How to eliminate non-positive circuits in periodic scheduling: a proactive strategy based on shortest path equations

S. Touati and S. Briais and K. Deschinkel

RAIRO 2013

Plan

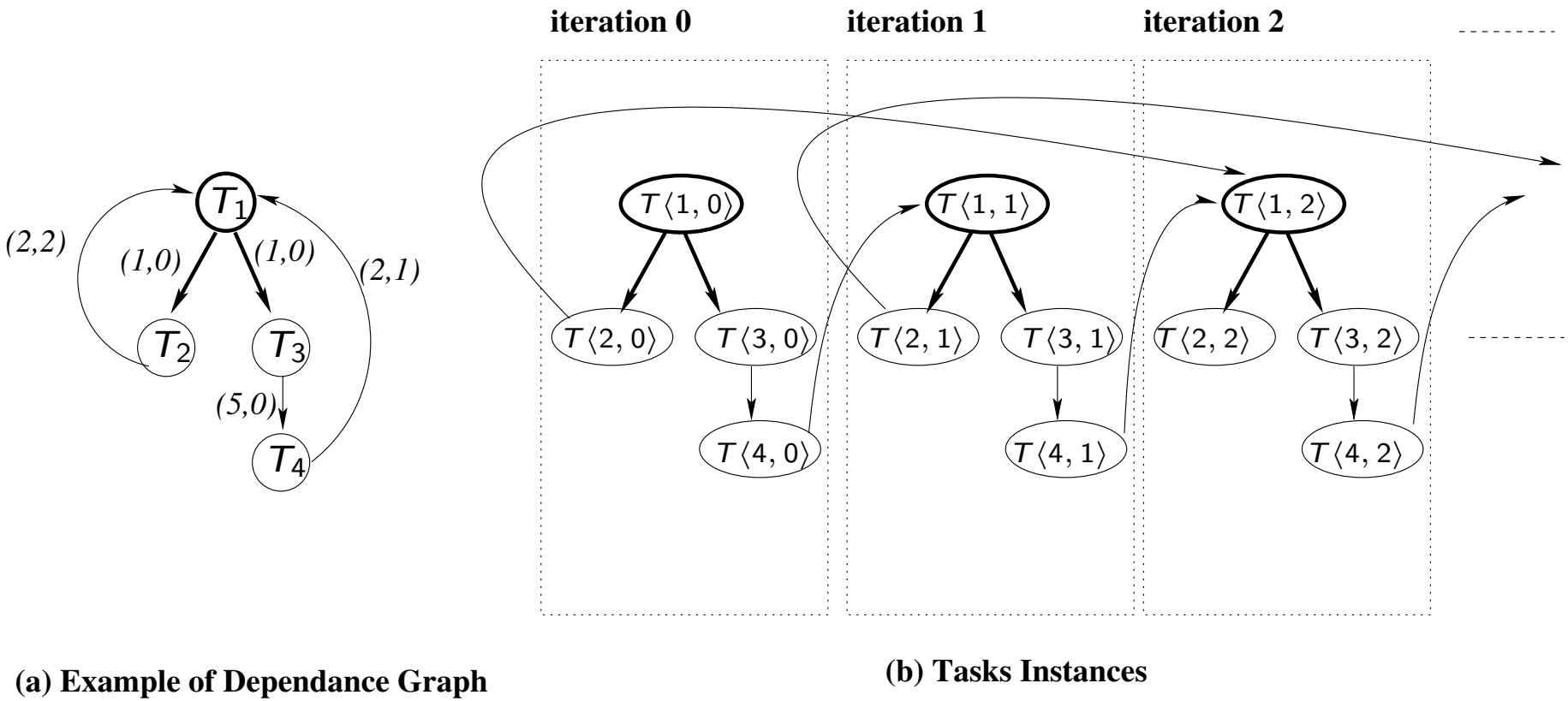
Plan

Framework

- Periodic scheduling of repetetive tasks;
- Software pipelining, instruction level parallelism.
- Somes tasks create a result to be stored inside a storage location/unit (warehouse, box, memory, processor, registers, etc.);
- The result of a task can be consumed by many other tasks;

Plan

Example of DDG

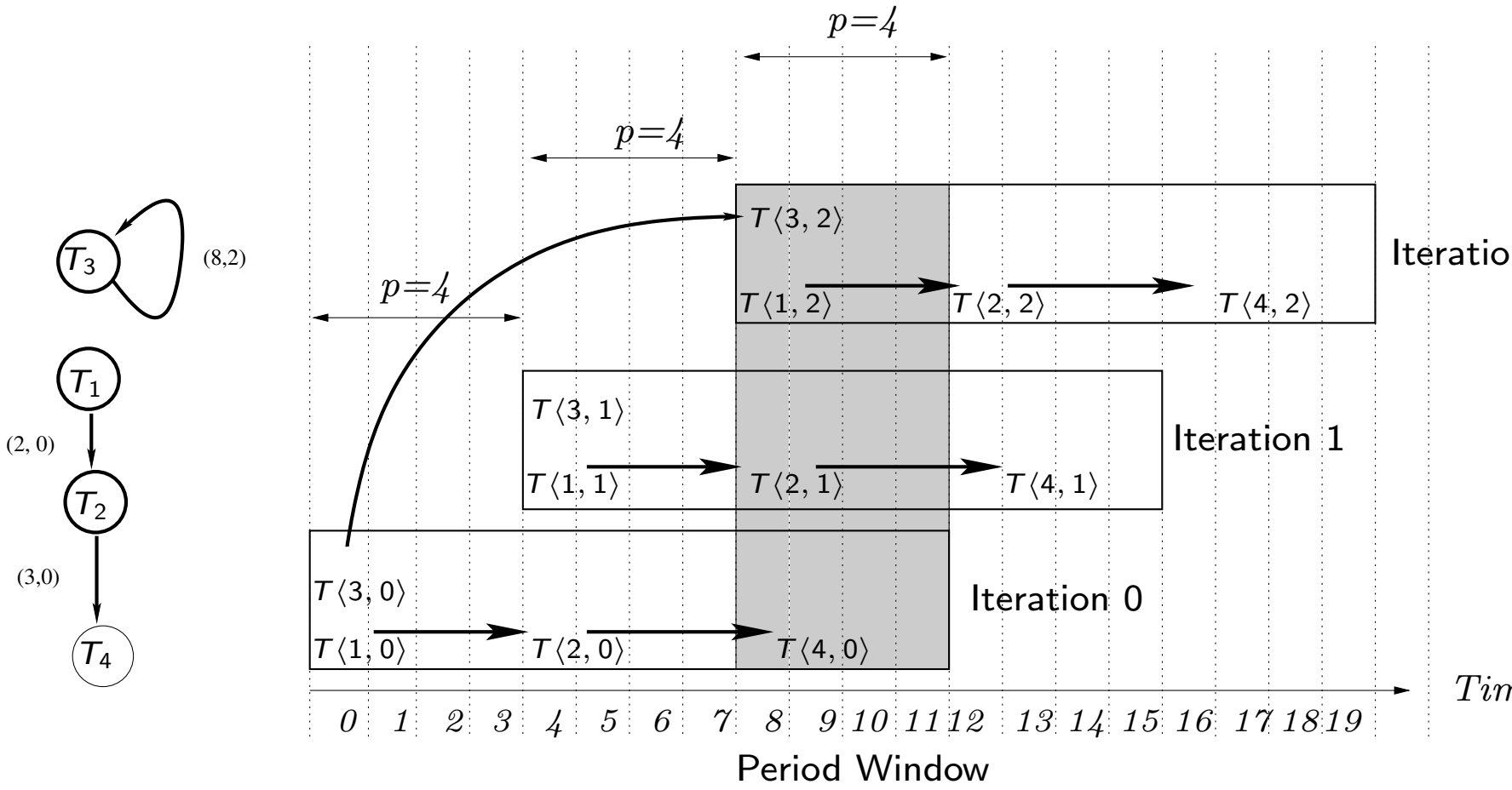


(a) Example of Dependance Graph

(b) Tasks Instances

Figure: Example of Data Dependence Graphs with Recurrent Tasks

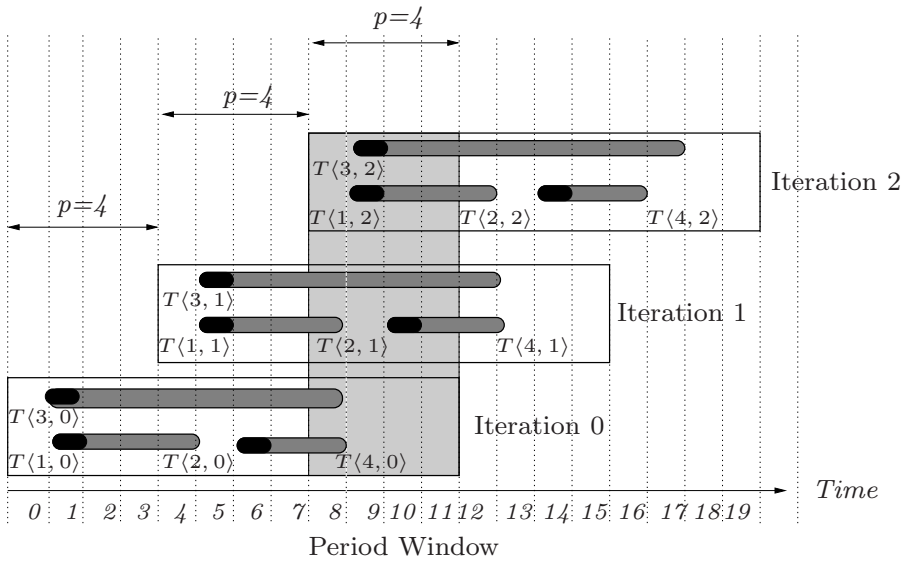
Example



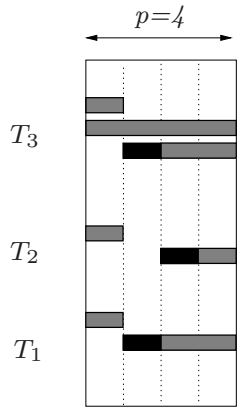
(a) Another dependance graph

(b) Example of Periodic Schedule

Circular lifetime intervals



(a) Periodic Lifetime Intervals



■ indicates the production of the task result

(b) Lifetime Intervals inside the Period Window

Problem

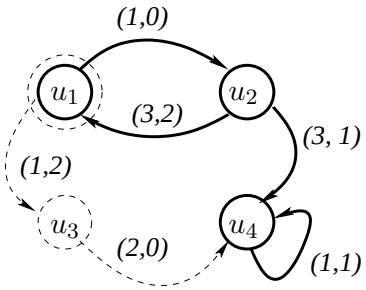
Problem (Storage Requirement Minimisation)

Given a desired cyclic scheduling period, how to minimise the storage requirement?

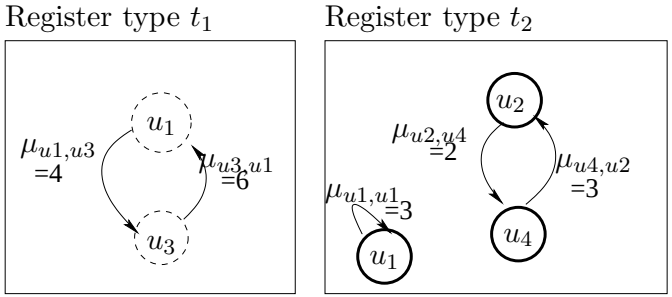
Problem (Period Minimisation)

Given a desired storage requirement, how to minimise the cyclic scheduling period?

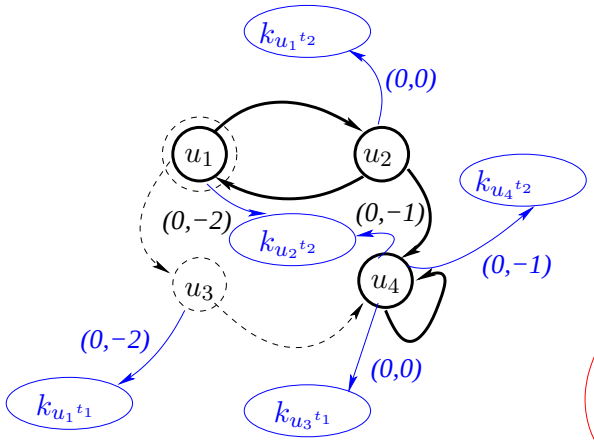
SIRA: Schedule Independent Register Allocation



(a) Initial DDG



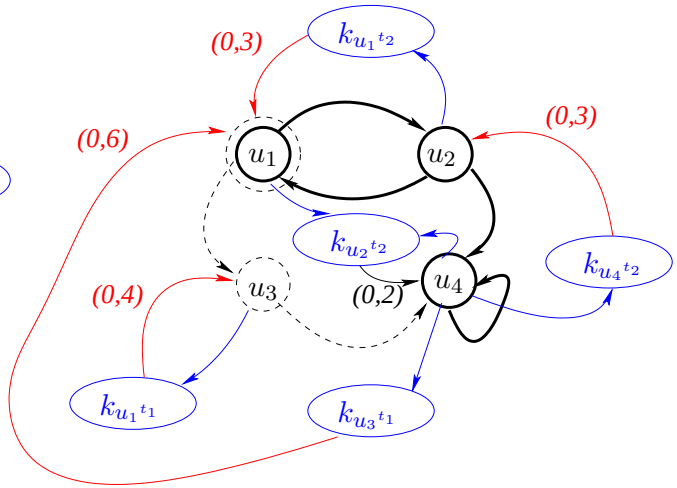
(b) Reuse Graphs for Register Types t1 and t2



(c) DDG with Killing Nodes

$V^k = \{k_{u^t} | u \in V^{R,t}\}$

$E^k = \{(v, k_{u^t}) | v \in Cons(u^t)\}$



(d) Preconditioned DDG after Applying SIRA

$E^{\mu} = \{(k_{u^t}, v) | (u, v) \in E^{\text{reuse},t}\}$

Plan

Data Dependence Graph (DDG)

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{I-1}\}$.
- E is the set of arcs representing precedence constraints.
- Any arc $e = (T_i, T_j) \in E$ has a:
  - a latency $\delta(e) \in \mathbb{Z}$ in terms of processor clock cycles
  - a distance $\lambda(e) \in \mathbb{Z}$ in terms of number of loop iterations.

  There is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$
- $V^{R,t}$ is the set of tasks producing data to be stored into registers of type t.
- $E^{R,t}$ is the set of flow dependence arcs through registers of type t.

Data Dependence Graph (DDG)

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{I-1}\}$.
- E is the set of arcs representing precedence constraints.
- Any arc $e = (T_i, T_j) \in E$ has a:
  - a latency $\delta(e) \in \mathbb{Z}$ in terms of processor clock cycles
  - a distance $\lambda(e) \in \mathbb{Z}$ in terms of number of loop iterations.

  There is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$
- $V^{R,t}$ is the set of tasks producing data to be stored into registers of type t.
- $E^{R,t}$ is the set of flow dependence arcs through registers of type t.

Data Dependence Graph (DDG)

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{l-1}\}$.
- E is the set of arcs representing precedence constraints.
- Any arc $e = (T_i, T_j) \in E$ has a:
  - a latency $\delta(e) \in \mathbb{Z}$ in terms of processor clock cycles
  - a distance $\lambda(e) \in \mathbb{Z}$ in terms of number of loop iterations.

  There is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$
- $V^{R,t}$ is the set of tasks producing data to be stored into registers of type t.
- $E^{R,t}$ is the set of flow dependence arcs through registers of type t.

Data Dependence Graph (DDG)

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{I-1}\}$.
- E is the set of arcs representing precedence constraints.
- Any arc $e = (T_i, T_j) \in E$ has a:
  - a latency $\delta(e) \in \mathbb{Z}$ in terms of processor clock cycles
  - a distance $\lambda(e) \in \mathbb{Z}$ in terms of number of loop iterations.

  There is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$
- $V^{R,t}$ is the set of tasks producing data to be stored into registers of type t.
- $E^{R,t}$ is the set of flow dependence arcs through registers of type t.

Data Dependence Graph (DDG)

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{l-1}\}$.
- E is the set of arcs representing precedence constraints.
- Any arc $e = (T_i, T_j) \in E$ has a:
  - a latency $\delta(e) \in \mathbb{Z}$ in terms of processor clock cycles
  - a distance $\lambda(e) \in \mathbb{Z}$ in terms of number of loop iterations.

  There is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$
- $V^{R,t}$ is the set of tasks producing data to be stored into registers of type t.
- $E^{R,t}$ is the set of flow dependence arcs through registers of type t.

Periodic scheduling constraints

Given a period $II \in \mathbb{N}^+$

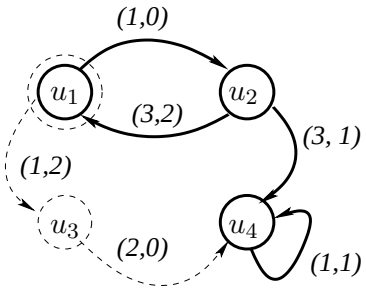
$$\forall e = (u, v) \in E : \sigma(u) + \delta(e) \leq \sigma(v) + \lambda(e) \times II$$

Other constraints: storage (registers), resources (functional units), code size, etc.

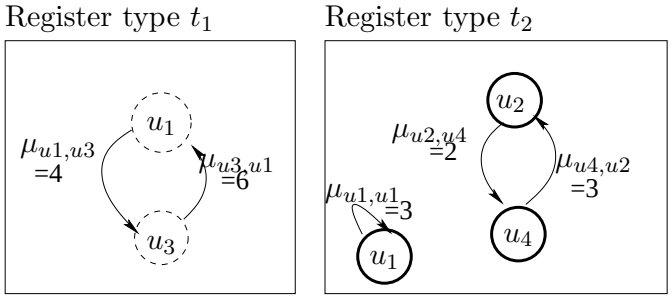
Problem of non positive circuits

- Some processor architectures exhibit micro-architectural properties (pipeline steps).
- These micro-architectural properties become visible to software (compilers).
- Solving periodic scheduling problems for storage minimization may lead to introducing non positive circuits inside DDG.
- AFAIK, scheduling theory for such graphs is not mature yet. Compilers are not ready yet.
- If no care is taken, compilation may fails while the compiled program is correct.

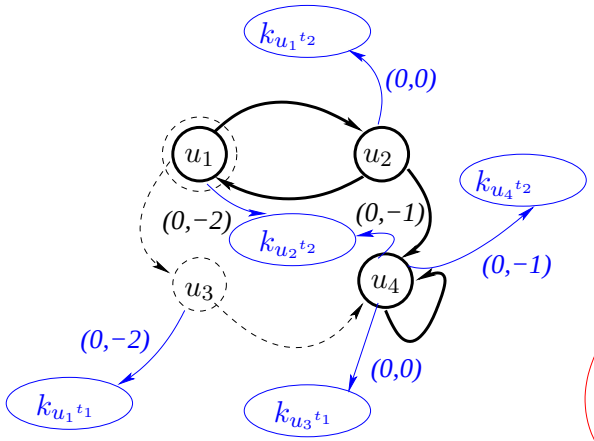
SIRA: Schedule Independent Register Allocation



(a) Initial DDG



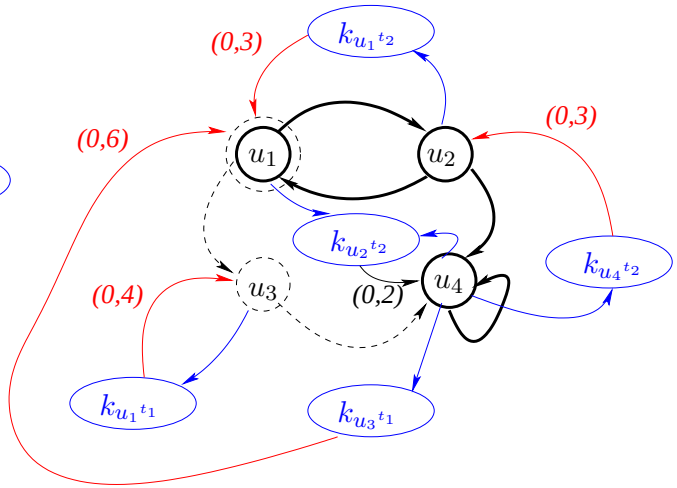
(b) Reuse Graphs for Register Types t1 and t2



(c) DDG with Killing Nodes

$V^k = \{k_{u^t} | u \in V^{R,t}\}$

$E^k = \{(v, k_{u^t}) | v \in Cons(u^t)\}$



(d) Preconditioned DDG after Applying SIRA

$E^{\mu} = \{(k_{u^t}, v) | (u, v) \in E^{\text{reuse},t}\}$

Relationship between the period II and the DDG circuits

Notations:

- $\lambda(C) = \sum_{e \in C} \lambda(e)$
- $\delta(C) = \sum_{e \in C} \delta(e)$

Let C^+ be the set of cycles such as $\lambda(C) > 0$ (resp. C^- pour $\lambda(C) < 0$ et C^0 pour $\lambda(C) = 0$). Then[1]:

$$\max_{C \in C^+} \frac{\delta(C)}{\lambda(C)} \leq II \leq \min_{C \in C^-} \frac{\delta(C)}{\lambda(C)}$$

[1] Alix Munier. *A graph-based analysis of the cyclic scheduling problem with time constraints: schedulability and periodicity of the earliest schedule*. Journal of Scheduling 2010.

Non positive circuits in DDG

A DDG G is schedulable with $II > 0$, then:

1. $\delta(C) \geq 0 \Longrightarrow \lambda(C) \geq 0$.
2. $\delta(C) \leq 0 \Longrightarrow \lambda(C)$ may be non-positive (either $\lambda(C) \leq 0$ or $\lambda(C) \geq 0$).

Problem (Storage optimisation)

Optimise storage with the condition that all produced circuits have $\lambda(C) > 0$.

Plan

Corolary[Briais and Touati 2010]

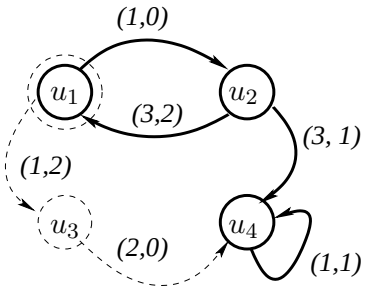
Corollary

Let $G = (V, E)$ be a directed graph and $w : E \to \mathbb{Z}$ a cost function. Then G has a cycle C with a cost $\sum_{e \in C} w(e) \leq 0$ iff the following linear program is unfeasible:

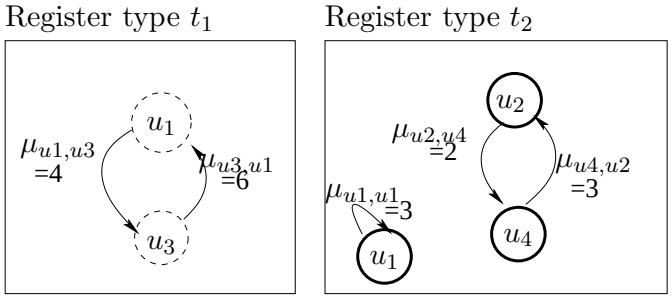
$$\forall e = (u, v) \in E, x_v - x_u \leq \|V\| \cdot w(e) - 1$$

Plan

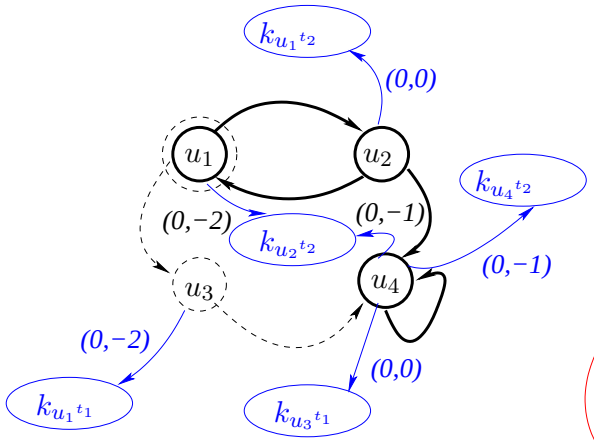
SIRA: Schedule Independent Register Allocation



(a) Initial DDG



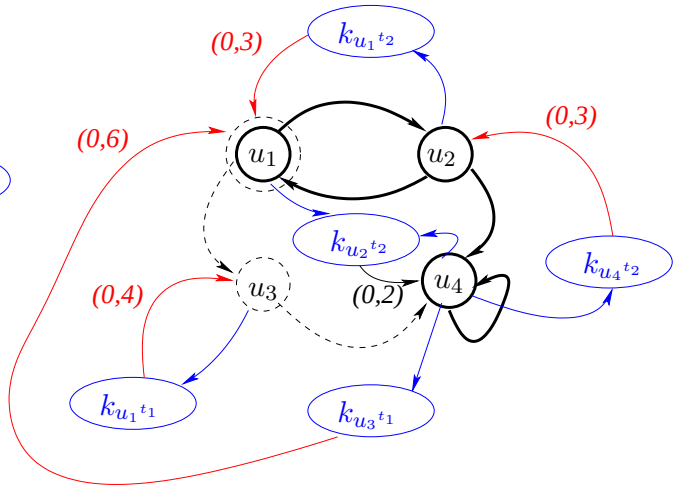
(b) Reuse Graphs for Register Types t1 and t2



(c) DDG with Killing Nodes

$V^k = \{k_{u^t} | u \in V^{R,t}\}$

$E^k = \{(v, k_{u^t}) | v \in Cons(u^t)\}$



(d) Preconditioned DDG after Applying SIRA

$E^{\mu} = \{(k_{u^t}, v) | (u, v) \in E^{\text{reuse},t}\}$

Linear program

Minimise

$$\sum_{t \in \mathcal{T}} \alpha_t \left(\sum_{(u,v) \in E^{\text{reuse},t}} \gamma^t(u,v) \right)$$

Subject to:

$\forall e \in E \cup E^k$, $\quad x_{tgt(e)} - x_{src(e)} \leq \|\mathcal{V}\| \cdot \lambda(e) - 1$

$\forall t \in \mathcal{T}, \forall e = (k_u^t, v) \in E^{\mu,t}$, $\quad x_{tgt(e)} - x_{src(e)} - \|\mathcal{V}\| \cdot \gamma^t(u,v) \leq \|\mathcal{V}\| \cdot \widehat{\mu}^t_{(i-1)}(u,v) -$

$\forall t \in \mathcal{T}, \forall (u,v) \in E^{\text{reuse},t}_{(i-1)}$, $\quad \gamma^t(u,v) \geq \widehat{\mu^*}^t(u,v) - \widehat{\mu}^t_{(i-1)}(u,v)$

$\forall u \in \mathcal{V}$, $\quad x_u \in \mathbb{R}$

$\forall t \in \mathcal{T}, \forall (u,v) \in E^{\text{reuse},t}_{(i-1)}$, $\quad \gamma^t(u,v) \in \mathbb{R}$

Plan

Experimental setup: DDG sizes (number of nodes)

	MEDIAB.	SPEC2000	SPEC2006	FFMPEG
MIN	3	3	5	4
FST	10	12	16	18
MED	16	22	24	37
THD	28	28	30	111
MAX	212	163	212	783

Experimental setup: number of DDG per register type

Type	MEDIABENCH	SPEC2000	SPEC2006	FFMPEG
FP	313	317	87	47
GR	1592	3838	1274	2030
BR	1592	3841	1274	2030

The frequency of non-positive circuits detection

	SPEC2000	SPEC2006	MEDIABENCH	FFMPEG
Proportion	30.77%	28.16%	41.90%	92.21%

Name of the architecture	FP registers	GR registers	BR registers
Small architecture	32	32	4
Medium architecture	64	64	8
Large architecture	128	128	8

Three experimented strategies

1. UAL: simplify the loop model (put all latencies as ≥ 1)
2. Reactive strategy: be optimistic, if non positive circuit detected, prevent it by using UAL.
3. Proactive strategy: prevent the problem as explained in this talk

Median execution times of each strategy in seconds

Strategy	SPEC2000	SPEC2006	MEDIABENCH	FFMPEG
UAL	0.001	0.002	0.03	0.03
Reactive	0.002	0.002	0.003	0.07
Proactive	0.012	0.015	0.015	0.27

Table: Execution times of each heuristic

Conclusion on experiments

- If static compilation time is not very important (embedded systems), use proactive strategy.
- If compilation is an interactive process (workstation, development process), use a reactive strategy.
- If just in time compilation, use UAL.

Plan

Perspectives à la compilation optimisante bas niveau

- Importance d'une modélisation de boucle fidèle à la capacité architecturale d'un processeur
- Revoir l'ordonnancement acyclique par liste pour prendre en compte les latences négatives ou nulles de certains arcs d'un DAG.
- Proposer des méthodes d'ordonnancement d'instructions sous contraintes de ressource avec des circuits non positifs.

Discussion sur le modèle polyédrique pour les nids de boucles

- Modèle pour nids de boucles haut niveau, pas prévu initialement pour la compilation bas niveau.
- Contraintes de légalité pour tout ordonnancement: les latences de chaque tache ≥ 1.
- Le modèle polyédrique (et les problèmes d'ordonnancement) doivent considérer des latences négatives ou nulles.
- Ne devrait pas causer de pb pour la parallélisation automatique haut niveau
- Peut causer des problèmes pour l'ordonnancement sous contraintes de ressources.
- Modèle universel pour représenter les boucles, remplacerait le modèle des GDD.