Lien du powerpoint
https://www.canva.com/design/DAGiYDjqfyk/XhkWhdRzUqB0uW95io_pEQ/edit?utm_content=DAGiYDjqfyk&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

1) Introduction à l'optimisation de code

L'optimisation en informatique désigne un ensemble de techniques et de stratégies visant à améliorer l'efficacité des systèmes informatiques, des programmes et des processus. Elle peut être mise en place tant par les entreprises que par les particuliers pour améliorer leurs opérations, réduire leurs coûts, et rendre leurs activités plus efficaces. L'objectif principal est de maximiser les performances tout en minimisant les ressources utilisées.

Aujourd'hui, nous allons nous concentrer spécifiquement sur l'optimisation du code. L'optimisation du code fait référence à l'ensemble des pratiques qui visent à améliorer l'efficacité du code source d'un programme ou d'une bibliothèque logicielle. Cela peut concerner plusieurs aspects :

1. **Réduction du temps d'exécution** : L'un des objectifs majeurs de l'optimisation du code est de rendre un programme plus rapide. Cela peut se traduire par la réduction du temps nécessaire pour exécuter une fonction ou un algorithme.

2. **Réduction de l'espace mémoire** : Optimiser un programme peut aussi consister à réduire la quantité de mémoire ou de ressources système nécessaires pour le faire fonctionner, ce qui est crucial, notamment dans les environnements à ressources limitées.

3. **Amélioration de la consommation d'énergie** : Avec la montée en puissance des appareils mobiles et des systèmes embarqués, l'optimisation du code peut également viser à réduire la consommation d'énergie des programmes, ce qui est particulièrement important pour prolonger la durée de vie des batteries.

4. **Rationalisation des ressources** : Cela inclut la gestion des ressources telles que le processeur, la mémoire et les entrées/sorties de manière plus efficace, réduisant ainsi le coût d'exécution global du programme.

Mais pourquoi optimiser un code ?

Le code optimisé peut offrir un certain nombre d'avantages, notamment des performances améliorées, une utilisation réduite de la mémoire et un temps de traitement réduit.

En plus d'une vitesse et de performances améliorées, le code optimisé peut également entraîner une utilisation réduite de la mémoire. Lorsque le code est optimisé, il est souvent plus compact et utilise moins de mémoire que le code non optimisé. Cela peut libérer des ressources mémoire pour d'autres tâches et peut aider à réduire l'empreinte mémoire globale d'un programme logiciel.

Enfin, le code optimisé peut également entraîner une réduction du temps de traitement. Étant donné que le code optimisé est généralement plus efficace, son exécution peut prendre moins de temps. Cela peut constituer un avantage considérable, en particulier pour les programmes logiciels sensibles au temps ou gourmands en ressources.

2) Types d'optimisation

Il y a 3 types d'optimisation de code :

1) L'optimisation des performances qui permet :
    - D'améliorer la vitesse d'exécution du programme
    - De réduire le temps de réponse et d'optimiser l'utilisation des ressources CPU
    - D'assurer une meilleure scalabilité pour traiter un plus grand volume de données

Cela vise à rendre le code plus rapide et plus réactif. Elle est essentielle dans les applications qui nécessitent un traitement rapide des données, comme les jeux vidéo, les logiciels financiers ou les systèmes embarqués.

2) L'optimisation de la mémoire qui permet :
    - De réduire l'utilisation de la mémoire RAM
    - D'éviter les fuites de mémoire et les dépassements de capacité
    - D'améliorer l'efficacité des structures de données pour un stockage optimal

Elle est cruciale pour éviter les gaspillages de ressources et assurer un bon fonctionnement sur des machines ayant des contraintes mémoire, comme les systèmes embarqués ou les applications mobiles.

3) L'optimisation de la lisibilité et de la maintenabilité qui permet :
    - De rendre le code plus compréhensible et facile à modifier
    - De faciliter la collaboration entre développeurs
    - De réduire les erreurs et simplifier le débogage et l'évolution du projet

Un code bien écrit ne doit pas seulement être performant et efficace en mémoire, il doit aussi être compréhensible et facile à modifier. Une bonne lisibilité améliore la collaboration et réduit le risque d'erreurs lors des évolutions du programme.

3) Approche d’optimisation

Plusieurs approches existent pour optimiser un code :

1. **Optimisation au niveau algorithmique :**
   L’optimisation algorithmique consiste à choisir des algorithmes ayant une **complexité mathématique inférieure**. Par exemple, préférer un **algorithme de tri rapide** (complexité O(n log n)) au lieu d’un **tri à bulle** (complexité $O(n^2)$) permet d’améliorer significativement la performance pour des grandes quantités de données.
   Il est également important de choisir des **structures de données adaptées** (par exemple, utiliser des tables de hachage pour des recherches rapides ou des arbres équilibrés pour des insertions et suppressions efficaces), ce qui permet de réduire le temps de calcul ou la consommation de mémoire.

2. **Optimisation au niveau du langage de développement :**
   Au niveau du langage de programmation utilisé, il est crucial d'**ordonner les instructions de manière optimale**. Par exemple, éviter les appels de fonction redondants dans des boucles serrées ou réduire les accès à la mémoire peut faire une différence importante.
   L’**utilisation des bibliothèques disponibles** permet également d’optimiser le code. Les bibliothèques sont souvent optimisées pour des performances maximales et peuvent offrir des solutions plus efficaces que celles que l’on pourrait écrire soi-même, comme dans le cas des algorithmes de tri ou des calculs mathématiques complexes.

3. **Optimisation en utilisant un langage de bas niveau :**
   Parfois, pour les **besoins les plus critiques**, il est nécessaire de recourir à des langages de programmation bas niveau comme le **C** ou même l’**assembleur**. Ces langages permettent un contrôle plus direct du matériel et de la gestion de la mémoire, ce qui peut améliorer la vitesse d'exécution et l’utilisation des ressources. Cependant, cette approche est souvent plus complexe et moins portable que d'autres, car elle dépend du matériel cible.

4) Optimisation des Algorithmes

L'optimisation des algorithmes est l'une des méthodes les plus efficaces pour améliorer les performances d'un programme. Elle consiste à sélectionner des algorithmes ayant une complexité asymptotique plus favorable, ce qui permet de réduire le temps d'exécution, en particulier pour les grands ensembles de données. Par exemple, remplacer un algorithme de tri inefficace (comme le tri à bulle) par un algorithme plus performant (comme le tri rapide ou le tri par fusion) permet de traiter plus rapidement les données.
L'optimisation algorithmique passe également par l'analyse de la **complexité temporelle et spatiale** des algorithmes, permettant de choisir ceux qui s'adaptent le mieux aux besoins spécifiques du projet.

5) Optimisation de la gestion de la mémoire

L'optimisation de la mémoire vise à réduire la consommation de ressources mémoire d'un programme tout en garantissant un fonctionnement optimal. Cela comprend la gestion de l'allocation et de la libération de mémoire, afin d'éviter les **fuites de mémoire** et d'améliorer l'utilisation de la **mémoire cache**. Les **structures de données** doivent également être choisies judicieusement pour garantir qu'elles consomment le moins de mémoire possible tout en offrant des performances adéquates pour les opérations courantes.
Cette optimisation est particulièrement cruciale dans des environnements limités en mémoire, comme les **systèmes embarqués** ou les **applications mobiles**, où chaque octet de mémoire compte.

- **Gestion efficace de la mémoire** :
    - Allocation et libération appropriées de la mémoire.
    - Éviter les fuites de mémoire.
- **Optimisation du cache** :
    - Utilisation du cache pour accélérer l'accès aux données fréquemment utilisées.
    - Stratégies de prélecture et de mise en cache.

6) Optimisation des Entrées/Sorties (E/S)

Les entrées et sorties (E/S) représentent une partie importante des coûts d'exécution dans de nombreuses applications, notamment celles qui traitent de grandes quantités de données (bases de données, traitements de fichiers, etc.). Optimiser les opérations d'E/S permet de réduire le temps d'attente lors de la lecture ou de l'écriture de données.
Parmi les techniques d'optimisation des E/S, on trouve la réduction des accès aux disques, l'utilisation de la **mise en cache** des données fréquemment sollicitées et l'organisation des opérations d'E/S pour qu'elles soient effectuées de manière plus séquentielle et donc plus rapide.
L’optimisation des **requêtes de bases de données** et la gestion efficace des **flux de données** sont également des aspects cruciaux de cette catégorie.

7) Optimisation du multithreading et de la parallélisation

L'optimisation pour les environnements multithreadés et parallélisés est essentielle pour tirer parti des **processeurs multicœurs** et maximiser l'utilisation du matériel. Cette optimisation permet d'exécuter plusieurs tâches simultanément, réduisant ainsi le temps

d'exécution global pour des programmes gourmands en ressources.
Cependant, paralléliser un programme comporte des défis, notamment en ce qui concerne la gestion des **conflits d'accès à la mémoire partagée** et l'**équilibrage des charges** entre les différents threads. Des outils et techniques comme les **verrous**, les **barrières**, et les **files d'attente synchronisées** sont utilisés pour résoudre ces problèmes.
L'optimisation du multithreading est particulièrement importante pour les applications en temps réel, les simulations, et les logiciels nécessitant un traitement parallèle massif.

8) Optimisation des performances pour des environnements spécifiques

L'optimisation peut aussi être spécifique à un environnement particulier, qu'il s'agisse de **systèmes embarqués**, de **serveurs**, ou d'**applications mobiles**.

- **Systèmes embarqués :** L'optimisation dans les systèmes embarqués vise à réduire la consommation de mémoire, de processeur et d'énergie. Les ressources sont souvent limitées, ce qui nécessite une gestion méticuleuse de chaque composant matériel.
- **Applications mobiles :** Dans le contexte mobile, l'optimisation vise principalement à minimiser la consommation d'énergie et à assurer une réactivité rapide, même dans des conditions de réseau variables ou de faible puissance de traitement.
- **Serveurs et cloud computing :** L'optimisation dans les environnements de serveur et cloud implique la gestion de la scalabilité et de la **répartition de charge** pour garantir des performances et une disponibilité maximales.

9) Défis de l'optimisation du code

L'optimisation du code n'est pas sans défis. Parmi les principales difficultés, on retrouve :

- **L'optimisation prématurée :** Elle peut entraîner des choix inefficaces, car l'on tente d'optimiser des parties du code qui ne sont pas les véritables goulots d'étranglement.
- **Compromis entre lisibilité et performance :** Un code ultra-optimisé peut devenir difficile à comprendre et à maintenir, ce qui entraîne des difficultés pour les autres développeurs ou même pour l'auteur du code lors de futures mises à jour.
- **Portabilité :** Les optimisations peuvent rendre le code dépendant de l'architecture ou du matériel spécifique, ce qui limite sa portabilité.
- **Coût du temps de développement :** Parfois, les gains d'efficacité sont si minimes qu'il peut être plus coûteux en temps de développement de les réaliser, par rapport aux bénéfices réels obtenus.

10) Outils et techniques pour mesurer les performances

Pour savoir si l'optimisation du code est réellement efficace, il est indispensable de mesurer les performances avant et après toute modification. Plusieurs outils peuvent aider à profiler et mesurer la performance du code :

- **Profilers** (comme **gprof**, **perf**) : Permettent d'analyser le temps d'exécution des différentes parties du programme et de détecter les goulots d'étranglement.
- **Analyseurs de mémoire** (comme **Valgrind**, **AddressSanitizer**) : Aident à identifier les fuites de mémoire et à optimiser l'utilisation des ressources mémoire.
- **Outils de gestion des dépendances** (par exemple, pour gérer les versions de bibliothèques en Python avec **pip** ou en JavaScript avec **npm**) : Permettent d'optimiser l'utilisation des bibliothèques et de s'assurer que seules les versions nécessaires sont utilisées.

11) Conclusion

L'optimisation de code est un aspect fondamental du développement logiciel qui permet d'améliorer l'efficacité d'un programme en termes de temps d'exécution, de consommation mémoire et d'autres ressources système. Bien que l'optimisation puisse apporter de nombreux avantages, elle doit être réalisée de manière mesurée et réfléchie, en tenant compte des compromis nécessaires entre performances et lisibilité du code.

Les bonnes pratiques, les outils de profilage et la sélection des bonnes stratégies d'optimisation permettent de garantir que les programmes sont à la fois performants et maintenables sur le long terme. Il est également crucial de tester systématiquement les optimisations pour vérifier leur impact réel sur les performances.