MANCHESTER
1824

The University of Manchester

Manchester University Transactions for Scala

Daniel Goodman
Daniel.Goodman@man.ac.uk

Euro TM
Paris, May 2011

Teraflux & Scala

- Teraflux: Exploiting Data Parallelism for 1000 Core Processors
  - Programming Models, Transactional Memory, Memory Models, Chip Design, Fault Tolerance, ...
  - European Commission’s FP7 funded
  - Using Scala as a basis for high productivity language

- Scala
  - Combination of Object Oriented and Functional Programming Models
  - Compiles to JVM Byte-Code

Why MUTS

Existing Scala STMs do not allow transactions to be added to code without restructuring the code.

We want:

- No difference between transaction syntax and other language constructs
- No restrictions on transaction granularity
- Works with legacy code
- Maintainability

MUTS Syntax

- ```
  atomic {
      body
  }
  ```
- ```
  atomic(test) {
      body
  }
  ```

Transaction that **does not** retry on failure

- ```
  atomic {
      body
  } retry;
  ```
- ```
  atomic(test) {
      body
  } retry;
  ```

Transaction that **does** retry on failure

- ```
  atomic {
      body
  } orElse {
      elseBody
  }
  ```
- ```
  atomic(test) {
      body
  } orElse {
      elseBody
  }
  ```

Transaction that runs alternative code on failure

Implementing MUTS

43 Possible locations within the compiler to add or augment passes to implement transactions

- Two phases
  - Modifications to the parser
  - Java Agent to instrument the Byte-Code
- Both of these work with well defined interfaces
  - Scala
  - JVM Byte-Code

Parser Modifications

- Add new keywords
- When an atomic section is detected:
  - Add the control logic using **existing** tree constructs
  - **Encase the transactional code with an try/catch**
    - Handle exceptions thrown by the body
    - **Mark the code that is transactional**
    - Allow transactions to abort
  - Create a context to store the transaction meta-data
  - Copy method variables so that active updates can be used on them by the body

Parser Modifications

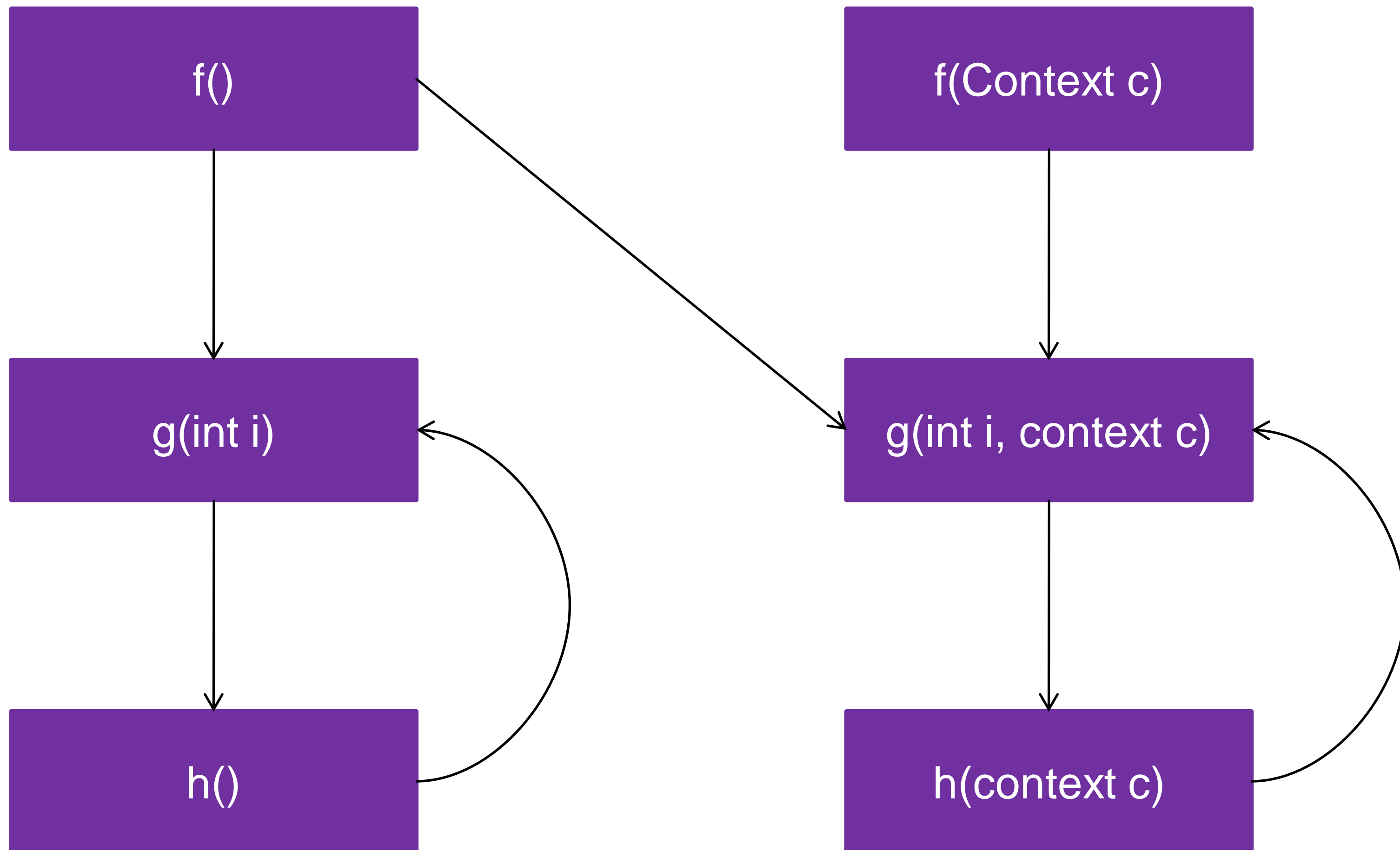
```
var committed$TM = false
Var context$TM = null
do{
  context$TM = TM.beginTransaction()
  atomic {
  try {
    Body
  } retry;
    committed$TM = TM.commit(context$TM)
  } catch {
    case e:TransactionArea => ()
    case e:TransactionException => context$TM.rollback()
    case e:Exception => {
      committed$TM = TM.commit(context$TM)
      if (committed$TM) throw e
    }
  }
} while(!committed$TM)
```

Byte-Code Rewrite

Modifying the Deuce Java Agent

- Add Duplicate Methods
- Detect atomic sections of methods
  - Detect the location of the context
  - Instrument field accesses
  - Augment method calls
  - Remove the marker exception

Byte-Code Rewrite



MANCHESTER 1824

The University of Manchester

Byte-Code Rewrite

```
var committed$TM = false
Var context$TM = null
do{
  context$TM = TM.beginTransaction()
  try {
    Body
    committed$TM = TM.commit(context$TM)
  } catch {
    case e:TransactionArea => ()
    case e:TransactionException => context$TM.rollback()
    case e:Exception => {
      committed$TM = TM.commit(context$TM)
      if (committed$TM) throw e
    }
  }
} while(!committed$TM)
```

MANCHESTER 1824

The University of Manchester

Protecting Against Code Reordering

```
try {
  System.out.println("This is the start of the test");

  try{ foo(); }
  catch(IOException e) { System.out.println("IO exception"); }
  catch(NullPointerException e) { System.out.println("Null Pointer
                                                        Exception");
   }

  System.out.println("This is the end of the test");
}
catch(Exception e) {  System.out.println("The final Exception"); }
```

Exception table:

```
   from    to  target type
     8     11     14   Class java/io/IOException
     8     11     26   Class java/lang/NullPointerException
     0     43     46   Class java/lang/Exception
```

Protecting Against Code Reordering

```
try {
  System.out.println("This is the start of the test");

  try{ foo(); }
  catch(IOException e) { System.out.println("IO exception"); }
  catch(NullPointerException e) { System.out.println("Null Pointer
                                                     Exception");
   }

  System.out.println("This is the end of the test");
}
catch(Exception e) {  System.out.println("The final Exception"); }
```

Exception table:

```
   from    to  target type
     8     11    14   Class java/io/IOException
     8     11    26   Class java/lang/NullPointerException
     0     43    46   Class java/lang/Exception
```

Protecting Against Code Reordering

```
try {
   System.out.println("This is the start of the test");

   try{ foo(); }
   catch(IOException e) { System.out.println("IO exception"); }
   catch(NullPointerException e) { System.out.println("Null Pointer
                                                          Exception");
    }

   System.out.println("This is the end of the test");
}
catch(Exception e) {  System.out.println("The final Exception"); }
```

Exception table:

```
   from    to   target type
     8     11     14    Class java/io/IOException
     8     11     26    Class java/lang/NullPointerException
     0     43     46    Class java/lang/Exception
```

What Have We Gained?

User:

- Native Constructs
- No change of syntax
- No restrictions on the granularity of transactions
- No restrictions on the use of legacy code
- Interoperable with Java

Implementer:

- Working against well defined interfaces
- Native constructs with minimal changes to the compiler

Conclusions

- MUTS is a much more intuitive and flexible STM for users
- Interoperable with Java
- Implemented with minimal changes to the parser, and no other changes to the compiler
- Exception handler overcomes code reordering
- This 2-phase approach can be applied to implementing other native constructs
- Part of a suite of Scala STMs at Manchester

Produced as part of the Teraflux Project http://www.teraflux.eu

Daniel.Goodman@Manchester.ac.uk

Other TM Research at Manchester

- Applications: Transactional Parallel Routing
  - Lee-TM (aka labyrinth in STAMP) [1][2]
- TM Contention Managers [3]
- Control of Optimistic Execution [4]
- Scheduling of Transactions [5, 6]
- Hardware TM [6, 7, 8]
- Distributed TM [9, 10]

http://apt.cs.manchester.ac.uk/projects/TM

Produced as part of the Teraflux Project http://www.teraflux.eu

Daniel.Goodman@Manchester.ac.uk

MANCHESTER 1824

The University of Manchester

Selected References

[1] Ian Watson, Chris Kirkham and Mikel Luján. A Study of a Transactional Parallel Routing Algorithm. PACT 2007.

[2] Mohammad Ansari, Christos Kotselidis, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Lee-TM: A Non-trivial Benchmark for Transactional Memory. ICA3PP 2008.

[3] Mohammad Ansari, Christos Kotselidis, Mikel Luján, Chris Kirkham and Ian Watson. On the Performance of Contention Managers for Complex Transactional Memory Benchmarks. *ISPDC* 2009.

[4] Mohammad Ansari, Mikel Luján, Christos Kotselidis, Kim Jarvis, Chris Kirkham and Ian Watson. Robust Adaptation to Available Parallelism in Transactional Memory Applications. *In* Transactions on High Performance and Embedded Architectures and Compilers, 2008.

[5] Mohammad Ansari, Mikel Luján, Christos Kotselidis, Kim Jarvis, Chris Kirkham and Ian Watson. Transaction Reordering to Reduce Aborts in Software Transactional Memory. In Transactions on High Performance and Embedded Architectures and Compilers, 2009.

[6] Mohammad Ansari, Behram Khan, Mikel Luján, Christos Kotselidis, Chris Kirkham and Ian Watson. Improving Performance by Reducing Aborts in Hardware Transactional Memory. In HIPEAC, 2010.

[7] Behram Khan, Matthew Horsnell, Ian Rogers, Mikel Luján, Andrew Dinn, and Ian Watson. An Object-Aware Hardware Transactional Memory System . In HPCC, 2008.

[8] Behram Khan, Matthew Horsnell, Mikel Luján, Andrew Dinn and Ian Watson. Exploiting object structure in hardware transactional memory. In EUROPAR, 2010.

[9] Christos Kotselidis, Mohammad Ansari, Kim Jarvis, Mikel Luján, Chris Kirkham and Ian Watson. DiSTM: A Software Transactional Memory Framework for Clusters. In ICPP, 2008

[10] Christos Kotselidis, Mikel Luján, Mohammad Ansari, Konstantinos Malakasis, Behram Khan, Chris Kirkham and Ian Watson. Clustering JVMs with Software Transactional Memory Support. In IPDPS, 2010.

http://apt.cs.manchester.ac.uk/projects/TM