Leveraging Transactional Memory to Extract Parallelism

M.A. Gonzalez-Mesa, E. Gutierrez, Oscar Plata
Dept. Computer Architecture
University of Malaga
Euro-TM
Motivation

**Parallel Loop**

```c
#pragma omp parallel for
for (i=1; i<N; i++)
    b[i]=(a[i]+a[i-1])/2.0;

#pragma omp parallel transfer
for (i=0; i<N; i++)
    bin[A[i]] = bin[A[i]]+1;

#pragma omp parallel transfer ordered
for (i=1; i<N; i++)
    bin[A[i]] = bin[A[i]]+bin[A[i-1]];
```

**Parallel Section**

```c
#pragma omp parallel sections
{
    #pragma omp section
    xaxis();
    #pragma omp section
    yaxis();
    #pragma omp section
    zaxis();
}

#pragma omp parallel sections ordered
{
    #pragma omp transsection
    xaxis();
    #pragma omp transsection
    yaxis();
    #pragma omp transsection
    zaxis();
}
```

Source: OpenMP API Ver. 3.1; OpenTM (PACT’07)
Contributions

• Leveraging TM to extract parallelism from non-analyzable codes
  - Transactions are statically numbered according to sequential ordering
  - A fully distributed commit manager decouples transaction completion from commit, and schedules commits preserving data dependences
  - A thread is allowed to execute several transactions in sequence before committing the first one
  - By scheduling commits, many aborts due to data conflicts are avoided
  - A restricted data forwarding technique, called FAC (Forwarding After Completion), reduces stall times when preserving certain flow dependences

• Initial evaluation
  - A first implementation of the above techniques is based on TinySTM
  - Some preliminary results on a simple benchmark (sparse matrix transpose)
Outline

• Motivation and Contributions
• Our approach for extracting parallelism
• Evaluation
Basics of our Approach: Parallel Loops

• Programming side
  ▪ A team of $N$ threads is formed to execute in parallel transactions
  ▪ Each transaction is block of iterations of size $S$ that are statically assigned to the threads in a round-robin fashion in the order of the thread number (equivalent to the OpenMP clause `schedule(static,S)`)
  ▪ Each transaction is assigned an order number ($TON$) according to the loop iteration ordering

• TM side
  ▪ Conflict detection is eager
    » Transaction ordering is used to filter out many conflicts avoiding aborts
    » Completed transactions may forward written data to subsequent transactions
  ▪ Conflict management is fully distributed:
    » Transaction completion (execution finished) is decoupled from transaction commit
    » When a transaction completes, the thread tries to commit all completed but still uncommitted transactions assigned to the same thread
    » A transaction commits when all previous transactions are completed or committed (this condition allows out of order commits when safe)
  ▪ Version management is lazy
    (although it could be eager in TM systems with strong isolation, like LogTM)
Conflict Management

• Fully distributed conflict manager
  ▪ Each thread executes the following loop when completing a transaction
  ▪ As committing is decoupled from completion, the thread starts a new assigned transaction instead of stalling waiting for commit
  (Stanford TCC has a similar mechanism but transactions always stall before committing)
Conflict Management: Example

- Fully distributed conflict manager
  - Transactions can **commit out of order** if it is safe
    (previous transactions are completed or committed and no conflicts detected)

```
Thread 1
Xact TON=1
Commit 1
Xact TON=4
Commit 4
Xact TON=7
Commit 7
Xact TON=10

Thread 2
Xact TON=2
Chkp 2
Xact TON=5
Chkp 5
Xact TON=8
Chkp 8
Xact TON=11

Thread 3
Xact TON=3
Commit 3
Xact TON=6
Commit 6
Xact TON=9
Commit 9
Xact TON=12
Commit 12
```

Time

Dept. Computer Architecture, University of Malaga
Data Dependences

- Conflict detection is eager
  - Possible data dependence violations are detected immediately

- Preservation of data dependences
  - Conflict manager preserves data dependences by scheduling transaction commits
    - DATM (Micro’08) proposes a similar approach but without explicit transaction ordering (TON)
  - A restricted version of data forwarding, called FAC (Forwarding After Completion), is used to improve parallelism
    - Only completed transactions can forward written values to subsequent (in TON order) transactions
    - FAC reduces the amount of forwarded data if a transaction writes several times in the same memory location
    - FAC avoids circular dependences between two transactions
    - DATM, in contrast, forwards data immediately to any requester
Data Dependences: Case Output-

- **Output dependences**
  - Transactions with a Write-Write conflict **must commit in order**
  - Both transactions run in parallel with no abort

```
Xact TON=1
Commit 1
Xact TON=4
Commit 4
Xact TON=7
Commit 7
Thread 1

Thread 2
Xact TON=2
Chkp 2
Write A
Xact TON=5
Chkp 5
Commit 2

Thread 3
Xact TON=3
Write A
Xact TON=6
Chkp 6
Commit 3
```
Data Dependences: Case Anti-

• Antidependences
  - Transactions with a Read-Write conflict can run safely
    (the baseline commit scheduling always preserve these dependences)
Data Dependences: Case Flow-

- **Flow dependences**
  - Transactions with a Write-Read conflict **must stall** or **abort** the reading transaction (depending on which operation executed first)
  - When stalling, the use of FAC may reduce the stall time

![Diagram showing the flow of transactions and checkpoints with stalls and forwards](image-url)
Data Dependences: Case Flow-

- Flow dependences
  - Transactions with a Write-Read conflict **must stall** or **abort** the reading transaction (depending on which operation executed first)
  - When stalling, the use of FAC may reduce the stall time
Outline

• Motivation and Contributions
• Our approach for extracting parallelism
• Evaluation
Evaluation

• Implementation
  ▪ The described techniques were implemented on TinySTM
  ▪ Tested benchmark code: Sparse matrix transpose
  ▪ All experiments were conducted on a shared-memory multiprocessor with four 8-core Intel i7 processors (32 cores in total) running Linux
Benchmark Code

- **Transpose of a sparse matrix**
  - Sparse matrix compressed in CRS format
  - Due to indirections through `rowT()`, loop-carried dependencies may exist depending on the distribution of the non-zero elements in the matrix

### Initialization

```
do i = 2, Ncols+1
    rowT(i) = 0
  enddo

do i = 1, row(Nrows+1)-1
    rowT(col(i)+2) = rowT(col(i)+2)+1
  enddo

rowT(1) = 1
rowT(2) = 1

do i = 3, Ncols+1
    rowT(i) = rowT(i) + rowT(i-1)
  enddo
```

### Transpose

```
do i = 1, Nrows
  do j = row(i), row(i+1)-1
    dataT(rowT(col(j)+1)) = data(j)
    colT(rowT(col(j)+1)) = i
    rowT(col(j)+1) = rowT(col(j)+1)+1
  enddo
```

```
  enddo
```

```
enddo
```
**Benchmark Code**

- **Transpose of a sparse matrix**
  - Sparse matrix compressed in CRS format
  - Due to indirections through rowT(), loop-carried dependencies may exist depending on the distribution of the non-zero elements in the matrix

---

### Initialization

```plaintext
do i = 2, Ncols+1
  rowT(i) = 0
endo
do i = 1, row(Nrows+1)-1
  rowT(col(i)+2) = rowT(col(i)+2)+1
endo
rowT(1) = 1
rowT(2) = 1
do i = 3, Ncols+1
  rowT(i) = rowT(i) + rowT(i-1)
endo
```

---

### Transpose

```plaintext
#pragma omp target schedule(static,S) ordered
do i = 1, Nrows
  do j = row(i), row(i+1)-1
    dataT(rowT(col(j)+1)) = data(j)
    colT(rowT(col(j)+1)) = i
    rowT(col(j)+1) = rowT(col(j)+1)+1
  enddo
endo
do i = 2, Ncols+1
  rowT(i) = 0
endo
do i = 1, row(Nrows+1)-1
  rowT(col(i)+2) = rowT(col(i)+2)+1
endo
rowT(1) = 1
rowT(2) = 1
do i = 3, Ncols+1
  rowT(i) = rowT(i) + rowT(i-1)
endo
```
## Benchmark Sparse Matrices

- **Sparse matrices**
  - Source: Matrix Market

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size</th>
<th>Non-zeros (Sparsity)</th>
<th>Symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>494 BUS</td>
<td>494 x 494</td>
<td>1666 (0.68%)</td>
<td>Yes</td>
</tr>
<tr>
<td>BCSPWR04</td>
<td>274 x 274</td>
<td>1612 (2.15%)</td>
<td>Yes</td>
</tr>
<tr>
<td>BCSPWR10</td>
<td>5300 x 5300</td>
<td>21842 (0.08%)</td>
<td>Yes</td>
</tr>
<tr>
<td>BEAUSE</td>
<td>497 x 506</td>
<td>50409 (20.04%)</td>
<td>No</td>
</tr>
<tr>
<td>WM3</td>
<td>207 x 260</td>
<td>2948 (5.48%)</td>
<td>No</td>
</tr>
</tbody>
</table>
Experimental Results

- Aborts avoided by the conflict manager
  - $S$ is the size of the block of iterations executed as a transaction

![](chart.png)
**Experimental Results**

- Aborts avoided by the conflict manager
  - $S$ is the size of the block of iterations executed as a transaction

![Bar chart showing experimental results](chart.png)
Experimental Results

- Aborts avoided by the conflict manager
  - $S$ is the size of the block of iterations executed as a transaction

![Bar Chart]

BCSPWR10

<table>
<thead>
<tr>
<th>$S$</th>
<th>Threads = 2</th>
<th>Threads = 4</th>
<th>Threads = 8</th>
<th>Threads = 16</th>
<th>Threads = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>S = 2</td>
<td>S = 4</td>
<td>S = 8</td>
<td>S = 16</td>
<td>S = 32</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Experimental Results

- Aborts avoided by the conflict manager
  - S is the size of the block of iterations executed as a transaction
Experimental Results

- Aborts avoided by the conflict manager
  - S is the size of the block of iterations executed as a transaction

![Graph showing aborts avoided by the conflict manager for different sizes of block iterations and thread counts.]

<table>
<thead>
<tr>
<th>Threads</th>
<th>S = 2</th>
<th>S = 4</th>
<th>S = 8</th>
<th>S = 16</th>
<th>S = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>No</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
</tbody>
</table>

Oscar Plata
Dept. Computer Architecture, University of Malaga
Experimental Results

- Extracted parallelism
  - TCR: Transaction Commit Rate

Baseline TinySTM

Our approach on TinySTM
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```

```
Experimental Results

- **Extracted parallelism**
  - TCR: Transaction Commit Rate

```

```
Baseline TinySTM

Our approach on TinySTM
```
Experimental Results

- Extracted parallelism
  - TCR: Transaction Commit Rate

Baseline TinySTM

Our approach on TinySTM
Experimental Results

• Extracted parallelism
  ▪ TCR: Transaction Commit Rate

Baseline TinySTM

Our approach on TinySTM
Experimental Results

- Extracted parallelism
  - TCR: Transaction Commit Rate

Baseline TinySTM

Our approach on TinySTM
Conclusions

• Leveraging TM to extract parallelism
  ▪ Design of a fully distributed conflict manager that decouples transaction completion from commit
  ▪ Conflict manager schedules commits in order to preserve data dependences, avoiding a great amount of aborts due to conflicts
  ▪ Data forwarding after completion technique to reduce stall times due to flow dependences

• Future work
  ▪ Test different options in the design space, and exploit new properties:
    » Eager version management instead of lazy, speeding up commits
    » Stall versus data forwarding
    » Forwarding chains of limited length
    » Dynamic transaction ordering instead of static
    » Data locality