Generic Programming Needs Transactional Memory

TRANSACT'13

Justin E. Gottschlich (Intel Labs)
Hans-J. Boehm (HP Labs)

The Problem

- **Popular belief:** *enforced locking ordering can avoid deadlock.*

- We show this is essentially impossible with C++ template programming.

- *Template programming is pervasive in C++. Thus, template programming needs TM.*

Don’t We Know This Already?

- Perhaps, but impact has been widely underestimated.
  - Templates are everywhere in C++.
- Move TM debate away from performance; focus on convincingly correct code.
- Relevant because of C++11 and SG5.

Motivating Example

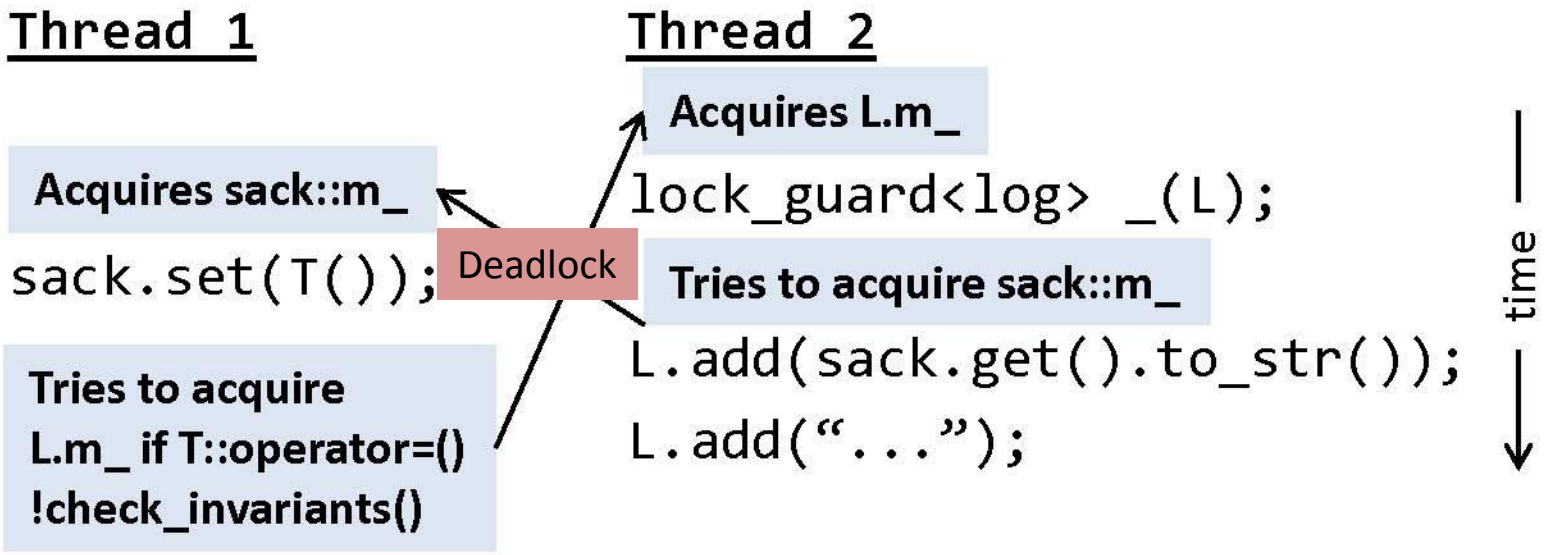
```
template <typename T>
class concurrent_sack
{
public:
   ...
   void set(T const &obj) {
      lock_guard<mutex> _(m_);
      item_ = obj;
   }
   T const & get() const {
      lock_guard<mutex> _(m_);
      return item_;
   }
private:
   T item_;
   mutex m_;
};
```

```
class log {
public:
   ...
   void add(string const &s) {
      lock_guard<recursive_mutex> _(m_);
      l_ += s;
   }
   void lock() { m_.lock(); }
   void unlock() { m_.unlock(); }
private:
   recursive_mutex m_;
   string l_;
} L;

class T {
public:
   ...
   T& operator=(T const &rhs) {
      if (!check_invariants(rhs))
      { L.add("T invariant error"); }
   }
   bool check_invariants(T const& rhs)
   { return /* type-specific check */; }
   string to_str() const { return "..."; }
};
```

Motivating Example

```
// Concurrent sack shared across multiple threads
concurrent_sack<T> sack;
```



This Time With Transactions

```
template <typename T>
class concurrent_sack
{
public:
   ...
   void set(T const &obj) {
      __transaction { item_ = obj; }
   }
   T const & get() const {
      __transaction { return item_; }
   }
private:
   T item_;
};
```

```
class log {
public:
   ...
   void add(string const &s) {
      __transaction { l_ += s; }
   }
private:
   string l_;
} L;

class T {
public:
   ...
   T& operator=(T const &rhs) {
      if (!check_invariants(rhs))
      { L.add("T invariant error"); }
   }
   bool check_invariants(T const& rhs)
   { return /* type-specific check */; }
   string to_str() const { return "..."; }
};
```

This Time With Transactions

```
// Concurrent sack shared across multiple threads
concurrent_sack<T> sack;
```

Thread 1

Begins sack transaction

```
sack.set(T());
```

Begins L transaction if T::operator=() !check_invariants()

Thread 2

Begins local transaction

```
__transaction {
```

Begins sack transaction, then L transaction

```
  L.add(sack.get().to_str());
  L.add("...");
}
```

Conclusion

- Given C++11, generic programming needs TM more than ever.

- To avoid deadlocks in ***all*** generic code, even those with irrevocable operations, we need (something like) relaxed transactions.

Questions?

Generic Programming Needs Transactional Memory

TRANSACT'13

Justin E. Gottschlich (Intel Labs)
Hans-J. Boehm (HP Labs)