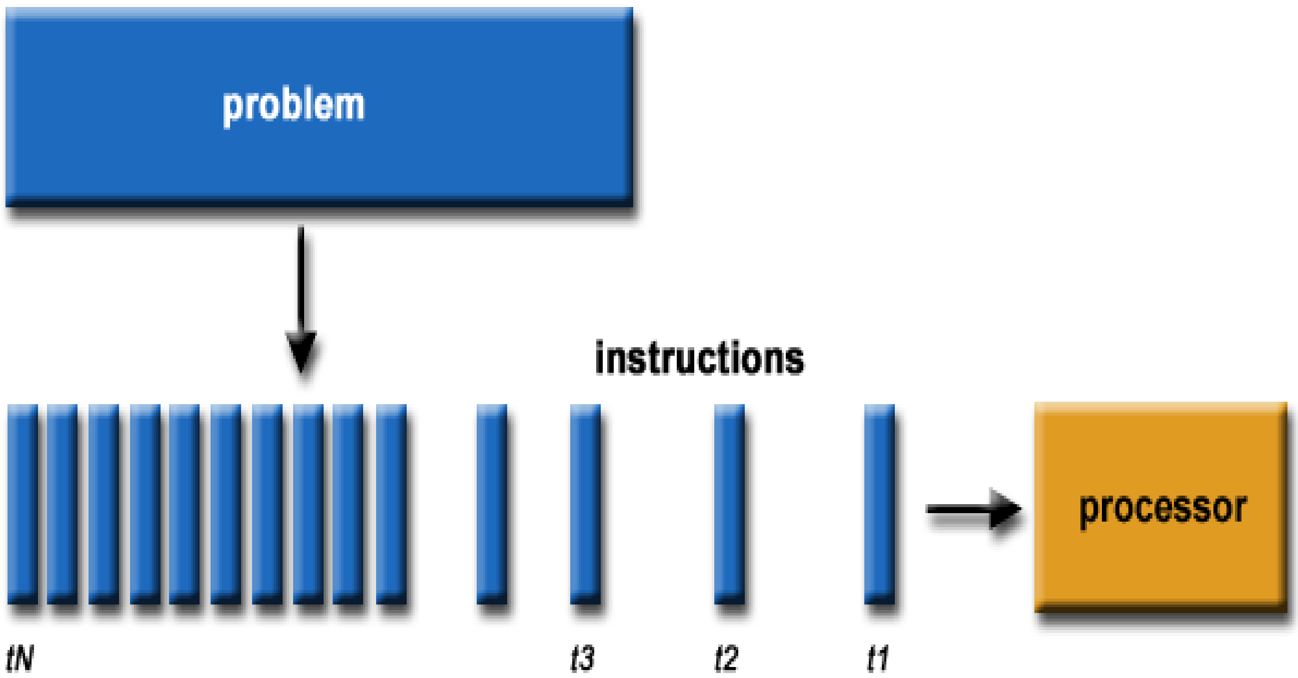
北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Overview

北京航空航天大學
COLLEGE OF SOFTWARE 软件学院
BEIHANG UNIVERSITY

Overview

- Serial Computing

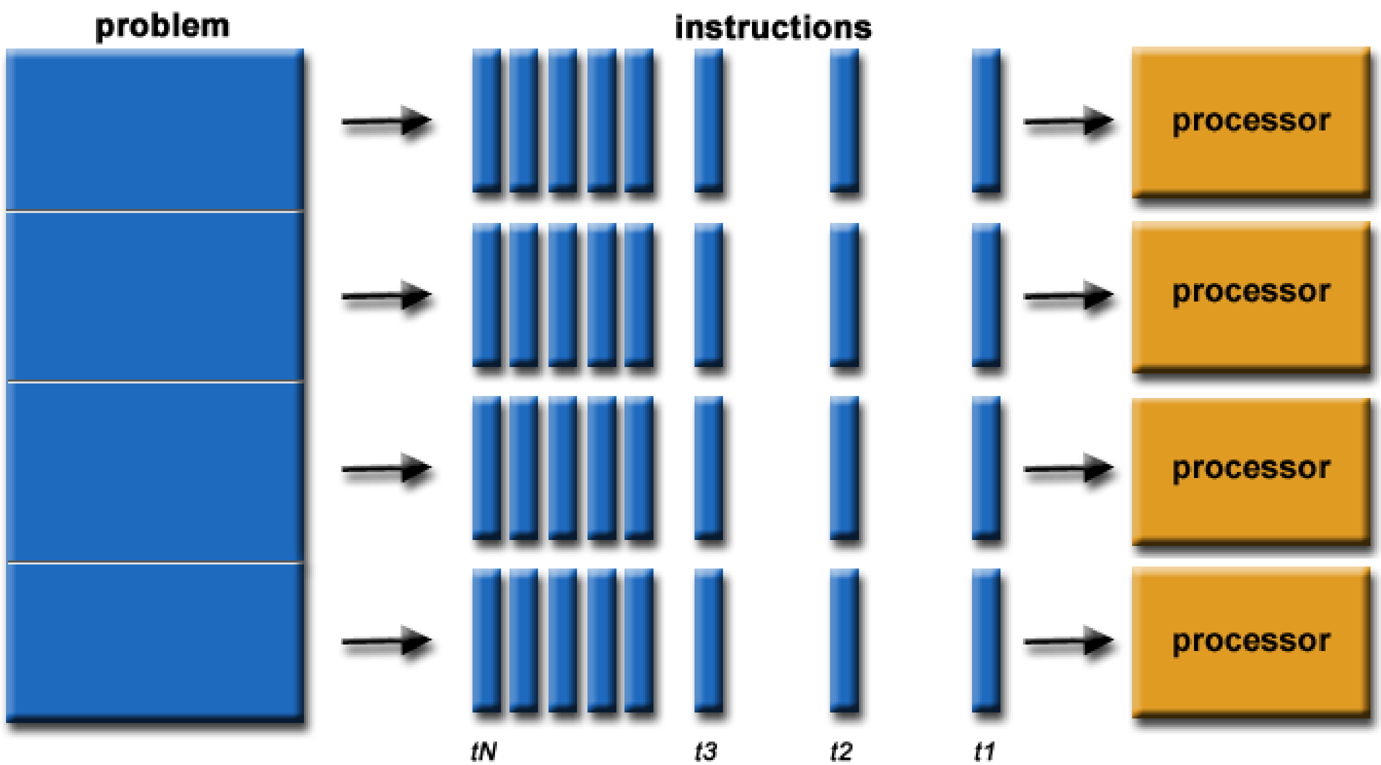


Overview

- **Serial Computing:**
  - 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

Overview

- ## Parallel Computing

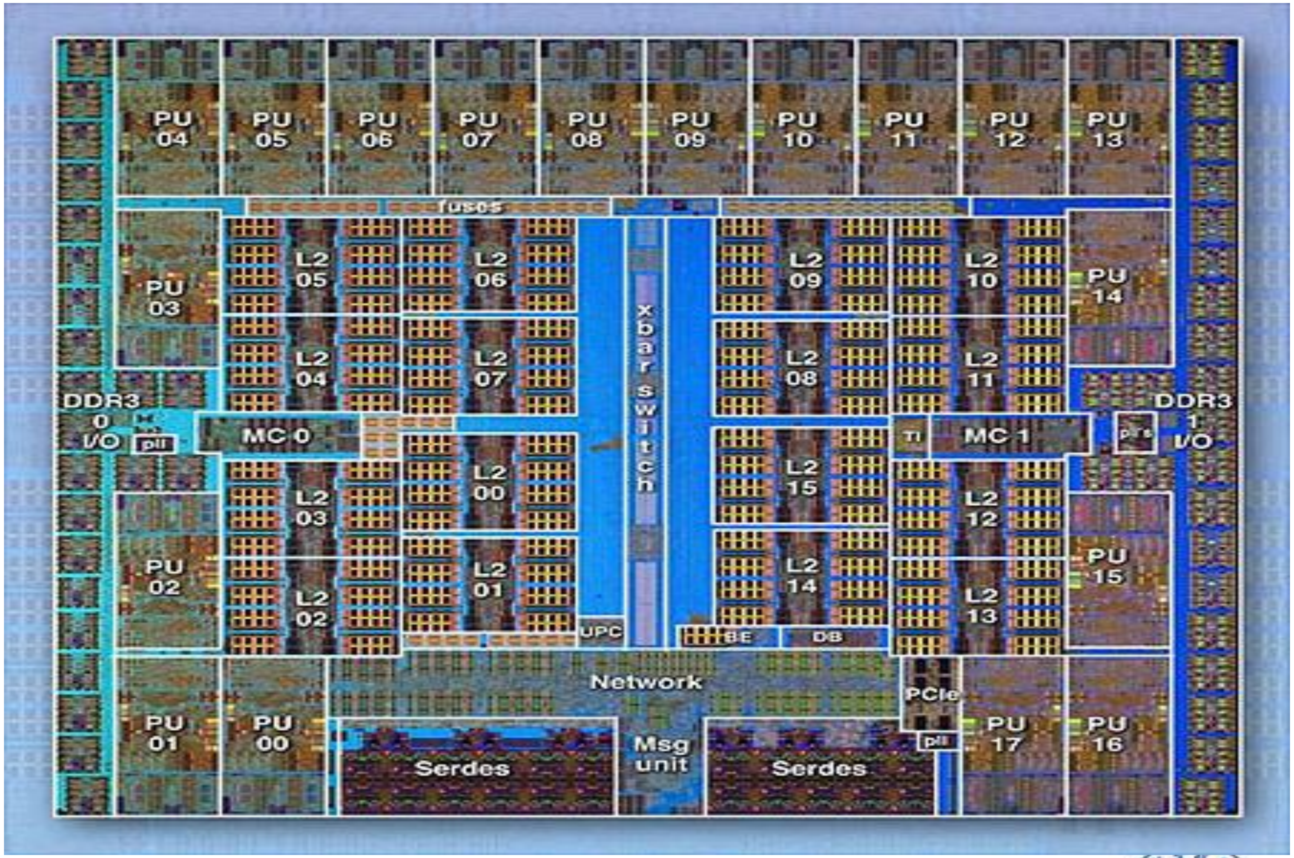


Overview

- **Parallel Computing:**
  - 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

https://www.ibm.com/ibm/history/ibm100/us/en/icons/bluegene/

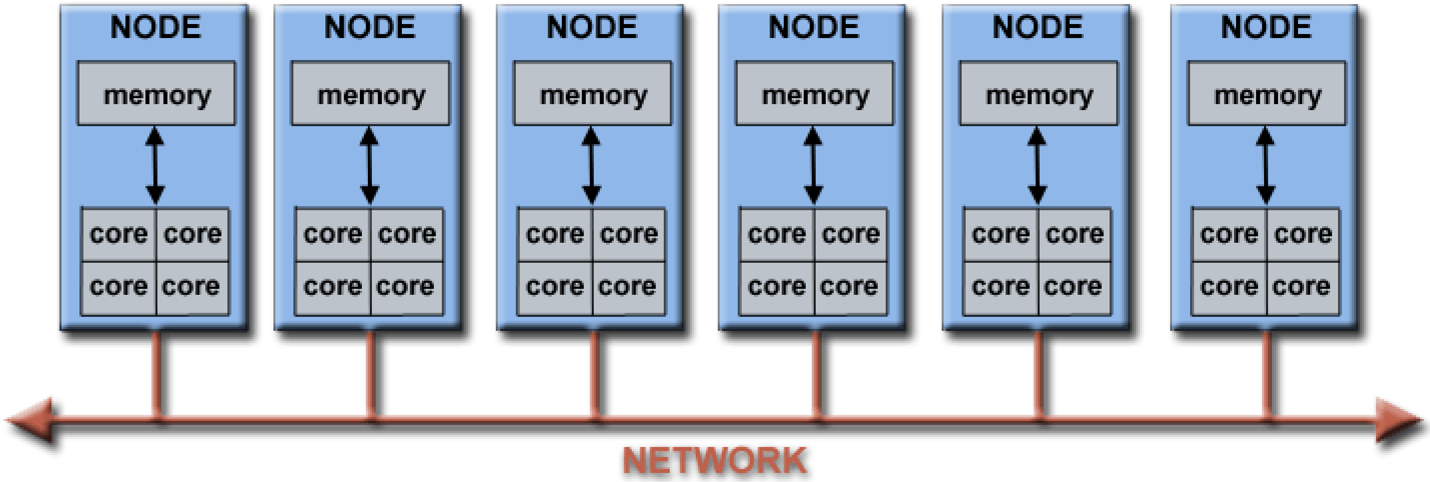
- **Parallel Computers:**
  - Hardware perspective



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

Overview

- **Parallel Computers:**
  - Networks connect perspective

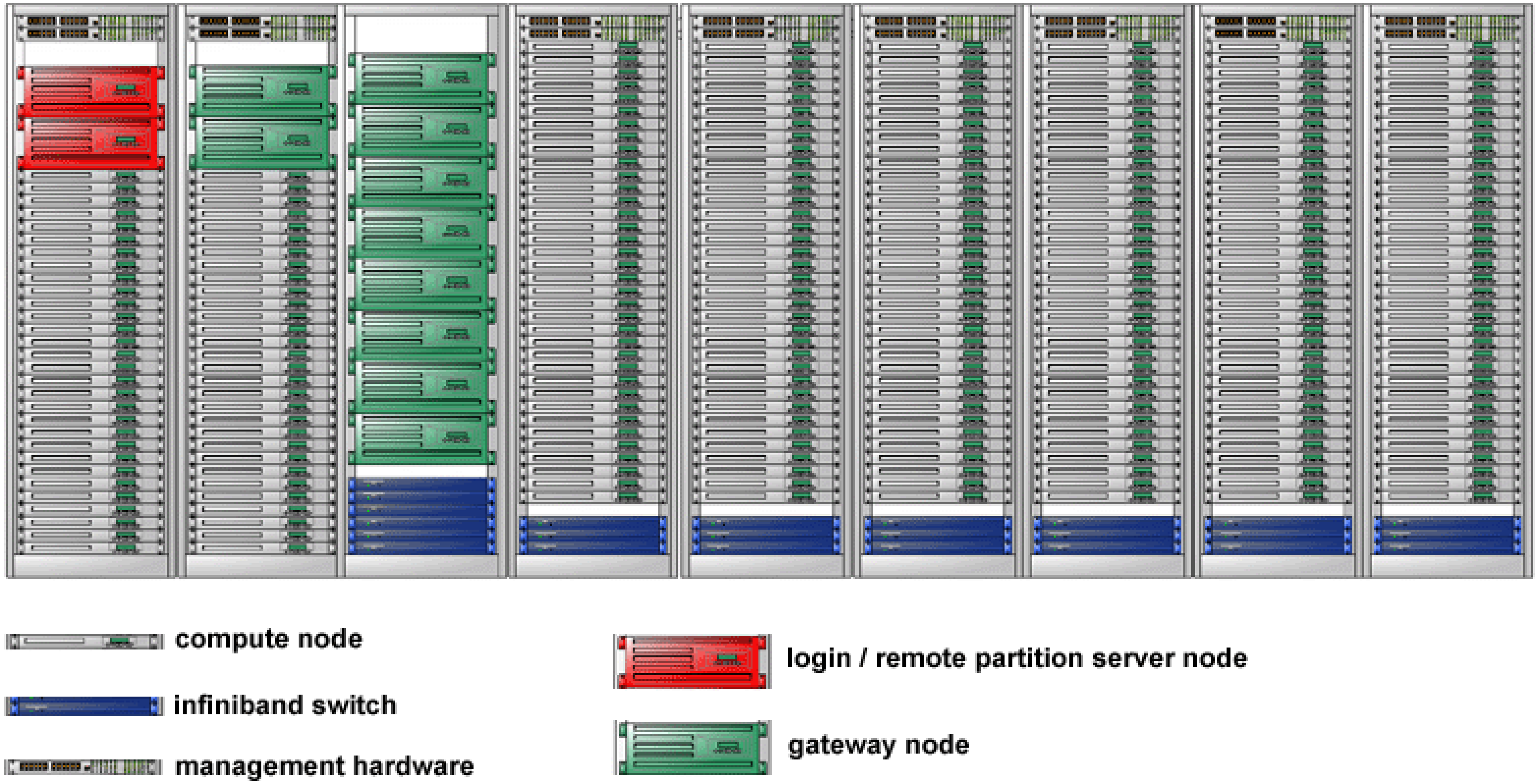


Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.

Overview

- **Parallel Computers:**

  for example: LLNL parallel computer



- **Why Use Parallel Computing?**
  - SAVE TIME AND MONEY:

    a. throwing more resources at a task will shorten its time to completion

    b. can be built from cheap, commodity components

Overview

- **Why Use Parallel Computing?**
  - 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.

- **Why Use Parallel Computing?**
  - PROVIDE CONCURRENCY:

    Multiple compute resources can do many things simultaneously.

Example: Web search engines/databases processing millions of transactions every second

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Overview

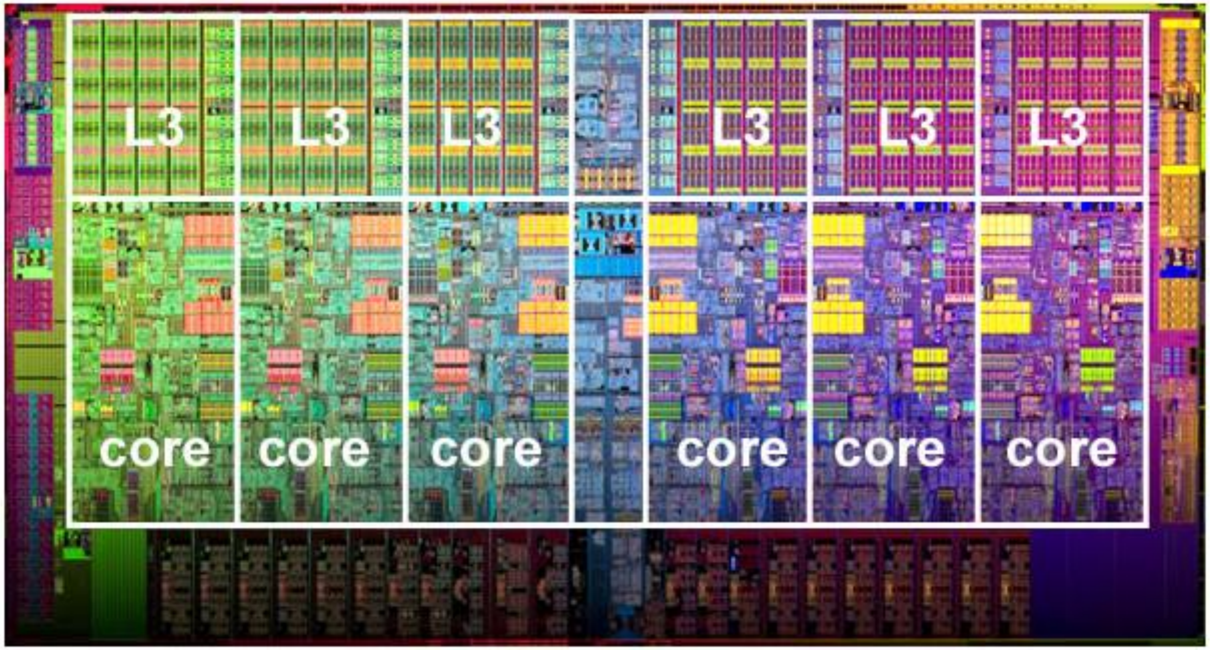
- **Why Use Parallel Computing?**
  - TAKE ADVANTAGE OF NON-LOCAL RESOURCES:
    - Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Overview

- **Why Use Parallel Computing?**
  - MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE:

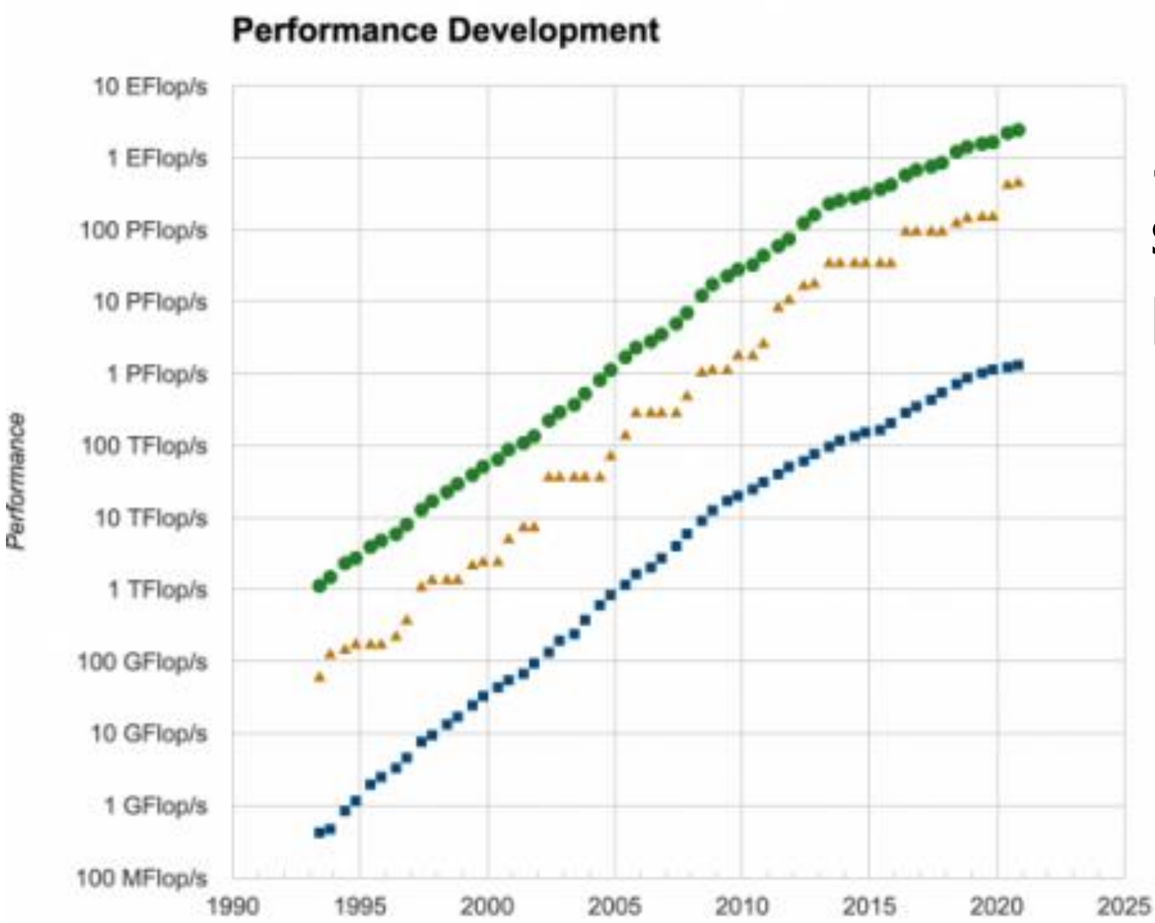
    Serial programs run on modern computers(are parallel in architecture with multiple processors/cores)"waste" potential computing power.



Intel Xeon processor with 6 cores and 6 L3 cache units

Overview

- **The future**
  - Exaflop = 10^{18} calculations per second.



500,000x increase in supercomputer performance

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

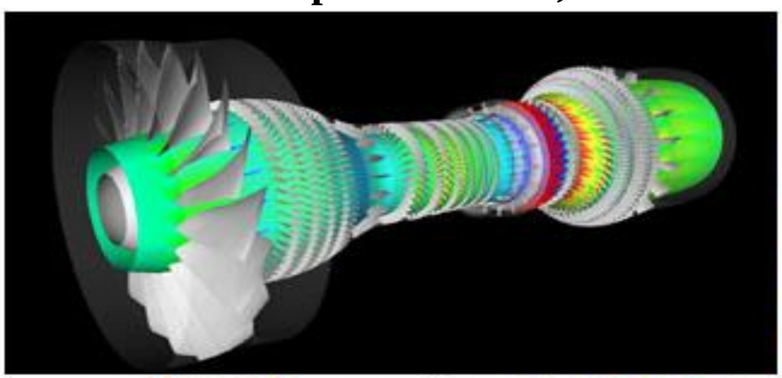
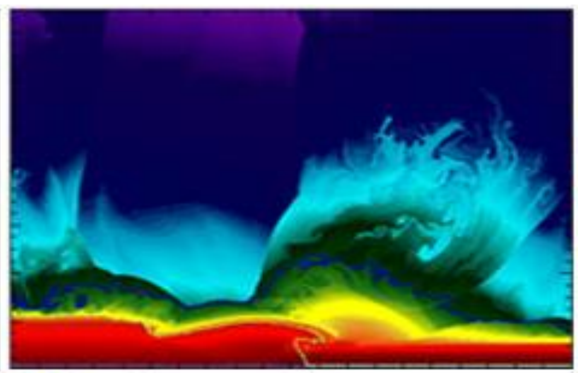
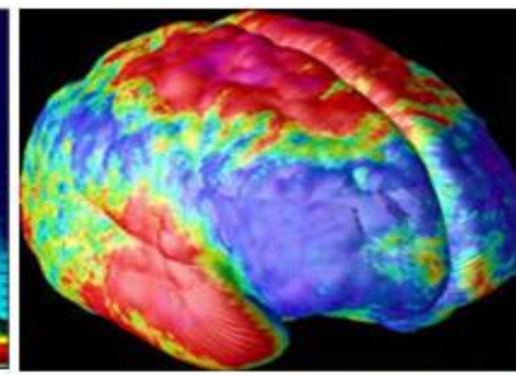
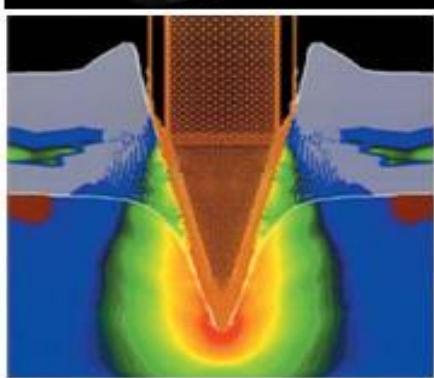
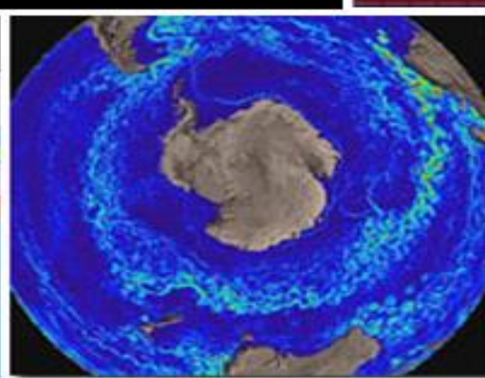
TOP10 Supercomputers (November, 2021) (https://www.top500.org/lists/top500/2021/11/)

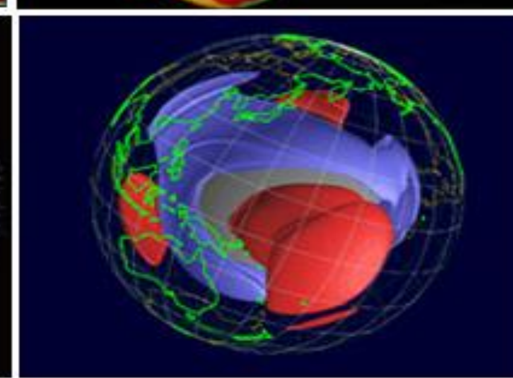
Rank	System	Country	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku	Japan	7,630,848	442,010.00	537,212.00	29,899
2	Summit	United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra	United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight**	China	10,649,600	93,014.6	125,435.9	15,371
5	Perlmutter	United States	761,856	70,870.0	93,750.0	2,589
6	Selene	United States	555,520	63,460.0	79,215.0	2,646
7	**Tianhe-2A**	China	4,981,760	61,444.5	100,678.7	18,482
8	JUWELS Booster Module	Germany	449,280	44,120.0	70,980.0	1,764
9	HPC5	Italy	669,760	35,450.0	51,720.8	2,252
10	Voyager-EUS2	United States	253,440	30,050.0	39,531.2	

Overview

- **Who is Using Parallel Computing?**
  - Science and Engineering:

1. Atmosphere, Earth, Environment
2. Physics
3. Bioscience, Biotechnology, Genetics
4. Chemistry, Molecular Sciences
5. Geology, Seismology
6. Mechanical Engineering
7. Electrical Engineering, Circuit Design, Microelectronics
8. Computer Science, Mathematics

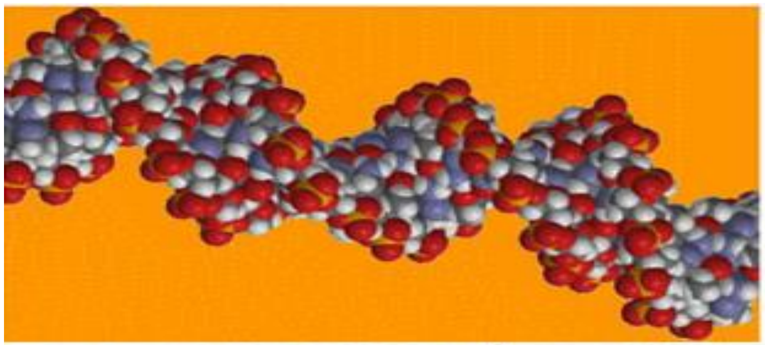
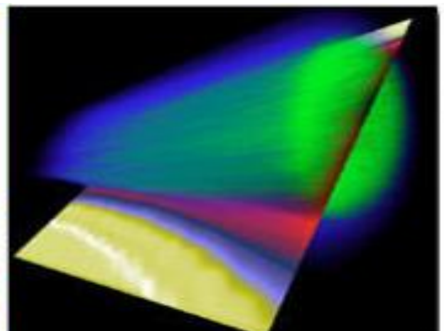
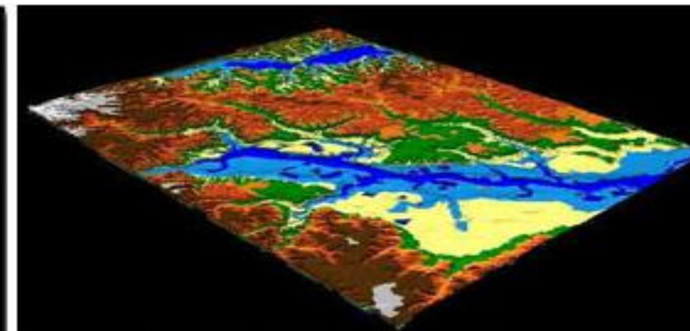
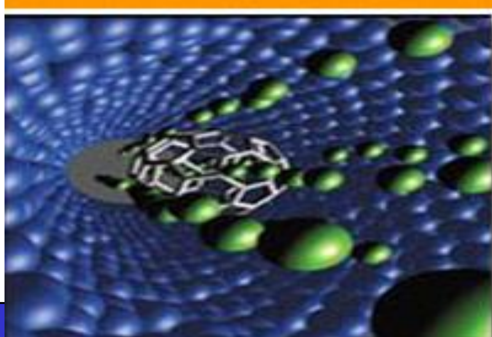


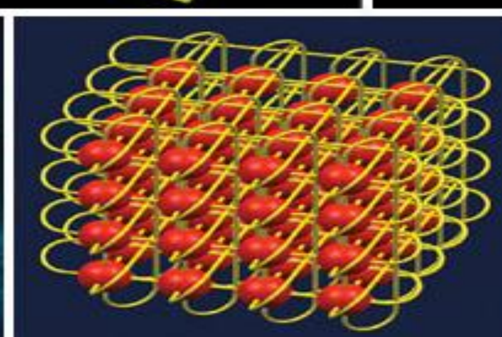
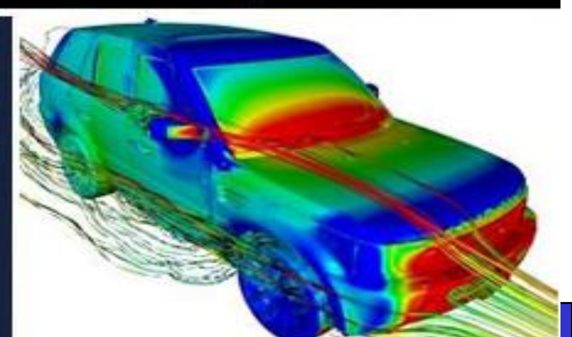
Overview

▪ Who is Using Parallel Computing?

- Industrial and Commercial:

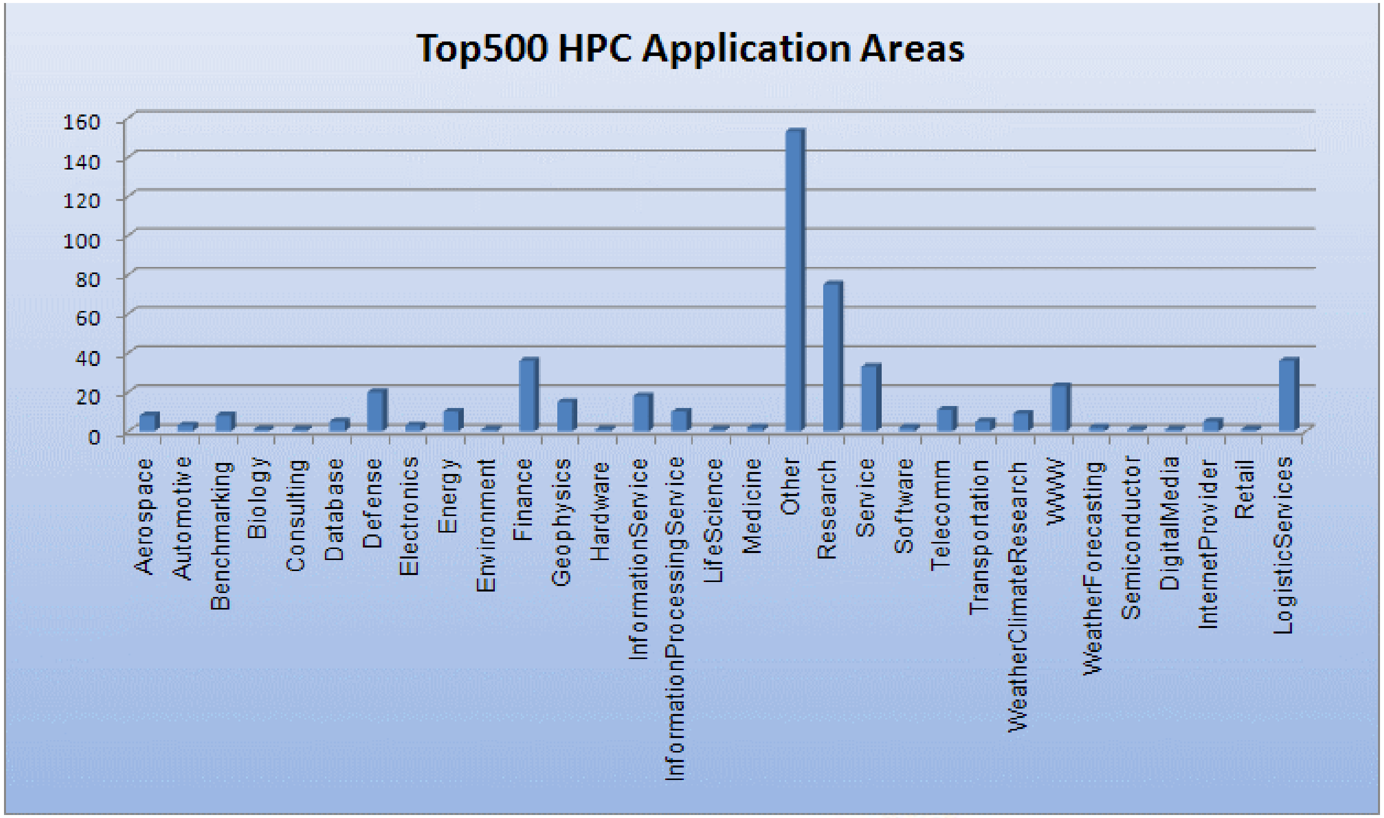
  1. "Big Data", databases, data mining
  2. Oil exploration
  3. Web search engines
  4. Medical imaging and diagnosis
  5. Pharmaceutical design
  6. Financial and economic modeling
  7. Management of national and multi-national corporations
  8. Advanced graphics and virtual reality
  9. Networked video and multi-media technologies
  10. Collaborative work environments

Who is Using Parallel Computing?

- Global Applications:

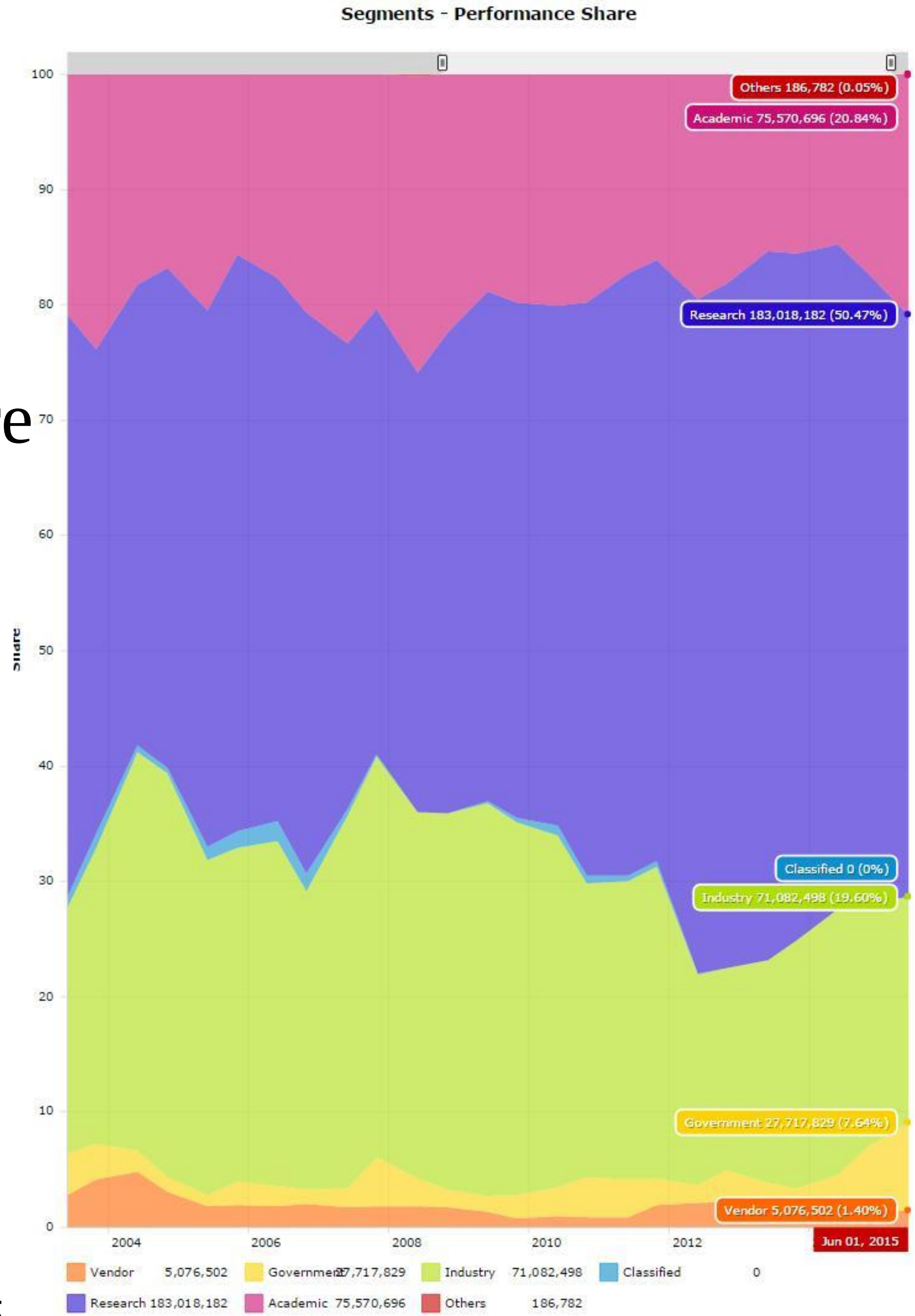


北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Overview

- **Who is Using Parallel Computing?**

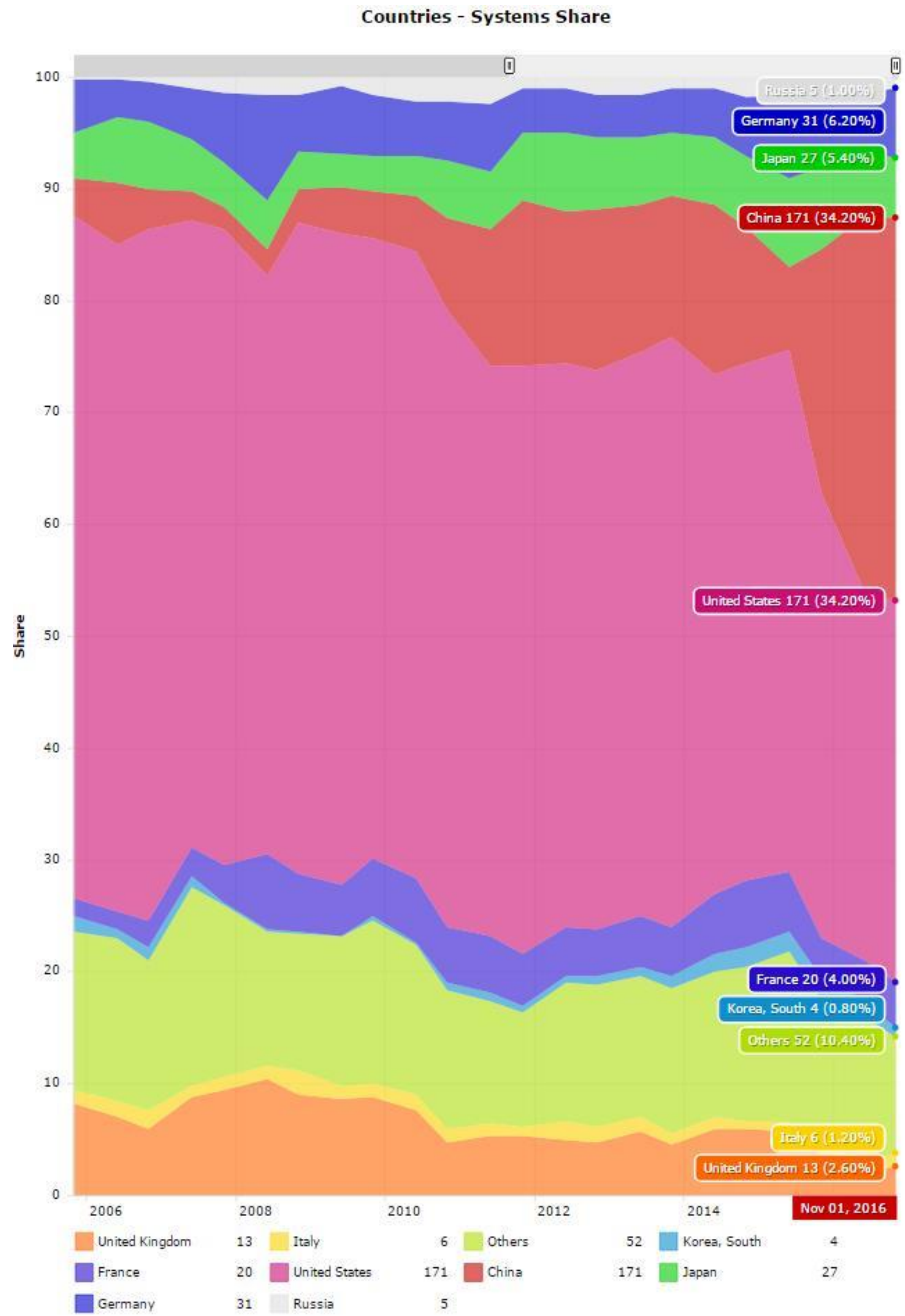
Segments – Performance Share



Overview

- **Who is Using Parallel Computing?**

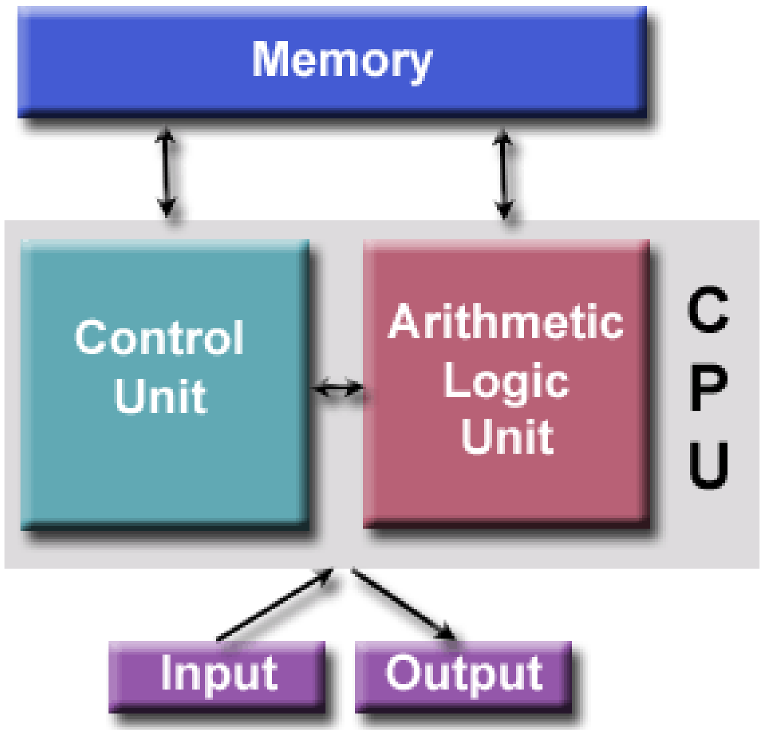
Countries – Systems share



Concepts

Concepts

- ## Von Neumann Architecture



Basic computing architecture

John von Neumann circa 1940s (Source: LANL archives)

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

- **Flynn's Classical Taxonomy**
  - according to how they can be classified along the two independent dimensions of ***Instruction Stream*** and ***Data Stream***

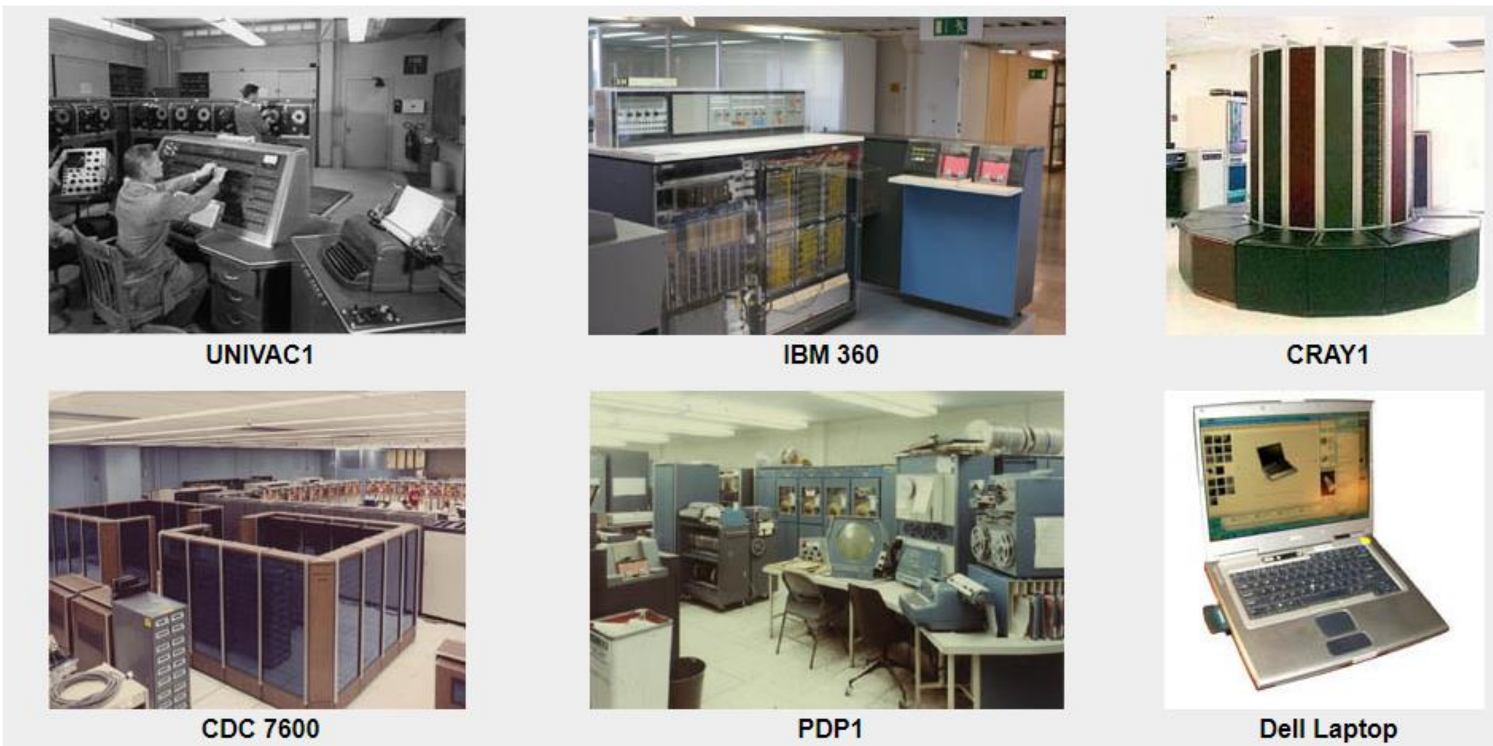
S I S D Single Instruction stream Single Data stream	**S I M D** Single Instruction stream Multiple Data stream
M I S D Multiple Instruction stream Single Data stream	**M I M D** Multiple Instruction stream Multiple Data stream

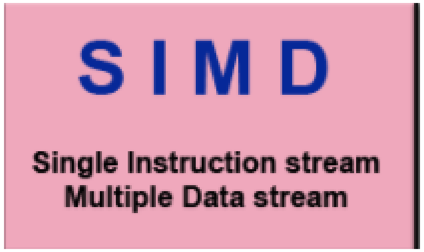
Concepts

S I S D

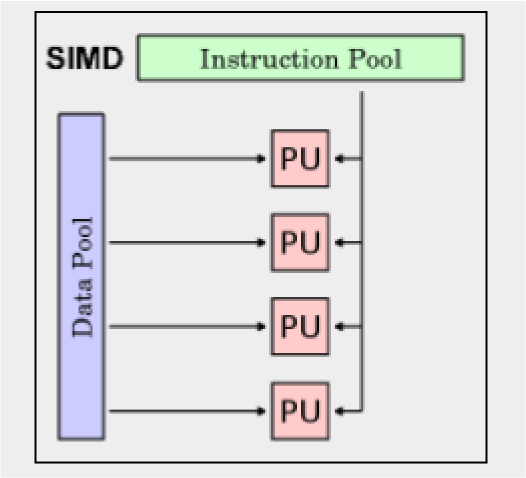
Single Instruction stream
Single Data stream

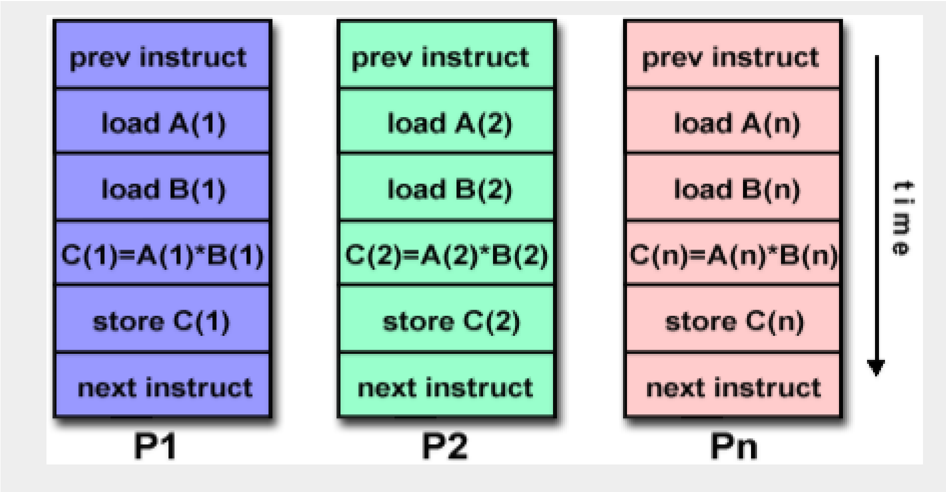
- **Flynn's Classical Taxonomy**
  - Single Instruction, Single Data (SISD):

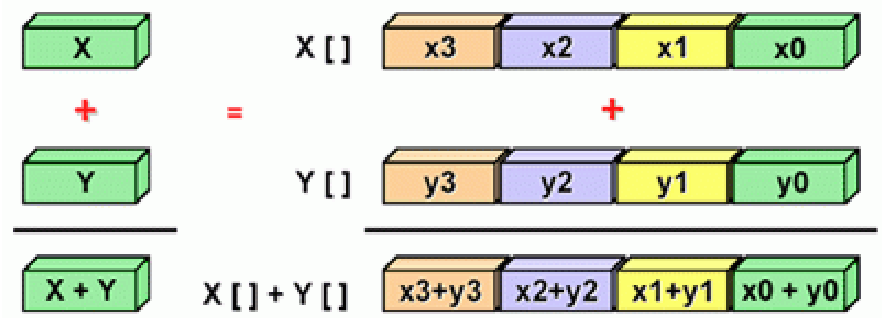




- **Flynn's Classical Taxonomy**
  - Single Instruction, Multiple Data (SIMD):







Concepts

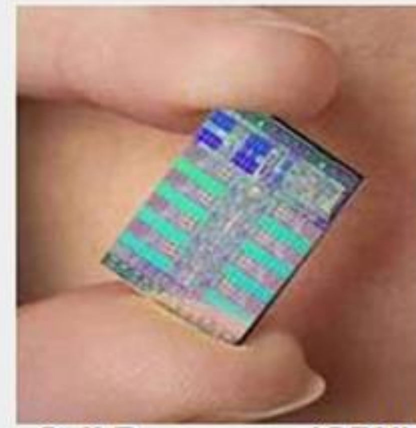
S I M D

Single Instruction stream
Multiple Data stream

- **Flynn's Classical Taxonomy**
  - Single Instruction, Multiple Data (SIMD):

ILLIAC IV

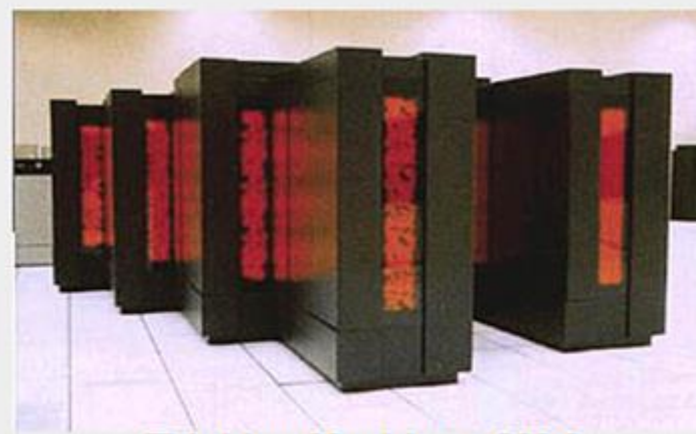
MasPar



Cell Processor (GPU)

Cray X-MP

Cray Y-MP



Thinking Machines CM-2

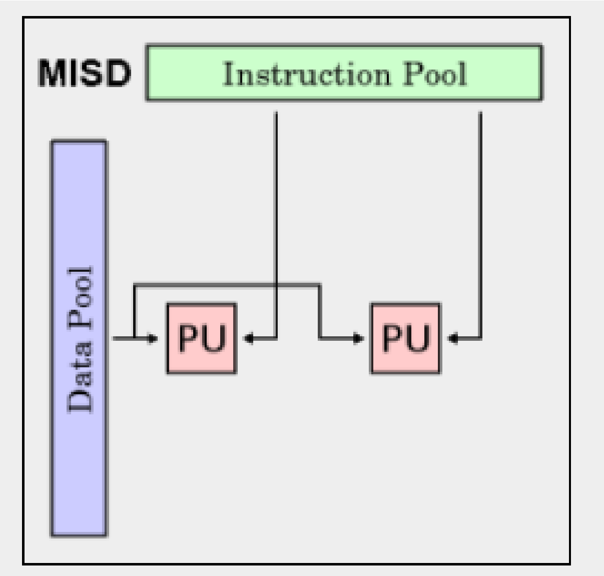
Concepts

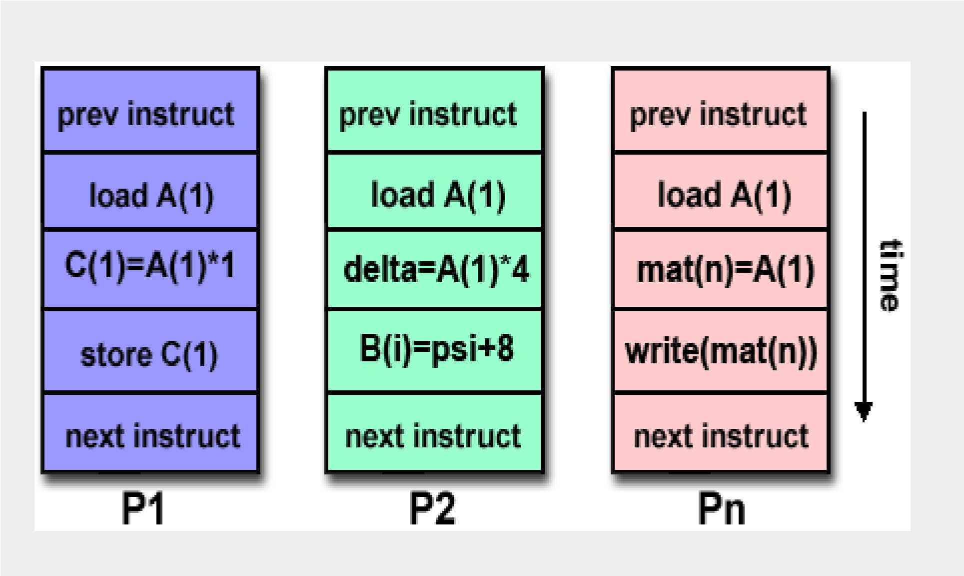
Flynn's Classical Taxonomy

- Multiple Instruction, Single Data (MISD):

  Few (if any) actual examples have ever existed. Some conceivable uses might be:

  - **multiple frequency filter**
  - **multiple cryptography algorithms**





Concepts

- **Flynn's Classical Taxonomy**
  - Multiple Instruction, Multiple Data (MIMD):

IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

IBM BG/L

Concepts

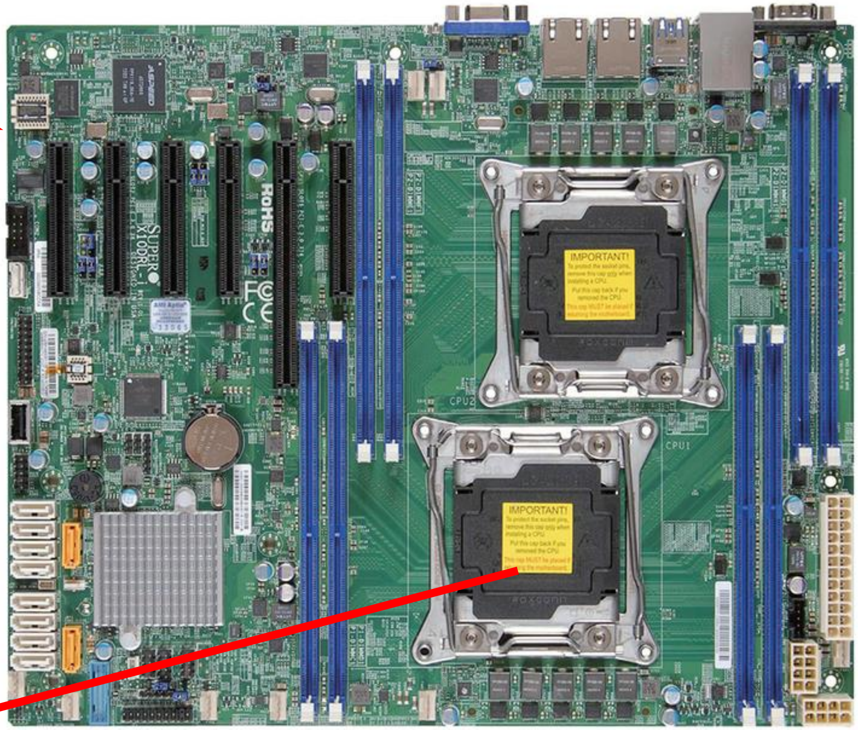
Some General Parallel Terminology

- Supercomputing / High Performance Computing (HPC)
- Node
- CPU / Socket / Processor / Core
- Task
- Pipelining
- Shared Memory
- Symmetric Multi-Processor (SMP)
- Distributed Memory

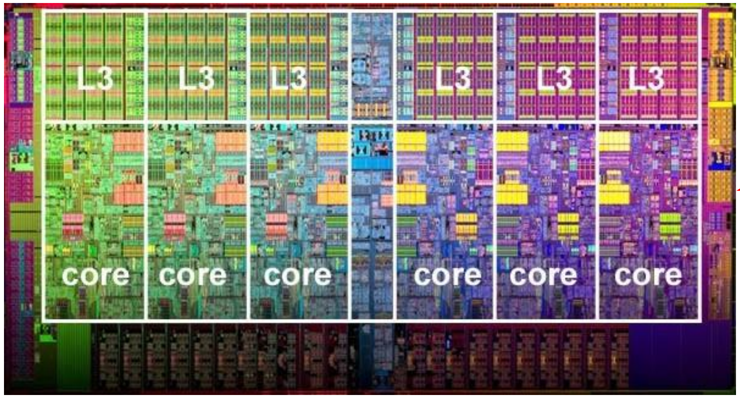
北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Supercomputer – each blue light is a node

Node – standalone Von Neumann computer



CPU/Processer/Socket – each has multiple cores/processers.



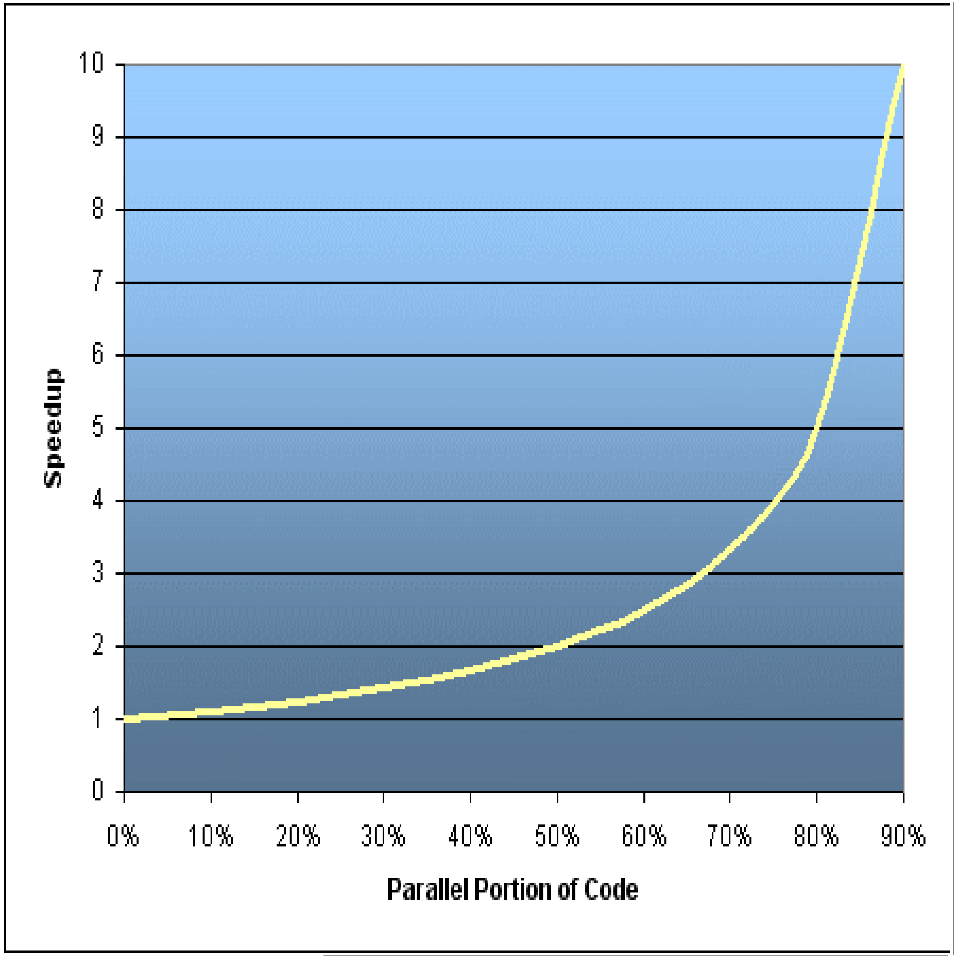
北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

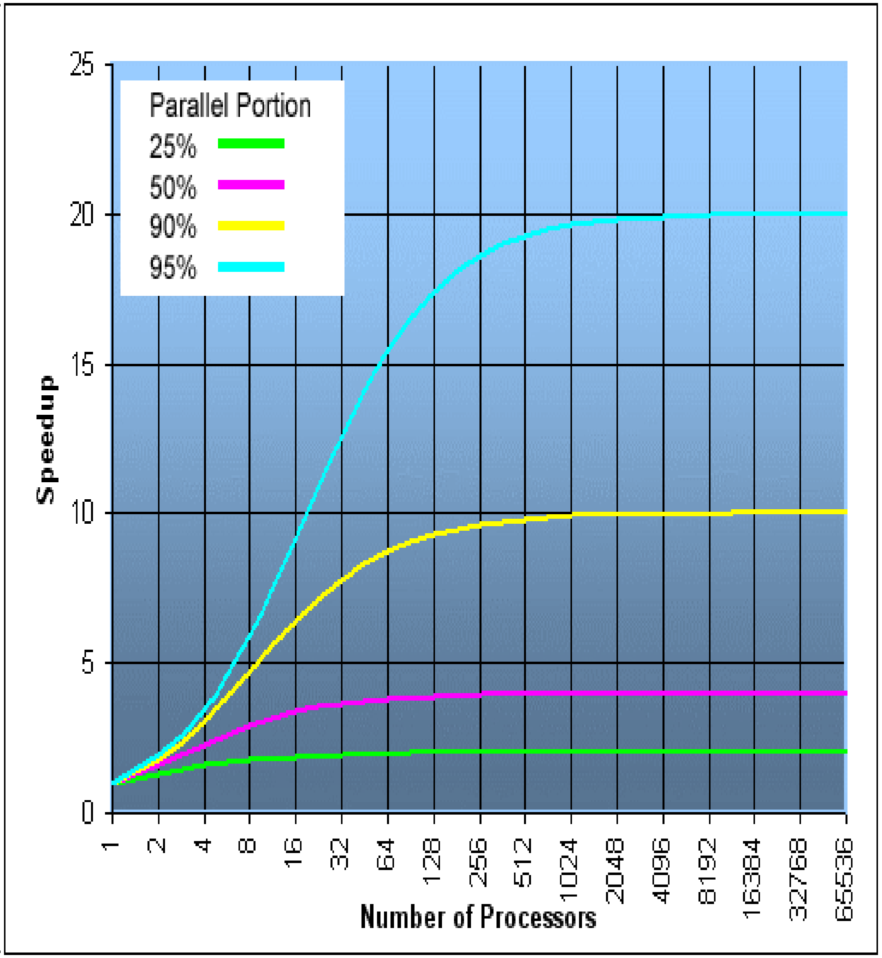
Some General Parallel Terminology

- Communications
- Synchronization
- Computational Granularity
- Observed Speedup
- Parallel Overhead
- Massively Parallel
- Embarrassingly Parallel
- Scalability

Concepts

- **Limits and Costs of Parallel Programming**





Concepts

- **Limits and Costs of Parallel Programming**
  - Complexity:
    - Design
    - Coding
    - Debugging
    - Tuning
    - Maintenance

Concepts

- **Limits and Costs of Parallel Programming**
  - Portability:
    - All of the usual portability issues associated with serial programs apply to parallel programs.
    - sometimes to the point of requiring code modifications in order to effect portability.
    - Operating systems can play a key role in code portability issues.
    - Hardware architectures are characteristically highly variable and can affect portability.

Concepts

- **Limits and Costs of Parallel Programming**
  - Resource Requirements:
    - The amount of memory required can be greater for parallel codes than serial codes,
    - For short running parallel programs, there can actually be a decrease in performance.

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

- **Limits and Costs of Parallel Programming**
  - Scalability:
    - **Strong scaling**
      - to run the same problem size faster
    - **Weak scaling**
      - to run larger problem in same amount of time

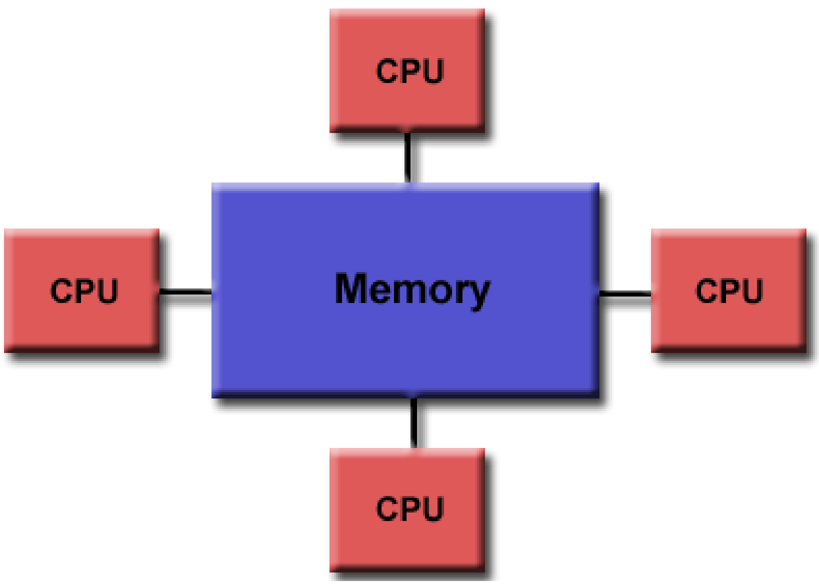
Parallel Computer Memory Architectures

Parallel Computer Memory Architectures

- **Shared Memory**
  - General Characteristics:
    - shared memory machines have been classified as ***UMA*** and ***NUMA***, based upon memory access times.

Parallel Computer Memory Architectures

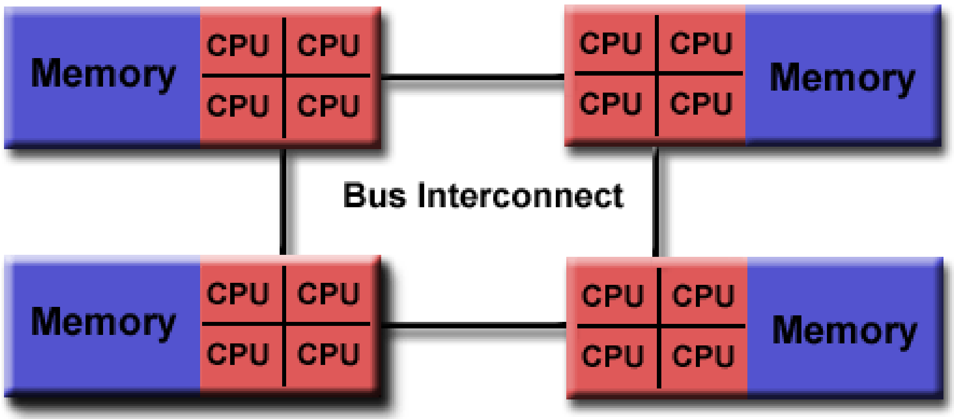
- Shared Memory
  - General Characteristics:
  - Uniform Memory Access: UMA



Shared Memory (UMA)

Parallel Computer Memory Architectures

- **Shared Memory**
  - General Characteristics:
  - Non-Uniform Memory Access: NUMA



Shared Memory (NUMA)

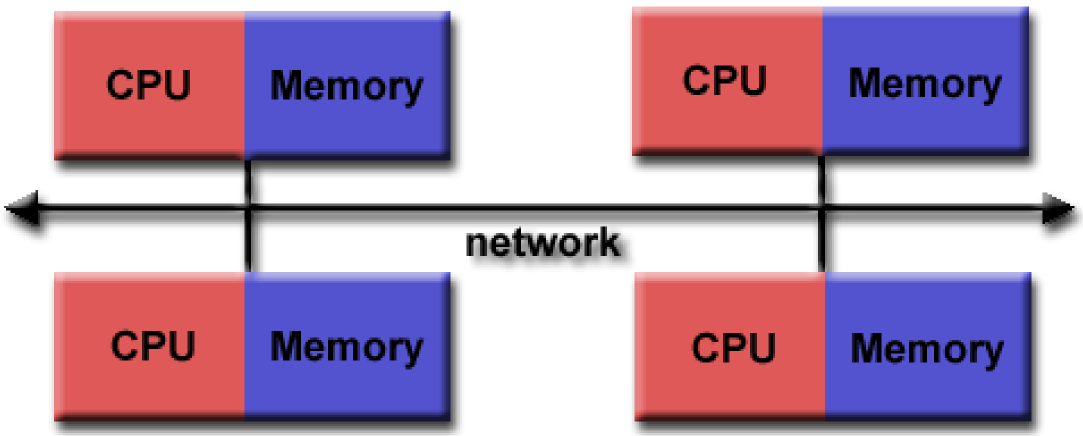
北京航空航天大學 COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Parallel Computer Memory Architectures

- **Shared Memory**
  - **Advantages**:
    - provides a user-friendly programming perspective to memory.
    - data sharing between tasks is both fast and uniform.
  - **Disadvantages**:
    - lack of scalability between memory and CPUs.

Parallel Computer Memory Architectures

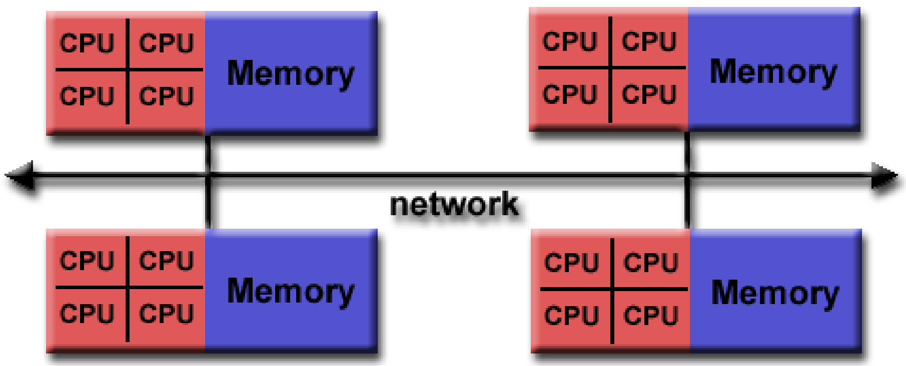
- **Distributed Memory**
  - distributed memory systems vary widely but share a common characteristic.

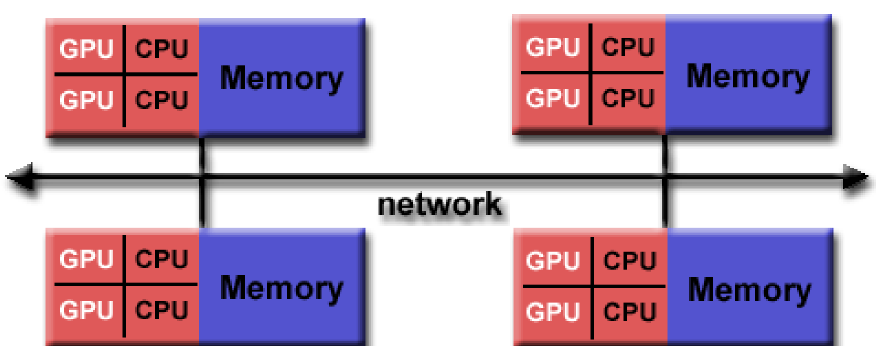


- **Distributed Memory**
  - **Advantage**:
    - Memory is scalable with the number of processors.
    - Can use commodity, off-the-shelf processors and networking.
  - **Disadvantage**:
    - It's difficult to map existing data structures, based on global memory, to memory organization.
    - Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Parallel Computer Memory Architectures

- **Hybrid Distributed-Shared Memory**
  - The largest and fastest computers in the world today employ both shared and distributed memory architectures.





北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

- **Hybrid Distributed-Shared Memory**
  - Advantages and Disadvantages
    - Whatever is common to both shared and distributed memory architectures.
    - Increased scalability is an important advantage
    - Increased programmer complexity is an important disadvantage

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Parallel Programming Models

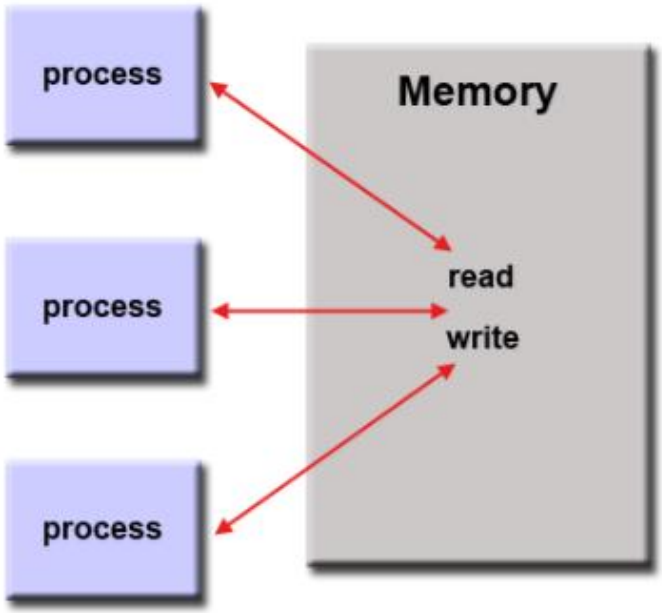
Parallel Programming Models

- **Models**
  1. Shared Memory (without threads)
  2. Threads
  3. Distributed Memory / Message Passing
  4. Data Parallel
  5. Hybrid
  6. SPMD and MPMD

Parallel Programming Models

1. Shared Memory Model (without threads)

Processes/tasks share a common address space, which they read and write to asynchronously.



Implementations: On stand-alone shared memory machines, native operating systems, compilers and/ or hardware provide support for shared memory programming

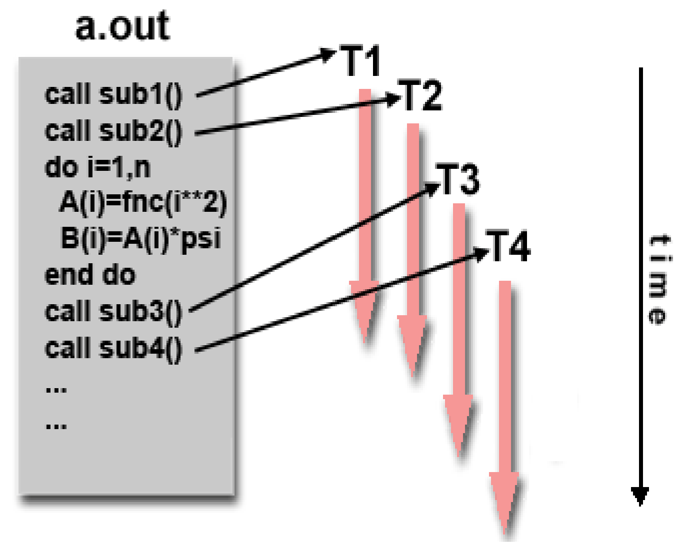
Parallel Programming Models

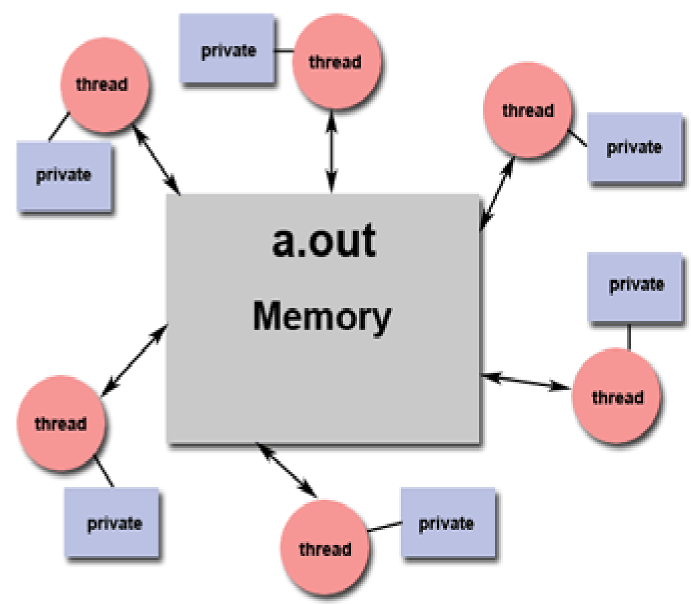
2. Threads Model

- shared memory programming.

Implementations:

- A library of subroutines that are called from within parallel source code.
- A set of compiler directives imbedded in either serial or parallel source code.

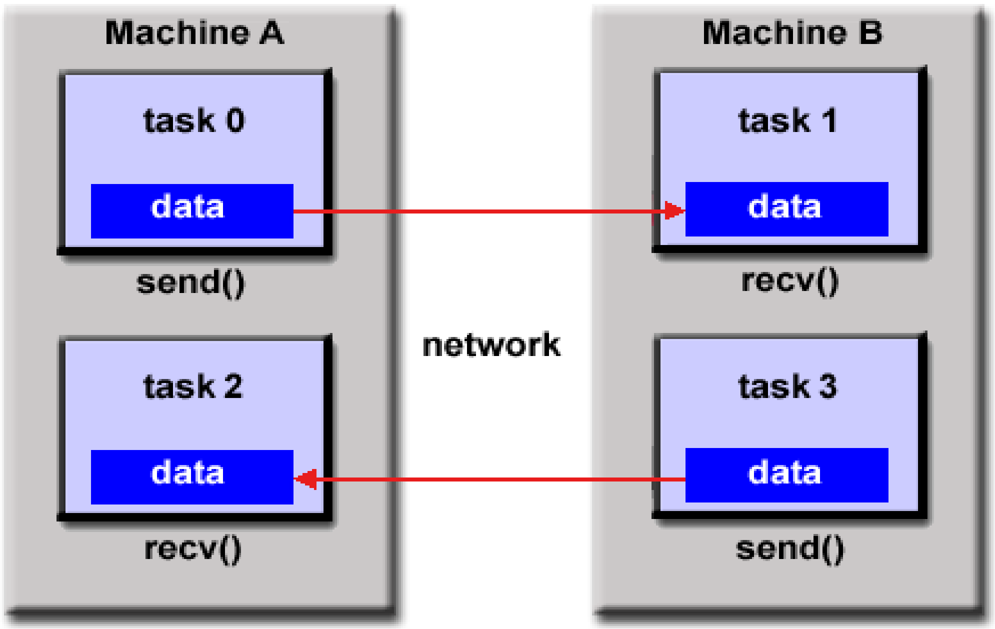




Parallel Programming Models

3. Distributed Memory / Message Passing Model

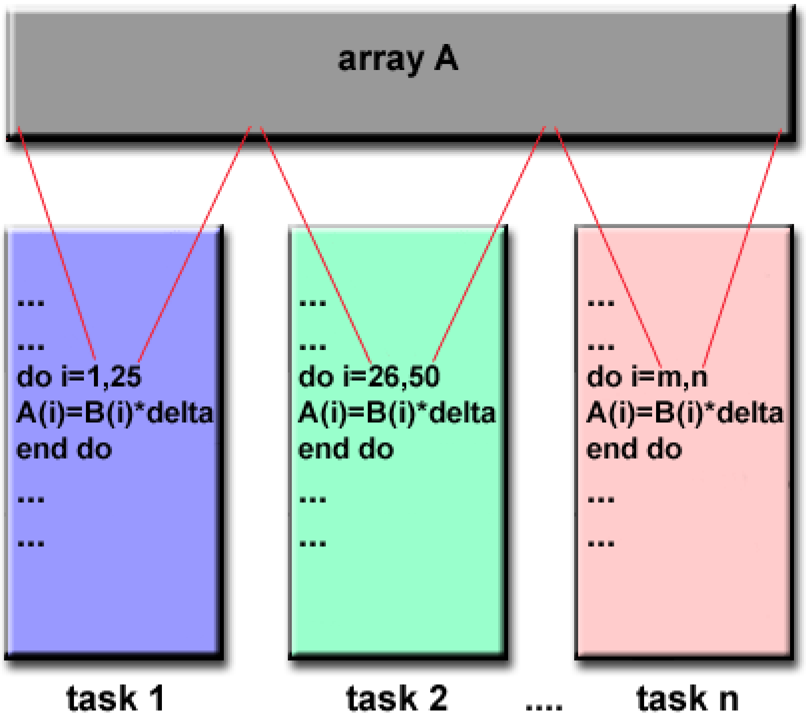
- Tasks that use their own local memory during computation.
- Data transfer usually requires cooperative operations.
- Data communications through sending and receiving messages.



Parallel Programming Models

4. Data Parallel Model (Partitioned Global Address Space -PGAS)

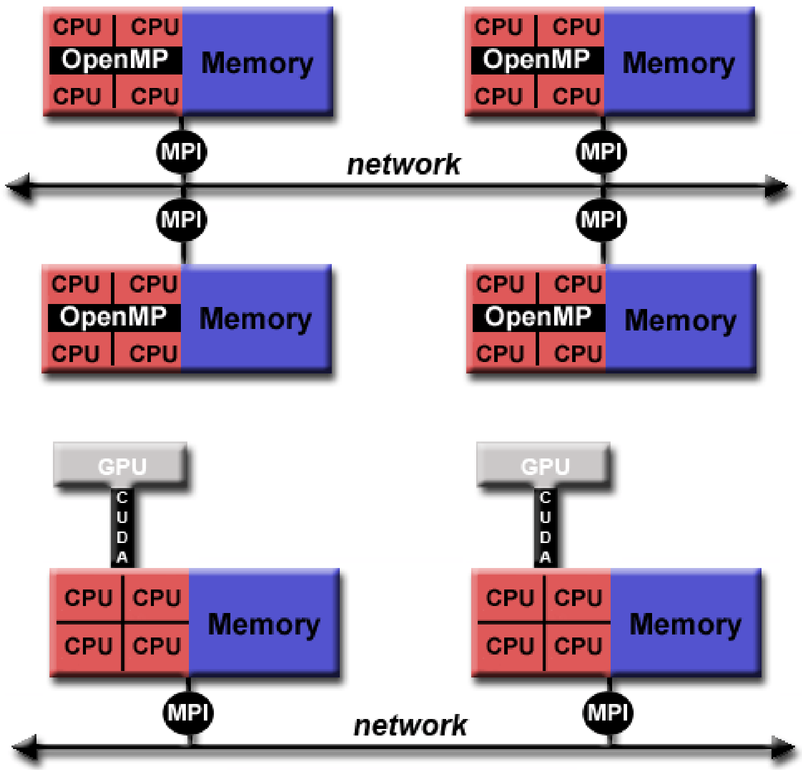
- Address space is treated globally
- Focuses on performing operations on a data set.
- Tasks perform the same operation on their partition of work.



Parallel Programming Models

5. Hybrid Model

- Threads perform computationally intensive kernels using local, on-node data.
- Communications between processes on different nodes occurs over the network using MPI.

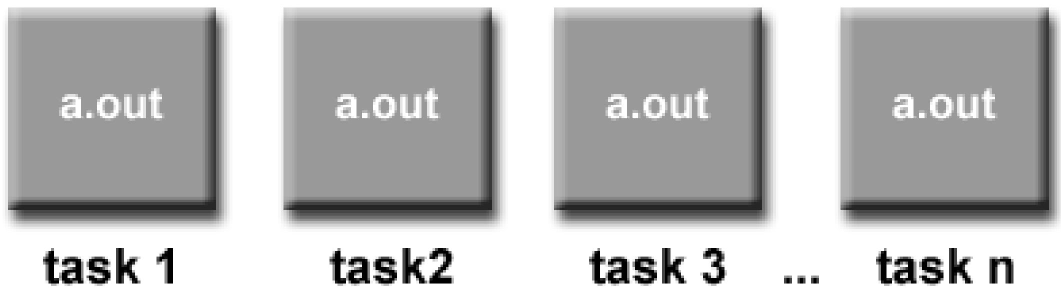


Lend to most popular (currently) hardware environment of clustered multi/many-core machines.

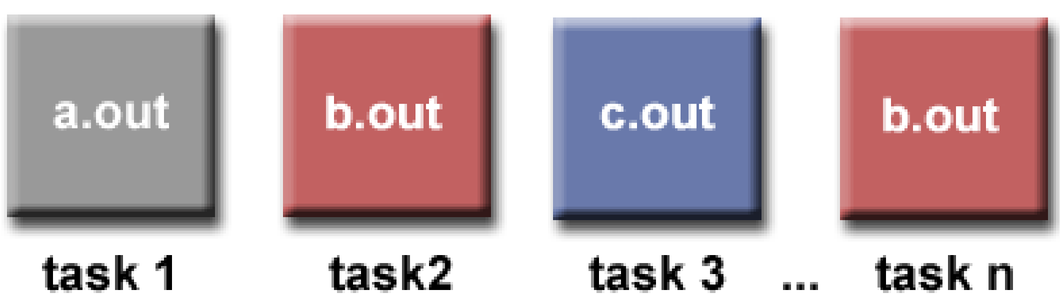
北京航空航天大學 COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

6. SPMD and MPMD

- **Single Program Multiple Data (SPMD):**



- **Multiple Program Multiple Data (MPMD):**



Designing parallel programs

1. Automatic vs. Manual Parallelization

- **Manual parallelization**: time consuming, complex, error-prone and ***iterative*** process.
- **Automatic parallelization:** ⇒ parallelizing compiler
  - **Fully Automatic**
    - compiler analyzes the source code and identifies opportunities for parallelism.
  - **Programmer Directed**
    - Using "compiler directives" or possibly compiler flags

2. Understand the Problem and the Program

1. **First step in developing parallel software is to first understand the problem.**

   If you are starting with a serial program, this necessitates understanding the existing code also.

2. **Determine whether or not the problem that can actually be parallelized.**

- A parallelizable problem: The calculation of the minimum energy conformation.
- a little-to-no parallelism problem : Calculation of the Fibonacci series (0,1,1,2,3,5,8,13,21,...) by use of the formula: $F(n) = F(n-1) + F(n-2)$

2. Understand the Problem and the Program

3. **Identify the program's *hotspots*.**

- Know where most of the real work is being done.
- Focus on parallelizing the hotspots, ignore those sections of the program that account for little CPU usage.

4. **Identify *bottlenecks* in the program.**

- disproportionately slow areas, or cause parallelizable work to halt or be deferred?
- reduce or eliminate

unnecessary slow areas.

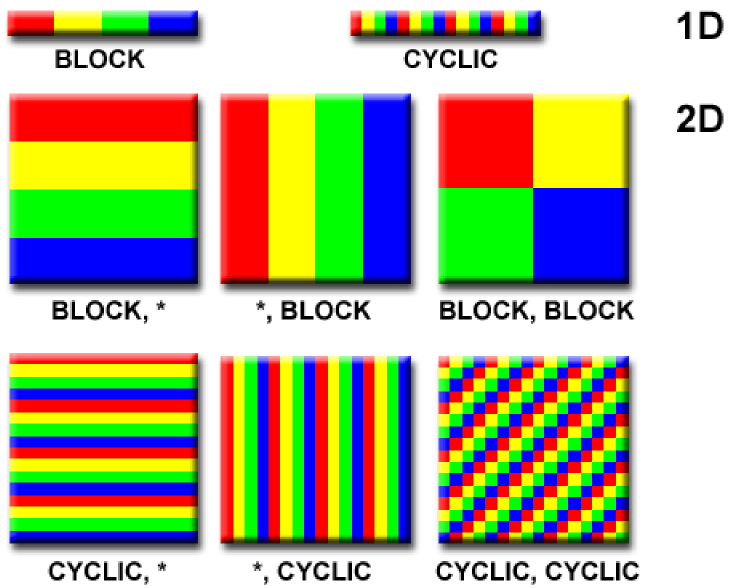
2. Understand the Problem and the Program

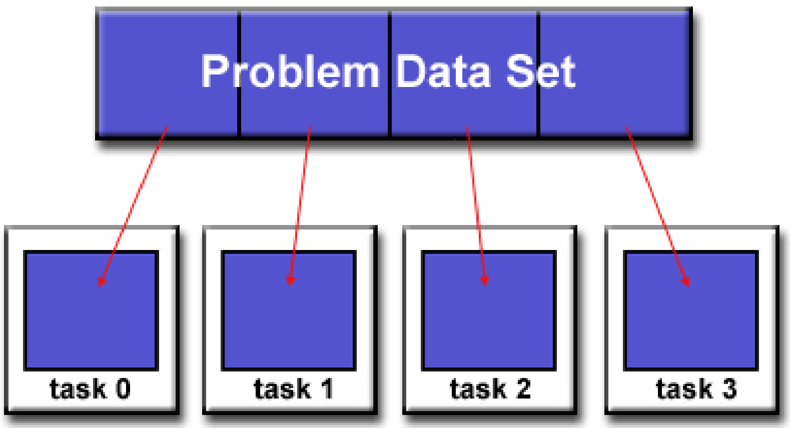
5. **Identify inhibitors to parallelism.**
   - One common class of inhibitor is *data dependence*
6. **Investigate other algorithms**
   - This may be the single most important consideration
7. **Take advantage of optimized third party parallel software.**
   - IBM's ESSL, Intel's MKL, AMD's AMCL, etc.

3. Partitioning

Domain Decomposition:

- the data associated with a problem is decomposed
- ways to partition data:

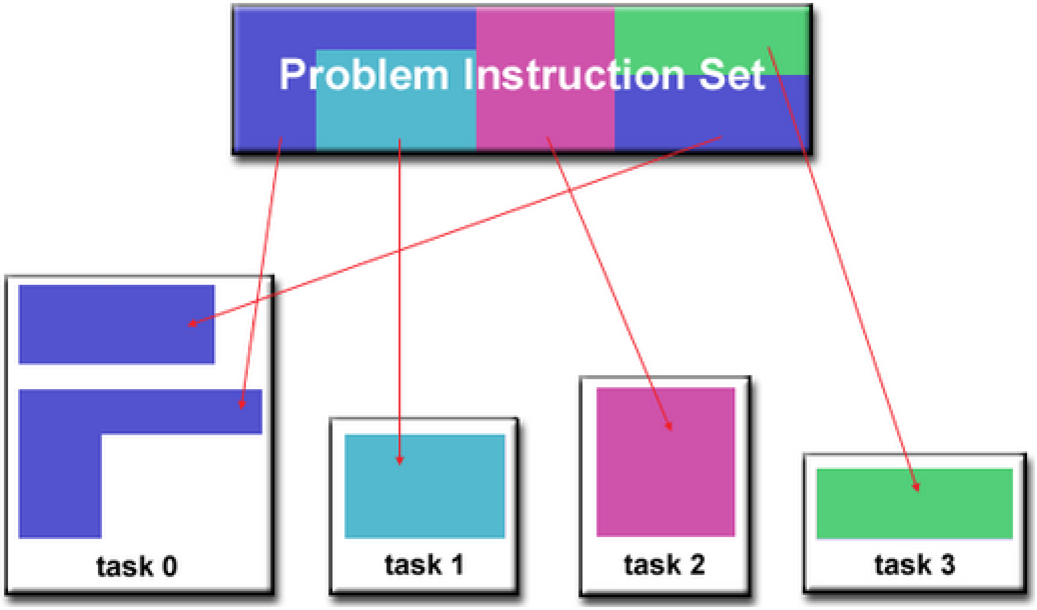




3. Partitioning

Functional Decomposition:

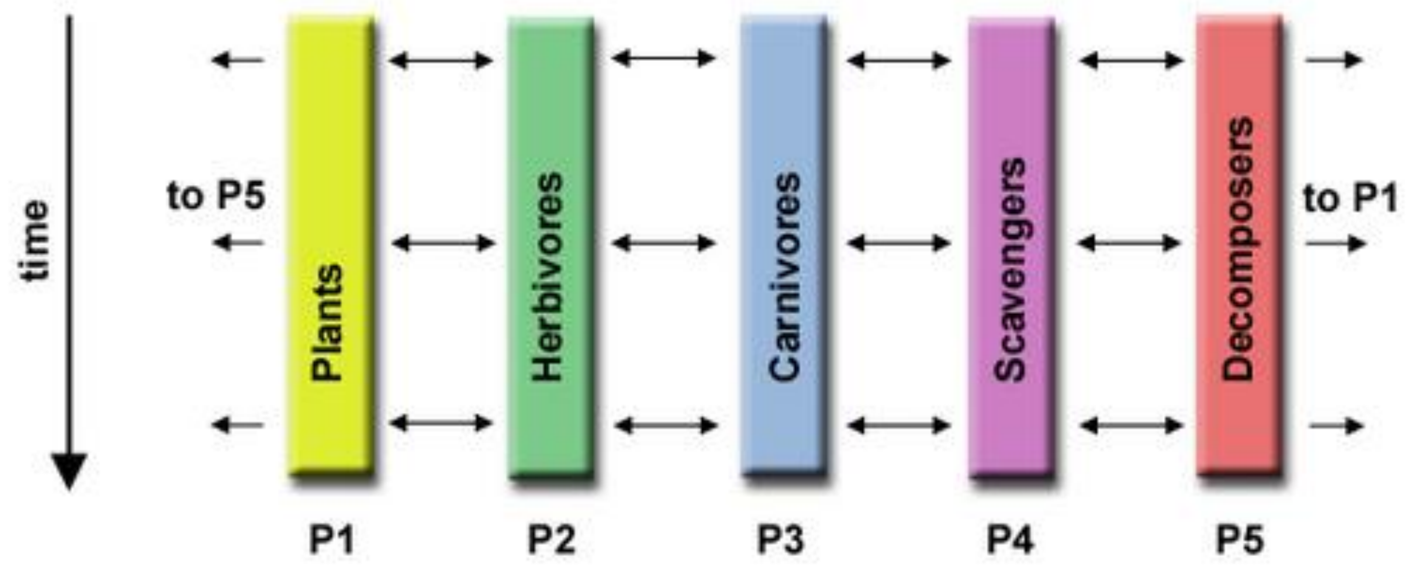
- Problem is decomposed according to the work that must be done.



3. Partitioning

Functional Decomposition:

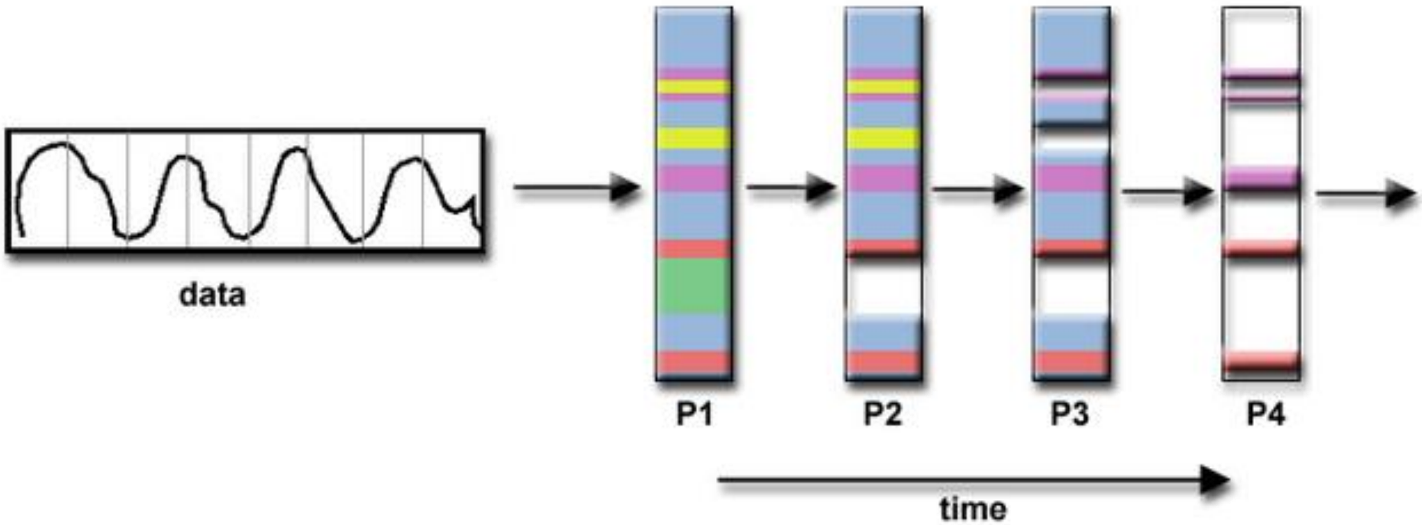
- Ecosystem Modeling



3. Partitioning

Functional Decomposition:

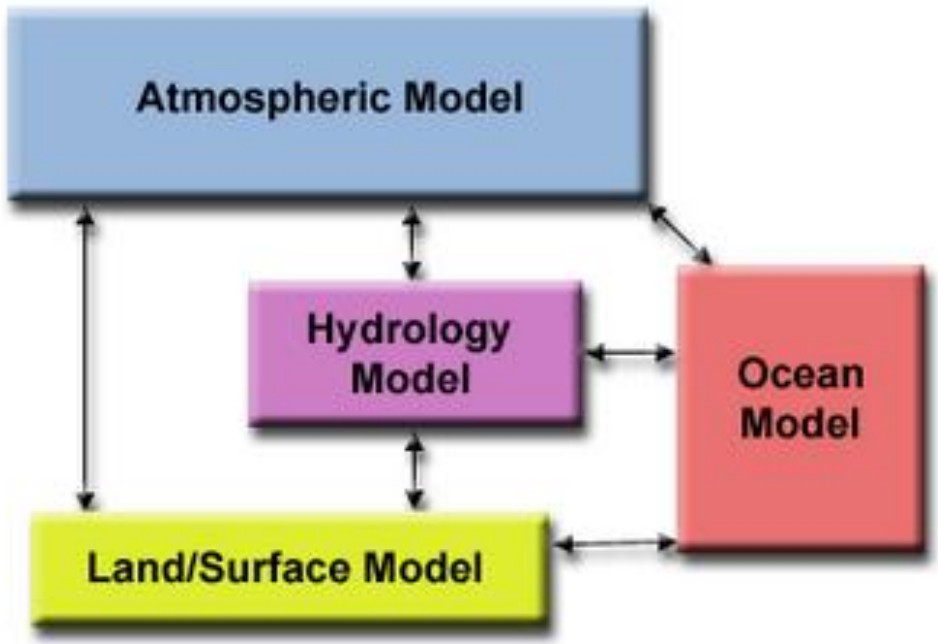
- Signal Processing



3. Partitioning

Functional Decomposition:

- Climate Modeling



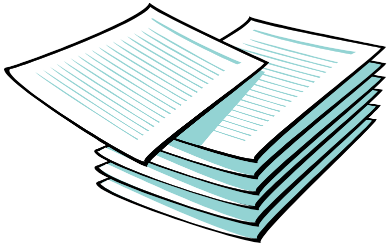
How do we write parallel programs?

- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it’s part of the data.

Professor P

15 questions

300 exams

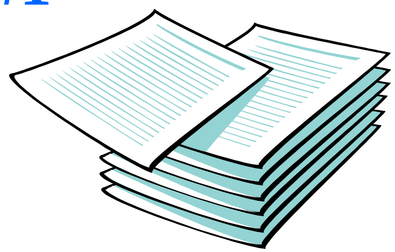


Professor P's grading assistants

北京航空航天大學
COLLEGE OF SOFTWARE
BEIHANG UNIVERSITY
软件学院

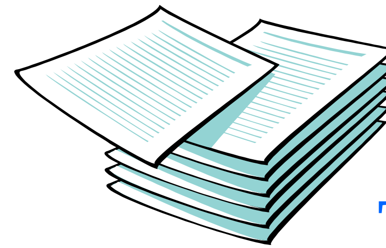
Division of work – data parallelism

TA#1

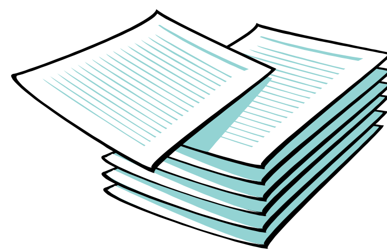


100 exams

TA#3



100 exams



TA#2

100 exams

Division of work – task parallelism

TA#1

Questions 1 - 5

Questions 11 - 15

TA#3

Questions 6 - 10

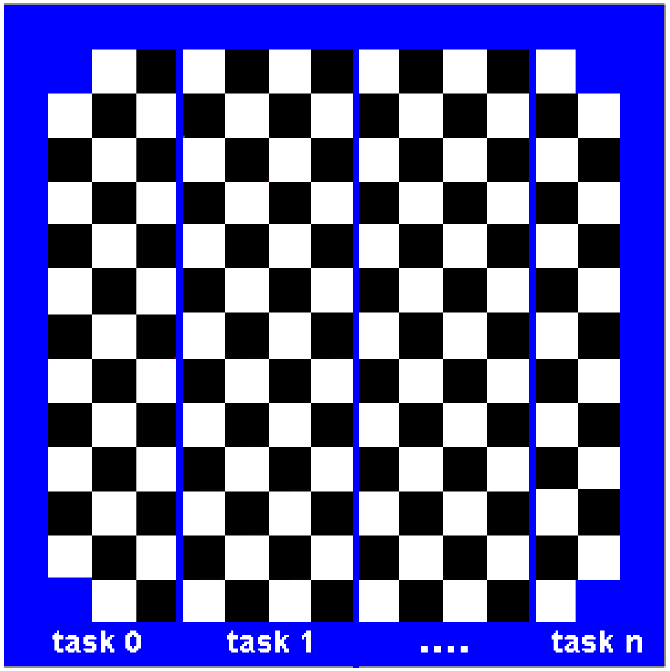
TA#2

4. Communications

Who Needs Communications?

You DON'T need communications:

- ***embarrassingly parallel*** - little or no communications are required , no need for tasks to share data.



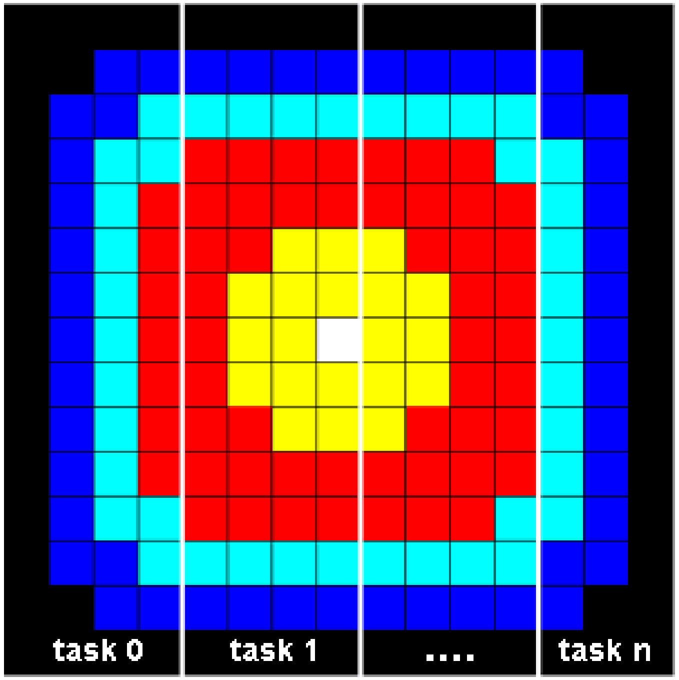
Eg. every pixel in a black and white image needs to have its color reversed.

4. Communications

Who Needs Communications?

You DO need communications:

- Most parallel applications are not quite so simple, and do require tasks to share data with each other.



Eg. 2-D heat diffusion problem

4. Communications

Factors to Consider:

- **Communication overhead**
- **Latency vs. Bandwidth**
- **Visibility of communications**
- **Synchronous vs. asynchronous communications**
  - Synchronous communications are often referred to as ***blocking*** communications
  - Asynchronous communications are often referred to as ***non-blocking*** communications

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

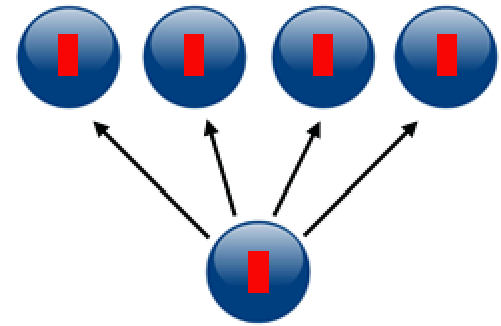
Designing parallel programs

4. Communications

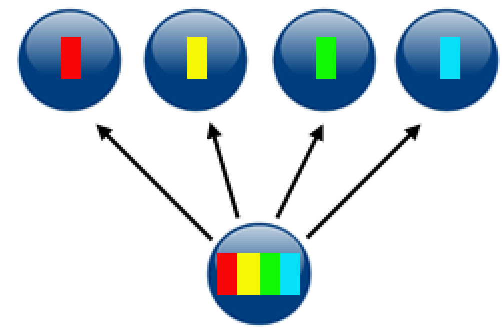
Factors to Consider:

- **Scope of communications**
  - *Point-to-point* - involves two tasks.
  - *Collective* - involves more than two tasks

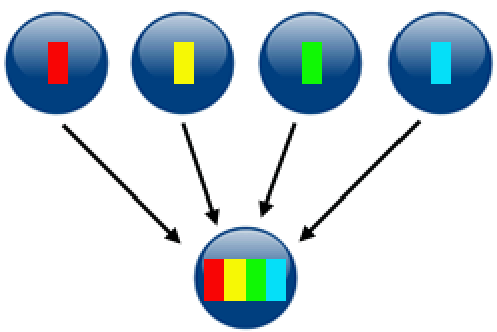
Eg.



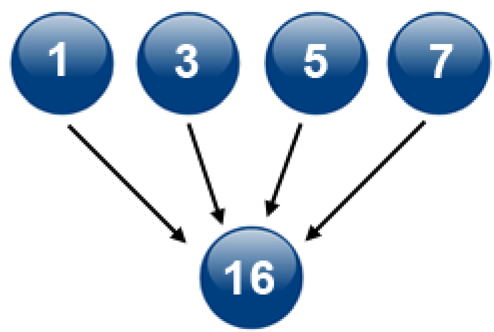
broadcast



scatter



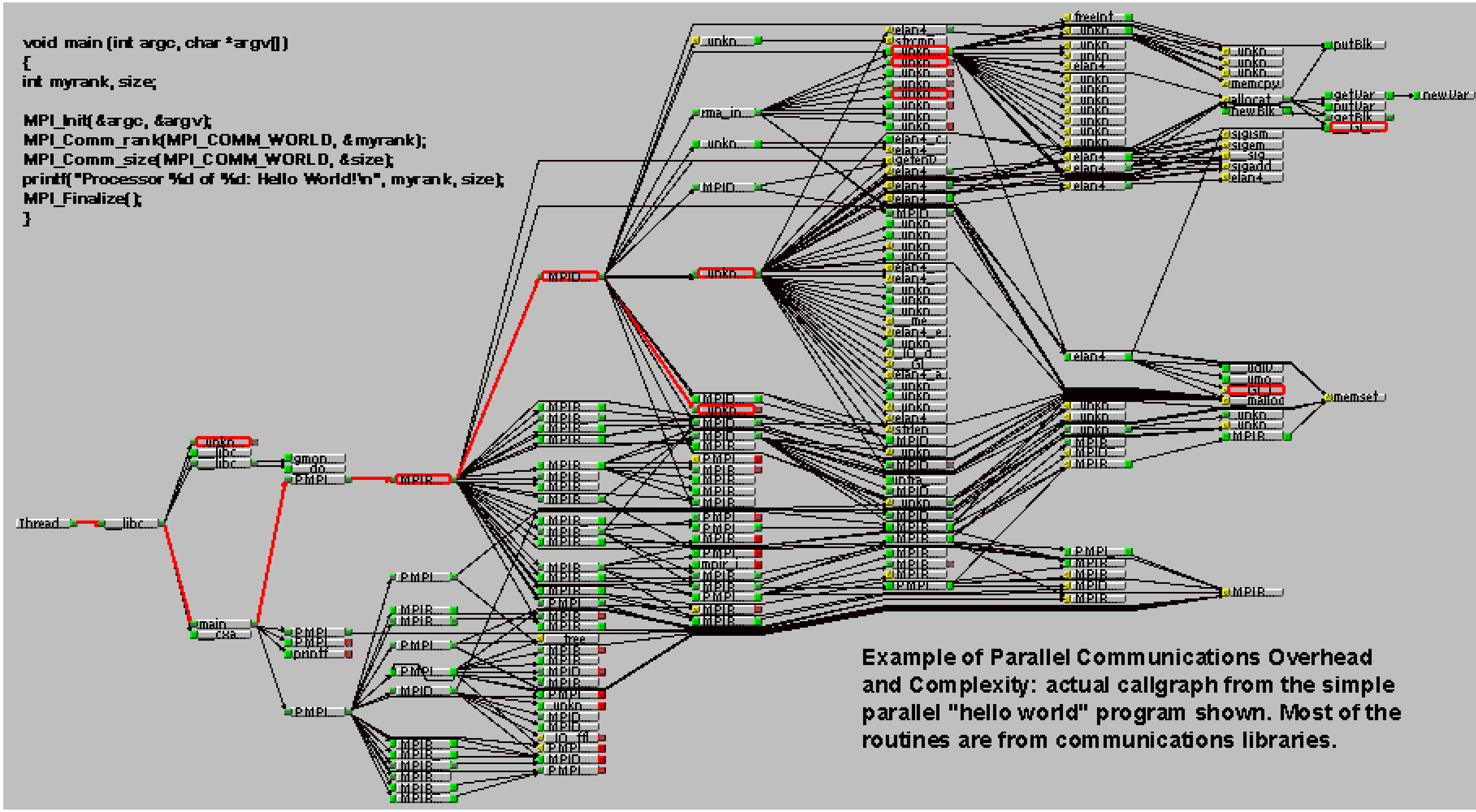
gather



reduction

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Designing parallel programs



Example of Parallel Communications Overhead and Complexity: actual callgraph from the simple parallel "hello world" program shown. Most of the routines are from communications libraries.

5. Synchronization

Types of Synchronization

- **Barrier:**

  Each task performs its work until it reaches the barrier. It then stops, or "blocks".

- **Lock / semaphore:**

  Typically used to serialize (protect) access to global data or a section of code.

- **Synchronous communication operations:**

  Involves only those tasks executing a communication operation

6. Data Dependencies

- **Definition:**
  - A ***dependence*** exists between program statements when the order of statement execution affects the results of the program.
  - A ***data dependence*** results from multiple use of the same location(s) in storage by different tasks.

6. Data Dependencies

- **Examples:**

Loop carried data dependence

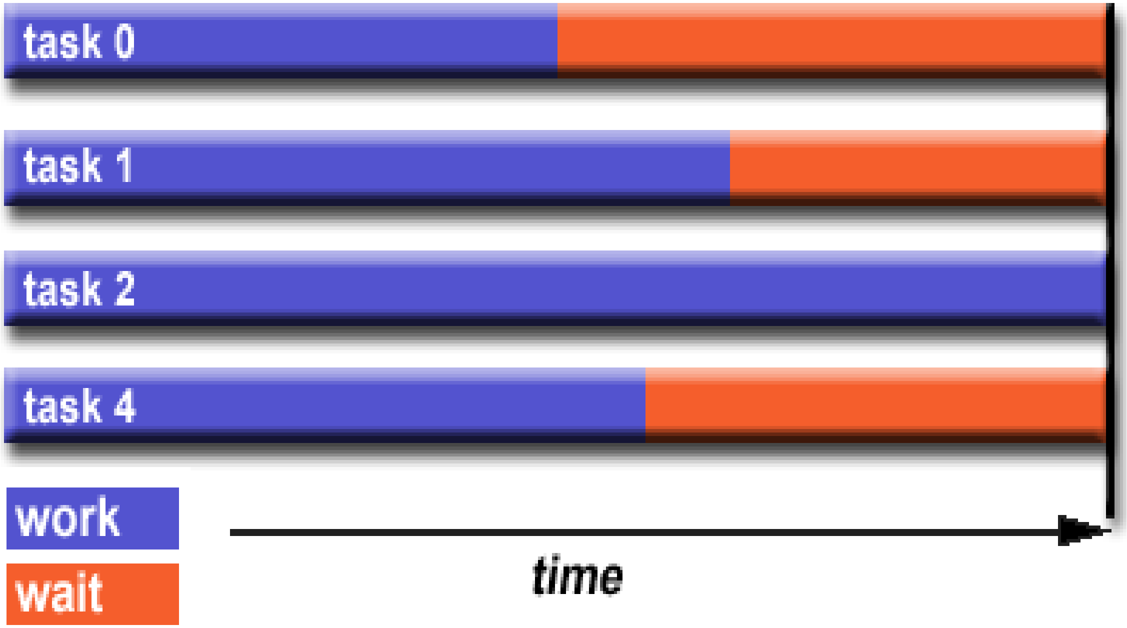
```
DO J = MYSTART, MYEND
   A (J) = A (J-1) * 2.0
END DO
```

Loop independent data dependence

```
task 1              task 2
------              ------

X = 2               X = 4
  .                   .
  .                   .
Y = X**2            Y = X**3
```

7. Load Balancing

- a minimization of task idle time.

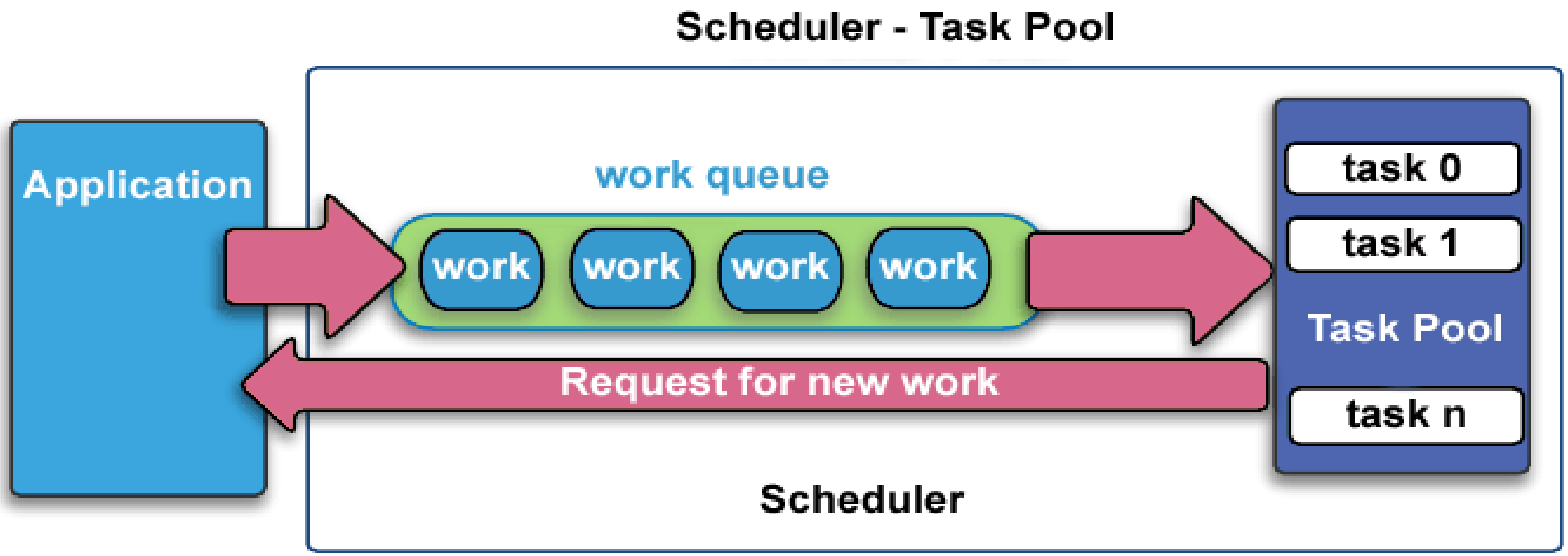


7. Load Balancing

- **How to Achieve Load Balance:**
  - **Equally partition the work each task receives**
  - evenly distribute the data set among the tasks.
  - evenly distribute the iterations across the tasks.
  - If a heterogeneous mix of machines with varying performance characteristics are being used, adjust work accordingly.

7. Load Balancing

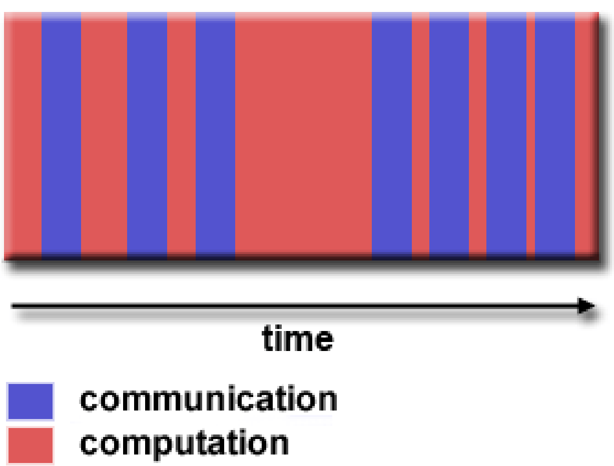
- **How to Achieve Load Balance:**
  - **Use dynamic work assignment**



scheduler-task pool approach

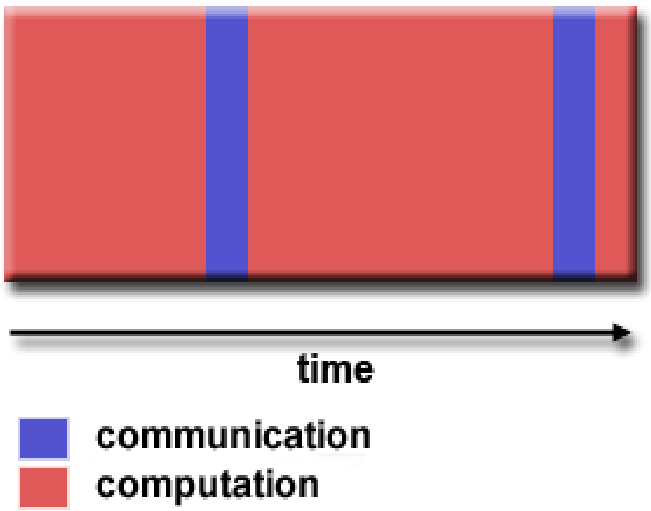
8. Granularity $\Rightarrow$ Computation / Communication Ratio

- **Fine-grain Parallelism:**
  - Low computation to communication ratio
  - Facilitates load balancing
  - high communication overhead and less performance enhancement



8. Granularity

- **Coarse-grain Parallelism:**
  - High computation to communication ratio
  - Implies more opportunity for performance increase
  - Harder to load balance efficiently



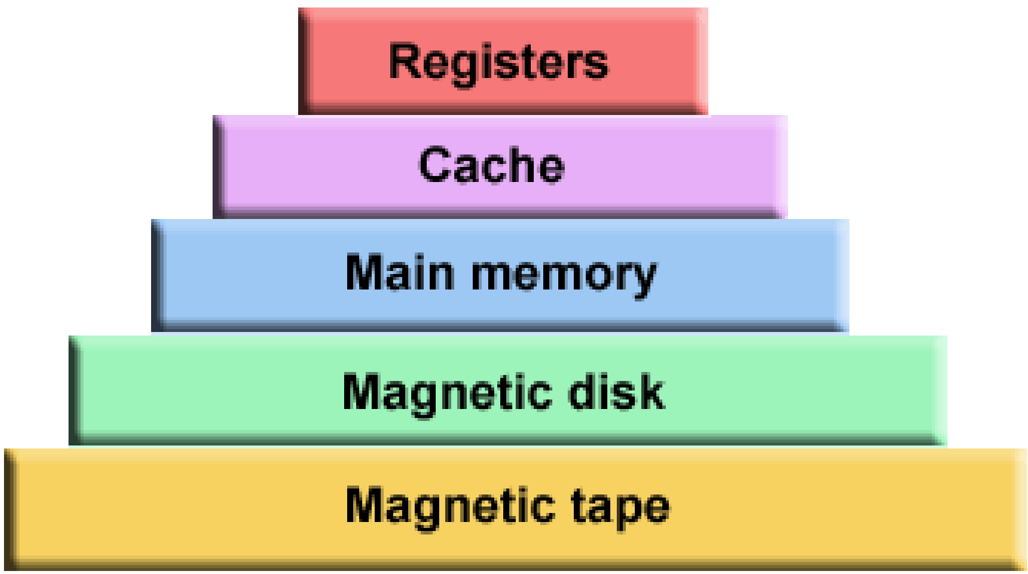
Which is Best? (Fine-grain & Coarse-grain)

- The overhead associated with communications and synchronization is high relative to execution speed, so it is advantageous to have *coarse granularity*.
- *Fine-grain* parallelism can help reduce overheads due to load imbalance.

9. I/O

- **The Bad News:**

Memory Hierarchy



1 ns	1x
10 ns	10x
100 ns	100x
100 ms	100,000,000x
10 s	1e+10x

北京航空航天大學
COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

9. I/O

- **The Good News:**
  - Parallel file systems are available
    - GPFS : General Parallel File System (IBM)
    - Lustre : for Linux clusters (Intel)
    - HDFS : Hadoop Distributed File System (Apache)
    - PanFS : Panasas ActiveScale File System for Linux clusters (Panasas, Inc.)
  - The parallel I/O programming interface specification for MPI is available.

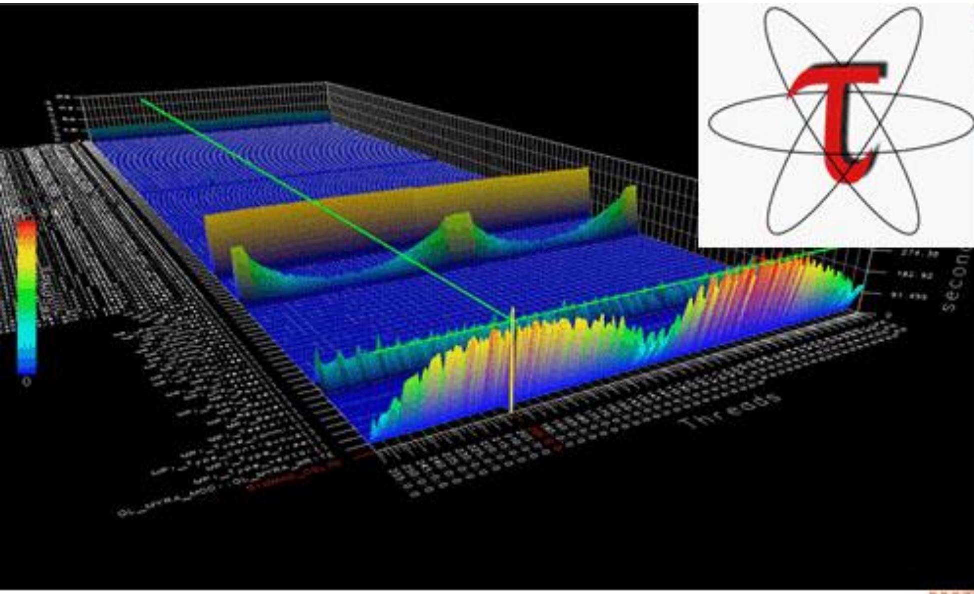
9. I/O

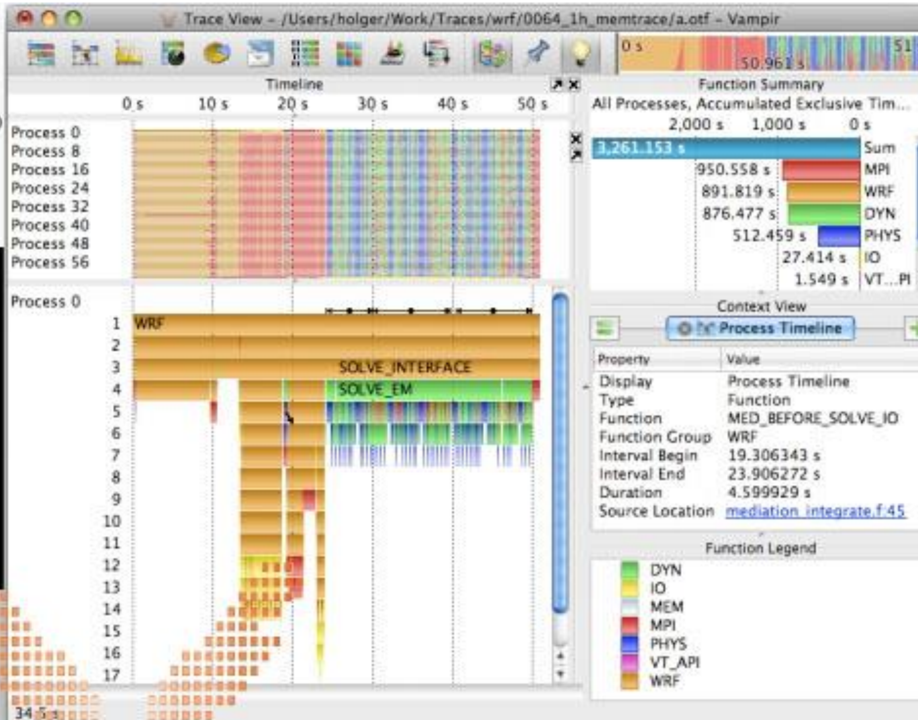
- **A few pointers:**
  - Reduce overall I/O as much as possible.
  - If you have access to a parallel file system, use it.
  - Writing large chunks of data rather than small.
  - Fewer, larger files performs better than many small files.
  - Combine I/O operations across tasks.
  - Confine I/O to specific serial portions of the job, and then use parallel communications to distribute data to parallel tasks.

10.Debugging

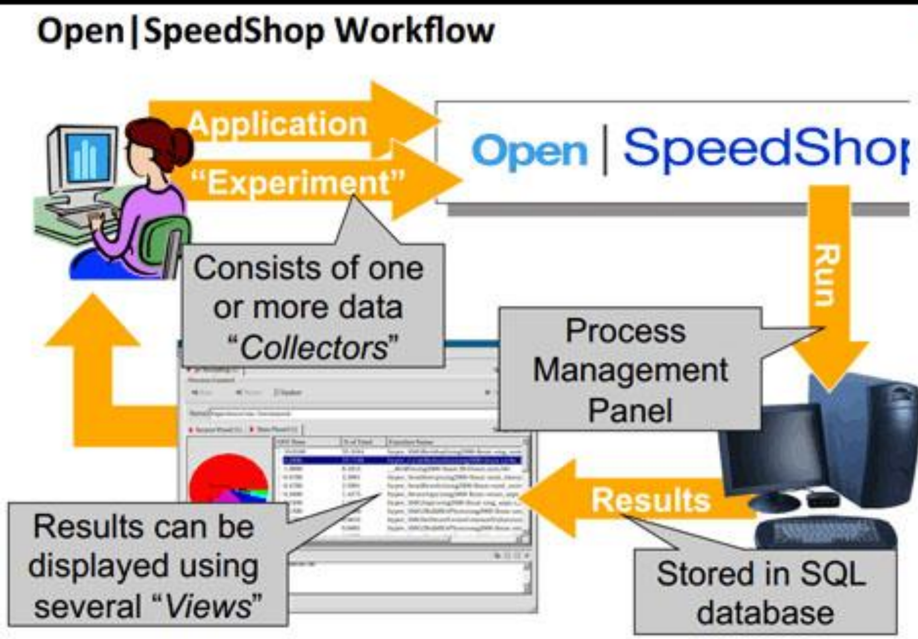
- can be difficult, particularly as codes scale upwards.
- some excellent debuggers available to assist:
  - Threaded - pthreads and OpenMP
  - MPI
  - GPU / accelerator
  - Hybrid
- For example:
  - TotalView from RogueWave Software
  - Stack Trace Analysis Tool (STAT)
  - DDT from Allinea
  - Inspector from Intel

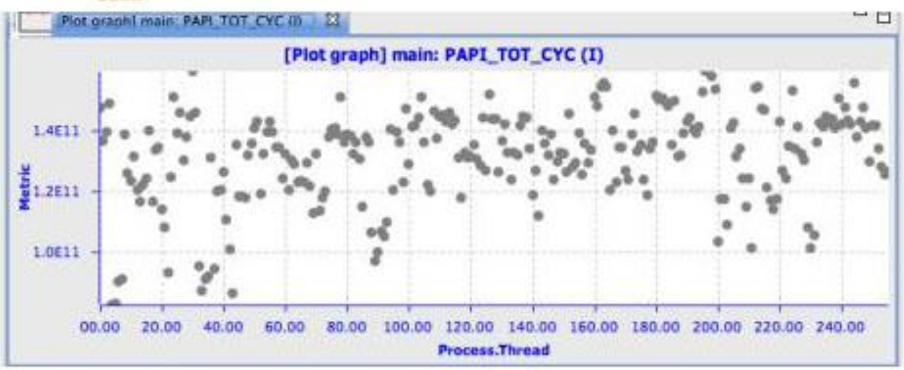
Designing parallel programs





VAMPIR

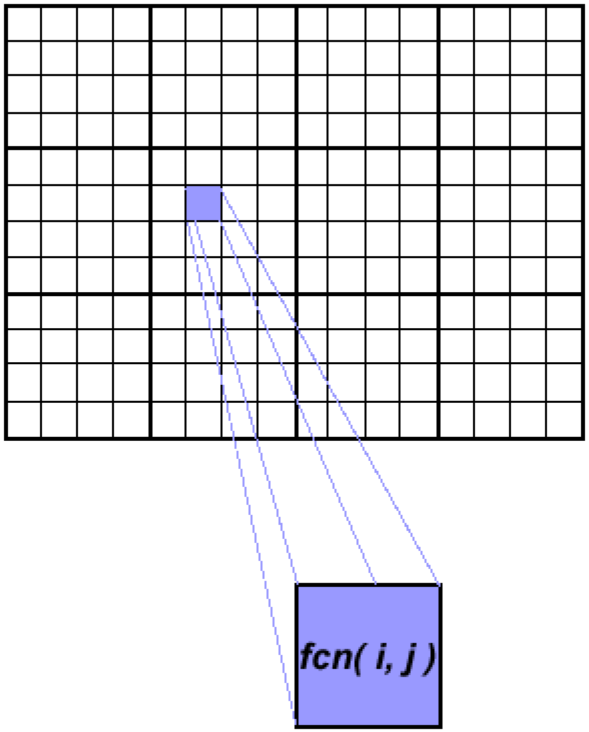




Parallel Examples

Parallel Examples

1. Array Processing



- The computation on each array element is independent from other array elements.
- The problem is computationally intensive.
- The serial program calculates one element at a time in sequential order
- Serial code could be of the form:

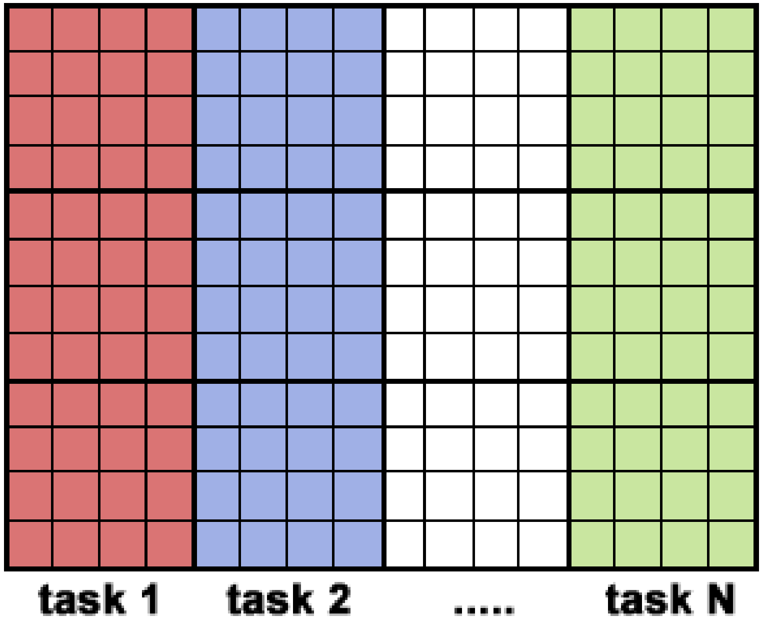
```
do j = 1, n
   do i = 1, n
      a(i, j) = fcn(i, j)
   end do
end do
```

A calculations on 2-dimensional array elements, a function is evaluated on each array element.

Parallel Examples

1. Array Processing

- **Parallel Solution 1**
  - The calculation of elements is independent.
  - Arrays elements are evenly distributed.
  - Independent calculation ensures there is no need for communication or synchronization between tasks.
  - There should not be load balance concerns.
  - After the array is distributed, each task executes the loop corresponding it owns.



1. Array Processing

- **One Possible Solution:**
  - Implement as a Single Program Multiple Data (SPMD) model - every task executes the same program.
  - Master process initializes array, sends info to worker processes and receives results.
  - Worker process receives info, performs its share of computation and sends results to master.
  - Using the Fortran storage scheme, perform block distribution of the array.

Parallel Examples

1. Array Processing

- **One Possible Solution:**

Pseudo code:

red highlights changes for parallelism.

```
find out if I am MASTER or WORKER

if I am MASTER

  initialize the array
  send each WORKER info on part of array it owns
  send each WORKER its portion of initial array

  receive from each WORKER results

else if I am WORKER
  receive from MASTER info on part of array I own
  receive from MASTER my portion of initial array

  # calculate my portion of array
  do j = my first column,my last column
    do i = 1,n
      a(i,j) = fcn(i,j)
    end do
  end do

  send MASTER results

endif
```

COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

1. Array Processing

- **Parallel Solution 2: Pool of Tasks**

  Solve load balance problem.

- **Pool of Tasks Scheme:**
  - Master Process:
    - Holds pool of tasks for worker processes to do
    - Sends worker a task when requested
    - Collects results from workers
  - Worker Process: repeatedly does the following
    - Gets task from master process
    - Performs computation
    - Sends results to master

Parallel Examples

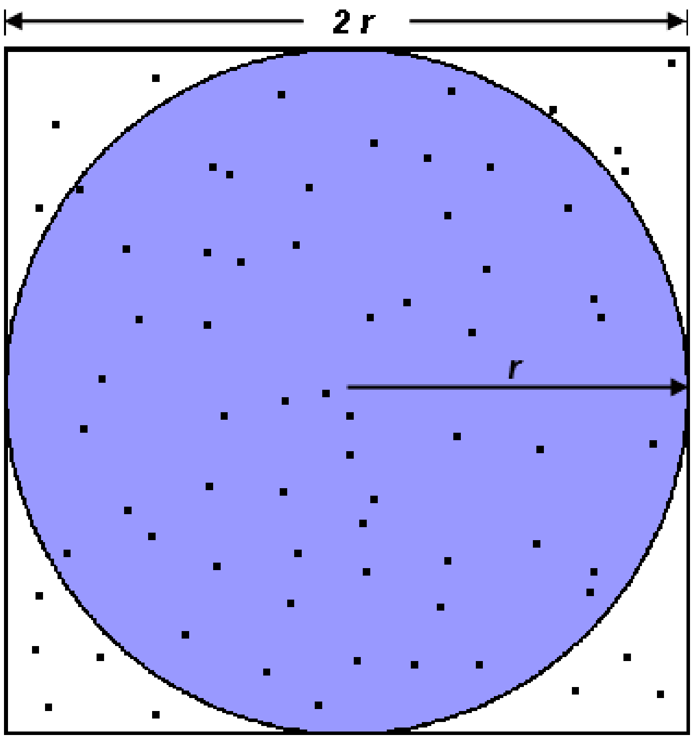
1. Array Processing

- **Pool of Tasks Scheme:**
  - The faster tasks will get more work to do
  - Pseudo code solution: **red** highlights changes for parallelism.

```
find out if I am MASTER or WORKER

if I am MASTER

  do until no more jobs
    if request send to WORKER next job
    else receive results from WORKER
  end do

else if I am WORKER

  do until no more jobs
    request job from MASTER
    receive from MASTER next job

    calculate array element: a(i,j) = fcn(i,j)

    send results to MASTER
  end do

endif
```

2. PI Calculation

- The value of PI can be calculated in various ways.



$A_S = (2r)^2 = 4r^2$

$A_C = \pi r^2$

$\pi = 4 \times \frac{A_C}{A_S}$

Monte Carlo method

Serial pseudo code

```
npoints = 10000
circle_count = 0

do j = 1,npoints
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

PI = 4.0*circle_count/npoints
```

Parallel Examples

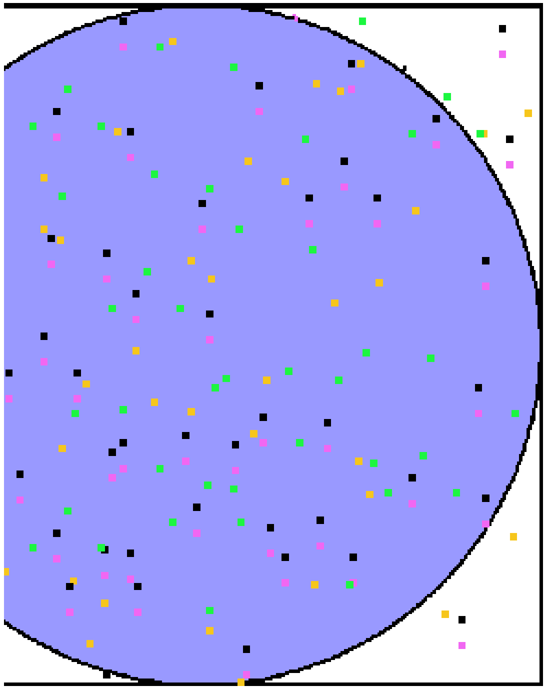
```
npoints = 10000
circle_count = 0

p = number of tasks
num = npoints/p

find out if I am MASTER or WORKER

do j = 1,num
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

if I am MASTER

  receive from WORKERS their circle_counts
  compute PI (use MASTER and WORKER calculations)

else if I am WORKER

  send to MASTER circle_count

endif
```

llelize:

:

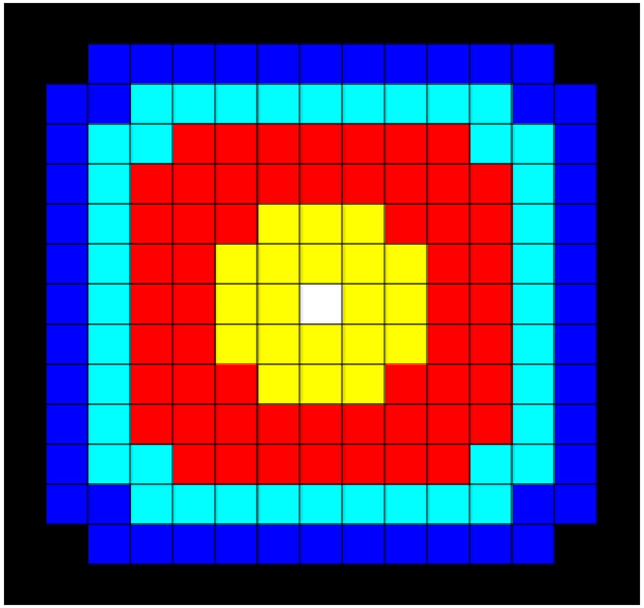
balance concerns

tasks.



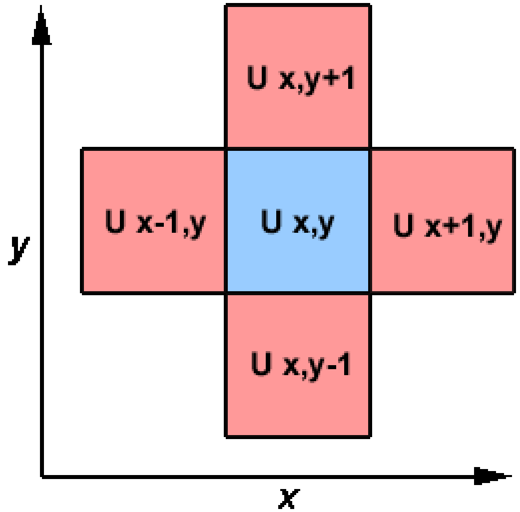
3. Simple Heat Equation

- Describes the temperature change over time, given initial temperature distribution and boundary conditions.
- A finite differencing scheme is employed to solve the heat equation numerically on a square region.



3. Simple Heat Equation

- The calculation of an element is ***depenent*** upon neighbor element values:



$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$

```
do iy = 2, ny - 1
  do ix = 2, nx - 1
    u2(ix, iy) =  u1(ix, iy)  +
        cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy)) +
        cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))
  end do
end do
```

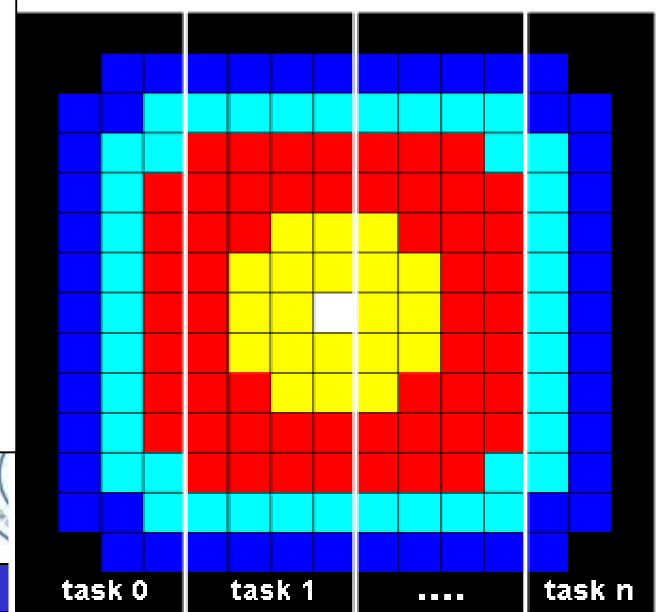
3. Simple Heat Equation

```
find out if I am MASTER or WORKER

if I am MASTER
  initialize array
  send each WORKER starting info and subarray
  receive results from each WORKER

else if I am WORKER
  receive from MASTER starting info and subarray

  # Perform time steps
  do t = 1, nsteps
    update time
    send neighbors my border info
    receive from neighbors their border info
    update my portion of solution array

  end do

  send MASTER results

endif
```

uted as subarrays to

er tasks

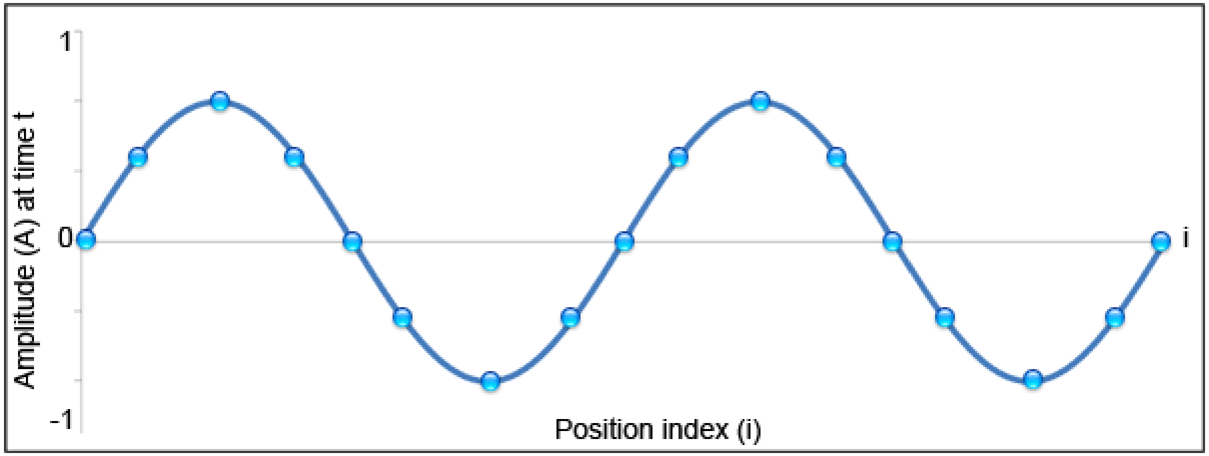
eighbor task's data,



4. 1-D Wave Equation

- the amplitude along a uniform, vibrating string is calculated after a specified amount of time has elapsed.

  the amplitude on the y axis，
  i as the position index along the x axis，
  node points imposed along the string，
  update of the amplitude at discrete time steps.



4. 1-D Wave Equation

The equation to be solved is the one-dimensional wave equation:

```
A(i,t+1) = (2.0*A(i,t)) - A(i,t-1) + (c*(A(i-1,t) - (2.0*A(i,t)) + A(i+1,t)))
```

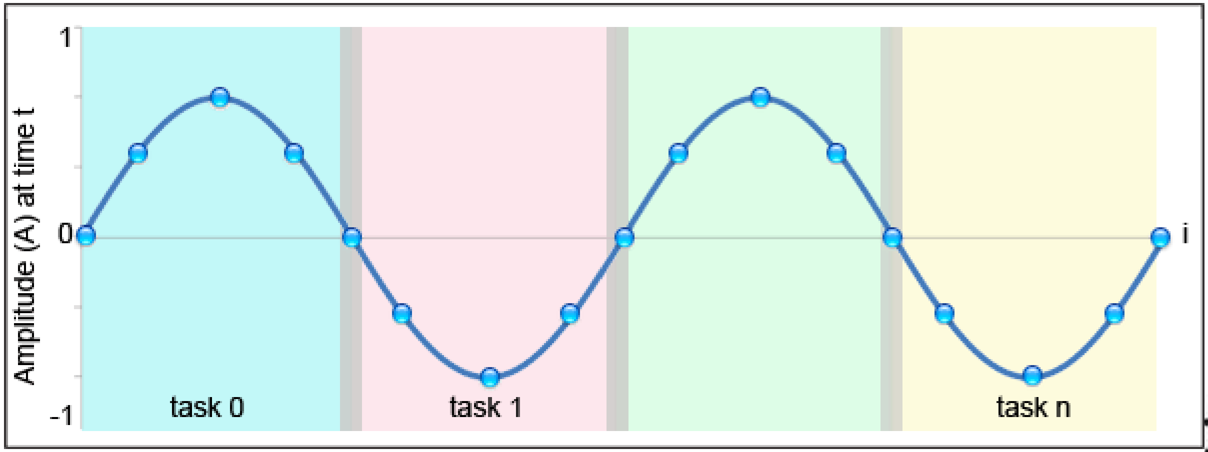
where c is a constant

4. 1-D Wave Equation

Parallel Solution

- Involve communications and synchronization.
- Each task owns an equal portion of the total array.
- Load balancing
- Communication need only occur on data borders.

Pseudo code solution



Parallel Examples

```
find out number of tasks and task identities

#Identify left and right neighbors
left_neighbor = mytaskid - 1
right_neighbor = mytaskid +1
if mytaskid = first then left_neigbor = last
if mytaskid = last then right_neighbor = first

find out if I am MASTER or WORKER
if I am MASTER
  initialize array
  send each WORKER starting info and subarray
else if I am WORKER`
  receive starting info and subarray from MASTER
endif

#Perform time steps
#In this example the master participates in calculations
do t = 1, nsteps
  send left endpoint to left neighbor
  receive left endpoint from right neighbor
  send right endpoint to right neighbor
  receive right endpoint from left neighbor

  #Update points along line
  do i = 1, npoints
    newval(i) = (2.0 * values(i)) - oldval(i)
    + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
  end do

end do
```

```
#Collect results and write to file
if I am MASTER
  receive results from each WORKER
  write results to file
else if I am WORKER
  send results to MASTER
endif
```

北京航空航天大學 COLLEGE OF SOFTWARE BEIHANG UNIVERSITY 软件学院

Thank you！

https://computing.llnl.gov/tutorials/parallel_comp

北京航空航天大學
COLLEGE OF SOFTWARE
BEIHANG UNIVERSITY
软件学院