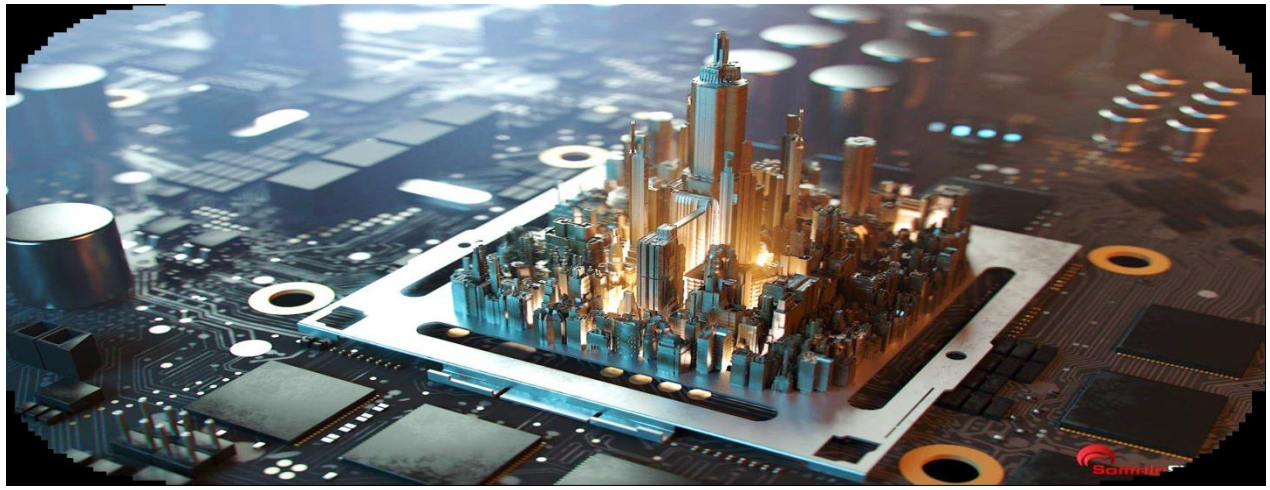
Université A. Mira Bejaia
Faculté des sciences exactess
Département d'informatique

Cours 5: Architectures des processeurs récents



Niveau: Ingénieur I année
Présentée par: Dr. SAAD Narimane

2024/2025

Introduction

- Le domaine du traitement parallèle concerne les méthodes **architecturales et algorithmiques** permettant d'améliorer les performances des ordinateurs numériques ou d'autres critères tels que la rentabilité et la fiabilité grâce à diverses formes de concurrence
- Résoudre rapidement des problèmes (le plus souvent numérique) **coûteux** en matière de temps de calcul : Simulation numérique, cryptographie, imagerie, SGBD,…
- Résoudre plus rapidement un problème donné, une idée naturelle consiste à faire coopérer simultanément Multi-cœurs au lieu plusieurs agents à sa résolution, qui travailleront donc **en parallèle**.
- Les mémoires caches sont devenues plus volumineuses en raison notamment de l'apparition des **microprocesseurs multi-cœurs**, elles ont été disponibles sur la carte mère, puis ont été progressivement intégrées au microprocesseur.

1. Multi-cœurs et parallélisme

1.1. Qu'est-ce le parallélisme?

- **Machine parallèle**, une machine qui possède **un ensemble de processeurs** (une architecture parallèle)qui **coopèrent et communiquent** ensemble et qui concourent dans **le but d'exécuter un programme parallèle**
- Un **programme parallèle** est un **ensemble fini de processus séquentiel**s communiquant entre eux par échange de messages.
- **La performance** d'une architecture parallèle est la **combinaison des performances** de ses ressources tels que le débit, leur coût de mise en œuvre, etc.

✓ Concept de parallélisme et multi-cœur :

- **Le parallélisme d'une application** consiste à faire **exécuter simultanément** plusieurs parties indépendantes de cette application sur plusieurs ressources matérielles.
- Selon les structures des applications, il existe deux formes de parallélismes :

✓ Parallélisme de contrôle

✓ Parallélisme de données

➢ Parallélisme de données

- Pour lequel la même opération est réalisée simultanément par des nombreux processeurs sur des données différentes.

```
Pour i = 1 à N faire
  Pour J = 1 à M faire
    C[i,j] ← A[i,j] + B[i,j]
  Fin_pour
Fin_pour
```

- Le parallélisme des données dépend directement de la taille des structures de données manipulées.
- ***Exemple d'application de parallélisme* de données:**

✓ Applications numériques

✓ Les traitements de base de données (avec des milliers d'entées)

✓ Le traitement du signal et de l'image (matrices de plusieurs milliers d'éléments : pixels).

- ***Parallélisme* de données:** Chaque processeur devra donc traiter plusieurs données dans la même opération.

Parallélisme de contrôle

- Pour lequel des opérations différentes sont réalisées simultanément.
- Ce parallélisme peut provenir de l'existence dans le programme des fonctions indépendantes (parties indépendantes) ou d'opérations indépendantes dans une suite d'opérations, qui est la source de parallélisme.
- Ce parallélisme ne dépend donc pas des données mais de la structure du programme à exécuter.
- ***Les systèmes d'exploitation des ordinateurs modernes sont des exemples de ce type de parallélisme: Ex le pipeline***

Les plateformes de calcul Parallèle

- Le calcul parallèle ne peut avoir lieu que sur des **plateformes de calcul Parallèle:**
- **Les machines multi-coeurs** :Processeurs possédant plusieurs coeurs physiques (unités de calcul) gravés au sein de la même puce, largement disponibles à des coûts de plus en plus faibles **Exemple : (Intel® Xeon Phi™ 7290** 16GB, 1.50 GHz, 72 coeurs)
- **Les supercalculateurs** : Ordinateurs conçus pour atteindre les plus hautes performances possibles avec les technologies disponibles au moment de leur conception. Ces machines ne sont généralement pas à la portée de tout le monde.
- **Autres plateformes de calcul parallèle :** en faisant collaborer des ordinateurs ordinaires interconnectés, telles que :

✓ **Les clusters** (ensemble d'ordinateurs reliées au sein d'un réseau local).

✓ **Les grilles de calcul** qui sont des agrégations de ressources de calcul hétérogènes et distribuées sur des sites distants.

✓ **Cloud Computing**

✓ **Les Processeurs graphiques (GPGPU) :** les processeurs graphiques modernes contiennent des milliers de coeurs (exp : 5120 coeurs dans la Tesla V100)

Pourquoi le parallélismes ?

- La réponse à cette question se résume dans les trois points suivants :

✓ **Les besoins croissants en haute performances,**

✓ **L'évolution de la technologie,**

✓ **L'évolution des applications.**

- Deux facteurs majeurs contribuent à l'amélioration des performances des processeurs modernes:

➢ La technologie rapide des circuits et les caractéristiques architecturales, des caches de plus en plus grands, des bus multiples, le pipelining,

➢ Le fait que les calculateurs avec une seule unité centrale de traitement (*central processing unit* CPU) sont souvent incapables de répondre à certains besoins comme : la **simulation des systèmes complexes**

❖ L'une des solutions à ce besoin en performances est la création des architectures dans lesquelles plusieurs CPUs fonctionnent dans le but de résoudre une application donnée.

❑ Les caractéristiques des architectures parallèles

- Les architectures parallèles sont caractérisées par une ou plusieurs caractéristiques parmi les suivantes :

✓ **Espace mémoire étendu.**

✓ **Plusieurs processeurs.**

✓ **Traitement partiel indépendant.**

✓ **Accélération des calculs complexes ou coûteux en temps d'occupation CPU.**

✓ **Calcul parallèle répétitif sur un large ensemble de données structuré**.

❑ Les défis des architectures parallèles

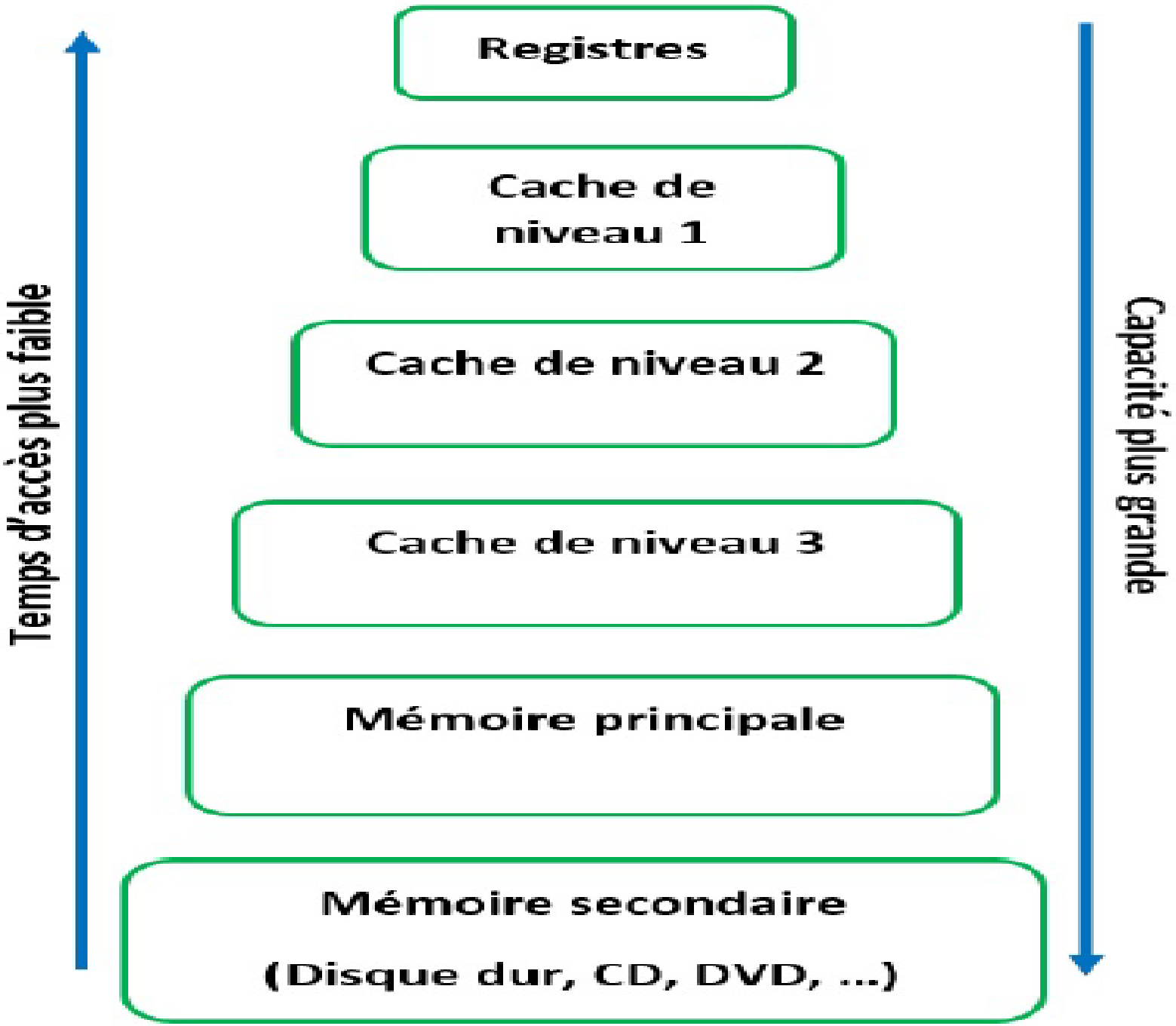
- Alors que les **performances architectures parallèles sont limités** lorsqu'il existe dans le programme parallèle :
- **Des dépendances fonctionnelles :** que ce soit des dépendances de données, ou des dépendances de contrôle.
- **Des dépendances de ressources :** si le nombre de processeurs est insuffisant par rapport le nombre des taches parallèles indépendantes.
- **Des délais de communication élevés :** les communications peuvent dan ce cas augmenter le temps d'exécution et donc il ne sera pas avantageux de paralléliser une application dans ce cas.

2. Mémoire cache et hiérarchie mémoire

- Le fait que la **mémoire principale** n'arrive plus à suivre le **rythme croissant** des fréquences d'horloge des processeurs. Alors, il faut d'intercaler entre la **mémoire principale** et **le processeur** une mémoire cache plus rapide
- **L'objectif des mémoires caches** est de permettre au processeur de mémoriser les informations les plus utilisée afin d'être en mesure de les récupérer le plus rapidement possible en cas de besoin.
- Le processeur n'a pas besoin d'aller chercher les informations auprès de la mémoire principale, **ce qui constitue un gain de temps très important**.
- **L'efficacité** d'un tel système dépend des caractéristiques de la mémoire cache, à savoir son efficacité et sa taille

Principe de la mémoire cache et hiérarchie mémoire

- Il est possible de **hiérarchiser la mémoire** en plaçant plusieurs autres niveaux de cache entre la mémoire principale et le processeur
- Dans les systèmes actuels, nous n'avons **pas une seule mémoire**, mais plutôt plusieurs mémoires organisées en hiérarchie :
- L'objectif des mémoires caches: doivent être rapides pour alimenter le processeur en instructions et en données, et de grande capacité pour stocker l'intégralité des informations disponibles
- Plus la mémoire et de grande capacité, plus son accès devient lent. Il faut donc utiliser **tous les types de mémoires** en hiérarchisant et **en répartissant les informations** en fonction de leur fréquence d'utilisation.


Registres
Cache de niveau 1
Cache de niveau 2
Cache de niveau 3
Mémoire principale
Mémoire secondaire
(Disque dur, CD, DVD, ...)
Temps d'accès plus faible
Capacité plus grande

Caractéristiques de la mémoire cache

Taille d'une mémoire cache

- Plus la mémoire cache est **de grande** capacité, plus le processeur a de chance d'y trouver de l'information **dont il cherche,** arriver à **des meilleurs performances**, on cherche toujours à avoir la mémoire cache avec **la plus grande capacité possible**.

Adressage

- Différents types d'adressage. C'est-à-dire, lorsqu'une information est transférée de la mémoire principale à la mémoire cache, il faut décider **où placer cette information**.

Algorithmes de remplacement

- Le processeur effectue une **opération de lecture** ou d'écriture dans une ligne qui n'existe pas dans le cache, Il existe principalement 5 méthodes :
- Aléatoire
- FIFO (First In First Out)
- LRU (Least Recently Used)
- LFU (Least Frequently Used)
- NMRU (Non-Most-Recently-Used)

➢ Politique de réécriture

- Outre les opérations de lecture en mémoire principale, le processeur effectue également des écritures, par exemple lorsqu'il modifie des données. Il existe deux méthodes pour la réécriture des données présentées ci-dessous:

✓ Cache write-through

- Écriture simultanée (write-through). C'est-à-dire, lorsqu'on écrit un mot d'une ligne du **cache**, on écrit **simultanément** le mot de la même ligne en **mémoire centrale.**

✓ Cache write-back

- Contrairement à la méthode write-through, la méthode write-back (à écriture différée) **ne modifie l'information en mémoire principale que lorsque la ligne correspondante disparaît du cache**. Cela permet de ne pas perdre trop de temps à accéder à la mémoire principale.

Gestion de la mémoire cache dans les architectures récentes

- La gestion de la mémoire cache est l'un des domaines où l'évolutivité présente des défis importants.
- La mise en cache était considérée comme un moyen simple d'améliorer les performances en stockant les données fréquemment consultées dans une zone d'accès rapide
- La **gestion de la mémoire cache** devient une tâche complexe, nécessitant des stratégies sophistiquées pour **garantir** à la fois les performances et la cohérence dans les systèmes distribués.

Cohérence entre les systèmes distribués

- Dans un système distribué, s'assurer que **tous les services accèdent de manière cohérente** aux données mises **en cache** est un défi important. Toute mise à jour du cache doit être propagée à tous les nœuds, ce qui peut entraîner des temps de latence et une certaine complexité.

Invalidation du cache

- La **détermination du moment** où il faut **invalider ou rafraîchir le cache** est un autre problème critique. Des données périmées peuvent entraîner des réponses incorrectes, tandis qu'une invalidation excessive peut dégrader les performances. Parmi les politiques les plus courantes, on peut citer la méthode LRU (Least Recently Used), la méthode FIFO (First In, First Out) et la méthode TTL (Time-To-Live), chacune d'entre elles présentant des avantages et des inconvénients.

Partage de la mémoire cache

- Lorsque **le volume de données augmente**, il devient nécessaire de **répartir le cache sur plusieurs nœuds**. Cependant, cela introduit une complexité dans la gestion et l'accès aux données partagées. Solutions pour surmonter les dilemmes de la gestion du cache

❑Les problèmes de la gestion des caches dans les systèmes évolutifs

❑ **Frameworks de gestion de cache distribué**

Un système de cache distribué consiste à **répartir l'infrastructure de mise en cache sur plusieurs nœuds** ou serveurs d'un réseau. Cette approche présente plusieurs avantages lorsqu'il s'agit de gérer les complexités de l'évolutivité :

- **Répartition de la charge ,Tolérance aux pannes, Évolutivité, Réduction de la latence, Cohérence et homogénéité,Architecture décentralisée**

❑ **Techniques d'invalidation des caches**

Ont essentielles au maintien de la cohérence des données dans un système de mise en cache, lorsqu'il s'agit de gérer les complexités . Parmi ces techniques:

- **Expiration basée sur le temps, Invalidation en fonction des, événements,Versioning, Mise en cache par lecture, Mise en cache en aval de l'écriture, Jetons ou clés, Invalidation basée sur des modèles, Diffusion globale des invalidations**

Protocoles de cohérence de la mémoire cache

- Les protocoles de cohérence de la mémoire cache sont des mécanismes essentiels utilisés dans les systèmes distribués pour garantir que les données mises en cache restent cohérentes entre les différents nœuds ou caches. Parmi ces protocoles de cohérence du cache :
- **Mise en cache par écriture et mise en cache en aval de l'écriture, Cohérence basée sur l'invalidation Écriture unique, lecture multiple (WORM), Validation en deux phases, Systèmes basés sur le quorum, Horloges vectorielles**

Monitoring et Metrics

- Ces outils peuvent aider à ajuster les paramètres et les politiques du cache en temps réel, garantissant ainsi des performances optimales, l'importance et de la mise en œuvre de la surveillance et des mesures dans un environnement évolutif :
- **Visibilité en temps réel, Identification des goulots d'étranglement, Planification de la scalabilité, Mécanismes d'alerte, Tracing distribué, Surveillance de l'expérience utilisateur, Mesures d'utilisation des ressources, Analyse historique, Optimisation des coûts, Tableaux de bord complets**

Conclusion

- Le domaine du traitement parallèle concerne les méthodes architecturales et algorithmiques permettant d'améliorer les performances des ordinateurs numériques ou d'autres critères tels que la rentabilité et la fiabilité grâce à diverses formes de concurrence, permettant d'obtenir de meilleures performances, ou un meilleur rapport coût-efficacité,

- La gestion de la mémoire cache est passée d'une simple amélioration des performances à un problème complexe aux multiples facettes. En s'appuyant sur des frameworks de gestion de cache distribués, en adoptant l'invalidation de cache pilotée par les événements, et en employant des stratégies de partage de cache efficaces, qui peuvent résoudre les probèmes de la gestion de cache dans les systèmes évolutifs.