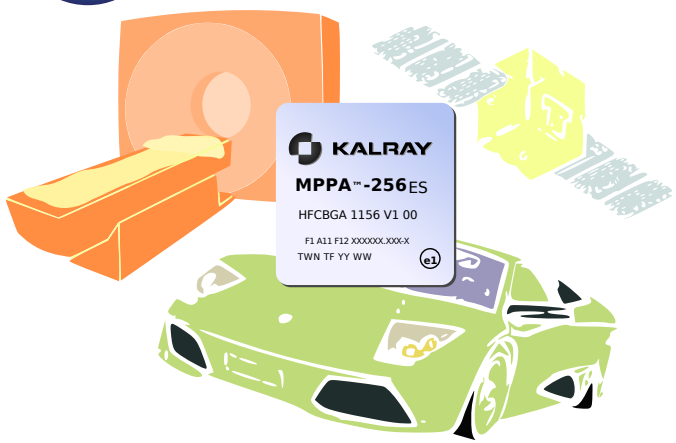
Analyse d'Applications Flot de Données pour la Compilation Multiprocesseur

Bruno Bodin

Soutenance de thèse, vendredi 20 décembre 2013

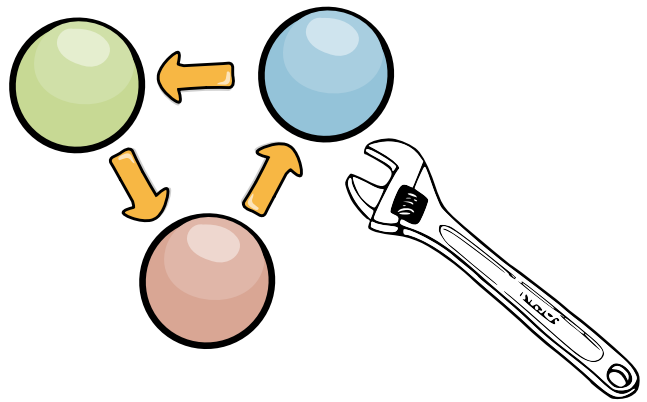
Introduction

KALRAY



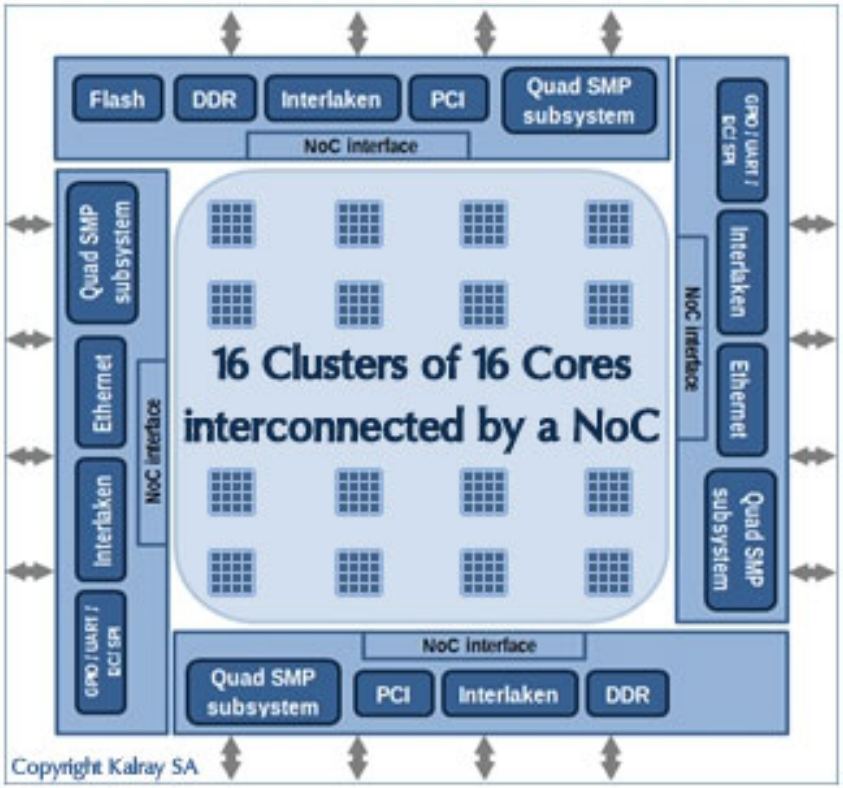
Une architecture multi-processeur

LIP6



Des outils d'analyse statique dataflow

Le processeur MPPA256



- 256 cœurs de calcul VLIW
- 16 clusters de 16 cœurs
- Faible consommation, 5 W
- Interface C/C++ POSIX
- Langage dataflow : SigmaC

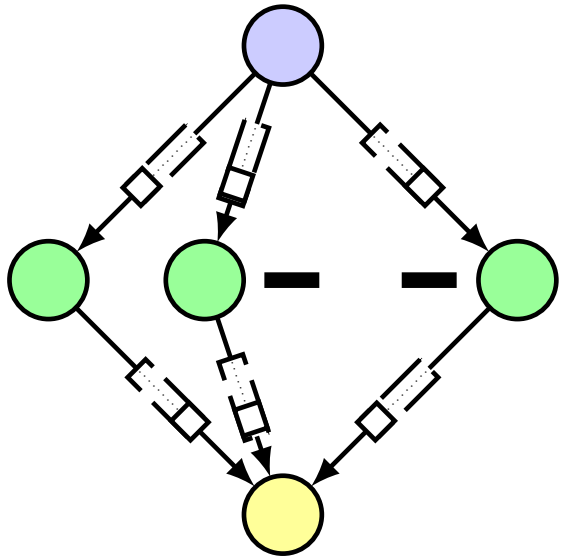
Le langage SigmaC [Goubier et al., 2011]

```
agent Reader(N) {
 interface {
  spec {/* N sorties */}}
 void start() {
  /* f => out */
 }
}
```

```
agent Writer(N) {
 interface {
  spec {/* N entrees */}}
 void start() {
  /* in => f */
 }
}
```

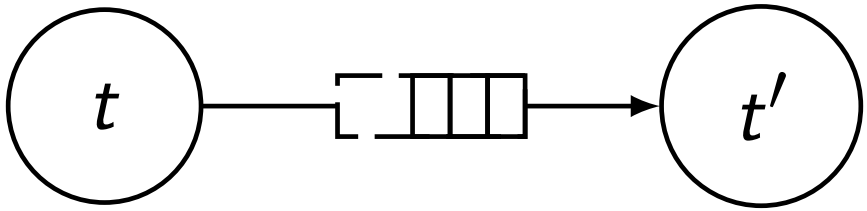
```
agent Worker {
 interface {
  spec {in[1];out[1]}}
  void start(in,out)) {
    /* in => out */
  }
}
```

```
subgraph root {
 map {
  agent inA = new Reader(N+1);
  agent outA = new Writer(N+1);
  agent w[N+1] = new Worker();
  for(int i=0;i<=N;i++){
   connect(inA.out[i],w[i].in);
   connect(w[i].out,outA.in[i]);
  }
 }
}
```



Qu'est-ce qu'un modèle dataflow ?

Réseaux de Kahn [Kahn, 1974]

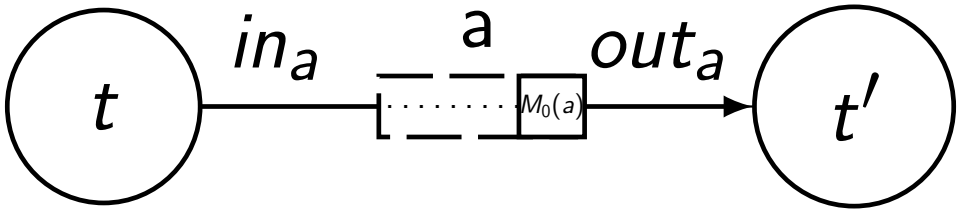


- Des processus communiquant au travers de canaux
- Les canaux sont des mémoires FIFO non bornées
- La lecture est bloquante
- Un modèle déterministe

Des problèmes d'analyse indécidables

Les réseaux de Kahn ne nous permettent pas de déterminer la quantité de mémoire nécessaire à leur exécution [Buck et Lee, 1993].

Synchronous Dataflow [Lee et Messerschmitt, 1987]

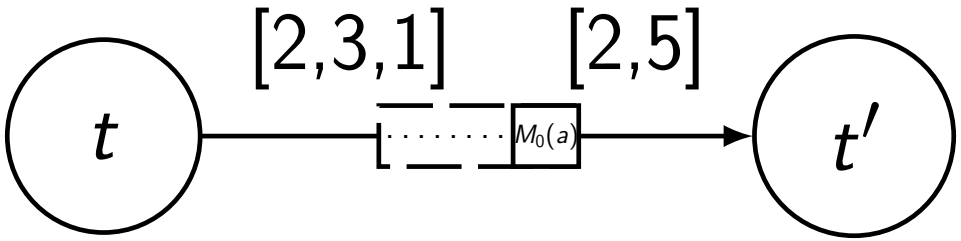


- in_a désigne la quantité de jetons produits
- out_a est la quantité de jetons consommés
- La quantité initiale de jetons dans le buffer est notée $M_0(a)$

Un modèle **statique**

Il est possible d'en étudier le comportement *a priori*, sans exécuter le programme modélisé : c'est un modèle statique. Plusieurs problèmes fondamentaux deviennent alors décidables.

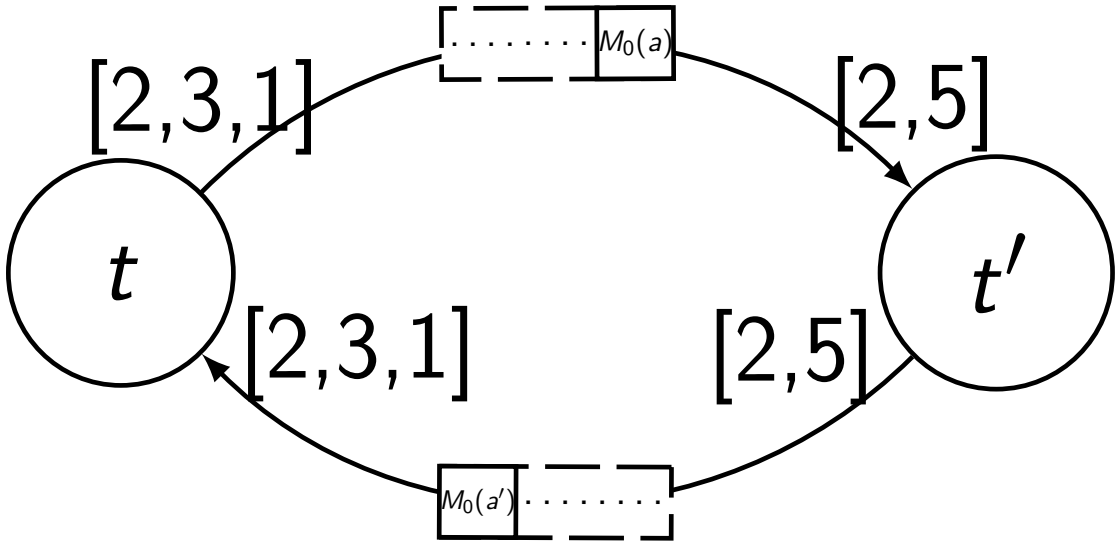
Cyclo-Static Dataflow [Bilsen et al., 1995]



- Une tâche se décompose en $\varphi(t)$ phases
- $in_a(k)$, la quantité produite par t_k la k^{e} phase de t
- $out_a(k')$, la quantité consommée par la k'^{e} phase de t'
- En somme, un modèle plus précis que les SDFG

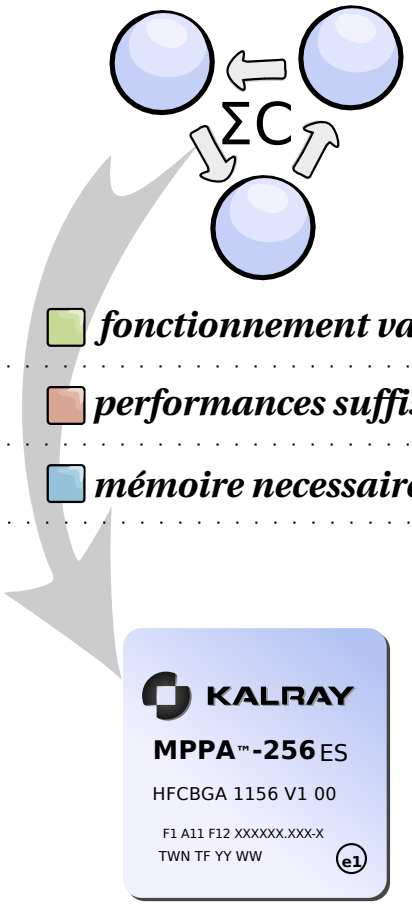
Modélisation des buffers

Un buffer a borné par une taille $B(a)$ se modélise par un arc retour a' (*feedback arc*) :

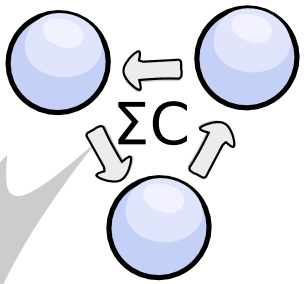


$$B(a) = M_0(a) + M_0(a')$$

La compilation Dataflow



La compilation Dataflow



2011

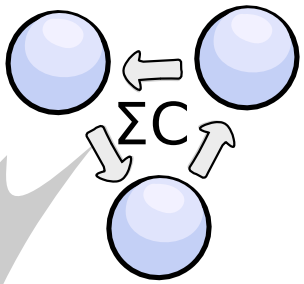
AccessCore

fonctionnement valide ?	**min**
performances suffisantes ?	**-**
mémoire necessaire ?	**min*** **-**

Pas d'évaluation du débit

Pas de dimension mémoire avec containte de performance

La compilation Dataflow



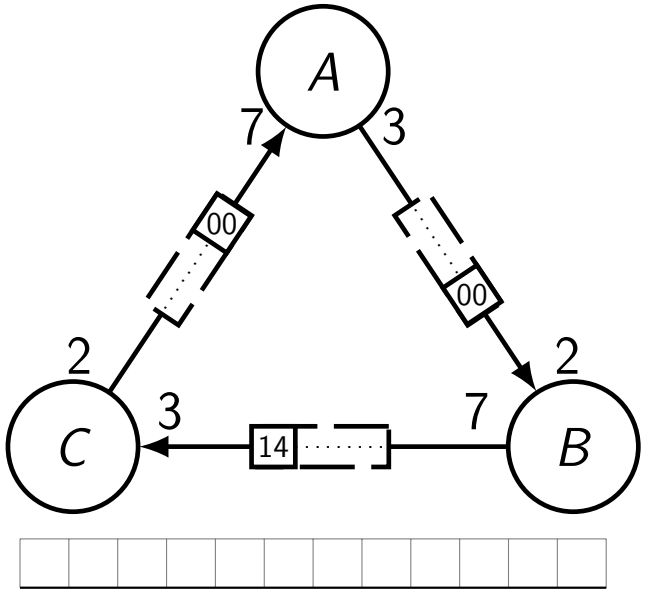
	2011 AccessCore	2011 État de l'art
fonctionnement valide ?	min	min
performances suffisantes ?	-	heure
mémoire necessaire ?	min* -	ms* ms*

Pas d'évaluation du débit

Pas de dimension mémoire avec containte de performance

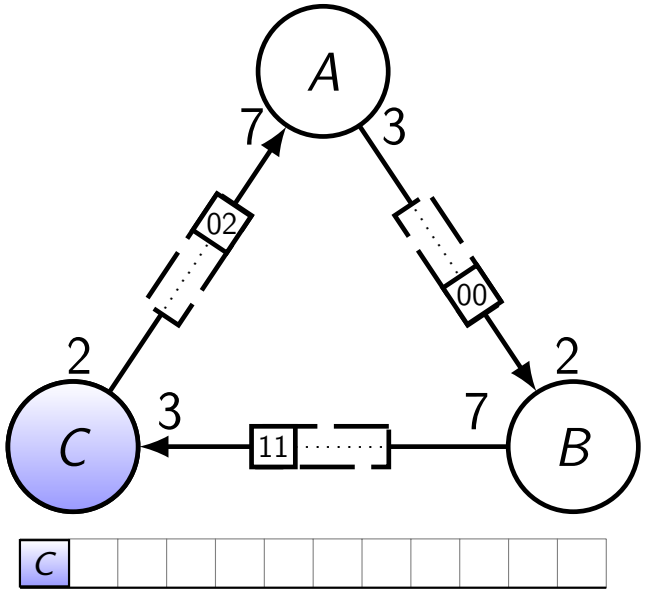
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



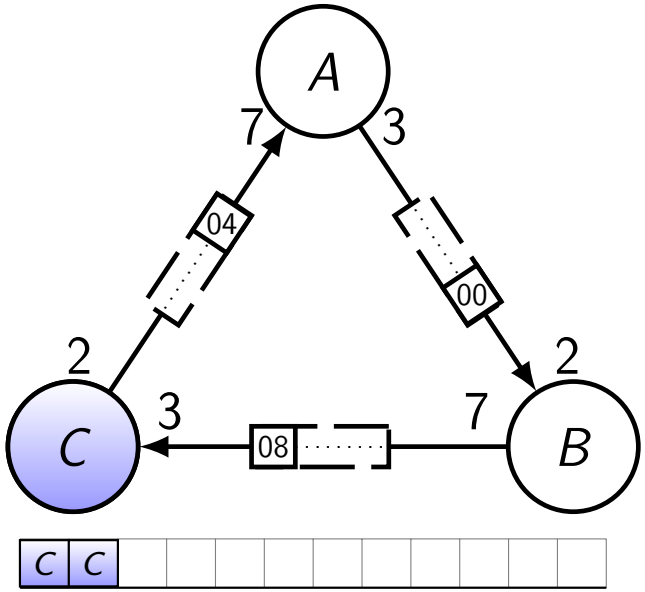
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



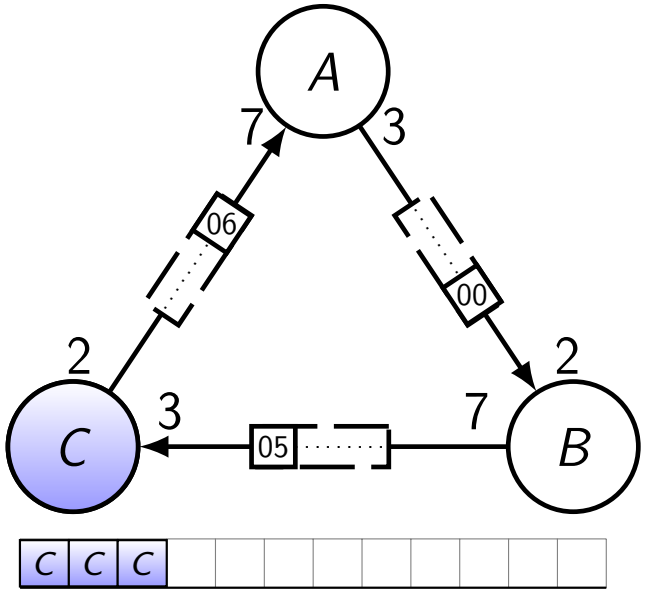
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



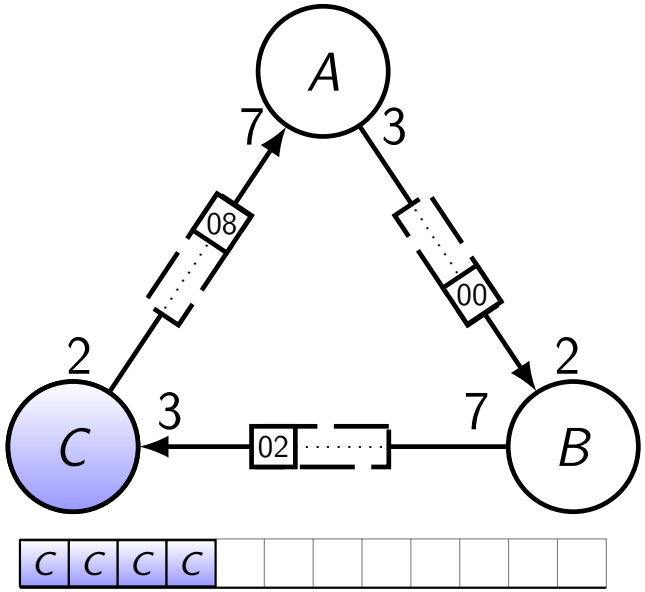
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



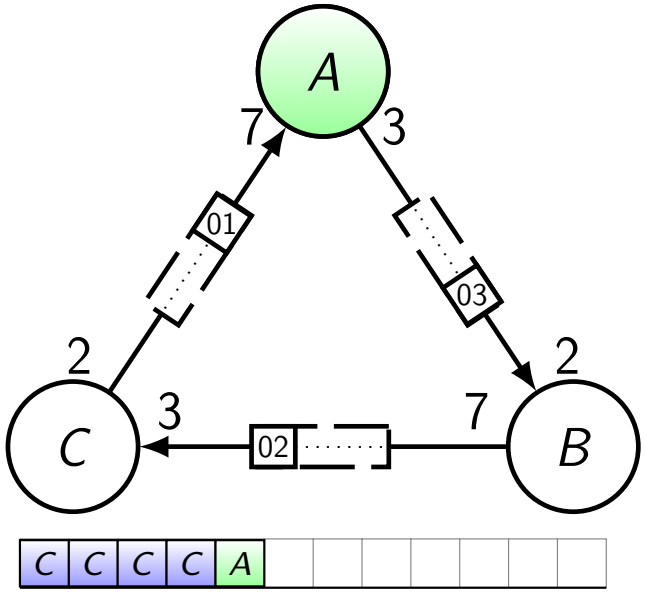
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



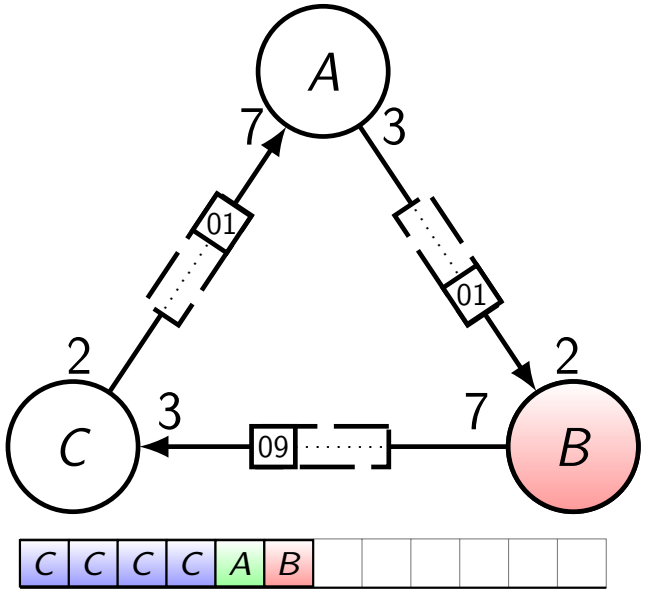
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



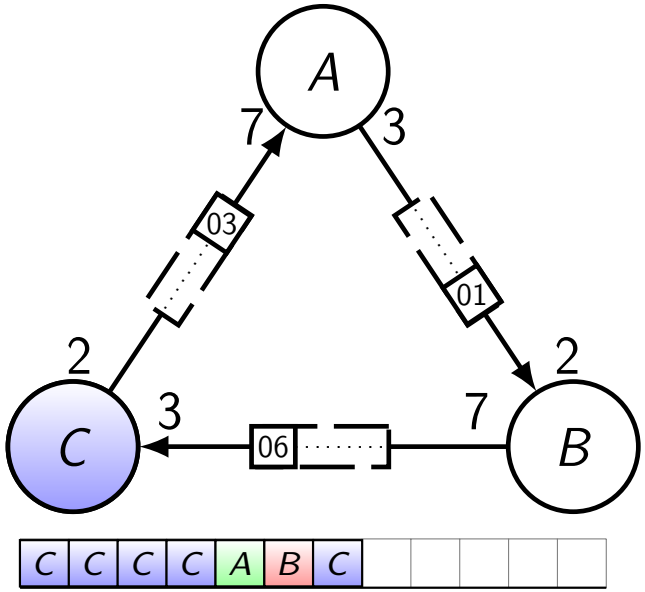
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



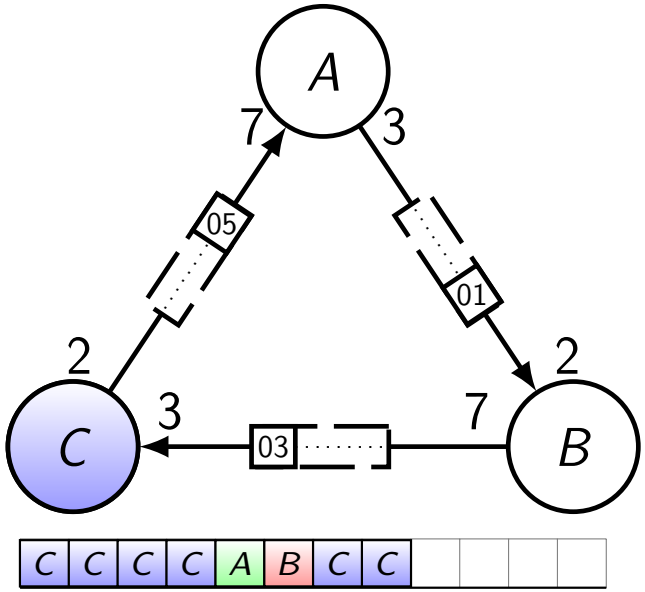
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



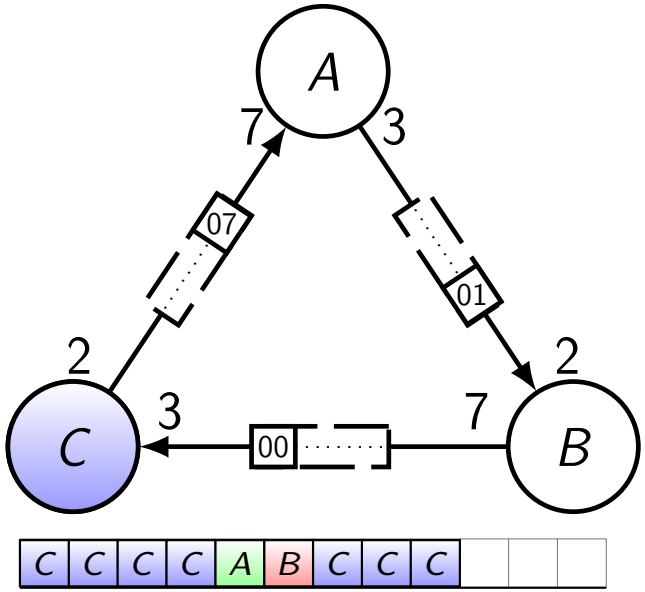
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



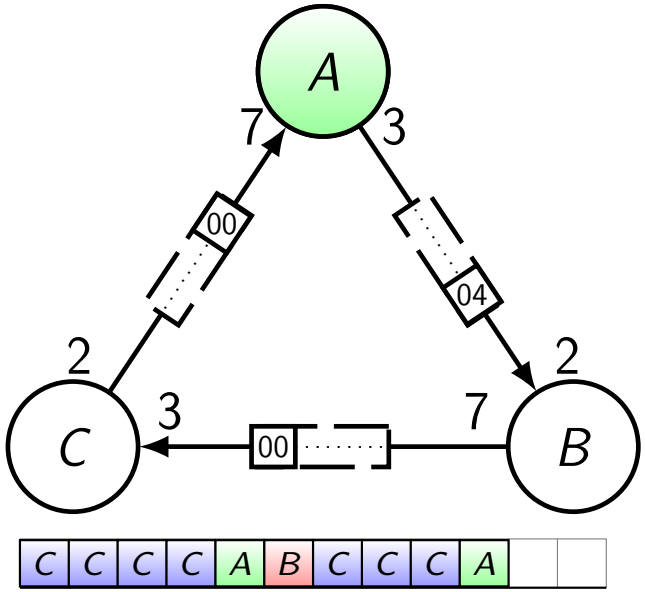
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



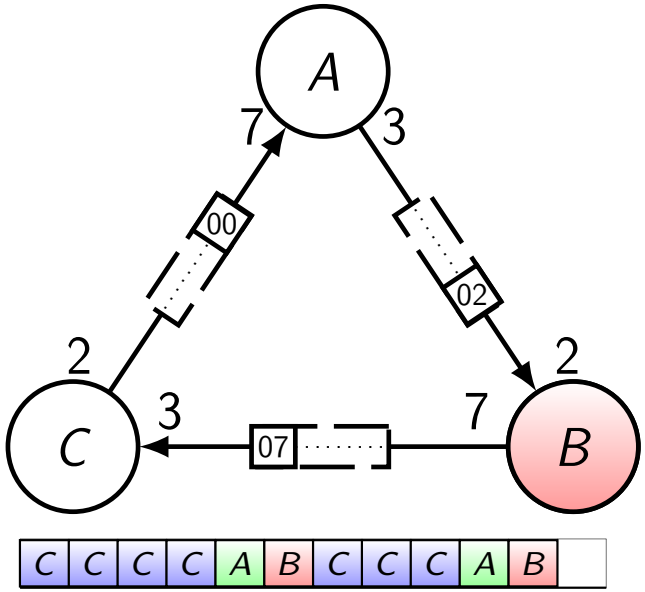
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



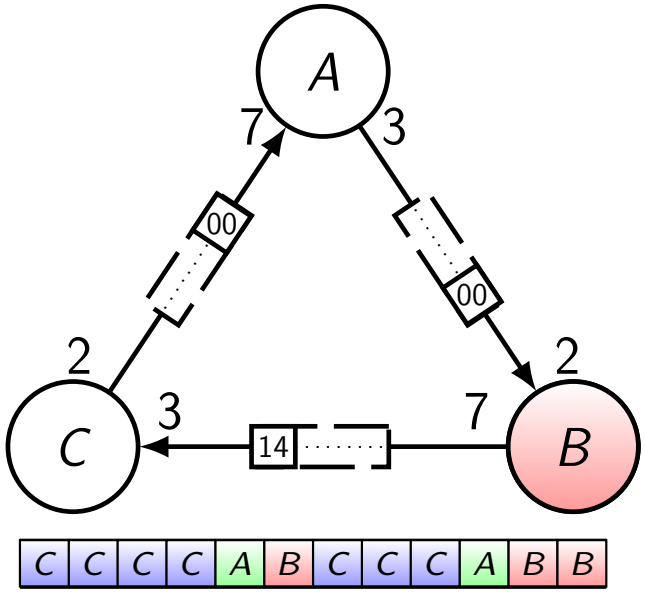
Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



Définition de la vivacité

- Capacité à s'exécuter sans blocage
- Comment vérifier la vivacité ?
  - Construire un ordonnancement
  - Cette technique est exponentielle



Le vecteur de répétition : $N_A^{\mathcal{G}} = 2,\ N_B^{\mathcal{G}} = 3,\ N_C^{\mathcal{G}} = 7$

- Nombre d'exécutions nécessaires pour revenir à un état initial
- Se calcule en temps polynomial
- Des valeurs exponentielles

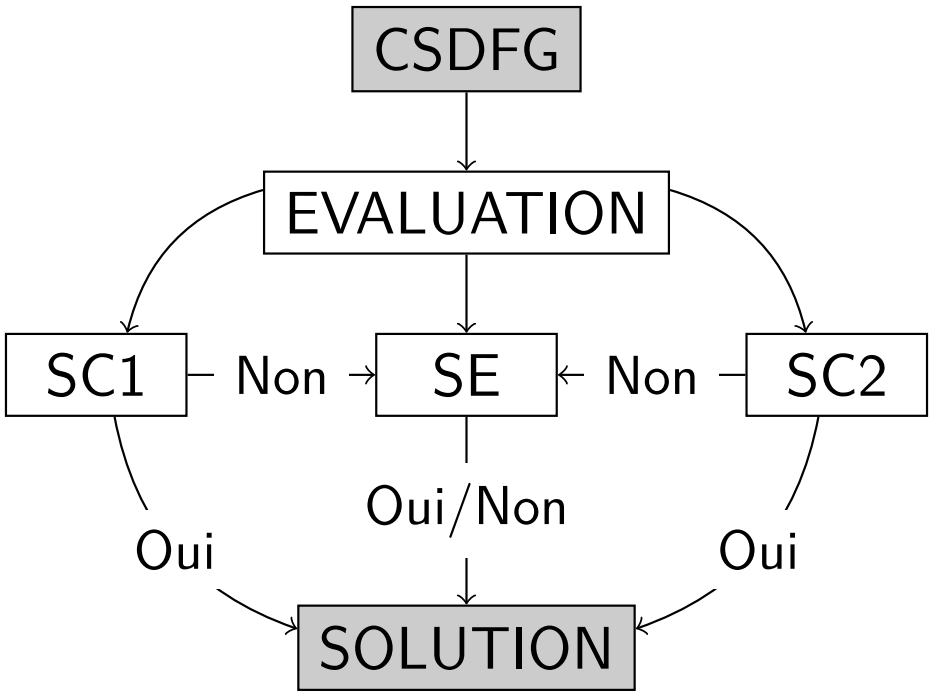
Méthodes Existantes

- Exécution symbolique [Anapalli et al., 2009]
  - Méthode exacte
  - Exponentielle
- Conditions SC1 et SC2 [Benazouz, 2012]
  - Conditions suffisantes uniquement
  - Sans algorithme pour les vérifier

Objectif : vérifier rapidement la vivacité d'un CSDFG

Algorithme de vérification de la vivacité

- Nous prouvons que CS1 et CS2 sont équivalentes
- Nous fournissons deux algorithmes polynomiaux pour les vérifier
- La complexité de CS1,CS2 et SE est connue
- On obtient l'algorithme suivant :



Jeux de test

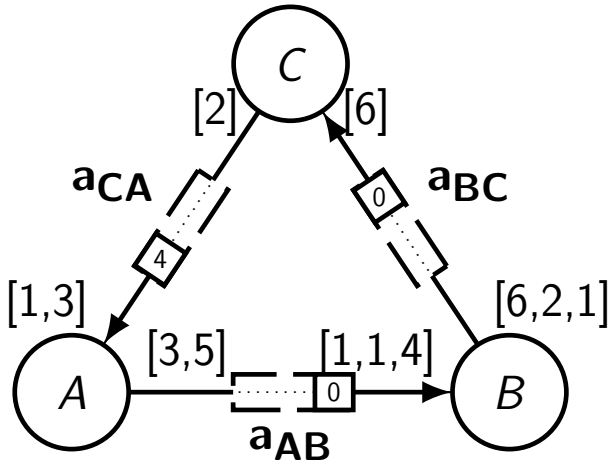
Application	Tâches	Buffers	Phases	Ordo. Minimal
BlackScholes	41	40	1560	79495
Echo	38	82	10332	784024
JPEG2000	240	703	2389497	1236486636
Pdetect	58	76	8664	1191979200
H264 Encoder	665	3128	55243608	120868020
autogen1	90	617	2651866	1361124
autogen2	70	473	1500829	8869482600
autogen3	154	671	1973411	11005386480
autogen4	2426	2900	10129700	1599780
autogen5	2767	4894	42851864	3614142

disponibles en ligne : `http ://www-soc.lip6.fr/~bodin/`

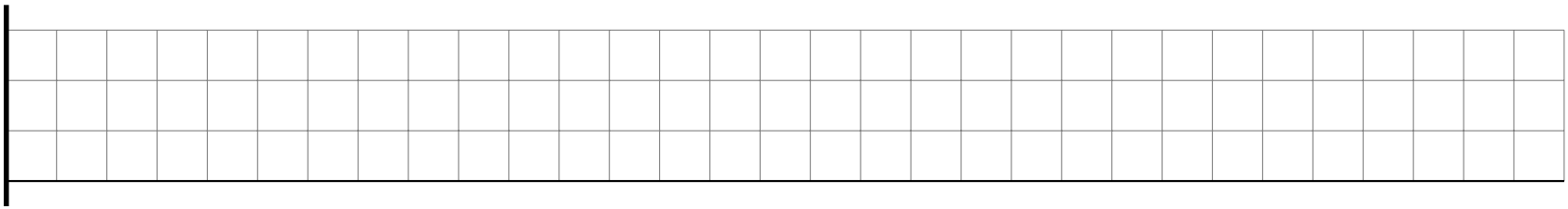
Résultats

Application	SC1		SC2		SE	
BlackScholes	2.10^5	14ms	$\boxed{1.10^3}$	0ms	2.10^5	1ms
JPEG2000	8.10^6	114ms	$\boxed{2.10^6}$	18ms	4.10^7	113ms
Echo	$\boxed{6.10^3}$	1ms	1.10^4	0ms	6.10^6	95ms
Pdetect	1.10^9	2500ms	$\boxed{8.10^3}$	4ms	6.10^5	5ms
H264	3.10^7	504ms	5.10^7	936ms	$\boxed{9.10^6}$	114ms
autogen1	$\boxed{1.10^5}$	13ms	2.10^6	55ms	3.10^8	1544ms
autogen2	$\boxed{7.10^5}$	41ms	1.10^6	37ms	3.10^{10}	4min
autogen3	$\boxed{1.10^6}$	55ms	1.10^6	55ms	4.10^{11}	21min
autogen4	7.10^7	217ms	$\boxed{1.10^7}$	71ms	2.10^8	132ms
autogen5	5.10^7	787ms	$\boxed{4.10^7}$	708ms	3.10^9	1442ms

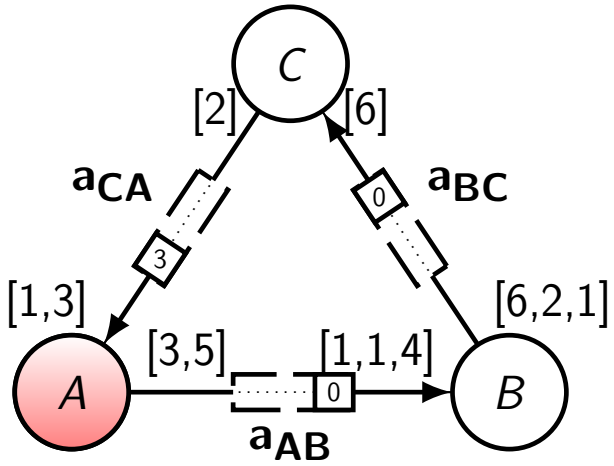
Ordonnancement au plus tôt



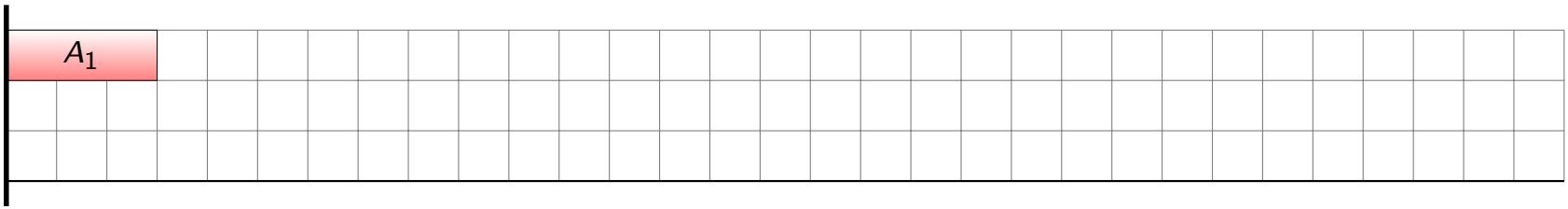
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



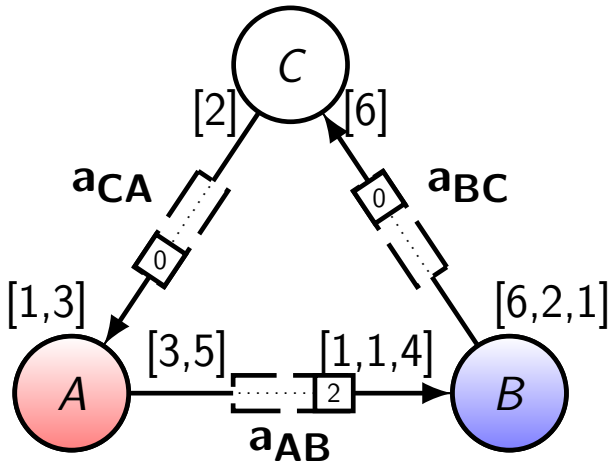
Ordonnancement au plus tôt



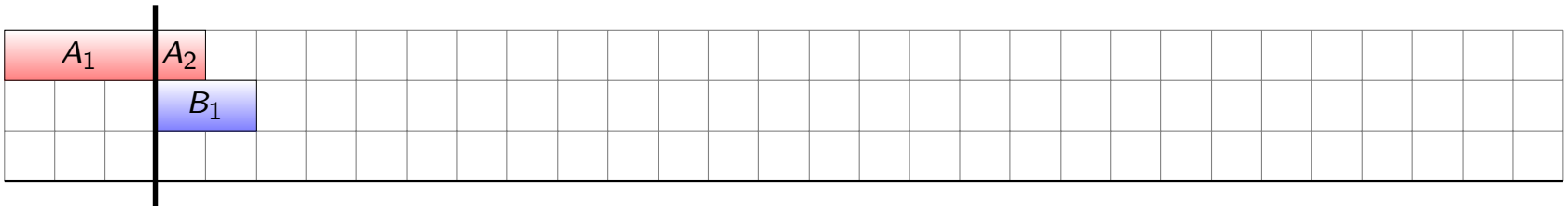
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



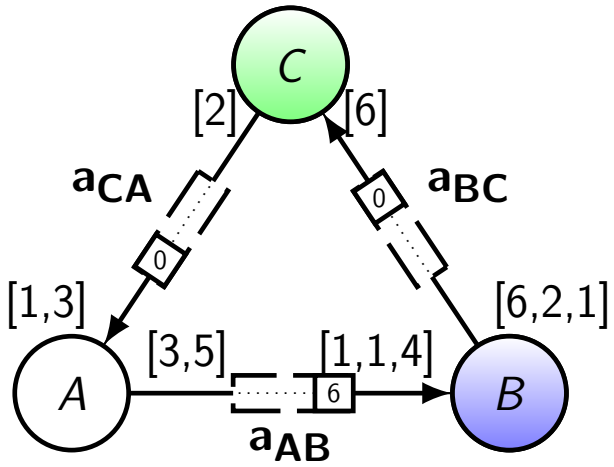
Ordonnancement au plus tôt



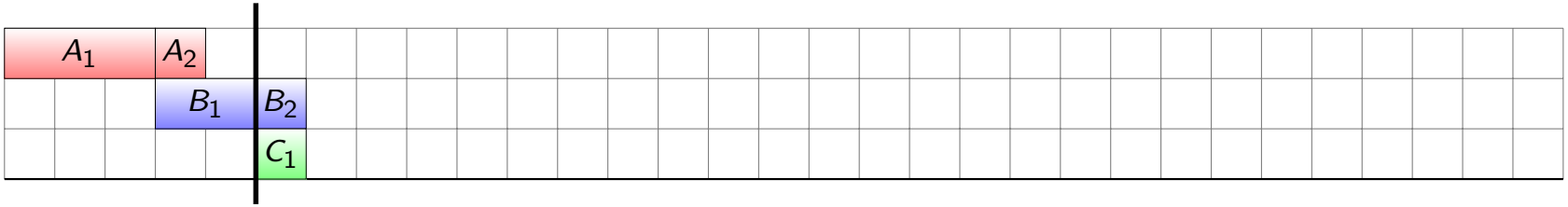
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



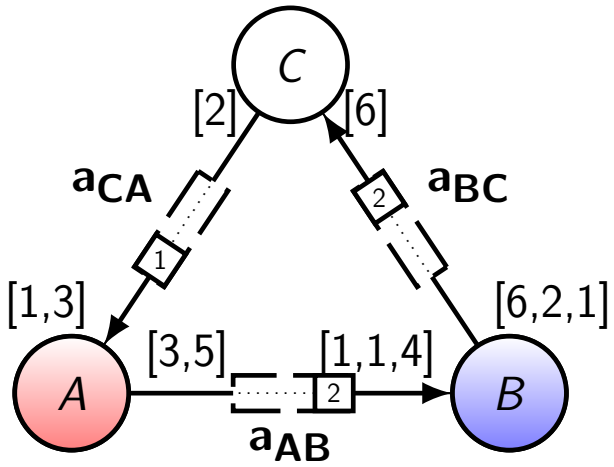
Ordonnancement au plus tôt



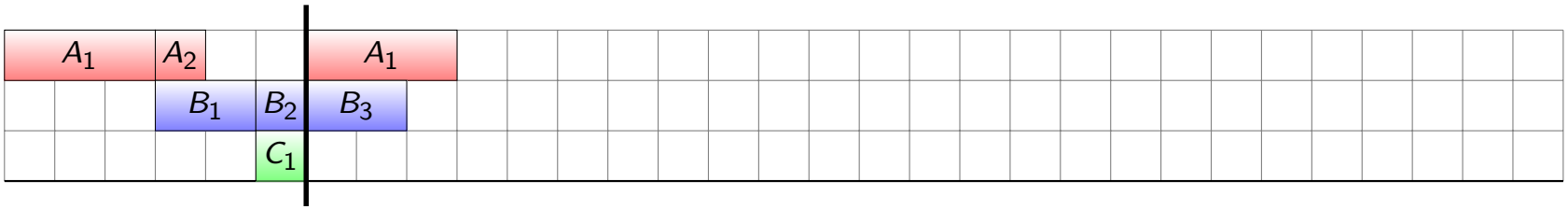
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



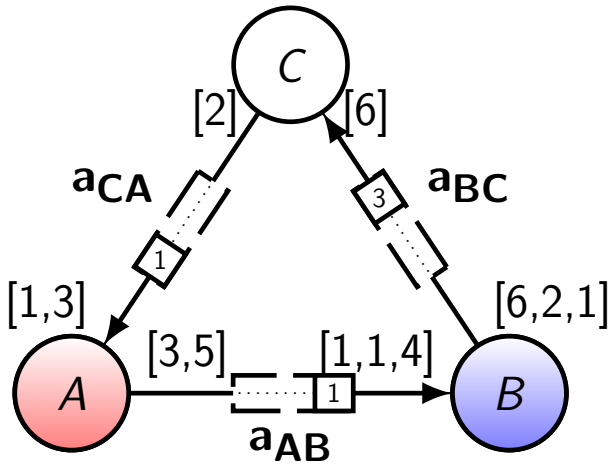
Ordonnancement au plus tôt



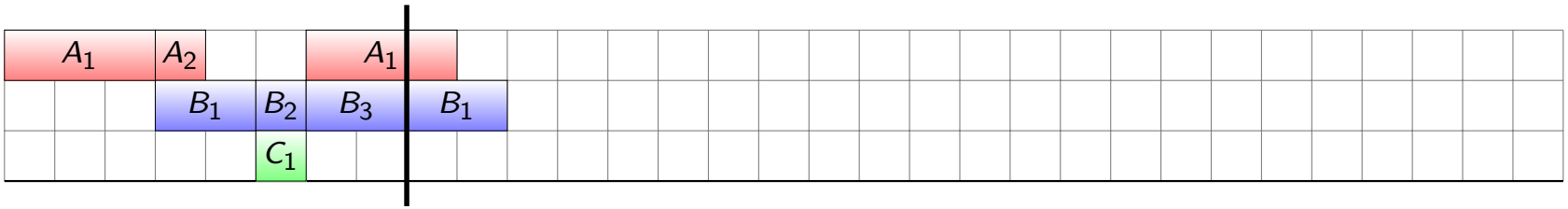
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



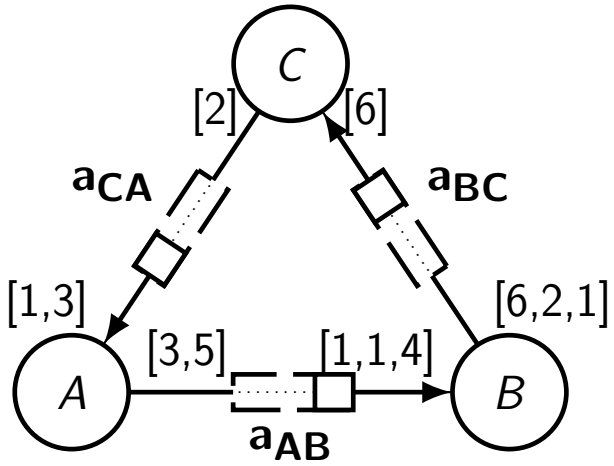
Ordonnancement au plus tôt



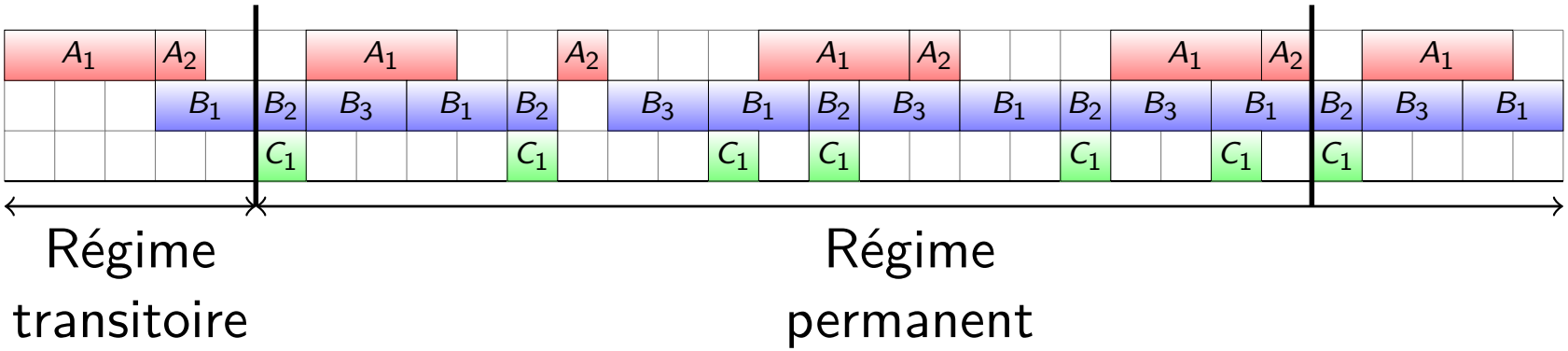
- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource



Ordonnancement au plus tôt



- Durée des tâches :
  - $d(A_1) = 3,\ d(A_2) = 1$
  - $d(B_1) = 2,\ d(B_2) = 1$
  - $d(B_3) = 2,\ d(C_1) = 1$
- Aucune contrainte de ressource

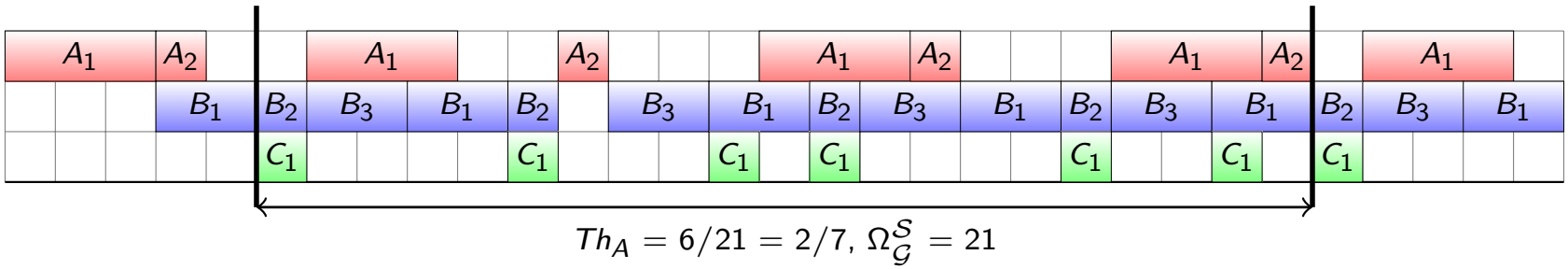


Méthodes existantes

Fréquence de fonctionnement : $Th_t^{\mathcal{S}} = \lim_{n\to\infty} \frac{n}{\mathcal{S}\langle t, n\rangle}$.

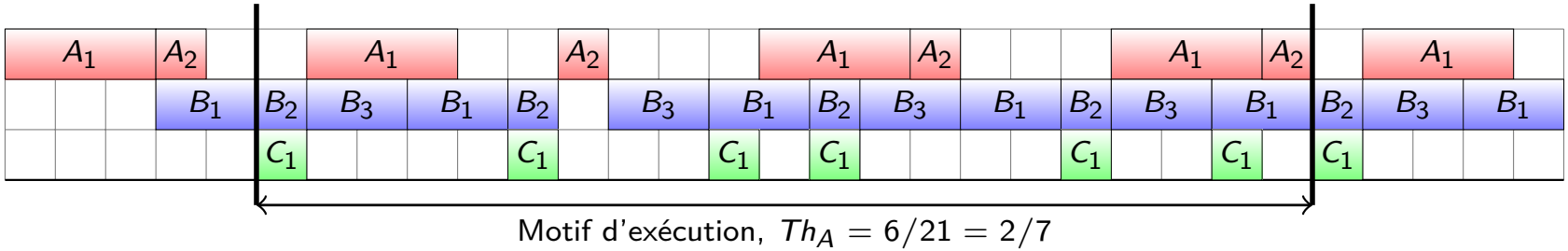
Lorsqu'un CSDFG est à mémoire bornée, il existe un équilibre entre la fréquence de fonctionnement de ses tâches.

$$\Omega_{\mathcal{G}}^{\mathcal{S}} = \frac{N_t^{\mathcal{G}}}{Th_t^{\mathcal{S}}} \quad \forall t \in \mathcal{T}$$



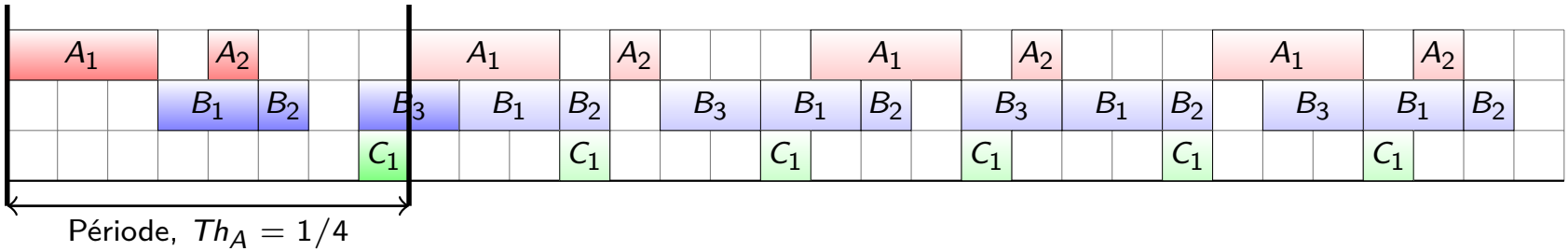
Méthodes existantes

Méthodes exactes (ordonnancements *au plus tôt*) :



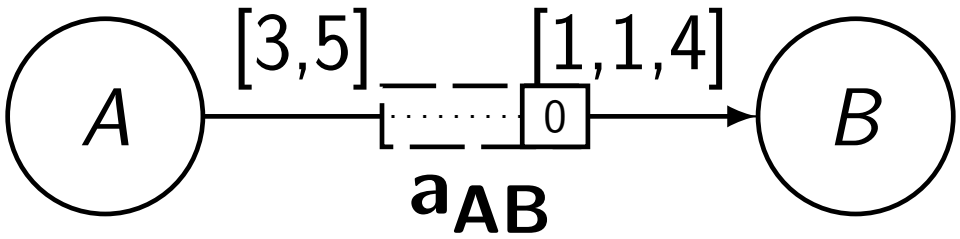
✓ Optimal ✗ Complexité exponentielle.

Méthodes approchées (ordonnancements périodiques) :

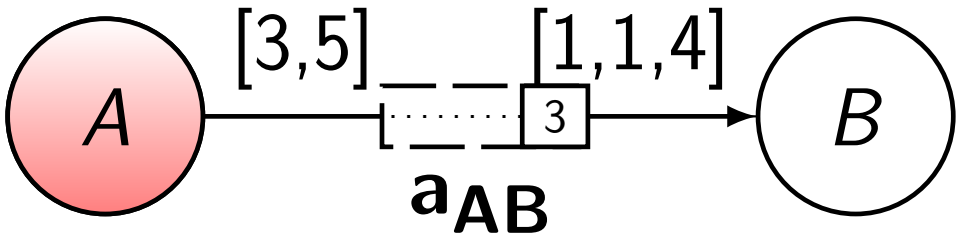


✓ Polynomiale ✗ Borne inférieure.

Définition d'une relation de précédence

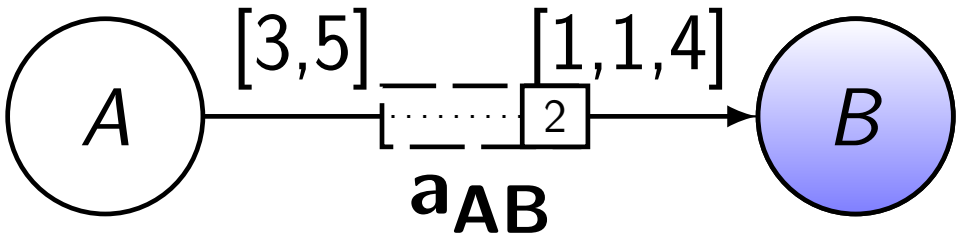


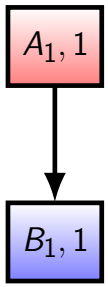
Définition d'une relation de précédence



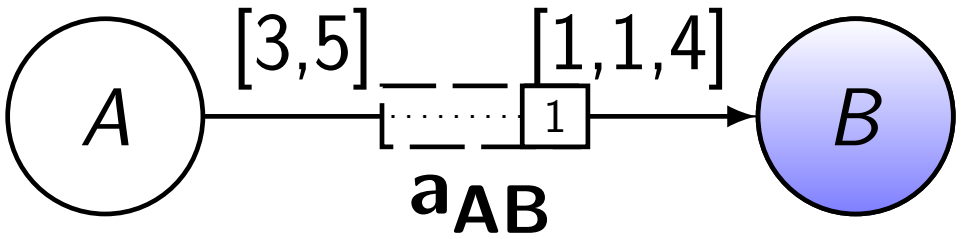
$A_1, 1$

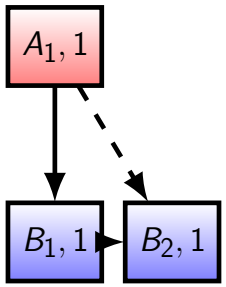
Définition d'une relation de précédence



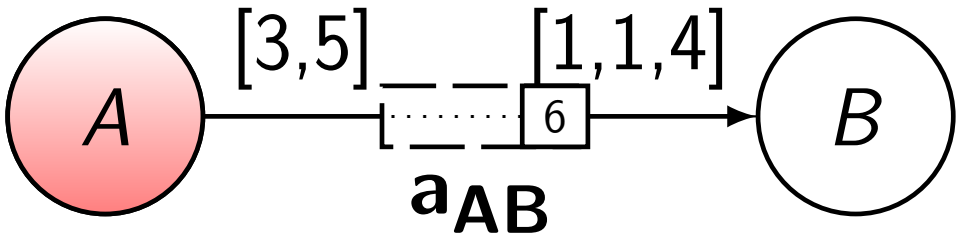


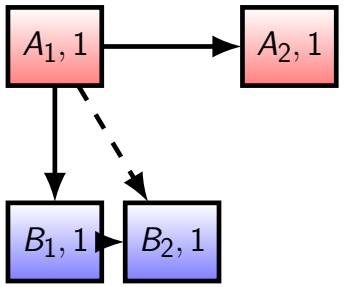
Définition d'une relation de précédence



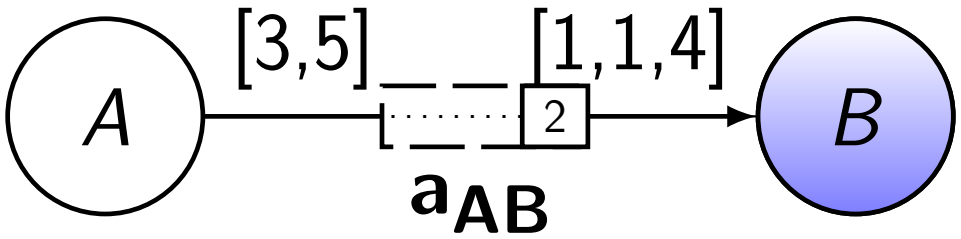


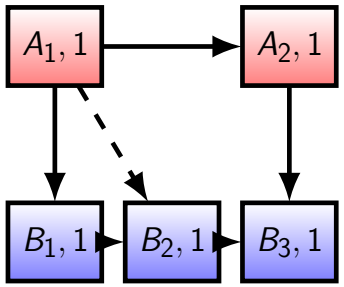
Définition d'une relation de précédence



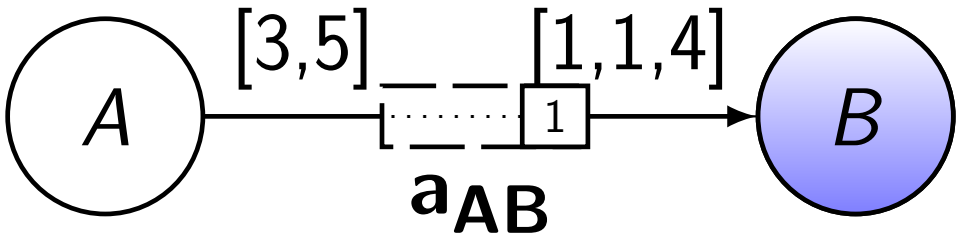


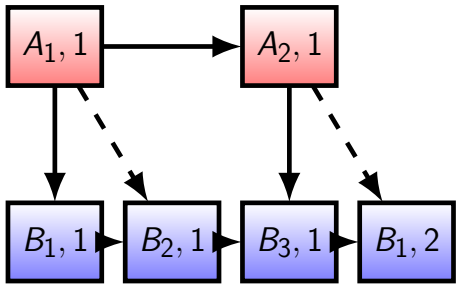
Définition d'une relation de précédence



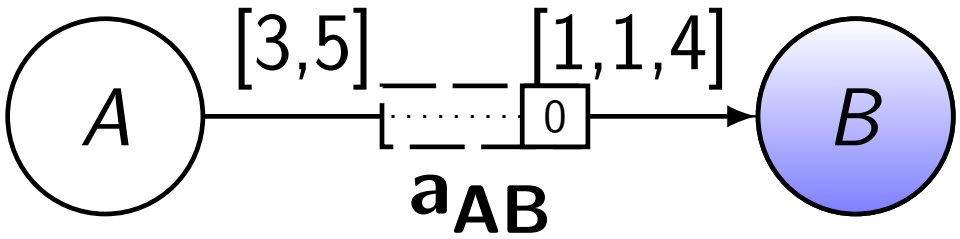


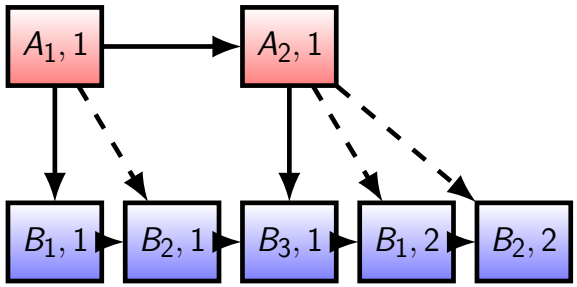
Définition d'une relation de précédence



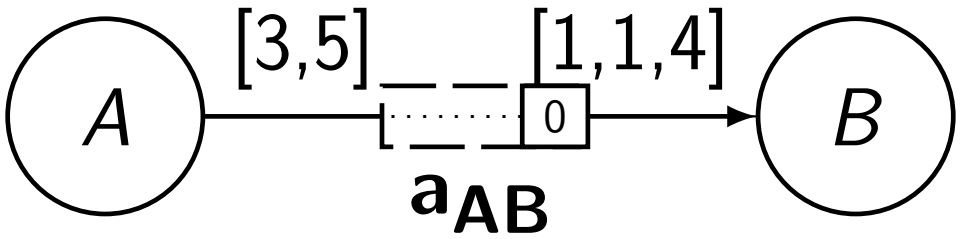


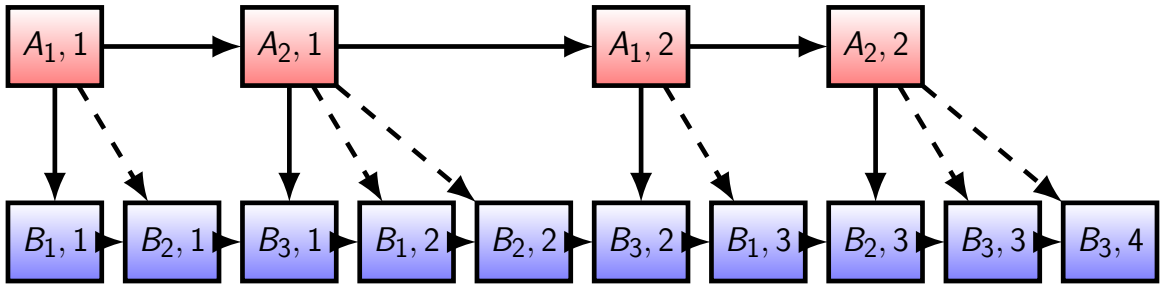
Définition d'une relation de précédence





Définition d'une relation de précédence





Définition d'un ordonnancement valide

Définition (Ordonnancement valide)

S'il existe une relation de précédence entre deux exécutions $\langle t_k, n \rangle$ et $\langle t'_{k'}, n' \rangle$, un ordonnancement valide doit vérifier

$$\mathcal{S}\langle t'_{k'}, n' \rangle \geq \mathcal{S}\langle t_k, n \rangle + d(t_k).$$

Objectif : calculer un ordonnancement périodique valide.

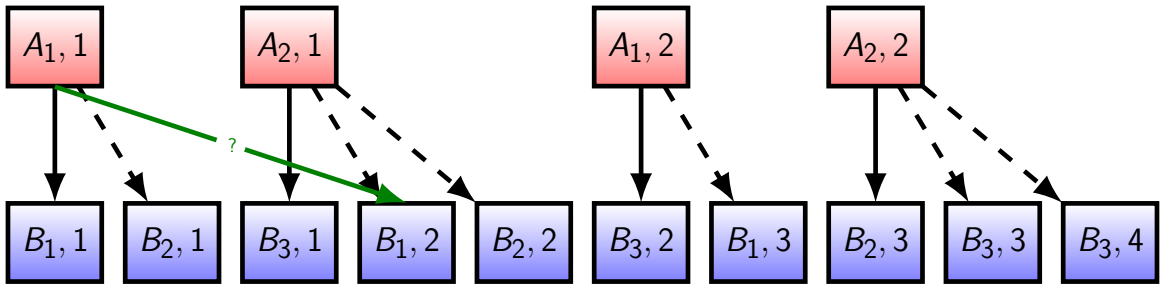
Relation de précédence entre deux exécutions

Lemme (Condition d'existence)

Soit le buffer $a = (t, t') \in \mathcal{A}$. Il existe une relation de précédence entre $\langle t_k, n \rangle$ et $\langle t'_{k'}, n' \rangle$ si et seulement si

$$in_a(k) > M_0(a) + I_a\langle t_k, n\rangle - O_a\langle t'_{k'}, n'\rangle \geq \max\{0, in_a(k) - out_a(k')\}.$$

Entre $\langle A_1, 1\rangle$ et $\langle B_1, 2\rangle$?



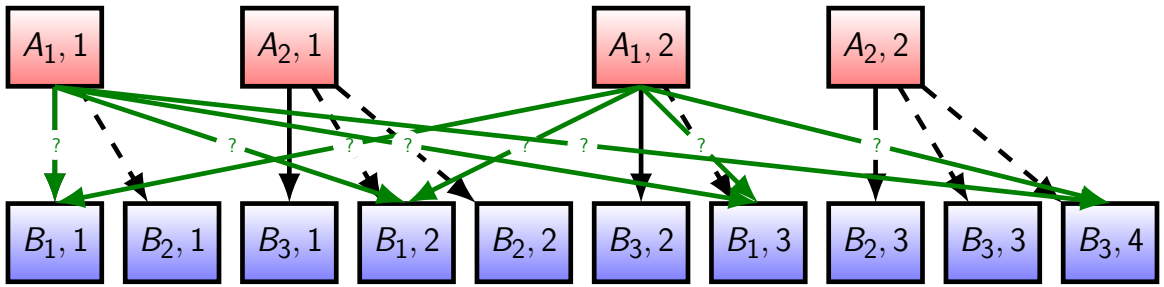
Relation de précédence entre deux phases

Lemme (Condition d'existence entre phases)

un buffer $a = (t, t')$ *induit une relation de précédence entre* $\langle t_k, n \rangle$ *et* $\langle t'_{k'}, n' \rangle$ *si et seulement si*

$$\alpha_a^{\min}(k, k') \leq \alpha(n, n') \leq \alpha_a^{\max}(k, k').$$

Entre A_1 et B_1 ?



Contraintes périodiques

Nous obtenons finalement la condition suivante :

Théorème (Contraintes périodiques)

Les relations de précédence induites par un buffer $a = (t, t')$ sont vérifiées par un ordonnancement périodique $\mathcal{S}$ si et seulement si

$$\mathcal{S}\langle t'_{k'}, 1\rangle - \mathcal{S}\langle t_k, 1\rangle \geq d(t_k) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \frac{\alpha_a^{\max}(k, k')}{N_t^{\mathcal{G}} \times i_a}$$

$\forall (k, k') \in \{1, \cdots, \varphi(t)\} \times \{1, \cdots, \varphi(t')\}$ *avec* $\alpha_a^{\min}(k, k') \leq \alpha_a^{\max}(k, k')$.

Calcul d'un ordonnancement périodique

Ces contraintes peuvent ensuite être utilisées pour calculer un ordonnancement périodique de débit maximum :

Minimiser $\Omega_{\mathcal{G}}^{\mathcal{S}}$ avec

$$\begin{cases} \forall a = (t, t') \in \mathcal{A}, \forall (k, k') \in \mathcal{Y}(a), \\ \quad \mathcal{S}\langle t'_{k'}, 1\rangle - \mathcal{S}\langle t_k, 1\rangle \geq d(t_k) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \dfrac{\alpha_a^{\max}(k, k')}{i_a \times N_t^{\mathcal{G}}} \\ \forall t \in \mathcal{T}, \forall k \in \{1, \cdots, \varphi(t)\}, \mathcal{S}\langle t_k, 1\rangle \in \mathbb{R}^+ \\ \Omega_{\mathcal{G}}^{\mathcal{S}} \in \mathbb{R}^+ - \{0\} \end{cases}$$

- ✓ Un programme linéaire fournissant le débit maximal périodique
- ✓ Polynomial (*Maximum Cycle Ratio Problem*)
- ✓ Optimal pour les ordonnancements périodiques
- ✗ Borne supérieure de la période optimale

Résultats expérimentaux

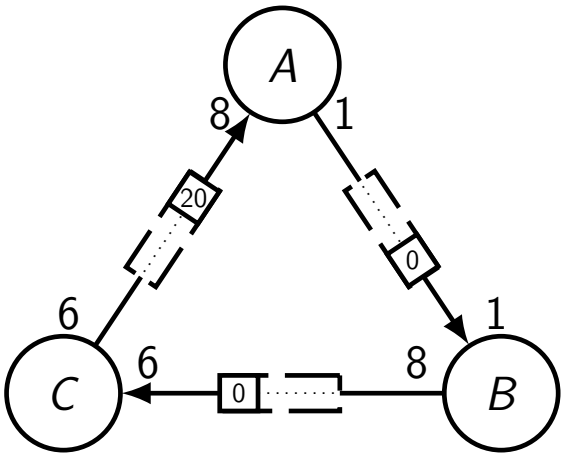
Application		Periodique		Optimale (Stuijk2008)	
sans borne	BlackScholes	100 %	13ms	100 %	42ms
	Echo	100 %	10ms	100 %	65ms
	JPEG2000	100 %	69ms	100 %	6sec
	Pdetect	100 %	80ms	100 %	216ms
	H264 Encoder	100 %	544ms	100 %	17j
bornées	BlackScholes	100 %	24ms	100 %	7sec
	Echo	28 %	18ms	100 %	499sec
	JPEG2000	N/S	148ms	-	>60h
	Pdetect	15 %	93ms	100 %	4928sec
	H264 Encoder	? %	1sec	-	>60h

? % : performances inconnues N/S : aucune solution périodique

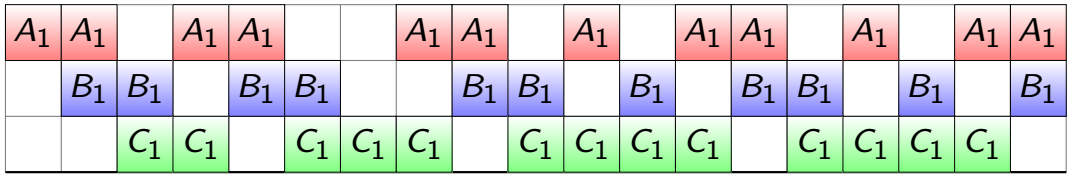
✓ rapide ✓ solution alternative ✗ borne inférieure

Comment améliorer ces résultats ?

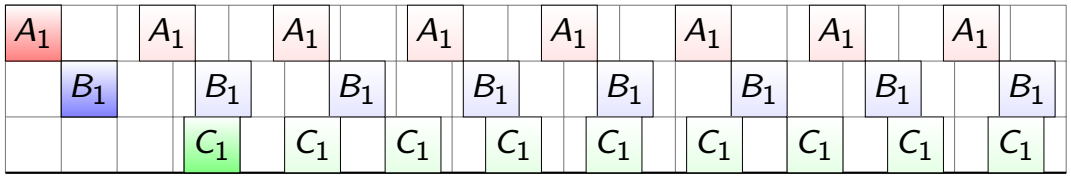
Ordonnancement K-périodique



Au plus tôt (✓ Optimal ✗ Exponentielle) :

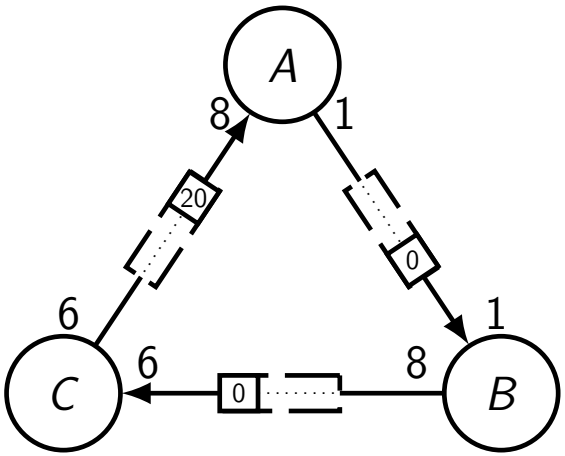


Périodique (✓ Polynomiale ✗ Non optimal) :



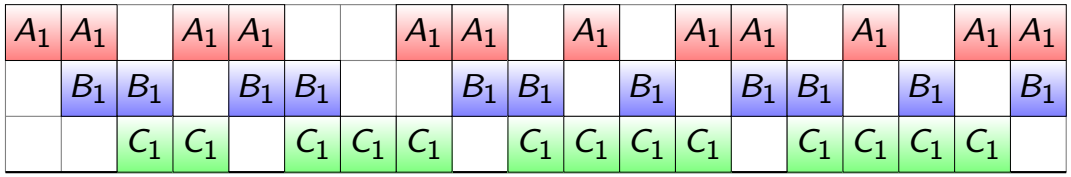
- $d(A_1) = 1$
- $d(B_1) = 1$
- $d(B_1) = 1$

Ordonnancement K-périodique

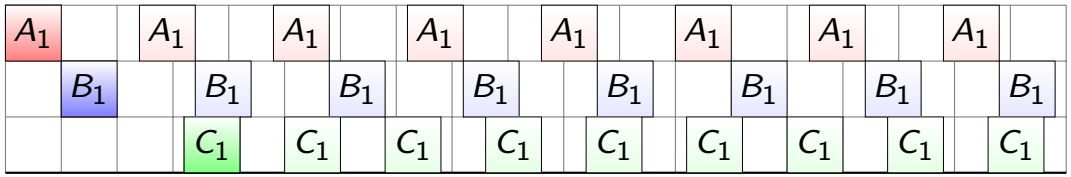


- $d(A_1) = 1$
- $d(B_1) = 1$
- $d(B_1) = 1$

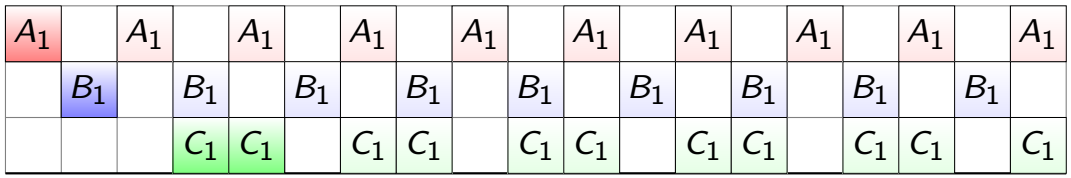
Au plus tôt (✓ Optimal ✗ Exponentielle) :



Périodique (✓ Polynomiale ✗ Non optimal) :



K-périodique (✓ Flexible) :

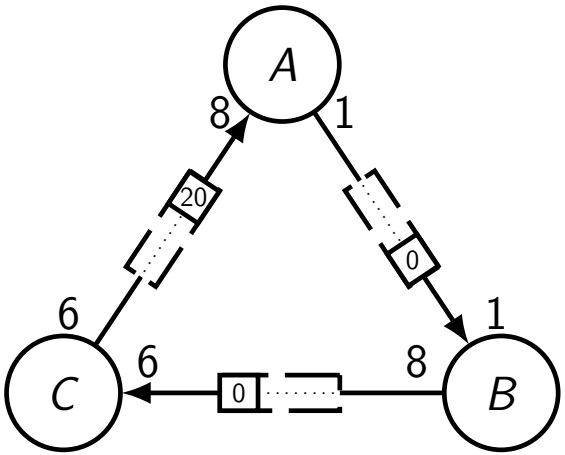


Vecteur de périodicité, $K = [1, 1, 2]$

Un vecteur atteignant le débit maximum

Propriété 1 - Optimalité

Soit $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ un SDFG, considérons alors un ordonnancement K-périodique de vecteur de périodicité $K^{\mathcal{G}} = N^{\mathcal{G}}$. Cet ordonnancement atteint le débit maximum de $\mathcal{G}$.



Ordonnancement *asap* :

A_1	A_1		A_1	A_1			A_1	A_1		A_1		A_1	A_1		A_1		A_1	A_1
	B_1	B_1		B_1	B_1			B_1	B_1		B_1		B_1	B_1		B_1		B_1
		C_1	C_1		C_1	C_1	C_1		C_1	C_1	C_1	C_1		C_1	C_1	C_1	C_1	

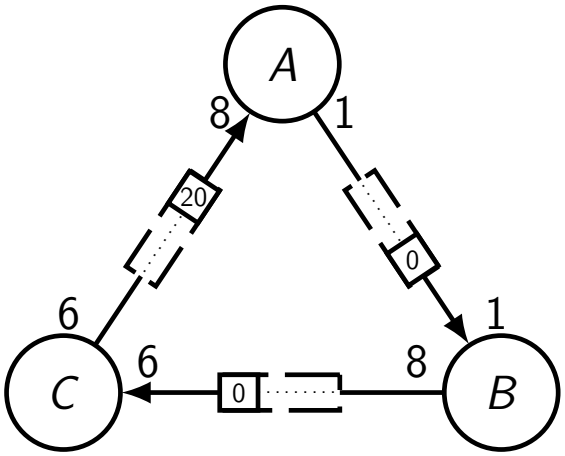
Vecteur $K^{\mathcal{G}} = N^{\mathcal{G}} = [3, 3, 4]$:

A_1		A_1	A_1		A_1		A_1	A_1		A_1		A_1	A_1		A_1		A_1	A_1
	B_1		B_1	B_1		B_1		B_1	B_1		B_1		B_1	B_1		B_1		B_1
		C_1		C_1	C_1	C_1	C_1		C_1	C_1	C_1	C_1		C_1	C_1	C_1	C_1	

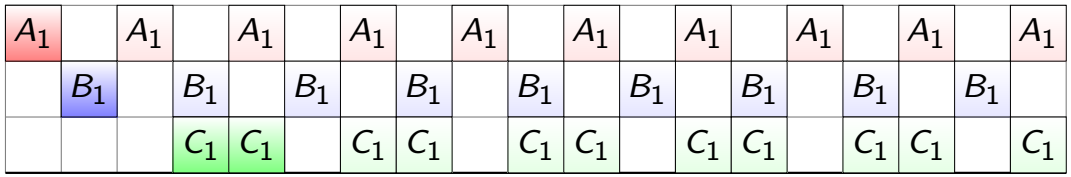
Des solutions non linéaires

Propriété 2 - Non-linéarité

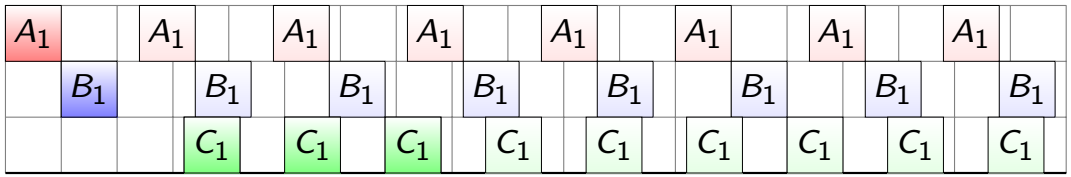
La fréquence de fonctionnement n'augmente pas nécessairement lorsque les facteurs de périodicité augmentent.



Vecteur $K^{\mathcal{G}} = [1, 1, 2]$:



Vecteur $K^{\mathcal{G}} = [1, 1, 3]$:

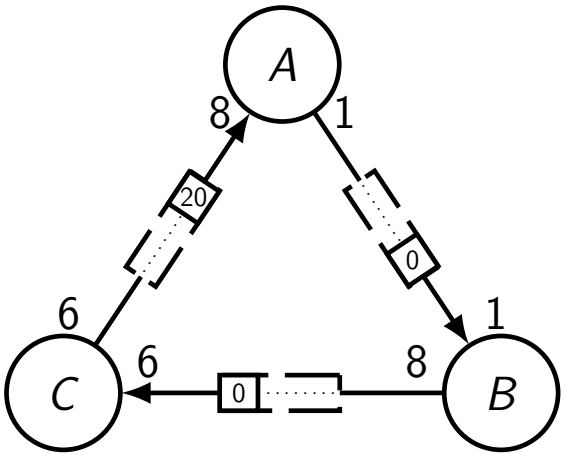


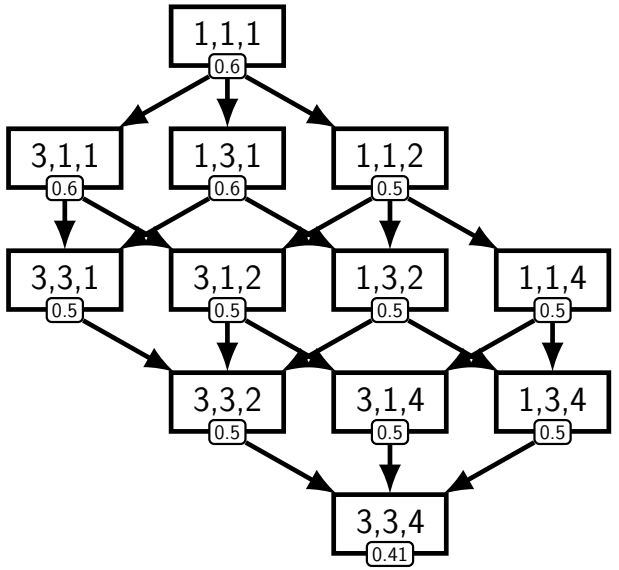
Dominance

Propriété 3 - Dominance

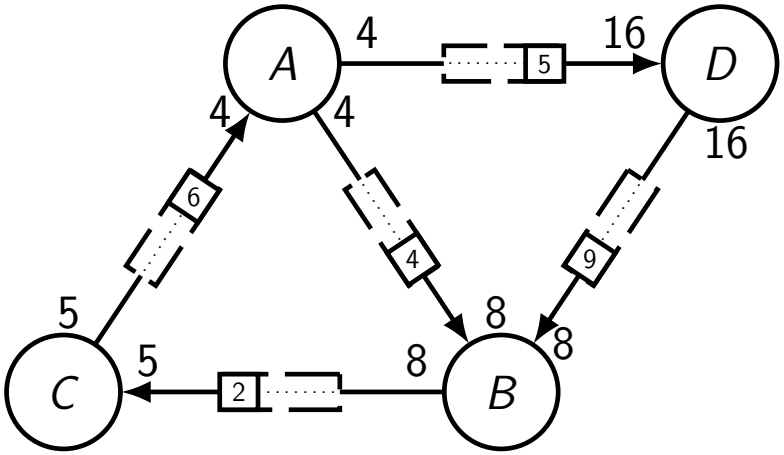
Soit un SDFG $\mathcal{G}$ et deux vecteurs périodicité $K^{\mathcal{G}}$ et $K'^{\mathcal{G}}$. Si $\forall t \in \mathcal{T}$, $K'^{\mathcal{G}}_t = f \times K^{\mathcal{G}}_t$ alors $\forall \mathcal{S} \in S^{opt}_{\mathcal{G}}(K^{\mathcal{G}})$, $\forall \mathcal{S}' \in S^{opt}_{\mathcal{G}}(K'^{\mathcal{G}})$

$$\Omega^{\mathcal{S}'}_{\mathcal{G}} \leq \Omega^{\mathcal{S}}_{\mathcal{G}}.$$

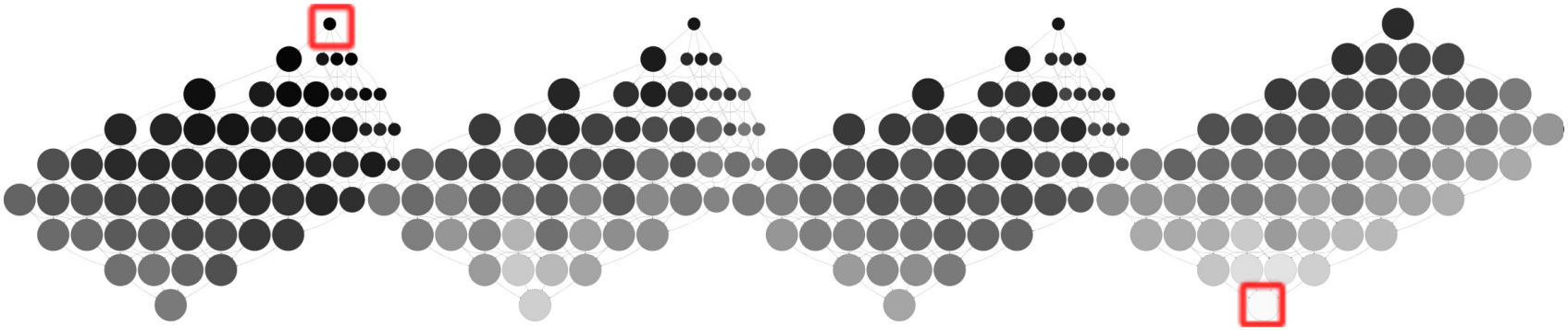




Dominance

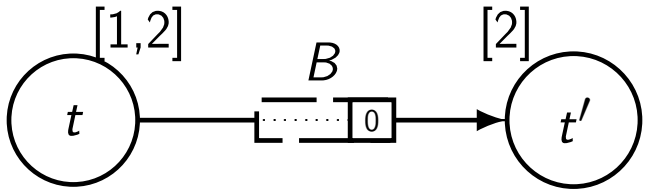


Vecteur $K^{\mathcal{G}}$	C_K	Perf.
$[1, 1, 1, 1]$	7	20%
$[2, 1, 1, 1]$	10	41%
$[\mathbf{4}, \mathbf{1}, \mathbf{1}, \mathbf{1}]$	**16**	**82%**
$[16, 20, 5, 10]$	87	100%

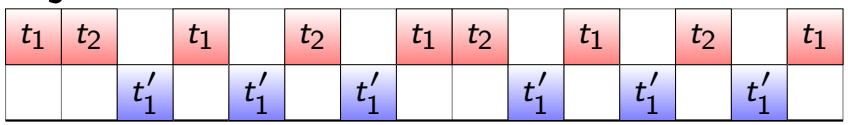


Dimensionnement mémoire

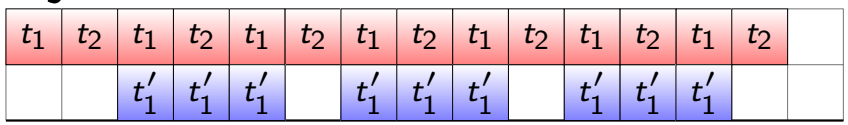
- Le dimensionnement mémoire a un impact sur la vivacité
- Il en a aussi sur les performances



3 jetons maximum :

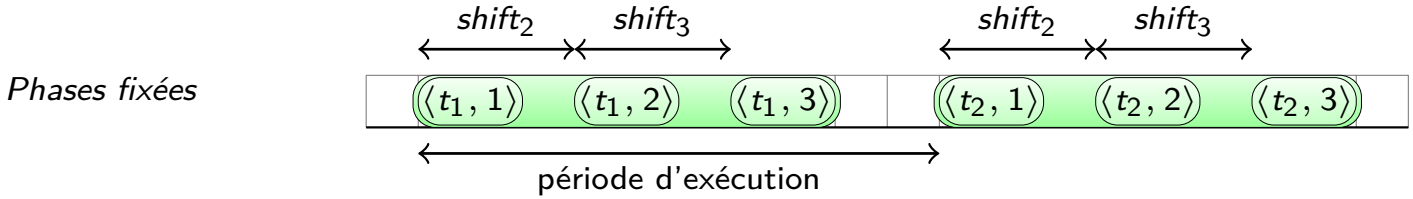


4 jetons maximum :



Méthodes existantes

- Méthode exacte [Stuijk et al., 2008]
  - Explore un ensemble (combinatoire) de solutions
  - Évaluation du débit maximal (asap)
  - Temps de calcul bien trop long sur les instances existantes
- Méthode approchée [Benazouz et al., 2013]
  - Une contrainte de débit est fixée
  - Considère des ordonnancements périodiques uniquement
  - Les écarts entre chaque phase sont prédéfinis



Objectif : une méthode rapide et plus efficace

Dimensionnement mémoire avec contrainte de débit

On adapte nos résultats au dimensionnement mémoire :

Minimiser $\sum_{a \in Fb(\mathcal{A})} M_0(a)$ avec

$$\begin{cases} \forall a = (t, t') \in \mathcal{A}, \forall (k, k') \in \mathcal{Y}(a), \\ \quad \mathcal{S}\langle t'_{k'}, 1\rangle - \mathcal{S}\langle t_k, 1\rangle \geq d(t_k) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \dfrac{\alpha_a^{\max}(k, k')}{i_a \times N_t^{\mathcal{G}}} \\ \forall a' = (t, t') \in Fb(\mathcal{A}), \forall (k, k') \in \mathcal{Y}(a'), \\ \quad \mathcal{S}\langle t'_{k'}, 1\rangle - \mathcal{S}\langle t_k, 1\rangle \geq d(t_k) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \dfrac{\alpha_{a'}^{\max}(k, k')}{i_{a'} \times N_t^{\mathcal{G}}} \\ \forall t \in \mathcal{T}, \forall k \in \{1, \cdots, \varphi(t)\}, \mathcal{S}\langle t_k, 1\rangle \in \mathbb{R}^+ \end{cases}$$

- `PSizing-ILP` : résolution du programme en nombre entier
- `PSizing-LP` : relaxation, résolution d'un programme linéaire

Expérimentations

Application	Min-max		PSizing-LP	
BlackScholes	16 16 22	77ms	16 16 22	155ms
Echo	28 28 28	40ms	28 28 28	79ms
H264 Encoder	N/S N/S N/S	3sec	1369 1369 1369	5sec
JPEG2000	3936 4027 4153	1sec	3635 3733 3864	2sec
Pdectect	4264 4327 5375	5sec	3959 4123 5191	4min

✓ Solutions améliorées (jusqu'à 10%) ✗ Sans garanties

Expérimentations

Application	PSizing-LP		PSizing-ILP	
BlackScholes	16 16 22	155ms	16 16 22	120ms
Echo	28 28 28	79ms	28 28 28	60ms
H264 Encoder	1369 1369 1369	5sec	1369 1369 1369	32sec
JPEG2000	3635 3733 3864	2sec	3502 3600 3725	5min
Pdetect	3959 4123 5191	4min	3958 3958 5068	10h

✓ écart de 5% max. ✓ Optimalité périodique

Intégration dans AccessCore

- Le modèle ΣC à beaucoup évolué, il évoluera encore
- Nous avons adapté nos méthodes à ce modèle
- Elles sont intégrées dans la chaine de compilation actuelle

Une fonction de seuil d'exécution

- Issue des *Computation Graph* [Karp et Miller, 1966]
- Principalement utilisé pour les fenêtres glissantes

Des phases d'initialisation

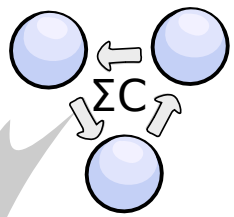
- *Phased Computation Graph* [Thies et al., 2002]
- Marquage plus flexible
- Utilisation mémoire réduite

Conclusion - Résumé

Nos contributions

- Vivacité
  - Preuve de l'équivalence SC1 et SC2
  - Nous proposons un algorithme vérifiant la vivacité d'un CSDFG
  - Dans nos travaux, nous avons aussi appliqué ces résultats
    - au dimensionnement minimal
    - à la génération d'instances aléatoires
- Évaluation du débit
  - Algorithme d'ordonnancement périodique
  - Études des ordonnancements K-périodiques
- Dimensionnement mémoire avec contrainte de débit
  - Nous proposons deux nouveaux algorithmes
    - `PSizing-ILP` : résolution du programme en nombre entier
    - `PSizing-LP` : relaxation, résolution d'un programme linéaire
  - Ces travaux sont intégrés dans la chaîne AccessCore

Résultat



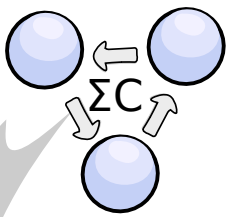
	2011 AccessCore	2011 État de l'art
fonctionnement valide ?	min	min
performances suffisantes ?	-	heure
mémoire necessaire ?	min* -	ms* ms*

Pas d'évaluation du débit

Pas de dimension mémoire avec containte de performance

Conclusion – Résumé

Résultat



	2011 AccessCore	2011 État de l'art	Aujourd'hui AccessCore
fonctionnement valide ?	min	min	sec
performances suffisantes ?	-	heure	sec*
mémoire necessaire ?	min* -	ms* ms*	sec* sec*

Pas d'évaluation du débit

Pas de dimension mémoire avec containte de performance

Améliorations et nouvelles applications

Amélioration de l'approche K-périodique

Nos travaux restent inachevés en deux points :

- Existence de dominances entre les vecteurs
- Méthode efficace pour les explorer

Application au problème de partitionnement

- Exploration des solutions à l'aide de métaheuristiques
- L'évaluation du débit périodique comme fonction de décision

Support des contraintes de latence

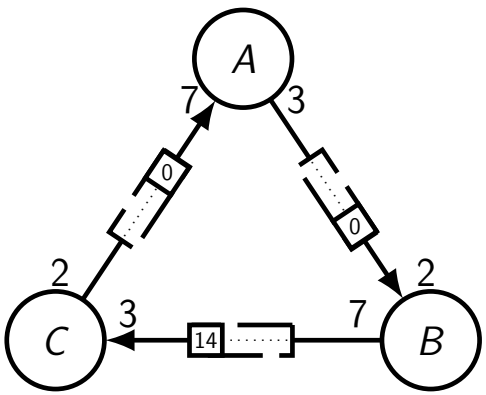
- Intégration de nouvelles contraintes
- Etudier l'impact (négatif ?) des ordonnancements périodiques

Q & R

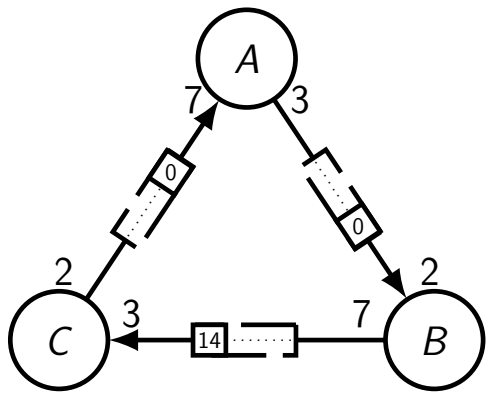
MERCI DE VOTRE ATTENTION

Normalisation [Marchetti et Munier-Kordon, 2009]

- Une transformation polynomiale
- Tout graphe consistant est normalisable
- Des propriétés séduisantes
  - Un taux unique par tâche
  - Le nombre de jeton dans chaque circuit est constant



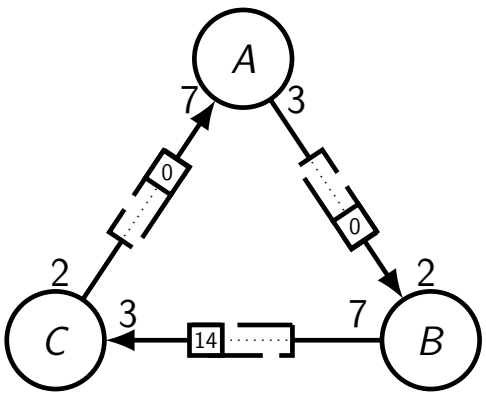
SDFG original



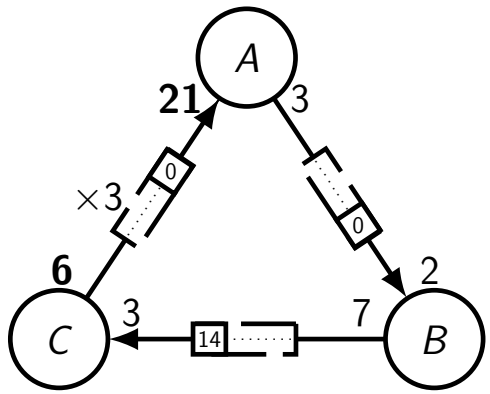
$\Rightarrow$

Normalisation [Marchetti et Munier-Kordon, 2009]

- Une transformation polynomiale
- Tout graphe consistant est normalisable
- Des propriétés séduisantes
  - Un taux unique par tâche
  - Le nombre de jeton dans chaque circuit est constant



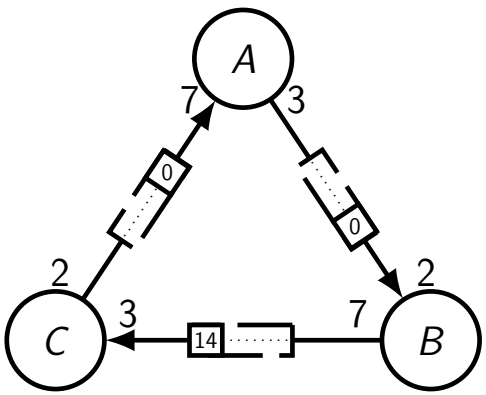
SDFG original



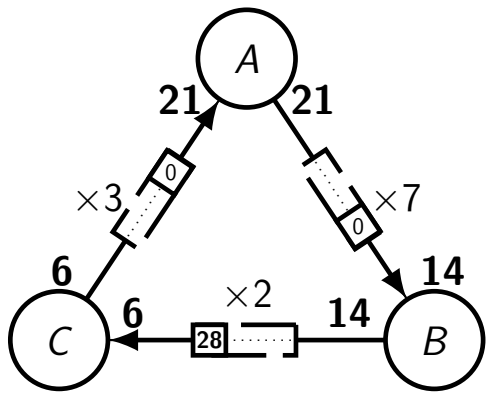
$\Rightarrow$

Normalisation [Marchetti et Munier-Kordon, 2009]

- Une transformation polynomiale
- Tout graphe consistant est normalisable
- Des propriétés séduisantes
  - Un taux unique par tâche
  - Le nombre de jeton dans chaque circuit est constant



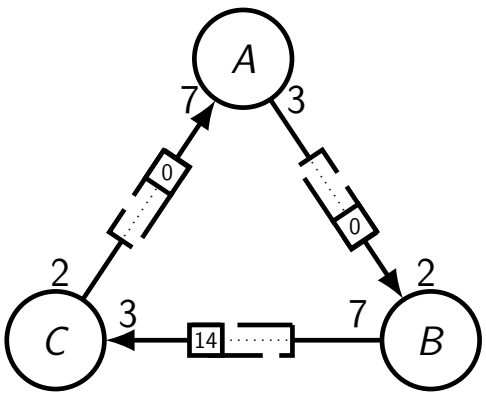
SDFG original



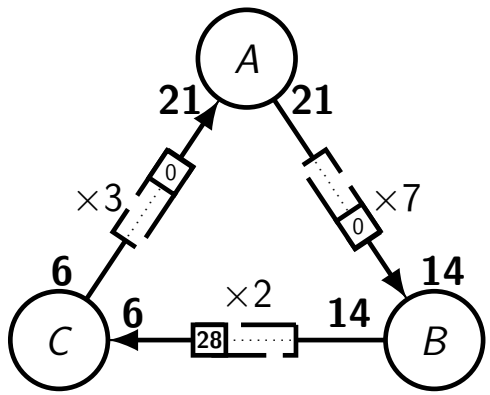
SDFG normalisé

Normalisation [Marchetti et Munier-Kordon, 2009]

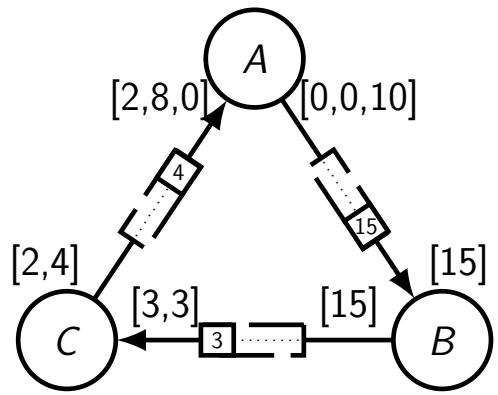
- Une transformation polynomiale
- Tout graphe consistant est normalisable
- Des propriétés séduisantes
  - Un taux unique par tâche
  - Le nombre de jeton dans chaque circuit est constant



SDFG original



SDFG normalisé



CSDFG normalisé

Une condition suffisante de vivacité

Théorème ([Marchetti et Munier-Kordon, 2009])

Le SDFG normalisé $\mathcal{G}$ est vivant si pour chacun de ses circuits $c \in \mathcal{C}$,

$$\sum_{a \in c} M_0(a) > \sum_{a \in c} [in_a - gcd_a]$$

Théorème ([Benazouz, 2012])

Le CSDFG normalisé $\mathcal{G}$ est vivant si pour chacun de ses circuits,
$c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$,

$$\sum_{i=1}^{m} M_0(a_i) > -\sum_{i=1}^{m} step_{a_i} + \sum_{i=1}^{m} \max_{k^i \in \{1,\cdots,\varphi(t^i)\}} \left[O_{a_{i-1}} \langle t^i, k^i \rangle - I_{a_i} Pr \langle t^i, k^i \rangle \right].$$

Equivalence des conditions SC1 et SC2

Théorème (SC1 équivalent à SC2)

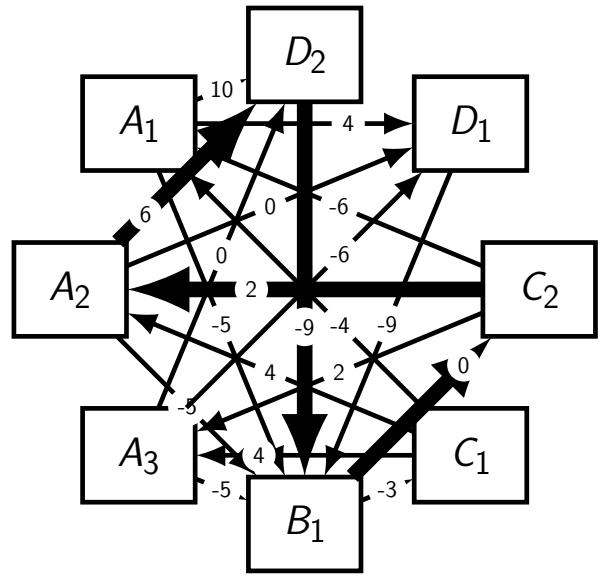
Soit $\mathcal{G}$ un CSDFG normalisé ; les conditions SC1 et SC2 appliquées au CSDFG $\mathcal{G}$ sont équivalentes.

Une méthode polynomiale pour vérifier la condition SC1

$$0 > \sum_{i=1}^{m} \underbrace{\left[O_{a_{i-1}} \langle {t^i}_{k^i}, 1 \rangle - I_{a_i} Pr \langle {t^i}_{k^i}, 1 \rangle - \text{step}_{a_i} - M_0(a_i) \right]}_{W_1(a)}$$

Construction

- $\sum_{t \in \mathcal{T}} \varphi(t)$ tâches
- $\sum_{a=(t,t') \in \mathcal{A}} (\varphi(t) \times \varphi(t'))$ arcs
- Poids des arcs, $W_1(a)$



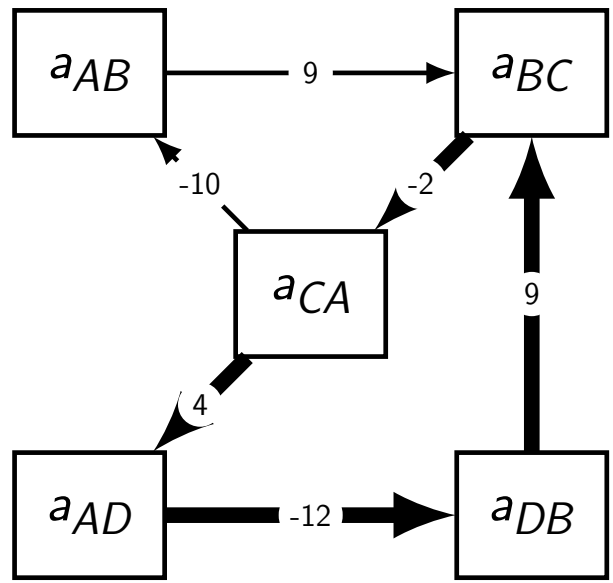
La condition se vérifie si le coût moyen maximum est strictement négatif. Le circuit de coût moyen max est $c = \{A_2, D_2, B_1, C_2\}$.

Une méthode polynomiale pour vérifier la condition SC2

$$0 > \sum_{i=1}^{m} \underbrace{\left[\max_{k^i \in \{1,\cdots,\varphi(t^i)\}} (O_{a_{i-1}}\langle t^i, k^i\rangle - I_{a_i} Pr\langle t^i, k^i\rangle) - M_0(a_i) - \text{step}_{a_i} \right]}_{W_2(a_i, a_{i-1})}$$

Construction

- $|\mathcal{A}|$ tâches
- $\sum_{t \in \mathcal{T}} (d^{in}(t) \times d^{out}(t))$ arcs.
- Poids des arcs, $W_2(a_i, a_j)$)



Le circuit de coût moyen maximum est $c = \{a_{BC}, a_{CA}, a_{AD}, a_{DB}\}$.

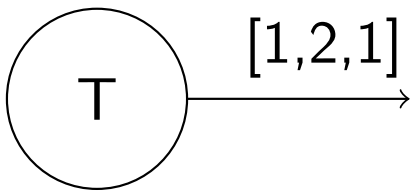
Méthode approchée pour les SDFG [Benazouz et al., 2010]

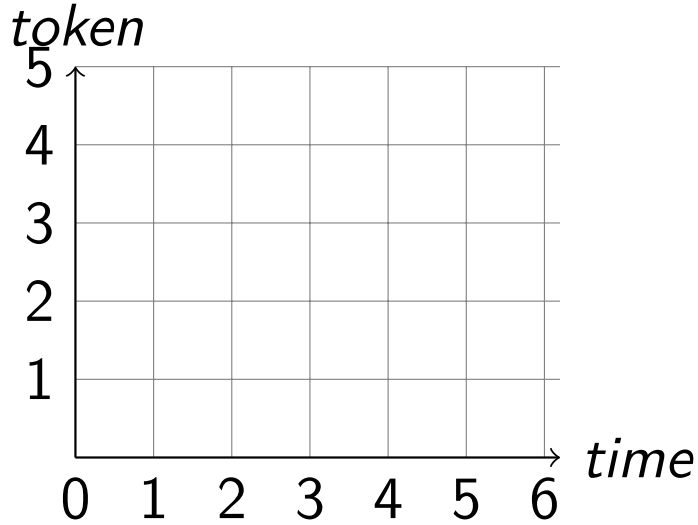
Minimiser $\sum_{a \in Fb(\mathcal{A})} M_0(a)$ avec

$$\begin{cases} \forall a = (t, t') \in \mathcal{A}, \\ \quad \mathcal{S}\langle t', 1\rangle - \mathcal{S}\langle t, 1\rangle \geq d(t) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \dfrac{\alpha_a^{\max}}{in_a \times N_t^{\mathcal{G}}} \\ \forall a' = (t, t') \in Fb(\mathcal{A}), \\ \quad \mathcal{S}\langle t', 1\rangle - \mathcal{S}\langle t, 1\rangle \geq d(t) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \dfrac{\alpha_{a'}^{\max}}{in_{a'} \times N_t^{\mathcal{G}}} \\ \forall t \in \mathcal{T}, \mathcal{S}\langle t, 1\rangle \in \mathbb{R}^+ \end{cases}$$

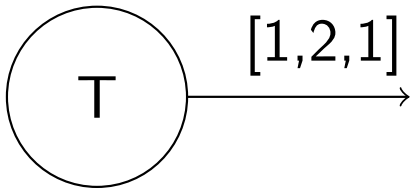
- ordonnancements périodiques [Benabid et al., 2012]
- Programme linéaire en nombre entier
- Une solution approchée existe par relaxation

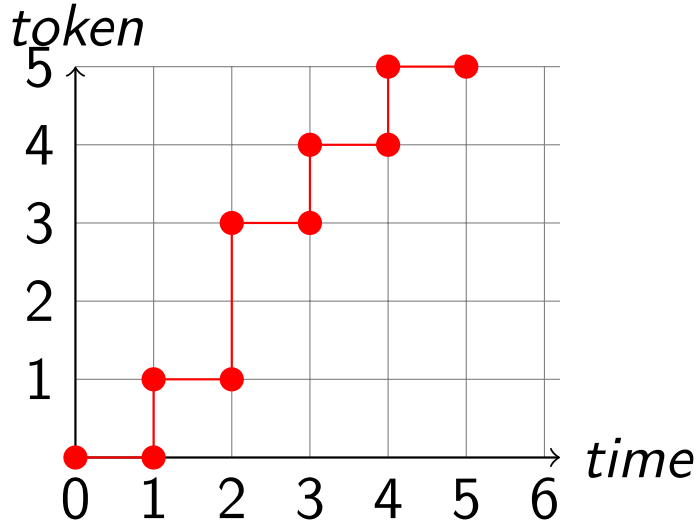
Approche périodique à phase fixe [Wiggers et al., 2007]



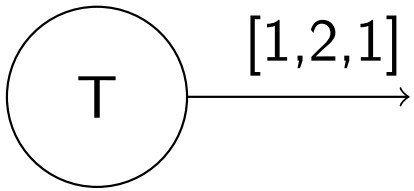


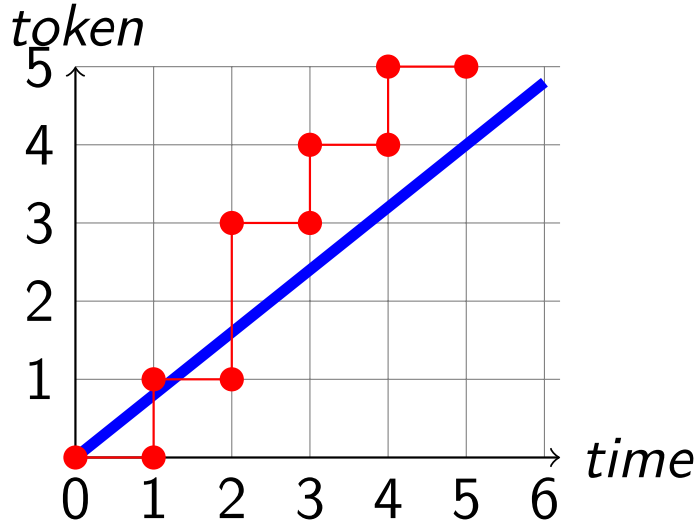
Approche périodique à phase fixe [Wiggers et al., 2007]



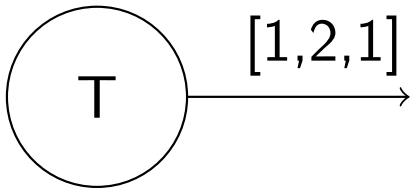


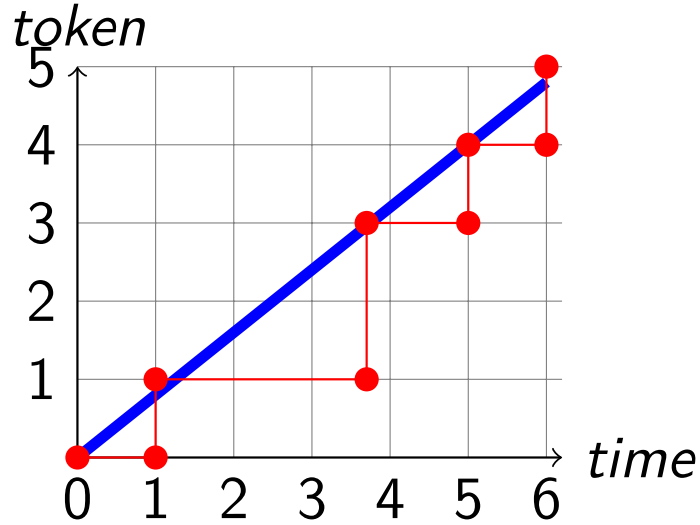
Approche périodique à phase fixe [Wiggers et al., 2007]



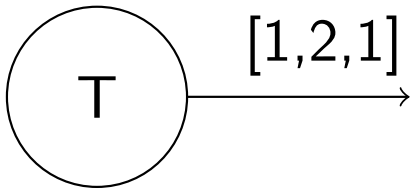


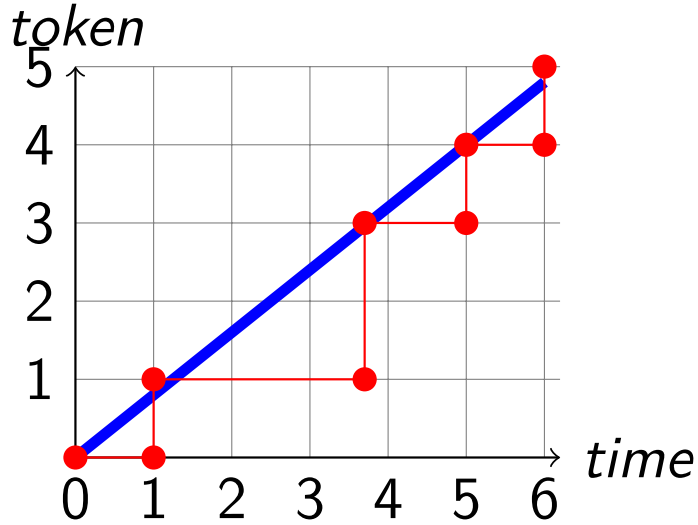
Approche périodique à phase fixe [Wiggers et al., 2007]



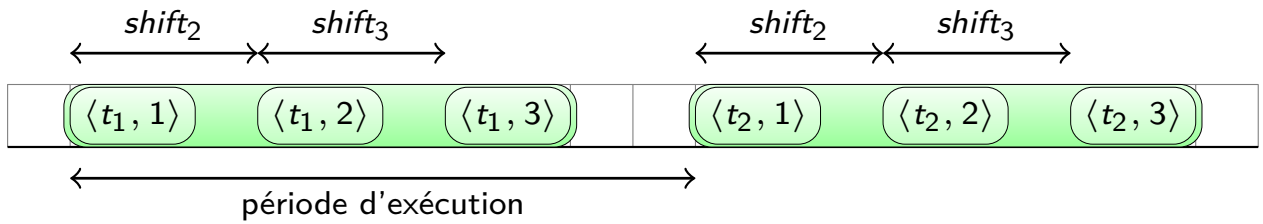


Approche périodique à phase fixe [Wiggers et al., 2007]

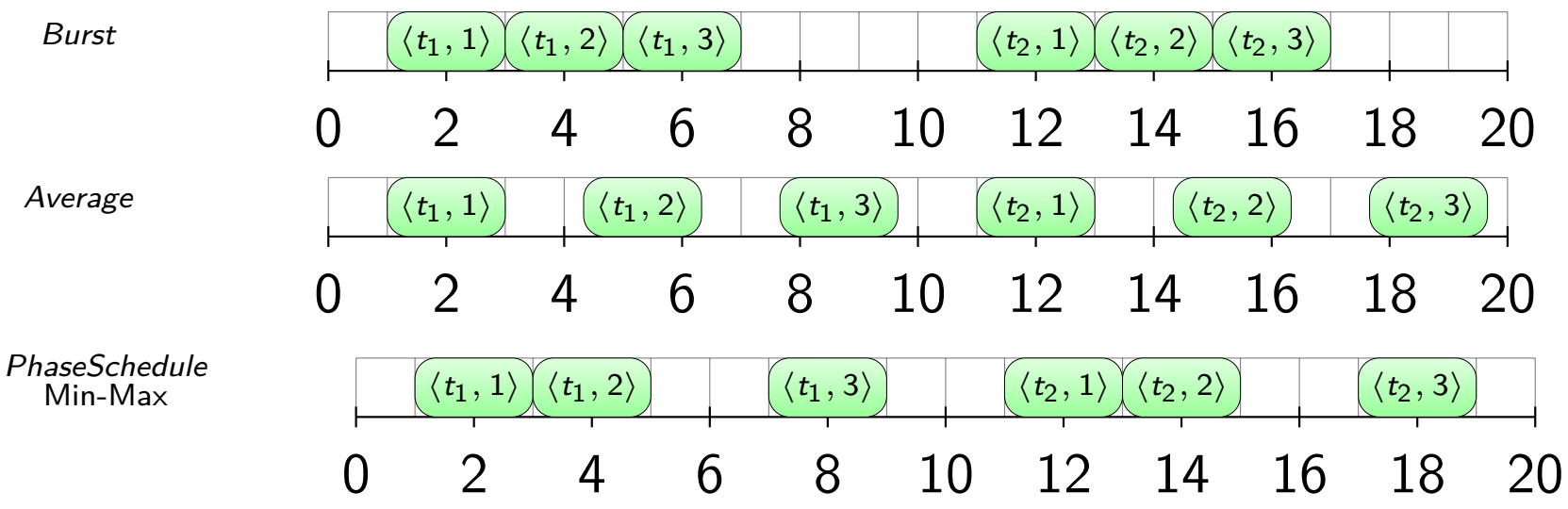




PhaseSchedule



Différentes politiques d'ordonnancement des phases



Relation de précédence

Un buffer $a = (t, t')$ induit une relation de précédence entre les exécutions $\langle t_k, n \rangle$ et $\langle t'_{k'}, n' \rangle$ si et seulement si :

1. $\langle t'_{k'}, n' \rangle$ peut être exécuté après $\langle t_k, n \rangle$
2. $\langle t'_{k'}, n' \rangle$ ne peut pas être exécuté avant $\langle t_k, n \rangle$
3. $\boldsymbol{Pr\langle t'_{k'}, n' \rangle}$ **peut s'exécuter avant la fin de** $\boldsymbol{\langle t_k, n \rangle}$

- Références

Sukumar R. Anapalli, Krishna C. Chakilam, et Timothy W. O'Neil. Static Scheduling for Cyclo Static Data Flow Graphs. In *Parallel and Distributed Processing Techniques and Applications, PDPTA 2009*, pages 302–306. CSREA Press, 2009.

Abir Benabid, Claire Hanen, Olivier Marchetti, et Alix Munier-Kordon. Periodic Schedules for Bounded Timed Weighted Event Graphs. *IEEE Transactions on Automatic Control*, 57(5) :1222 – 1232, 2012.

Mohamed Benazouz. *Buffer Sizing for Stream Processing Applications*. PhD thesis, Université P. et M. Curie, Paris, France, 2012.

Mohamed Benazouz, Olivier Marchetti, Alix Munier-Kordon, T Michel, et Pascal Urard. A new method for minimizing buffer sizes for Cyclo-Static Dataflow graphs. In *Embedded Systems for Real-Time Multimedia (ESTIMedia '10)*, pages 11–20, 2010.

Mohamed Benazouz, Alix Munier-Kordon, Thomas Hujsa, et Bruno Bodin. Liveness Evaluation of a Cyclo-Static DataFlow Graph. In *Design Automation Conference (DAC'13)*, pages 3 :1—-3 :7, Austin, TX, USA, 2013. ACM Press.

Greet Bilsen, Marc Engels, Rudy Lauwereins, et Jean A. Peperstraete. Cyclo-static data flow. *IEEE Transactions on Signal Processing*, pages 3255–3258, 1995.

Joseph T. Buck et Edward A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-93)*, number September, pages 429–432. Ieee, 1993.

Thierry Goubier, Renaud Sirdey, Stéphane Louise, et Vincent David. ΣC : A programming model and language for embedded manycores. *Proceedings of the 11th international conference on Algorithms and architectures for parallel processing (ICA3PP'11)*, 7016 :385–394, 2011.

Gilles Kahn. The semantics of a simple language for parallel programming. *Information processing*, 1974.

Richard M. Karp et Raymond E. Miller. Properties of a model for parallel computations : Determinancy, termination, queueing. *SIAM Journal on Applied Mathematics*, 14(6) :1390–1411, 1966.

Edward A. Lee et David G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9) :1235–1245, 1987.

- *Références*

Olivier Marchetti et Alix Munier-Kordon. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research*, 197(2) :532–540, September 2009.

Sander Stuijk, Marc Geilen, et Twan Basten. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. *IEEE Transactions on Computers*, 57(10) :1331–1345, 2008.

William Thies, Jasper Lin, et Saman Amarasinghe. Phased Computation Graphs in the Polyhedral Model. Technical Report August 2002, 2002.

Maarten H. Wiggers, Marco J.G. Bekooij, Pierre G. Jansen, et Gerard J.M. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 281–292, April 2007.