Efficient Detection of Determinacy Race in Transactional Cilk Programs

Xie Yong

Singapore-MIT Alliance
Outline

- **Definition** determinacy race in transactional Cilk
- **Algorithm** T. E. R. D.
- **Implementation** Cilk runtime system & cilk2c
- **Performance** Time: $O(T\alpha(v, v))$, Space: $O(v)$
  
  *Empirical: 15 times slowdown vs. serial execution*

- **Conclusion & Future Work**
- **Performance of Transactional Cilk**
  
  *Impossibility of achieving linear speedup*
Definition of Determinacy Race

- Atomization of Cilk program
- Efficiency \( \uparrow \) Size of transaction \( \downarrow \)
- Only if correctness is not affected

Kai’s definition:
- *Atomic-thread atomization*
- *Detection: NP-complete*

List Insertion (read & write “head“)
Definition of Determinacy Race

Diagram:

- **X** reads **l**
- **Y** writes **l**
- **Z** writes **l**

- **X** reads **l**
- **Y** reads **l**
- **Z** writes **l**

- **X** writes **l**
- **Y** reads **l**
- **Z** reads **l**

- **X** reads **l**
- **Y** writes **l**
- **Z** writes **l**

December 10, 2003

Copyright © Xie Yong, 2003
TERD Algorithm

Record access: 14 shadow spaces
TERD Algorithm

Spawn procedure $F$:  
\[ S_F \leftarrow \text{Make-Set}(F) \]

Return from $F'$ to $F$:  
\[ P_F \leftarrow \text{Union}(S_F, P_{F'}) \]

Sync in procedure $F$:  
\[ S_F \leftarrow \text{Union}(S_F, P_F) \]
\[ P_F \leftarrow \emptyset \]

Transaction Begin:  
\[ \text{Current-transaction-id} \leftrightarrow \text{Current-transaction-id} \]

- Extension of SP-bags algorithm
- Disjoint-Set data structure
TERD Algorithm

Read memory location $l$ by Transaction $T$ Procedure $F$:

If $(\text{trans-id-read}[l] \neq T \land \land \text{trans-id-write}[l] \neq T)$

\[
\text{trans-id-read}[l] \leftarrow T
\]

Eval-Read ($l, T, F$)

Eval-Read ($l, T, F$)

// check and report determinacy race

// update record (shadow spaces)
TERD Algorithm

write memory location \( l \) by Transaction \( T \) Procedure \( F \):

\[
\text{If } (\text{trans-id-write}[l] \neq T) \\
\quad \text{trans-id-read}[l] \leftarrow T \\
\quad \text{trans-id-write}[l] \leftarrow T \\
\quad \text{Eval-Write} \ (l, T, F)
\]

\[
\text{Eval-Write} \ (l, T, F) \\
\quad \text{// check and report determinacy race} \\
\quad \text{// update record (shadow spaces)}
\]
TERD Algorithm

Basic idea:
TERD Algorithm

$T$ : serial execution time
$v$ : number of shared locations being monitored
$\alpha$ : inverse of Ackermann’s function

Time: $O( T \alpha(v, v) )$

Space: $O( v )$
**Transactional Nondeterminator**

Implemented *T.E.R.D.* in *Cilk runtime system*

Cracked *Cilk compiler “cilk2c”*

Tested *15 times slowdown vs. serial execution*

<table>
<thead>
<tr>
<th>Programs</th>
<th>Serial (no T.D)</th>
<th>Serial (with T.D.)</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib (30)</td>
<td>3.1 sec</td>
<td>9.6 sec</td>
<td>3.21</td>
</tr>
<tr>
<td>C.K. (5, 8)</td>
<td>2.2 sec</td>
<td>31.2 sec</td>
<td>14.18</td>
</tr>
<tr>
<td>L.U. (512x512)</td>
<td>1.1 sec</td>
<td>10.6 sec</td>
<td>9.63</td>
</tr>
</tbody>
</table>
Transactional Cilk Performance

$T_1$ : total work for serial execution, parallel execution 

$T_\infty$ : critical path length, parallel execution 

Best case: no abort/retry, or abort/retry does not affect $T_P$

Worst case: $T_1$ (no parallelism, although many spawns)

$T_1/P \gg T_\infty \Rightarrow$ Linear Speedup

Randomized Work-Stealing
Linear Speedup: Impossible
There exists a transactional Cilk program with \( T_1 \) as the serial execution time and \( T_\infty \) as the minimum time required by the execution of infinite number of processor, where \( T_\infty \) is \( O(p^{1/2}) \), and \( T_1/p >> T_\infty \) the execution time on \( p \) processor is greater or equal to \( p^{1/2} (T_1/p) \) – not linear speedup.
Linear Speedup: Impossible

\( p \) is total number of processors

\( X_n \) is the number of working processors

\( Y_n \) is the number of trapped processors

\( n \) is from 1 to \( T_\infty \), \( X_1 = 1 \), \( Y_1 = 0 \)

\[
X_n = \begin{cases} 
X_n + 1 & 1 - \frac{(p-2)}{(p-1)} p^{X_n - Y_n} \\
X_n & \text{otherwise}
\end{cases}
\]

\[
Y_n = \begin{cases} 
Y_n + 1 & \frac{(p-X_n)}{(p-1)} p^{X_n - Y_n} \\
Y_n & \text{otherwise}
\end{cases}
\]
Linear Speedup: Impossible

\[ E[X_n] = -\frac{2n}{p} + \frac{n}{8} + \frac{n^2}{16p} + \frac{n^2}{4p^2} \]

\[ n = T_\infty = p^{1/2} \]

\[ E[X_n] = O(p^{1/2}) \]

Note that, \( E[X_n] \) always increasing
Conclusion & Future Work

Determinacy race definition: *Semantics*?

Algorithm and data-structure for maintaining relationship between transactions: *linear time*

More Language features: *inlet, wildcard, etc*

Performance of transactional Cilk: 😊
Backup Slides
Backup Slides

TERD algorithm & proof, lemma
N-queens Problem

Cilk char *nqueens(char *board, int n, int row)
{
    char *new_board;
    ...
    new_board = malloc(row+1);
    memcpy(new_board, board, row);
    for (j=0; j<n; j++) {
        ...
        new_board[row] = j;
        spawn nqueens(new_board, n, row+1);
        ...
    }
    sync;
}
N-queens Problem

No blocking case
N-queens Problem

blocking case
N-queens Problem

summary