

Mémoire transactionnelle logicielle

Félix-Antoine Ouellet

Université de Sherbrooke

23 octobre 2014

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

Plan

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

Programmation parallèle traditionnelle

Exemple

```
class Account {  
private:  
    int m_Balance;  
  
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
- Difficile à composer
- Penser parallèle

Plan

- 1 Motivation
- 2 Mémoire transactionnelle
 - Concept
 - Possible implémentation
- 3 Évaluation
- 4 Support présent
- 5 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

Créer un *log* local au *thread* exécutant la section atomique;

repeat

 Exécuter instructions sur une copie courante des variables;

 Valider transaction (atomiquement);

if *Transaction valide* **then**

 Écrire transaction (atomiquement);

end

until *Transaction valide*;

Possible implémentation

Exemple

```
From::m_Balance = 10;  
To::m_Balance = 10;  
  
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type

Possible implémentation

Exemple

```
From::m_Balance = 10;  
To::m_Balance = 10;  
  
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L

Possible implémentation

Exemple

```
From::m_Balance = 10;  
To::m_Balance = 10;  
  
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L
&From::m_Balance	0	É

Possible implémentation

Exemple

```
From::m_Balance = 10;  
To::m_Balance = 10;  
  
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L
&From::m_Balance	0	É
&To::m_Balance	10	L

Possible implémentation

Exemple

```
From::m_Balance = 10;  
To::m_Balance = 10;  
  
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L
&From::m_Balance	0	É
&To::m_Balance	10	L
&To::m_Balance	20	É

Possible implémentation

Exemple

```
From::m_Balance = 10;
```

```
To::m_Balance = 10;
```

```
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L
&From::m_Balance	0	É
&To::m_Balance	10	L
&To::m_Balance	20	É

Possible implémentation

Exemple

```
From::m_Balance = 10;
```

```
To::m_Balance = 20;
```

```
atomic {  
    From.withdraw(10);  
    To.deposit(10);  
}
```

Adresse	Valeur	Type
&From::m_Balance	10	L
&From::m_Balance	0	É
&To::m_Balance	10	L
&To::m_Balance	20	É

Plan

- 1 Motivation
- 2 Mémoire transactionnelle
- 3 **Évaluation**
 - Avantages
 - Problèmes potentiels
- 4 Support présent
- 5 Conclusion

Avantages

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

Avantages

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    atomic {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}  
/*...*/  
atomic {  
    transfer(Account1, Account2, 500);  
    transfer(Account2, Account3, 300);  
}
```

- Composable

Avantages

```
void transfer(Account & From, Account & To,  
              int Amount) {  
    atomic {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}  
/*...*/  
atomic {  
    transfer(Account1, Account2, 500);  
    transfer(Account2, Account3, 300);  
}
```

- Composable
- Évite les *deadlocks*

Avantages

Facilite le raisonnement

Niveau professionnel

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

Niveau étudiant

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

Problèmes potentiels

Problèmes d'implémentation et de performance

- Solution avec verrous: Comment être performant?
- Solution avec verrous: Voir présentation précédente
- Solution hybride: Requiert beaucoup de finesse

Problèmes potentiels

Interactions avec code non-transactionnel

// Initialement: $x_b = \text{true}$; $x_i = 0$;

Thread 1

```
atomic {  
    x_b = false;  
}  
x_i = 100;
```

Thread 2

```
atomic {  
    if (x_b) {  
        ++x_i;  
    }  
}
```

Que vaut x à la fin du programme?

Problèmes potentiels

Gestions des exceptions

```
void fonction() {  
    /*...*/  
    atomic {  
        i++;  
        throw 42;  
    }  
    /*...*/  
}
```

Que faire?

Plan

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

C++

Extension disponible dans GCC depuis GCC 4.7

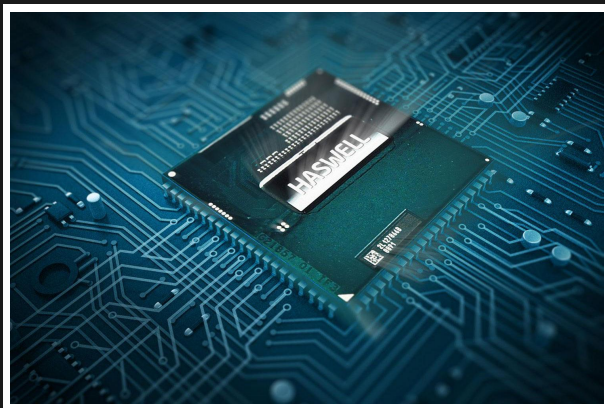
```
void transfer(Account & From, Account & To,  
              int Amount) {  
    __transaction_atomic {  
        From.withdraw(Amount);  
        To.deposit(Amount);  
    }  
}
```

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 Évaluation
- 4 Support présent
- 5 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