

Comportement et exploitation de la cache en multiprogrammation

Félix-Antoine Ouellet

Université de Sherbrooke

20 novembre 2014

- 1 Motivation
- 2 Fonctionnement de la cache
- 3 Modèle insensible à la cache
- 4 Conclusion

Plan

- 1 Motivation
- 2 Fonctionnement de la cache
- 3 Modèle insensible à la cache
- 4 Conclusion

Un simple problème

Étape 1

Générer aléatoirement N entiers et les insérer dans une séquence de sorte qu'ils soient triés en ordre croissant.

Par exemple, 5 1 4 2 donne:

- 5
- 1 5
- 1 4 5
- 1 2 4 5

Un simple problème

Étape 2

Enlever les éléments de la séquence 1 à 1, et ce, de manière aléatoire.

Par exemple, 1 2 0 0 donne:

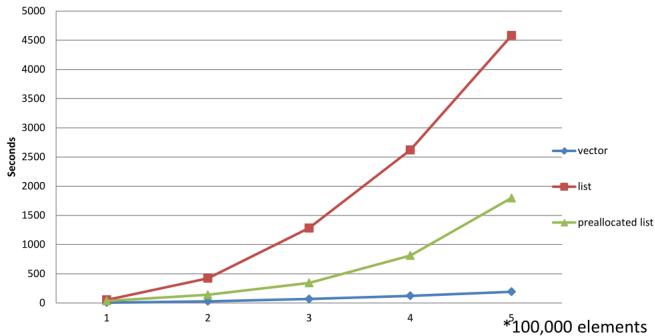
- 1 2 4 5
- 1 4 5
- 1 4
- 4

Un simple problème

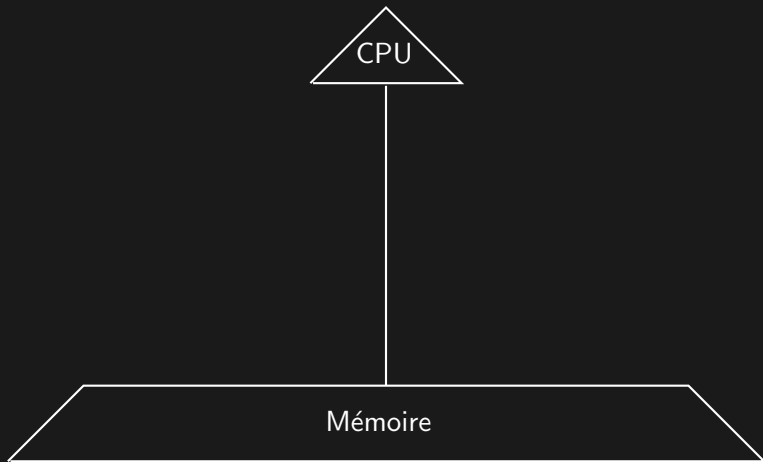
Résultats

Vector vs. List

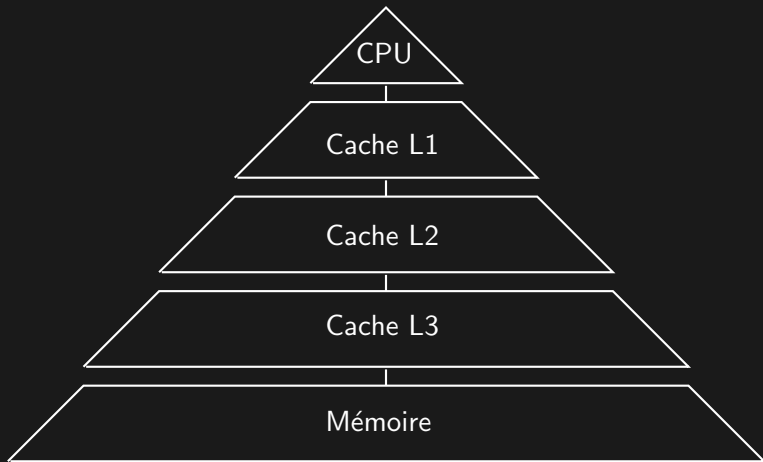
sequence test



Modèle académique



Réalité



Architecture multi-cœurs

- Plus de cœurs veut souvent dire réduction de la taille de cache par cœurs
- Mauvaise gestion de la cache peut faire perdre tous les gains potentiels du parallélisme

Plan

- 1 Motivation
- 2 Fonctionnement de la cache
 - Concepts de base
 - Cohérence
- 3 Modèle insensible à la cache
- 4 Conclusion

Cache

Illustration

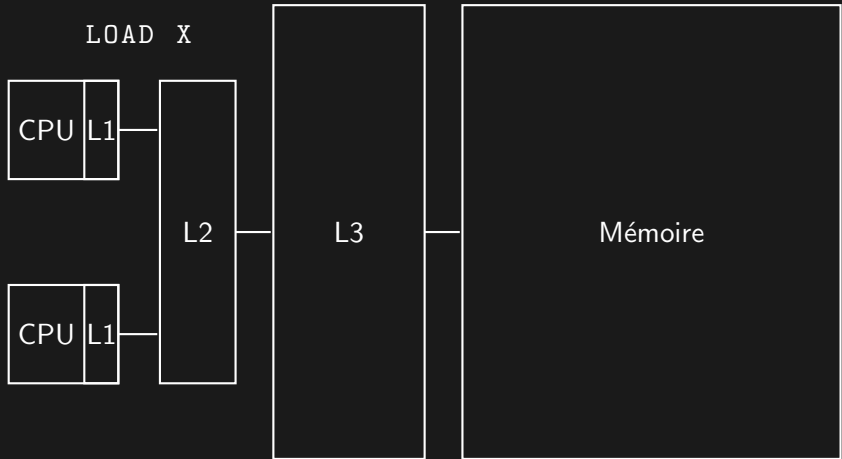


- Comporte B blocs (*cache lines*)
- Taille M
- Chaque bloc à une taille B/M

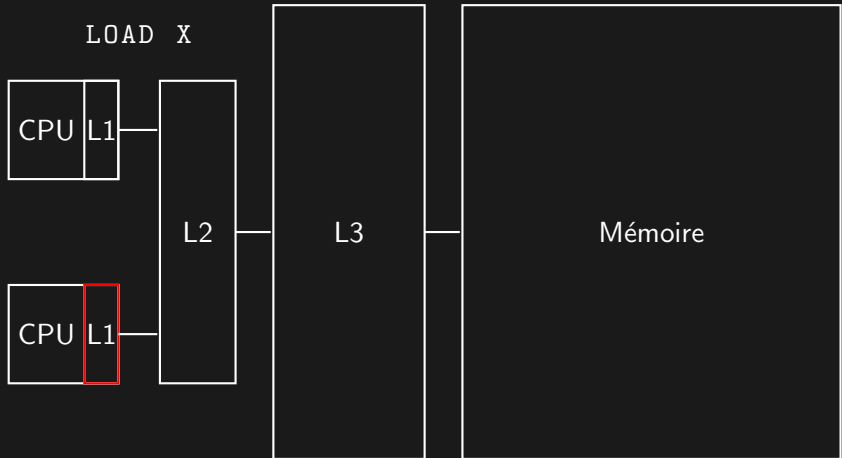
Accès à la mémoire

- En général, les processeurs ne peuvent accéder directement à la mémoire
- Les accès mémoire se font au travers d'une hiérarchie de caches
- Les lectures se font par blocs (*cache lines*)

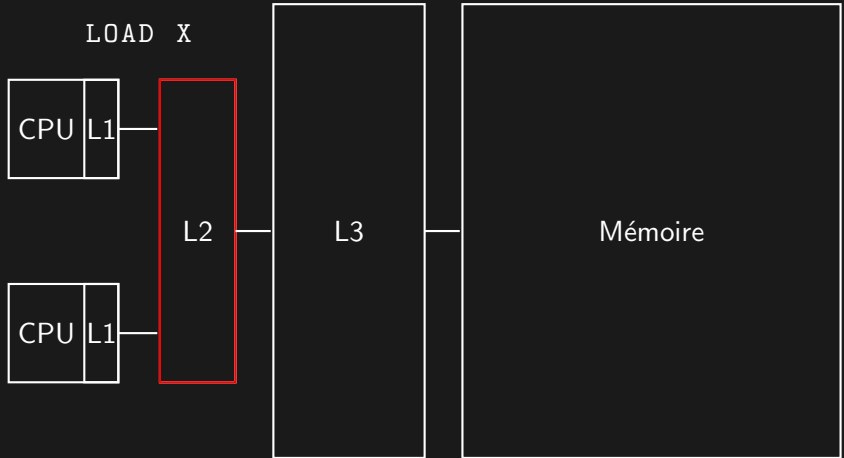
Lecture



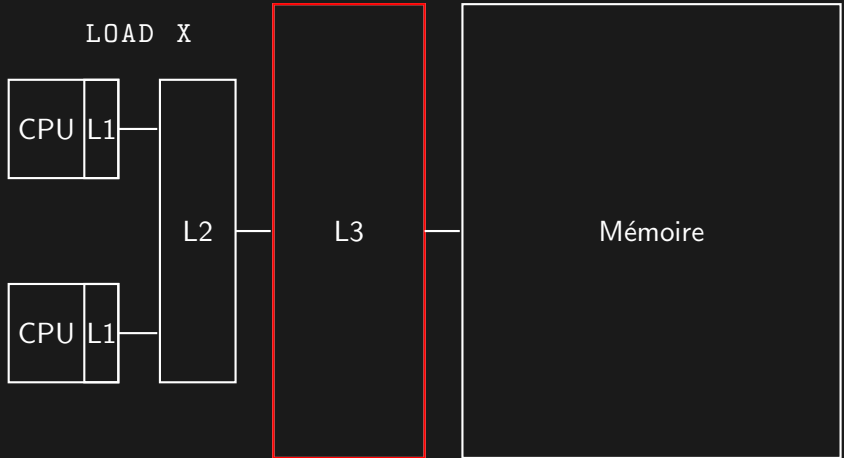
Lecture



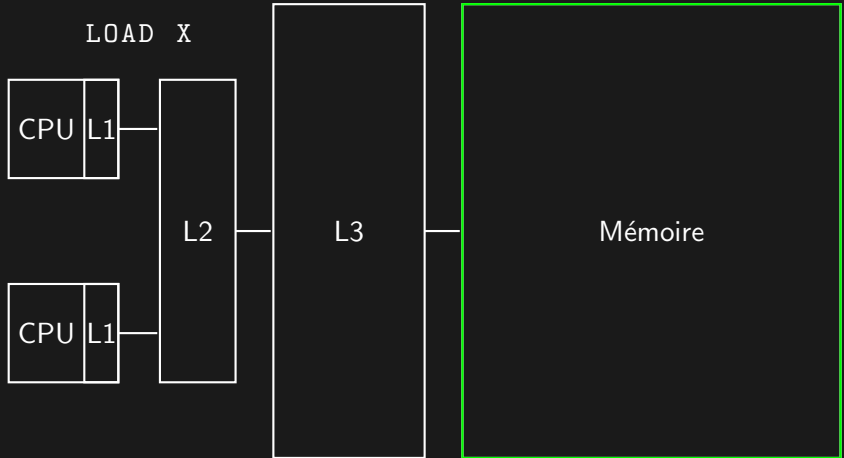
Lecture



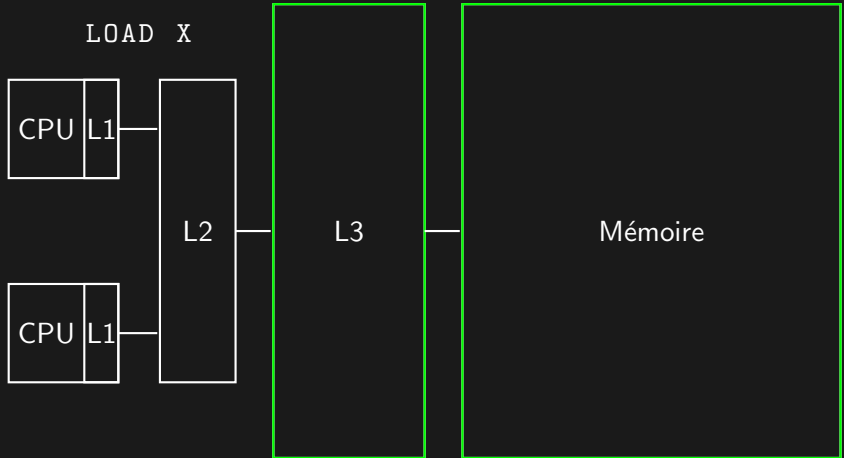
Lecture



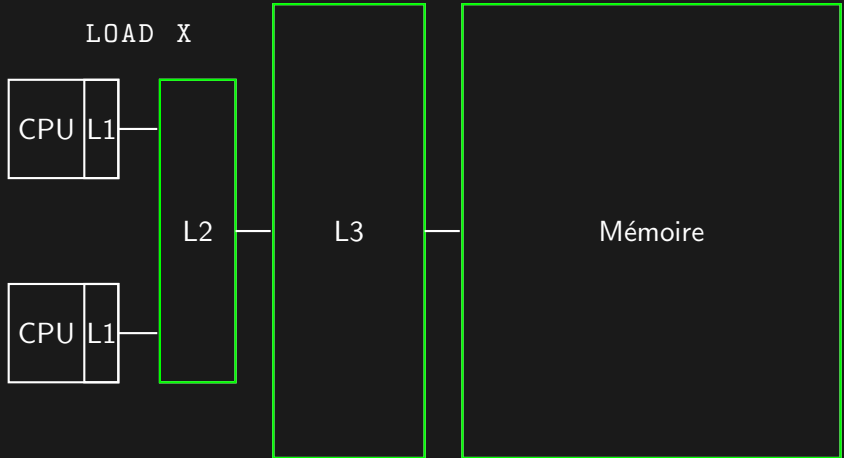
Lecture



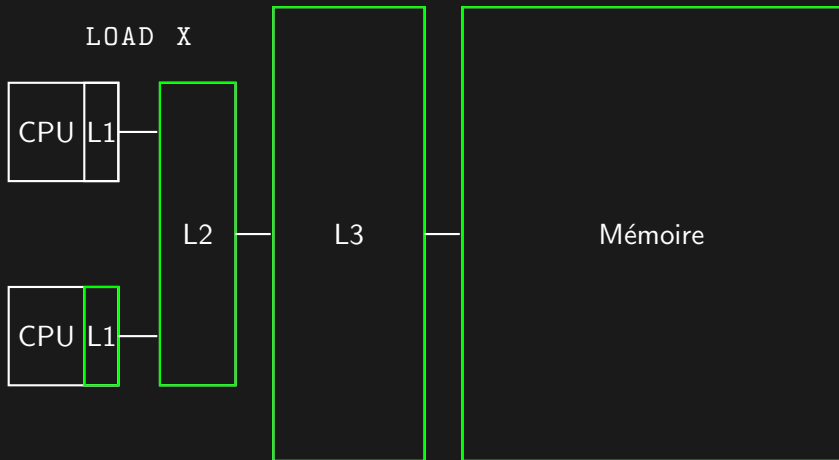
Lecture



Lecture



Lecture



Écriture

Deux approches:

- *Write-through*
- *Write-back*

Write-through

- Le contenu de la cache et de la mémoire sont mis à jour simultanément
- Le contenu de chaque cache est identique au contenu de la mémoire en tout temps
- Minimise la perte de données

Write-back

- La mise à jour de la mémoire est différée à des moments précis
 - Exemple: Lecture de l'emplacement mémoire, *cache line* sur le point de se faire évincer
- Offre des gains de performance en vitesse

Problème

Que se passe-t-il dans un contexte parallèle?

Protocole de cohérence

Assurer que le contenu de multiples caches demeurent cohérent

- Toutes les écritures sont éventuellement vues par une lecture
- Ordonnancement des écritures

Deux grandes familles de protocole

- Cohérence par répertoire
- Cohérence par espionnage

Cohérence par répertoire

- Répertoire central conservant des informations sur ce qui est partagé
- Messages inter-processeurs pour faire la requête de données
- *Scale* mieux

Cohérence par espionnage

Concept

- Tout les processeurs espionnent le bus
- Une seule cache effectue une lecture ou une écriture en mémoire dans un cycle donné
- Que faire avec écriture de type *write-back*?

Protocole MESI

Définitions

4 états possibles:

M → Modifiée: Copie modifiée d'une valeur en mémoire

E → Exclusive: Copie propre d'une donnée en mémoire présente uniquement dans la cache courante

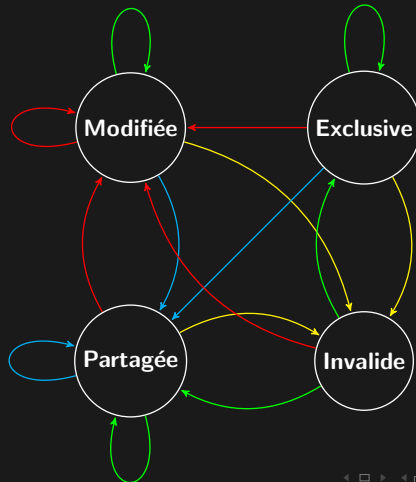
S → Partagée: Copie propre d'une donnée en mémoire que plusieurs caches peuvent posséder

I → Invalide: Ne peut être utilisée

Protocole MESI

Illustration

Écriture locale
Lecture locale
Lecture distante
Écriture distante

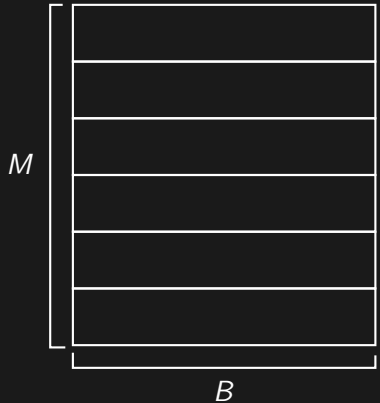


Plan

- 1 Motivation
- 2 Fonctionnement de la cache
- 3 Modèle insensible à la cache**
 - Concept
 - Algorithmes
- 4 Conclusion

Concept

- Conscience de l'existence d'une cache
- Inconscience des détails spécifiques
- Paramétrés par M et B
- Algorithmes souvent récurifs (diviser pour régner)



Élimination de la récursion

Les compilateurs modernes comprennent la récursivité et peuvent l'optimiser dans certains cas. Par exemple:

```
int factorial(int x) {  
    if(x > 1)  
        return x * factorial(x-1);  
    else return 1;  
}
```

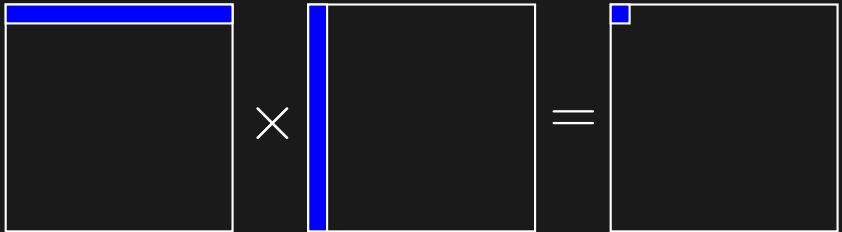


```
int factorial(int x) {  
    int result = 1;  
    while(x > 1)  
        result *= x--;  
    return result;  
}
```


Multiplication de matrices

Approche traditionnelle

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
    for (int k = 0; k < N; ++k)
      C[i][j] += A[i][k] * B[k][j];
```



Multiplication de matrices

Approche optimisée

```
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++i)
        B[i][j] = B[j][i];
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++i)
        for (int k = 0; k < N; ++k)
            C[i][j] += A[i][k] * B[j][k];
```



Multiplication de matrices

Approche insensible à la cache

```
for(int i = 0; i < N; ++i)
    for(int j = 0; j < N; ++i)
        for(int k = 0; k < N; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

Comparaison

Plan

- 1 Motivation
- 2 Fonctionnement de la cache
- 3 Modèle insensible à la cache
- 4 Conclusion

Conclusion

- Les meilleures performances proviennent souvent d'un programme conscient de la cache.

Conclusion

- Les meilleures performances proviennent souvent d'un programme conscient de la cache.
- Une façon d'exploiter la localité est d'utiliser des algorithmes et des structures de données insensibles à la cache.

Conclusion

- Les meilleures performances proviennent souvent d'un programme conscient de la cache.
- Une façon d'exploiter la localité est d'utiliser des algorithmes et des structures de données insensibles à la cache.
- Les algorithmes insensibles à la cache offrent des opportunités de parallélisation.