CSE502: Foundations of Parallel Programming

Lecture 04: Concurrency Decomposition

Vivek Kumar

Computer Science and Engineering

IIIT Delhi

vivekk@iiitd.ac.in

Last Class

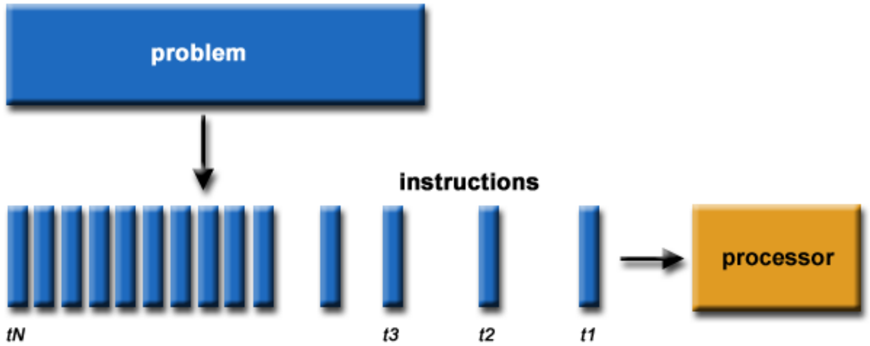
- Why parallel programming
  - Technological push
    - Multicore processors
  - Application push
    - Compute intensive applications operating on large data
- Why multicores?
  - Moore’s law and Dennard scaling
  - Multicore saves power
- Parallel hardware in the large
  - Multicores are also available in modern supercomputers
- FLOPS – theoretical peak performance of a computers for scientific computing
- Different parallel programming models
  - Automatic
  - Shared memory
  - Distributed memory
  - Hybrid shared and distributed memory

Today's Class

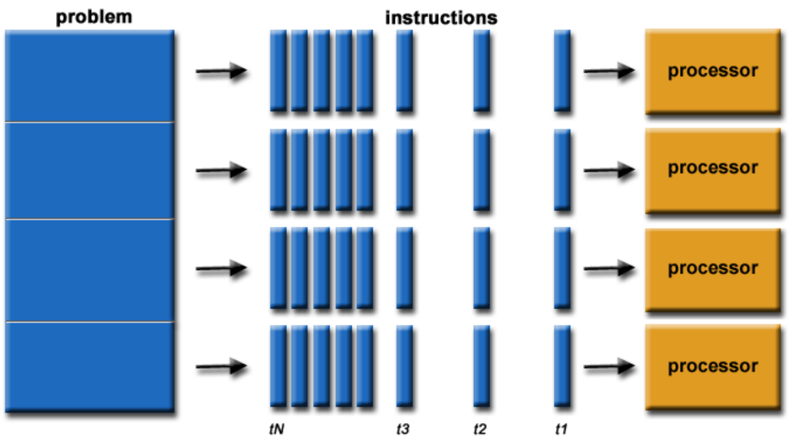
- Decomposition of sequential program into parallel program
  - Tasks and decomposition
  - Amdahl's law
  - Tasks and mapping
  - Decomposition techniques
    - Recursive
    - Data
    - Exploratory
    - Speculative

Concurrency v/s Parallelism

- Concurrency
  - "**Dealing**" with lots of things at once



- Parallelism
  - "**Doing**" lots of things at once



Concurrency v/s Parallelism

- Concurrency
  - Refers to tasks that appear to be running simultaneously, but which may, in fact, actually be running serially

Concurrency v/s Parallelism

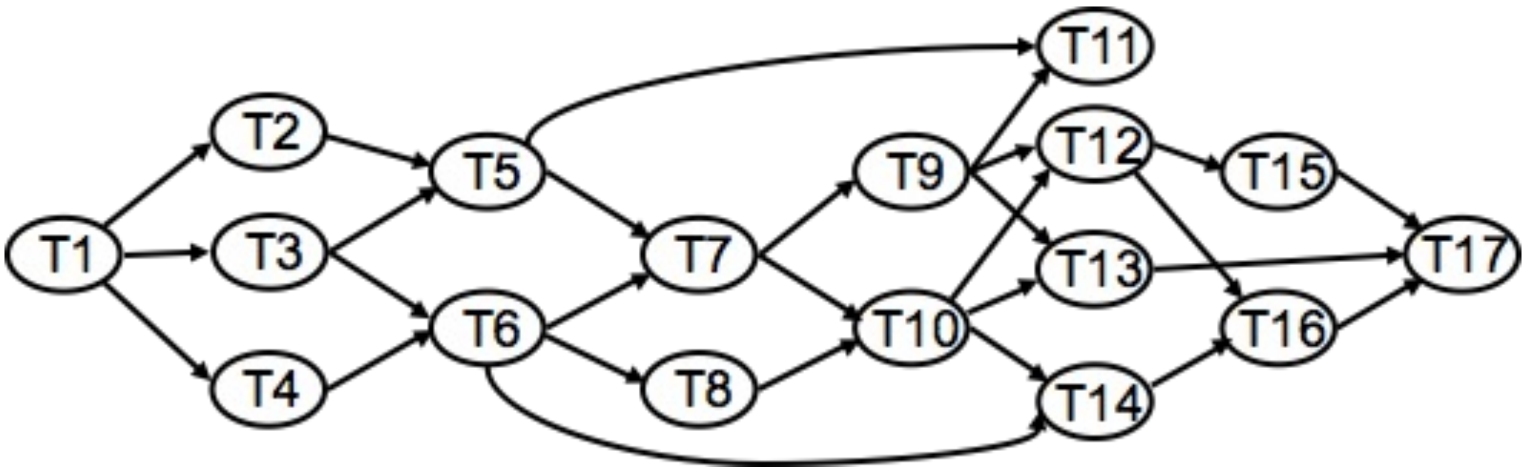
- Parallelism
  - Refers to concurrent tasks that actually run at the same time
  - Always implies multiple processors
  - Parallel tasks always run concurrently, but not all concurrent tasks are parallel.

Recipe to Solve a Problem using Parallel Programming

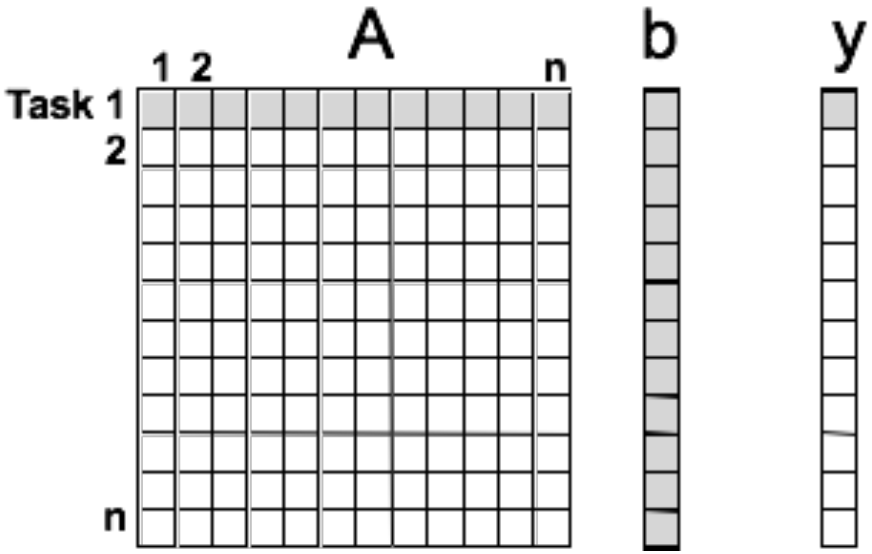
- **Typical** steps for constructing a parallel algorithm
  - identify what pieces of work can be performed concurrently
  - partition concurrent work onto independent processors
  - distribute a program's input, output, and intermediate data
  - coordinate accesses to shared data: avoid conflicts
  - ensure proper order of work using synchronization
- Why "**typical**"? Some of the steps may be omitted.
  - if data is in shared memory, distributing it may be unnecessary
  - if using message passing, there may not be shared data
  - the mapping of work to processors can be done statically by the programmer or dynamically by the runtime

Decomposing Work for Parallel Execution

- Divide work into tasks that can be executed concurrently
- Many different decompositions possible for any computation
- Tasks may be same, different, or even indeterminate sizes
- Tasks may be independent or have non-trivial order
  - Conceptualize tasks and ordering as computation graph
    - Node = task
    - Edge = control dependency



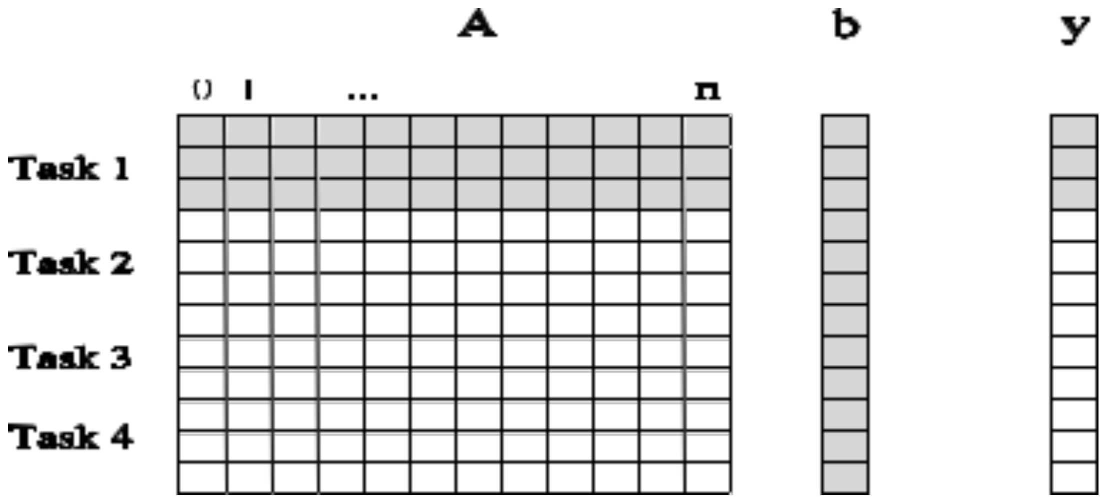
Example: Dense Matrix Vector Product



- Computing each element of output vector y is independent
- Easy to decompose dense matrix-vector product into tasks
  - one per element in y
- Observations
  - task size is uniform
- no control dependences between tasks
  - tasks share b

Granularity of Task Decomposition

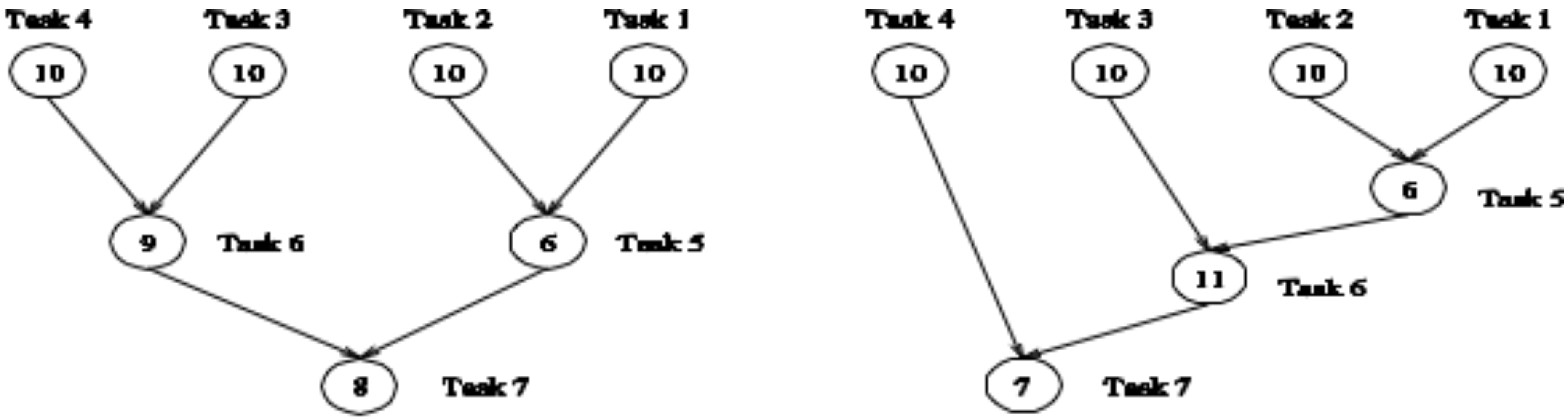
- Granularity = task size
  - depends on the number of tasks
- Fine-grain = large number of tasks
- Coarse-grain = small number of tasks
- Granularity examples for dense matrix-vector multiply
  - fine-grain: each task represents an individual element in y
  - coarser-grain: each task computes 3 elements in y



Critical Path

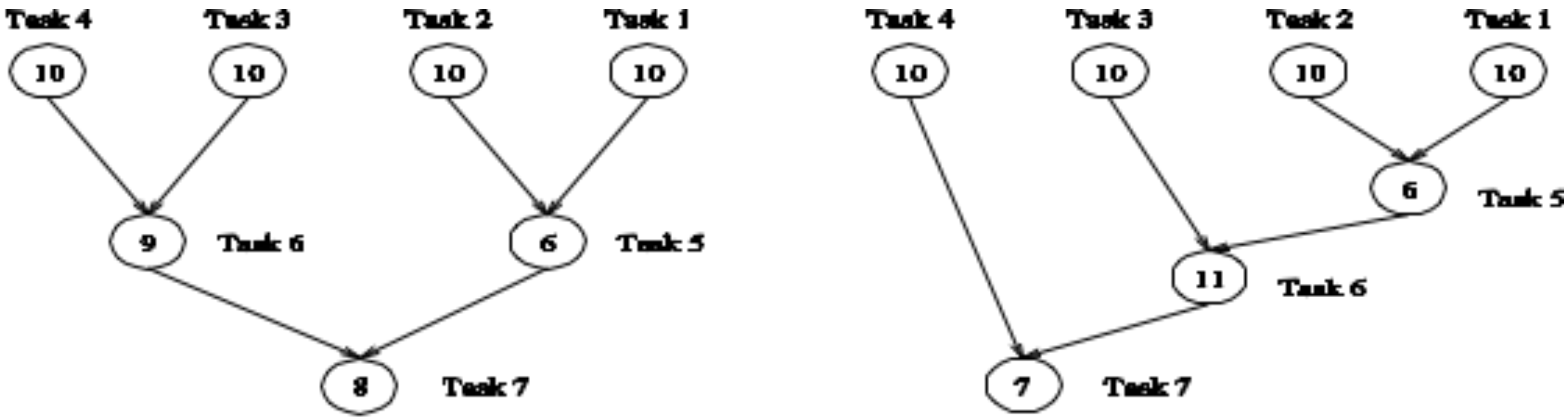
- Edge in computation graph represents task serialization
- Critical path = longest weighted path though graph
- Critical path length = lower bound on parallel execution time

Critical Path Length



Note: number in vertex represents task cost

Critical Path Length



Note: number in vertex represents task cost

Questions:

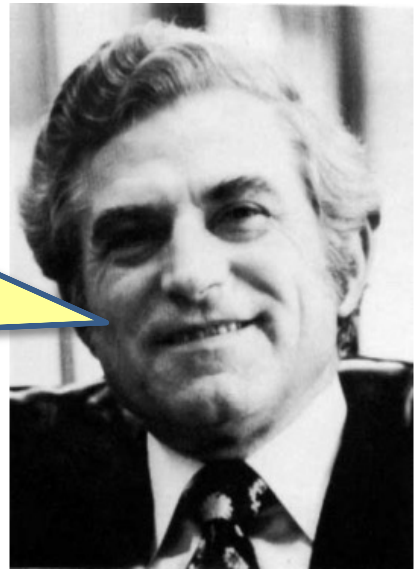
- What are the tasks on the critical path for each dependency graph?
- What is the shortest parallel execution time for each decomposition?

Limits on Parallel Performance

- What bounds parallel execution time?
  - minimum task granularity
    - e.g. dense matrix-vector multiplication ≤ n^2 concurrent tasks
  - dependencies between tasks
  - parallelization overheads
    - e.g., cost of communication between tasks
  - fraction of application work that can't be parallelized
    - Amdahl's law

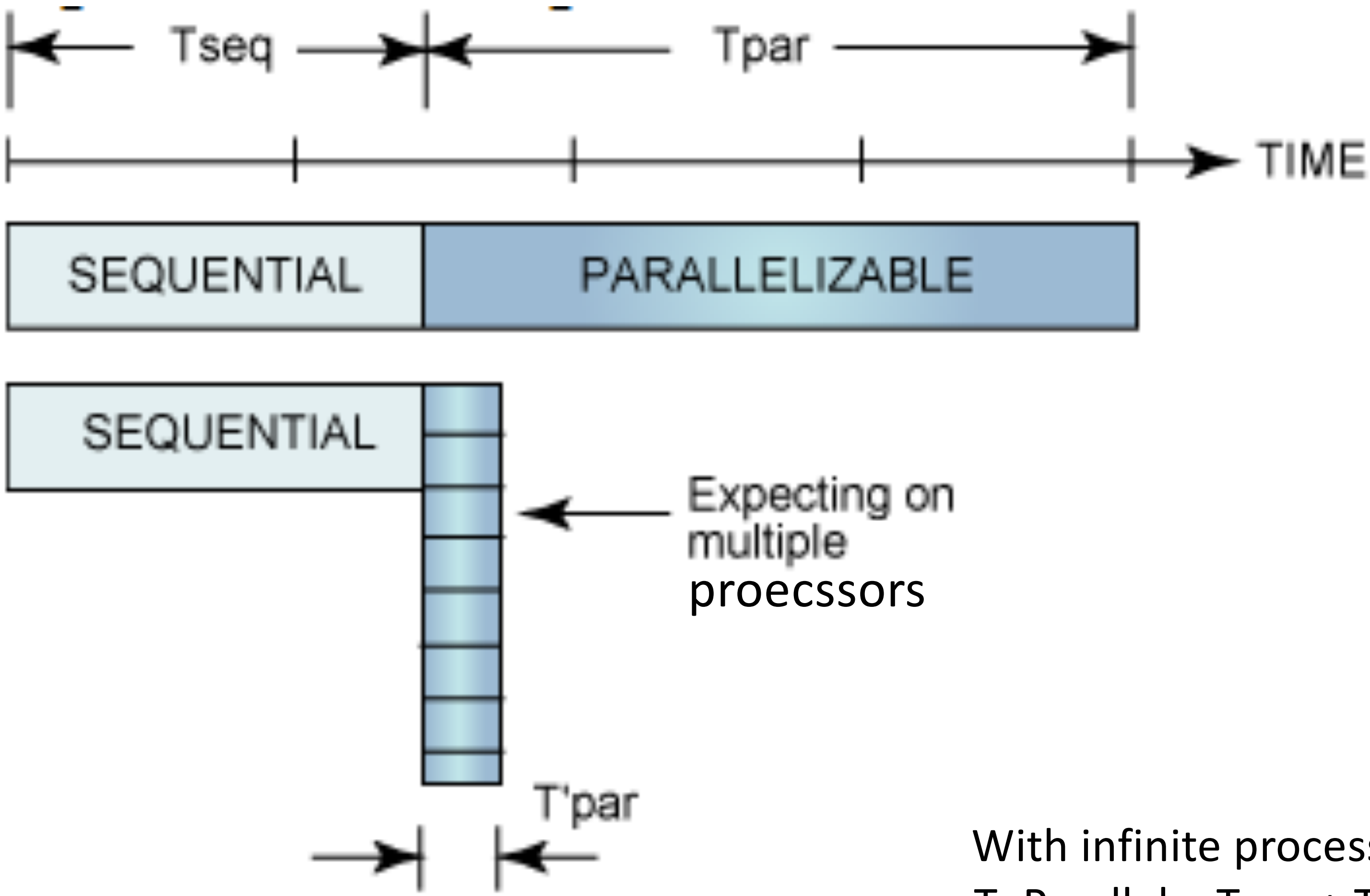
Amdahl's Law

If 50% of your application is parallel and 50% is serial, you can't get more than a factor of 2 speedup, no matter how many processors it runs on.



Gene M. Amdahl

Amdahl's Law



With infinite processors,
T_Parallel = Tseq + T'par

Measures of parallel performance

- Speedup = $T_{serial}/T_{parallel}$
- Parallel efficiency = $T_{serial}/(pT_{parallel})$

1. Do disproportionately less work
2. Harness disproportionately more resources

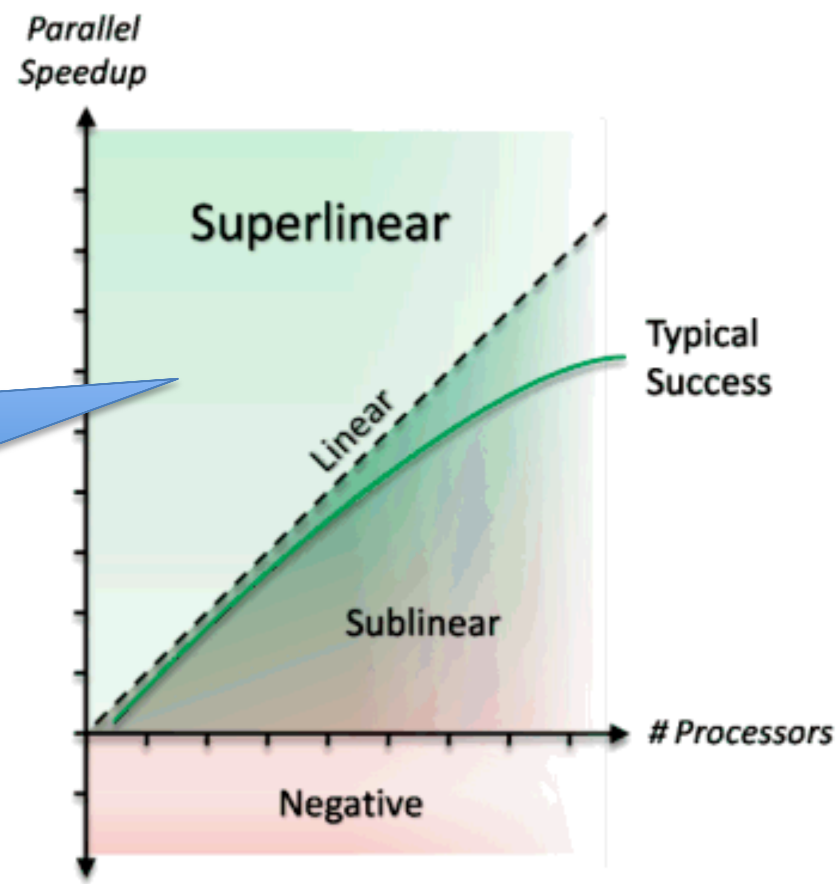


Fig. source: http://www.drdobbs.com/cpp/going-superlinear/206100542

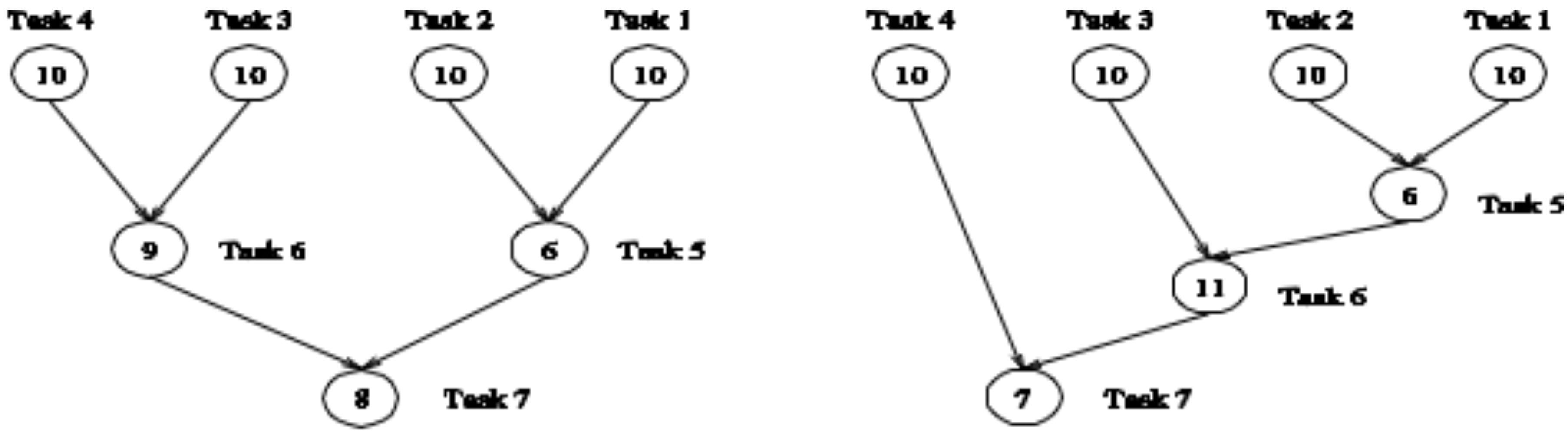
Today's Class

- Decomposition of sequential program into parallel program
  - Tasks and decomposition
  - Amdahl's law
  - Tasks and mapping
  - Decomposition techniques
    - Recursive
    - Data
    - Exploratory
    - Speculative

Mapping Tasks to Cores

- Generally
  - # of tasks ≥ # threads available
  - parallel algorithm must map tasks to threads
  - schedule independent tasks on separate threads (consider computation graph)
  - threads should have minimum interaction with one another

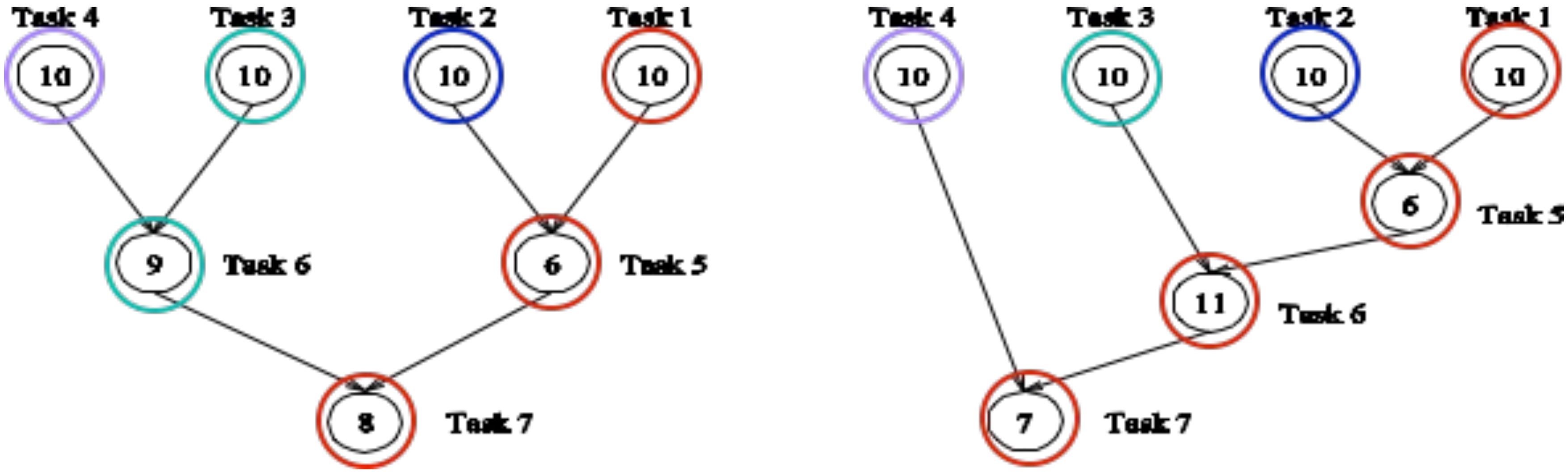
Tasks, Threads, and Mapping Example



Note: number in vertex represents task cost

- How to best map these tasks on threads?

Tasks, Threads, and Mapping Example



- No tasks in a level depend upon each other
- Assign all tasks within a level to different threads

Mapping Techniques

Static vs. dynamic mappings

- Static mapping
  - *a-priori* mapping of tasks to threads or processes
    - requirements
      - a good estimate of task size
      - even so, computing an optimal mapping may be hard
- Dynamic mapping
    - map tasks to threads or processes at runtime
    - why?
      - tasks are generated at runtime, or
      - their sizes are unknown

Static Mapping

- Data partitioning
- Computation graph partitioning
- Hybrid strategies

Dynamic Mapping

- Dynamic mapping AKA dynamic load balancing
  – load balancing is the primary motivation for dynamic mapping
- Styles
  – centralized
  – distributed

Today's Class

- Decomposition of sequential program into parallel program
  - Tasks and decomposition
  - Amdahl's law
  - Tasks and mapping
  - Decomposition techniques
    - Recursive
    - Data
    - Exploratory
    - Speculative

Decomposition Techniques

How should one decompose a task into various subtasks?

- No single universal recipe
- In practice, a variety of techniques are used including
  - Data decomposition
  - Recursive decomposition
  - Exploratory decomposition
  - Speculative decomposition

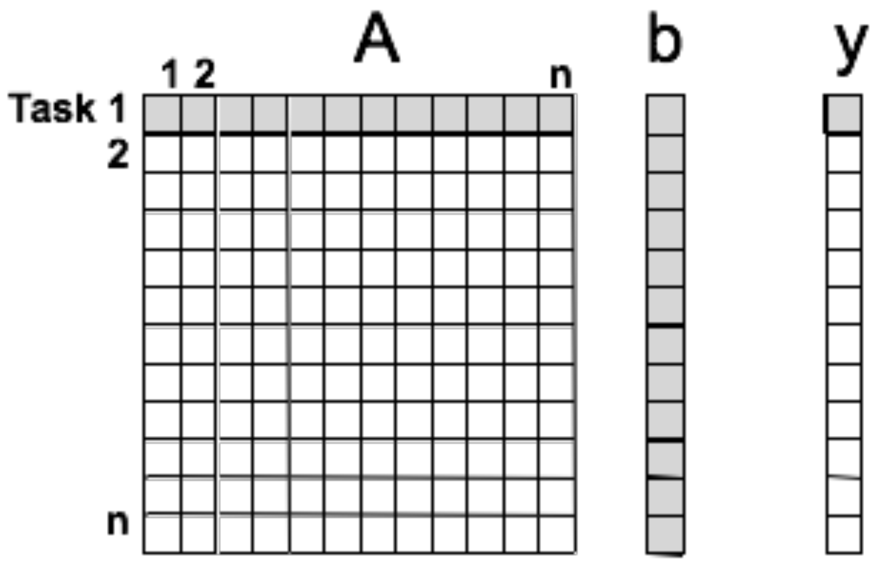
Data Decomposition

- Steps
  1. identify the data on which computations are performed
  2. partition the data across various tasks
     - partitioning induces a decomposition of the problem

Data Decomposition Example

- If each element of the output can be computed independently
- Partition the output data across tasks
- Have each task perform the computation for its outputs

Example:
dense matrix-vector multiply



Recursive Decomposition

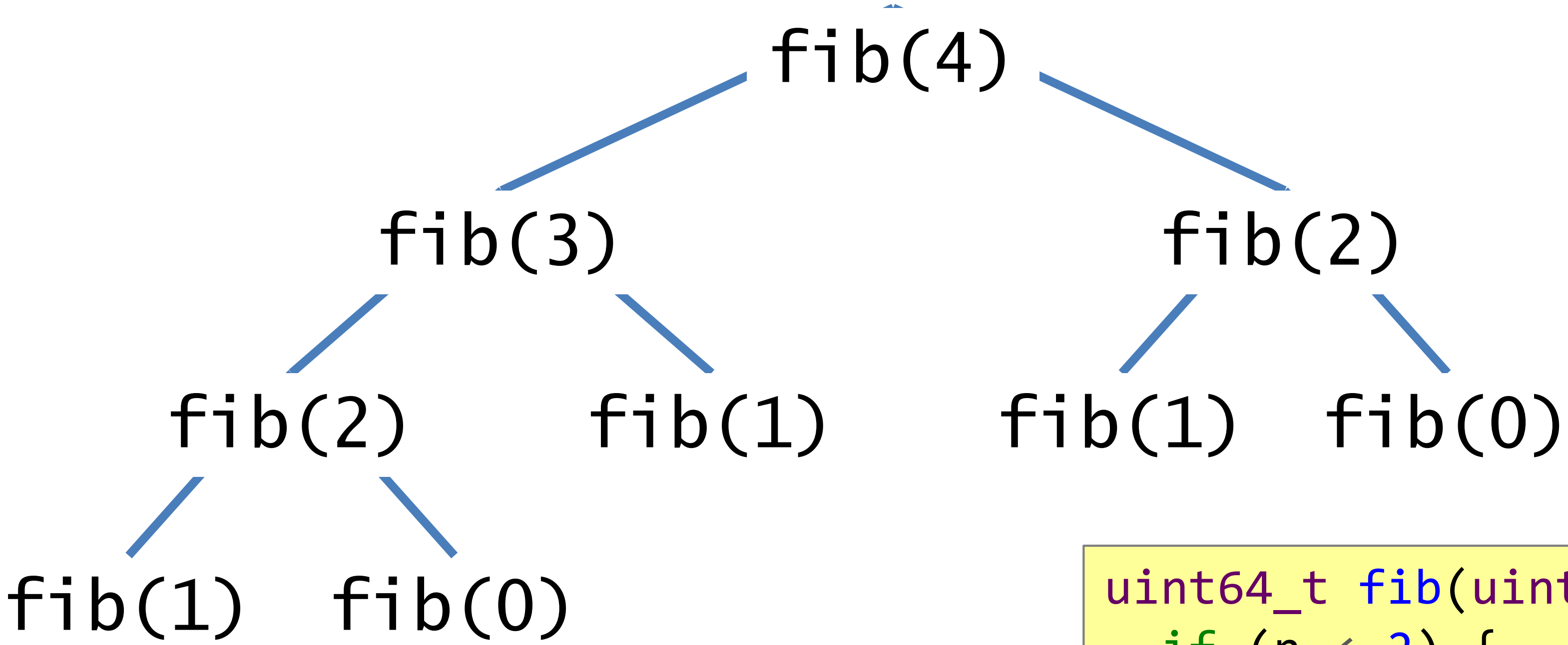
The **Fibonacci numbers** are the sequence ⟨0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …⟩, where each number is the sum of the previous two.

Recurrence:

$F_0 = 0$,
$F_1 = 1$,
$F_n = F_{n-1} + F_{n-2}$ for $n > 1$.

The sequence is named after Leonardo di Pisa (1170–1250 A.D.), also known as Fibonacci, a contraction of *filius Bonaccii* —"son of Bonaccio." Fibonacci's 1202 book *Liber Abaci* introduced the sequence to Western mathematics, although it had previously been discovered by Indian mathematicians.

Recursive Decomposition of Fibonacci




```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}
```

Question: what kind of mapping is suited for this scenario?

Exploratory Decomposition

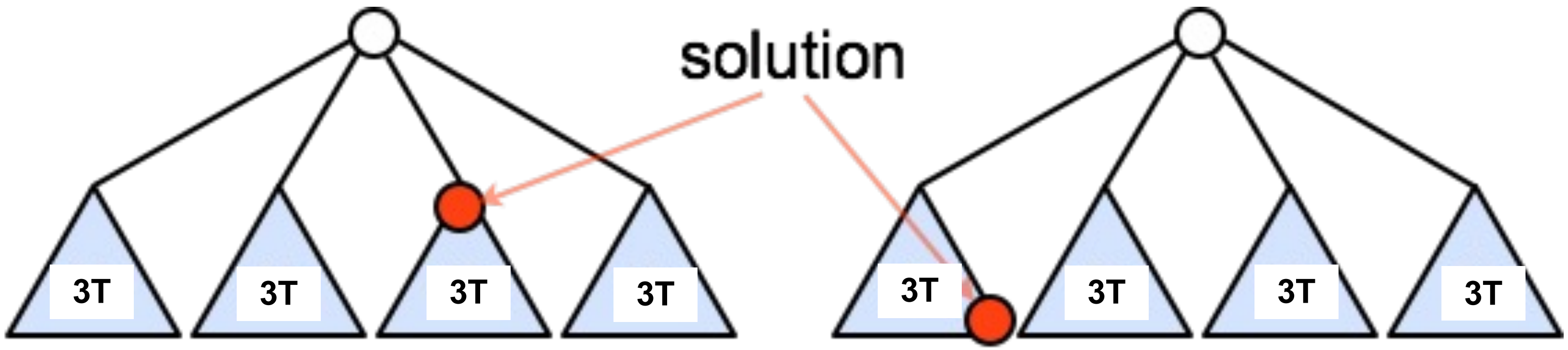
- Exploration (search) of a state space of solutions
  - problem decomposition reflects shape of execution

Exploratory Decomposition

- Parallel formulation may perform a different amount of work

Exploratory Decomposition Speedup

- Parallel formulation may perform a different amount of work
  - **Can cause super- or sub-linear speedup**
- Assume each vertex of the triangles represents a computation that takes 'T' unit of time to compute and execution begins from leftmost triangle to the rightmost



- Serial execution time = 7T
- Parallel execution time using 4 threads to compute each triangle in parallel = T
- Speedup (4 threads) = 7T/T = 7
- **Super**-linear speedup

- Serial execution time = 3T
- Parallel execution time using 4 threads to compute each triangle in parallel = 3T
- Speedup (4 threads) = 3T/3T = 1
- **Sub**-linear speedup

Question

- How exploratory decomposition (ED) differs from data decomposition (DD)?
  1. Unlike ED, **all** partial tasks contribute to final result in DD
  2. Unlike DD, unfinished tasks in ED can be terminated once final solution is found

Speculative Decomposition

- <u>Example</u>: when program may take one of many possible compute-intensive branches depending on the output of preceding computation

```
int val = T1                 //compute intensive
switch(val) {                // cases may be computed speculatively
    case 0: T2; break;
    case 1: T3; break;
    …..
    case n: Tn; break;
}
```

Next Class

- Introduction to dynamic task creation and termination
- Quiz-1 during Thursday's lecture
  - **Syllabus**: Lectures 02, 03, 04

Reading Materials

- Decomposition techniques
  - http://users.atw.hu/parallelcomp/ch03lev1sec2.html

Acknowledgements

- Several of the slides used in this course are borrowed from the following online course materials:
  - Course COMP322, Prof. Vivek Sarkar, Rice University
  - Course COMP 422, Prof. John Mellor-Crummey, Rice University
  - Course CSE539S, Prof. I-Ting Angelina Lee, Washington University in St. Louis
- Contents are also borrowed from following sources:
  - “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003
  - https://computing.llnl.gov/tutorials/parallel_comp/
  - https://images.google.com/