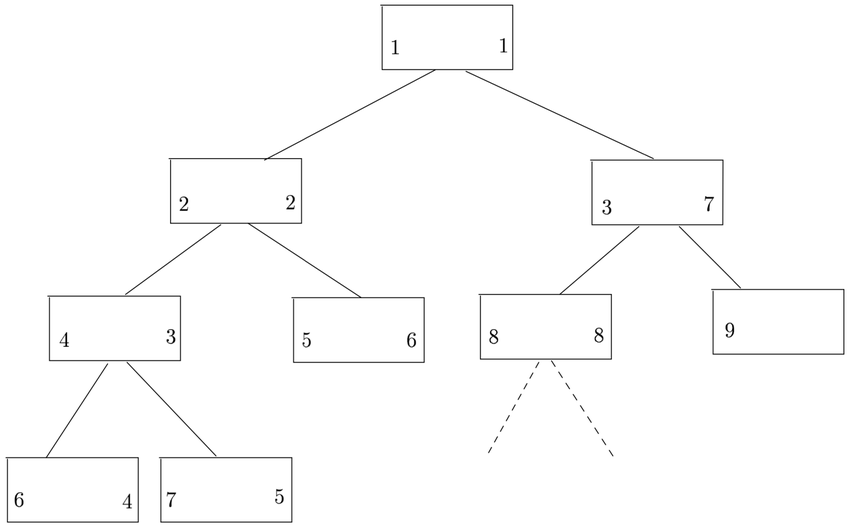
Parallel Depth-First Search for Directed Acyclic Graphs

Dharani Srinivas
25th January 2024

- **Introduction**

The Depth-first search (DFS) is a general technique used in Artificial Intelligence for solving a variety of problems in planning, decision making, theorem proving, expert systems, etc. It is also used under the name of backtracking to solve various combinatorial problems and constraint satisfaction problems. Execution of a Prolog program can be viewed as depth-first search of a proof tree. Iterative-Deepening DFS algorithms are used to solve discrete optimisation problems and for theorem proving. A major advantage of the depth-first search strategy is that it requires very little memory. Since many of the problems solved by DFS are highly computation intensive, there has been a great interest in developing parallel versions of depth-first search. We have developed a parallel formulation of the depth-first search which retains the storage efficient of DFS.

- **Architecture of Simple Depth-First Search**



- **Applications**

  - DFS is commonly used for various applications, such as topological sorting, connected components, cycle detection, and pathfinding.
  - Topological Sorting: Ordering DAG vertices consistently with edge dependencies.
  - Critical Path Analysis: Identifying the longest path in a DAG, crucial for task scheduling.
  - Connectivity Testing: Determining if all vertices in a DAG are reachable from a source vertex.
  - Cyclicity Detection: Checking if a directed graph contains cycles (not applicable to acyclic DAGs).
  - Dependency Resolution: Managing software dependencies based on build graphs.
  - Dataflow Pipelines: Modeling and executing complex computations as DAGs.

- **Parallel Depth First Search**
  - **Algorithm**
    - We parallelise DFS by sharing the work to be done among a number of processors. Each processor searches a disjoint part of the search space in a depth-first fashion. When a processor has finished searching its part of the search space, it tries to get an un searched part of the search space from the other processors. When a goal node is found, all of them quit. If the search space is finite and has no solutions, then eventually all the processors would run out of work, and the (parallel) search will terminate.
  - **Architecture**

- **Parallelism Approaches**
  - **Data Parallelism**: Exploits multiple processers/threads to explore different vertices concurrently, often limited by dependencies in DAG's.
    - BFS-Based Exploration: Leverages level-wise expansion, but may require frequent synchronization.
    - Wavefront Parallelism: Utilizes topological levels based on vertex completion time, efficient for sparse DAGs.
    - Batch Exploration: Groups vertices with similar properties for coordinated exploration.
  - **Task Parallelism**: Divides the DFS traversal into independent tasks (e.g., exploring subtrees), suitable for highly branching DAGs.
    - Dynamic Task Generation: Creates tasks on-demand, potentially creating overhead.
    - Static Task Partitioning: Divides the graph into partitions beforehand, but may lead to load imbalance.

- **Platforms**
  - Multi-core processors, parallel computing clusters, or distributed computing environments can be suitable platforms.
  - CUDA: NVIDIA GPUs for data-parallel computations.
  - OpenMP: Shared-memory multiprocessors.
  - MPI: Distributed-memory systems.
  - Graph Processing Frameworks (e.g., Apache Spark GraphX): High-level abstractions for distributed graph algorithms.
- **Timeline**
  - **Parallelisation Development**:
    - Designing and implementing parallel DFS algorithms: 1 month.
  - **Testing and Optimisation**:
    - Ensuring correctness, optimising for performance: 1-2 months.

- **Challenges**
  - **Load Balancing**: Balancing the workload across processors to maximise parallel efficiency.
  - **Memory Bottlenecks**: Large graphs may not fit entirely in local memory, requiring careful data movement and caching strategies.
  - **Synchronisation Overhead**: Managing dependencies and maintaining coherence in parallel exploration can be challenging, especially for sparse DAGs.
  - **Communication Overhead**: Minimising communication overhead for synchronisation and data sharing.
  - **Scalability:** Ensuring efficient parallel performance as the size of the graph and the number of processors increase.

- **References**

Vipin Kumar and V. Nageshwara Rao, Parallel Depth-first Search, Part I: Analysis,International Journal of Parallel Programming,16(6):501–519 (1988).

Vipin Kumar and V. Nageshwara Rao, Parallel Depth-first Search, Part II: Analysis,International Journal of Parallel Programming,16(6):501–519 (1988).

Parallel Depth-First Search for Directed Acyclic Graphs Maxim Naumov1, Alysson Vrielink1,2, and Michael Garland1 1NVIDIA, 2701 San Tomas Expressway, Santa Clara, CA 95050 2Stanford University, 2575 Sand Hill Road, Menlo Park, CA, 94025