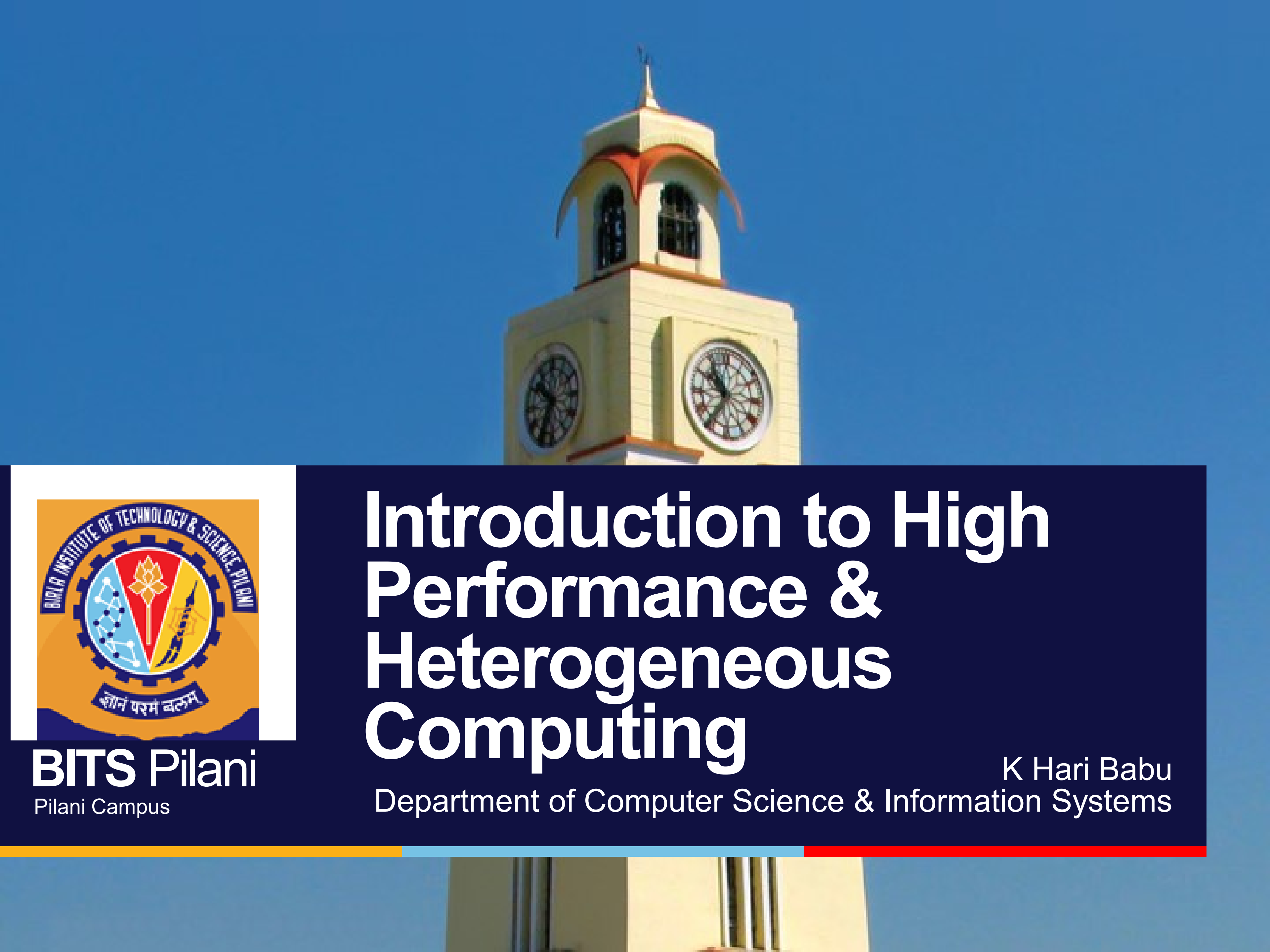


BITS Pilani
Pilani Campus

Introduction to High Performance & Heterogeneous Computing

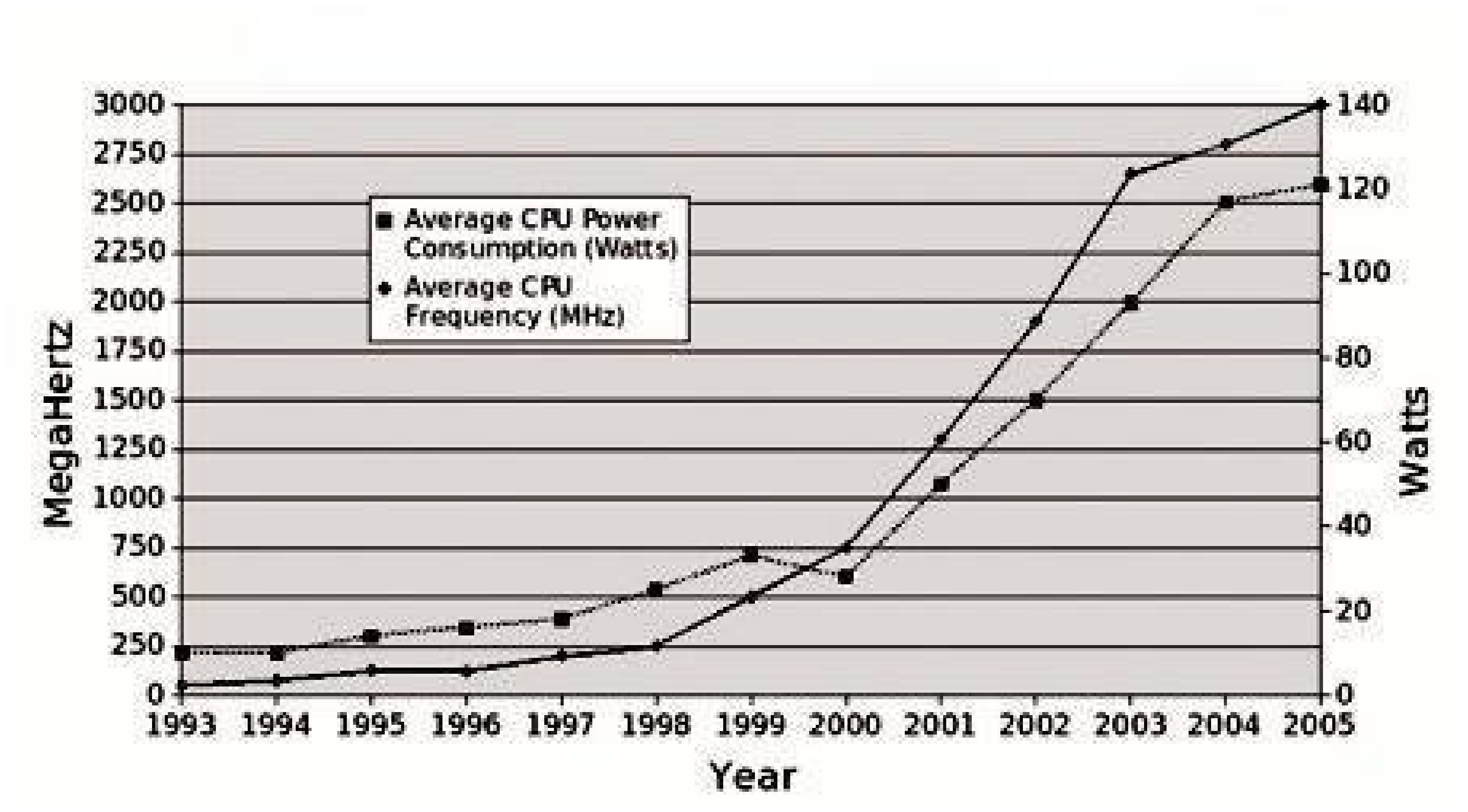
K Hari Babu
Department of Computer Science & Information Systems

Computing Speed

- Floating point operations per second (FLOPS, flops or flop/s) is a measure of computer performance
  - FLOPS = no of cores x cycles/second x FLOPS/cycle
- Intel Pentium 4 was a single core processor with 3.5 GHz. Its FP64 flops per cycle is 2. FLOPS = 1 x 3.5G x 2= 7 GFLOPS
- The ENIAC (Electronic Electronic Numerical Numerical Integrator Integrator And Computer) Computer (1946) had 350 FLOPS.
- This relentless drive for performance improvement from 1950s to early 2000s allowed application software to provide more functionality, have better user interfaces, and generate more useful results
  - since 2003, energy consumption and heat dissipation issues limited clock speed

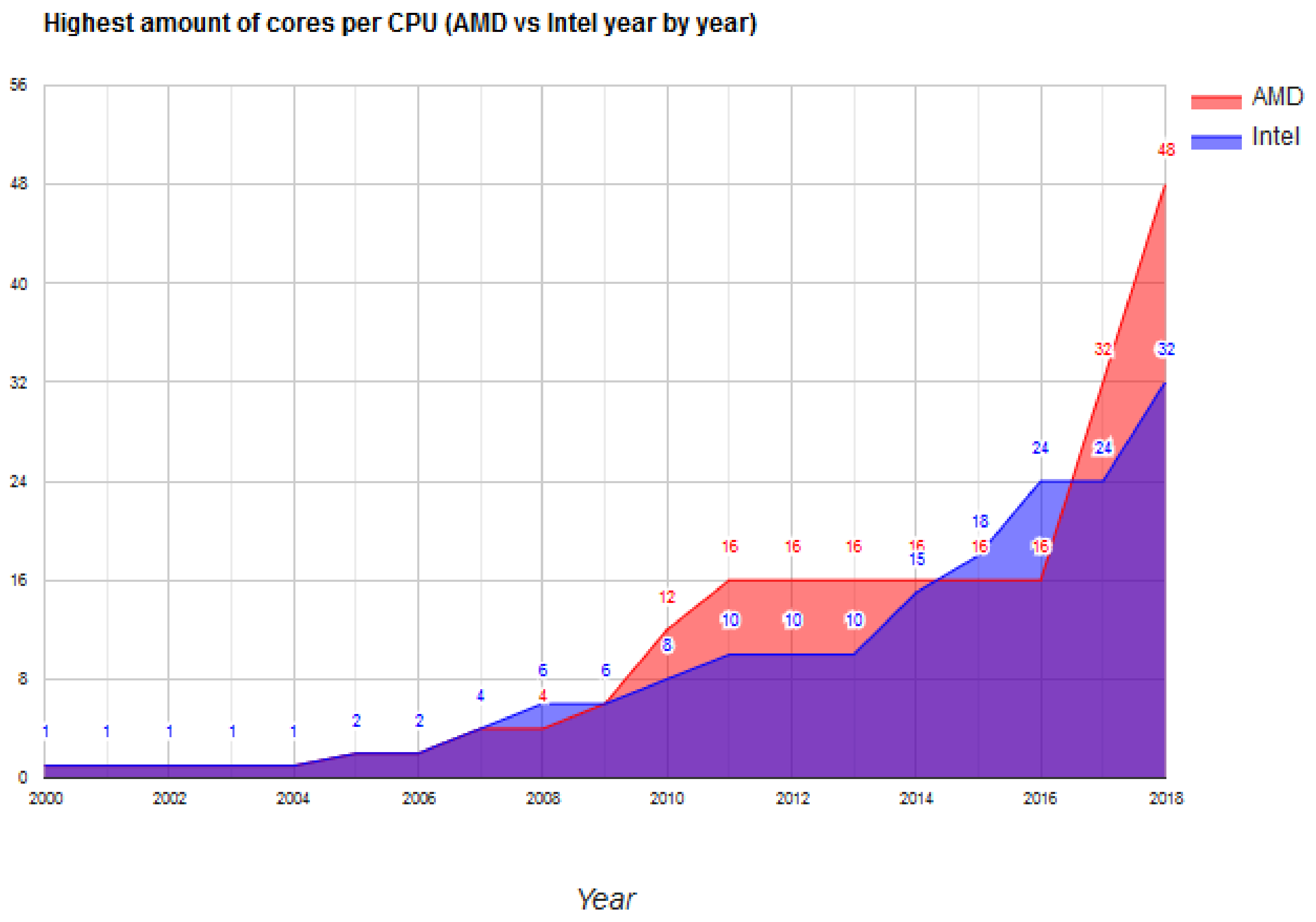
Heat problem

BITS Pilani



Number of cores per CPU

BITS Pilani



Multicore Processors

BITS Pilani

- All microprocessor vendors have switched to models multi-core models
  - where multiple processing units, referred to as processor cores, are used in each chip to increase the processing power
- This switch has exerted a tremendous impact on the software developer community
  - vast majority of software applications are written as sequential programs which can run on single core only
  - The expectation that by using advanced processor, performance increases is no longer valid for sequential programs
- Therefore approach to programming has to be parallel

Evolution of Computing

- Phase 1 (1950s): sequential instruction execution
- Phase 2 (1960s): sequential instruction issue
  - Pipeline execution, reservations stations
  - Instruction Level Parallelism (ILP)
- Phase 3 (1970s): vector processors
  - Pipelined arithmetic units
  - Registers, multi-bank (parallel) memory systems
- Phase 4 (1980s): SIMD and SMPs
  - Processor arrays
- Phase 5 (1990s): MPPs and clusters
  - Communicating sequential processors
- Phase 6 (>2000): many cores, accelerators, scale, …

What is a parallel computer?

BITS Pilani

- A parallel computer is a type of computer that uses multiple processing elements, such as processors or cores, to perform computations simultaneously
- These processing elements work together to solve a problem or perform a task faster than a single processing element would be able to on its own
- Parallel processing includes techniques and technologies that make it possible to compute in parallel
  - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, …

Concurrency

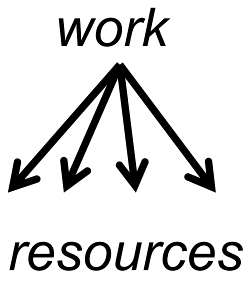
- Tasks are concurrent with respect to each if
  - They can execute at the same time (concurrent execution)
  - Implies that there are no dependencies between the tasks
- Dependencies
  - If a task requires results produced by other tasks in order to execute correctly, the task's execution is dependent
  - If two tasks are dependent, they are not concurrent
  - Some form of synchronization must be used to enforce (satisfy) dependencies
- Concurrency is fundamental to computer science
  - Operating systems, databases, networking, …

Concurrency and Parallelism

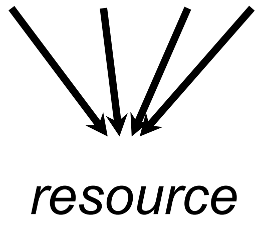
- Concurrent is not the same as parallel
- Parallel execution
  - Concurrent tasks actually execute at the same time i.e. not logicallybut physically multiple tasks execute in parallel
  - For that multiple (processing) resources have to be available
- Parallelism = concurrency + “parallel” hardware
  - Both are required
  - Find concurrent execution opportunities
  - Develop application to execute in parallel
  - Run application on parallel hardware
- Concurrency is a necessary condition for parallelism

Parallelism vs. Concurrency

- Parallelism:
  - Use extra resources to solve a problem faster



- Concurrency:
  - Correctly and efficiently manage access to shared resources



Parallelism

- There are granularities of parallelism (parallel execution) in programs
  - Processes, threads, routines, statements, instructions, …
- These must be supported by hardware resources
  - Processors, cores, … (execution of instructions)
  - Memory, DMA, networks, … (other associated operations)

Parallel Computing vs Distributed Computing

- Both try solving a problem with multiple processing elements.
- Main differences:
  - The level of communication and coordination between the processing elements or nodes.
    - In parallel computing, the processing elements are tightly coupled and share a common memory space. They communicate with each other frequently and quickly, and are often part of the same computer or device.
    - In distributed computing, on the other hand, the nodes are typically loosely coupled and have their own memory and processing power. They communicate over a network, and the communication is generally slower and less frequent.
  - Parallel computing is mainly used to increase the performance of a single application by dividing the work among multiple processors, while distributed computing is used to build large-scale systems that can provide services and share resources over a network.

Parallel Computing vs Distributed Computing

- Lines are blurring, but broadly:
  - Tightly coupled vs. Loosely coupled
  - Shared memory vs. Message passing
  - Bus/Switch vs. Network Interconnect
  - Monolithic vs. Composite
- Clusters form the dividing line
  - Treated as Parallel or Distributed depending on context.

Why Parallel or Distributed Computing?

- Limitations of Sequential computer:
  - By using multiple processing elements or nodes, a computation can be divided into smaller parts that can be executed simultaneously, which can lead to a significant reduction in the time it takes to complete the computation.
- Handling large-scale data:
  - As data sets continue to grow in size, parallel and distributed computing provide a way to process, analyze, and store large amounts of data in a reasonable amount of time. Data can be divided into smaller chunks and processed by multiple processors or nodes, which can help to speed up the analysis process.
- Cost-effectiveness:
  - Parallel and distributed computing can also be cost-effective, as it allows organizations to use less powerful, less expensive processors or nodes to perform a computation, rather than relying on a single, expensive supercomputer.

Nowadays, all computers are essentially parallel

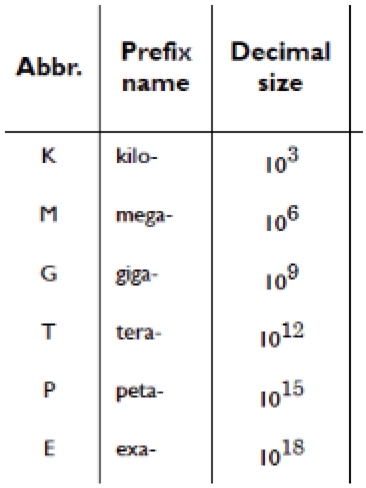
1. Parallelism is present deep in the processor microarchitecture
   - Pipelining
2. Any commercial computer, tablet, and smartphone contain a processor with multiple cores
3. Many servers contain several multi-core processors
4. Even consumer-level computers contain graphic processors capable of running hundreds or even thousands of threads in parallel

Reasons for making modern computers parallel

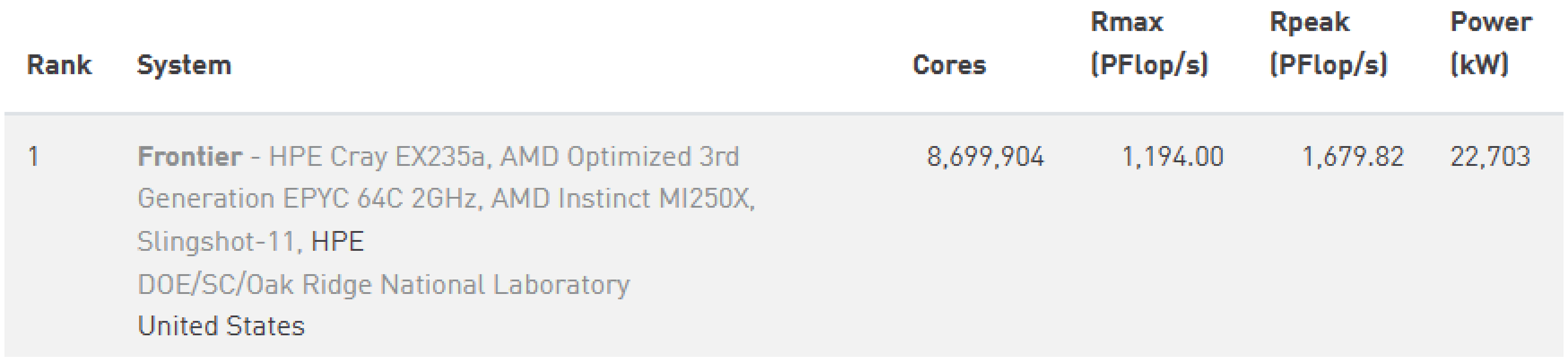
- First, it is not possible to increase processor and memory frequencies indefinitely
  - to increase computational power of computers, new architectural and organizational concepts are needed
- Second, power consumption rises with processor frequency while the energy efficiency decreases.
  - if the computation is performed in parallel at lower processor speed, this can be avoided
- Finally, parallelism has become a part of any computer and this is likely to remain unchanged

Computing capacities

- Distributed computing uses the Internet to link personal computers to achieve more FLOPS:
  - As of April 2020, the Folding@home network has over 2.3 exaFLOPS of total computing power
- Rank 1 super computer (parallel computer) with 8699904 cores has 1194 petaFlops



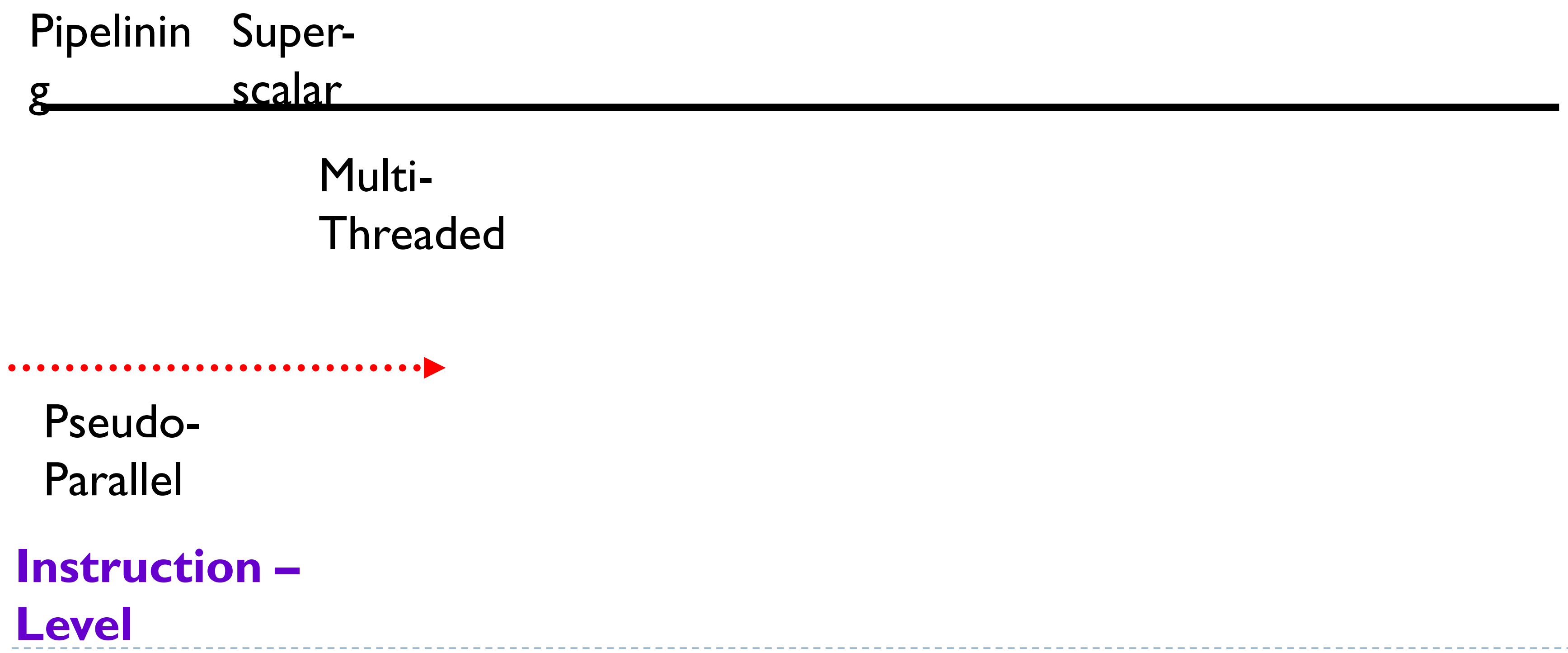
Abbr.	Prefix name	Decimal size
K	kilo-	10^3
M	mega-	10^6
G	giga-	10^9
T	tera-	10^{12}
P	peta-	10^{15}
E	exa-	10^{18}



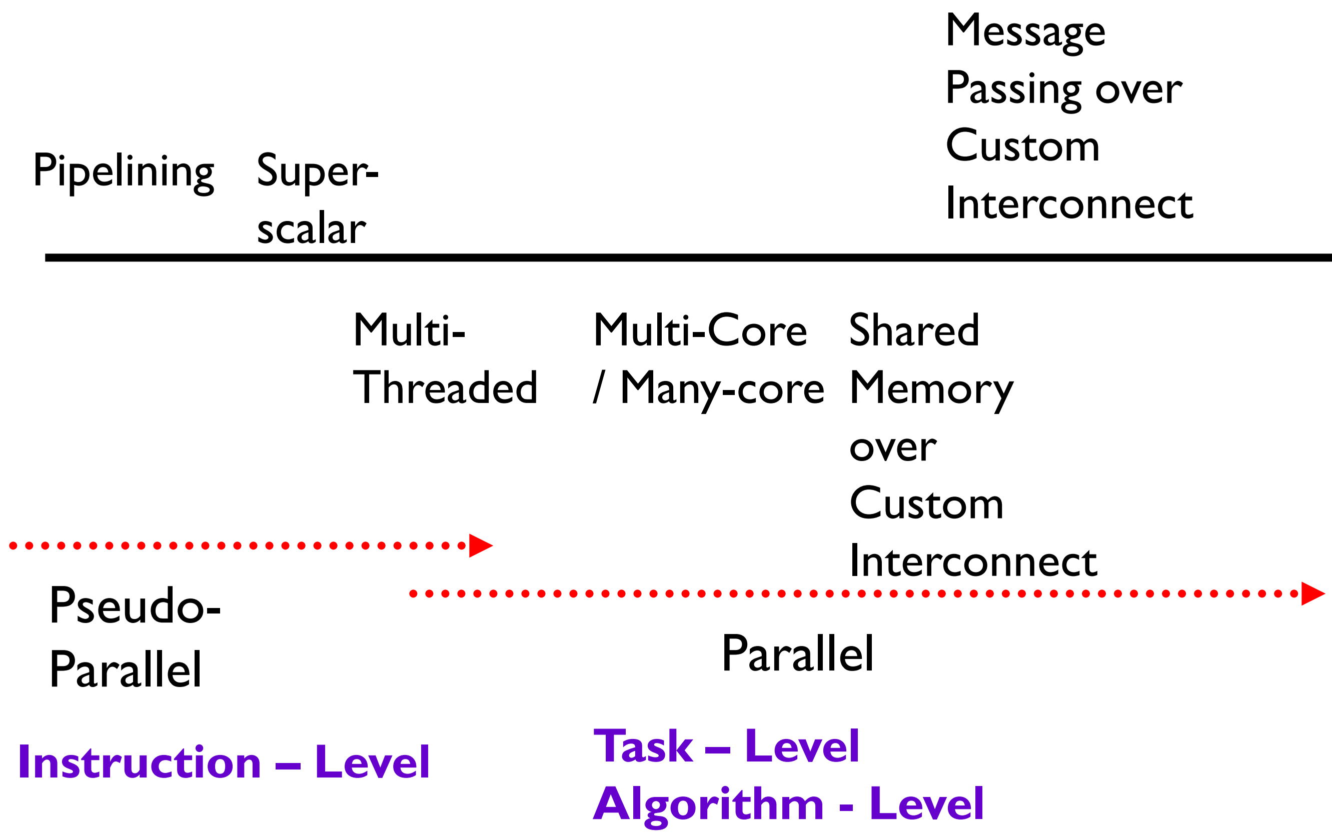
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703

TOP500 List - June 2023 | TOP500

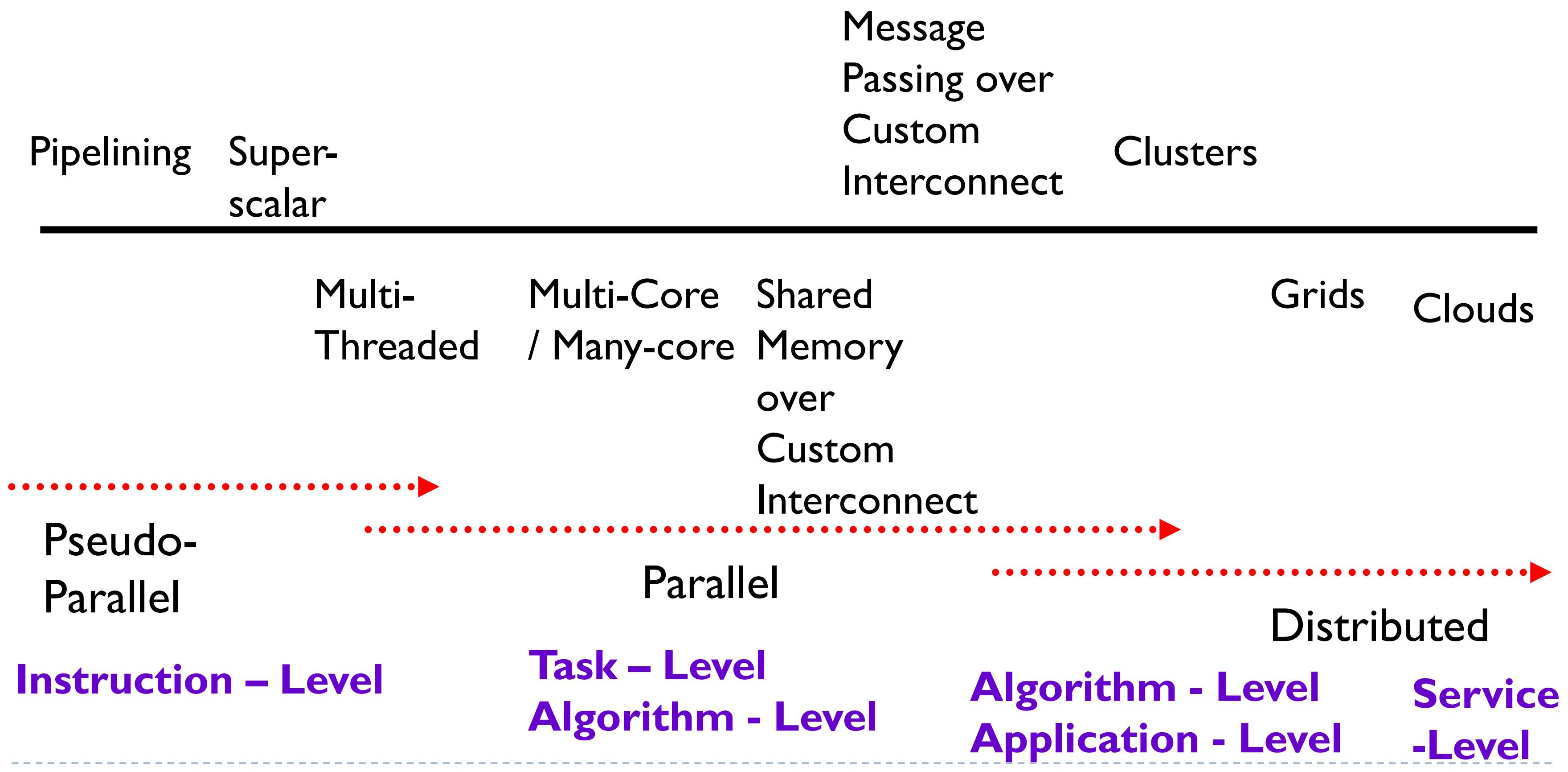
Under the Hood: Spectrum of Parallelism



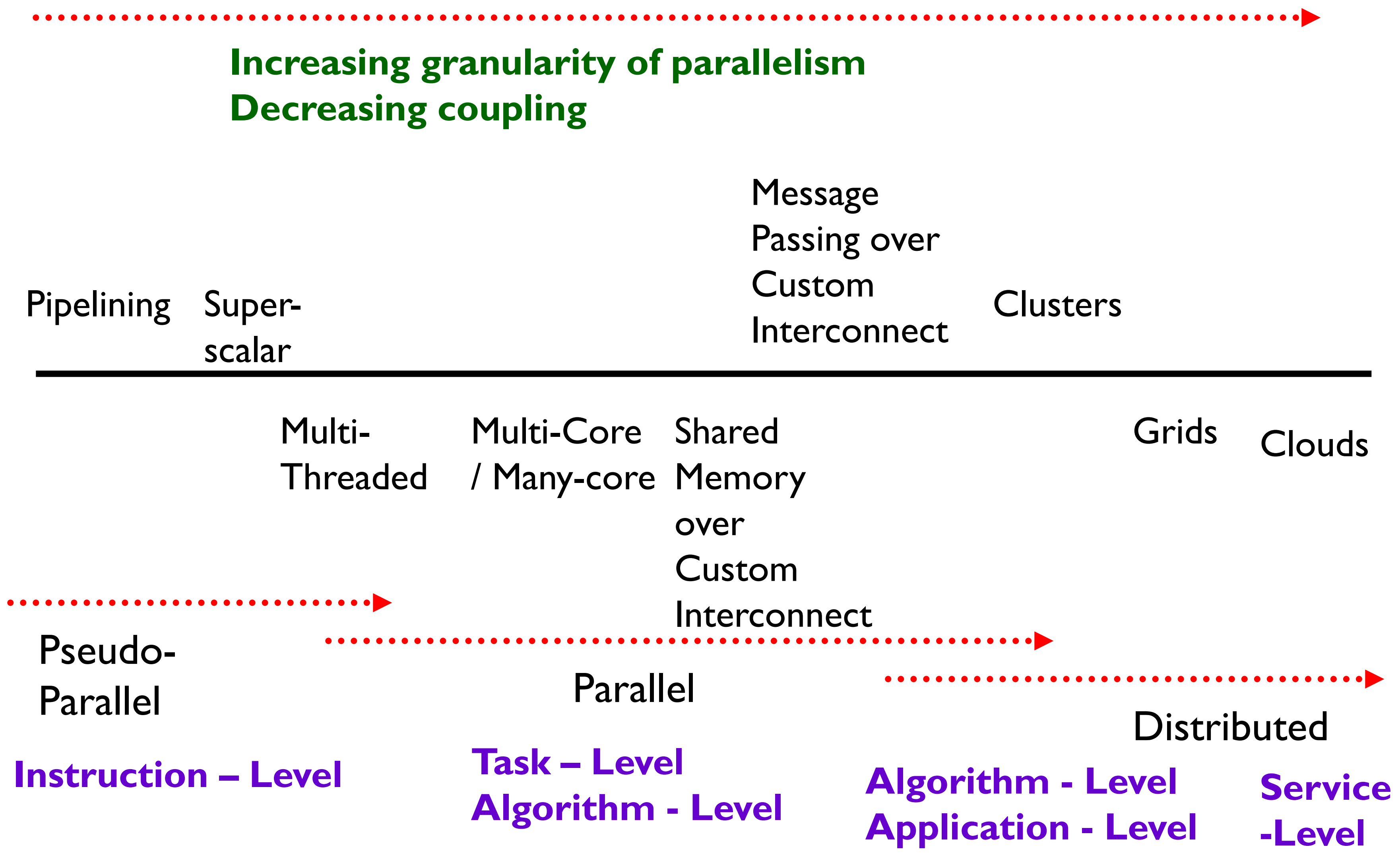
Under the Hood: Spectrum of Parallelism



Under the Hood: Spectrum of Parallelism

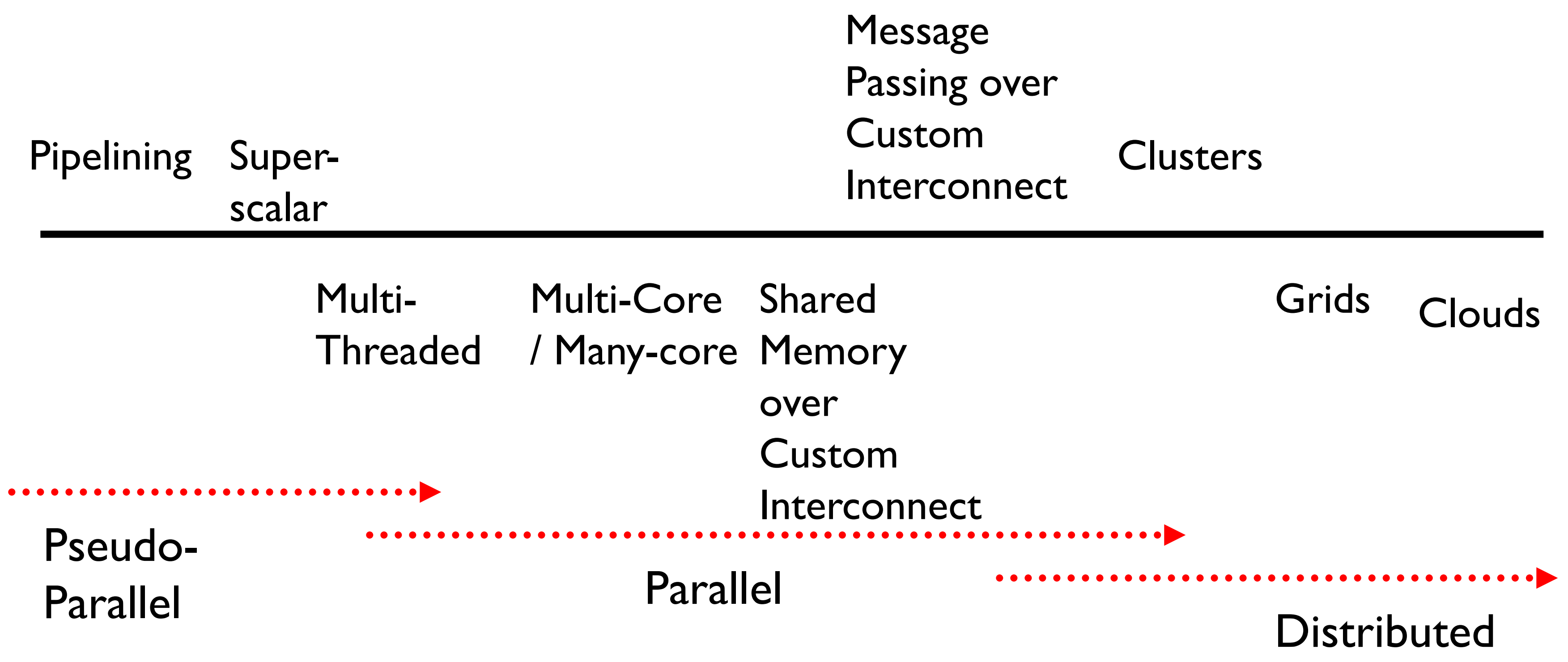


Under the Hood: Spectrum of Parallelism



Under the Hood: Spectrum of Parallelism

Increasing levels of abstraction (due to increased complexity)
Increasing redundancy requirement (due to increased failure probabilities)



Spectrum of Parallelism – Coarse-Grained Classification

- Data Parallelism
  - Partition your data and run the same task on different parts

vs.

- Task Parallelism
  - Partition your program into multiple tasks

vs.

- Request Parallelism
  - Partition your solution into a computation (i.e. task) per request

Designing Parallel Programs

- Steps for parallel program design and implementation
- Identifying Parallelism
  - Computational task partitioning. Aggregate tasks when needed.
  - Dependence analysis to derive a task graph
- Mapping & Scheduling of parallelism
  - Map tasks =⇒ processors (cores)
  - Order execution
- Parallel Programming
  - Coding
  - Debugging
- Performance Evaluation

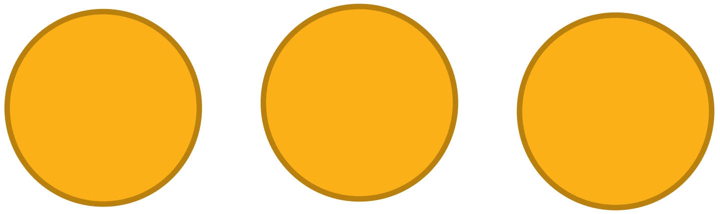
Task Decomposition

- The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently
- A given problem may be decomposed into tasks in many different ways.
- Tasks may be of same, or different sizes.
- A decomposition can be illustrated in the form of a directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called a *task dependency graph or data dependency graph*.

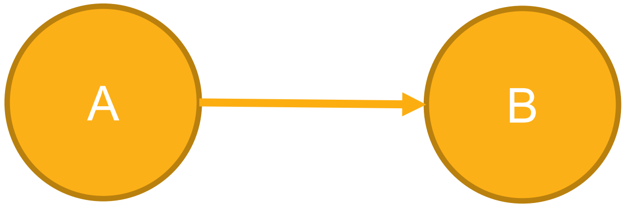
Task Graphs

BITS Pilani

- A set of Tasks



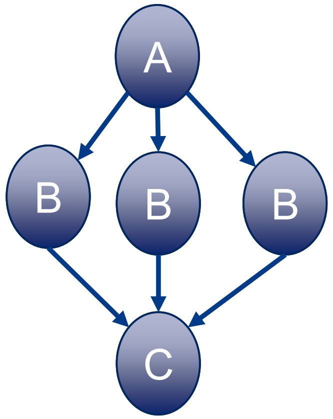
- Data dependence among tasks

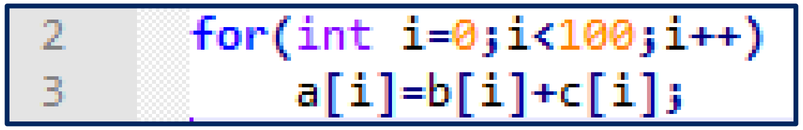


Seeking Concurrency

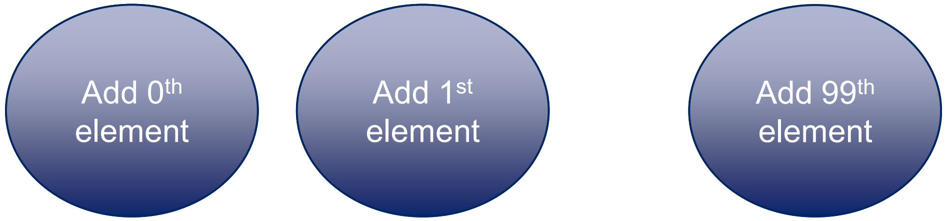
BITS Pilani

- Parallel computers are more available than ever, but in order to take advantage of multiple processors, programmers and/or compilers must be able to identify operations that may be performed in parallel (i.e., concurrently)
- Data parallelism
  - A data dependence graph exhibits data parallelism when there are independent tasks applying the same operation to different elements of a dataset.





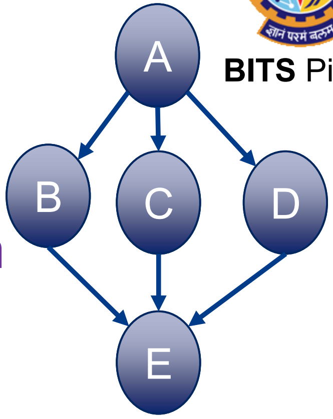
```
for(int i=0;i<100;i++)
    a[i]=b[i]+c[i];
```

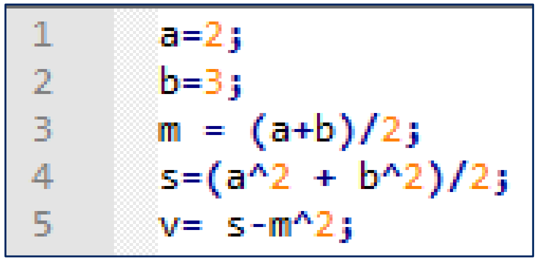


While we could draw one vertex for each step in the algorithm, it is better to draw a vertex for each step of the algorithm for each data element because it exposes more opportunities for parallelism.

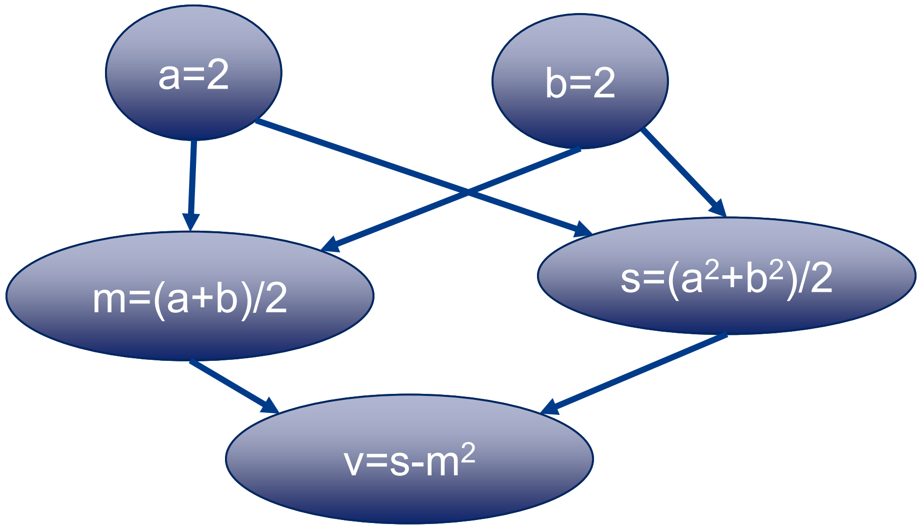
Seeking Concurrency

- Functional parallelism
  - A data dependence graph exhibits functional parallelism when there are independent tasks applying different operations to different data elements



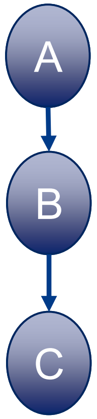


```
a=2;
b=3;
m = (a+b)/2;
s=(a^2 + b^2)/2;
v= s-m^2;
```



While we could draw one vertex for each step in the algorithm, it is better to draw a vertex for each step of the algorithm for each data element because it exposes more opportunities for parallelism.

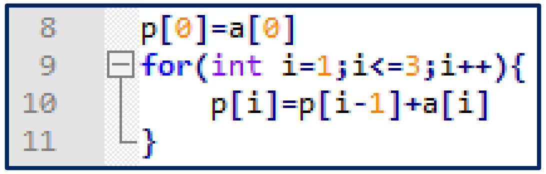
Seeking Concurrency



BITS Pilani

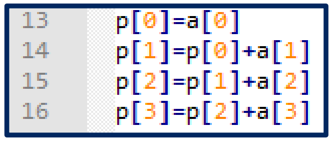
- Pipelining
  - A data dependence graph forming a simple path or chain admits no parallelism if only a single problem instance must be processed.
  - If problem instances need to be processed, each problem instance an be divided into stages and concurrency equal to the number of stages can be achieved.

If multiple instances of a problem are there, operation C may be performed on instance i, while operation B is performed on instance i+1 and operation A is performed on instance i+2.

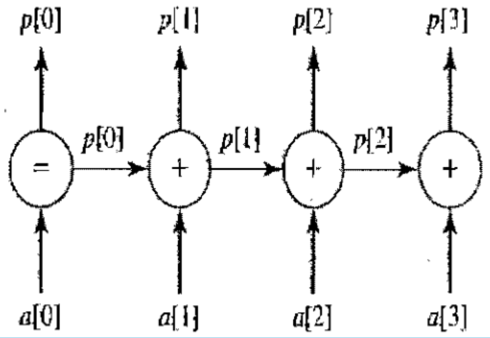


```
p[0]=a[0]
for(int i=1;i<=3;i++){
    p[i]=p[i-1]+a[i]
}
```

Suppose 2 sets of nos:
{1,2,3,4}
{5,6,7,8}



```
p[0]=a[0]
p[1]=p[0]+a[1]
p[2]=p[1]+a[2]
p[3]=p[2]+a[3]
```



Data Clustering

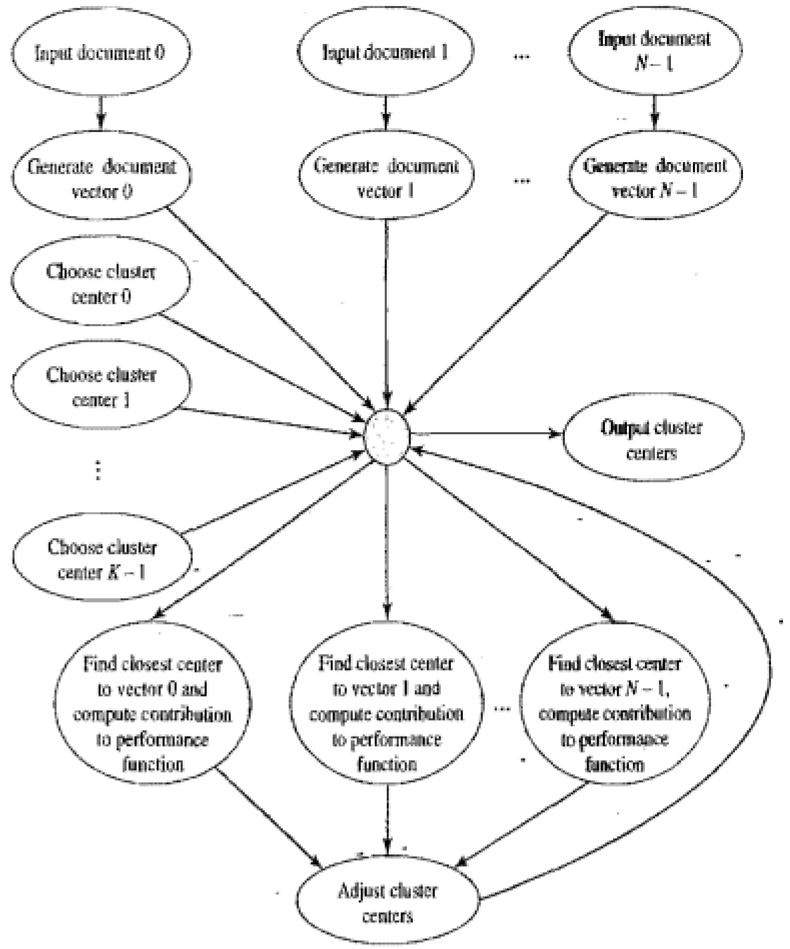
- Data clustering is the process of organizing a dataset into groups, or clusters of "similar" items
- Algorithm
  1. Input N documents
  2. For each of the N documents generate a D-dimensional vector indicating how well it covers the D diffent topics
  3. Choose the K initial cluster centers using a random sample
  4. Repeat the following steps for *I* iterations or until the performance function converges whichever comes first

     (a)for each of the N documents, find the closest center and compute its contribution to the performance function

     (b) Adjust the K cluster centers to try to improve the value of the performance function
  5. OUTPUT K centers

Data Clustering

- Our first step in the analysis is to draw a data dependence graph.


Input document 0
Input document 1
...
Input document N − 1
Generate document vector 0
Generate document vector 1
...
Generate document vector N − 1
Choose cluster center 0
Choose cluster center 1
⋮
Choose cluster center K − 1
Output cluster centers
Find closest center to vector 0 and compute contribution to performance function
Find closest center to vector 1 and compute contribution to performance function
...
Find closest center to vector N − 1, compute contribution to performance function
Adjust cluster centers

A good data dependence graph makes data and functional parallelism easy to find.

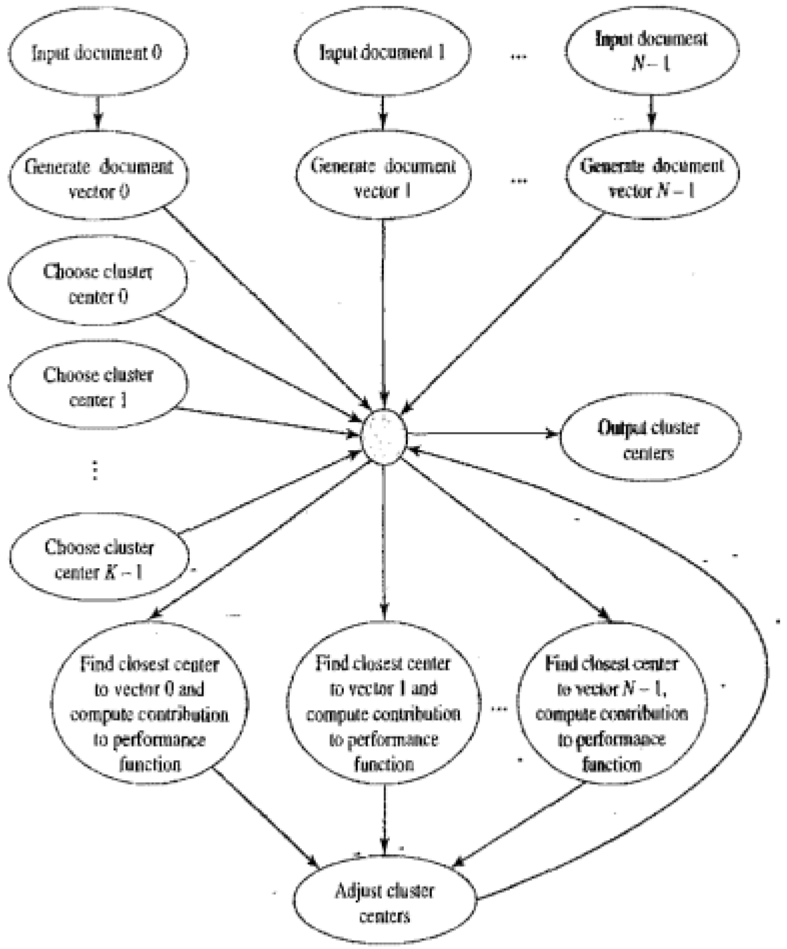
<u>Opportunities for data parallelism:</u>

- Each document may be input in parallel.
- Each document vector may be generated in parallel.
- The original cluster centers may be generated in parallel.
- The closest cluster center to each document vector and that vector's contribution to the overall performance function may be computed in parallel.

Data Clustering

BITS Pilani

- Our first step in the analysis is to draw a data dependence graph.


Input document 0
Input document 1
Input document N – 1
Generate document vector 0
Generate document vector 1
Generate document vector N – 1
Choose cluster center 0
Choose cluster center 1
Choose cluster center K – 1
Output cluster centers
Find closest center to vector 0 and compute contribution to performance function
Find closest center to vector 1 and compute contribution to performance function
Find closest center to vector N – 1, compute contribution to performance function
Adjust cluster centers

A good data dependence graph makes data and functional parallelism easy to find.

Opportunities for functional parallelism:
The only independent sets of vertices are those representing the document input and vector generation tasks and those representing the center generation tasks. These two sets of tasks could be performed concurrently

Example: Database Query Processing

- Consider the execution of the query:

  MODEL = ``CIVIC" AND YEAR = 2001 AND (COLOR = ``GREEN" OR COLOR = ``WHITE)

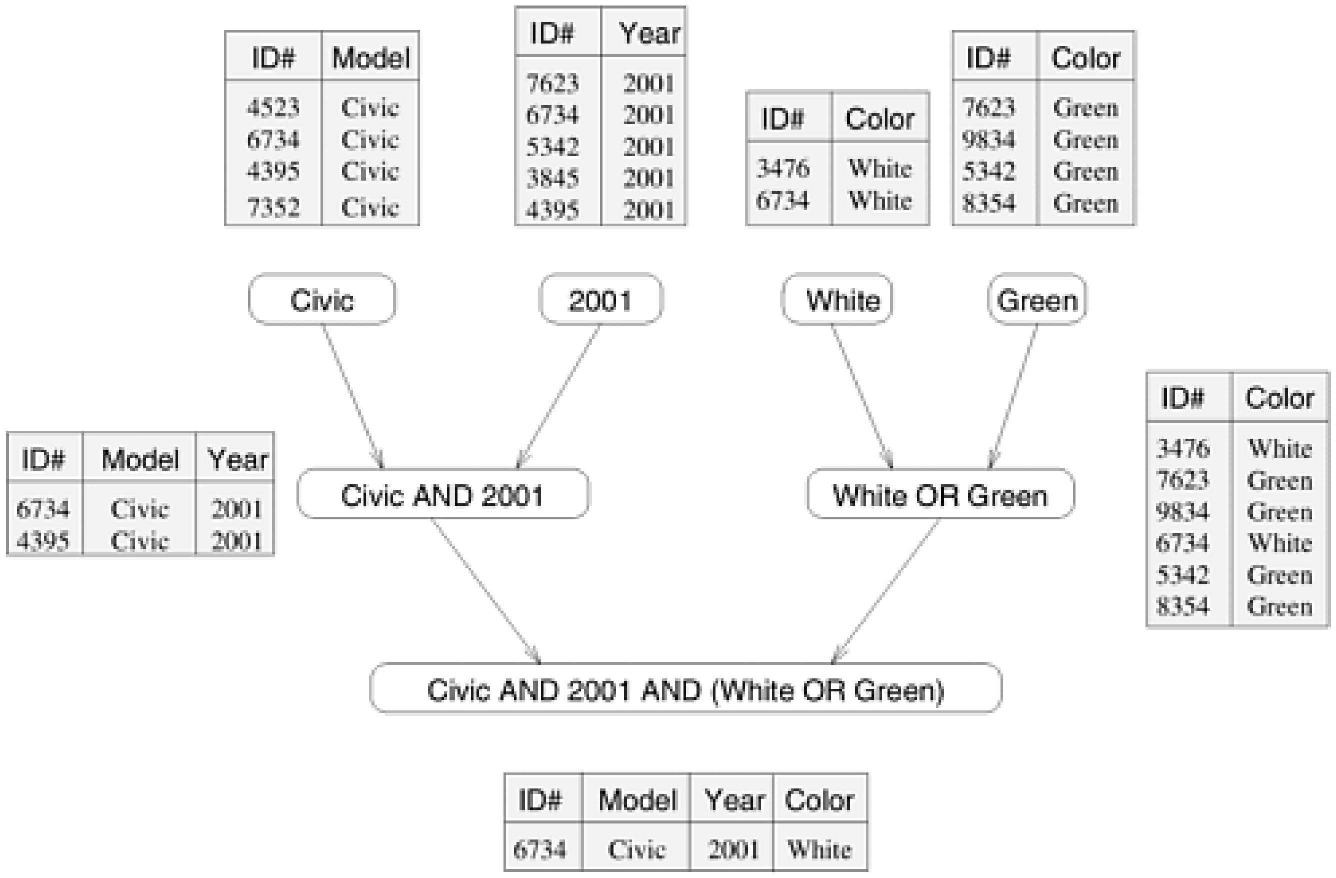
- on the following database:

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Decomposition 1

BITS Pilani

- The execution of the query can be divided into subtasks in various ways. Each task can be thought of as generating an intermediate table of entries that satisfy a particular clause.



ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

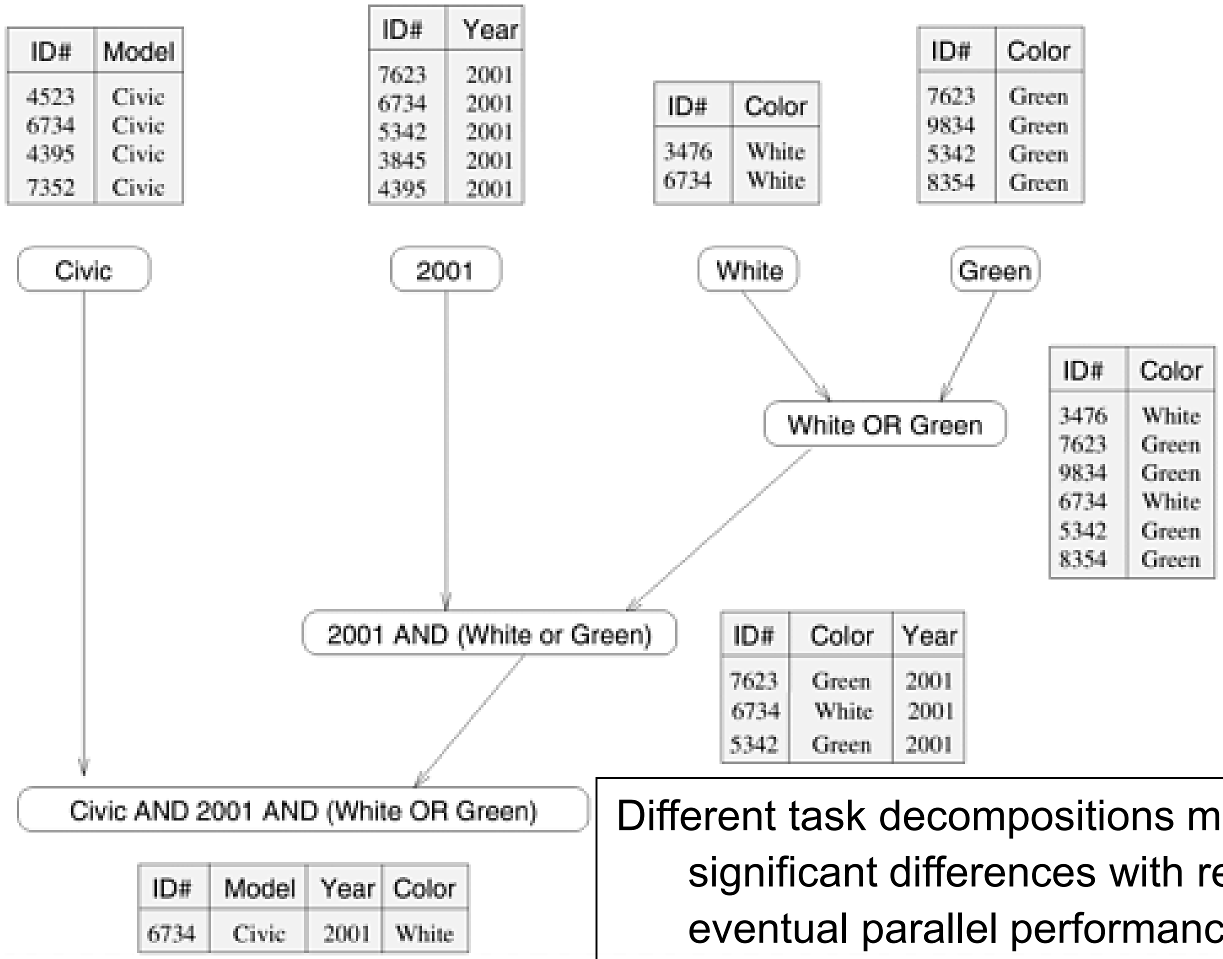
ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Model	Year	Color
6734	Civic	2001	White

Decomposition 2

- Note that the same problem can be decomposed into subtasks in other ways as well.


Civic
2001
White
Green
White OR Green
2001 AND (White or Green)
Civic AND 2001 AND (White OR Green)

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

ID#	Model	Year	Color
6734	Civic	2001	White

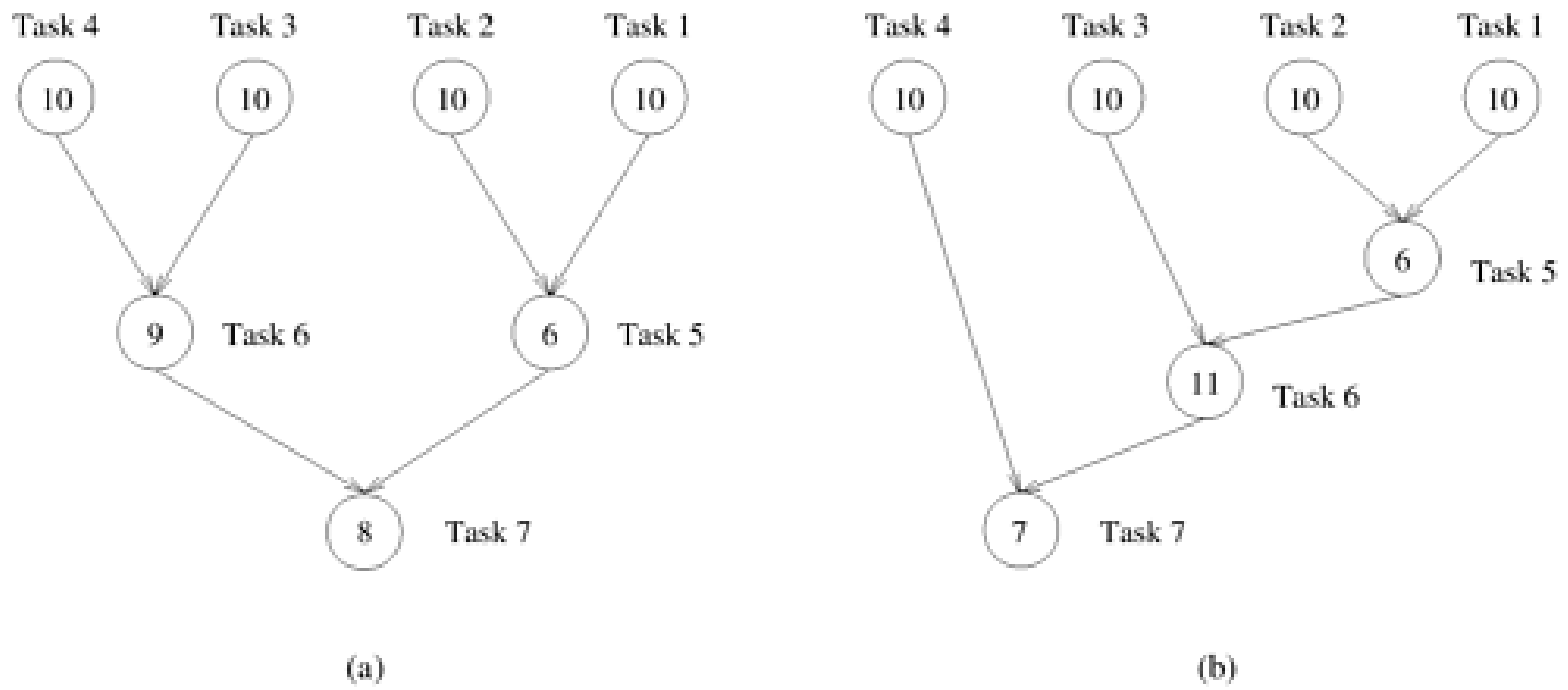
Different task decompositions may lead to significant differences with respect to their eventual parallel performance.

Comparing Decompositions

BITS Pilani

- The number of tasks that can be executed in parallel is the *degree of concurrency* of a decomposition.
- The length of the longest path in a task dependency graph is called the critical path length.
- The ratio of the total amount of work to the critical-path length is the average degree of concurrency.

Comparing Decompositions



- Maximum degree of concurrency: 4   4
- Critical path length: 27   34
- Total work: 63   64
- Average degree of concurrency: 63/27=2.34   64/34=1.88

References

- Chapter 1.6 from M.J. Quinn, *Parallel Programming in C using MPI and OpenMP*, McGraw Hill Indian Edition. 2003

Approaches to parallel programming

- Three different approaches to parallel programming exist:
  - Threads model for shared memory systems
  - Message passing model for distributed systems
  - Stream-based model for GPUs

Why Study Parallel and distributed programming?

- It's a critical skill for high-performance computing: With the increasing demand for computing power in many fields, such as scientific simulation, data analytics, artificial intelligence and machine learning, the ability to write parallel programs is becoming an essential skill for computer scientists and engineers.

Why Study Parallel and distributed programming?

BITS Pilani

- It's needed for large scale data processing: The amount of data being generated and collected is growing rapidly, and the ability to process this data in parallel is critical for many applications such as big data analytics and machine learning
- It's needed to solve large and complex problems: Parallel computing allows for solving large and complex problems that would be otherwise intractable with a single processor.
- It's needed to take advantage of new technologies: Advances in hardware technologies, such as GPUs and many-core processors, as well as new parallel programming models and languages, such as OpenMP, MPI, CUDA, OpenCL, and others, are making parallel computing more accessible and easier to use.

Applications

- Parallel and distributed computing has a wide range of applications, including:
  - Scientific simulations: can be used to simulate complex physical systems, such as weather patterns, fluid dynamics, and chemical reactions.
  - Data analysis: can be used to process and analyze large sets of data, such as in data mining and machine learning.
  - Computer graphics: can be used to render complex 3D images and animations in real-time.
  - Drug discovery: can be used to simulate the interactions between potential drug compounds and biological systems, helping researchers to identify new drugs more quickly.
  - Climate modeling: can be used to simulate and predict future climate patterns and to understand the causes and effects of climate change.

Applications

- Parallel and distributed computing has a wide range of applications, including:
  - Computer Vision: can be used to process images and videos in real-time, using techniques such as object detection, image recognition, and facial recognition.
  - Artificial Intelligence: can be used to train and deploy large AI models, such as deep learning neural networks, and to speed up the inference process.
  - Financial modeling: can be used to perform complex financial simulations, such as risk analysis and portfolio optimization.

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You