COT 6410: Pipeline Scheduling

Michael Gabilondo

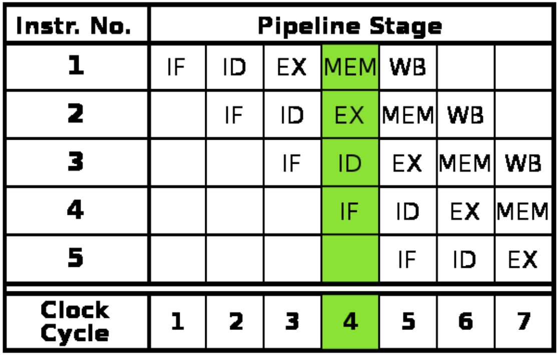
Overview

Real-World Problem

- I chose a real-world problem: **Instruction Scheduling on a Pipeline with precedence constraints between pipeline stages of jobs**. or, Pipeline-Scheduling for short.

Real-World Problem

- I chose a real-world problem: **Instruction Scheduling on a Pipeline with precedence constraints between pipeline stages of jobs**. or, PIPELINE-SCHEDULING for short.
- It's this problem, from your Computer Architecture course:



Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Real-World Problem

- A dynamic-scheduling pipelined processor with m stages tries to execute every instruction in a program without issuing a no-op instruction due to data dependencies between two instructions.

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**

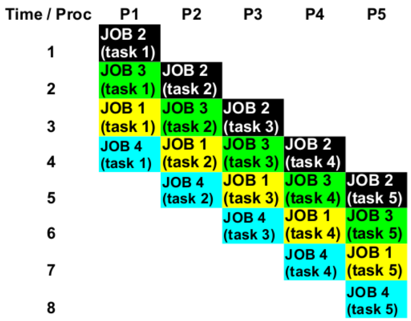
Real-World Problem

- A dynamic-scheduling pipelined processor with m stages tries to execute every instruction in a program without issuing a no-op instruction due to data dependencies between two instructions.

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

- The IBM System/360, which implemented Tomasulo's algorithm, is a notable example of a dynamic scheduling processor [Hennessy, "Computer Architecture"].

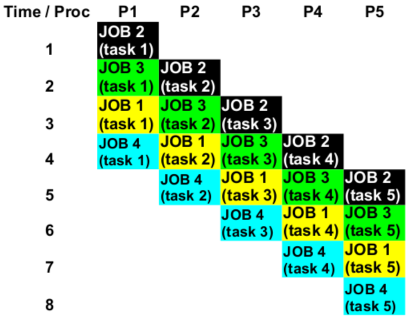
A different view of the pipeline chart



Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

- Here, the P1 P2 P3 P4 P5 are pipeline stages: **IF ID EX MEM WB.**

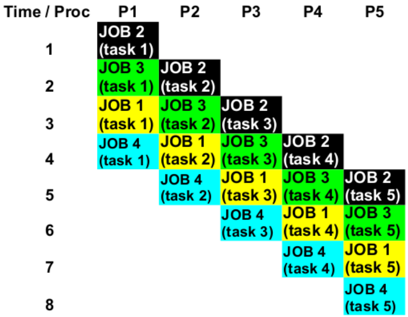
A different view of the pipeline chart



Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

- Here, the P1 P2 P3 P4 P5 are pipeline stages: **IF ID EX MEM WB.**
- Each instruction is a **JOB.**

A different view of the pipeline chart



Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

- Here, the P1 P2 P3 P4 P5 are pipeline stages: **IF ID EX MEM WB.**
- Each instruction is a **JOB.**
- Each Instruction/Job has m tasks, the number of pipeline stages. Here, m = 5.

A different view of the pipeline chart

Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

- The jobs come in through P1 from the left, and go to P2 in the next cycle, then P3, P4 and P5; in general up to P_m.

A different view of the pipeline chart

Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

- The jobs come in through P1 from the left, and go to P2 in the next cycle, then P3, P4 and P5; in general up to P_m.
- This appears to be a **shop-scheduling problem**.

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.
- The m tasks of job j are denoted $t_1(j), t_2(j), ..., t_m(j)$, task $t_i(j)$ is to be executed on processor i

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.
- The m tasks of job j are denoted $t_1(j), t_2(j), \ldots, t_m(j)$, task $t_i(j)$ is to be executed on processor i
- A processor cannot execute more than one task at a time

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.
- The m tasks of job j are denoted $t_1(j), t_2(j), \ldots, t_m(j)$, task $t_i(j)$ is to be executed on processor i
- A processor cannot execute more than one task at a time
- Two tasks of the same job cannot be executed at the same time

Defining a shop-scheduling problem

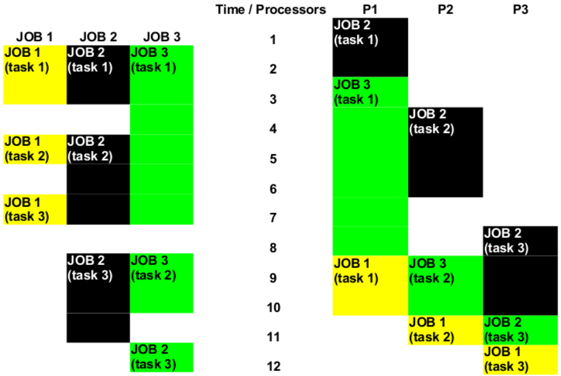
- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.
- The m tasks of job j are denoted $t_1(j), t_2(j), ..., t_m(j)$, task $t_i(j)$ is to be executed on processor i
- A processor cannot execute more than one task at a time
- Two tasks of the same job cannot be executed at the same time
- Task $i+1$ of a job j cannot start until task i has completed; i.e., the tasks are ordered

Defining a shop-scheduling problem

- Starting point is FLOW-SHOP SCHEDULING problem, which is NP-Complete; informal description
- A set J of jobs, each job $j \in J$ consisting of m tasks, and m also is the number of processors.
- The m tasks of job j are denoted $t_1(j), t_2(j), \ldots, t_m(j)$, task $t_i(j)$ is to be executed on processor i
- A processor cannot execute more than one task at a time
- Two tasks of the same job cannot be executed at the same time
- Task $i+1$ of a job j cannot start until task i has completed; i.e., the tasks are ordered
- Tasks have a length, $l(t)$; if a task t on processor i is scheduled on $\sigma_i(t)$, then nothing else can be scheduled on that processor until $\sigma_i(t) + l(t)$

Flow-Shop Scheduling

- QUESTION: Can the tasks be scheduled on the processors such that the above constraints are obeyed and the finishing time of the last task is less than D?



How is the pipeline problem different from the m-Machine flow-shop problem?

- All tasks are unit length, i.e., $l(t_i(j)) = 1$, for all $1 \leq i \leq m$, $1 \leq j \leq n$.

How is the pipeline problem different from the m-Machine flow-shop problem?

- All tasks are unit length, i.e., $l(t_i(j)) = 1$, for all $1 \leq i \leq m$, $1 \leq j \leq n$.
- $\sigma_{i+1}(j) = \sigma_i(j) + 1$, i.e., task $i+1$ of job j must begin immediately on processor $i+1$ after task i finishes on processor i.

How is the pipeline problem different from the m-Machine flow-shop problem?

- All tasks are unit length, i.e., $l(t_i(j)) = 1$, for all $1 \leq i \leq m$, $1 \leq j \leq n$.
- $\sigma_{i+1}(j) = \sigma_i(j) + 1$, i.e., task $i+1$ of job j must begin immediately on processor $i+1$ after task i finishes on processor i.
- This makes our problem a type of **no-wait** flow-shop, i.e., m-machine no-wait flow-shop.

How is the pipeline problem different from the m-Machine flow-shop problem?

- All tasks are unit length, i.e., $l(t_i(j)) = 1$, for all $1 \leq i \leq m$, $1 \leq j \leq n$.
- $\sigma_{i+1}(j) = \sigma_i(j) + 1$, i.e., task $i + 1$ of job j must begin immediately on processor $i + 1$ after task i finishes on processor i.
- This makes our problem a type of **no-wait** flow-shop, i.e., m-machine no-wait flow-shop.
- A partial order $\prec$ is defined over the tasks. $t \prec t'$ means that task t' cannot begin until task t finishes because, for example, t produces data that is required by t'.

Define our problem formally

- PIPELINE SCHEDULING

Define our problem formally

- Pipeline Scheduling
- INSTANCE:
  - A set J of jobs, a number m of processors, each job j has tasks $t_1(j), t_2(j), ..., t_m(j)$, each task having a length $l(t_i(j)) = 1$.
  - A partial order $\prec$ over the tasks, indicating precedence constraints between tasks.
  - An integer D, the deadline for the last task.

Define our problem formally

- PIPELINE SCHEDULING
- INSTANCE:
  - A set J of jobs, a number m of processors, each job j has tasks $t_1(j), t_2(j), ..., t_m(j)$, each task having a length $l(t_i(j)) = 1$.
  - A partial order $\prec$ over the tasks, indicating precedence constraints between tasks.
  - An integer D, the deadline for the last task.
- QUESTION:
  - Is there a schedule with the finishing time of the last task less than D, and the schedule is a **no-wait flow-shop schedule** and if $t_{i'}(j') \prec t_i(j)$, then $\sigma_{i'}(t_{i'}(j')) < \sigma_i(t_i(j))$?

Investigate the Complexity of Created Problem

- Is PIPELINE SCHEDULING NP-Complete?

Investigate the Complexity of Created Problem

- Is PIPELINE SCHEDULING NP-Complete?
- This is formulated as a scheduling (rather than one-processor sequencing) problem, so I attempted to use 3-Machine No-Wait Flow-Shop, but no luck.

Investigate the Complexity of Created Problem

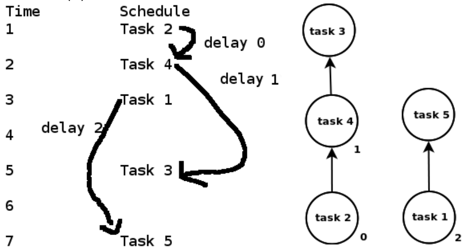
- Is PIPELINE SCHEDULING NP-Complete?
- This is formulated as a scheduling (rather than one-processor sequencing) problem, so I attempted to use 3-Machine No-Wait Flow-Shop, but no luck.
- I found another problem which represents the pipeline problem more closely.

MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS

- An NP-Complete problem

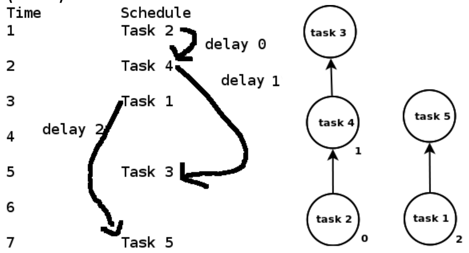
MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS

- An NP-Complete problem
- **INSTANCE**: Set T of tasks, a directed acyclic graph $G = (T,E)$ defining precedence constraints for the tasks, a positive integer D, and for each task an integer delay $0 \le d(t) \le D$.



MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS

- **QUESTION**: Is there a one-processor schedule S for T that obeys the precedence constraints and the delays, and the maximum $S(t) \leq D$? The schedule S is an injective function $S : T \to Z^+$ such that, for each edge $\langle t_i, t_j \rangle \in E, S(t_j) - S(t_i) > d(t_i) \Leftrightarrow S(t_j) \geq S(t_i) + d(t_i) + 1$



Proving NP-Completeness

- I tried to find a polynomial transformation from sequencing problem to the pipeline problem

Proving NP-Completeness

- I tried to find a polynomial transformation from sequencing problem to the pipeline problem
- In precedence constrained sequencing, the integers $d(t)$ and D can be exponential in magnitude, and the instance will still be polynomial.

Proving NP-Completeness

- I tried to find a polynomial transformation from sequencing problem to the pipeline problem
- In precedence constrained sequencing, the integers $d(t)$ and D can be exponential in magnitude, and the instance will still be polynomial.
- In pipeling scheduling, if m is exponential, then that instance will contain sets with exponential number of elements

Proving NP-Completeness

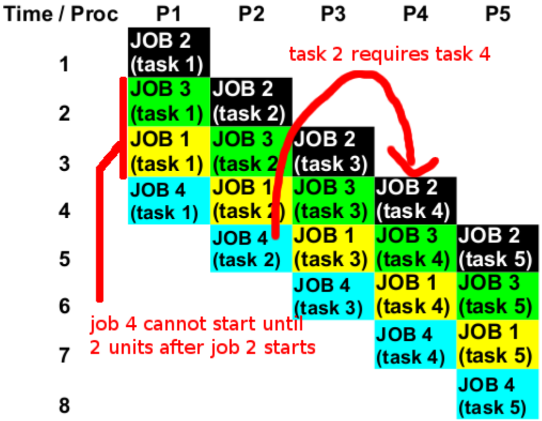
- I tried to find a polynomial transformation from sequencing problem to the pipeline problem
- In precedence constrained sequencing, the integers $d(t)$ and D can be exponential in magnitude, and the instance will still be polynomial.
- In pipeling scheduling, if m is exponential, then that instance will contain sets with exponential number of elements
- My transformation created sets of elements with D or $d(t)$ elements, so it is not a polynomial transformation

Proving NP-Completeness

- I tried to find a polynomial transformation from sequencing problem to the pipeline problem
- In precedence constrained sequencing, the integers $d(t)$ and D can be exponential in magnitude, and the instance will still be polynomial.
- In pipeling scheduling, if m is exponential, then that instance will contain sets with exponential number of elements
- My transformation created sets of elements with D or $d(t)$ elements, so it is not a polynomial transformation
- I modified pipeline scheduling so that sets of tasks were not part of the problem, and modified the precedence constraints; I will show how this new formulation is very similar to the old problem

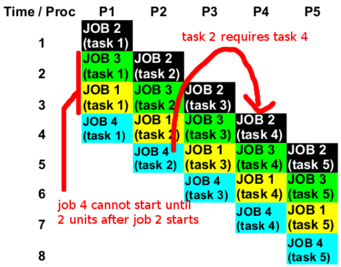
Redefining Pipeline Scheduling

- What is the effect of the precedence constraints between tasks on the starting times of the first task of each job?



Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

Redefining Pipeline Scheduling



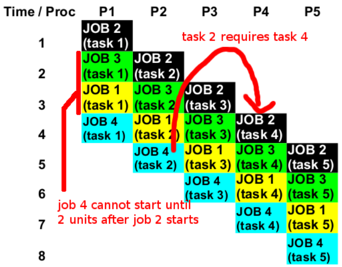
Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

task 2 requires task 4

job 4 cannot start until 2 units after job 2 starts

- **Observation**. If $t_k(j') < t_i(j)$ and $k \geq i$, then the starting time of the first task of job j, $\sigma_1(j) \geq \sigma_1(j') + k - i + 1$, if $k \geq i$.

Redefining Pipeline Scheduling

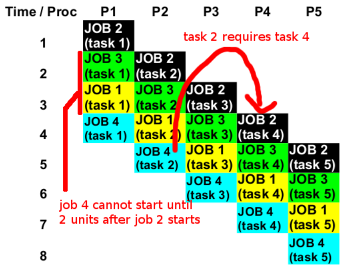


Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
8					JOB 4 (task 5)

task 2 requires task 4
job 4 cannot start until
2 units after job 2 starts

- **Observation**. If $t_k(j') < t_i(j)$ and $k \geq i$, then the starting time of the first task of job j, $\sigma_1(j) \geq \sigma_1(j') + k - i + 1$, if $k \geq i$.
- If $t_k(j') < t_i(j)$ and $k < i$, then the first task task of job j must scheduled at least one time unit after job $t_1(j')$ is scheduled.

Redefining Pipeline Scheduling



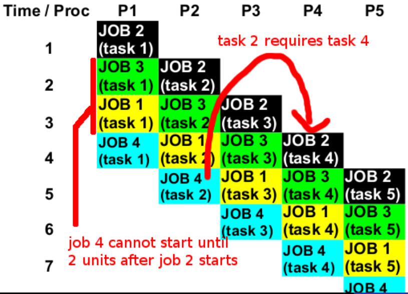
- **Observation**. If $t_k(j') < t_i(j)$ and $k \geq i$, then the starting time of the first task of job j, $\sigma_1(j) \geq \sigma_1(j') + k - i + 1$, if $k \geq i$.
- If $t_k(j') < t_i(j)$ and $k < i$, then the first task task of job j must scheduled at least one time unit after job $t_1(j')$ is scheduled.
- Also, the maximum "delay" a precedence constraint can have is $k - i + 1 \leq m$.

Redefining Pipeline Scheduling

- We don't have to worry about tasks and can just represent jobs. We can create a DAG to represent $<$, and put weights on the edges using a weight function W:

Redefining Pipeline Scheduling

- We don't have to worry about tasks and can just represent jobs. We can create a DAG to represent $<$, and put weights on the edges using a weight function W:
- $\langle t_i, t_j \rangle \in E \Leftrightarrow S(t_j) \geq S(t_i) + W(\langle t_i, t_j \rangle) + 1$
  In this case, we put an edge between Job 4 and Job 2 with weight 2.



Time / Proc	P1	P2	P3	P4	P5
1	JOB 2 (task 1)				
2	JOB 3 (task 1)	JOB 2 (task 2)			
3	JOB 1 (task 1)	JOB 3 (task 2)	JOB 2 (task 3)		
4	JOB 4 (task 1)	JOB 1 (task 2)	JOB 3 (task 3)	JOB 2 (task 4)	
5		JOB 4 (task 2)	JOB 1 (task 3)	JOB 3 (task 4)	JOB 2 (task 5)
6			JOB 4 (task 3)	JOB 1 (task 4)	JOB 3 (task 5)
7				JOB 4 (task 4)	JOB 1 (task 5)
					JOB 4

task 2 requires task 4

job 4 cannot start until 2 units after job 2 starts

Definition of Pipeline Scheduling Revised

- PIPELINE SEQUENCING problem

Definition of Pipeline Scheduling Revised

- PIPELINE SEQUENCING problem
- INSTANCE: A set I of instructions: $\{I_1, I_2, ..., I_n\}$, m pipeline stages, $G = (I, E)$, a weighted directed acyclic graph (DAG), a weight function $W : E \rightarrow \{1, 2, 3, ..., m-1\}$.

Definition of Pipeline Scheduling Revised

- Pipeline Sequencing problem
- INSTANCE: A set I of instructions: $\{I_1, I_2, ..., I_n\}$, m pipeline stages, $G = (I, E)$, a weighted directed acyclic graph (DAG), a weight function $W : E \rightarrow \{1, 2, 3, ..., m-1\}$.
- QUESTION: Is there a one-processor schedule $\sigma : I \rightarrow \mathbb{Z}_0^+$, such that σ is one-to-one,

Definition of Pipeline Scheduling Revised

- PIPELINE SEQUENCING problem
- INSTANCE: A set I of instructions: $\{I_1, I_2, ..., I_n\}$, m pipeline stages, $G = (I, E)$, a weighted directed acyclic graph (DAG), a weight function $W : E \rightarrow \{1, 2, 3, ..., m-1\}$.
- QUESTION: Is there a one-processor schedule $\sigma : I \rightarrow \mathbb{Z}_0^+$, such that σ is one-to-one,
- $\left\langle I_j, I_k \right\rangle \in E \Leftrightarrow \sigma(I_k) \geq \sigma(I_j) + W(\left\langle I_j, I_k \right\rangle) + 1$

Definition of Pipeline Scheduling Revised

- PIPELINE SEQUENCING problem
- INSTANCE: A set I of instructions: $\{I_1, I_2, ..., I_n\}$, m pipeline stages, $G = (I, E)$, a weighted directed acyclic graph (DAG), a weight function $W : E \to \{1, 2, 3, ..., m-1\}$.
- QUESTION: Is there a one-processor schedule $\sigma : I \to \mathbb{Z}_0^+$, such that σ is one-to-one,
- $\left\langle I_j, I_k \right\rangle \in E \Leftrightarrow \sigma(I_k) \geq \sigma(I_j) + W(\left\langle I_j, I_k \right\rangle) + 1$
- For the maximum $\sigma(I)$, does $\sigma(I) + (m-1) \leq D$?

Proof that Pipeline Sequencing is NP-Complete

- Accept an instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS.

Proof that Pipeline Sequencing is NP-Complete

- Accept an instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS.
- INSTANCE: Set T of tasks, a directed acyclic graph $G = (T, E)$ defining precedence constraints for the tasks, a positive integer D, and for each task an integer delay $0 \leq d(t) \leq D$.

Proof that Pipeline Sequencing is NP-Complete

- Accept an instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS.
- INSTANCE: Set T of tasks, a directed acyclic graph $G = (T, E)$ defining precedence constraints for the tasks, a positive integer D, and for each task an integer delay $0 \leq d(t) \leq D$.
- QUESTION: Is there a one-processor schedule S for T that obeys the precedence constraints and the delays, and the maximum $S(t) \leq D$? The schedule S is an injective function $S : T \rightarrow Z^+$ such that, for each edge $\left\langle t_i, t_j \right\rangle \in E, S(t_j) - S(t_i) > d(t_i) \Leftrightarrow S(t_j) \geq S(t_i) + d(t_i) + 1$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
    - $I \leftarrow T$,

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
    - $I \leftarrow T$,
    - $E' \leftarrow E$
    - For each edge $\left\langle t_i, t_j \right\rangle \in E$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$
  - For each edge $\left\langle t_i, t_j \right\rangle \in E$
    - $W(\left\langle t_i, t_j \right\rangle) \leftarrow d(t_i)$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$
  - For each edge $\left\langle t_i, t_j \right\rangle \in E$
    - $W(\left\langle t_i, t_j \right\rangle) \leftarrow d(t_i)$
  - $m \leftarrow \max_{t \in T}\{d(t)\} + 1$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$
  - For each edge $\left\langle t_i, t_j \right\rangle \in E$
    - $W(\left\langle t_i, t_j \right\rangle) \leftarrow d(t_i)$
  - $m \leftarrow \max_{t \in T}\{d(t)\} + 1$
  - $D' \leftarrow D + (m - 1)$

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$
  - For each edge $\left\langle t_i, t_j \right\rangle \in E$
    - $W(\left\langle t_i, t_j \right\rangle) \leftarrow d(t_i)$
  - $m \leftarrow \max_{t \in T}\{d(t)\} + 1$
  - $D' \leftarrow D + (m - 1)$
- Solve PIPELINE SEQUENCING and return the answer

Proof that Pipeline Sequencing is NP-Complete

- Create an instance of PIPELINE SEQUENCING, using the existing instance of MINIMUM PRECEDENCE CONSTRAINED SEQUENCING WITH DELAYS
  - $I \leftarrow T$,
  - $E' \leftarrow E$
  - For each edge $\langle t_i, t_j \rangle \in E$
    - $W(\langle t_i, t_j \rangle) \leftarrow d(t_i)$
  - $m \leftarrow \max_{t \in T}\{d(t)\} + 1$
  - $D' \leftarrow D + (m-1)$
- Solve PIPELINE SEQUENCING and return the answer
- The time complexity is $O(T+E)$; this is a polynomial transformation.

Proof that Pipeline Sequencing is NP-Complete

- Notice that if $\left\langle t_i, t_j \right\rangle \in E$, then $S(t_j) \geq S(t_i) + d(t_i) + 1$. The equivalent condition can hold in PIPELINE SEQUENCING by adding that edge to E' and and setting the weight on that edge to $d(t_i)$.

Proof that Pipeline Sequencing is NP-Complete

- Notice that if $\langle t_i, t_j \rangle \in E$, then $S(t_j) \geq S(t_i) + d(t_i) + 1$. The equivalent condition can hold in PIPELINE SEQUENCING by adding that edge to E' and and setting the weight on that edge to $d(t_i)$.
- The respective instances and constraints of the two problems have been constructed to be equivalent:

Proof that Pipeline Sequencing is NP-Complete

- Notice that if $\langle t_i, t_j \rangle \in E$, then $S(t_j) \geq S(t_i) + d(t_i) + 1$. The equivalent condition can hold in PIPELINE SEQUENCING by adding that edge to E' and and setting the weight on that edge to $d(t_i)$.
- The respective instances and constraints of the two problems have been constructed to be equivalent:
- If the SEQUENCING WITH DELAYS instance is true and the last job is scheduled at time t, then the last job of PIPELINE SEQUENCING will also be scheduled at time t. The PIPELINE SEQUENCING instance adds $m-1$ time units to the finishing time t, so we set D' to $D+(m-1)$.

Proof that Pipeline Sequencing is NP-Complete

- If the PIPELINE SEQUENCING instance is true, then it has a schedule that meets its constraints; if the last job of the PIPELINE SEQUENCING instance is scheduled at time t, then so will the corresponding job in SEQUENCING WITH DELAYS also be scheduled at time t. In PIPELINE SEQUENCING, the problem ends $m+1$ time units after the last job; this is why $m+1$ time units were added to the deadline D when constructing the PIPELINE SEQUENCING instance.

Proof that Pipeline Sequencing is NP-Complete

- If the PIPELINE SEQUENCING instance is true, then it has a schedule that meets its constraints; if the last job of the PIPELINE SEQUENCING instance is scheduled at time t, then so will the corresponding job in SEQUENCING WITH DELAYS also be scheduled at time t. In PIPELINE SEQUENCING, the problem ends $m+1$ time units after the last job; this is why $m+1$ time units were added to the deadline D when constructing the PIPELINE SEQUENCING instance.
- Thus, the constructed instance is true if and only if the base instance is true.

Proof that Pipeline Sequencing is NP-Complete

- If the PIPELINE SEQUENCING instance is true, then it has a schedule that meets its constraints; if the last job of the PIPELINE SEQUENCING instance is scheduled at time t, then so will the corresponding job in SEQUENCING WITH DELAYS also be scheduled at time t. In PIPELINE SEQUENCING, the problem ends $m+1$ time units after the last job; this is why $m+1$ time units were added to the deadline D when constructing the PIPELINE SEQUENCING instance.
- Thus, the constructed instance is true if and only if the base instance is true.
- Now, once I show Pipeline Sequencing is in NP, then that proves it is NP-Complete.

Pipeline Sequencing is in NP

- Is PIPELINE SEQUENCING in NP? An oracle can provide the starting time for each of the instructions, and an algorithm simply needs to check the DAG to see if the schedule is valid. For each edge in the DAG, the algorithm simply needs to check if the starting time of the dependant task is large enough and comes after the the first task; it also needs to check the deadline is met by simply checking the starting time of the last instruction. So the problem is NP.

Conclusion

- This has show that the real-life problem of **Instruction Scheduling on a Pipeline with precedence constraints between pipeline stages of jobs** is NP-Hard for an **unbounded** number of processors. In practice, this result does not apply because most machines have a small number of pipeline stages.
- We were unable to show that the **pipeline scheduling problem with sets of m tasks** was NP-Complete because our **polynomial transormation** was creating instances where m was exponential. We have not shown it is not NP-complete, however; we don't know.

Conclusion

- But, I do not think it is NP-Complete, because the paper by Hennesy showed that the SEQUENCING WITH DELAYS problem was NP-Complete for **unbounded** m. They had to introduce extra constraints into the problem to get a bound on m.

References

- "A compendium of NP optimization problems", Pierluigi Crescenzi and Viggo Kann. 1998.
- "Computers and Intractability: A Guide to the Theory of NP-Completeness", Garey and Johnson. 1979.
- "Postpass Code Optimization of Pipeline Constraints", Hennessy, John L. and Gross, Thomas. 1983.
- "The Three-Machine No-Wait Flow Shop is NP-Complete", Hans Röck. 1984.