COL380

Introduction to
Parallel & Distributed Programming

Recap

- Hierarchical core, memory and network organizations
  - ➡ Parallelism at multiple levels
- SIMD vs MIMD
- Compute-unit addressing, network distance, resilience to failure
- Code-supported routing (data forwarding)
- Unified and non-unified addressing
- Non-uniform memory access

A Simple Abstraction of Parallelism

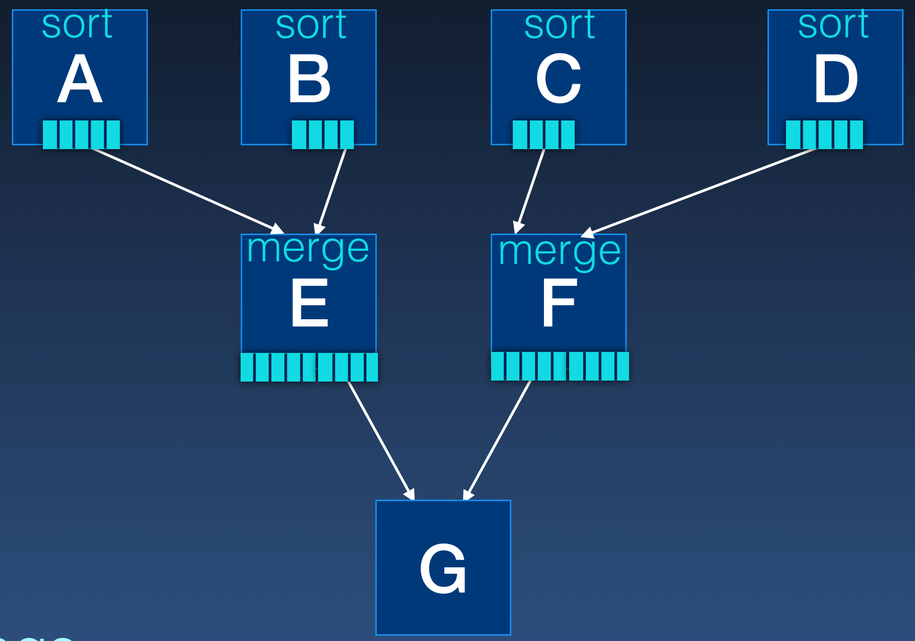
- Many threads of execution
  - ➡ Execute independently with others
    - ▸ Interaction is indirectly through shared memory Or directly between threads
  - ➡ Each instruction takes finite (non-deterministic) time to complete
- Unified memory Or distributed
  - ➡ A thread may (or may not) view the impact of memory operations of threads
    - ▸ at an undetermined time
      - unless ‘synchronized’

Parallel & Distributed Program Concerns

- Devise parallel algorithm
  - ➡ Break into independent tasks (statically, dynamically)
- Initiate threads of execution
  - ➡ How many (statically or dynamically determined)
- Map tasks to threads of execution
  - ➡ Who does what when
- Inter-task interactions (+ Inter thread interaction)
  - ➡ Send message, Wait for others, Shared state
- How to determine completion

Parallel Programs

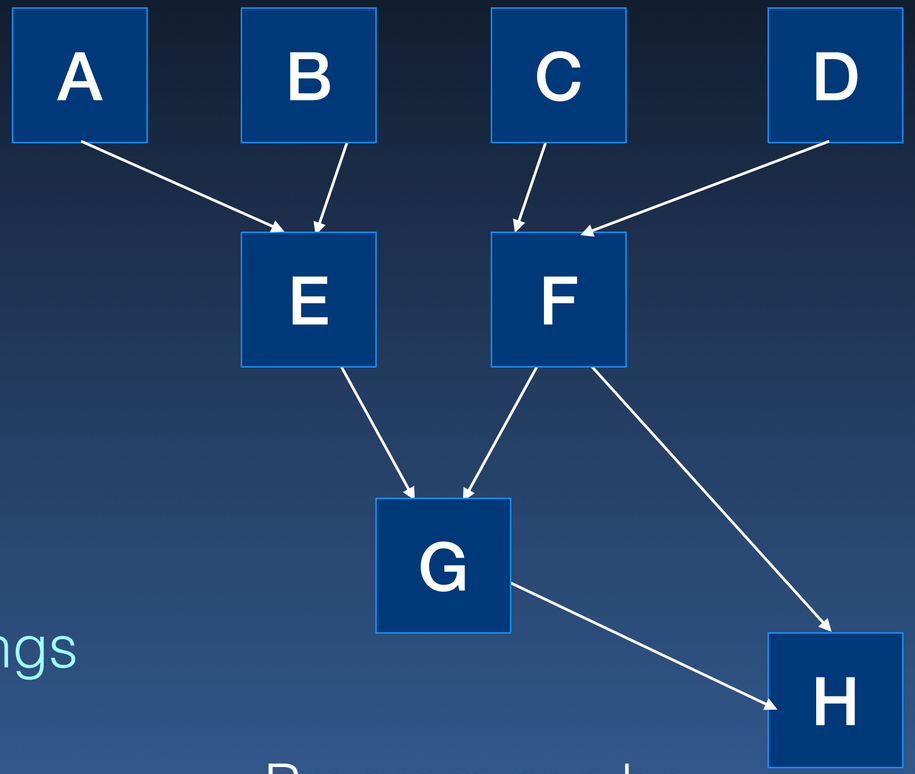
- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Program can be represented as a graph

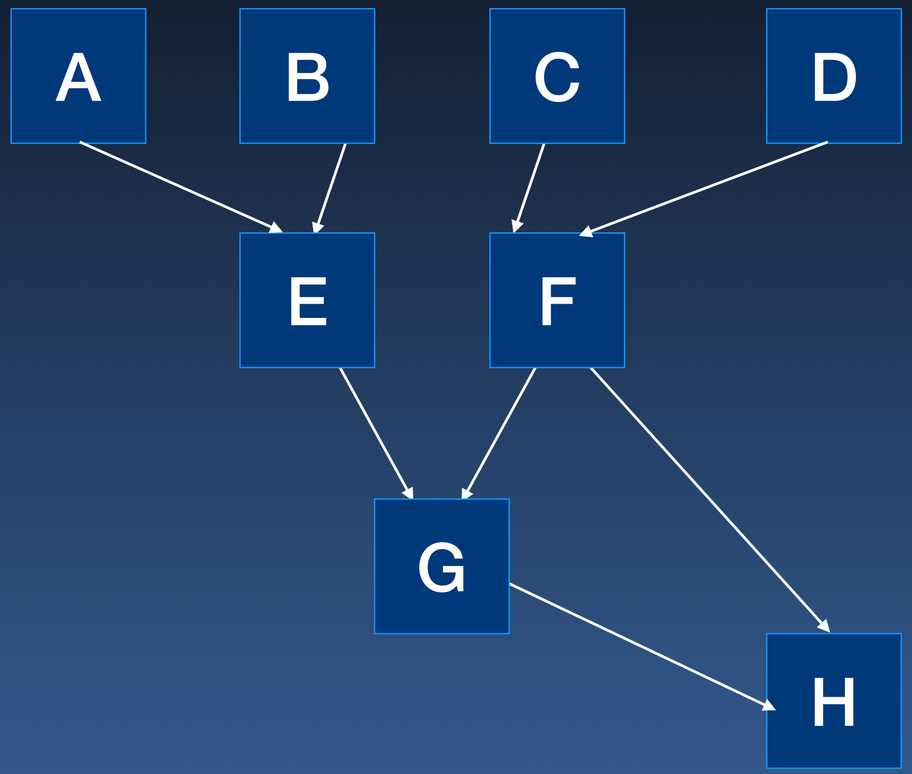
Parallel Programs

- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Program can be represented as a graph

(Dependency)
Task Graph



- Granularity
- Critical Path Length
- Maximum Concurrency
- Average Concurrency

$$= \frac{\textbf{\#tasks}}{\textbf{Critical path length}}$$

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph model
- Fork-join model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

```
void processAB (int a[], int b[], int n)
{
    for(int i=0; i<n; i += 3) {
        #pragma omp task  depend(out: x)
            subtaskA(a+i);
        #pragma omp task  depend(in: x)
            subtaskB(b+i);
    }
}
……
#pragma omp task
    processAB(a, b, n);
```