The Xfork in the Road to Coordinated Sibling Transactions

Hany E. Ramadan and Emmett Witchel
University of Texas at Austin

TRANSACT  2/15/2009
TM solving the wrong problem?

• Finding parallelism is the key challenge of the multi-core era
• Transactional memory makes creating critical regions easier
• Transactional memory can also make...
  • finding parallelism easier
  • … but not with current model/API!

“Numerous types of processing are not well-served by [the flat transaction model’s simplicity]“  - Gray & Reuter
How will TM find threads?

- **Approach**
  - Intratransaction parallelism
- **Benefits**
  - Improve performance of individual transactions & utilize cores
- **Methodology**
  - Revisit transaction model, API
- **Advice**
  - “Intratransaction parallelism requires genuine support for nested transactions.” – Gray & Reuter
Nested Transactions

- Two types
  - Closed Nesting (commits into *parent*)
  - Open Nesting (commits into *world*)
- Serial closed nesting
  - Simpler semantics (than open)
  - Small or no performance gain
- Parallel closed nesting
  - Defined semantics for siblings
  - How much performance gain?

```plaintext
Begin Tx (parent)
// work a
Begin Tx (nested)
// work b
End Tx (nested)
End Tx (parent)
```
Forms of intra-transaction parallelism

- Independent
- Dependent
- Speculative

```
atomic Foo {
    // Code
}
```

```
atomic Foo {
    // Code
}
```
Programmer’s mental picture

```
atomic Foo
{
    FooTop
        AND
            ○
            △
    FooBottom
        XOR
            [red]
            [green]
            [blue]
}
```

# of leafs = maximum amount of intratransaction parallelism
Coordination forms

- **OR** (*independent*)
  - Independent siblings
  - No short circuit

- **AND** (*dependent*)
  - All siblings must commit or none do

- **XOR** (*speculative*)
  - Exactly one is allowed to commit
  - Once one sibling commits, all others made to abort

- More forms conceivable
Coordinated Sibling Model

- Parallel closed nested transactions
  - Safety and semantics of closed nesting
- Coordination forms express multiple paradigms
  - fork/join
  - distributed transactions
  - speculative execution
- Sibling code returns Success/Fail
  - Commit/retry / abort decision made based on form, and status of siblings
xfork: API

- xfork API shields programmer from the dirty work
  - Threading, coordination, aborting, speculation, etc.

```c
bool xfork (xforkForm, numForks,
            xforkProc, optionalData);
```

```c
denum xforkForm { AND, OR, XOR };
delegate xforkResult xforkProc (forkNum, optionalData);
denum xforkResult { Success, Failure };
```
xfork: Example

using (TransactionScope ts = new TransactionScope())
{
    sstm.txStore.write (myAddr, “hello”);
    sstm.xfork( xforkForm.XOR, // form
                2,  // numForks
                new Delegate (parallelSearchProc), // xfProc
                myList);  // data
    ts.Complete();
}

xforkResult parallelSearchProc(int forkNum, object myList)
{
    if (forkNum == 0) { FwdTraverse(myList) } else { ReverseTraverse(myList) }
    return xforkResult.Success;
}
Sibling STM (SSTM): Architecture

- System.Transactions provides database-derived framework
- DB terminology:
  - TM: Transaction Manager (executes 2 phase commit protocol)
  - RM: Resource Manager (holds transactional data / votes in 2pc)
  - ‘Enlisting’: when RM first meets TM.
- Sibling Executive (SibEx) (RM)
  - Implements xfork semantics
- TxStore (RM)
  - Conventional STM
SSTM: SibEx

1. Schedule work for each fork on some thread
   Use existing thread-pool
2. Create sibling nested transactions
3. Invoking user-procedure for each fork
4. Enforce semantics of each sibling form
   May involve finding transaction outcome (OR-form), or enlisting and voting on outcome (AND/XOR-form)
5. Stalling the parent appropriately
SSTM: TxStore

- Extends TL2 to support nesting
  - Key change: per-object version-number (for write set)
  - Additional synchronization to supported parallel access
- Extend System.Transactions to support nesting
  - Thread-local transactional context extended with “parents” list
- Low-level API, based on System.Object/ICloneable
  - object TxStore.Read(int address);
  - void TxStore.Write(int address, object obj);
SSTM: Prototype

• System configuration
  - Intel Core2 Quad CPU running at 2.66 Ghz
  - Microsoft .NET v2 Framework, MS Windows Vista
  - SSTM: 2,345 lines of C# code

• Benchmarks (~600 loc)
  - SearchList:
    - Search linked list in parallel
  - Transfer
    - Debit/credit of bank accounts using two lists

• Compare SSTM to SSTM without xfork
  - Measure speedup, from 1 to 4 forks
SearchList benchmark

- Four-fork version also searches from middle of list
Transfer benchmark

- Higher overhead due to AND-form coordination
Related Work

- Nested Transactions
  - Formalized by [Moss81], [Beeri83]
  - Used in Argus[Liskov88], Camelot [Eppinger91]
- TM Nesting
  - HTM [Moravan06, McDonald06]
  - STM [Ni07, Harris07, Moss05]
- Parallelism
  - Fortress[Allen07], XCilk[Agrawal08]
  - Parallel for-loops, etc. in OpenTM[Baek07]
- Composition constructs (e.g. orElse) [Harris05]
Conclusion

- Intratransaction parallelism
  - Makes TM even more relevant to the challenges of multi-core
- Coordinated siblings
  - Transaction model that suits this type of processing
- xfork
  - Makes coordinated siblings easy to use

Will intratransaction parallelism become a commodity?