# 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();
  }
};
```

# Problèmes

- Problèmes de synchronisation

# Problèmes

- Problèmes de synchronisation
- Coût des verrous

# Problèmes

- Problèmes de synchronisation
- Coût des verrous
- Difficile à composer

# Problèmes

- Problèmes de synchronisation
- Coût des verrous
- Difficile à composer
- Penser parallèle

# Plan

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

# Concept

- Semblable aux transactions dans une base de données

# Avantages

# Problèmes potentiels

- Problèmes de performance

# Problèmes potentiels

- Problèmes de performance
- Interactions avec code non-transactionnel

# Problèmes potentiels

- Problèmes de performance
- Interactions avec code non-transactionnel
- Gestions des exceptions

# Possible implémentation

# Plan

1 Motivation

2 Mémoire transactionnelle

3 Support présent
- Langages de programmation
- Matériel

4 Conclusion

# Haskell

- Existe déjà dans Concurrent Haskell

```
credit :: Integer -> Account -> STM ()
credit amount account = do
    current <- readTVar account
    writeTVar account (current + amount)

debit :: Integer -> Account -> STM ()
debit amount account = do
    current <- readTVar account
    writeTVar account (current - amount)
```

# C++

- Extension disponible dans GCC depuis GCC 4.7

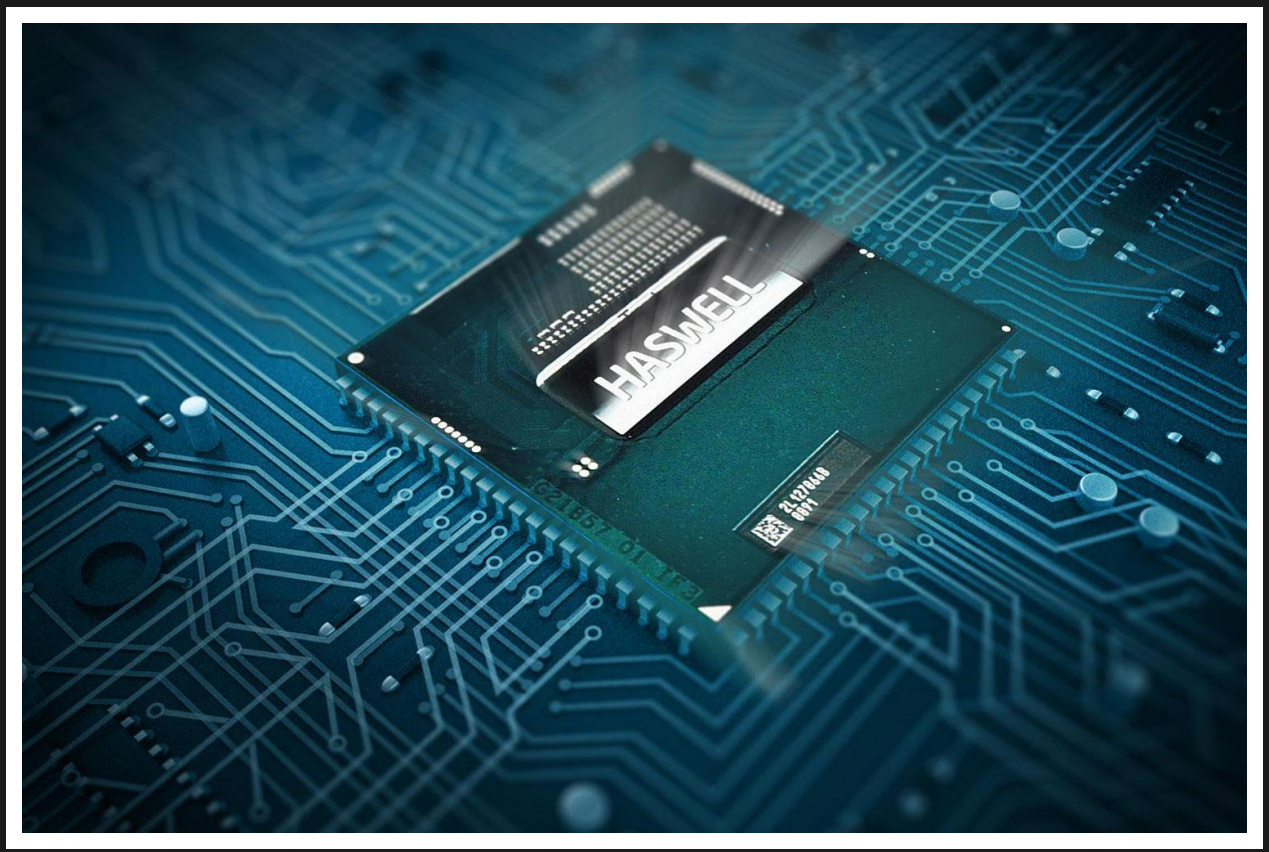
```
void func()
{
  for (int i = 0; i < 100; ++i) {
    __transaction_atomic {
      ++a;
      b += 2;
      c = a + b;
    }
  }
}
```

# C++

- En voie d'être standardisé pour C++17

```
void func()
{
  for (int i = 0; i < 100; ++i) {
    atomic_noexcept {
      ++a;
      b += 2;
      c = a + b;
    }
  }
}
```

# 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