

# Mémoire transactionnelle logicielle

Félix-Antoine Ouellet

Université de Sherbrooke

23 octobre 2014

- 1 Motivation
- 2 Mémoire transactionnelle
- 3 Support présent
- 4 Conclusion

# Plan

- 1 Motivation
  - Programmation parallèle traditionnelle
  - Problèmes
- 2 Mémoire transactionnelle
- 3 Support présent
- 4 Conclusion

# Programmation parallèle traditionnelle

## Exemple

```
class Account {  
public:  
    void deposit(int n) {  
        m_Mutex.lock();  
        m_Balance += n;  
        m_Mutex.unlock();  
    }  
  
    void withdraw(int n) {  
        m_Mutex.lock();  
        if (m_Balance >= n) {  
            m_Balance -= n;  
        }  
        m_Mutex.unlock();  
    }  
};
```

# Programmation parallèle traditionnelle

## Exemple

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    From.withdraw(Amount);  
    To.deposit(Amount);  
}
```

# Programmation parallèle traditionnelle

## Exemple

```
void transfer(Account & From, Account & To,  
             int Amount) {  
    From.lock();  
    To.lock();  
    From.withdraw(Amount);  
    To.deposit(Amount);  
    From.unlock();  
    To.unlock();  
}
```

# Problèmes

- Problèmes de synchronisation

# Problèmes

- Problèmes de synchronisation



# Problèmes

- Problèmes de synchronisation
- Difficile à composer

# Problèmes

- Problèmes de synchronisation
- Difficile à composer
- Penser parallèle

# Plan

- 1 Motivation
- 2 Mémoire transactionnelle
  - Concept
  - Possible implémentation
  - Avantages
  - Problèmes potentiels
- 3 Support présent
- 4 Conclusion

# Concept

- Suite d'instructions exécutées d'une manière semblable aux transactions dans une base de données

# Concept

- Suite d'instructions exécutées d'une manière semblable aux transactions dans une base de données
  - Atomique: Aucun état intermédiaire visible

# Concept

- Suite d'instructions exécutées d'une manière semblable aux transactions dans une base de données
  - Atomique: Aucun état intermédiaire visible
  - Isolée: Non affectée par les autres *threads*

# Concept

## Exemple

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    atomic {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}
```

# Possible implémentation

Associer un *log* à chaque variable impliquée dans une transaction (globalement);

**while** *Transaction non valide* **do**

| Exécuter instructions sur une copie courante des variables;

| Valider transaction (atomiquement);

| **if** *Transaction valide* **then**

| | Écrire transaction (atomiquement);

| **end**

**end**



# Avantages

- Composable

# Avantages

- Composable
- Élimine les possibilités de *deadlock*

# Avantages

- Composable
- Élimine les possibilités de *deadlock*
- Facilite le raisonnement

# Problèmes potentiels

- Problèmes d'implémentation et de performance

# Problèmes potentiels

- Problèmes d'implémentation et de performance
- Interactions avec code non-transactionnel

# Problèmes potentiels

- Problèmes d'implémentation et de performance
- Interactions avec code non-transactionnel
- Gestions des exceptions

# Plan

- 1 Motivation
- 2 Mémoire transactionnelle
- 3 Support présent**
  - Langages de programmation
  - Matériel
- 4 Conclusion

# C++

Extension disponible dans GCC depuis GCC 4.7

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    __transaction_atomic {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}
```

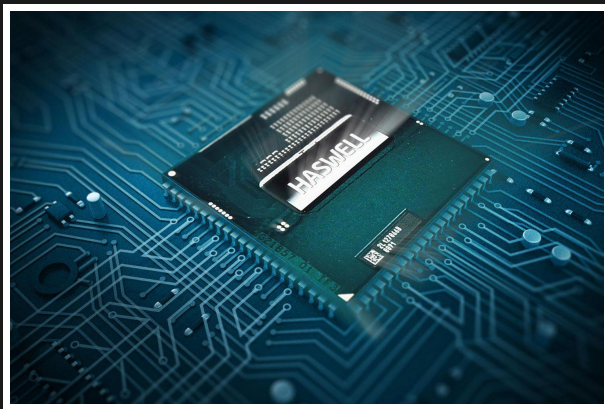


# C++

En voie d'être standardisé pour C++17

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    atomic_noexcept {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}
```

# Matériel



# Plan

- 1 Motivation
- 2 Mémoire transactionnelle
- 3 Support présent
- 4 Conclusion

# Conclusion

- Avenue intéressante pour le futur de la programmation parallèle

# Conclusion

- Avenue intéressante pour le futur de la programmation parallèle
- Loin d'être arriver à maturité