COMP 322: Fundamentals of Parallel Programming

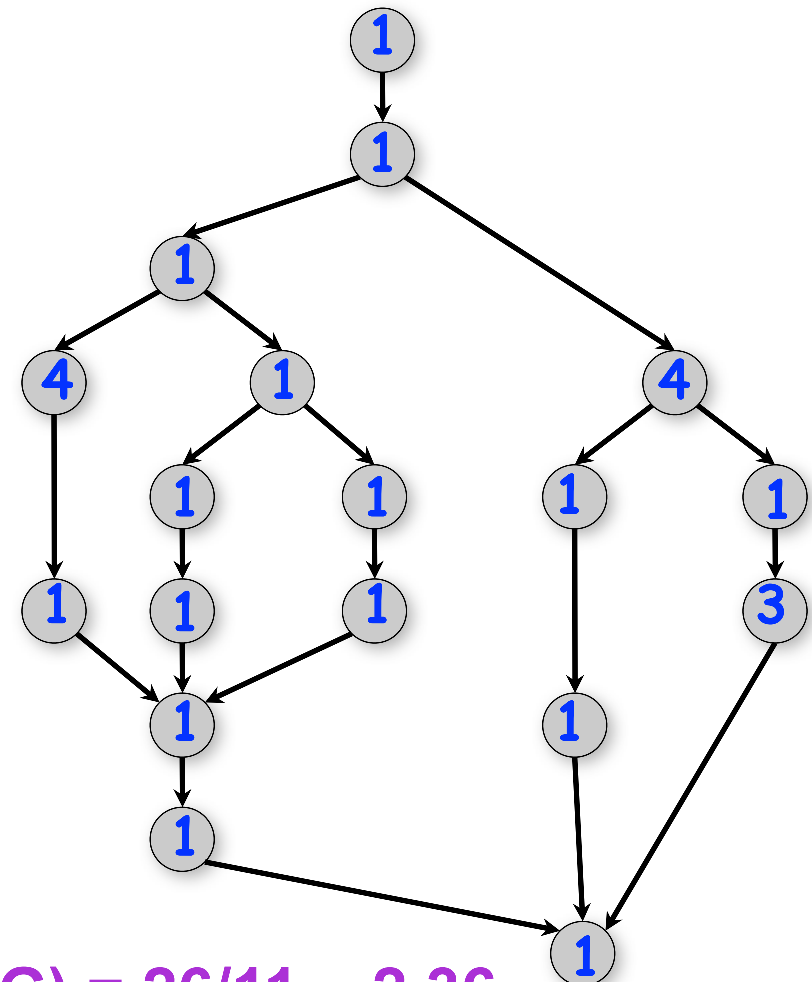
Lecture 3: Multiprocessor Scheduling, Abstract Performance Metrics

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Ideal Parallelism (Recap)

- Define **ideal parallelism** of Computation G Graph as the ratio, WORK(G)/CPL(G)
- Ideal Parallelism is independent of the number of processors that the program executes on, and only depends on the computation graph



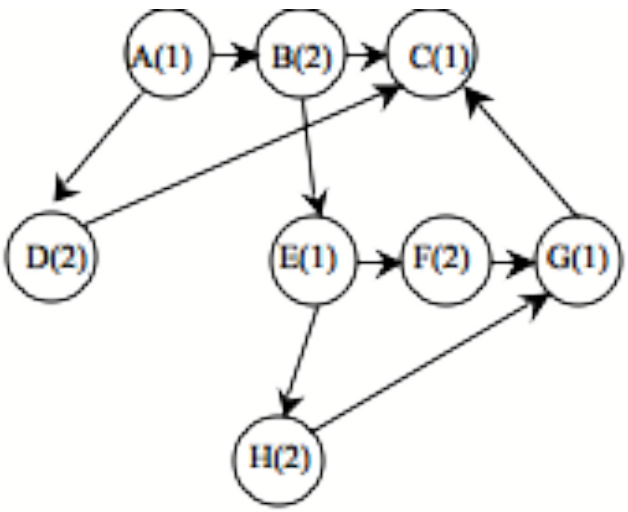
Example:

WORK(G) = 26

CPL(G) = 11

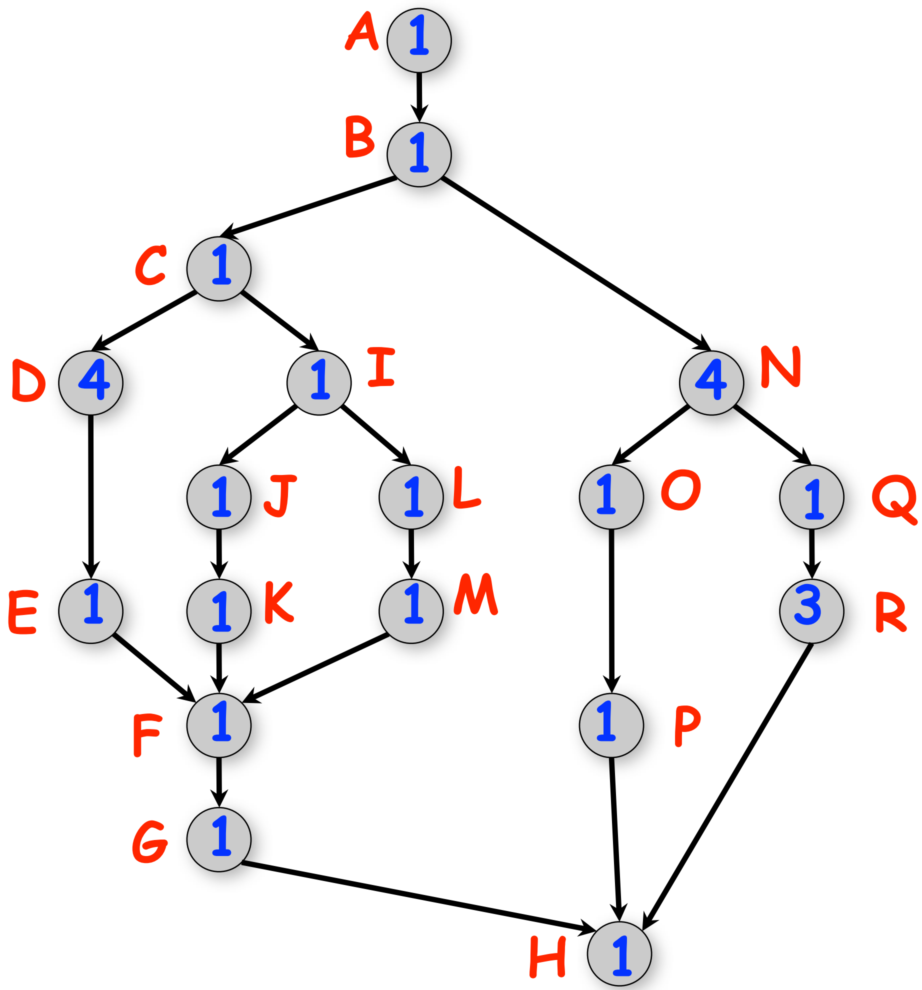
Ideal Parallelism = WORK(G)/CPL(G) = 26/11 ~ 2.36

Solution to Worksheet 2
(Reverse Engineering a Computation Graph)




```
1.A;
2.finish {
3.  async D;
4.  B;
5.  E;
6.  finish {
7.    async H;
8.    F;
9.  } // finish
10. G;
11.} // finish
12.C;
```

Scheduling of a Computation Graph on a fixed number of processors: Example



Start time	Proc 1	Proc 2	Proc 3
0	A		
1	B		
2	C	N	
3	D	N	I
4	D	N	J
5	D	N	K
6	D	Q	L
7	E	R	M
8	F	R	O
9	G	R	P
10	H		
11			

Scheduling of a Computation Graph on a fixed number of processors, P

- Assume that node N takes TIME(N) regardless of which processor it executes on, and that there is no overhead for creating parallel tasks
- A schedule specifies the following for each node
  - START(N) = start time
  - PROC(N) = index of processor in range 1...P

  such that
  - START(i) + TIME(i) <= START(j), for all CG edges from i to j (Precedence constraint)
  - A node occupies consecutive time slots in a processor (Non-preemption constraint)
  - All nodes assigned to the same processor occupy distinct time slots (Resource constraint)

Greedy Schedule

- A greedy schedule is one that never forces a processor to be idle when one or more nodes are ready for execution
- A node is ready for execution if all its predecessors have been executed
- Observations
  - T_1 = WORK(G), for all greedy schedules
  - T_∞ = CPL(G), for all greedy schedules
- where T_P = execution time of a schedule for computation graph G on P processors

Lower Bounds on Execution Time of Schedules

- Let T_P = execution time of a schedule for computation graph G on P processors
  - Can be different for different schedules
- Lower bounds for all greedy schedules
  - Capacity bound: $T_P \geq WORK(G)/P$
  - Critical path bound: $T_P \geq CPL(G)$
- Putting them together
  - $T_P \geq \max(WORK(G)/P, CPL(G))$

Upper Bound on Execution Time of Greedy Schedules

Theorem [Graham '66]. Any greedy scheduler achieves

$$T_P \leq WORK(G)/P + CPL(G)$$

Proof sketch:

Define a time step to be complete if $\geq$ P nodes are ready at that time, or incomplete otherwise

complete time steps $\leq$ WORK(G)/P

incomplete time steps $\leq$ CPL(G)

Start time	Proc 1	Proc 2	Proc 3
0	A		
1	B		
2	C	N	
3	D	N	I
4	D	N	J
5	D	N	K
6	D	Q	L
7	E	R	M
8	F	R	O
9	G	R	P
10	H		
11			

Bounding the performance of Greedy Schedulers

Combine lower and upper bounds to get

$\max(WORK(G)/P,\ CPL(G)) \leq T_P \leq WORK(G)/P + CPL(G)$

Corollary 1: Any greedy scheduler achieves execution time T_P that is within a factor of 2 of the optimal time (since max(a,b) and (a+b) are within a factor of 2 of each other, for any $a \geq 0, b \geq 0$).

Corollary 2: Lower and upper bounds approach the same value whenever

- There's lots of parallelism, $WORK(G)/CPL(G) >> P$
- Or there's little parallelism, $WORK(G)/CPL(G) << P$

Parallel Speedup

- Define Speedup(P) = T_1 / T_P
  - Factor by which the use of P processors speeds up execution time relative to 1 processor, for a fixed input size
  - For ideal executions without overhead, 1 <= Speedup(P) <= P
  - Linear speedup
    - When Speedup(P) = k*P, for some constant k, 0 < k < 1
- Ideal Parallelism = Parallel Speedup on an unbounded number of processors

Abstract Performance Metrics

- **Basic Idea**
  - —Count operations of interest, as in big-O analysis
  - —Abstraction ignores overheads that occur on real systems
- **Calls to doWork()**
  - —Programmer inserts calls of the form, perf.doWork(N), within a step to indicate abstraction execution of N application-specific abstract operations
    - – e.g., adds, compares, stencil ops, data structure ops
  - —Multiple calls dynamically add to the execution time of current step in computation graph
- **Abstract metrics are enabled by calling**
  - **—System.setProperty(HjSystemProperty.abstractMetrics.propertyKey(), "true");**
- **If an HJ program is executed with this option, abstract metrics are printed at end of program execution with WORK(G), CPL(G), Ideal Parallelism = WORK(G)/ CPL(G)**

Course Announcements

- All Unit 1 lecture and demonstration quizzes are due by Jan 24th
  - —Quizzes are still being uploaded into edX
- Homework 1 assigned today, and is due on Jan 31st
- Next week's schedule (Jan 20-24)
  - —No lecture on Monday (MLK Jr Day)
  - —No lab next week on Monday or Wednesday
  - —We will have lectures on Wednesday & Friday as usual
- Course grading rubric (see course wiki for details)
  - —Six homeworks = 40% total (6.67% per homework)
  - —Exam 1 = 20% (Take home, assigned Feb 26th, due by Feb 28th)
  - —Exam 2 = 20% (Take home, assigned April 25th, due by May 2nd)
  - —edX quizzes = 10% total
  - —Class participation = 10% total (labs, worksheets, in-class Q&A, Piazza Q&A, bug reports, demonstration volunteers, ...)