# BatchQueue: file producteur/consommateur optimisée pour les multi-cœurs

Thomas Preud'homme

Équipe REGAL
Encadrants : Gaël Thomas et Julien Sopena
Directeur de thèse : Bertil Folliot

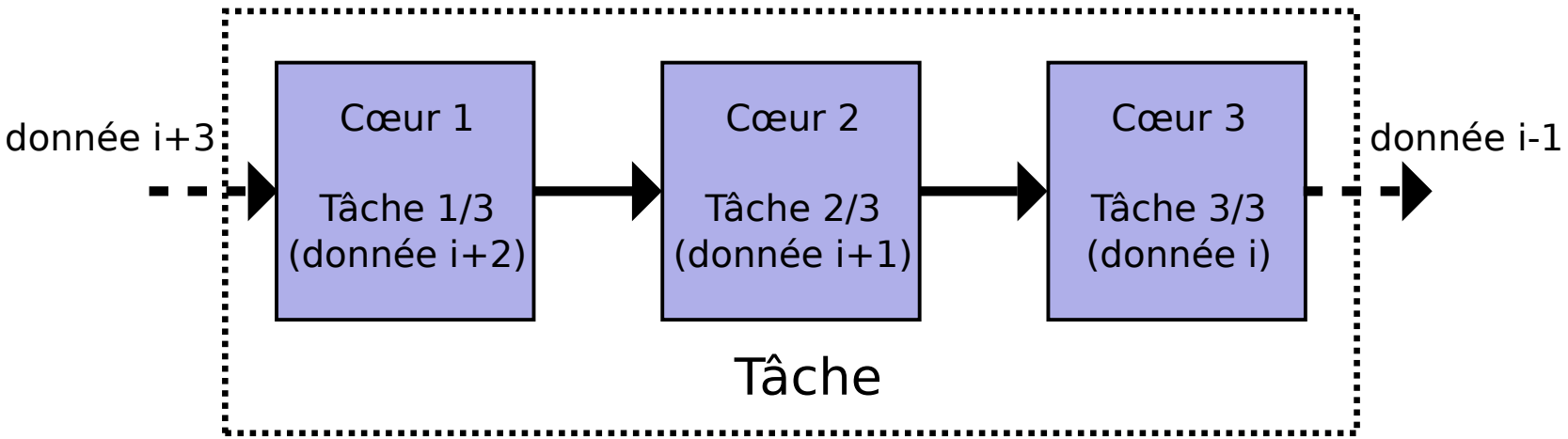


Jeudi 12 mai 2011

# Parallélisme pipeline : intérêt et limite

## Principe

- Diviser un code séquentiel en plusieurs sous-tâches
- **Exécuter chaque sous-tâche sur des cœurs différents**
- Faire transiter les données d'une sous-tâche à une autre
  $\Rightarrow$ Idéalement : latence inchangée et débit multiplié

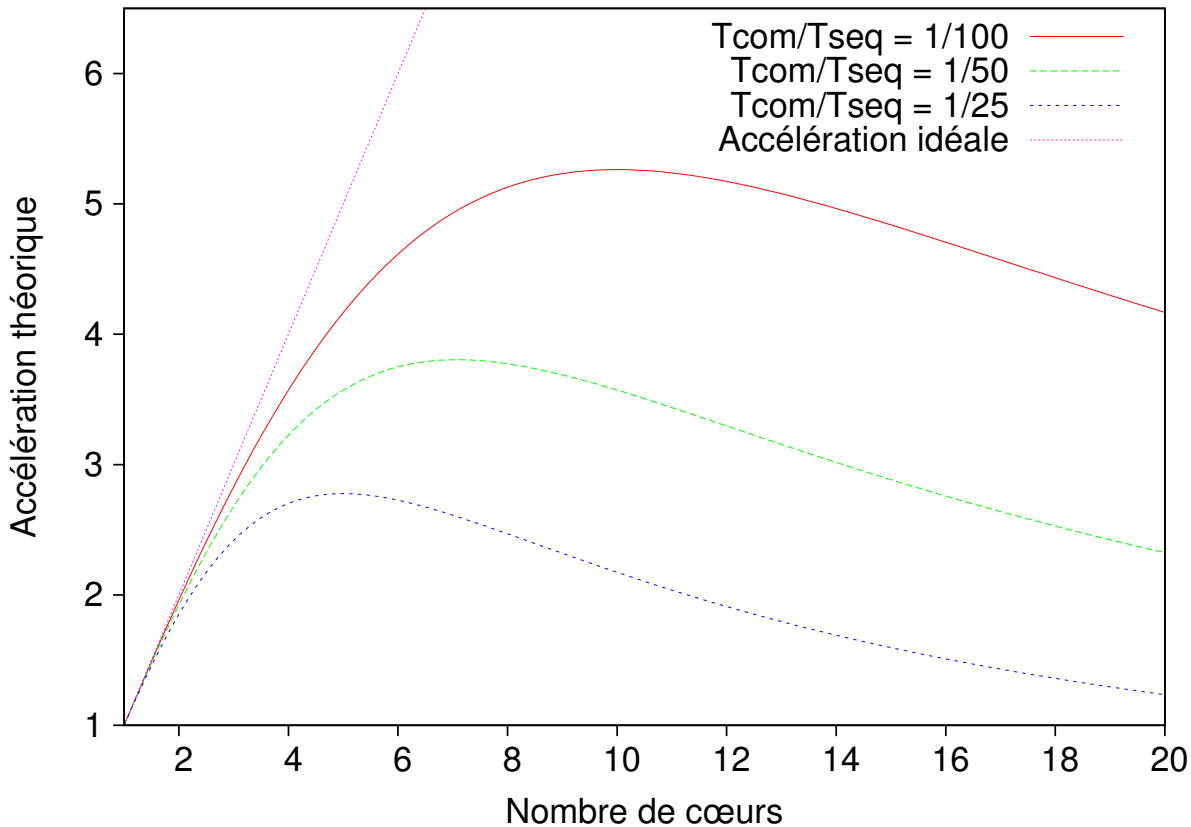



## Limite

Accélération limitée par le temps de communication
$\Rightarrow$ Temps de communication fixe quelque soit le nombre de cœurs

# Parallélisme pipeline : limites

**La communication doit être rapide** : $Accél(i) = \frac{1}{\frac{1}{i} + (i-1)\frac{T_{com}}{T_{seq}}}$




**Problème** : $T_{com}$ borne l'accélération maximale possible

## Mécanismes de communication inter-cœurs

- protocole IP (127.0.0.1)
- socket unix
- tube posix
- **mémoire partagée**

## Intérêt de la mémoire partagée

- Pas d'encapsulation des données
- Pas de changement de contexte (pas d'appel système)

$\Rightarrow$ Mécanisme le plus rapide

# BatchQueue : objectif

**Objectif** : concevoir un nouveau système de communication inter-cœurs rapide et asynchrone utilisant la mémoire partagée

**Principe** : optimiser l'utilisation du cache matériel

**Méthode** : minimiser le partage de variables

**Solution** : Augmenter le débit en relâchant les contraintes sur la latence si nécessaire

# Plan

# Protocole de cohérence des caches

La cohérence des caches est maintenue par le protocole MESI (ou dérivé, ex : MOESI).

## Protocole MESI

- Chaque ligne de chaque processeur possède un état : Modifiée, Exclusive, Partagée (Shared) et Invalide
- L'état d'une ligne définit si celle-ci est :
  - à jour
  - partagée entre plusieurs cœurs
  - modifiée par rapport à la mémoire centrale
- Le protocole MESI décris les relations entre les états
  $\Rightarrow$ Implémente un **verrou lecteur / écrivain**.

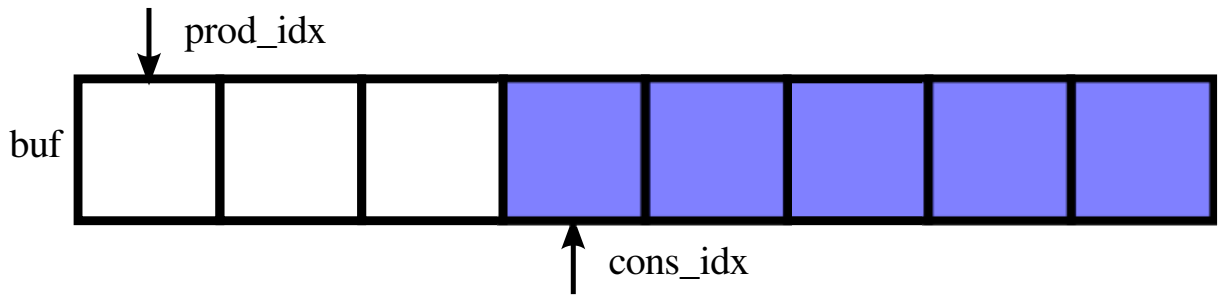
# Protocole de cohérence des caches : ralentissements

## 2 sources de ralentissement

- Lecture depuis Invalide : **récupération** d'une ligne distante
  $\Rightarrow$ Besoin d'une communication inter-processeurs
- Écriture depuis Partagée : **invalidation** des caches distants
  $\Rightarrow$ Besoin d'une synchro avec les autres proc.

# Plan

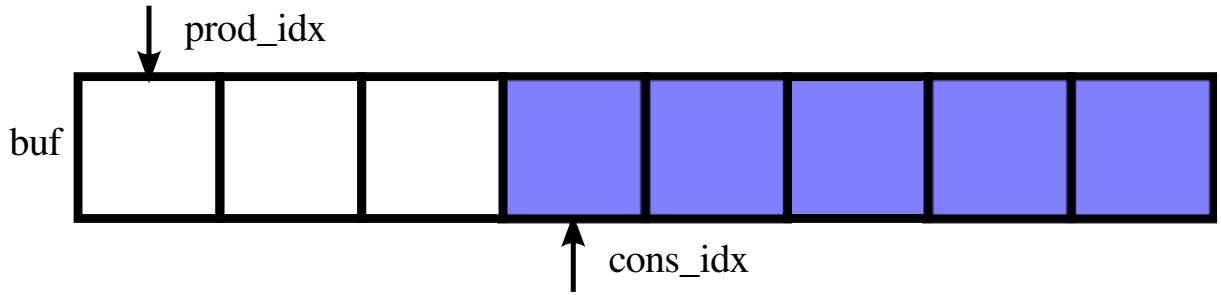
# File Concurrente Sans Verrou de Lamport : structure




**3 variables partagées** :

- *buf* : tampon d'échange entre producteur et consommateur
- *prod_idx* : Indice de la prochaine entrée vide du tampon
- *cons_idx* : Indice de la prochaine entrée pleine du tampon

# File CSV de Lamport : principe

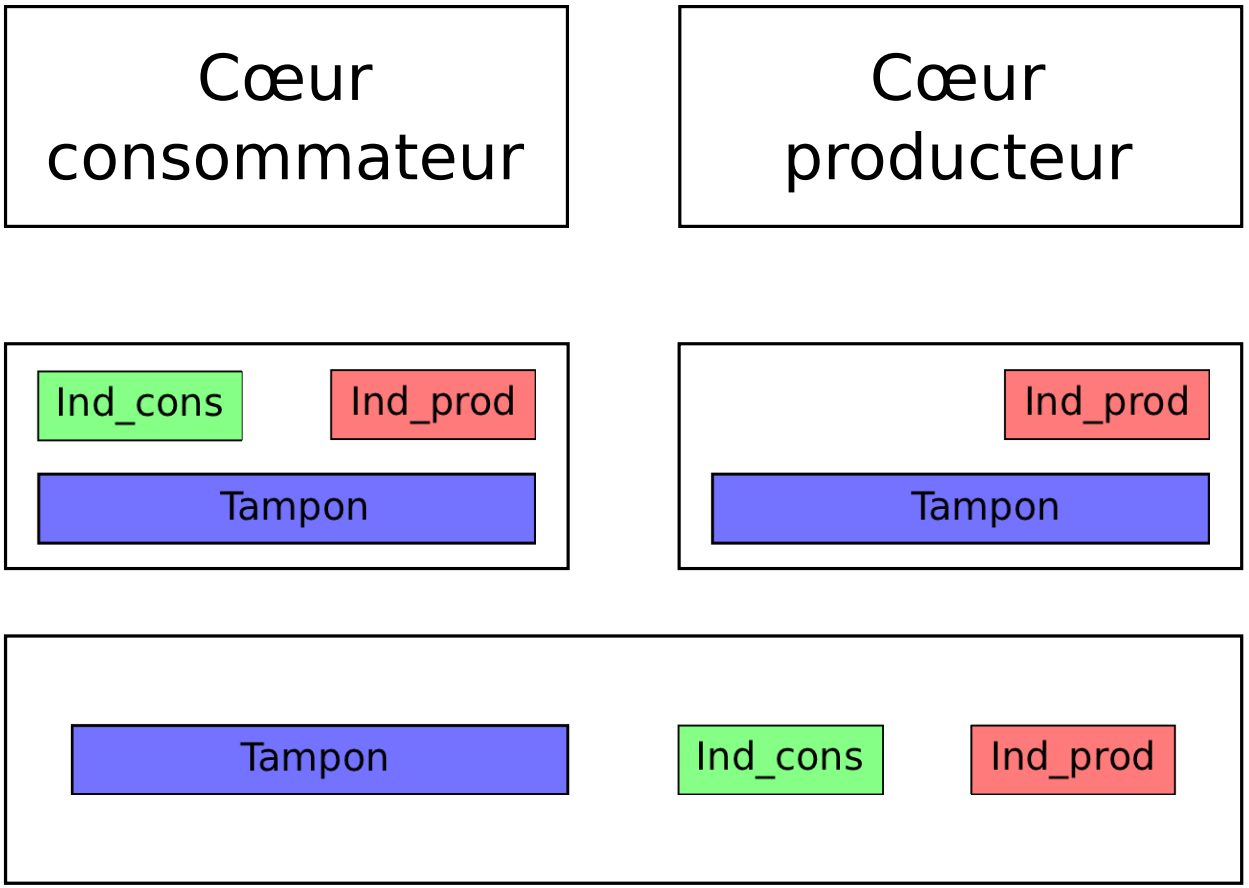



## Algorithm sans verrou

Le producteur et consumateur ne doivent pas se dépasser :

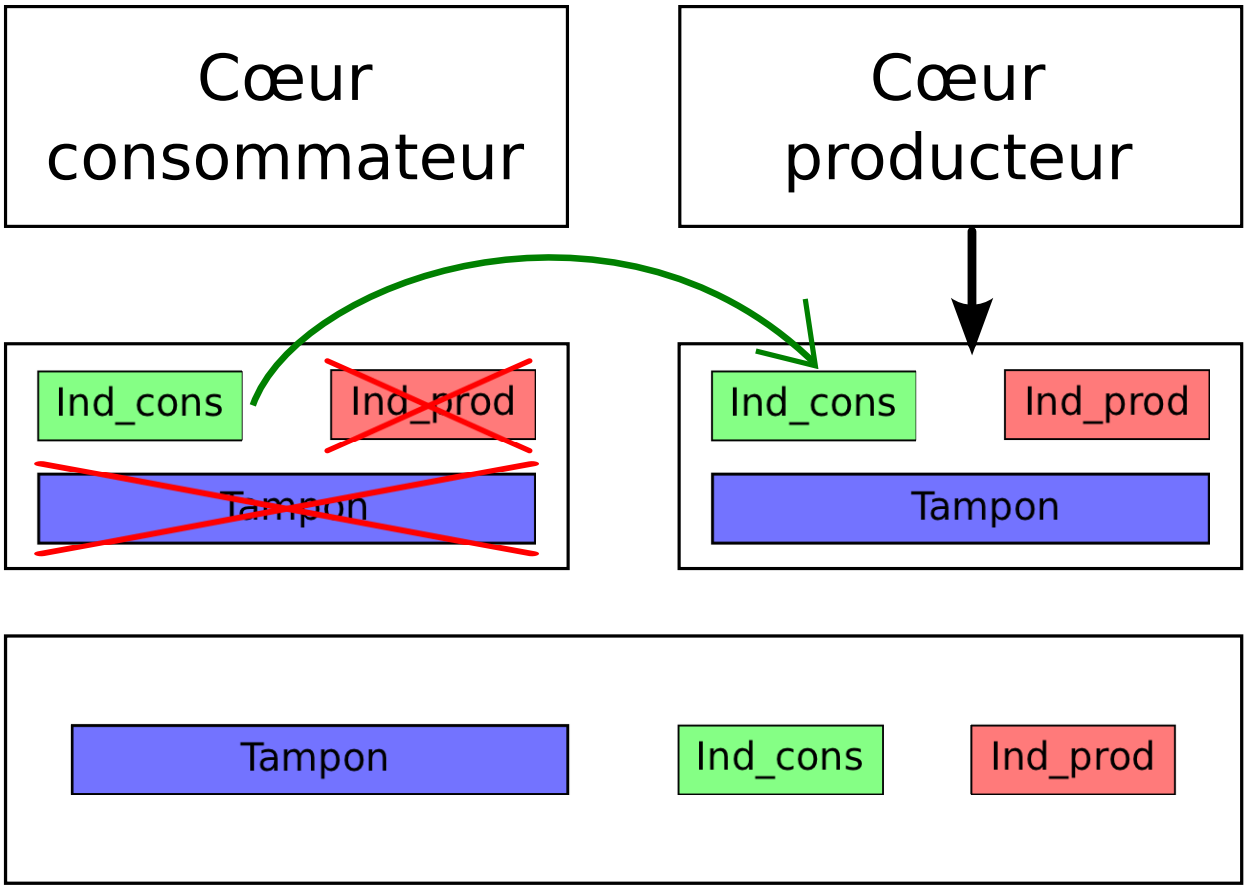
- Un unique écrivain pour chaque variable partagée
- Indices incrémentés après la production/consommation
- Pas de dépassement :
  - Consommateur : $prod\_idx \neq cons\_idx$
  - Producteur : $cons\_idx \neq \text{NEXT}(prod\_idx)$
    $\Rightarrow$ Une entrée est perdue

# File CSV de Lamport : comportement du cache



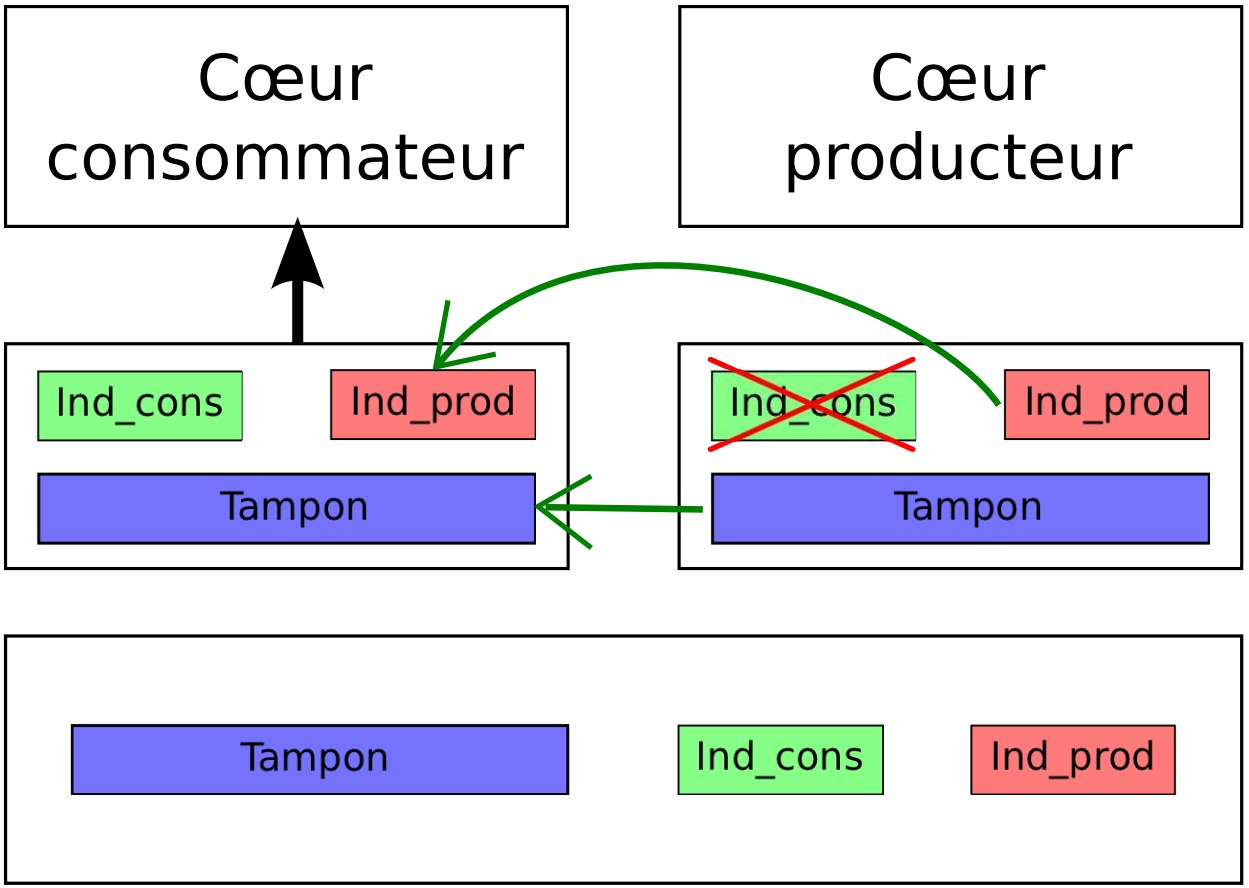

- Lecture depuis un cœur distant : 0
- Invalidations : 0

# File CSV de Lamport : comportement du cache




- Lecture depuis un cœur distant : 1
- Invalidations : 2

# File CSV de Lamport : comportement du cache




- Lecture depuis un cœur distant : 3
- Invalidations : 3

# File CSV de Lamport : ralentissements

Cas d'utilisation : production et consommation en parallèle

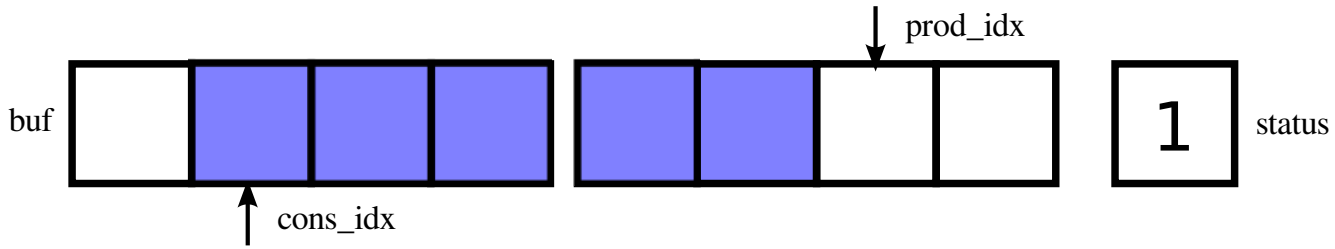
Rappel : parallelisme pipeline = production et consommation en parallèle

*Occurrences des événements causant des ralentissements*

| **Métrique** | **File CSV de Lamport** |
|---|---|
| Variables partagées | 3 |
| Freq. modifications | à chaque donnée échangée |
| Freq. lectures distantes | à chaque donnée échangée |

# Plan

# BatchQueue : structures



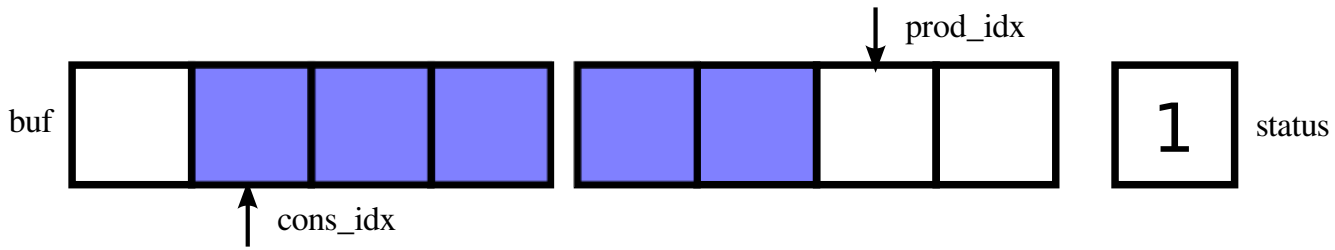

**2 variables partagées** :

- *buf* : Tampon d'échange entre producteur et consommateur
- *status* : bit de synchro entre le producteur et le consommateur

**2 variables privées** :

- *prod_idx* : Indice de la prochaine entrée vide du tampon
- *cons_idx* : Indice de la prochaine entrée pleine du tampon

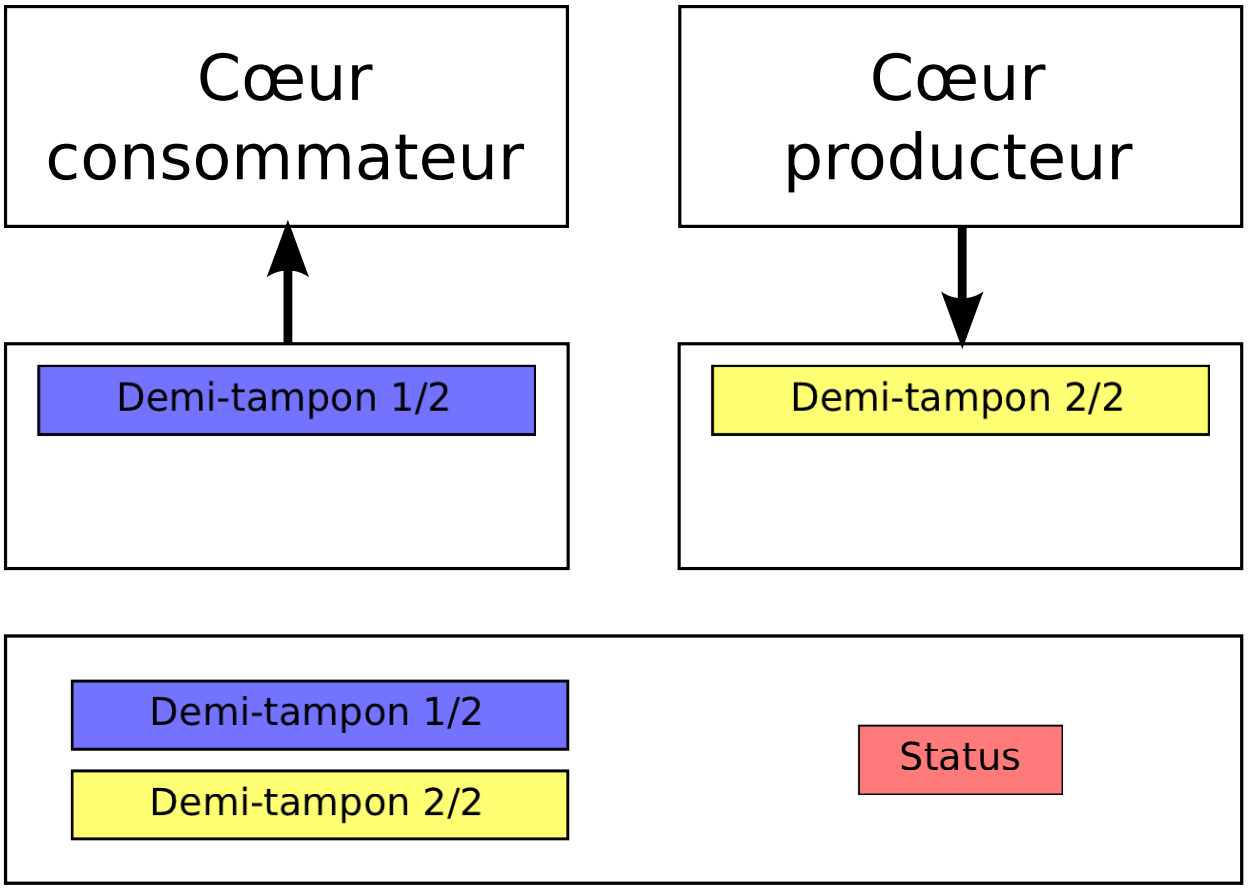
# BatchQueue : principe




## Algorithme sans verrou adapté au cache matériel

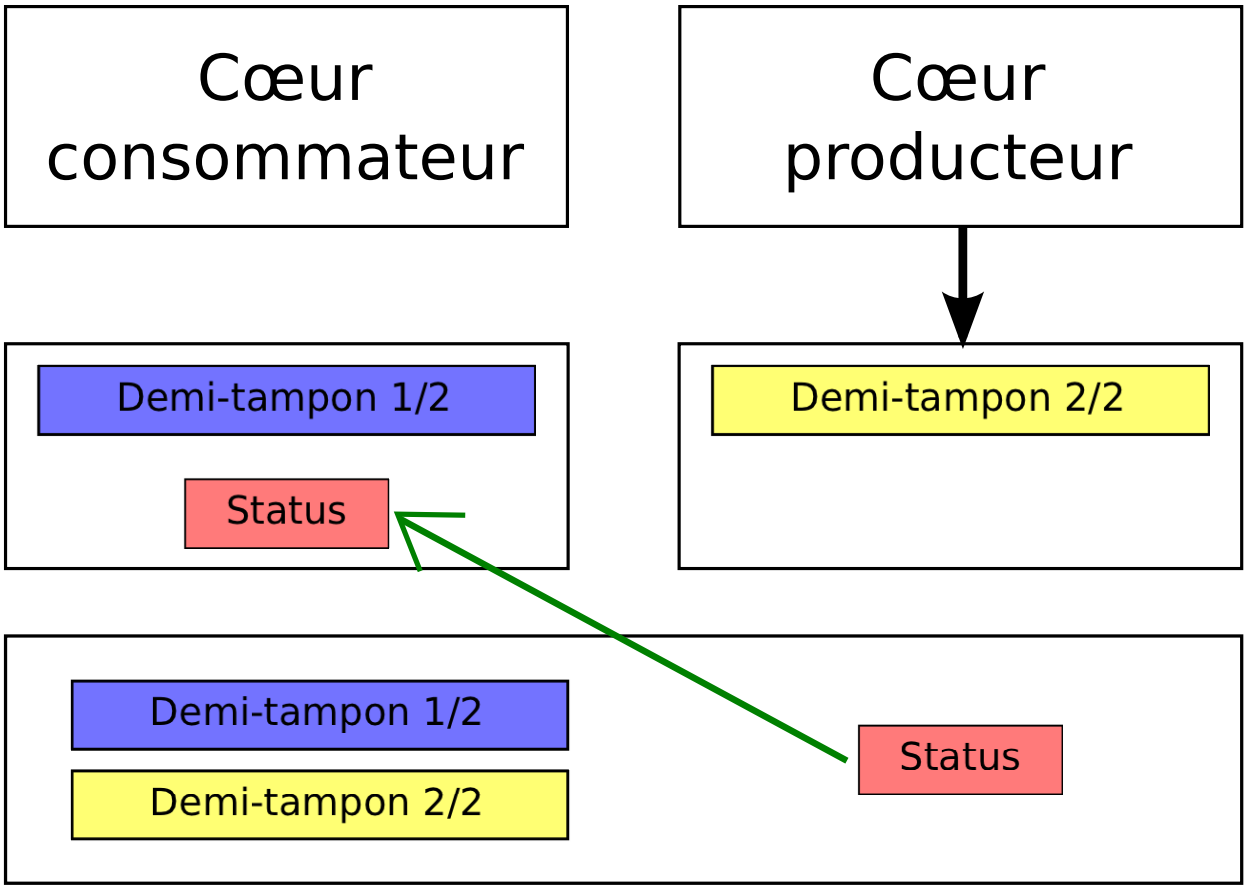
- 2 demi-tampons résidant dans des lignes de cache separées
- Aucun accès concurrent aux demi-tampons $\Rightarrow$ tamporisation
- Producteur passe au demi-tampon suivant si *status* vaut 0
- Consommateur passe au demi-tampon suivant si *status* vaut 1
  $\Rightarrow$ *status* inversé quand une ligne de cache a été traitée
  $\Rightarrow$ demi-tampons échangés quand *status* a été inversé 2 fois

# BatchQueue : comportement du cache



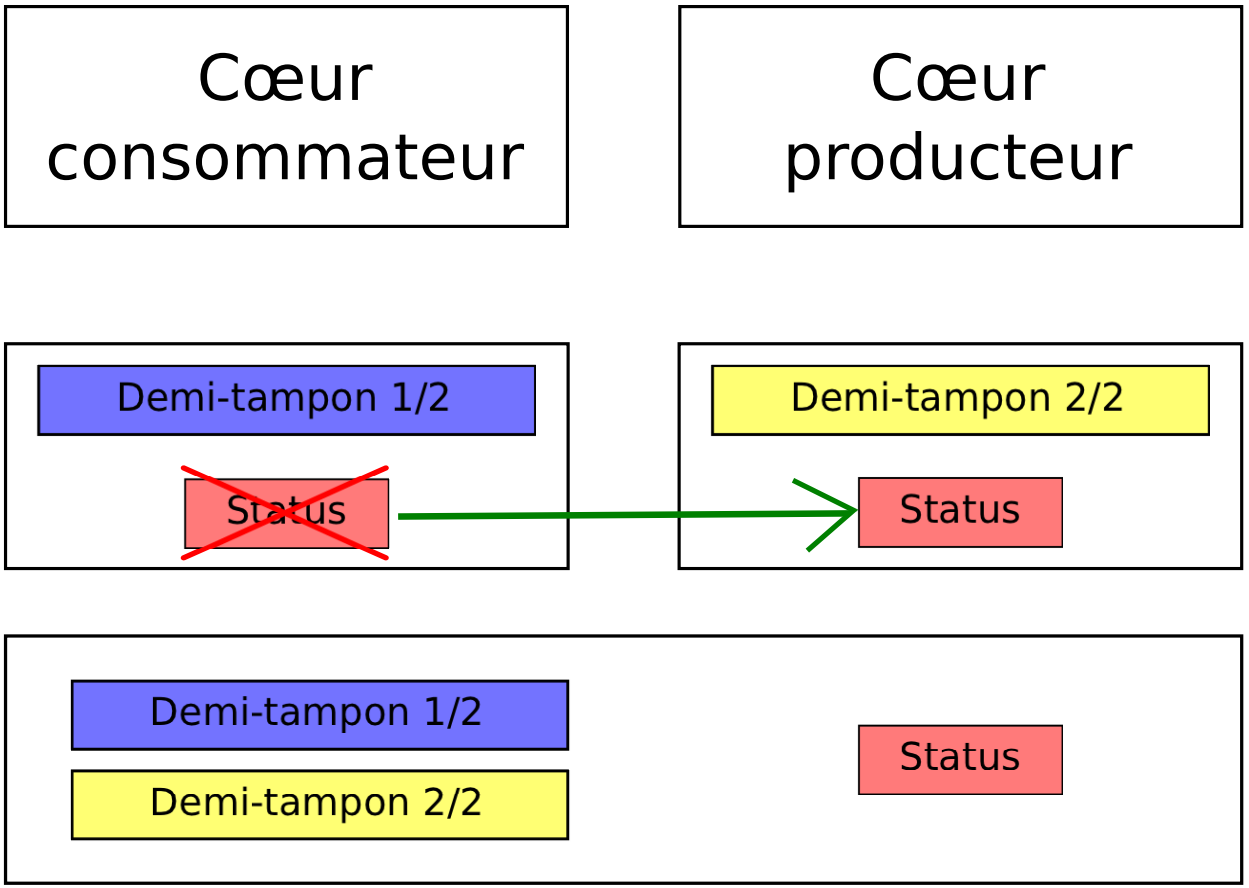

- Lecture depuis un cœur distant : 0
- Invalidations : 0

# BatchQueue : comportement du cache



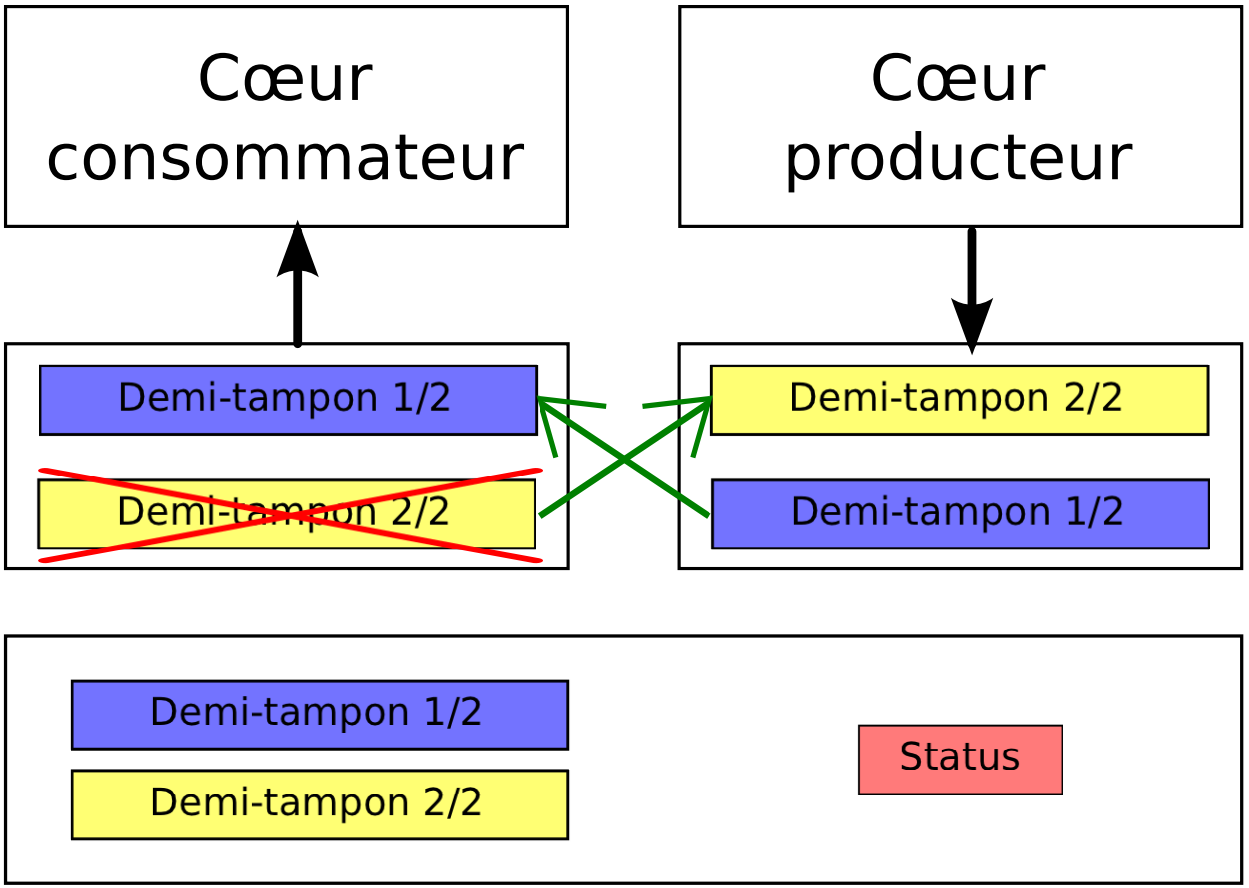

- Lecture depuis un cœur distant : 1
- Invalidations : 0

# BatchQueue : comportement du cache




- Lecture depuis un cœur distant : 2
- Invalidations : 1

# BatchQueue : comportement du cache




- Lecture depuis un cœur distant : 4
- Invalidations : 2

# BatchQueue : ralentissements

Cas d'utilisation : production et consommation en parallèle

Rappel : parallelisme pipeline = production et consommation en parallèle

*Occurrences des événements causant des ralentissements*

| **Métrique** | **File CSV de Lamport** | **BatchQueue** |
|---|---|---|
| Variables partagées | 3 | 2 |
| Freq. modifications | donnée | ligne de cache |
| Freq. lectures distantes | donnée | ligne de cache |

**Cas d'utilisation du parallélisme pipeline**

- Taille des lignes de cache augmente avec le temps
- Lignes de cache + grandes ⇒ BatchQueue + efficace

⇒ BatchQueue deviendra de plus en plus intéressant

# Plan

# Description des tests

L'évaluation consiste en 2 tests :

- Test avec communication intensive (*aucun*)
- Test avec utilisation intensive du CPU (*matrice*)

## Test avec communication intensive

- 160 millions mots mémoire envoyés depuis un cœur à un autre
- aucune action entre 2 envois

  $\Rightarrow$ **But** : mesurer le débit maximum atteignable

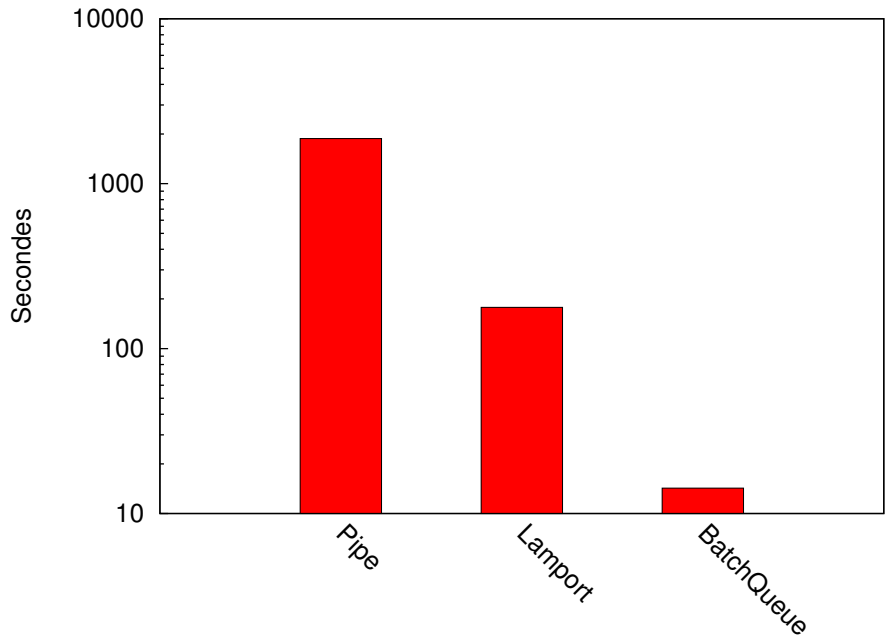
## Test avec utilisation intensive du CPU

- 160 millions mots mémoire envoyés depuis un cœur à un autre
- Multiplication de matrice effectuée entre 2 envois

  $\Rightarrow$ **But** : Mesurer le débit avec le cache L1 utilisé de façon intensive par une application externe

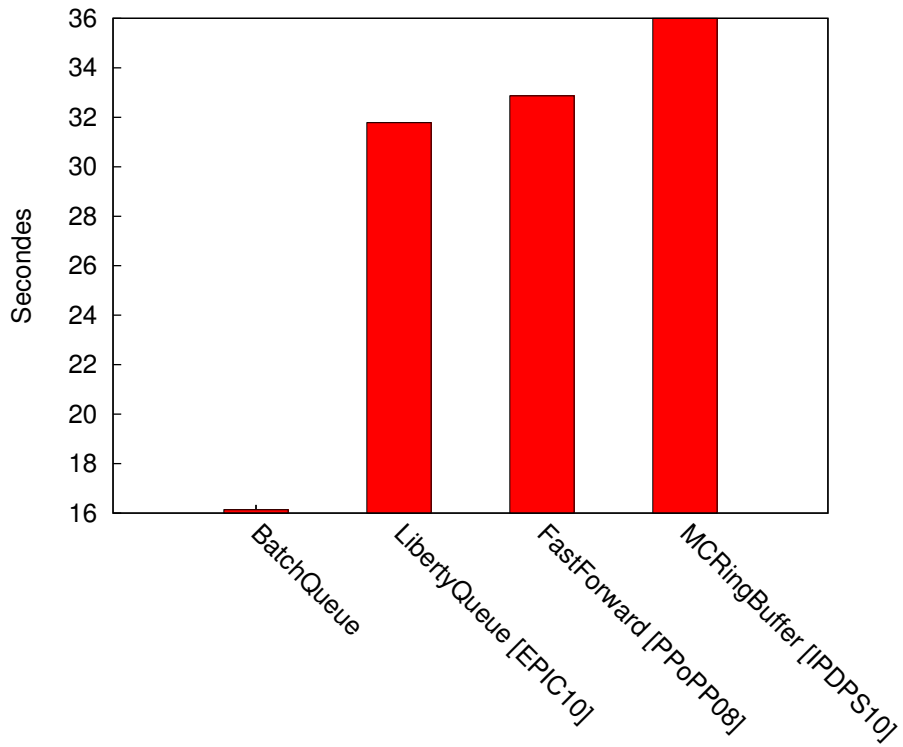
# BatchQueue : évaluation (1/3)

**Attention** : Utilisation d'une échelle logarithmique




Performance des différents mécanismes de communication inter-cœurs

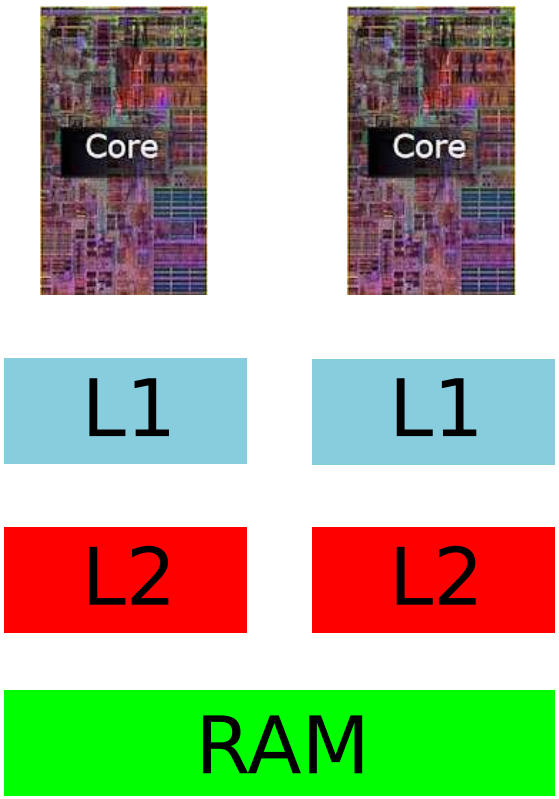
# BatchQueue : évaluation (2/3)



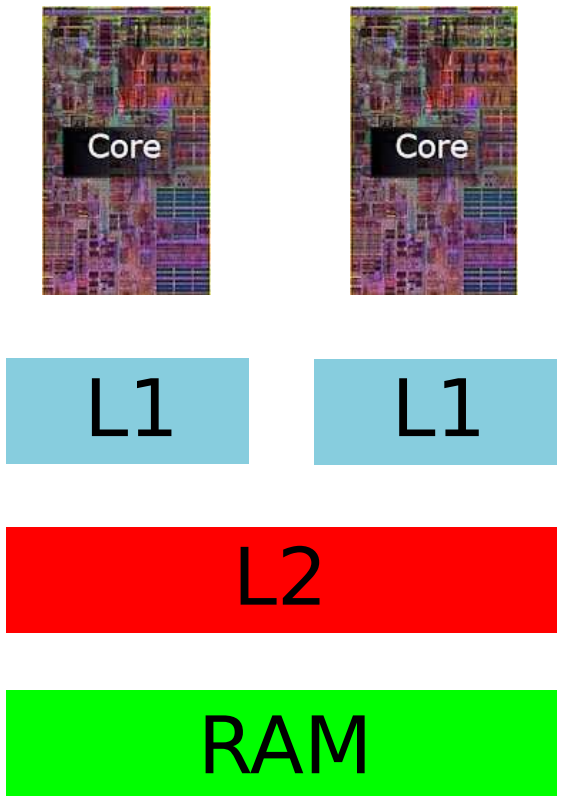

Comparaison en mode pipeline de BatchQueue avec l'état de l'art

# Architectures de caches matériels

2 configurations de caches matériels sur la même machine
(2 processeurs Intel X5472 contenant quatre cœurs)



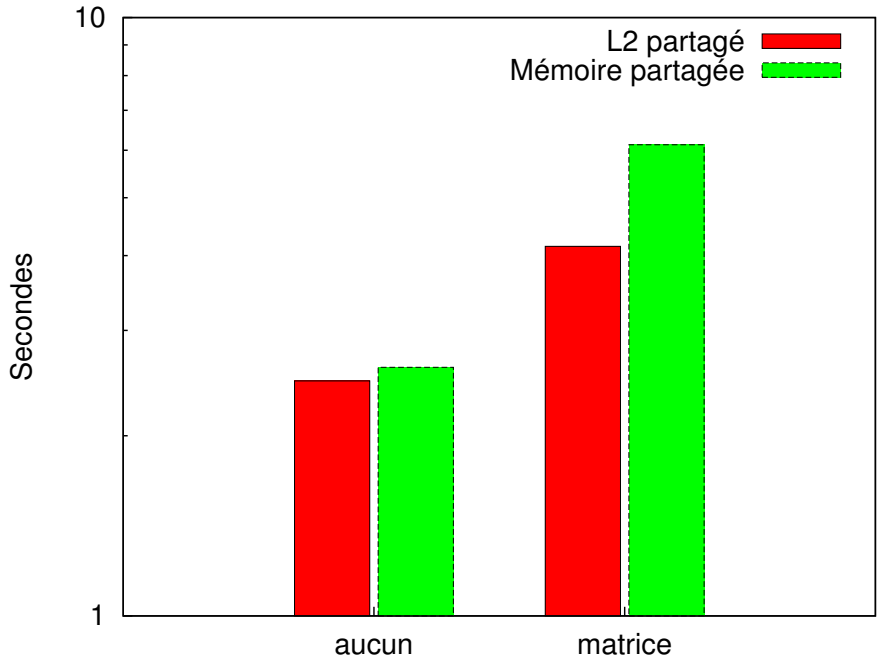

Pas de cache matériel partagé




Cache matériel partagé

# BatchQueue : évaluation (3/3)

**Warning** : Utilisation d'une échelle logarithmique




Influence du partage de cache sur BatchQueue

# Conclusion

## Communication cœur à cœur rapide

BatchQueue est adapté au parallélisme pipeline car il fournit :

- une communication cœur à cœur rapide
  $\Rightarrow$ 2 fois plus rapide que LibertyQueue (état de l'art)
- une prise en compte du préchargement des lignes de cache
  $\Rightarrow$ réduit encore le nombre de défauts de cache
- un surcoût mémoire faible
  $\Rightarrow$ Seulement un bit supplémentaire par file

## Perspectives

- Evaluer BatchQueue sur une application existante
  $\Rightarrow$ Par exemple : instrumentation sur un cœur dédié
- Gestion de plusieurs consommateurs (diffusion)

# Des questions ?


J. Giacomoni, T. Mosely, and M. Vachharajani.
Fastforward for efficient pipeline parallelism : A cache-optimized concurrent lock-free queue.
In *In PPoPP '08 : Proceedings of the The 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM Press, 2008.


T.B. Jablin, Y. Zhang, J.A. Jablin, J. Huang, H. Kim, and D.I. August.
Liberty queues for epic architectures.
In *Proceedings of the Eigth Workshop on Explicitly Parallel Instruction Computer Architectures and Compiler Technology (EPIC)*, 2010.


Leslie Lamport.
Specifying concurrent program modules.
*ACM Trans. Program. Lang. Syst.*, 5(2) :190–222, 1983.


P.P.C. Lee, T. Bu, and G. Chandranmenon.

A Lock-Free, Cache-Efficient Multi-Core Synchronization Mechanism for Line-Rate Network Traffic Monitoring.
In *IPDPS '10 : Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium*, 2010.


Cheng Wang, Ho-seop Kim, Youfeng Wu, and Victor Ying.
Compiler-managed software-based redundant multi-threading for transient fault detection.
In *CGO '07 : Proceedings of the International Symposium on Code Generation and Optimization*, pages 244–258, Washington, DC, USA, 2007. IEEE Computer Society.