Integrating Transactional Memory into C++

Victor Luchangco

Scalable Synchronization Research Group
Sun Microsystems Laboratories
16 Aug 2007

(joint work with Lawrence Crowl, Yossi Lev, Mark Moir and Dan Nussbaum)

Pragmatic approach

- useful to “working programmer”
- sooner rather than later

Pragmatic approach

- useful to “working programmer”
- sooner rather than later

We want a design that can we can implement and make available to real C++ programmers
and that they will want to use.

Desiderata

- composability
- minimality
- predictability
- orthogonality
- implementability
- incremental development
- incremental adoption
- scalability
- efficiency

Desiderata

- composability
- minimality
- predictability
- orthogonality
- implementability
- incremental development
- incremental adoption
- scalability
- efficiency

Desiderata

- composability
- minimality
- predictability
- orthogonality
- implementability
- incremental development
- incremental adoption
- scalability
- efficiency

compiler support

vs.

library support

“object-based”

vs.

“word-based”

```
transaction {
  myAccount.deposit(x);
  yourAccount.withdraw(x);
}
```

- begin on block entry
- commit/abort on block exit
- entry and exit based on **dynamic** extent
  - like try statement

```
transaction {
  myAccount.deposit(x);
  yourAccount.withdraw(x);
}
```

- commit on “normal” termination
  - including break, continue, return
- what about “abnormal” termination?
  - conflict
  - asynchronous events
  - exceptions

```
transaction {
  myAccount.deposit(x);
  yourAccount.withdraw(x);
}
```

What if `withdraw` throws an exception?

- commit
- abort “on exit”
- abort “on throw”

```
transaction {
  myAccount.deposit(x);
  yourAccount.withdraw(x);
} exception { printf("error"); }
```

What if `withdraw` throws an exception?

- commit
- abort “on exit”
- abort “on throw”

strong atomicity

vs.

weak atomicity

privatization

Other issues

- nesting: flat, closed, open
- input/output
- system calls
- programmer-directed contention control
- explicit abort
- transactional “control codes”
- customizing transactions

Conclusion

We want a design that can we can implement and make available to real C++ programmers
and that they will want to use.

- incremental
- sensitive to C++ programming style
- available sooner than later
- experience to guide research