Parallel Depth-first (PDF) Scheduling

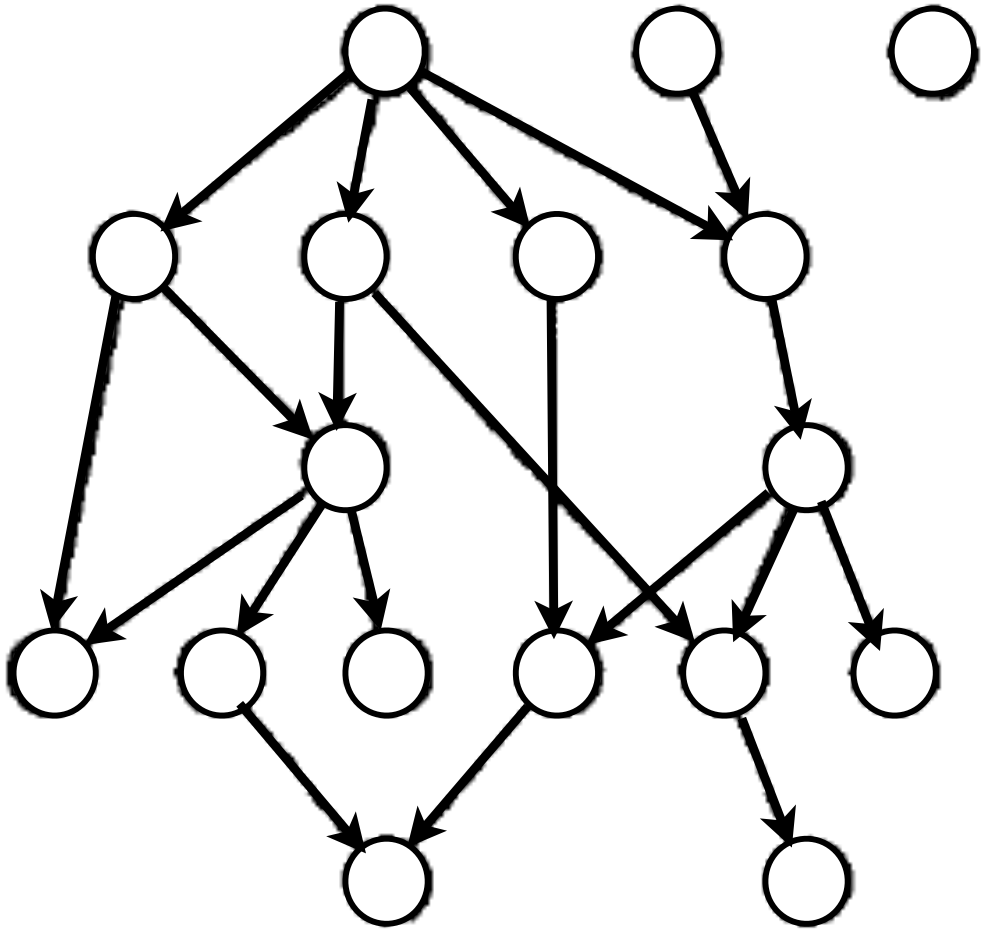
John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu

Computation DAGs and Scheduling

DAG Model for Computations



DAG Scheduling

- Map a DAG computation onto P threads
- Determine what executes on each thread at each time step
- A schedule must obey the following constraints
  - —each node appears only once in a schedule
  - —all predecessors of a node must execute before a node executes
    - node is ready at time step t if its ancestors executed at times ≤ t - 1
  - —any ready node may be scheduled
- Valid schedules must exist since computation graph is a DAG

Dynamic DAGs and Online Scheduling

A DAG is incrementally revealed as execution proceeds

- **When a node is scheduled, its outgoing edges are revealed**
- **When all its incoming edges are revealed, a node is revealed and available for scheduling**
- **Online scheduling: based only on revealed nodes**

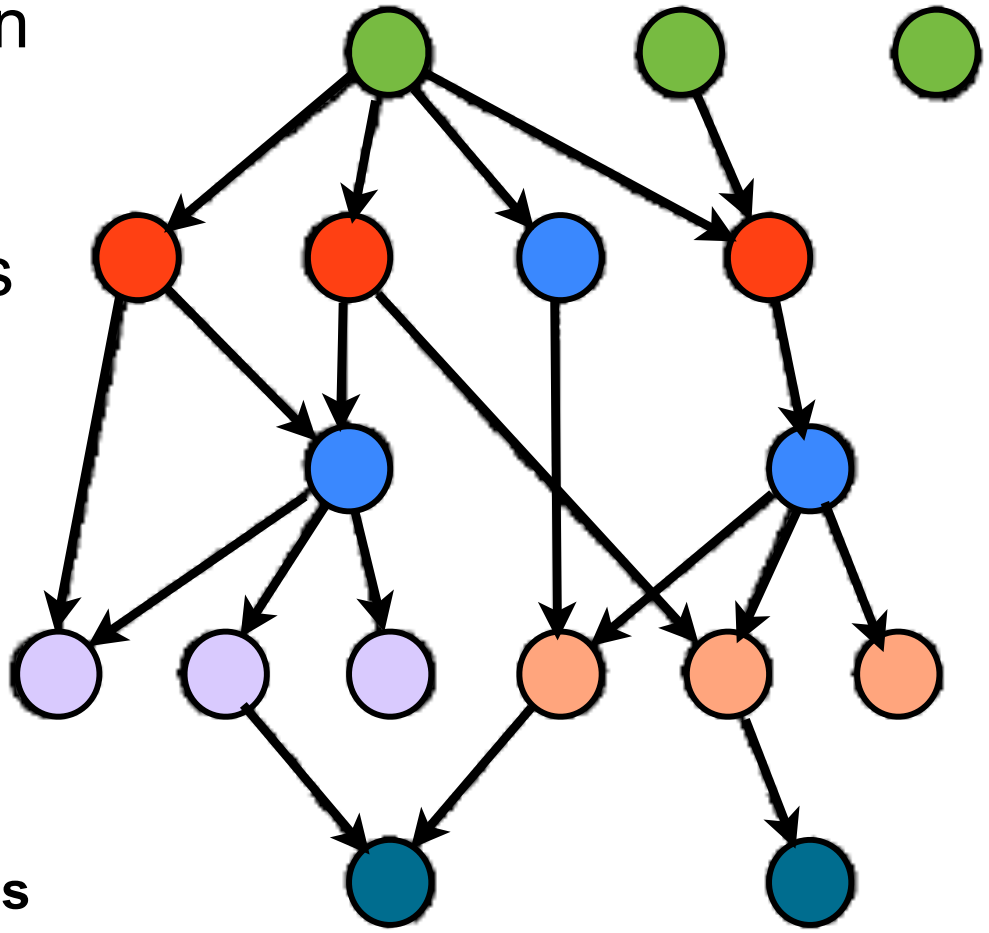
Greedy Scheduling

- **Definition: At each step in an execution ...**
  - **if P tasks are ready, then P execute**
  - **if fewer than P tasks are ready, all execute**
- **Theorem**
  - **for any multithreaded computation with work T_1 and DAG depth T_∞, for any number P of processors, any greedy schedule achieves $T_P \leq T_1/P + T_\infty$**
- **Commentary**
  - **generally, want linear speedup, i.e., $T_P = O(T_1/P)$**
  - **greedy schedules achieve linear speedup when avg. parallelism, defined as $T_1/T_\infty = \Omega(P)$; namely $T_1/T_\infty \geq cP$, or $T_1/P \geq cT_\infty$**
    - **if # processors is ≤ average parallelism, there will be enough work to keep the processors adequately busy for good speedup**
    - **too many processors leads to idleness, which degrades speedup**

A Greedy Schedule on 3 Processors

Schedule constraints

- Each node appears once in the schedule
- Each node is scheduled after all of its predecessors
- Each step consists of at most P nodes



$T_1 = 17$, $T_\infty = 5$

Parallel execution time $T_P = 6$ steps

$T_1/P + T_\infty = 17/3 + 5 = 10.66 \geq T_P = 6$

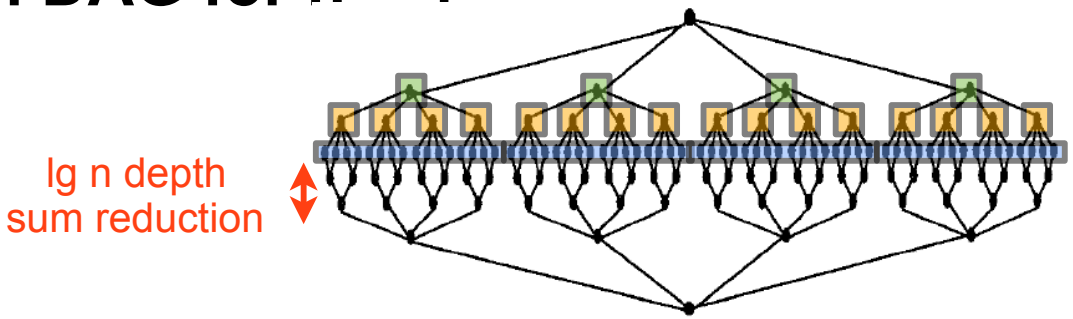
The Importance of Schedule Choice

Scheduling and space

- **Consider matrix multiplication with fine-grain parallelism**

```
In parallel for i from 1 to n
    In parallel for j from 1 to n
        r[i,j] = TreeSum(In parallel for k from 1 to n
                             a[i,k]*b[k,j])
```

- **Computation DAG for n = 4**



- **Sequential schedule uses only $O(n^2)$ space**
- **Level-by-level parallel schedule would use $O(n^3)$ space**

What Schedules are Best?

- Greedy parallel schedules deliver good speedup
  - —good bound on the number of steps
- Which greedy schedules (if any) have good bounds on space?
  - —memory and cache

Provably Efficient Scheduling for Languages with Fine-grain Parallelism

Effectively Sharing a Cache Among Threads

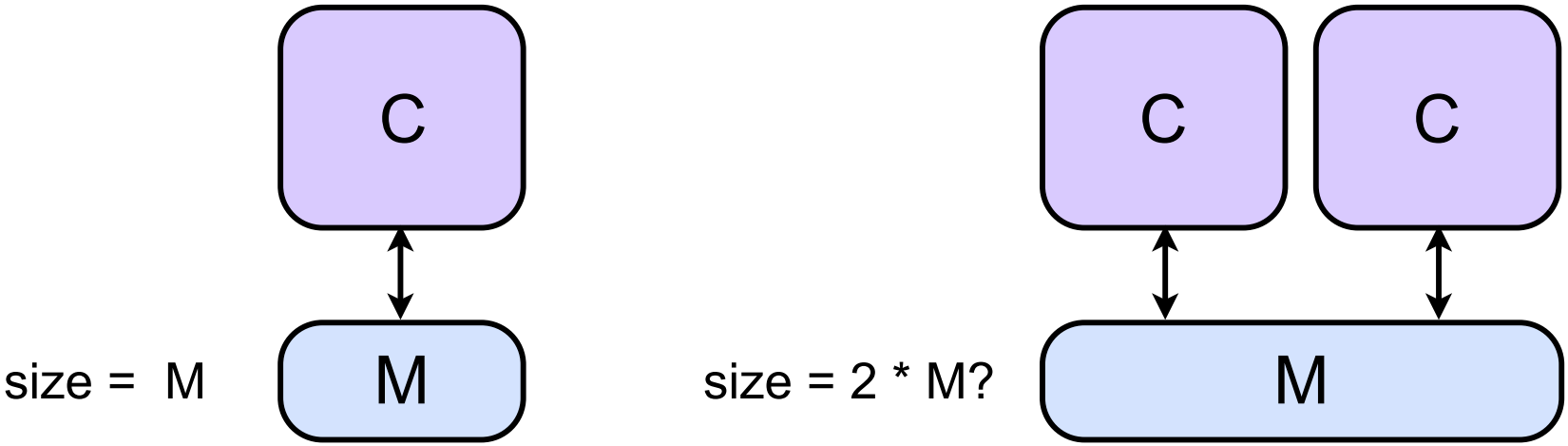
Space-efficient Scheduling of Nested Parallelism

NESL DAG Schedules

- Task structure is a dynamically unfolding series-parallel DAG
- Certain nodes can have arbitrary fan out or fan in
  - —fork or join nodes respectively
  - —more general than that of Blumofe and Leiserson
    - fanout 2

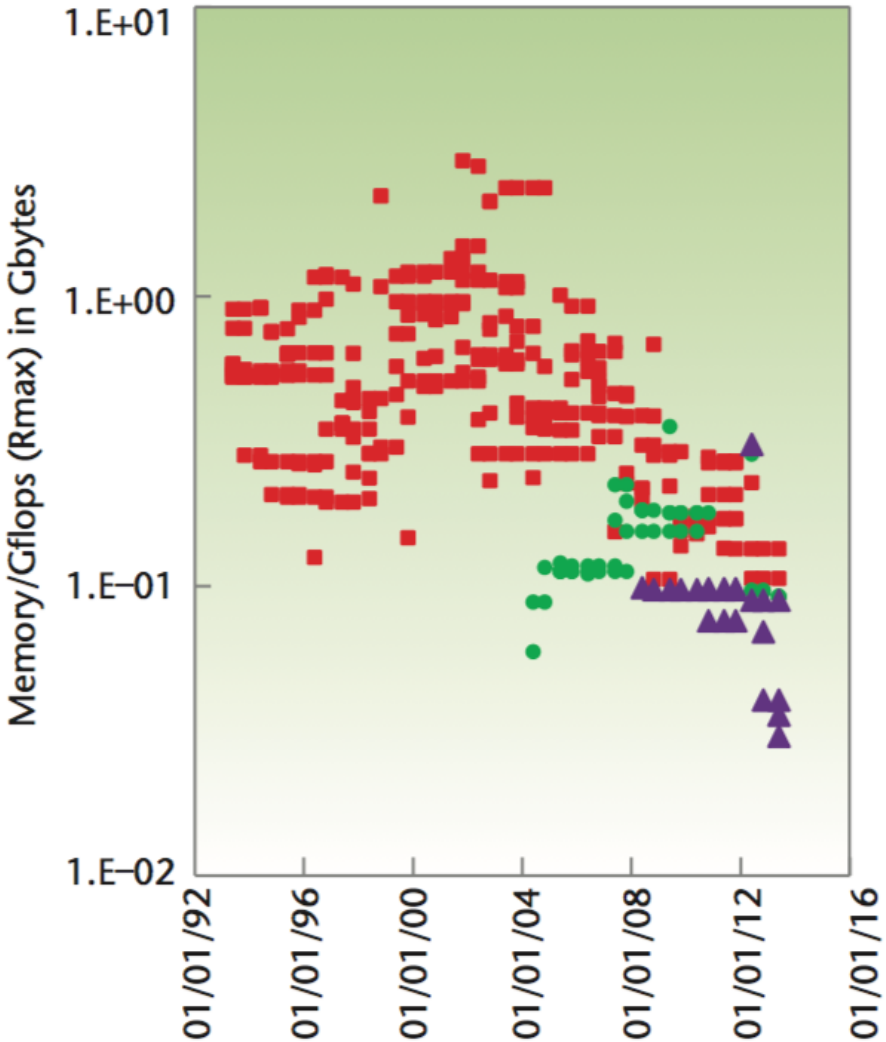
Motivation: Memory Footprint

- **Single core system uses memory of size M**
- **For a multicore with P cores, do we need memory of size PM?**



Why Does Memory Footprint Matter?

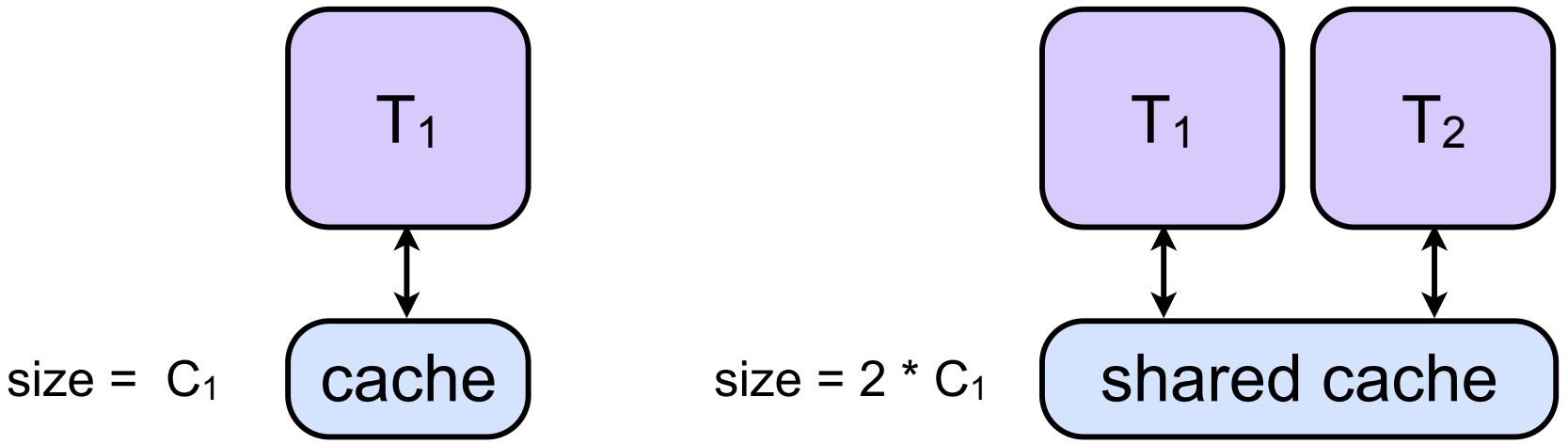
Decrease in relative memory capacity per FLOP/second



Peter Kogge and John Shalf. 2013. Exascale Computing Trends: Adjusting to the "New Normal"' for Computer Architecture. *Computing in Science and Engineering.* 15, 6 (November 2013), 16-26.

What about Cache Footprint?

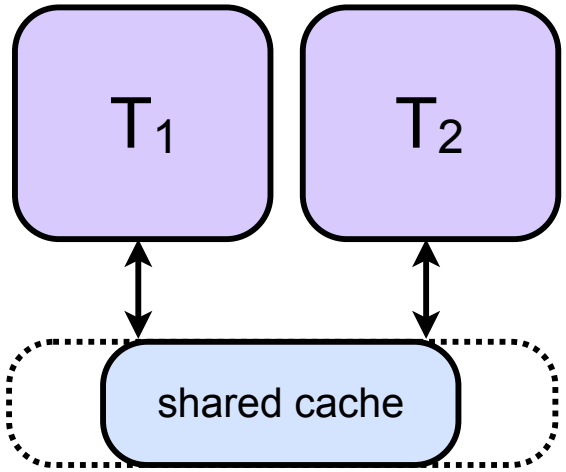
- **Single core system uses cache of size C_1**
- **For a core with T HW threads, do we need cache of size TC_1 ?**



Problem Formulation for Cache

- **Consider a parallel execution of a computation DAG**
  - **—on a multithreaded core with T threads and a shared ideal cache**
- **How much extra cache is needed for same hit rate as 1 thread?**

size = ?



Outline

- Efficient parallel DAG schedules
- Cache miss rate bounds
- Practical issues when implementing PDF schedules
- Value of PDF schedules
- Space-efficient scheduling of nested parallelism
- Summary

Toward Space-efficient Parallel Schedules

- Greedy parallel schedules evaluate nodes out of order with respect to a sequential execution
- Poor choice of schedule may require excessive space
  - —see the parallel matrix multiply example
- Three key ideas for space-efficient scheduling
  - —define a class of parallel schedules based on sequential ones
  - —focus on nodes scheduled "prematurely" ahead of their position in normal sequential order
  - —bound number of premature nodes to establish bounds on space

DAG Scheduling Strategies

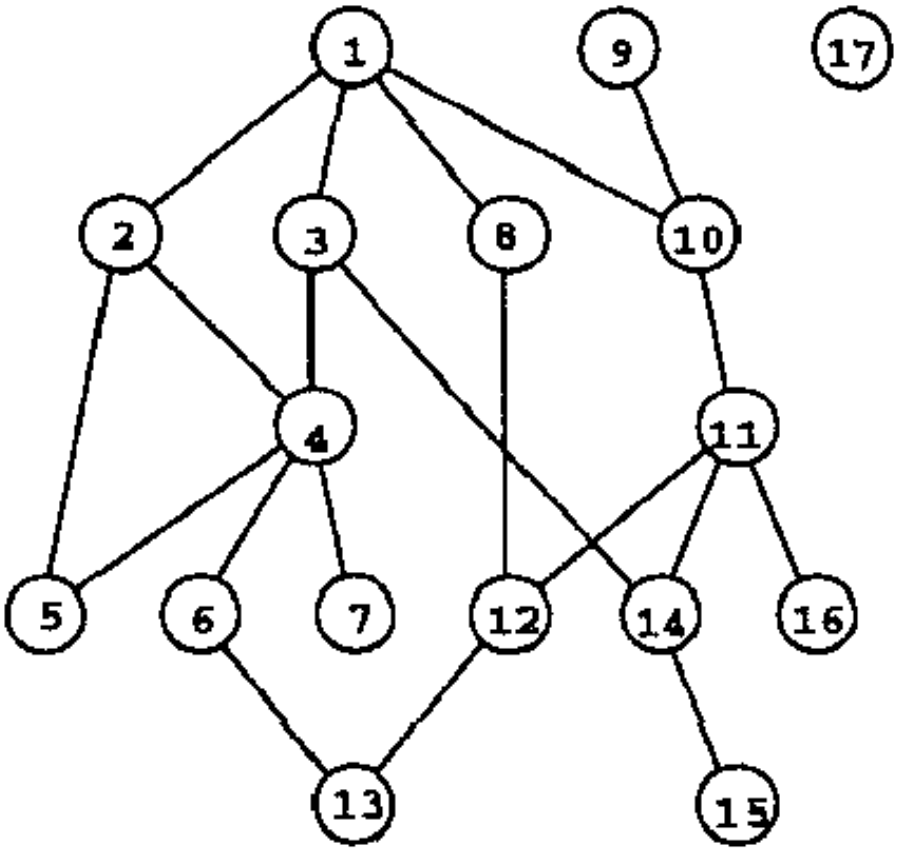
- 1DF schedule
  - —schedule one ready node per time step
  - —schedule in depth first order
    - simple using stack of ready nodes
- PDF schedule
  - —schedule up to P ready nodes per time step
  - —bias towards the 1DF ordering
    - if u precedes v in 1DF schedule, both are scheduled, neither are scheduled, or only u is scheduled

1DF Schedule

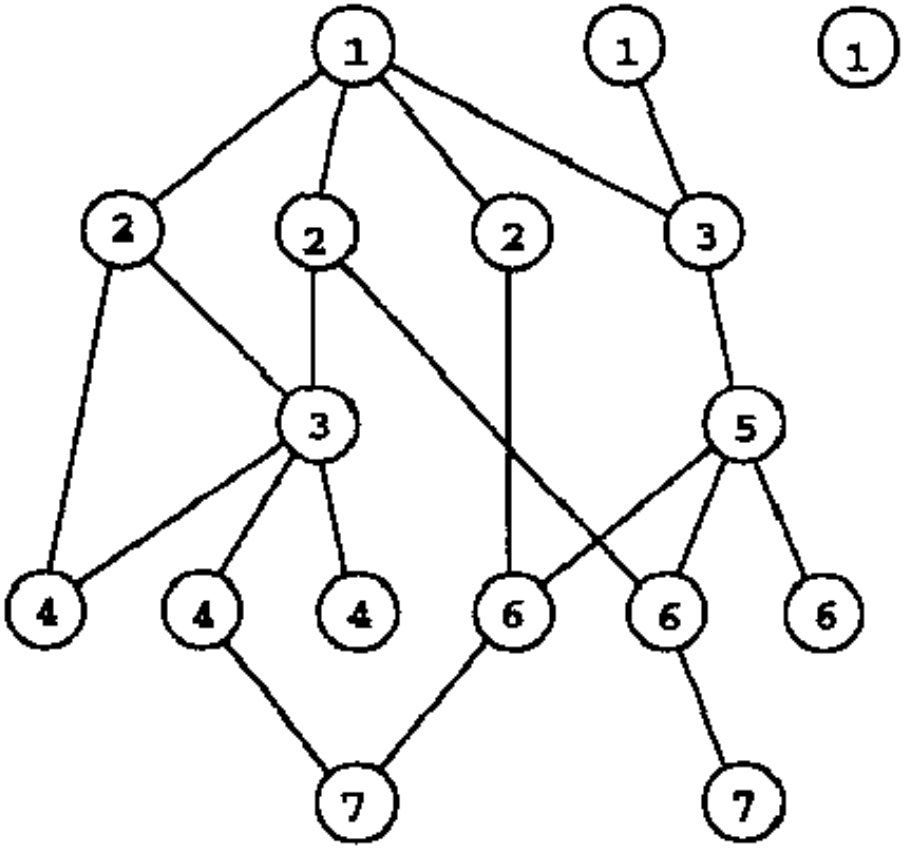
```
for each v in G (in any order)
    if v is ready (i.e., indegree == 0)
        push(v, stack)
while (v = pop(stack) != null)
    if v has not already been scheduled
        schedule(v)
        for each ready child u of v
            push(u, stack)
```

DAG Scheduling

1 DF Schedule

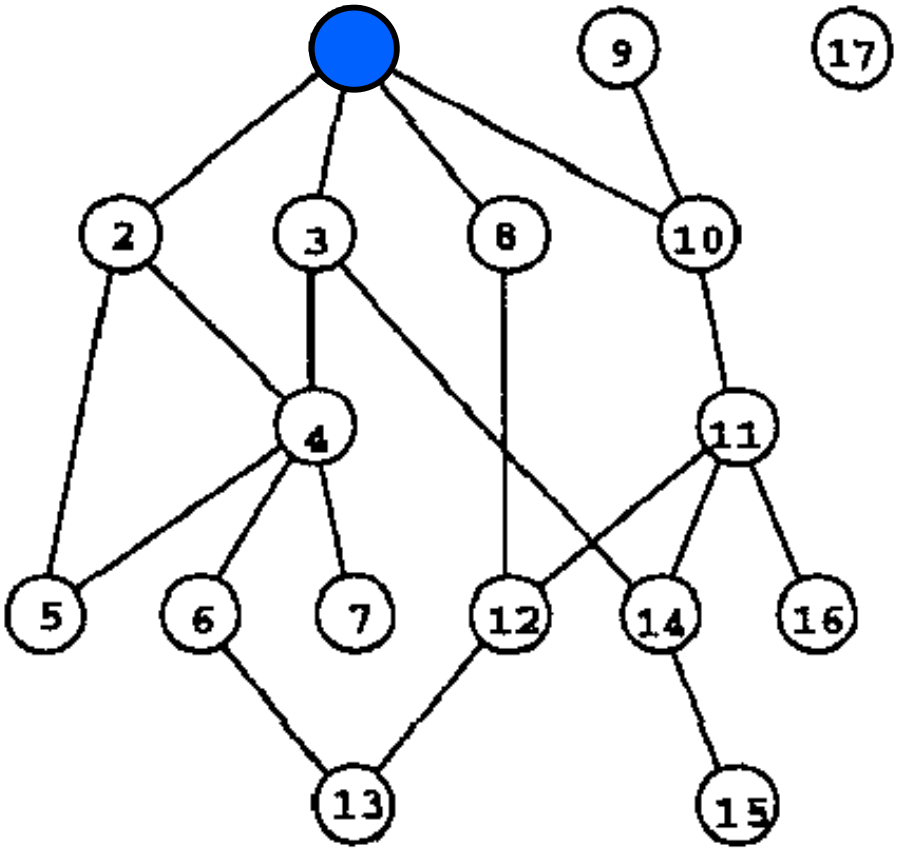


PDF Schedule for p=3

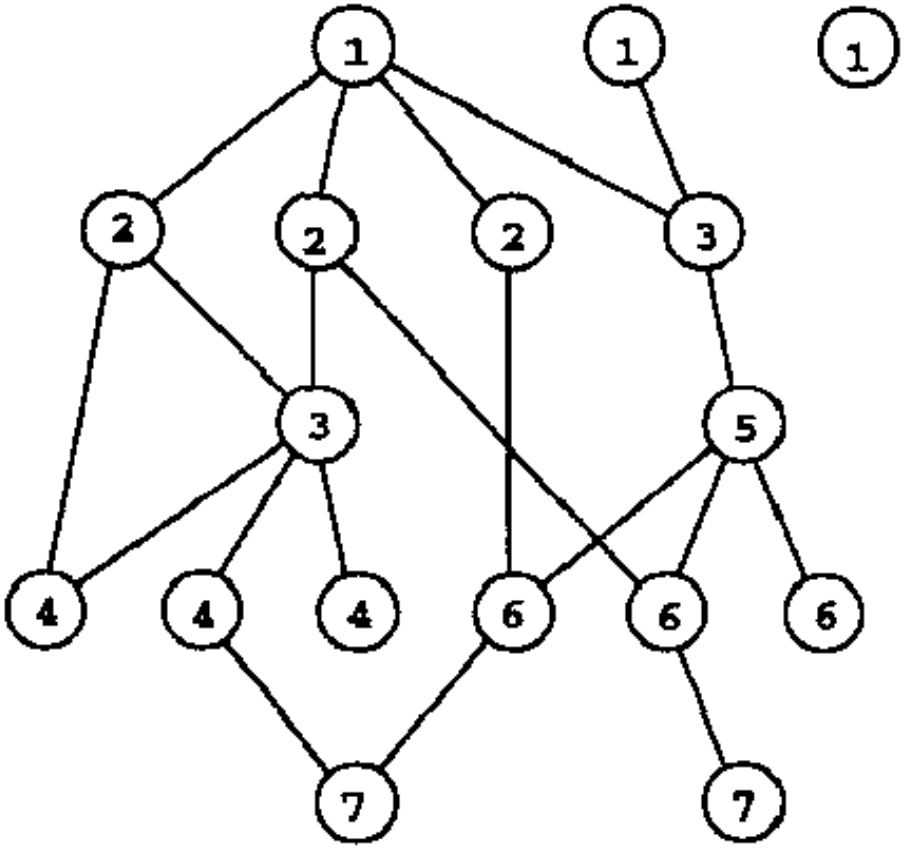


DAG Scheduling

1 DF Schedule

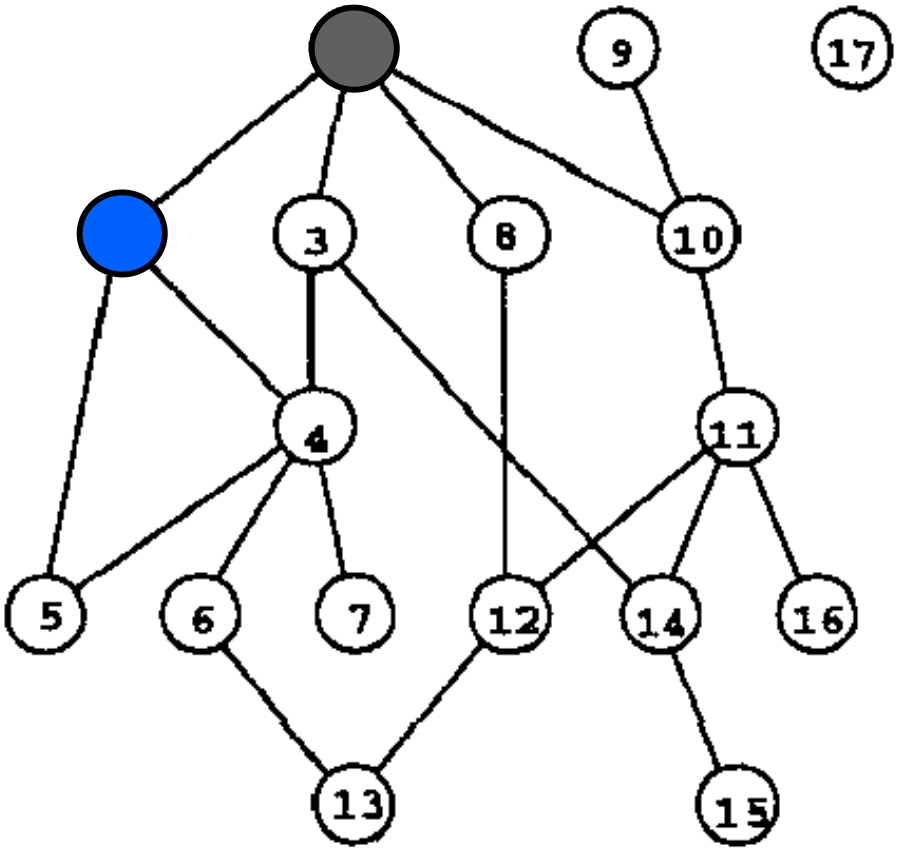


PDF Schedule for p=3

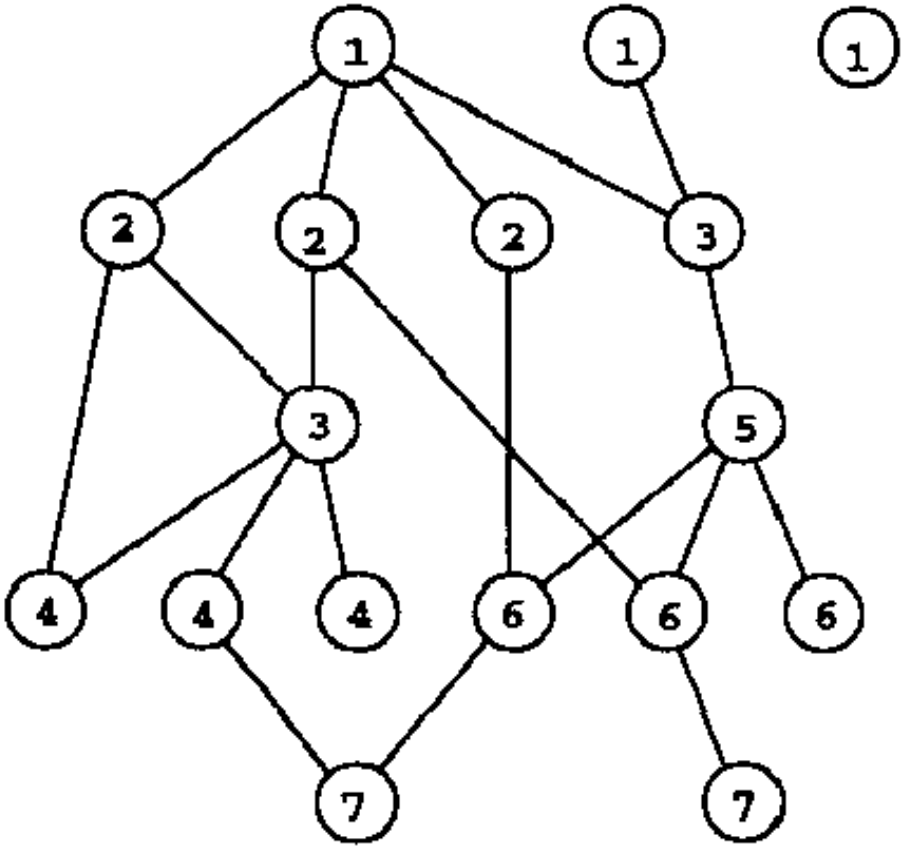


DAG Scheduling

1 DF Schedule

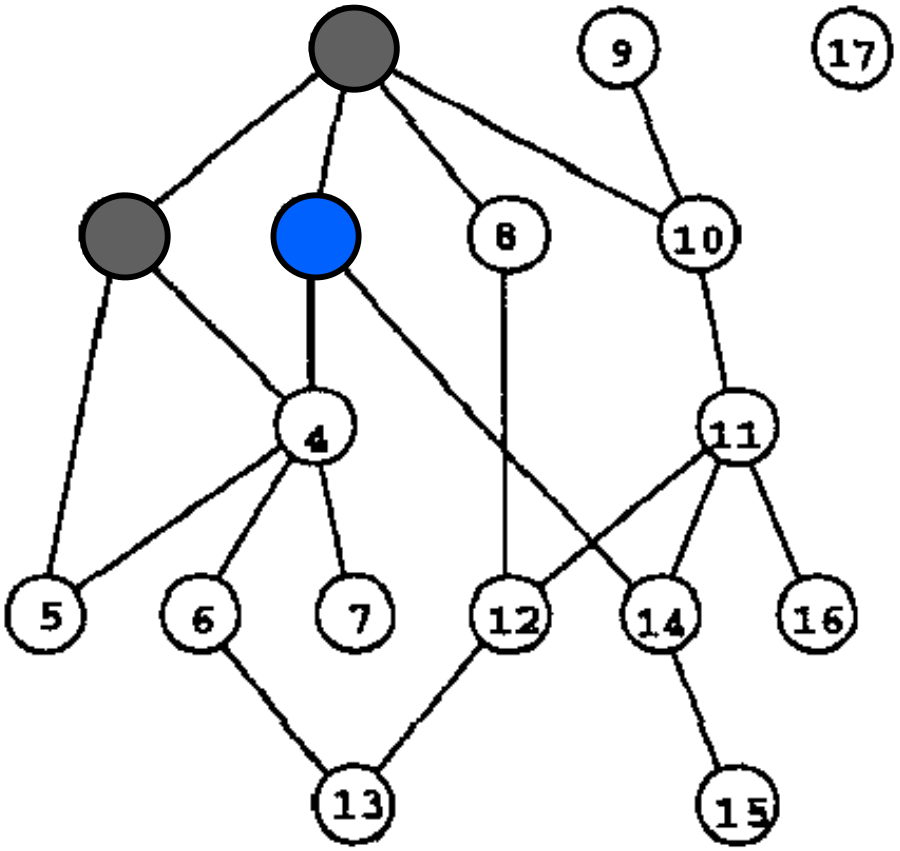


PDF Schedule for p=3

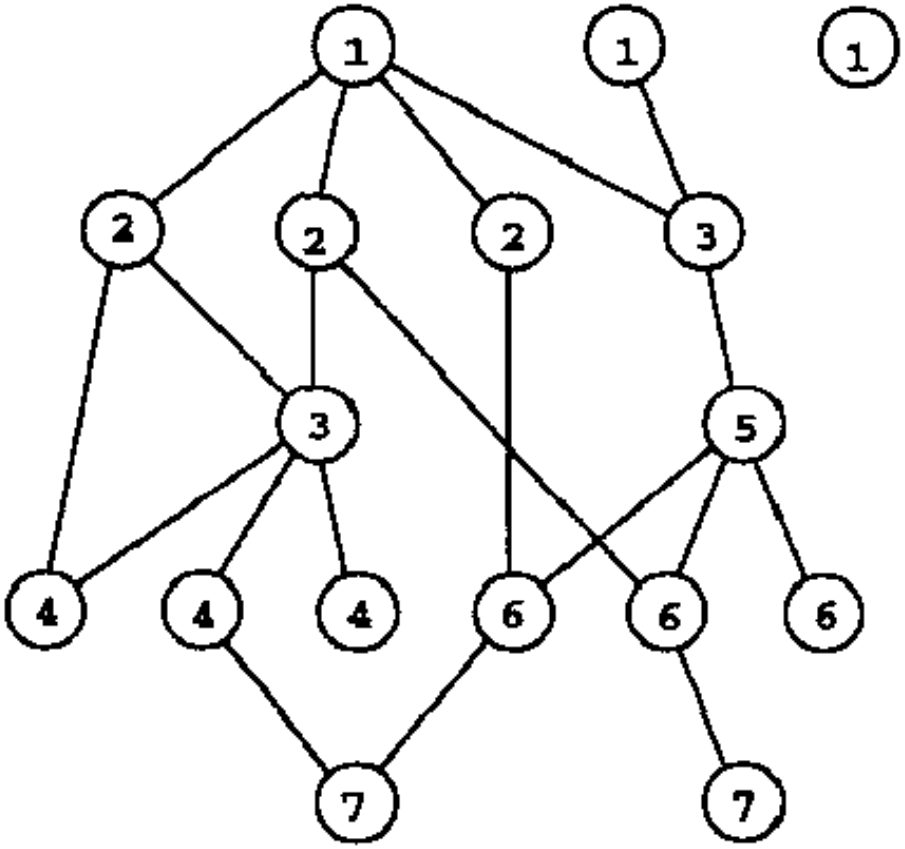


DAG Scheduling

1 DF Schedule

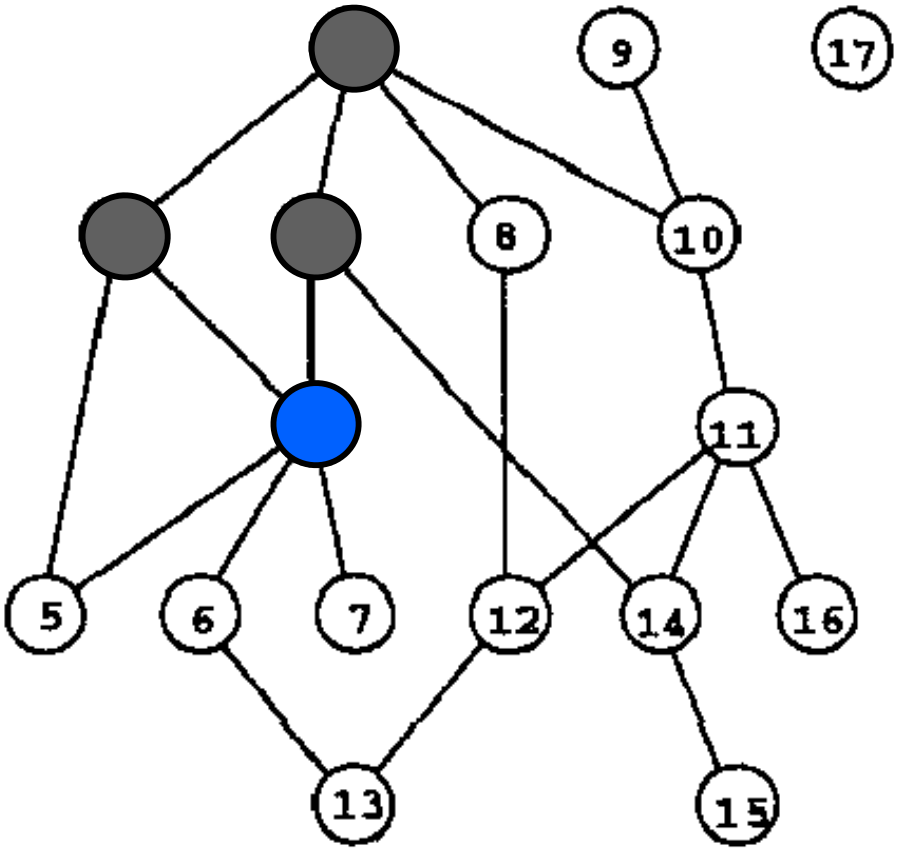


PDF Schedule for p=3

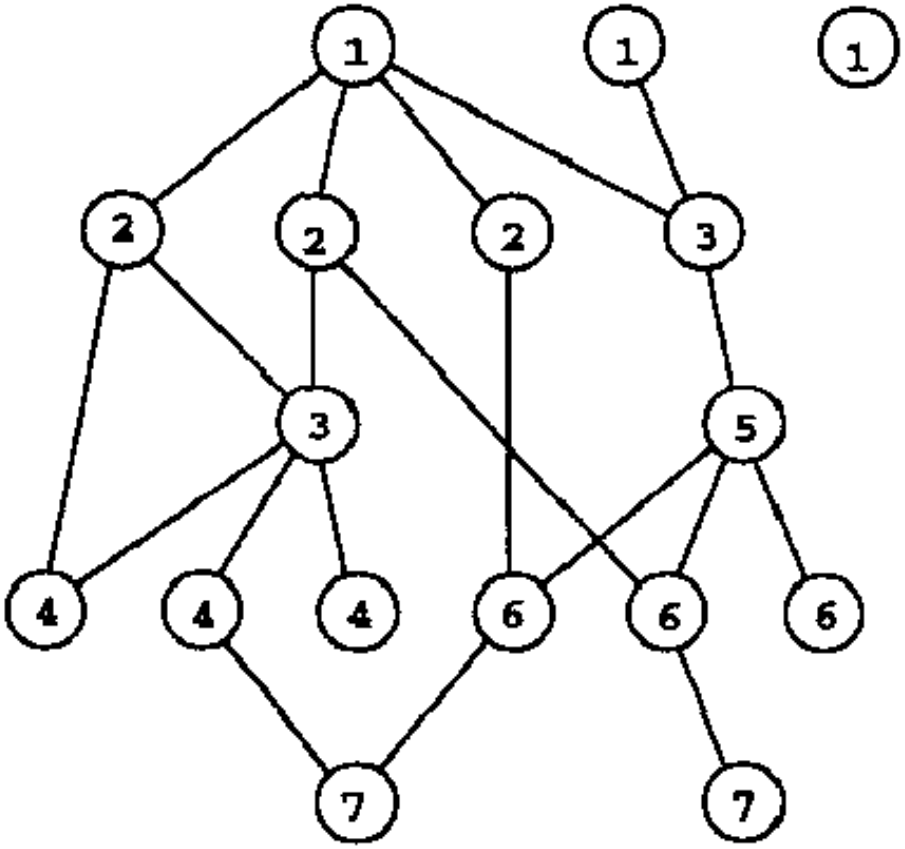


DAG Scheduling

1 DF Schedule

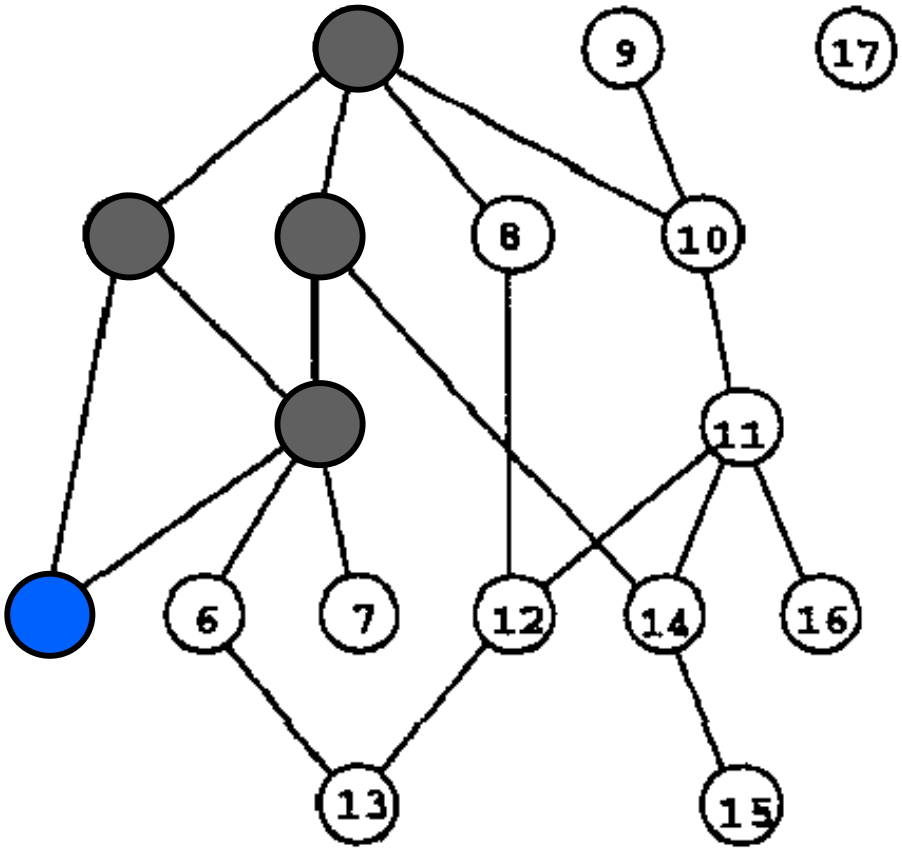


PDF Schedule for p=3

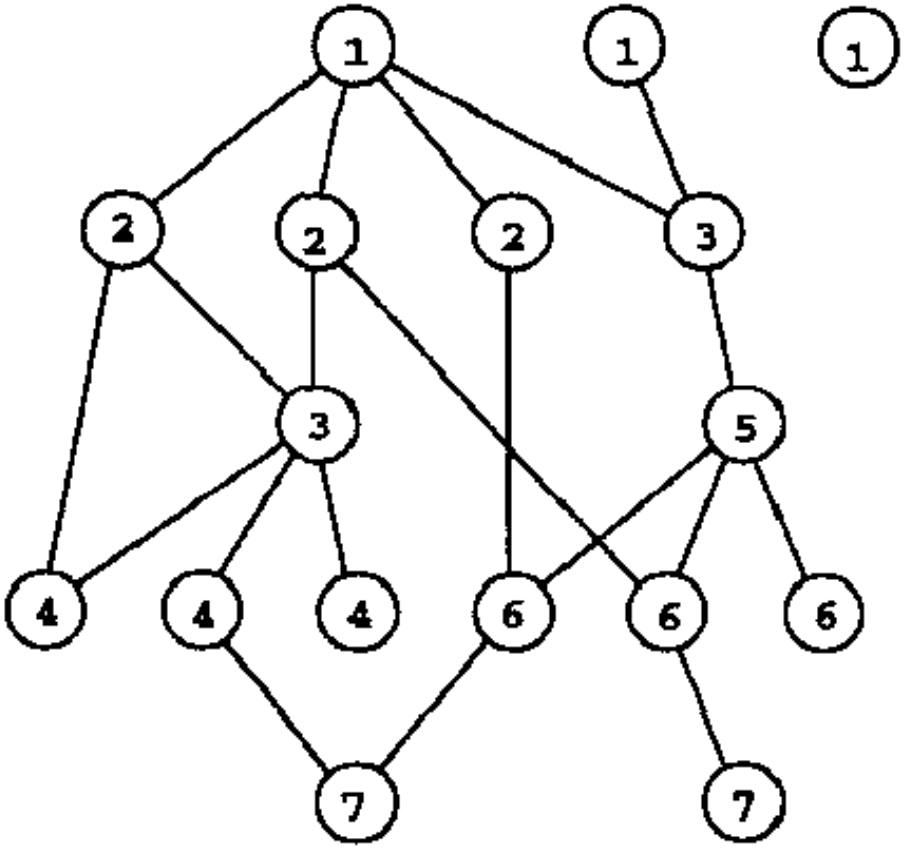


DAG Scheduling

1 DF Schedule

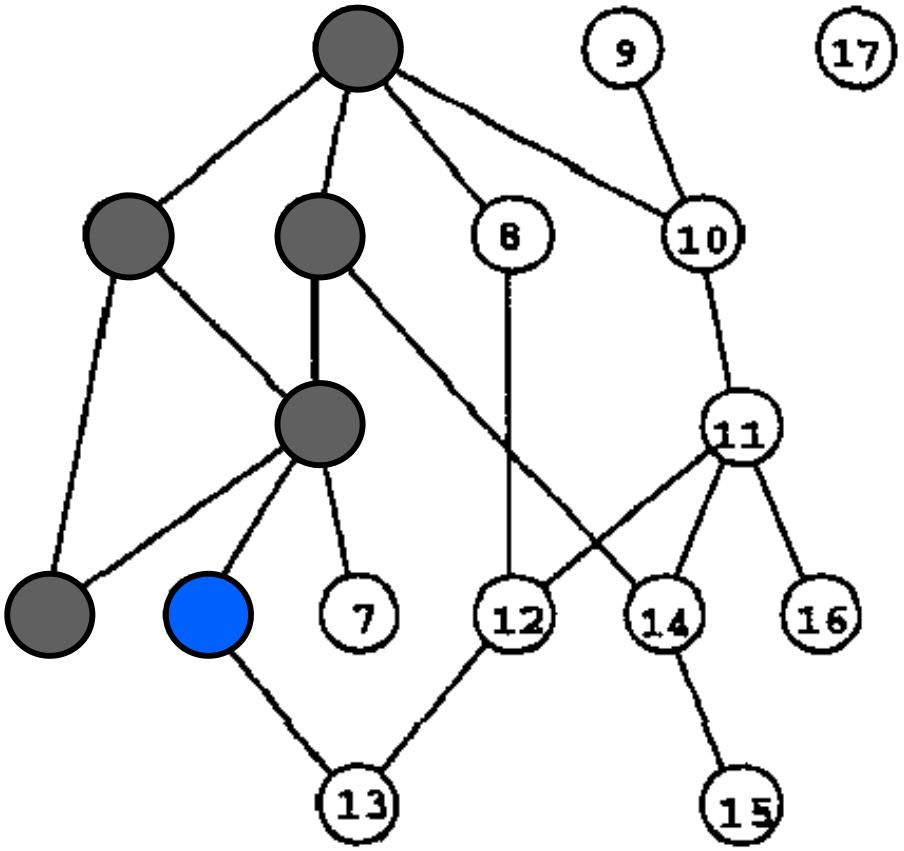


PDF Schedule for p=3

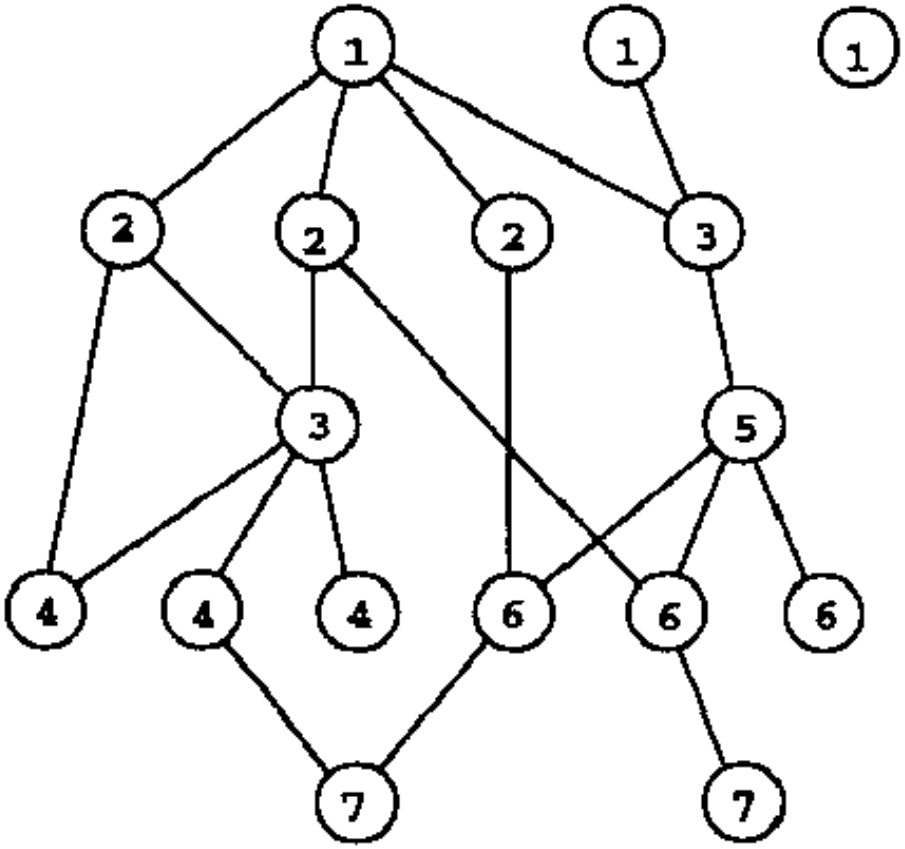


DAG Scheduling

1 DF Schedule

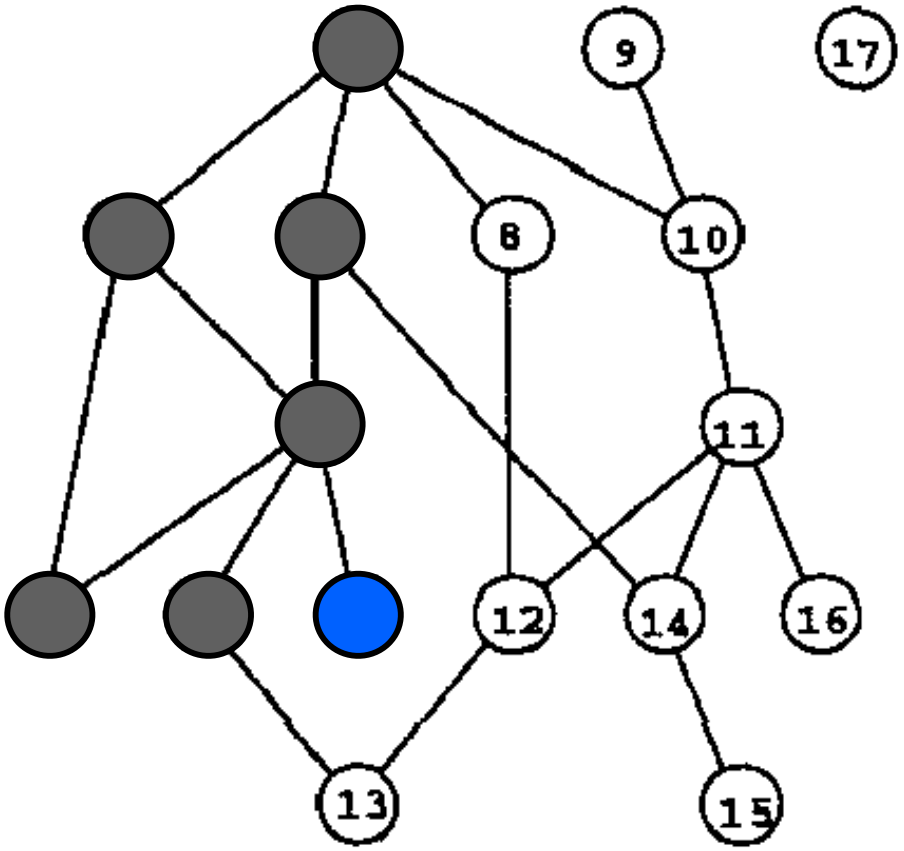


PDF Schedule for p=3

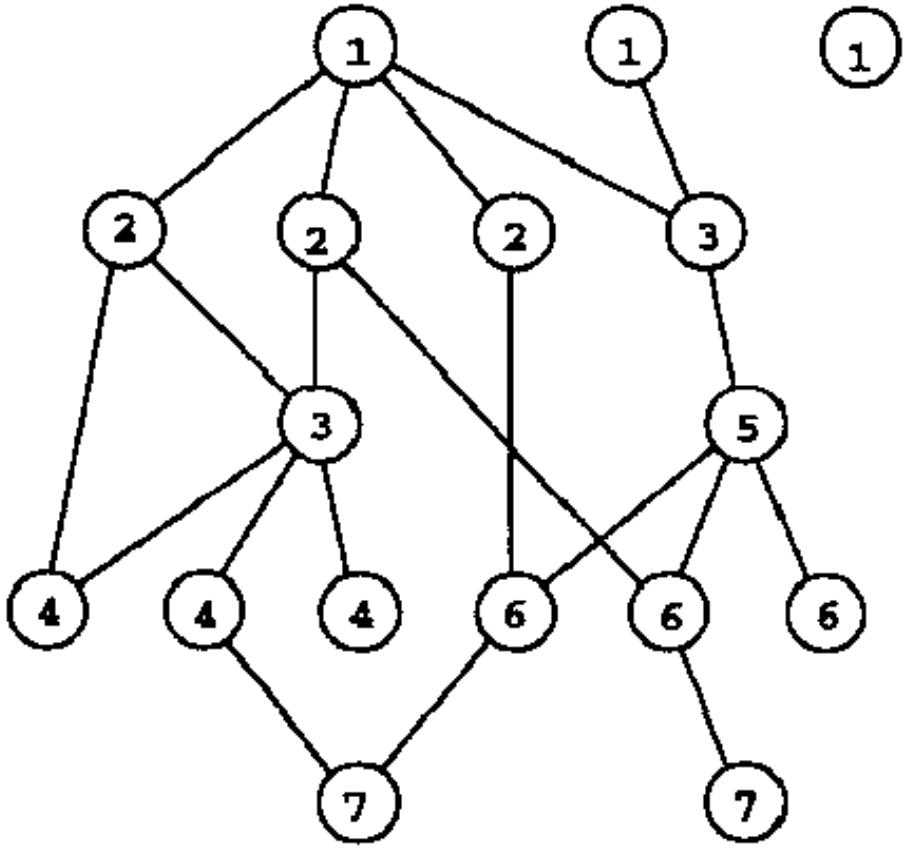


DAG Scheduling

1 DF Schedule

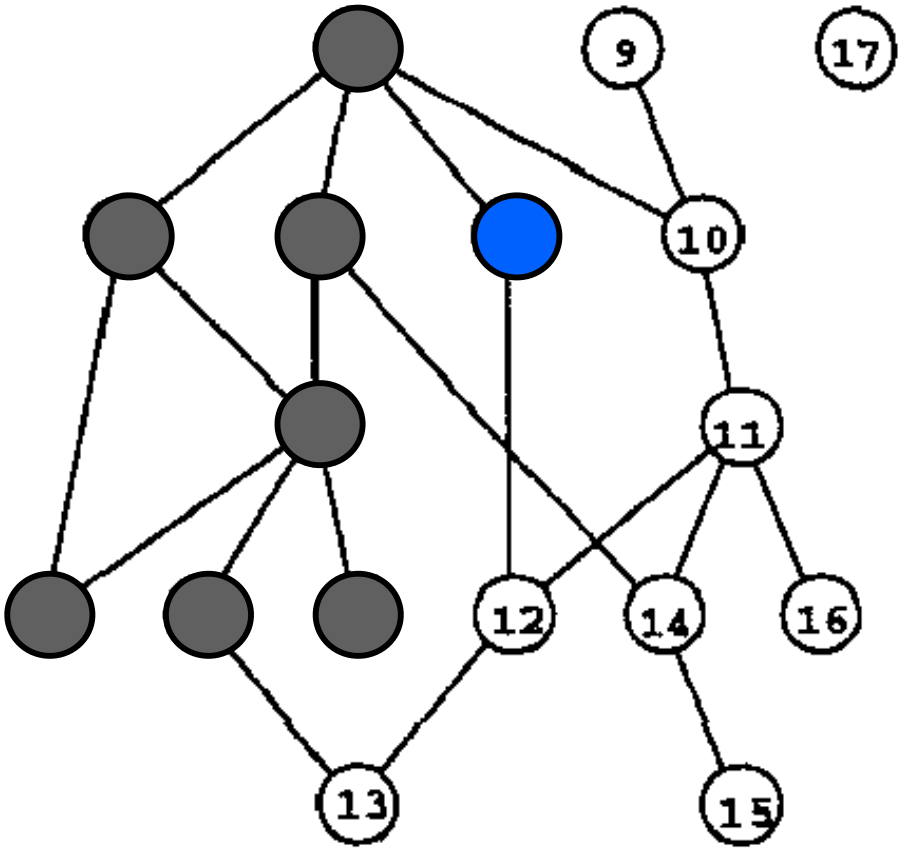


PDF Schedule for p=3

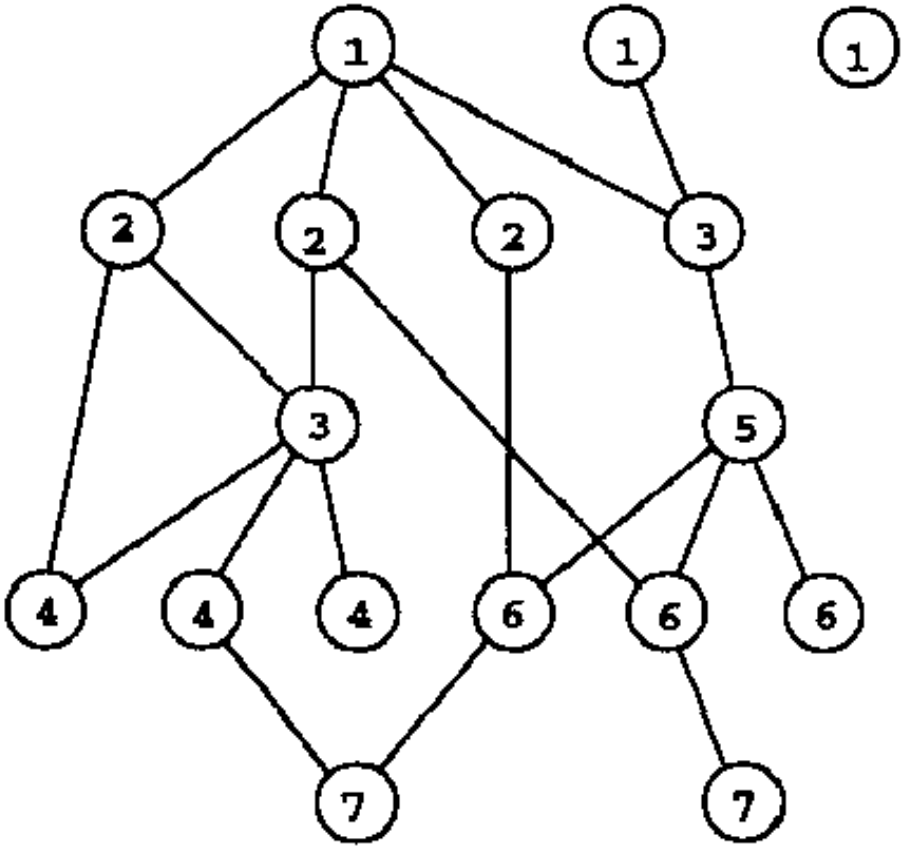


DAG Scheduling

1 DF Schedule

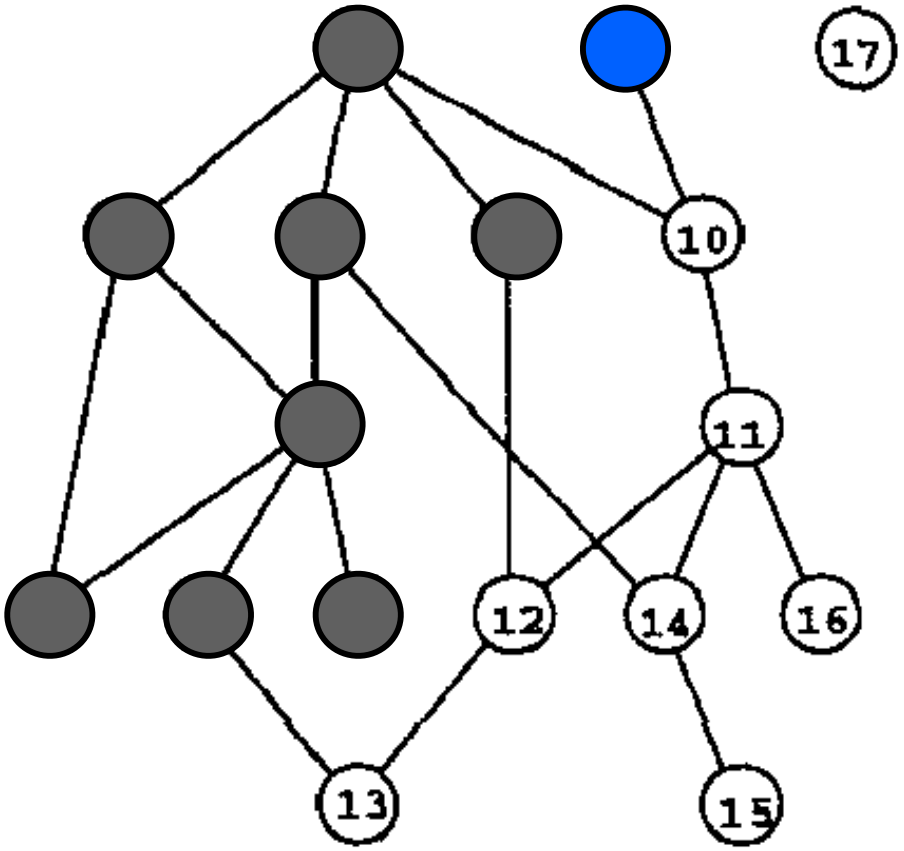


PDF Schedule for p=3

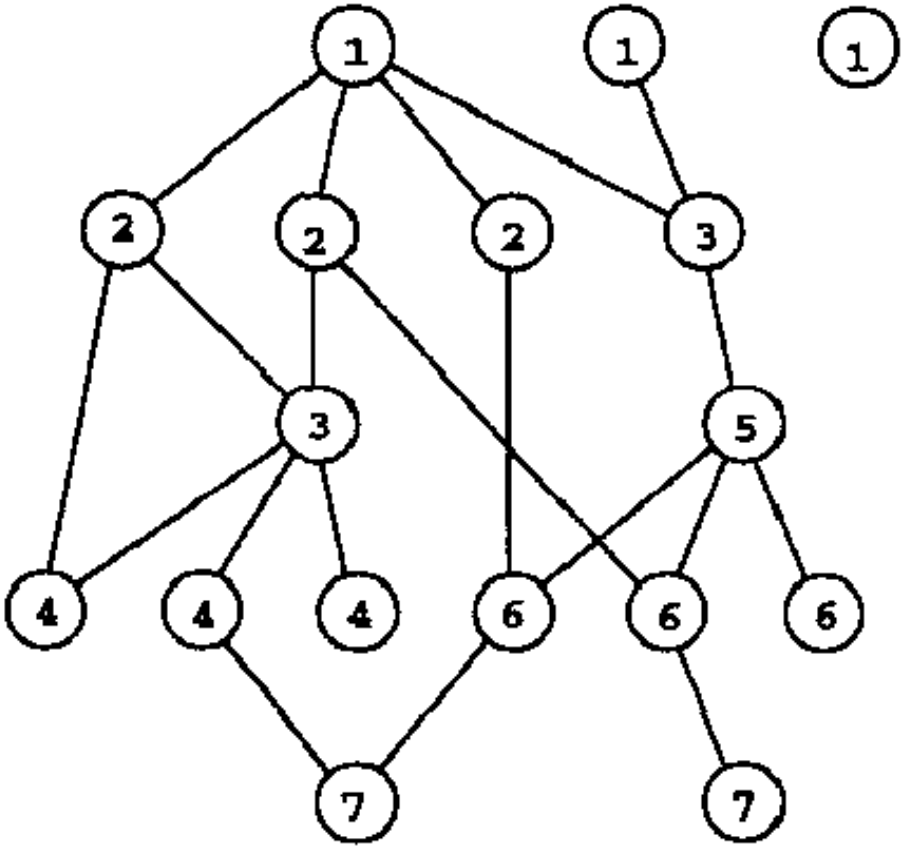


DAG Scheduling

1 DF Schedule

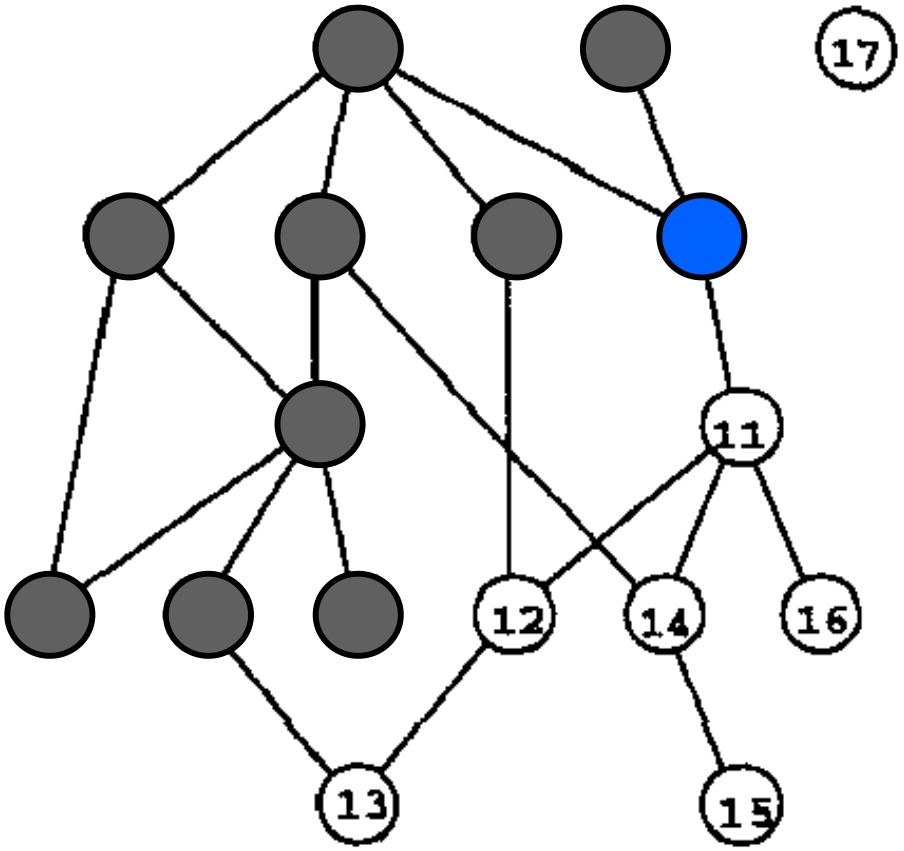


PDF Schedule for p=3

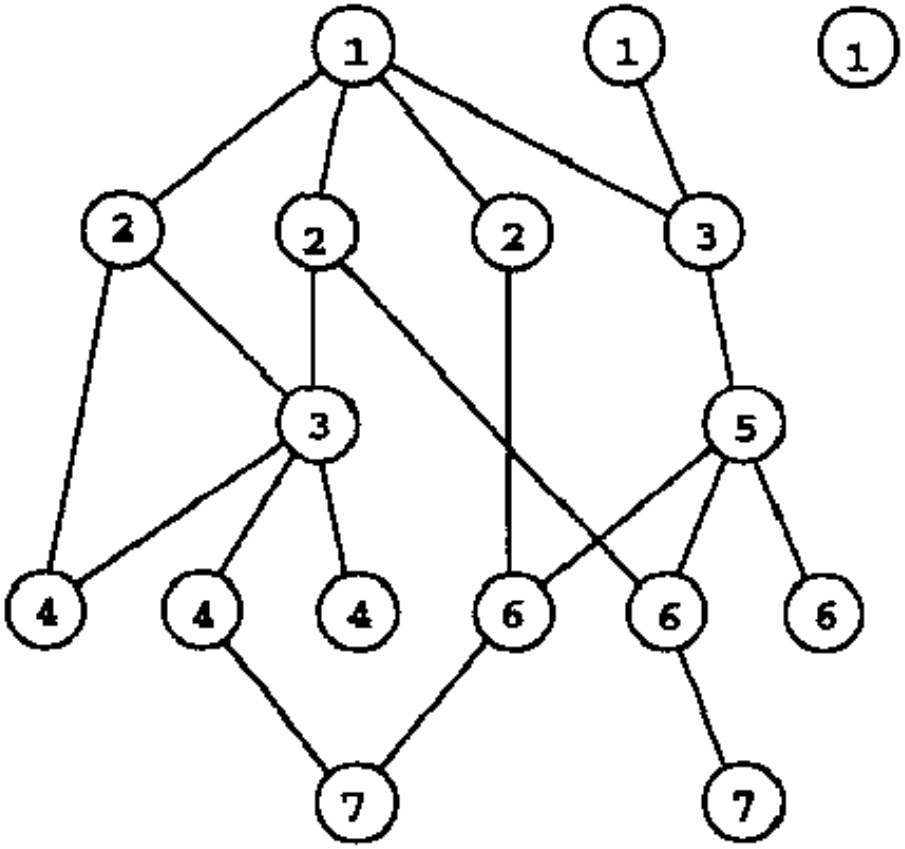


DAG Scheduling

1 DF Schedule

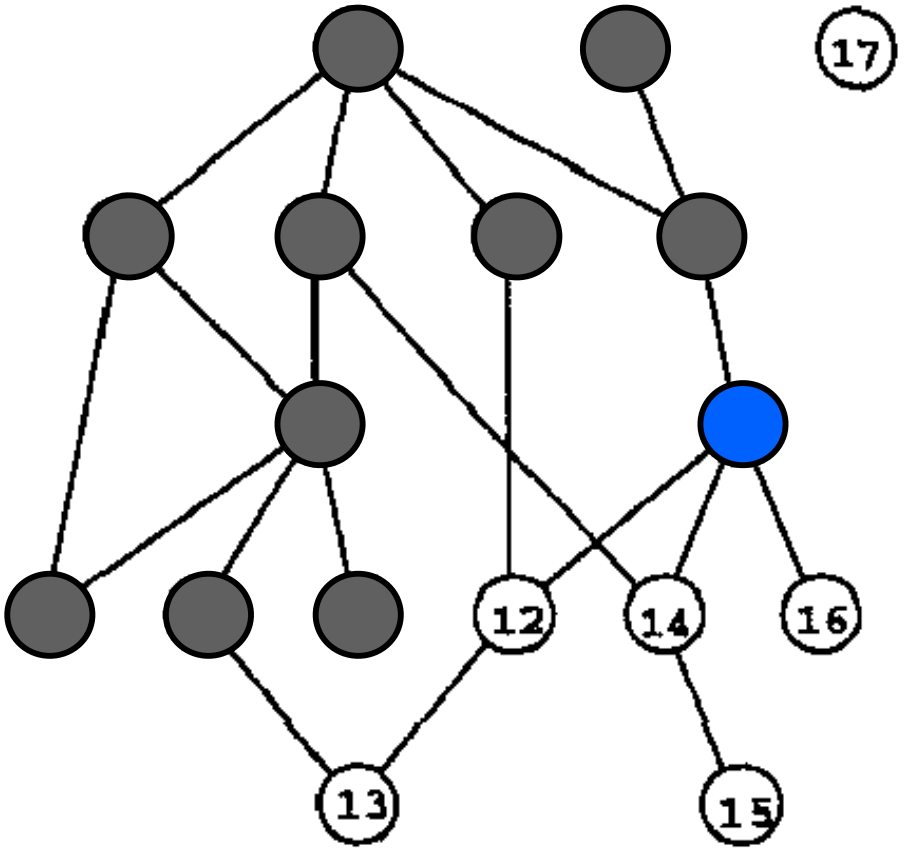


PDF Schedule for p=3

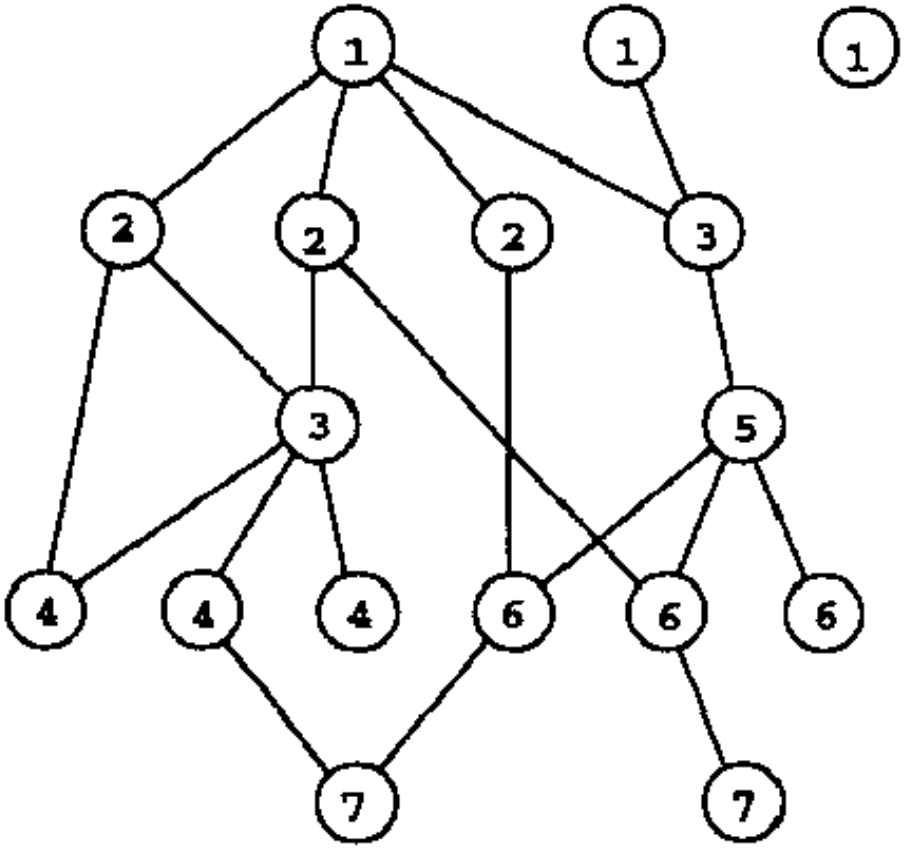


DAG Scheduling

1 DF Schedule

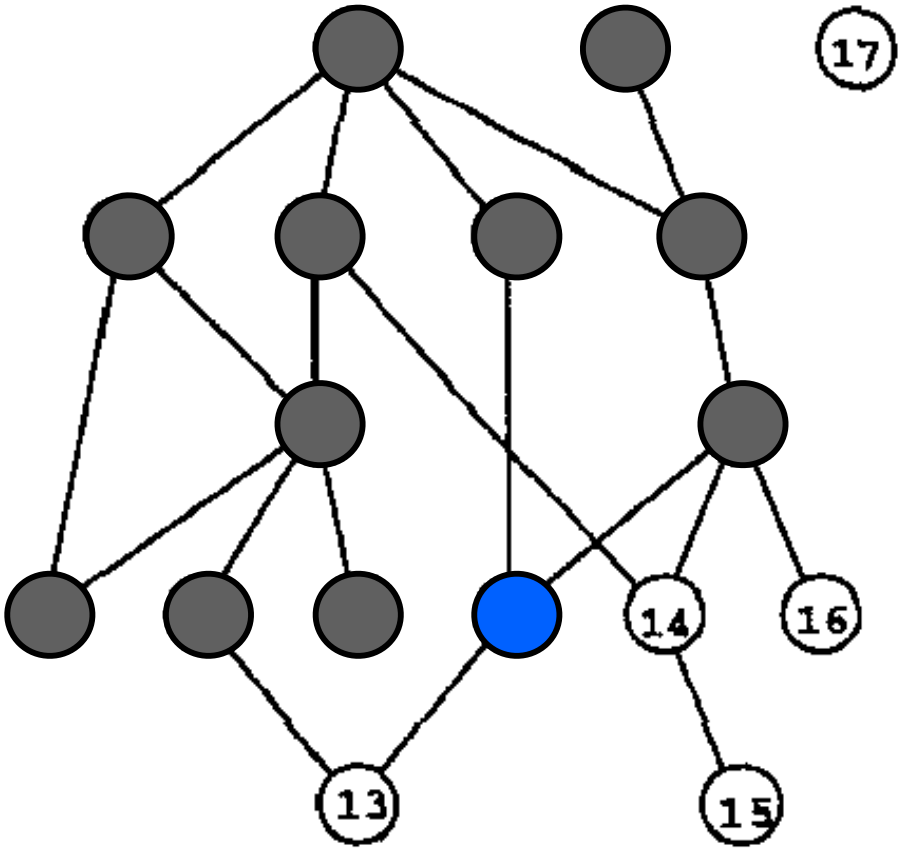


PDF Schedule for p=3

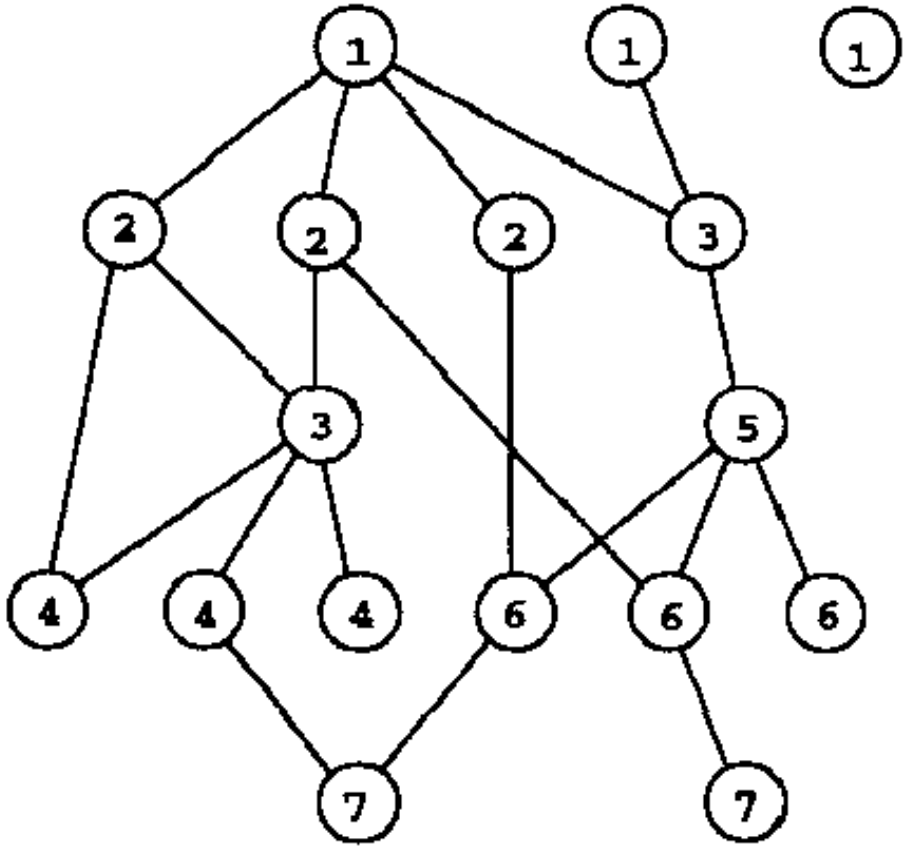


DAG Scheduling

1 DF Schedule

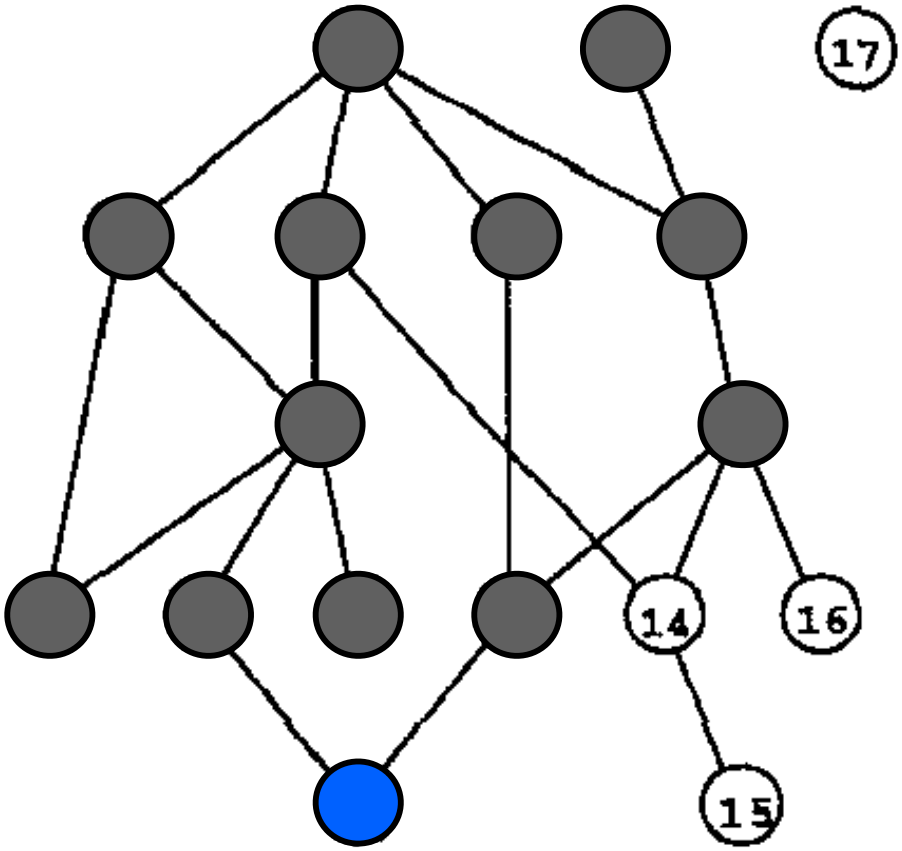


PDF Schedule for p=3

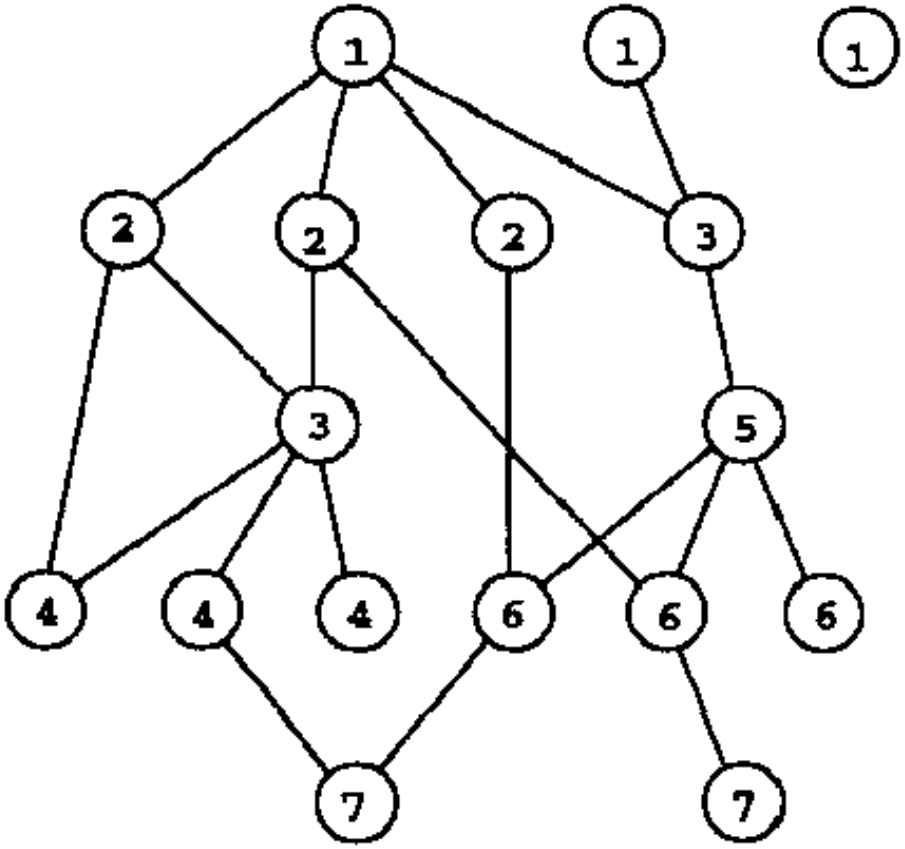


DAG Scheduling

1 DF Schedule

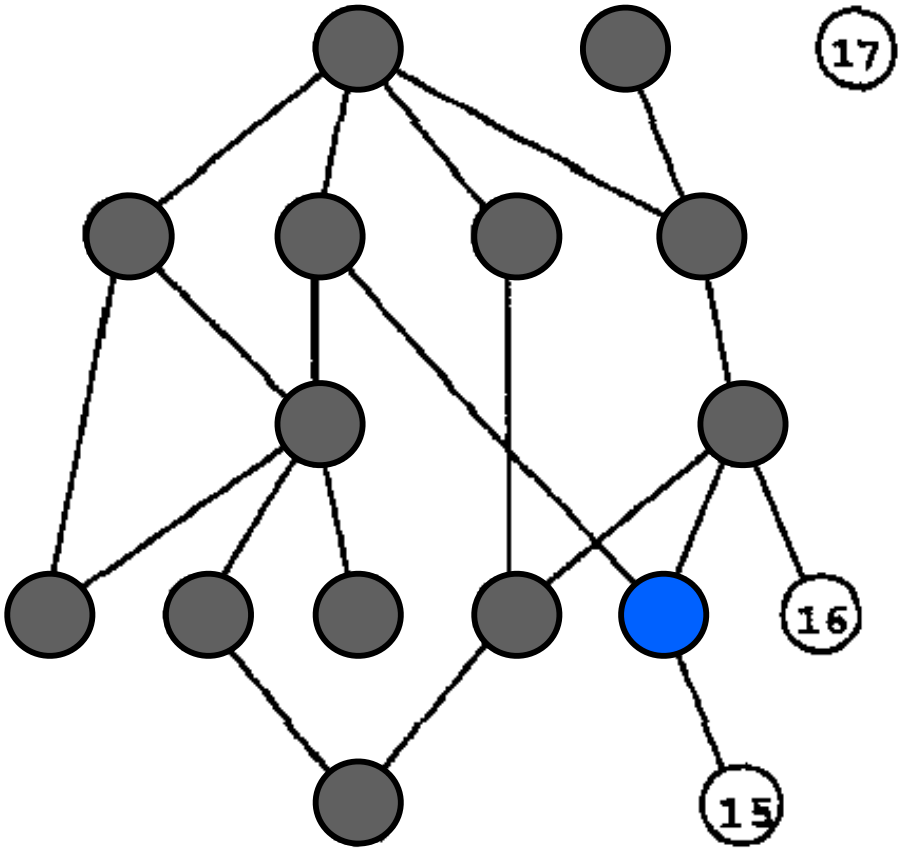


PDF Schedule for p=3

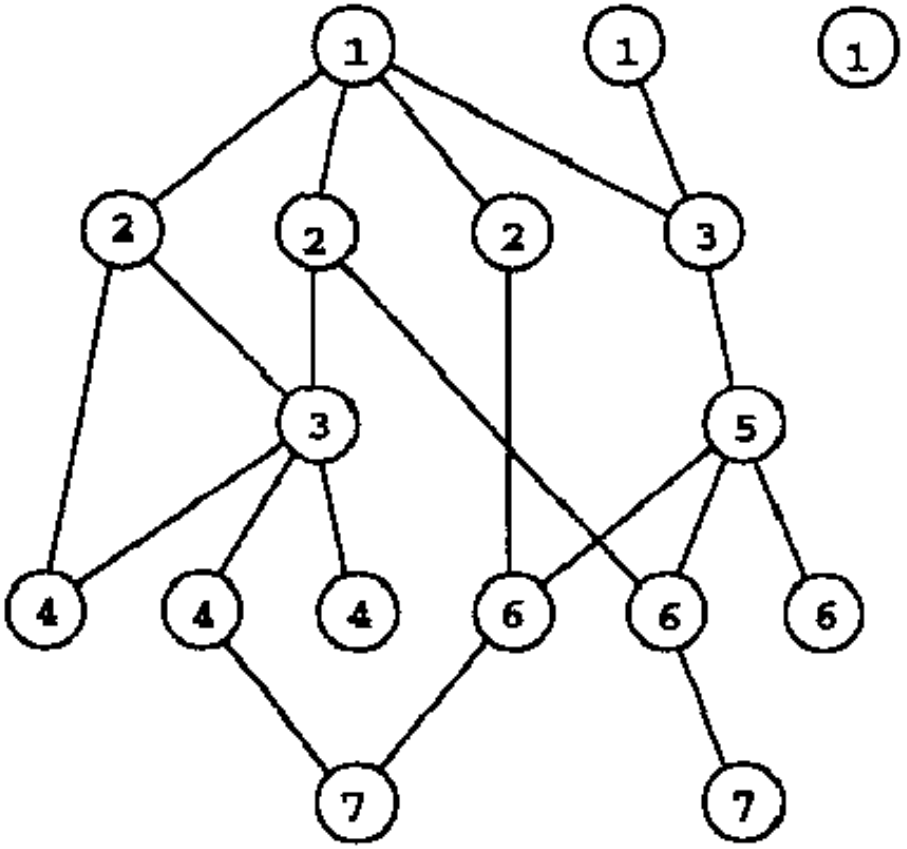


DAG Scheduling

1 DF Schedule

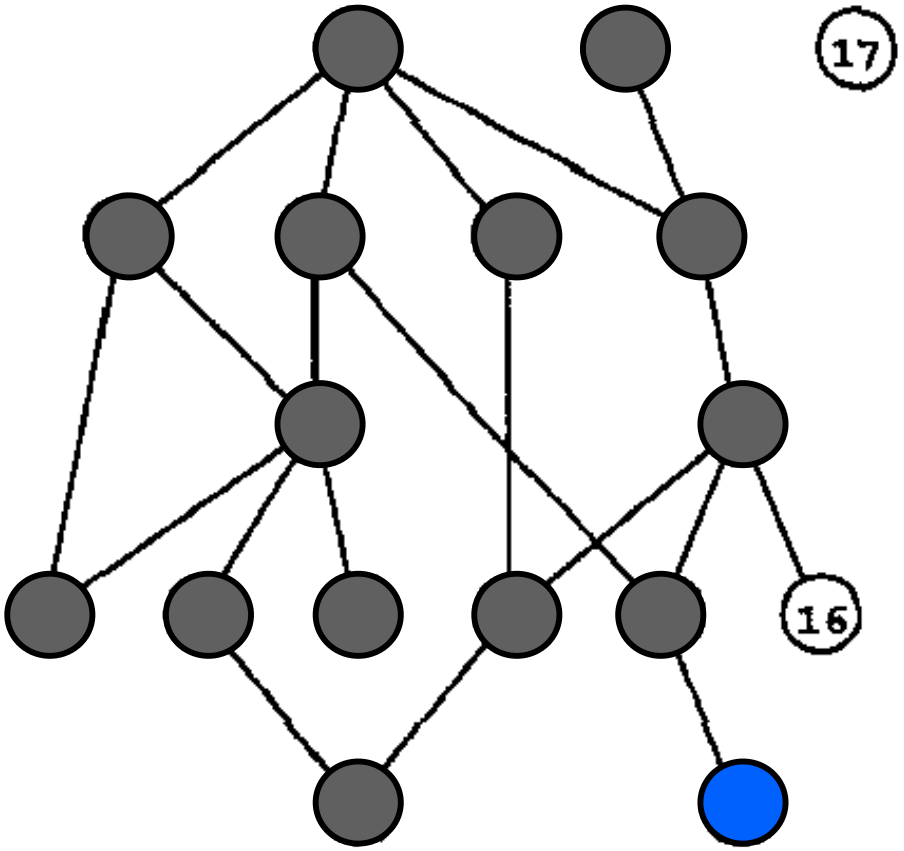


PDF Schedule for p=3

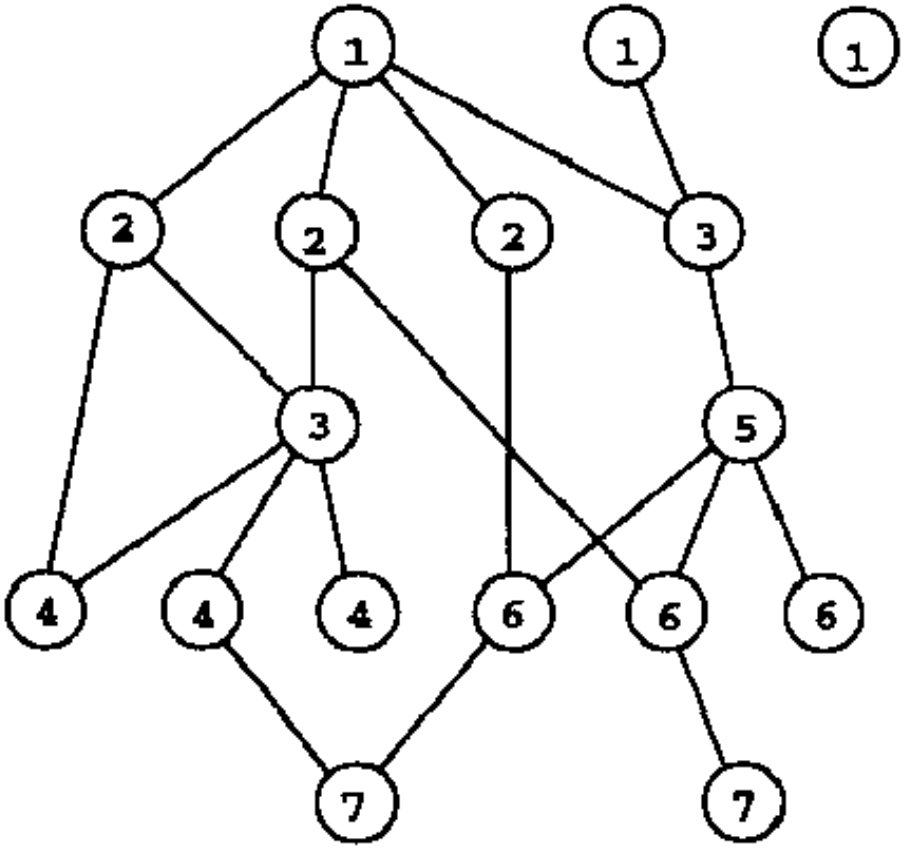


DAG Scheduling

1 DF Schedule

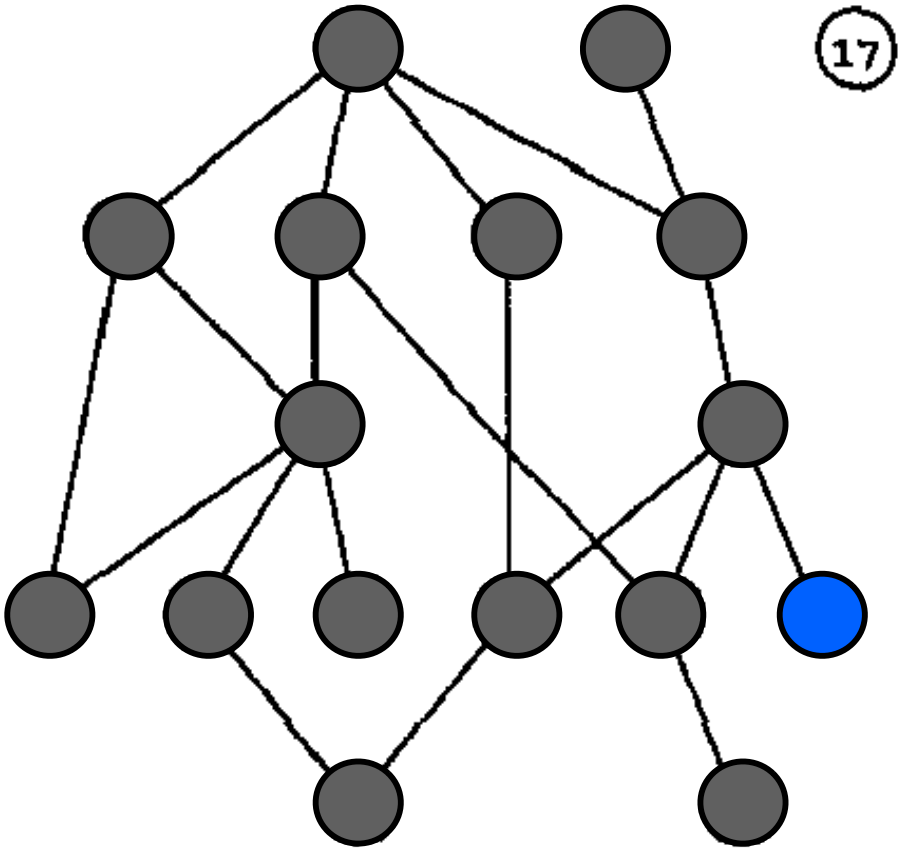


PDF Schedule for p=3

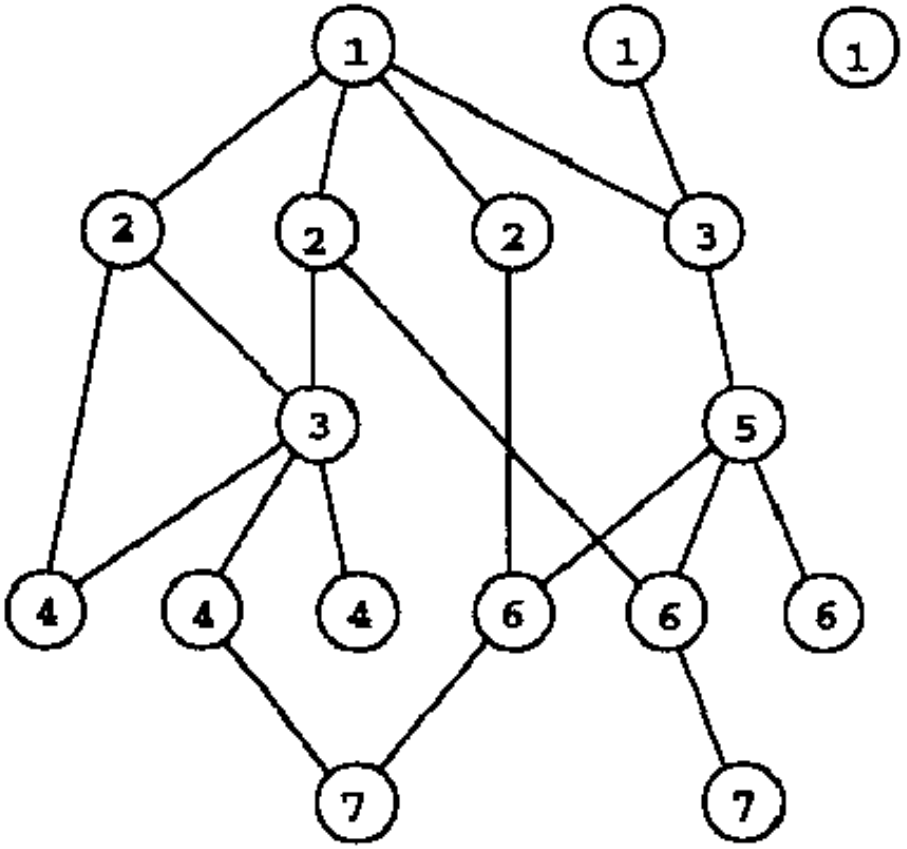


DAG Scheduling

1 DF Schedule

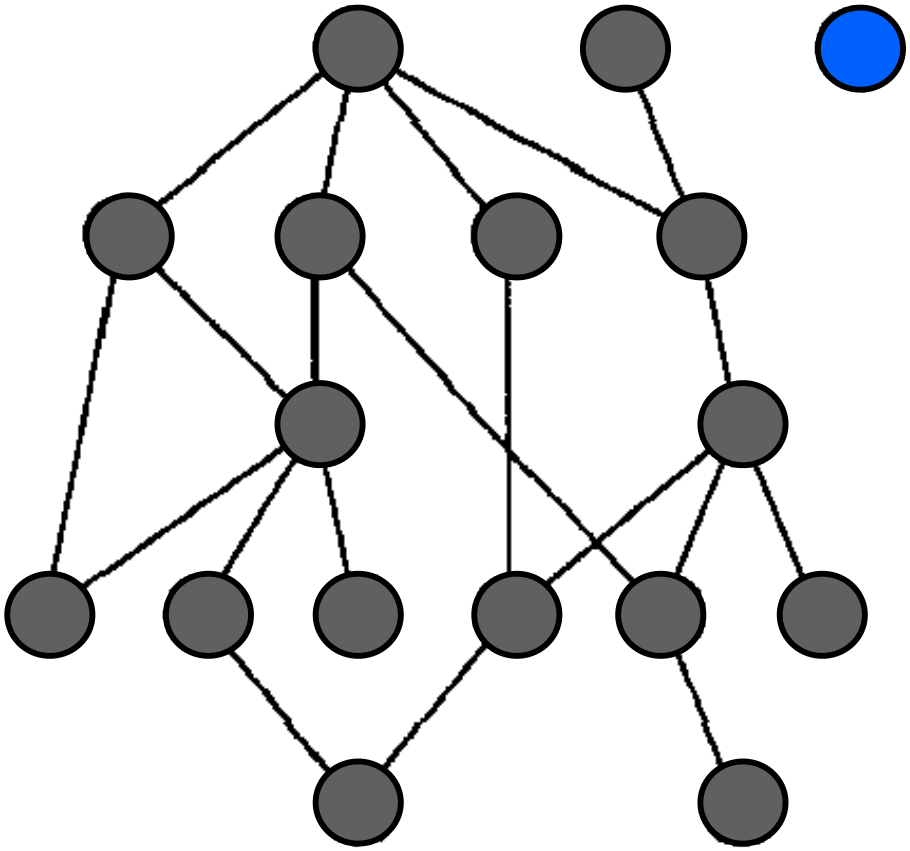


PDF Schedule for p=3

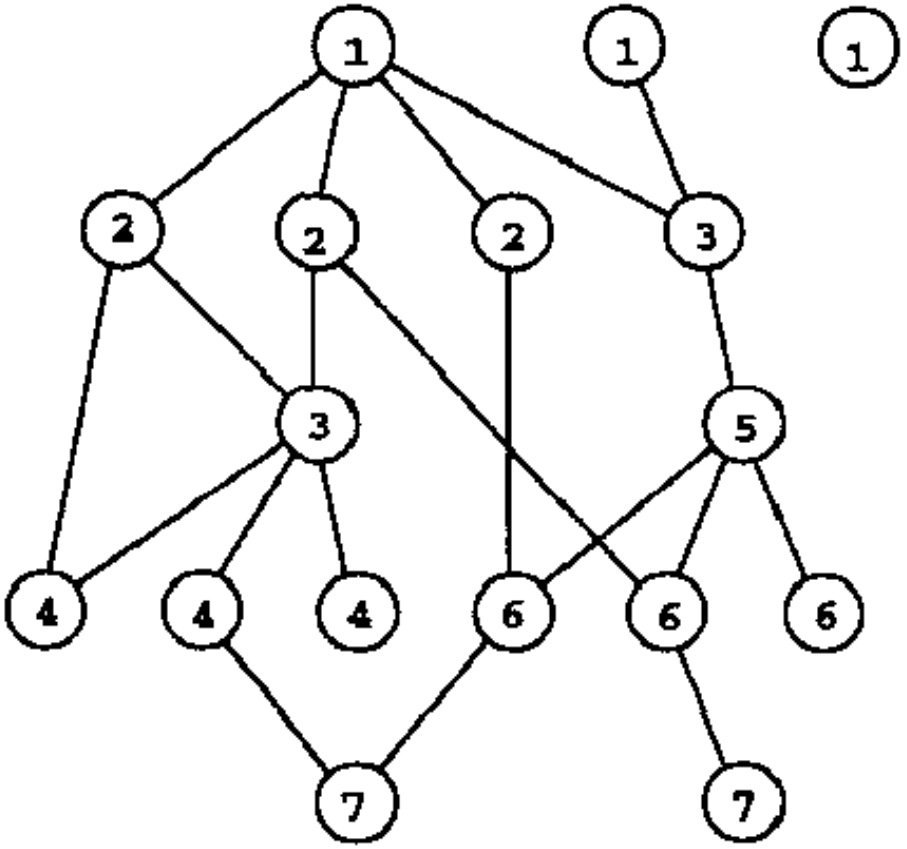


DAG Scheduling

1 DF Schedule



PDF Schedule for p=3

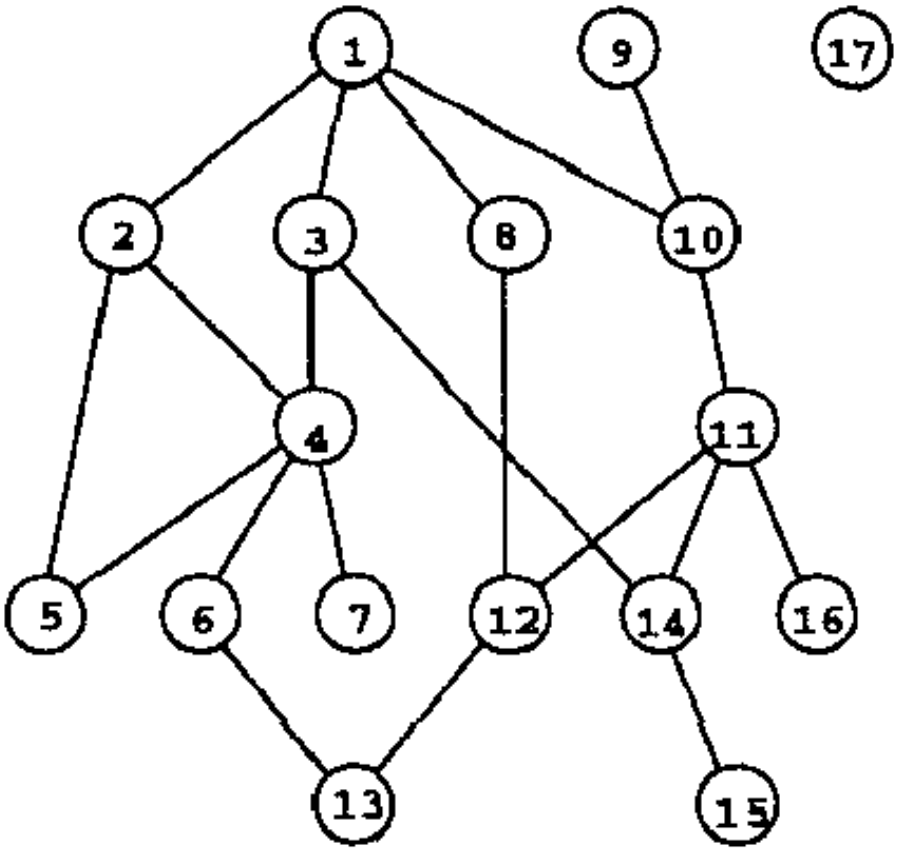


PDF Schedule

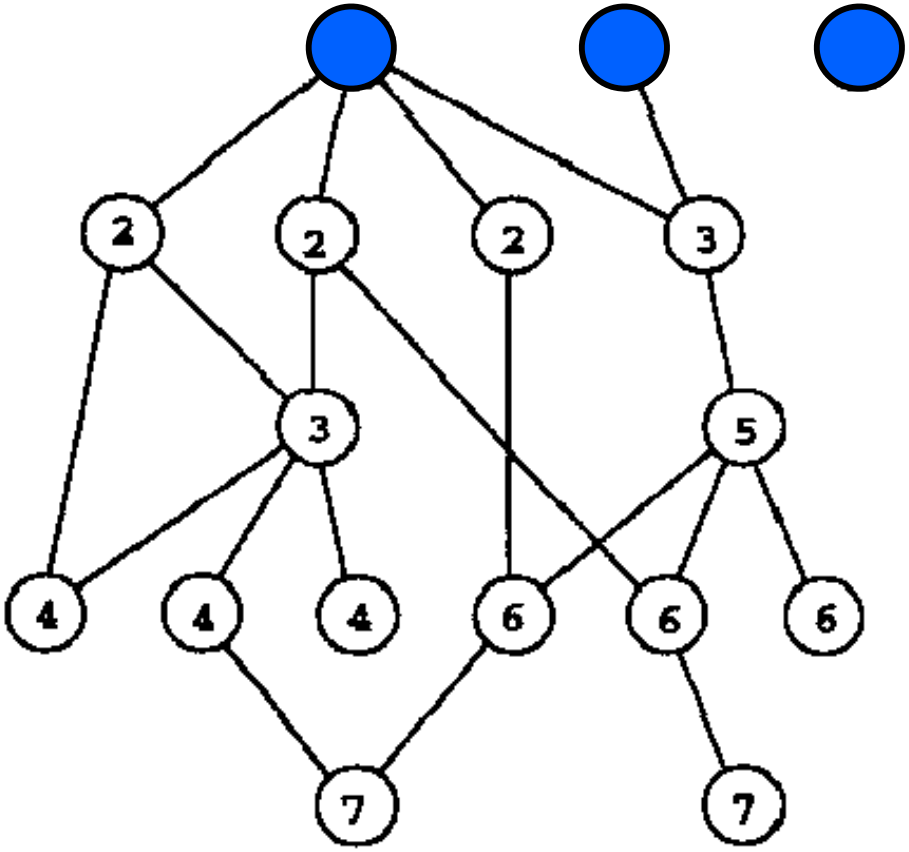
- Let R be a vector containing only the root node
- Schedule the first min(P, |R|) nodes from R with the i^{th} node in R assigned to processor i
- Replace each newly scheduled node in by its ready children, in left to right order, in place in R

DAG Scheduling

1 DF Schedule

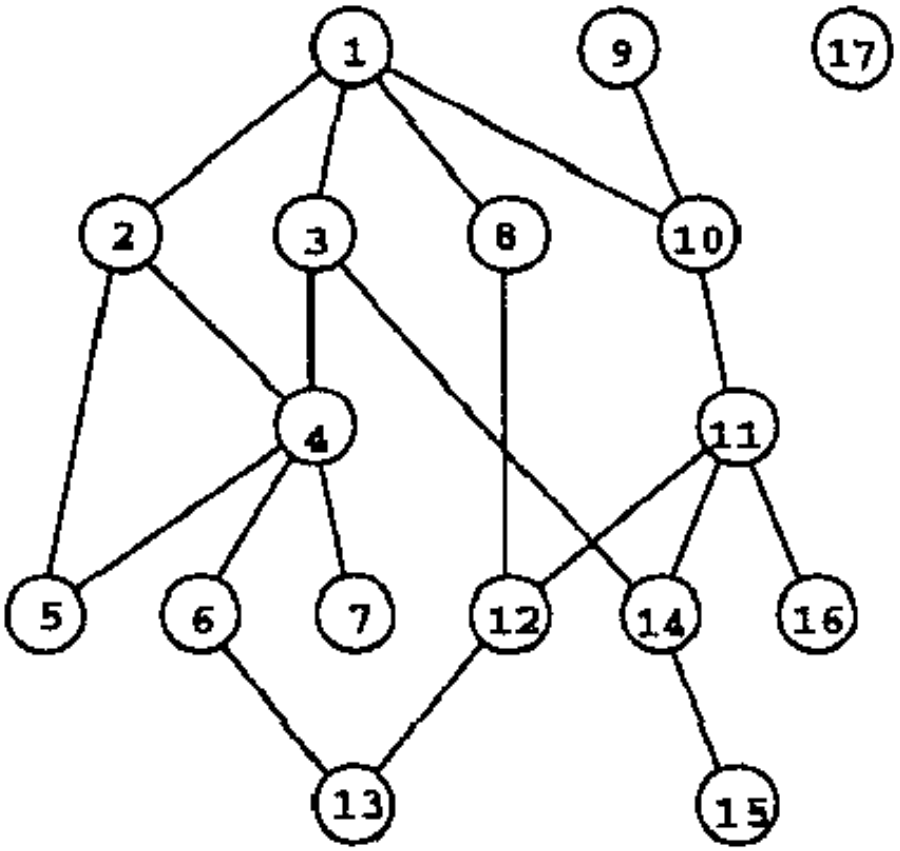


PDF Schedule for p=3

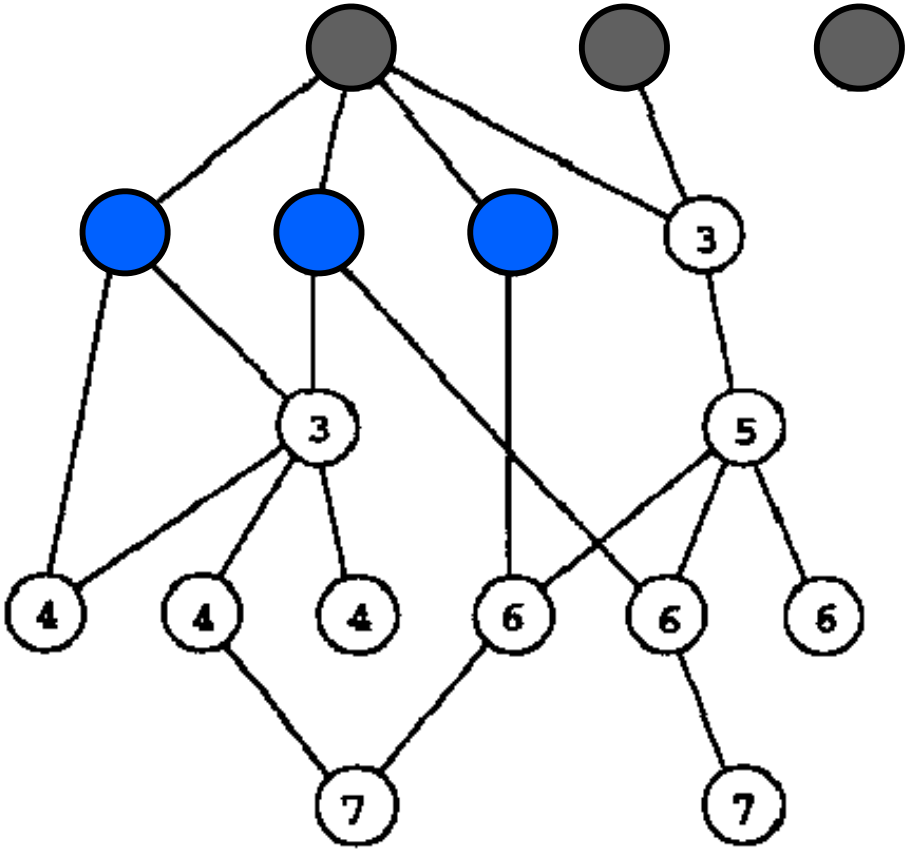


DAG Scheduling

1 DF Schedule

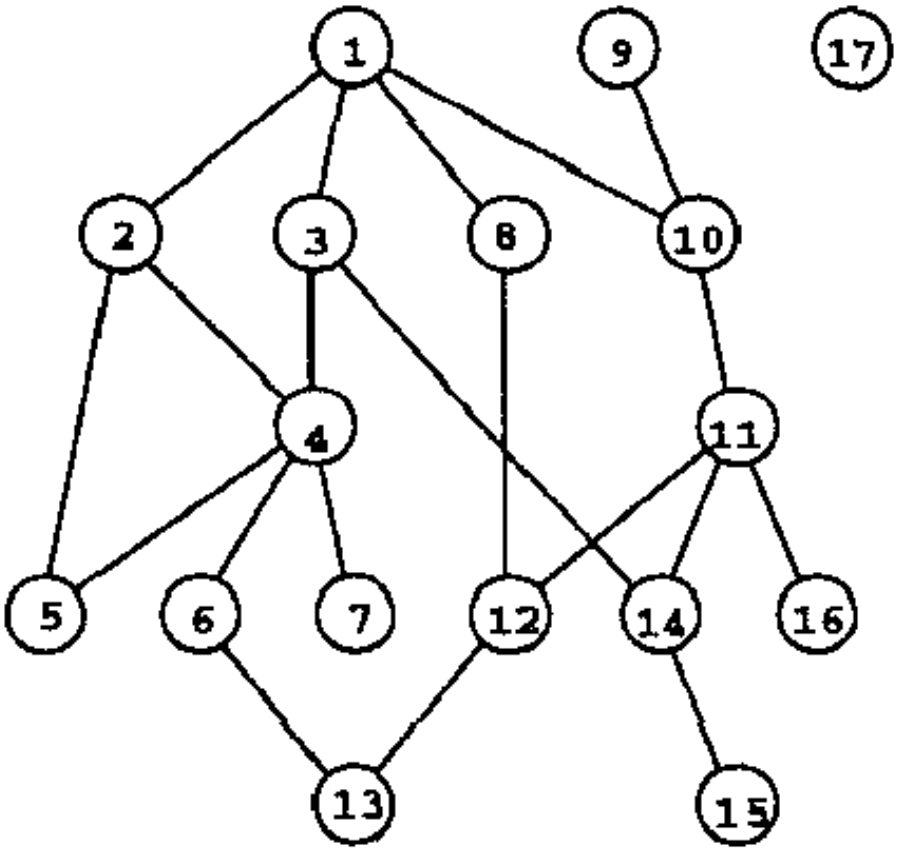


PDF Schedule for p=3

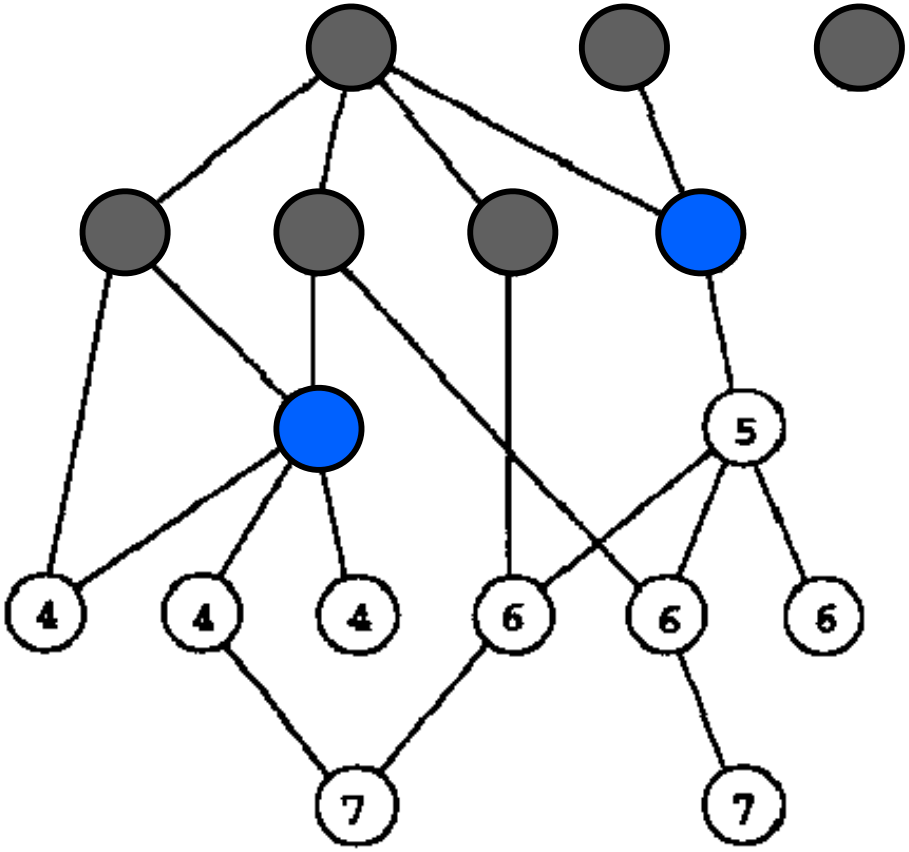


DAG Scheduling

1 DF Schedule

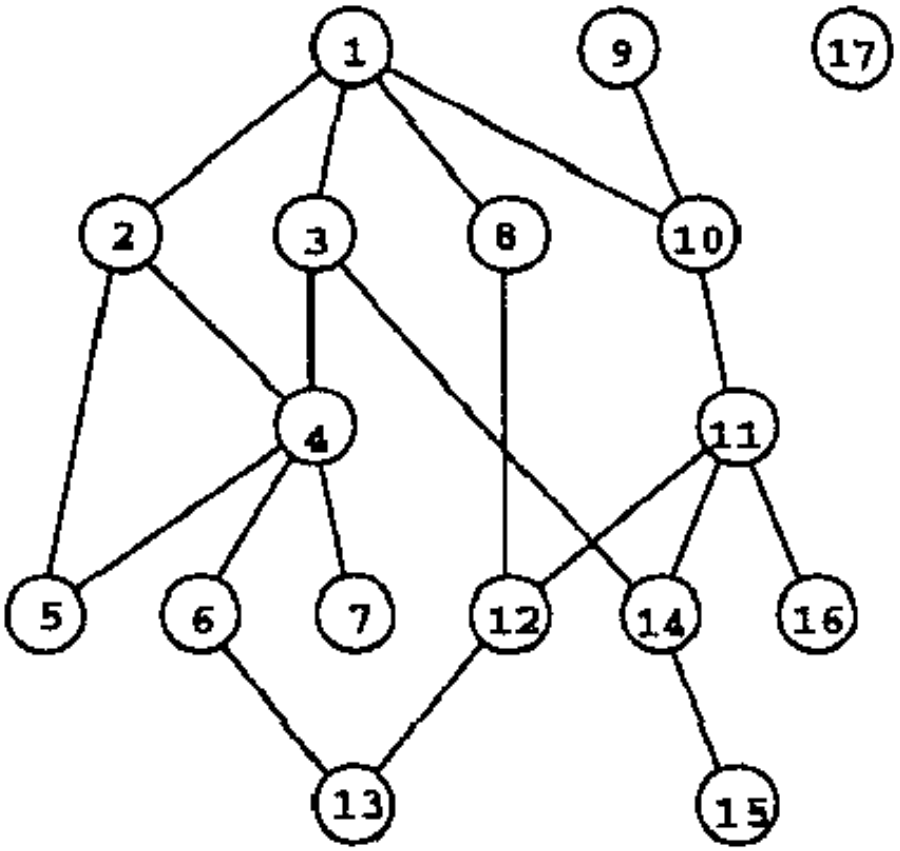


PDF Schedule for p=3

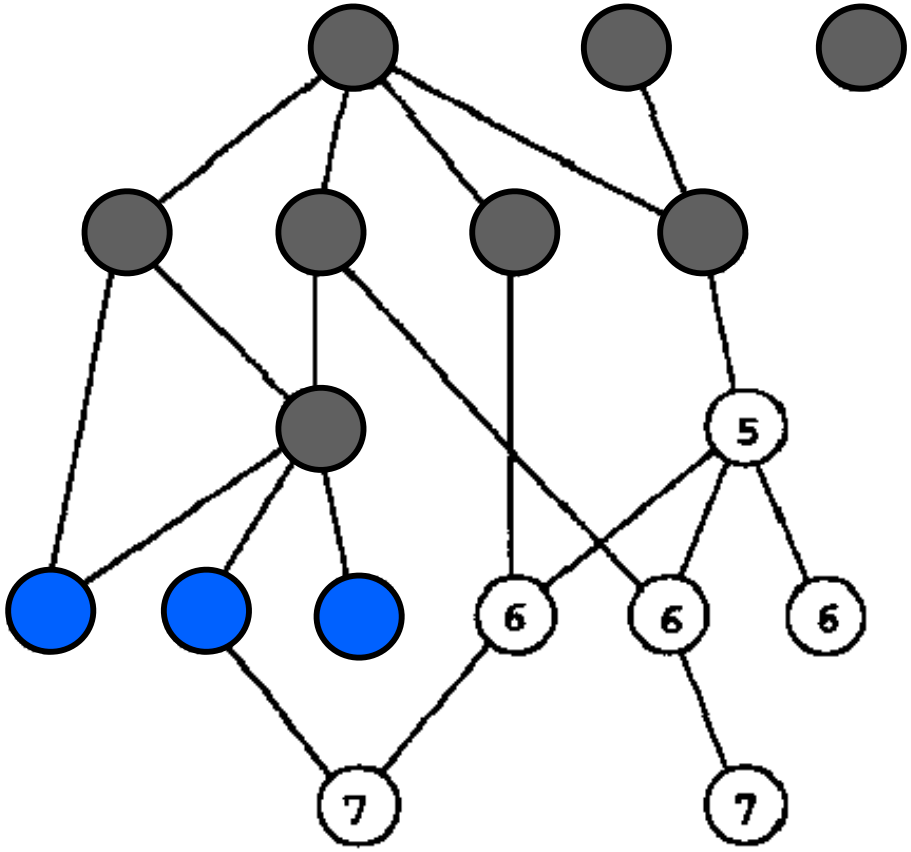


DAG Scheduling

1 DF Schedule

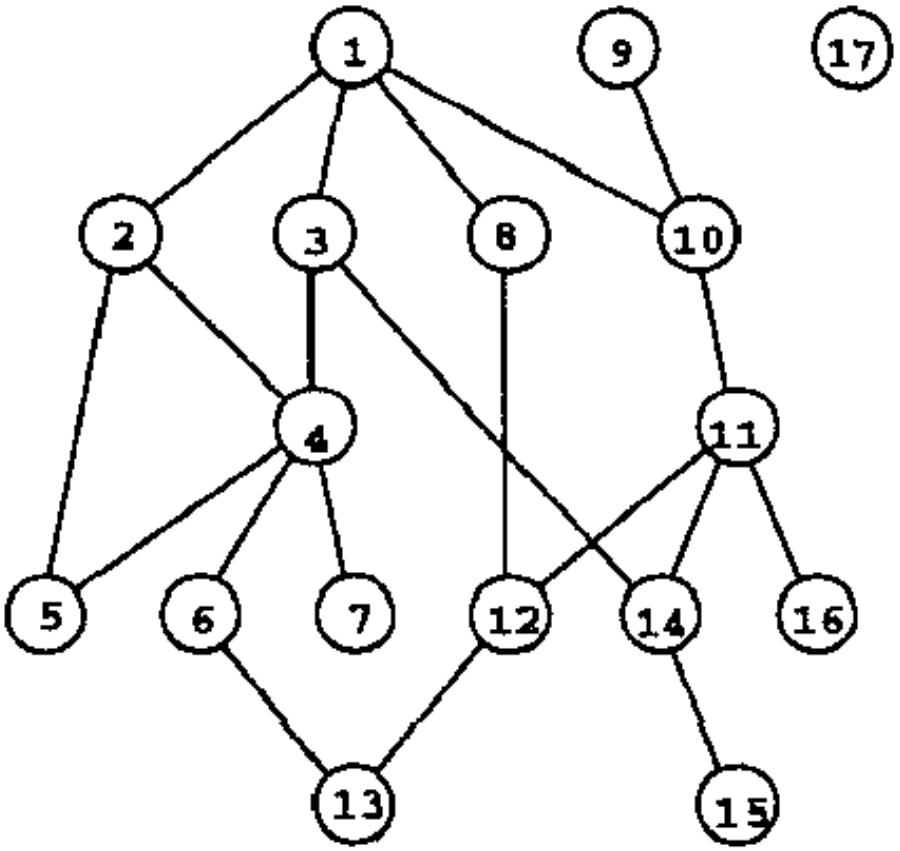


PDF Schedule for p=3

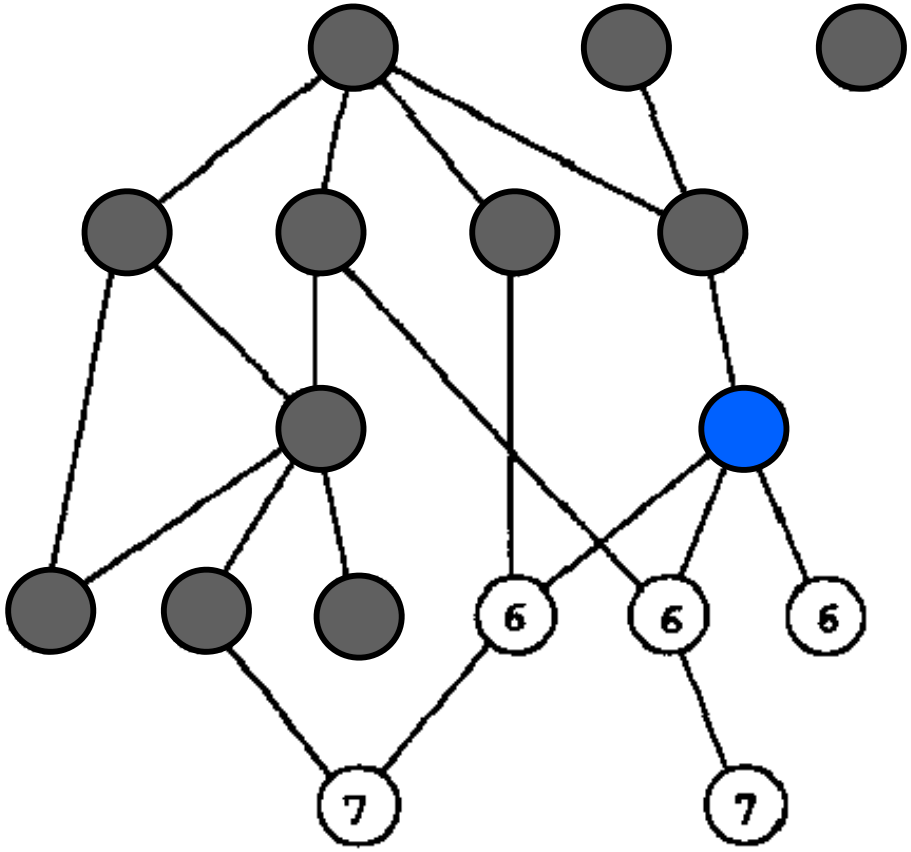


DAG Scheduling

1 DF Schedule

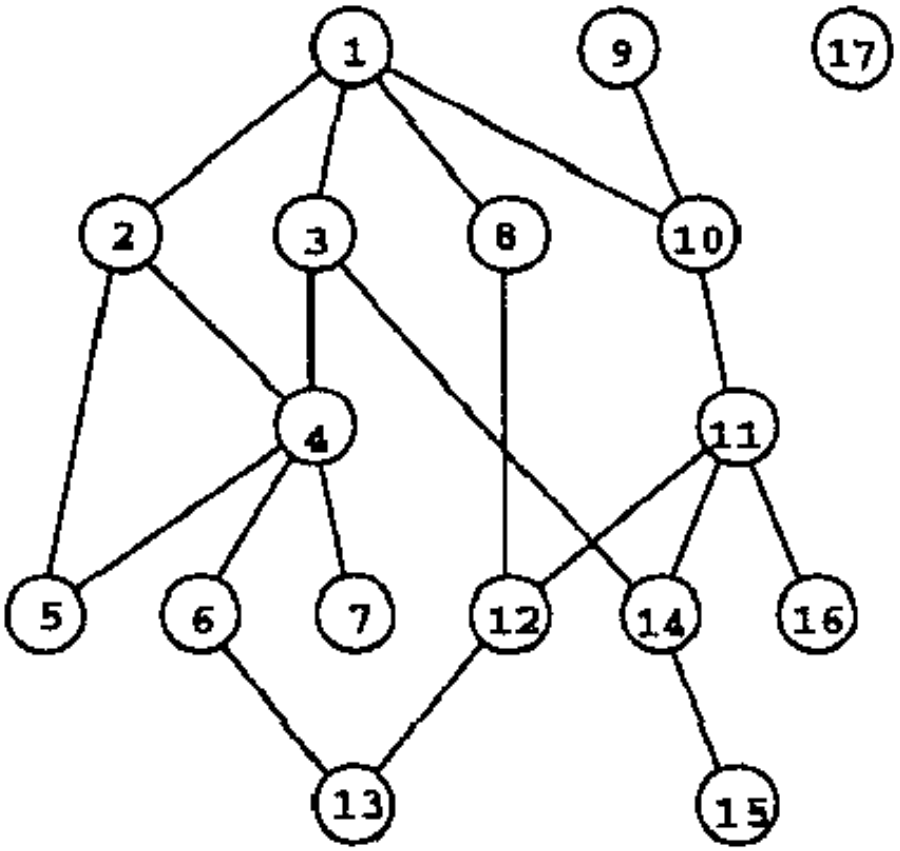


PDF Schedule for p=3

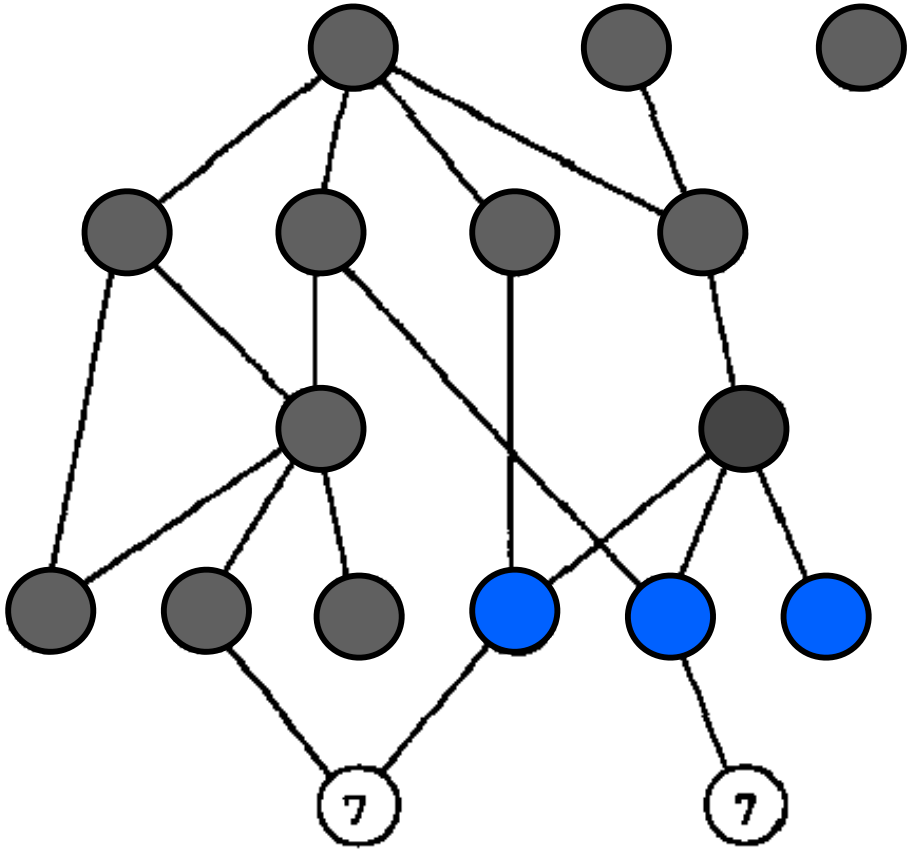


DAG Scheduling

1 DF Schedule

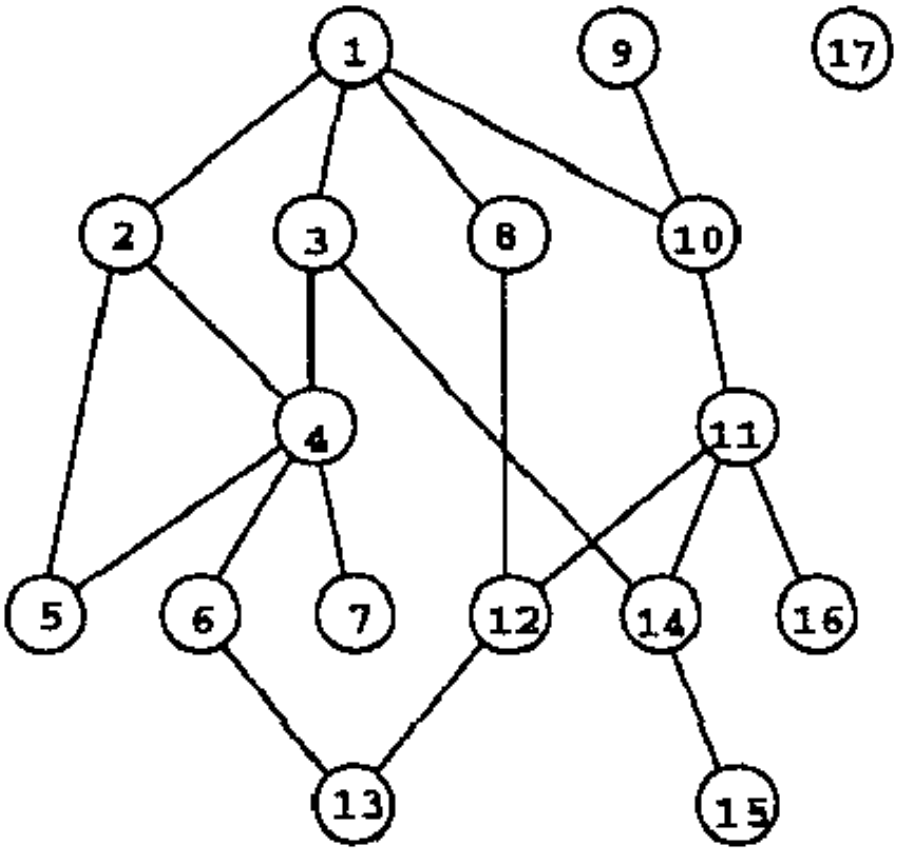


PDF Schedule for p=3

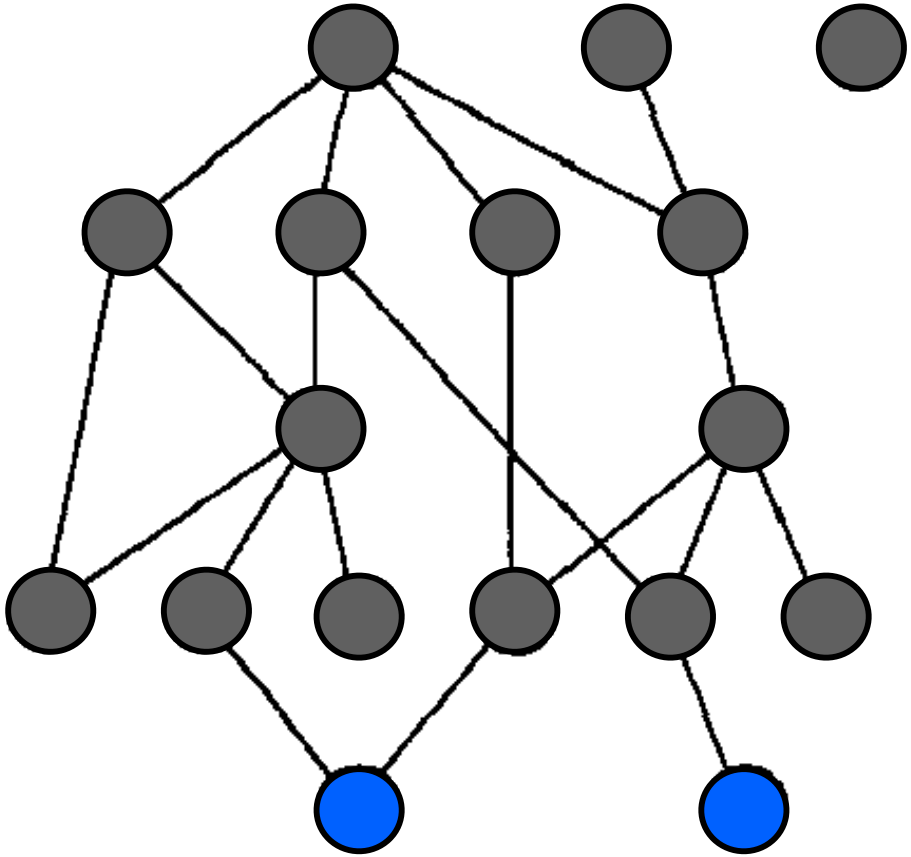


DAG Scheduling

1 DF Schedule



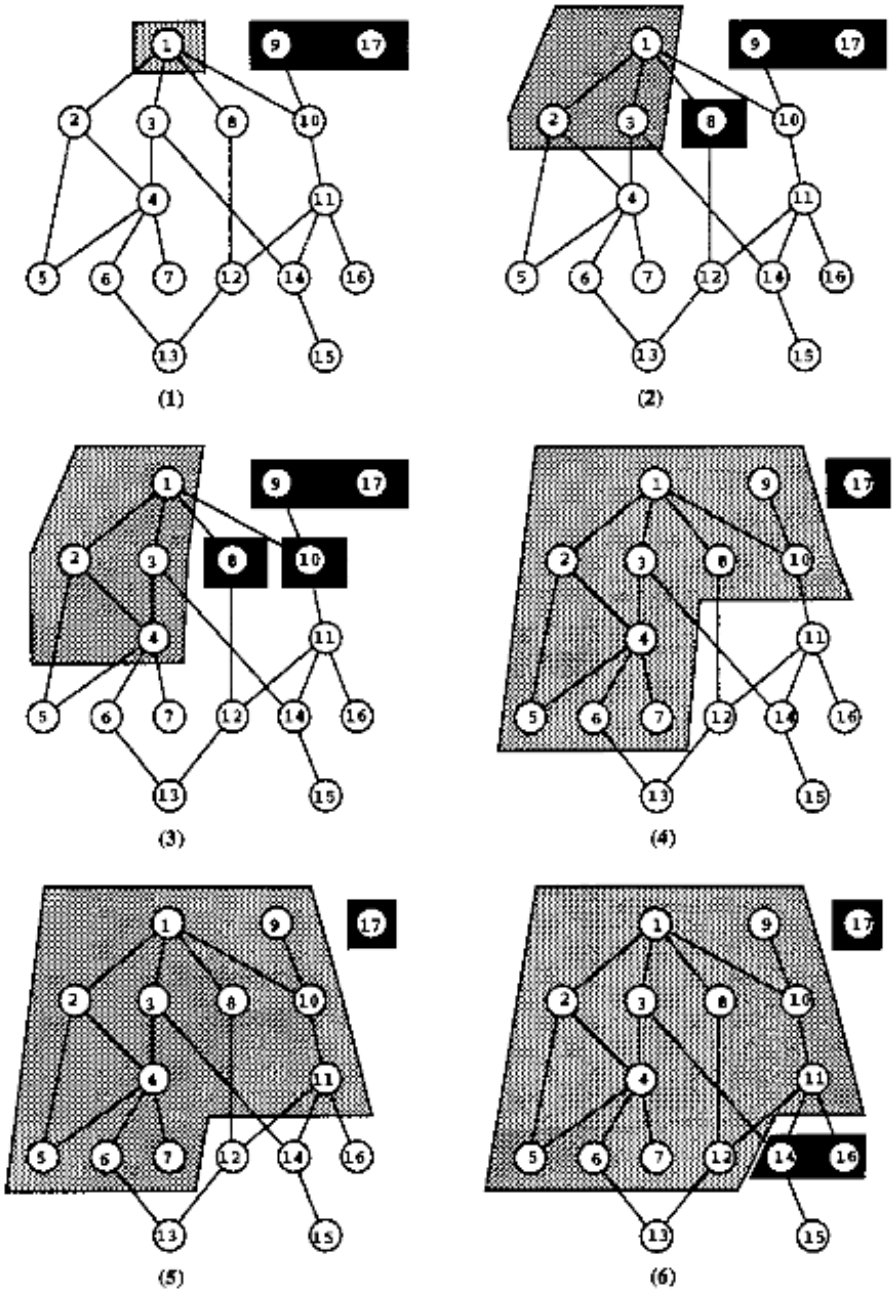
PDF Schedule for p=3



7 steps

PDF Schedule Properties

- Each time step completes work on the critical path
- Only (P - 1) processors available for “other work”
- At most (P - 1) * d nodes are “premature”
  - —ahead of 1DF order
- Premature node bound leads to cache miss bound



PDF Schedules vs. Work Stealing

- PDF schedule uses only $S_1 + O(DP)$ space
  - —S_1 is the space required to execute the computation serially
  - —D is depth of the computation
- Bound is asymptotically lower than the previous bound of S_1P for work stealing when $D = o(S_1)$, which is true for parallel computations that have a sufficient degree of parallelism
  - —Example: a simple algorithm to multiply two n x n matrices has depth $D = \Theta(\log n)$ and serial space $S_1 = \Theta(n^2)$, giving space bounds of $O(n^2 + P \log n)$ instead of $O(n^2 P)$
- How?
  - —have parallel computation follow an order as close as possible to the serial execution

Coordinating a PDF Schedule

- **Uses constant # of prefix-sums per scheduling round**
- **Time complexity $O(T_1/P + d * \lg P)$**

- **Reducing scheduling overhead**
  - **—maintaining R can exceed space bound**
    - **maintain parents instead of kids ... see paper for details**
  - **—reducing time overhead**
    - **approach**
      - **schedule multiple tasks per processor per round**
        - **use a parallel DFT with wider width than P**
    - **impact**
      - **increases additive term in time and space complexities**
      - **ensures work is within constant factor of optimal**

Ideal Cache Miss Bounds

- Ideal cache = fully associative, optimal replacement policy
- For ideal cache, if $C_p \geq (C_1 + (P - 1) * D)$, then $M_p \leq M_1$
  - —size of shared cache size for P threads only needs to be P*d larger than the cache for a single thread for same miss rate
- Intuition
  - —at most (P - 1) * d nodes can be executed prematurely
  - —cache is (P - 1) * d blocks larger
    - premature nodes have space for them in the cache
  - —cache is ideal, won't cause extra evictions until cache is full
  - —there is a 1-1 correspondence between bad and good nodes
    - bad node incurs a hit in 1DF schedule but a miss in PDF schedule
    - good node incurs a miss in 1DF schedule but a hit in PDF schedule

LRU Cache Miss Bounds

- Tarjan and Sleator show that an ideal cache of size C_1 can be simulated with an LRU cache of size $2 * C_1$
- Use this result to extend previous bound to LRU caches
  - —for an LRU cache, if $C_p \geq 2 * (C_1 + (P - 1) * d)$, then $M_p \leq M_1$

Value of PDF Schedules

- Shows that PDF schedules can manage shared cache effectively for multiple cores
  - —provides upper and lower bounds on cache misses
- For computations based on nested data parallelism, the size of memory and cache need not grow linearly with number of cores and threads in manycore designs

Space-efficient Implementation of Nested Parallelism

- **Problem: PDF scheduler tightly controls space, but has too many expensive scheduling steps**
- **Goal: allow threads to execute non-preemptively and asynchronously to improve locality and reduce scheduling overhead**
- **Approach: allocate a pool of a constant K units of memory to each thread when it starts up, and allow a thread to execute non-preemptively until it runs out of memory from that pool (and reaches an instruction that requires more memory), or until it suspends**
  - **—instead of preallocating a pool of K units of memory for each thread, assign it a counter that keeps track of its net memory allocation. We call this runtime, user-defined constant K the memory threshold for the scheduler**

AsyncDF Scheduler Sketch

- When the scheduler forks (creates) child threads, insert them into R (priority Q of ready threads) immediately to the left of their parent thread
  - —maintain the invariant that the threads in R are always in the order of increasing 1DF-numbers of their leading nodes
- At every scheduling step, P ready threads whose leading nodes have the smallest 1DF-numbers are moved to Q_{out} (FIFO of ready threads removed from R)
  - —every time a ready thread is picked from Q_{out} and scheduled on a worker processor, it may allocate space from a global pool in its first action
  - —a thread must preempt itself before any subsequent action that requires more space
- Child threads are forked only when they are to be added to Q_{out}, that is, when they are among the leftmost P ready threads in R

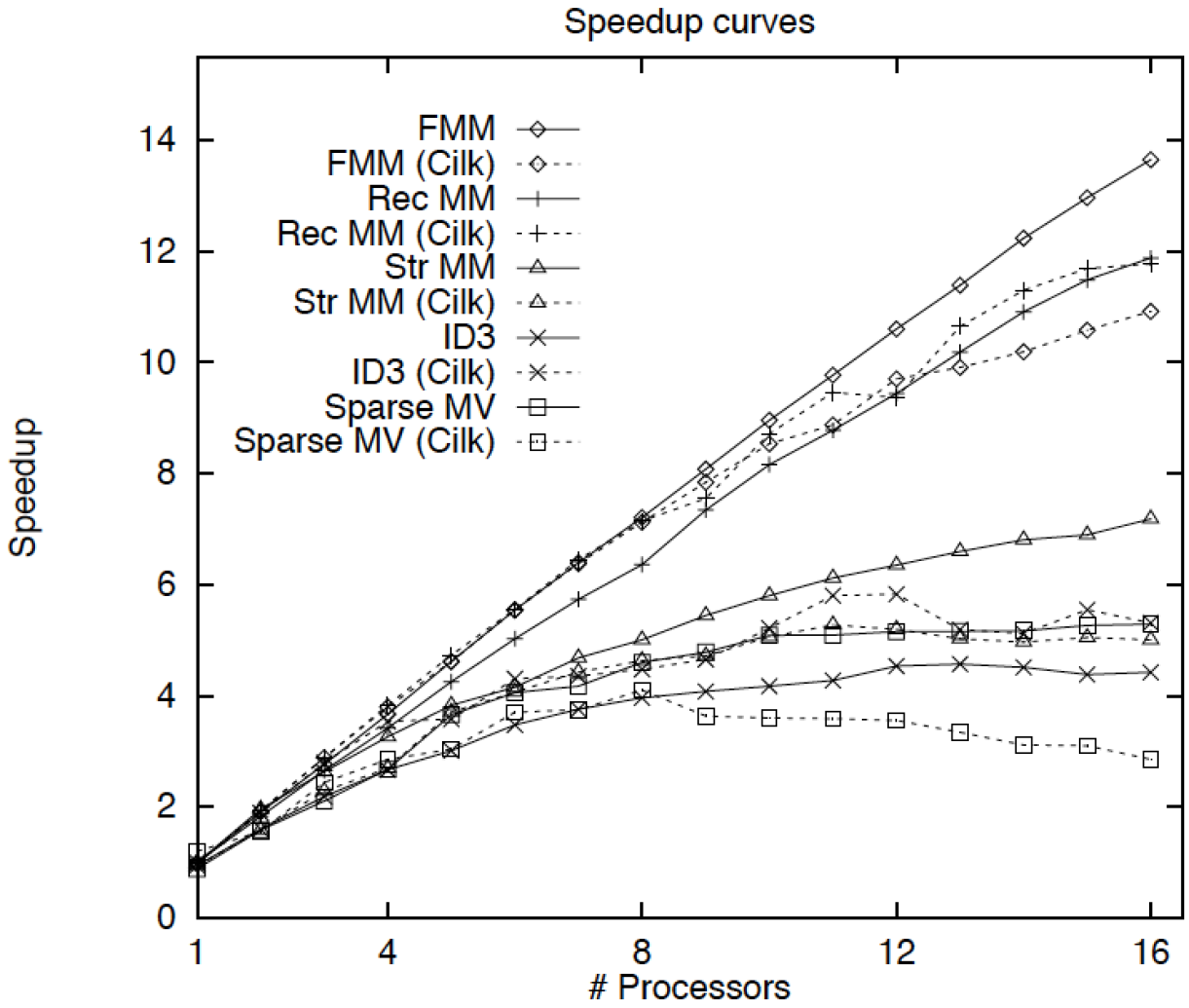
AsyncDF Scheduler Result

Theorem

Let S_1 be the space required by a 1DF-schedule for a computation with work W and depth D, and let S_a be the total space allocated in the computation.

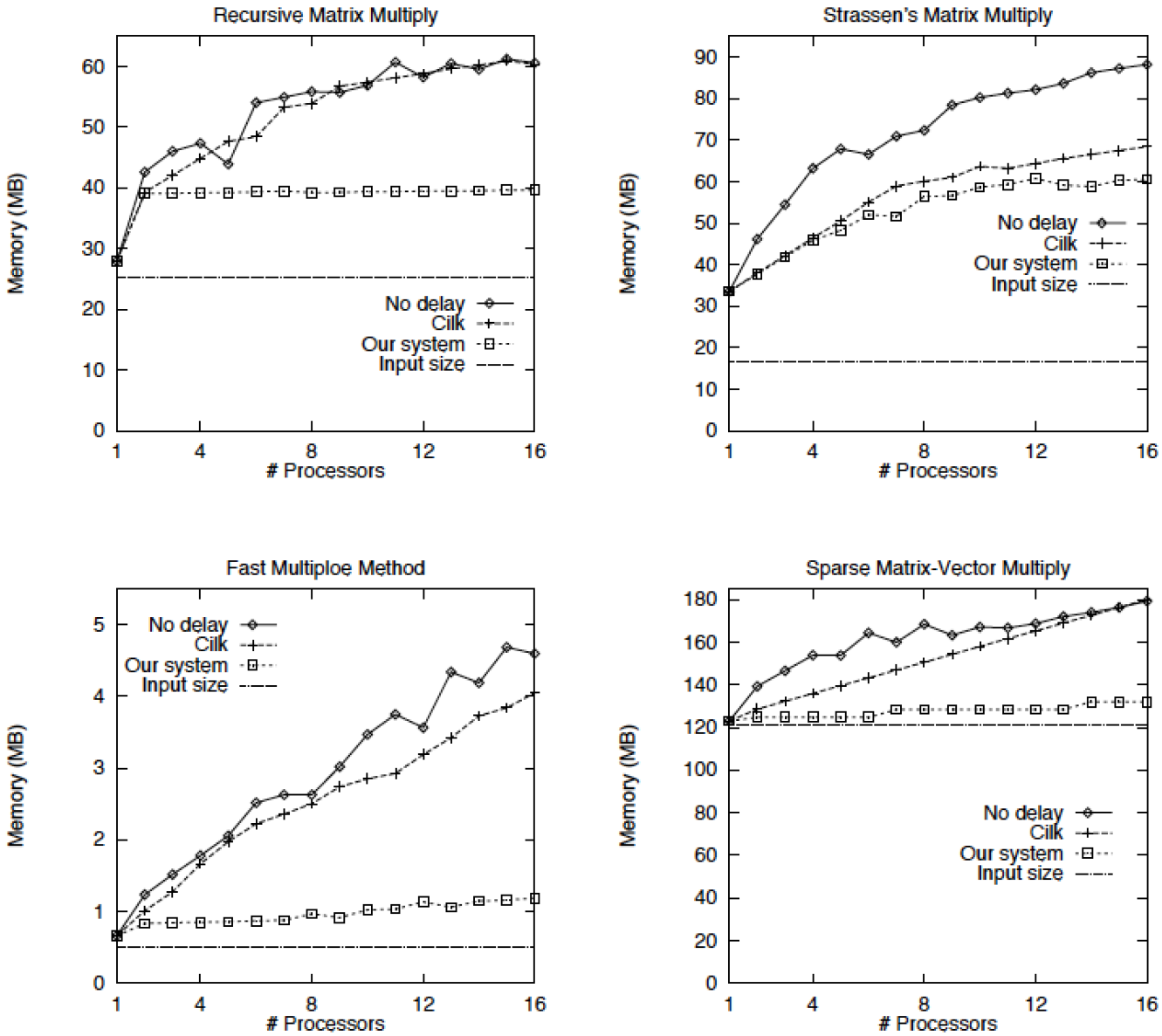
The parallelized scheduler with a memory threshold of K units, generates a schedule on P processors that requires $S_1 + O(KDP \log P)$ space and $O(W/P + S_a/PK + D \log P)$ time to execute

AsyncDF vs. Work Stealing: Speedup



Girija J. Narlikar and Guy E. Blelloch. 1999. Space-efficient scheduling of nested parallelism. *ACM TOPLAS.* 21, 1 (January 1999), 138-173

AsyncDF vs. Work Stealing: Space



Girija J. Narlikar and Guy E. Blelloch. 1999. Space-efficient scheduling of nested parallelism. *ACM TOPLAS.* 21, 1 (January 1999), 138-173

Summary

- PDF schedules have the same number of asymptotic time steps as work stealing schedulers, though there they incur log P scheduling overhead at each step
- PDF schedules use asymptotically smaller space than work stealing
- PDF scheduling overhead can be reduced by letting processors operate asynchronously within a bounded amount of memory
- PDF space bounds can be maintained by having asynchronous processors coordinate when they hit a local space bound to ensure that the collection of processors achieve the desired global asymptotic bound

Differences

- Data storage
  - —Blumofe and Leiserson allows for only stack allocation
  - —PDF scheduling results allow for arbitrary heap allocation/ deallocation, both in the case that it is explicitly handled by the programmer and when it is implicitly handled by a garbage collector.
- Scheduling
  - —Blumofe and Leiserson use distributed “work stealing”. As long as each processor has tasks to execute, no overheads are incurred.
  - —PDF scheduling is a centralized, parallel approach
- Communication
  - —Blumofe and Leiserson’s work-stealing scheduler is shown to provide good bounds on the total amount of interprocessor communication for the task model they consider.

References

- Provably efficient scheduling for languages with fine-grained parallelism. Guy E. Blelloch, Phillip B. Gibbons, and Yossi Matias, Y. 1995. In Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (Santa Barbara, California, United States, June 24 - 26, 1995). SPAA '95. ACM Press, New York, NY, 1-12.
- Space-efficient implementation of nested parallelism. Girija J. Narlikar and Guy E. Blelloch. In Proceedings of the 6th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPOPP '97). ACM, New York, NY, USA, 25-36.
- Effectively sharing a cache among threads. Guy E. Blelloch and Phillip B. Gibbons. In Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures (SPAA '04). ACM, New York, NY, USA, 235-244.