Exceptions and Transactions in C++

Ali-Reza Adl-Tabatabai[1], Victor Luchangco[2], Virendra J. Marathe[2], Mark Moir[2], Ravi Narayanaswamy[1], Yang Ni[1], Dan Nussbaum[2], Xinmin Tian[1], Adam Welc[1], Peng Wu[3]

[1]Intel [2]Sun [3]IBM

Background

TM requires language support

Multiple projects extend C/C++ with TM constructs

Adoption requires common TM language extensions

Intel, Sun, IBM discussions on C++ extensions

- Agree where possible on a specification
- Understand differences

How to define semantics of TM extensions in C++?

Today's talk: TM & C++ exceptions interaction

TM extensions

```
__tm_atomic {
   <stmts>
   if (cond)
      __tm_abort;
}
```

Atomic block executes as a transaction

Abort rolls back atomic block

What happens if an exception is thrown out of the atomic block?

The problem

```
__tm_atomic {

  x++;
  if (cond)
    throw MyException();
}
```

When MyException escapes the atomic statement

- Should the effects of x++ be committed?
- Or should they be rolled back?

Active debate in the community

To commit?

"Commit-on-escape"

Exception is just another exit from atomic block

Similar to the behavior expected for locks or single-threaded programs

- Exception safety up to the programmer
- Fits naturally with lock-based TM semantics

Easier to implement

Exceptions & concurrency control should be orthogonal

But …

Against commit

```
__tm_atomic {
  withdraw(euro_amount,euro_account);
  dollar_amount = convert(euro_amount);
  deposit(dollar_amount,dollar_account);
}
```

Throws Exception

Invariant: Total balance remains constant

Commit exposes broken invariants to other threads

- Programmer can easily overlook hidden exceptions paths

Concurrency exacerbates exception safety issues

Or not to commit?

"Abort-on-escape"

On exception: Rollback atomic block

Leaves program state as it was before atomic block

Automatic strong exception safety guarantee

Programmers can reason about atomic statements as "all-or-nothing"

- Failure atomicity: A key value proposition of transactions

But …

Against abort

```
__tm_atomic {
   ...
   if (amount < 0)
     throw ConvertException(amount);
}
```

Exception reports incorrectly converted amount

But state of atomic statement is rolled back including exception object

- What exception object should propagate out of atomic?

Overkill if code already provides exception safety

Both sides are right

Some programs behave surprisingly under commit-on-escape

Others under abort-on-escape

Observations:

- Exceptions that can escape an atomic statement without being clear are potentially dangerous
- No single behavior appropriate for all cases
  - Only the programmer can determine what's appropriate

Our approach

Support both semantics & let programmer decide

New syntax for

- Exception specifications on atomic blocks
- Throwing exceptions that abort

Significant progress towards an agreement

- An open issue still remains

Exception specification

Specify which exceptions may escape an atomic

```
__tm_atomic throw(E1,E2){…}// E1 or E2
__tm_atomic throw()      {…}// no exceptions
__tm_atomic throw(…)     {…}// all exceptions
```

Terminate if exception does not match specification

No specification?

```
__tm_atomic              {…}// default ???
```

Default behavior still under debate

- As if **no** exceptions allowed to escape
- As if **all** exceptions allowed to escape

Commit-on-escape

Standard syntax for exception throw

```
__tm_atomic throw(MyException) {
  ...
  throw MyException();
}
```

Easy to specify that any exception commits

```
__tm_atomic throw(…) {
  exception_throwing_fun();
}
```

Abort-on-escape

New syntax for exception throw

```
__tm_atomic throw(MyException) {
  ...
  __tm_abort throw MyException();
}
```

Exception object is not rolled back

Programmer must ensure exception object makes sense after rollback

- E.g., avoid dangling pointers

Aborting on any exception

```
__tm_atomic throw(...) {
  try {
    <stmts>
  } catch (...) {
    __tm_abort throw;
  }
}
```

Any exception aborts atomic block & propagates exception

Conclusion

We need common C++ language extensions for TM

Flexible integration of exceptions & atomic blocks in C++

Worked out jointly by Sun, IBM, and Intel

- Debate continues over default for no exception specification
- More work remains to complete full specification

Try it out using the Intel C++ STM compiler

- Download from whatif.intel.com
- 32-bit & 64-bit Windows & Linux
- Compiler-runtime ABI specification also available