

Scheduling DAGs: recent results and challenges

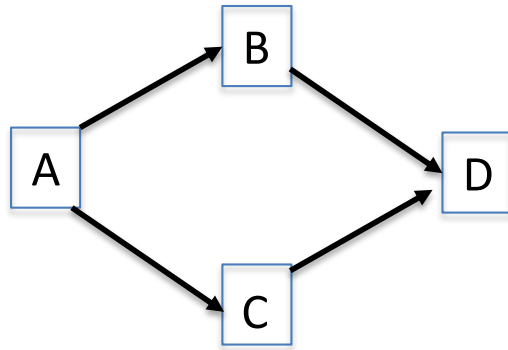
Alberto Marchetti-Spaccamela

Sapienza University of Rome

The DAG model

- **DAGs can be used to describe a wide range of complex applications and parallel computing jobs**
- **Problem: given a DAG G and a multiprocessor platform find a schedule of G**
- **Hu [1961]:** *“Parallel sequencing and assembly line problems”*
- **Graham [1966,1969]:** *“Bounds on multiprocessing timing anomalies”*

The DAG model



Parameters of a DAG-job

- a directed acyclic graph $G_i = (V_i, E_i)$
- node $v \in V_i$ has WCET C_v
- $\text{arc}(u, v)$ indicates precedence constraint $u < v$

The model captures the intra-parallelism of jobs

Feasibility problem

Given: a DAG **G**, **m** identical parallel machines, a deadline **D**

Goal: Decide if **G** can be scheduled before deadline **D**

- NP-complete - unitary processing time [Ullman 1975]
- Easy to approximate within $2 - 1/m$ - list scheduling [Graham 1969]
- Hard to approximate better than $4/3$ - unitary processing time [Lenstra et al. 1978]
- Assuming the unique game conjecture It is hard to approximate better than $2 - 1/m$ [Svensson 2011]

The DAG model

The DAG model has been studied for a long time:

- survey published in 1999 has received more than 1700 citations
- Google 'dag scheduling' 74000 documents - since 2021: 7470

Main focus:

Design and analysis of scheduling algorithms in multi-core processor or in a computing cluster with the objective to minimize the total execution time (makespan) or other objective and/or the amount of computing resources (cost)

Extensions of the model

- Communication among nodes
- Recurrent DAG jobs
- Energy requirements
- Memory requirements

Today's new challenges

- New applications (e.g. autonomous cars) are being realized via an evolving repertoire of artificial-intelligence (AI) algorithms that realize new capabilities (e.g. visual perception and decision-making)
- These algorithms often have complex dataflow dependencies expressible using processing graphs that can be computationally intensive, requiring the usage of hardware accelerators (HACs),

Challenges

- **complex graph-based workloads**
- **complex heterogeneous hardware**

This talk

I will focus on extensions of the DAG model

- 1. The conditional C-DAG model**
- 2. Recurrent DAG-jobs: sporadic DAG model**
- 3. Heterogenous hardware: the HPC-DAG model**

The focus is theoretical mainly addressing the

- Complexity of feasibility problems

I do not present/discuss

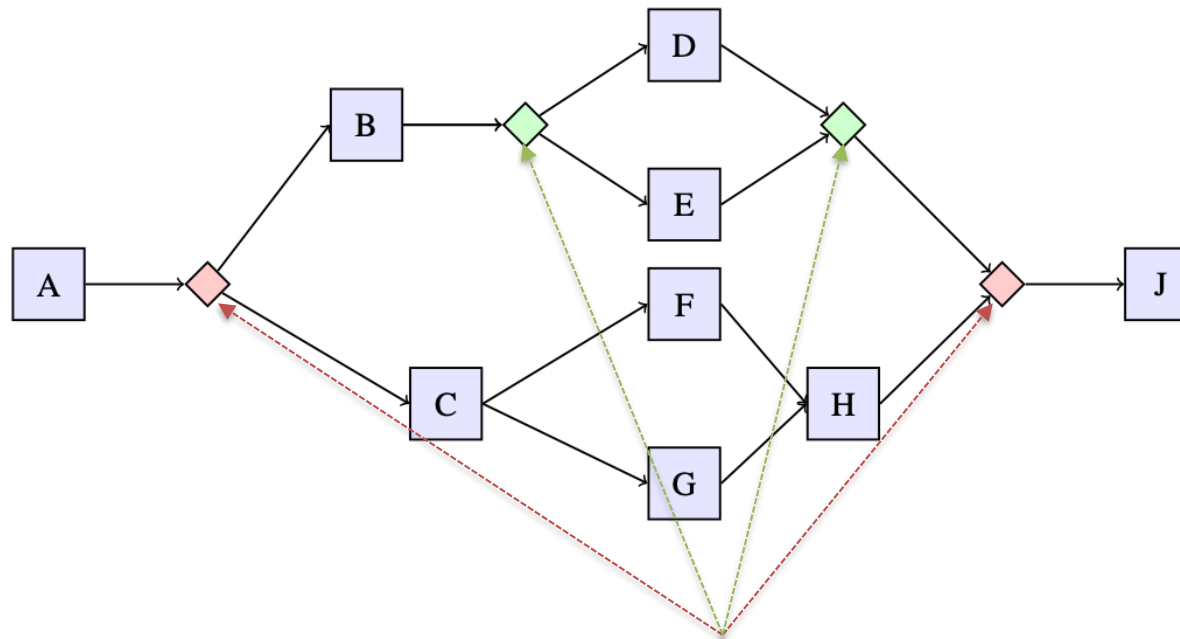
- heuristics (and their comparison)
- other objective functions
- detailed presentation of the many problem variations

The conditional C-DAG model

The conditional C-DAG model

- Feasibility of C-DAG
- A crash course on PSPACE
- Feasibility of C-DAG is PSPACE complete
- List scheduling C-DAGs
- Open problems

Conditional DAG: C-DAG model



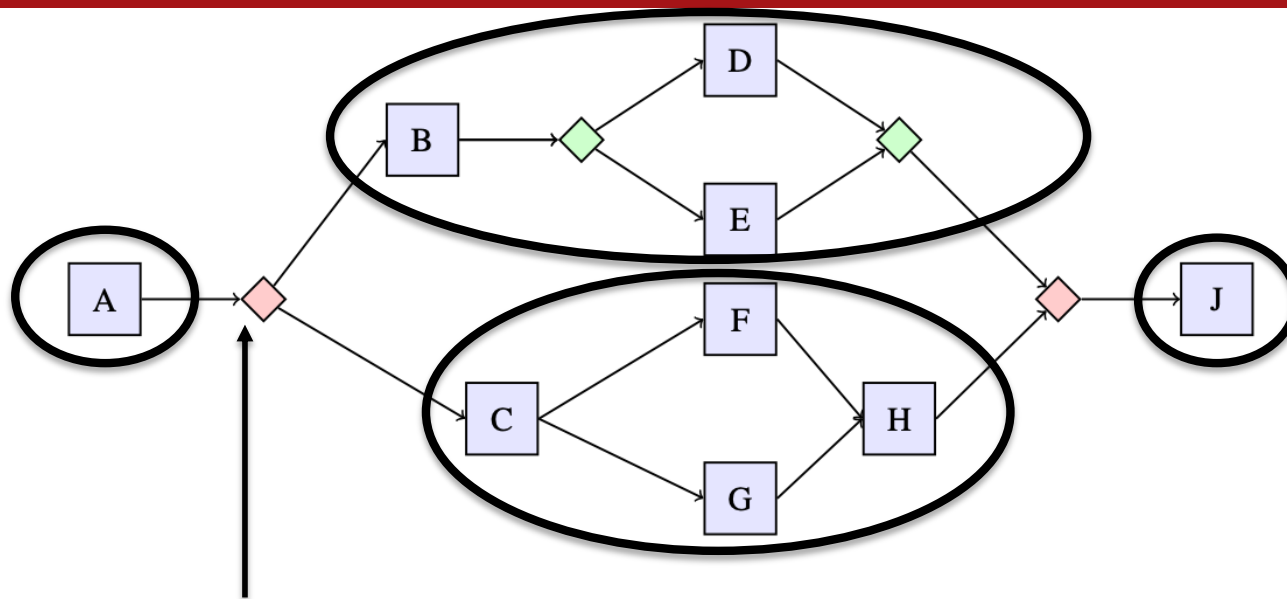
A **conditional construct**

A generalization of the **DAG task model**

Exposes **intra-task parallelism** (as do “regular” DAG tasks)

Models **conditional execution** of code

Conditional DAG (C-DAG)

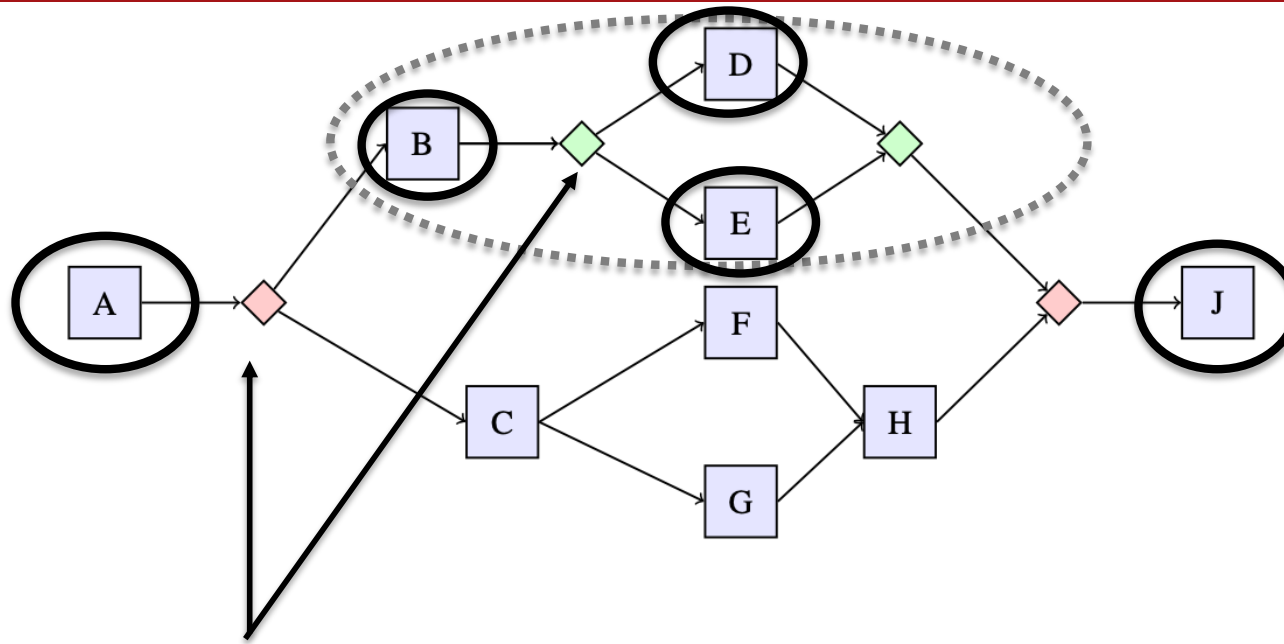


A **boolean expression** is evaluated

Conditional nodes are special nodes that are defined in pairs

- a conditional is evaluated
- based on the outcome, control flows along one of the possible paths different paths join at a common point

Conditional DAG (C-DAG)



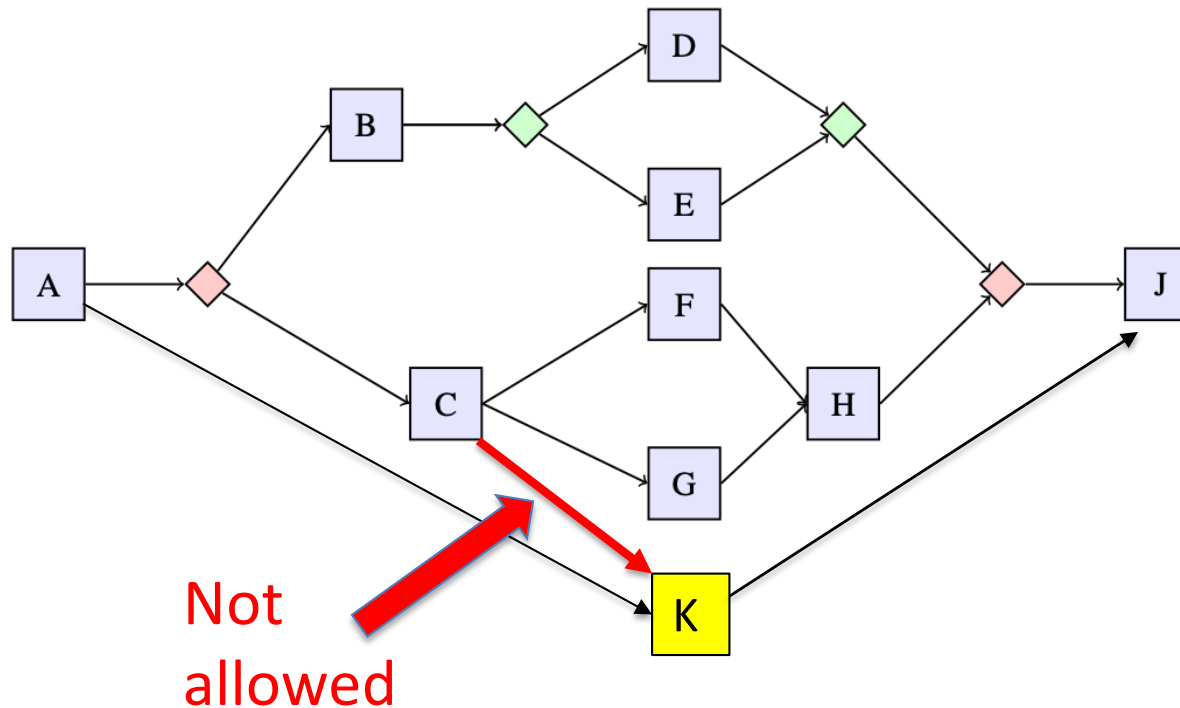
A **boolean expression** is evaluated

A generalization of the **DAG task model**

Exposes **intra-task parallelism** (as do “regular” DAG tasks)

Models **conditional execution** of code

Conditional DAG (C-DAG)



Conditional branches are **well nested**

Wlog we assume

- each node has unitary execution time
- conditional nodes () have 0 execution time

Feasibility analysis

Feasibility problem

Determine, prior to deployment, whether all timing constraints will **always** be met: relevant in real time scheduling

Formally

Given: a C-DAG G , m identical parallel machines and a deadline D

Goal: Decide if G can be scheduled before deadline D

Feasibility analysis should be **efficient**

Complexity Classes

Feasibility analysis of conditional DAG is **PSPACE complete** [Baruah MS 2021]

First natural scheduling problem to be shown to be PSPACE complete

(Poly with

A problem that is **hard** for PSPACE cannot be efficiently solved using ILP-solvers

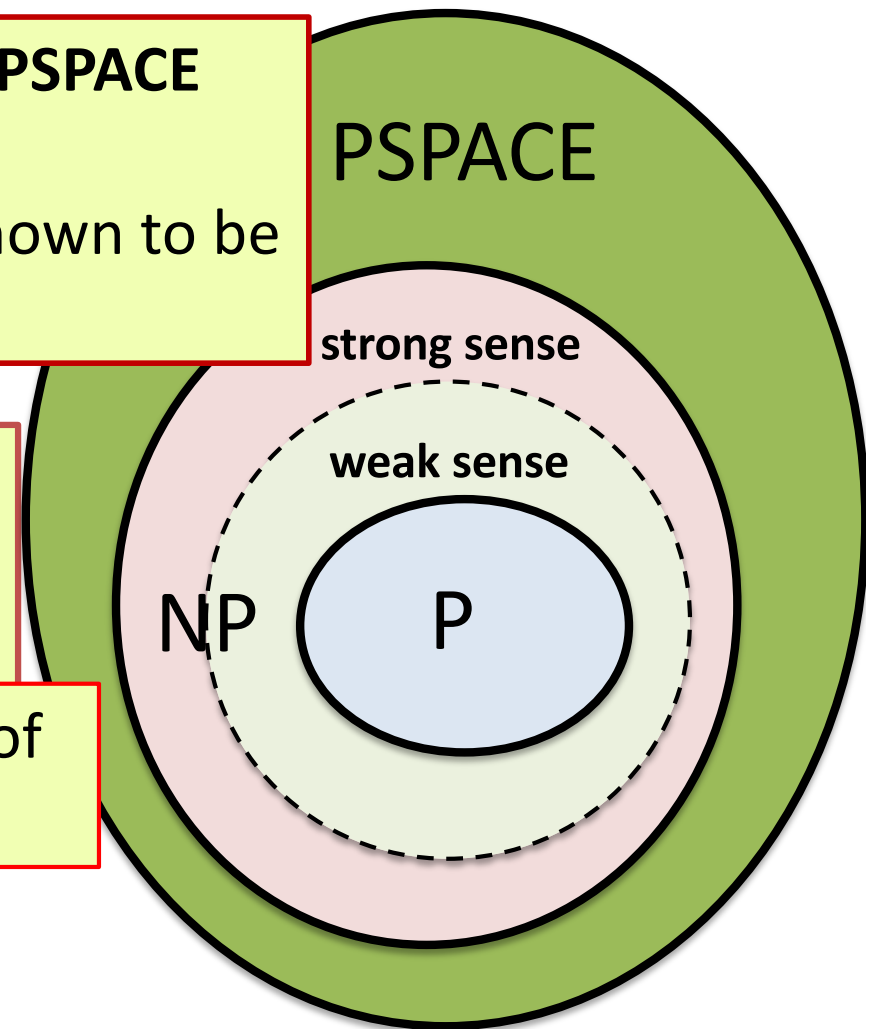
NP - strong

Feasibility analysis of **DAG** is NP strong

NP - weak

Polynomial Time

P



Why the C-DAG feasibility problem is difficult

Given a C-DAG G with k conditions and a deadline D we want to know whether

- is it possible to schedule G within in D time units for all possible outcomes of conditional instructions?

We now see that

- feasibility of the 2^k (non conditional) DAGs, obtained by considering all possible outcomes of the k conditional instructions

is NOT SUFFICIENT to claim that

- G can be feasible scheduled within the deadline

Why?

The feasibility test of G is not clairvoyant: it does not know the outcome of conditional instructions before their execution

Why the C-DAG feasibility problem is difficult

Given a C-DAG G with k conditions and a deadline D we want to know whether

- is it possible to schedule G within in D time units for all possible outcomes of conditional instructions?

We now see that

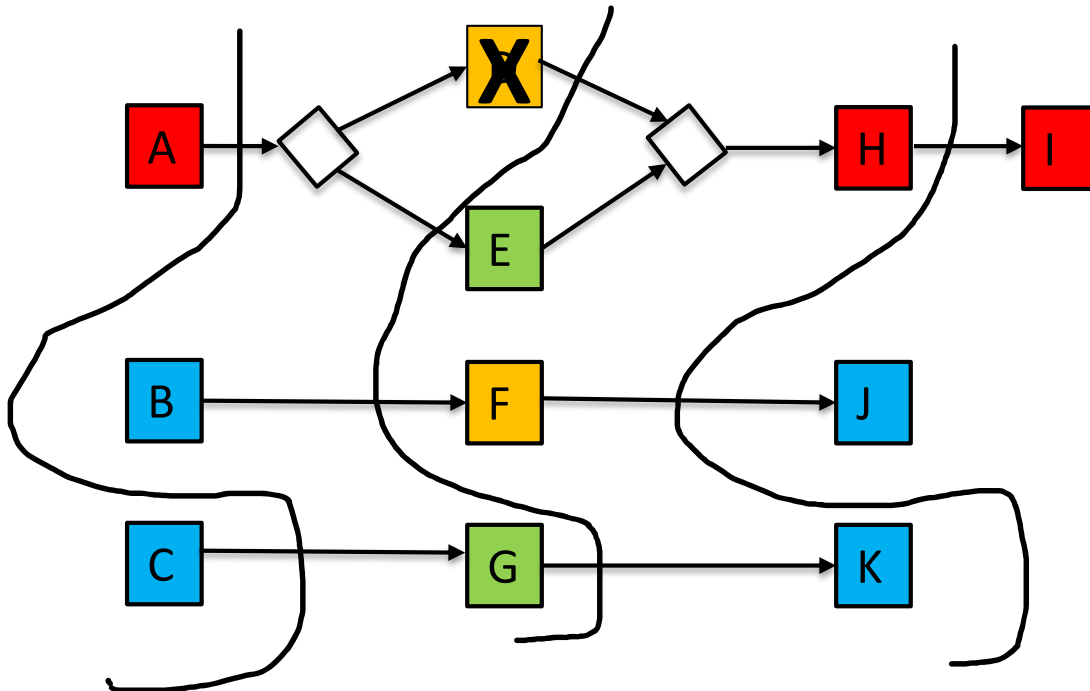
- **feasibility of the 2^k (non conditional) DAGs, obtained by considering all possible outcomes of the k conditional instructions**

is NOT SUFFICIENT to claim that

- **G can be feasible scheduled within the deadline**

In the sequel for simplicity we restrict to the case of dedicated processors (for every node is executed on a specific processor)

Feasibility Analysis of Conditional DAGs



Each square vertex has WCET equal to one (conditional vertices –WCET zero)

Processor assignments are color-coded:

- A, H, & I share processor, ■
- B, C, J, & K; ■
- D & F; ■
- E & G. ■

The DAG is always schedulable within 4 time units.

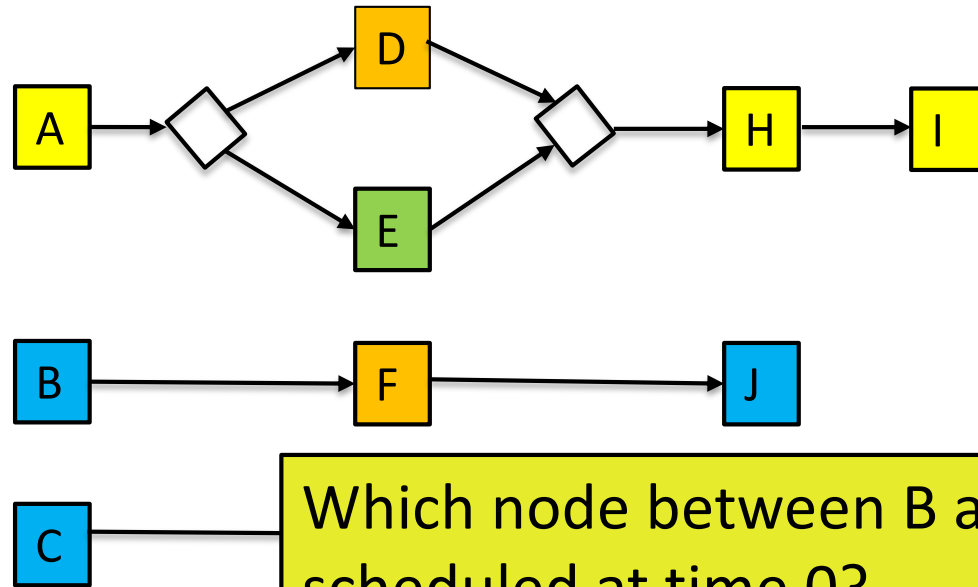
If the conditional construct executes D then

- t=0 schedule A, C
- t=1 schedule B, D, G
- t=2 schedule H, K, F
- t=3 schedule I, J

If the conditional construct executes E then: invert execut. order of (B,C) (K,J)

- t=0 schedule A, ~~C~~, B
- t=1 schedule ~~B~~, C, ~~D~~, E, ~~G~~, F
- t=2 schedule H, ~~K~~, J, ~~F~~, G
- t=3 schedule I, ~~J~~, K

Feasibility Analysis of Conditional DAGs



Each vertex has WCET equal to one (except the conditional vertices –WCET zero)

Processor assignments are color-coded:

- A, H, & I share a processor,

Which node between B and C should be scheduled at time 0?

- E & G.

The DAG is always schedulable within 4 time units.

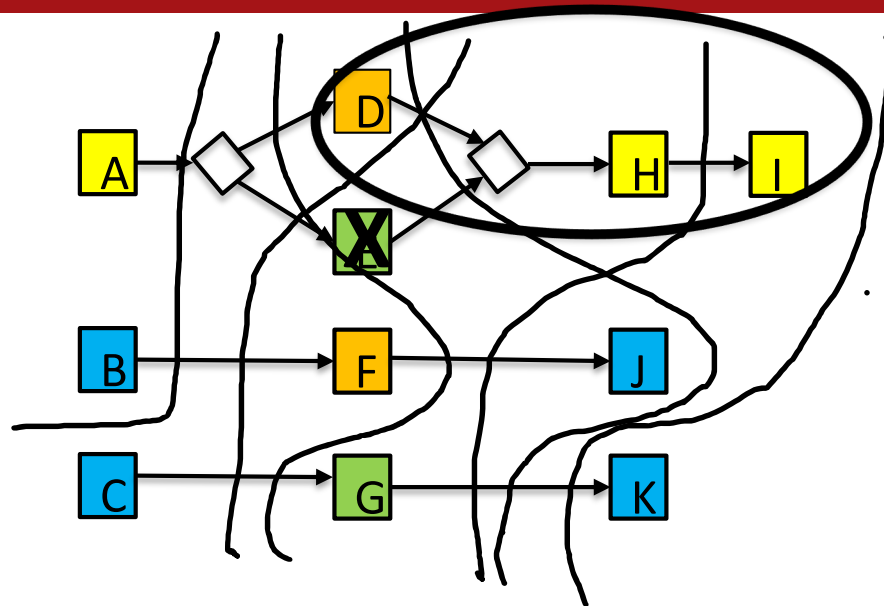
If the conditional construct executes D then

- t=0 schedule A, C
- t=1 schedule B, D, G
- t=2 schedule H, K, F
- t=3 schedule I, J

If the conditional construct executes E then

- t=0 schedule A, ~~C~~, B
- t=1 schedule ~~B~~, C, ~~D~~, E, ~~G~~, F
- t=2 schedule H, ~~K~~, J, ~~F~~, G
- t=3 schedule I, ~~J~~, K

Feasibility Analysis of Conditional DAGs



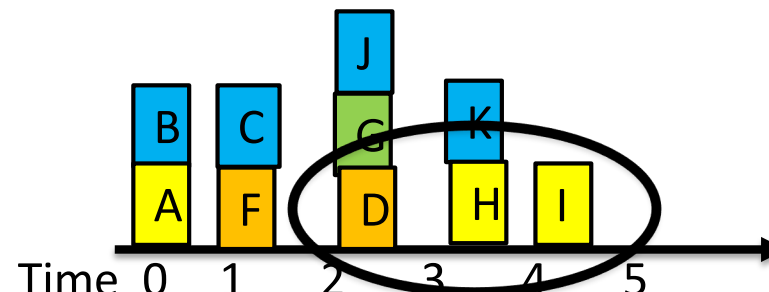
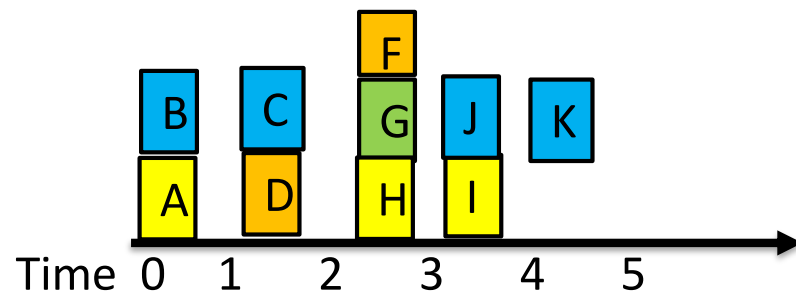
Each vertex has WCET equal to one (except the conditional vertices –WCET zero). Processor assignments are color-coded:

- A, H, & I share a processor,
- B, C, J, & K;
- D & F;
- E & G.

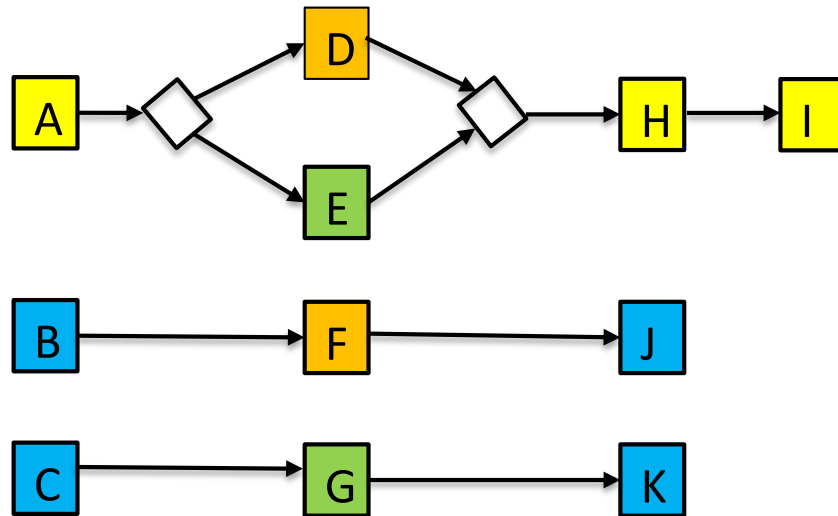
NOTE: if at $t=0$ A, B (i.o. A, C) are scheduled and the conditional construct executes D then makespan is 5. In fact

$t=1$ schedule C, and one between D and F

If D is scheduled at $t=1$ then at time 3 both J and K require the blue processor
If F is scheduled at $t=1$. then execution of D, H, I will complete at time 5



Feasibility Analysis of Conditional DAGs



Each vertex has WCET equal to one (except the conditional vertices –WCET zero). Processor assignments are color-coded:

- A, H, & I share a processor,
- B, C, J, & K;
- D & F;
- E & G.

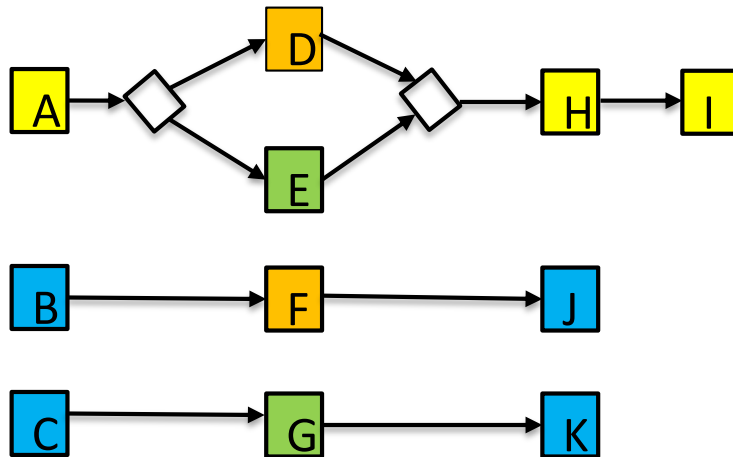
We have shown that

1. To schedule the graph in 4 time units we must know at time $t=0$ the outcome of the conditional instruction
2. This is not possible for nonclairvoyant algorithms

Therefore

The C-DAG cannot be feasibly scheduled in 4 time units

Feasibility Analysis of Conditional DAGs



Each vertex has WCET equal to one (except the conditional vertices –WCET zero). Processor assignments are color-coded:

- A, H, & I share a processor,
- B, C, J, & K;
- D & F;
- E & G.

- The previous example (1 conditional node) can be generalized
- If the DAG G has k conditions to prove feasibility of G it is not sufficient to show feasibility for 2^k subproblems (obtained by considering all possible outcomes of k conditions)

PSPACE hardness of the problem formalizes this observation

A crash course on PSPACE

NP: solvable in polynomial time by a non deterministic algorithm

- Property: 'YES' answers of NP-complete problems have short (polynomial) certificates.

PSPACE: solvable in polynomial space by a deterministic algorithm

- Property: a 'YES' answer of PSPACE-complete problems is a winning strategy for a two-player game with perfect information, where players make alternate moves (e.g. chess, checkers).
- **Winning strategy: Player 1 has a winning strategy if there exists a 1st move, such that for all possible 1st moves of Player 2, there exists a 2nd move for Player 1 s.t. and Player 1 wins at the end.**

It is conjectured that
 $P \subsetneq NP \subsetneq PSPACE$ (proper inclusion)

A crash course on PSPACE

To formalize 2 players games we need

1. inputs of any dimension

- Checkers, chess: introduce a board of size $n \times n$ (instead of 8×8)
Question: is there a winning strategy for the first player?
- **Geography.** Alice names **capital city c** of a country. Bob names a **capital city c'** that starts with the letter on which c ends. Alice and Bob repeat this game until one player is unable to continue.

Example: Budapest → Tokyo → Ottawa → Ankara →
Amsterdam → Moscow → Washington → Nairobi → ...

We formalize Geography using graphs: Given a directed graph $G = (V, E)$ and a start node s , two players alternate turns by following, if possible, an edge leaving the current node to an unvisited node.

Question: is there a strategy for the first player that guarantees to do the last move?

A crash course on PSPACE

Geography on graphs

Given a directed graph $G = (V, E)$ and a start node s , two players P_1 and P_2 alternate turns by following, if possible, an edge leaving the current node to an unvisited node.

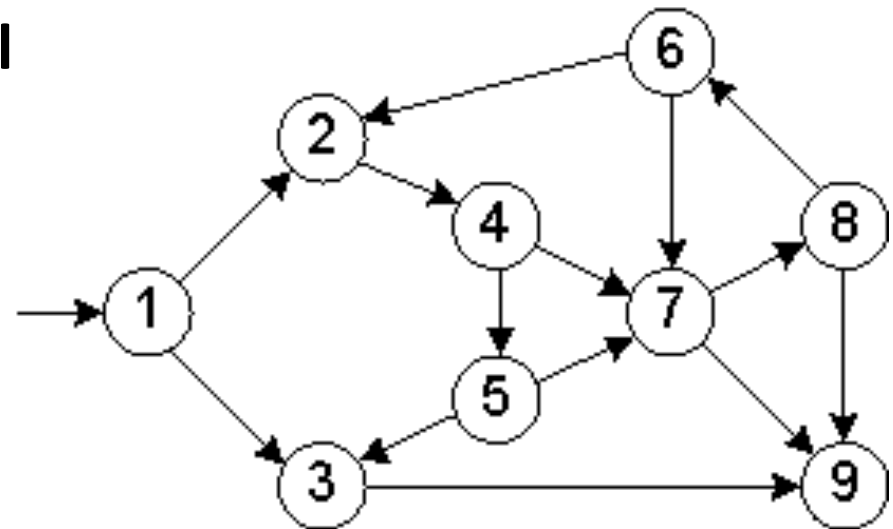
The game ends when there is no a feasible move.

The player that makes the last legal

Question: Is there a winning strategy for Player 1?

Starting node 1

- if a player reaches either node 3 or 8 then she loses
- if a player reaches 9 then she wins



A crash course on PSPACE

Geography on graphs

Given a directed graph $G = (V, E)$ and a start node s , two players P_1 and P_2 alternate turns by following, if possible, an edge leaving the current node to an unvisited node.

The game ends when there is no a feasible move.

The player that makes the last legal move wins.

Starting node 1

There is a winning strategy for P_1

• P_1 chooses 2

•

•

•

•

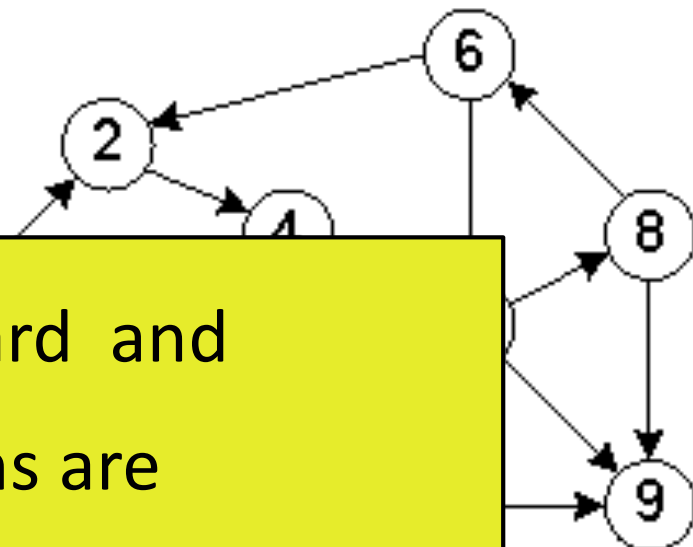
•

•

Checkers on $N \times N$ board and

Geography on graphs are

PSPACE-complete



A crash course on PSPACE

To formalize 2 players games we need

✓ 1. Inputs of any dimension

2. Quantifiers

2. **Quantifiers**: We need quantifiers to express the meaning of English words including 'all' and 'some':

“All men are Mortal” or “Some cats do not have fur”

Two kinds of quantifiers: Universal and Existential

- **Universal Quantifier: represented by \forall**

The symbol is translated as and means “for all”, “given any”, “for each,” or “for every”.

- **Existential Quantifier: represented by \exists**

The symbol is translated as and means variously “for some,” “there exists,” “there is a,” or “for at least one”.

A crash course on PSPACE

Quantified Boolean Formula (QBF)

- Quantified Boolean Formula is the canonical problem for PSPACE (like Satisfiability is for NP)
- A QBF is a boolean formula in which **boolean variables are quantified using \exists and \forall**
- A Fully Quantified Boolean Formula QBF where \exists and \forall alternate and F is a boolean formula has the form (n is even)

$$\exists x_1 \forall x_2 \exists x_3 \dots \forall x_n F(x_1, x_2, \dots, x_n)$$

- **All variables are quantified; therefore, a Fully QBF is either TRUE or FALSE; examples**
 $[\exists x \forall y (x \text{ OR } y)]$ is TRUE $[\exists x \forall y (x \text{ AND } y)]$ is FALSE

Note: ordering of quantifiers is essential. In fact
 $\exists x \forall y (x = y)$ is FALSE $\forall x \exists y (x = y)$ is TRUE

Feasibility Analysis of Conditional DAG Tasks

Part 2: Putting all pieces together

2.1 The conditional DAG feasibility problem as a two-players game (the **scheduler** vs the **environment**)

Given a C-DAG and a deadline D , then

- the first move of player 1 (the **scheduler**) is to decide the set of jobs to be scheduled until the first branch is executed;
- then player 2 (the **environment**) decides the outcome of the branch;
- then player 1 decides the set of jobs to be scheduled until the second branch is executed
-
- the game continues until the scheduling is completed

The scheduler wins the game if and only if her strategy is able to complete the schedule in D time units for all possible decisions of the second player (**i.e. for all possible sequences of branches**).

Feasibility Analysis of Conditional DAG Tasks

Part 2: Putting all pieces together

2.2 Reducing QBF to C-DAG feasibility

Given a Fully QBF F we define a C-DAG and a deadline D

- For each existentially quantified variable x_i , $i=1,3,5,\dots$ we define two subdags A_{i1} A_{i2}
- For each universally quantified variable x_i , $i=2,4,6,\dots$ we define one conditional subdag B_i , whose branches define two sets of nodes to be scheduled
- There is a subdag C that encodes the formula
- Precedence constraints are defined among subdags A_{i1} A_{i2} B_i C

The scheduler completes the schedule in D time units for all possible decisions of the second player (**i.e. for all possible sequences of branches**) if and only if the QBF F is TRUE

List Scheduling of C-DAGs

Since the feasibility problem is hard we rely on heuristics

List scheduling [Graham 66]: Given a DAG G and a fixed-priority order \prec , at any point of time schedule the available node with higher priority

Graham's bound

- for any DAG G and any fixed-priority order \prec , list scheduling is $2 - 1/m$ approximate [Graham 1966]

Generalizing Graham's bound:

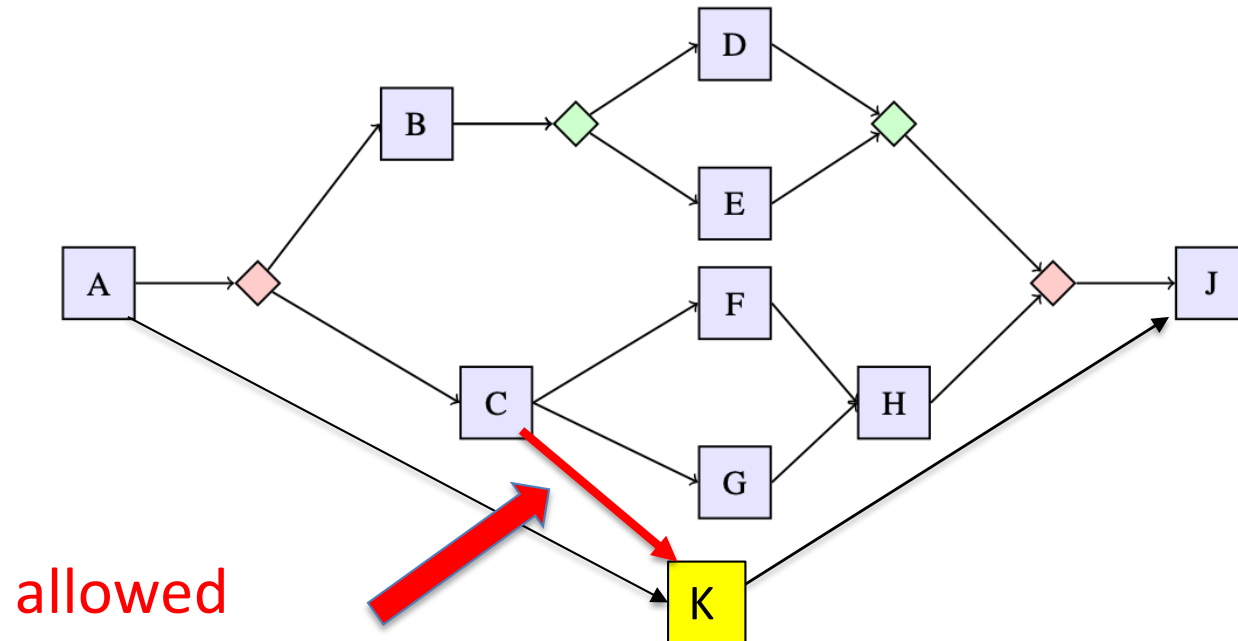
- for any C-DAG and any fixed-priority order \prec , list scheduling is $2 - 1/m$ approximate [MS Megow Skutella Stougie Schlöter 2020]

Feasibility of List Scheduling for C-DAGs

List scheduling feasibility

- Input (G, \prec, D) : a C-DAG G , a fixed-priority order \prec and a deadline D we want to know whether
- is it possible to schedule G within in D time units for all possible outcomes of conditional instructions using list scheduling?
- **Complexity of of list scheduling feasibility**
[MS Megow Skutella Stougie Schlöter 2020]
- C-DAG is strongly coNP-complete ($m > 2$)
- if $m = 2$ C-DAG is weakly coNP-complete
- approximating C-DAG in a ratio better than $7/5$ is NP-hard
- C-DAG is weakly coNP-complete even if each possible execution is a constant number of disjoint chains.

General conditional DAGs



Not well nested C-DAG (also C-DAG with shared nodes)

- List scheduling Feasibility is NP-hard even on a single processor [MS Megow Skutella Stougie Schlöter 2020]
- Computing the maximum volume (sum of processing times) is NP-hard [Sun et al. 2020]

Open problems: Feasibility of C-DAG

1. If the number of conditions is constant then the problem is in NP and there exists an ILP formulation to check feasibility [Baruah,MS]. However the proposed formulation it is not practical even for small instances

- Find an ILP formulation with an acceptable complexity

2. previous hardness results use reductions to instances with an unbounded no. of machines.

- What is the complexity in the case of a fixed number of machines? (both for feasibility and list scheduling)

3. Special classes

- Are there other restricted (and interesting) classes that make the problem (more) tractable? Equivalently is there a input parameter s.t. if the parameter is bounded then the problem is polynomially solvable?

Recurrent DAGs

Recurrent DAGs

- The sporadic DAG model
- Performance metric: analysis of EDF, DM
- The sporadic C-DAG model
- Open problems

Sporadic DAG task model

In real-time systems we are interested in scheduling a set of recurrent jobs that are periodically released

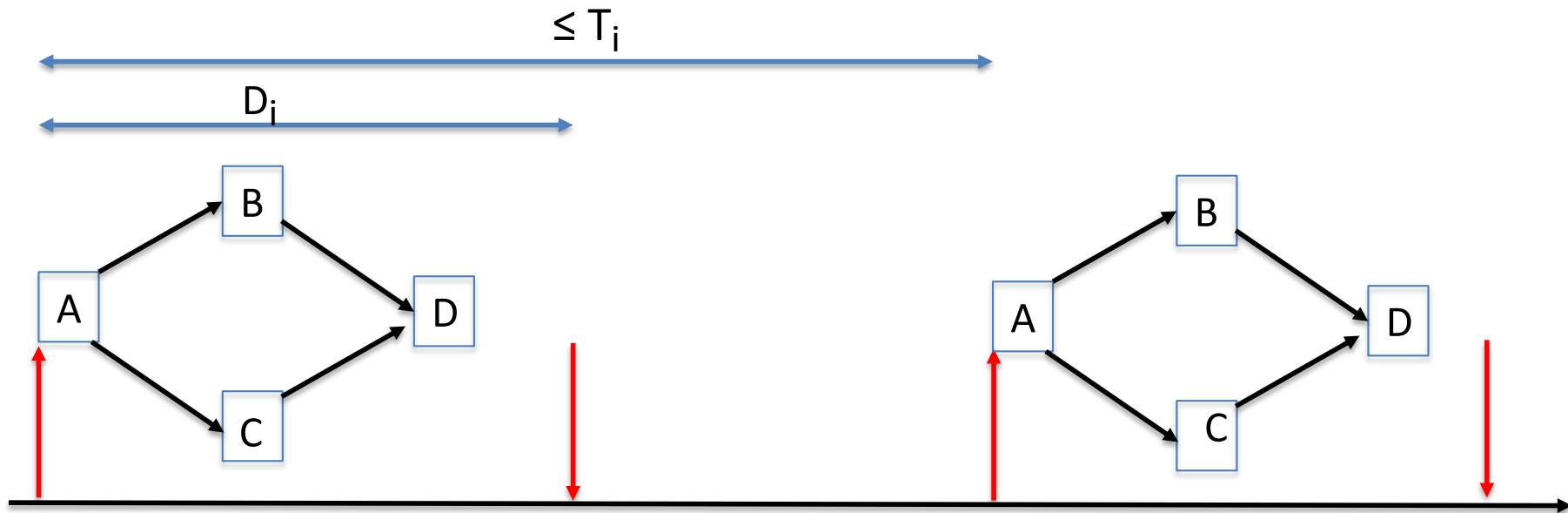
A sporadic DAG task is defined by a triple (G,D,T)

- a DAG G , a deadline D , a period T

A sporadic DAG task releases a sequence of DAG jobs;

- if a dag-job is released at time t then the next DAG-job is released at or after time $t+T$
- A sporadic DAG task releases infinite sequences of infinite length of the same DAG

System of sporadic DAG-tasks



A sporadic DAG task releases a sequence of DAG-jobs

Parameters of a DAG-task τ_i

- a directed acyclic graph $G_i = (V_i, E_i)$,
- a deadline D_i
- a minimum inter-arrival time T_i

System of sporadic DAG-tasks

A task set S , $S = \{\tau_1, \tau_2 \dots \tau_n\}$, of DAG tasks τ_i defines a system of sporadic DAG tasks [Baruah, Bonifaci,MS, Stougie, Wiese 2012]

A task set S is said to be feasible upon a specified platform if a valid schedule exists on that platform for every collection of jobs that may be generated by the task set S

Feasibility problem Given a set S of DAG-tasks there exists a feasible schedule for all possible release sequences of DAG-jobs in S ?

Scheduling algorithm and performance metric

Given a scheduling algorithm ALG, **a task set is said to be ALG-schedulable** upon a specified platform if ALG meets all deadlines when scheduling any collection of jobs that may be generated by the task set upon that platform.

Speedup of a Scheduling Algorithm

Algorithm ALG has speed-up bound σ ($\sigma \geq 1$) if for each task set τ ,

- τ feasible on speed-1 platform \Rightarrow
 τ correctly scheduled by ALG on a speed- σ platform

Speedup of a Schedulability Test TA

Test TA has speedup bound σ if for each task set τ ,

- $(TA(\tau) = \text{NO}) \Rightarrow \tau$ infeasible on speed-1 platform,
- $(TA(\tau) = \text{YES}) \Rightarrow \tau$ schedulable by A on speed- σ platform

Scheduling algorithm and performance metric

(Preemptive) Global Earliest Deadline First (EDF)

- At each time step, schedule the m available jobs with earliest deadlines

Sporadic jobs

Jobs released on-line (no DAG) [Phillips Stein Torng 1997]

- EDF has speedup $2-1/m$

Sporadic (non conditional) DAG task system [Bonifaci MS, Stiller Wiese 2013]

- EDF has speedup $2-1/m$
- pseudopolynomial time test for EDF with speedup $2-1/m+\epsilon$
- the test is polynomial time when $D_i \leq T_i \forall i$.

Scheduling algorithm and performance metric

(Preemptive) Deadline Monotonic (DM)

- Order DAGs in non decreasing order of their relative deadline
- At each time step, schedule the m available jobs of DAGs with smaller priorities

Sporadic (non conditional) DAG task system

[Bonifaci,MS,Stiller, Wiese 2013]

- DM has speedup $3-1/m$
- pseudopolynomial time test for EDF with speedup $3-1/m+\epsilon$
- the test is polynomial time when $D_i \leq T_i \forall i$.

Scheduling algorithm and performance metric

Conditional DAG task set

A task set S , $S = \{\tau_1, \tau_2 \dots \tau_n\}$, of conditional DAG tasks τ_i defines a system of sporadic DAG tasks

Sporadic C-DAG task system [Baruah, Bonifaci MS 2015]

There exists an efficient transformation from conditional DAG tasks to unconditional DAG tasks that preserves the guarantees provided by the test in [Bonifaci,MS, Stiller, Wiese 2013]

Sporadic (conditional) DAG task system

- EDF has speedup $2 - 1/m$
- DM has speedup $3 - 1/m$
- pseudopolynomial time test for EDF (speedup $2 - 1/m + \epsilon$) and DM (speedup $3 - 1/m + \epsilon$)
- the tests are polynomial time when $D_i \leq T_i \forall i$.

Sporadic DAG task model

OPEN PROBLEMS

The PSPACE hardness for non conditional DAG task set uses a reduction to instances with a constant number of DAGs but with an unbounded no. of machines.

Open:

- What is the complexity of the feasibility problem in the case of a constant number of machines?

Sporadic task model is not the only considered model

Open:

- Which is the complexity for non sporadic task system (e.g. periodic task systems in which DAG-jobs of task τ_i are released exactly after T_i time units)?

Learning augmented algorithms

BEYOND WORST CASE ANALYSIS

Predictions (beyond worst case analysis)

- worst case analysis is too pessimistic in the evaluation of online algorithms
- machine learning techniques can provide good predictions on future requests

Challenge: design learning-augmented algorithms with a performance that

- is close to the best off-line algorithms when given accurate predictions (**consistency**)
- It is not (much) worse than the best off-line algorithm that has no access to predictions (**robustness**)

Predictions: sporadic DAG task

Recurrent DAGs: soft deadlines vs hard deadlines

- **Hard deadlines:** all jobs released by a task system meet their deadlines
- **Soft deadlines:** we might tolerate some deadline miss
- Hard deadlines are often too pessimistic

Soft deadlines possible objectives:

- Compute probability of deadline miss
- Compute probability of consecutive deadlines misses
- Reduce consecutive deadlines misses
- Bound the maximum tardiness of a job

Predictions: sporadic DAG task

Predictions with soft deadlines

Sporadic DAG task:

- worst case execution time of a job is not deterministic; it is heavily influenced by the concurrent execution of other jobs (memory access, resource reservation, time separation...)

Sporadic C-DAG:

- outcome of conditional branches; branch outcomes differ (most frequent case and the less frequent one), also branch outcome is influenced by the result of previous / concurrent programs

Predicting conditional branches

Predicting the outcome of conditional branches

[Ueter Chen et al 2021]:

- Assume that probabilities of outcome of branches are known; probabilities are **independent**
- **pC-DAG: a probabilistic Conditional DAG**

They are interested in showing under which conditions the probability of k consecutive deadline misses of a pC-DAG is no more than a user specified upper bound

Predicting conditional branches

Predicting the outcome of conditional branches

- [Ueter Chen et al 2021]: Assume that probabilities of outcome of branches are known; probabilities are **independent**
- pC-DAG: a probabilistic Conditional DAG

Open:

- **remove assumption that probabilities are independent**
- **use predictions to optimize algorithms (e.g. priorities in list scheduling) and compute robustness bounds**

Today's challenge

Today's challenge

- **Complex heterogenous hardware and complex graph-based workload**
- **The HPC-DAG model**

Today's challenge

We witness sophisticated autonomous features (e.g autonomous driving) in the realization of safety critical applications

- **Complex graph-based workload:**
AI algorithms realize capabilities (e.g. visual perception, decision making) in safety critical applications
- **Complex heterogeneous hardware:**
These algorithms often require the usage of hardware accelerator (graphics processing units, deep learning etc.)
- **Integration of separately developed components:**
Multiple functions are integrated on a common multicore high performance computing platform (to reduce cost and energy)

Today's challenge

We witness sophisticated autonomous features (e.g autonomous driving) in the realization of safety critical applications

New features

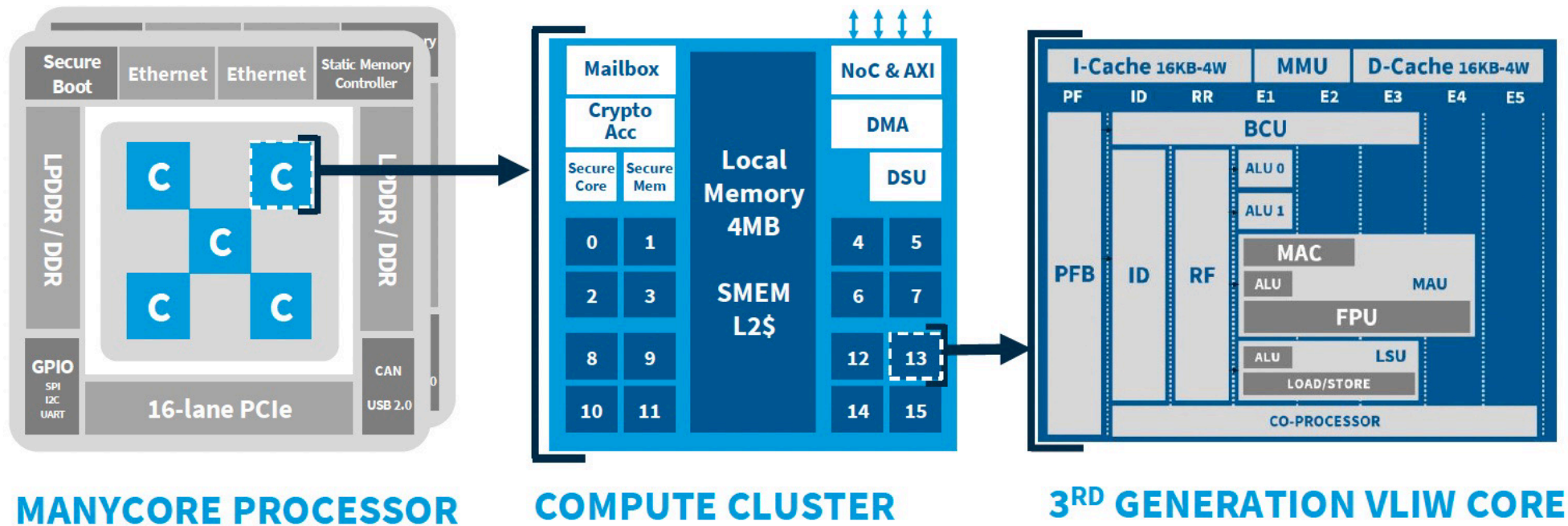
- Complex graph-based workloads
- Complex heterogeneous hardware
- Need to integrate separately developed components

These factors create a new challenge for certification

We must

- analyse algorithms that include above features
- design new scheduling algorithms

Cluster architectures



Kalray MPPA3-80 Coolidge: 80 processors, 5 clusters (16 proc. per cluster), a SRAM per cluster (16 modules one per CPU); Network On Chip communication architecture

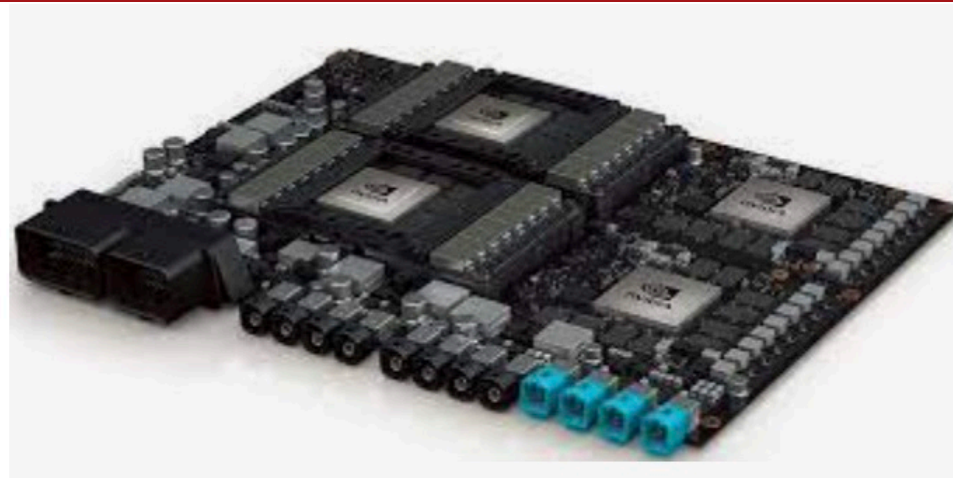
Allocation of nodes to cluster to reduce memory contention among processors allocated to different clusters

Problem: allocation of nodes to cluster

Heterogenous architectures

COMPLEX HETEROGENOUS HARDWARE

integration of different engines
in a single platform



NVIDIA Orin pegasus platform (2017) 16 engines: 8 CPU, 2 dGPU (discrete GPU), 2 iGPU (integrated GPU), 2 DLA (for deep learning inference), 2 PVA (Programmable Vision Accelerator) total 320 TOPS int8

A piece of code (a node of the DAG) can be executed on different engines (with different performance)

- HP (GPU, DLA, PVA) engines are scarce wrt to CPU
- HP engines have large set up time ==> non preemptable

Problem: allocation of nodes to CPU; consider preemption

Today's challenge

New computer architectures pose a number of difficulties in real time systems [Voronov Tanga et al 2021]

- **Graph based workload**
 - Presence of cycles in the graph
 - Dependencies among different instances (e.g. analysis to decide the motion requires info on previous frame)
- **Allocation of DAG nodes to heterogenous engines**
- **Access to Hardware accelerators (nonpreemptive)**
- **Component isolation**
- **Concurrent access to memory**

Today's challenge

Concurrent access to memory is critical in many-core architectures

- [Koike et al 2020] ILP A time-triggered schedule coordinates access to shared resources (global banks) so that the read and write phases do not overlap propose an Integer Linear Programming (ILP) to reduce task switching overhead.
- [Bathie et al. 2020] [Marchal et al. 2018] ILP to compute maximum size of memory requirement during execution with the goal to bound the maximum amount of memory that may be needed by any schedule to execute the DAG.

Concurrent access is not the only issue

We need a more general model

The HPC-DAG task model

HPC-DAG (Heterogenous Parallel Conditional)–DAG task model [Houssam-Eddine, Capodieci, Cavicchioli et al.2021]

Heterogenous architecture:

- a node x of the C-DAG can be executed on different engines (with different performance)
- $\text{tag}(x)$ = list of machines eligible for executing x

The number of possible assignments of nodes to machine is too large

- Many combinations can be (heuristically) reduced

Two steps

1. Preprocessing of the program to obtain a **Specification graph G** with a limited number of alternatives
2. Schedule **G** on the given platform

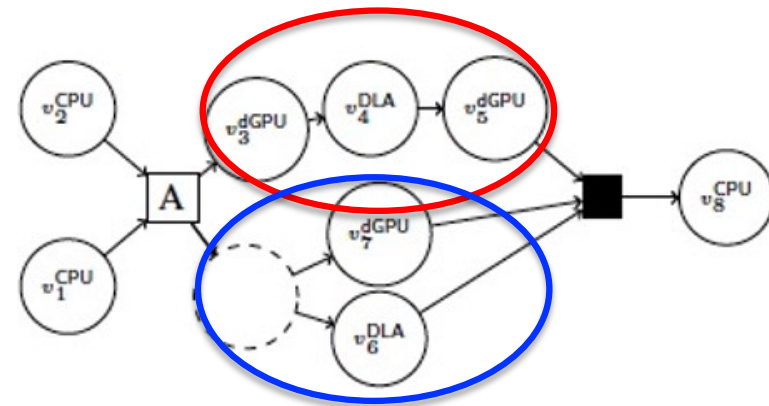
The HPC-DAG model

Specification graph G

A specifies two alternatives

Alternative 1

Alternative 2



Specification graph

G has both conditional branches and alternative nodes

- An **alternative node** describes **off-line alternative implementations** of a subdag, each one having a different parallelism
- Alternative executions can be also expressed using conditional nodes for expressing alternative options to be chosen online based on runtime conditions (more complicated)


HPC-DAG task model: HPC-DAG jobs are recurrent

The HPC-DAG model

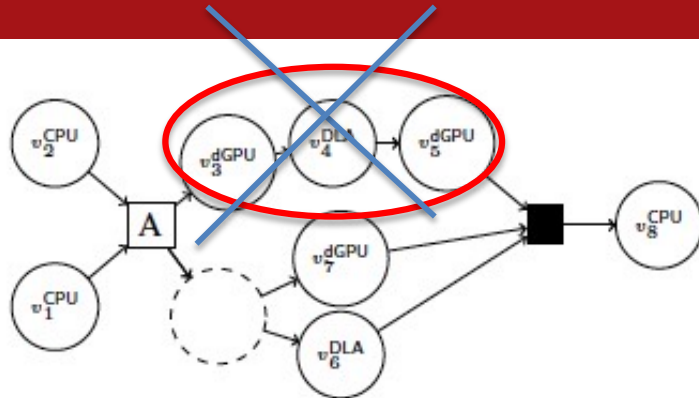
Specification graph G

- G has both conditional branches and alternative nodes
- An **alternative node** describes two possible choices of executing a piece of code (possibly different set of machines)
- Alternative nodes are either defined a priori or can represent scheduling

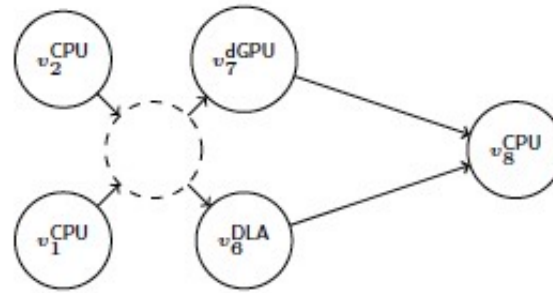
The scheduling of a **task specification HPC-DAG G** requires to

1. decide for each alternative node which choice should be executed  obtaining a **concrete task C-DAG G'**
2. assign to each node x of G' an engine (eligible for executing x)
3. schedule the restricted processor G'

The HPC-DAG model



Specification graph



Concrete C-DAG

Concrete task:

- a node x of the DAG can be executed on different engines (with different performance)
- $\text{tag}(x)$ = list of machines eligible for executing x
- the HPC DAG model has conditional branches and alternative branches
- an alternative node describes two possible choices any choice is ok

The HPC-DAG model

2. Schedule the specification HPC-DAG

High complexity

- The large number of combinations (alternative nodes, conditional nodes, allocation decisions) gives a too large ILP (number of constraints / variables) Preliminary results deal with very small instances
- Use of heuristics (first assign nodes to machine, then use list scheduling) is complicated by the many conflicting constraints

Open

- **under which condition we can obtain a tractable ILP?**
- **design and analysis of on-line heuristics (that exploit outcome of branches)**

Many thanks to my co-authors



Sanjoy Baruah



Vincenzo Bonifaci



Nicole Megow



Jens Schlöter



Martin Skutella



Sebastian Stiller



Leen Stougie



Andreas Wiese

THANK YOU - QUESTIONS?