Asynchronisme

Comment s’émanciper des threads.

Le problème de l‘approche par les threads

- Les std::thread correspondent à une notion système
- Les std::thread ne retournent pas des valeurs
- Les std::thread sont toujours exécuté en parallèle

Penser programmation concurrentielle

- Combien de processeurs (disponibles) ?
- Combien de coeurs (disponibles) ?
- Dois je créer un thread? Ou au contraire excécuter séquentiellement ?
  - Pas de réponse simle
  - Dépend du hardware et du contexte d‘éxécution

Asynchronie vs. Concurrence

- Idée : transformer une opération « longue » en opération « asynchrone »
- L'opération peut-être déléguée à un thread.
- L'opération retourne un objet qui indique si le résultat est disponible ou pas.

Mise en pratique en C++

```
std::future<double> result0 =
        std::async(std::launch::async,
        [] { return exp(1.0); });
```

`std::future<double> result0 :` objet donnant les informations sur l'exécution du code.

`result0.get()`	Retourne le résultat.
`result0.valid()`	Indique l'état du calcul.
`result0.wait()`	Attend que le résultat soit disponible.
`result0.wait_for()`	
`result0.wait_until()`	

TD --- Partie 1

- Transformer des fonctions standards en fonctions asynchrones.

C++ essaye de proposer de nouveaux paradigmes (C++ 17/C++ 20)

- Proposer pour les algorithmes de la STL un ensemble d'algorithmes parallèle

```
std::sort(array.begin(), array.end());

std::sort(std::execution::parallel_policy(),
      array.begin(), array.end());
```

C++ essaye de proposer de nouveaux paradigmes (C++ 17/C++ 20)

- La parallélisation d'ensemble de tâches

```
std::for_each_n(
   array.begin(), 3,
   [](auto& n) { n *= 2; });
```

Produit :
1, 2, 3, 4, 5,
2, 4, 6, 4, 5,

Et les strutures de données?

- Problème: comment garantir qu‘un élément ne soit pas modifié et accédé en même temps.
- Solution: (C++11/C++20)
  - La classe „atomic“: un objet „atomic“ est un objet qui garantit que l‘opération d‘accés à l‘objet natif supporte les accès concurrents.

```
template< class T > struct atomic;
```

Problème des atomic

- Introduit souvent un verrou (cf. Cours de la semaine dernière), risque d‘attente et coût de la requête.
- Solution: implantation „lock free“ quand possible (C++20)

  atomic_signed_lock_free

  atomic_unsigned_lock_free

Et pour les structures de données plus complexes

- Pas de solutions dans la STL
  - Pour des raisons de performance.
- Possibilité d‘utiliser des bibliothèques comme la Parallel Patterns Library (PPL)
  - Voir GitHub: cpp-docs/docs/parallel/concrt/parallel-patterns-library-ppl.md
  - Offre des structures comme un :
    - concurrency::concurrent_vector
    - concurrency::concurrent_queue