Efficient Detection of Determinacy Race in Transactional Cilk Programs

Xie Yong

Singapore-MIT Alliance

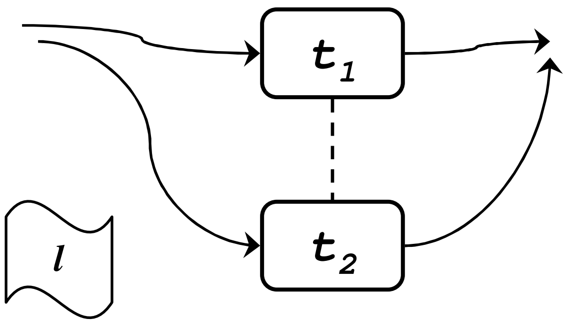
Outline

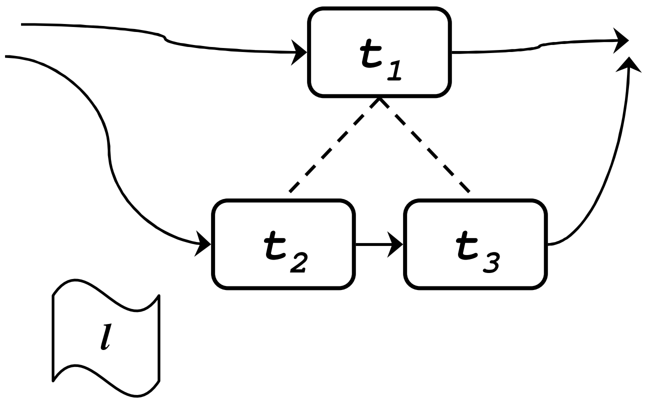
- **Definition** ***determinacy race in transactional Cilk***
- **Algorithm** ***T. E. R. D.***
- **Implementation** ***Cilk runtime system & cilk2c***
- **Performance** ***Time: $O(T\alpha(v, v))$, Space: $O(v)$***

  Empirical: 15 times slowdown vs. serial execution
- **Conclusion & Future Work**
- **Performance of Transactional Cilk**

  Impossibility of achieving linear speedup

Definition of Determinacy Race

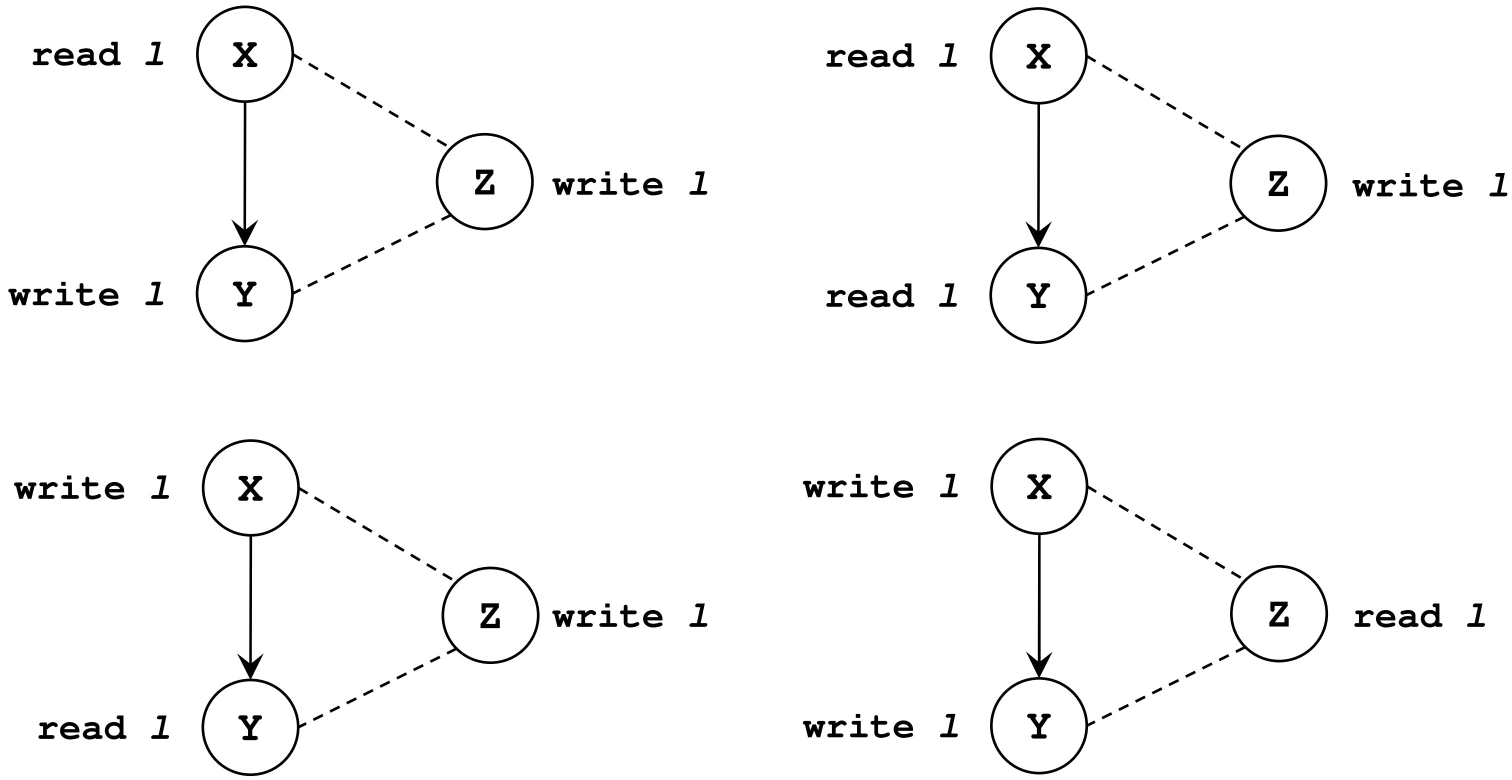




List Insertion (read & write "`head`")

- **Atomization of Cilk program**
- **Efficiency ↑ Size of transaction ↓**
- **Only if correctness is not affected**
- **Kai's definition:**
  - ***Atomic-thread atomization***
  - ***Detection: NP-complete***

Definition of Determinacy Race



TERD Algorithm

read-read-1[l],	*read-read-2[l],*
read-write-1[l],	*read-write-2[l],*
writ-read-1[l],	*write-read-2[l],*
write-write-1[l],	*write-write-2[l],*
last-read[l],	*last-parallel-read[l],*
last-write[l],	last-parallel-write[l],
trans-id-read[l],	*trans-id-write[l]*

Record access: 14 shadow spaces

TERD Algorithm

Spawn procedure F:

$$S_F \leftarrow \textit{Make-Set}(F)$$

Sync in procedure F:

$$S_F \leftarrow \textit{Union}(S_F, P_F)$$

$$P_F \leftarrow \emptyset$$

Return from F′ to F:

$$P_F \leftarrow \textit{Union}(S_F, P_{F'})$$

Transaction_Begin:

Current-transaction-id ++

- ***Extension* of *SP-bags* algorithm**
- **Disjoint-Set data structure**

TERD Algorithm

```
Read memory location l by Transaction T Procedure F:
   If (trans-id-read[l]!=T && trans-id-write[l]!=T)
            trans-id-read[l] ← T
            Eval-Read (l,T,F)

 Eval-Read (l,T,F)
      // check and report determinacy race
      // update record (shadow spaces)
```

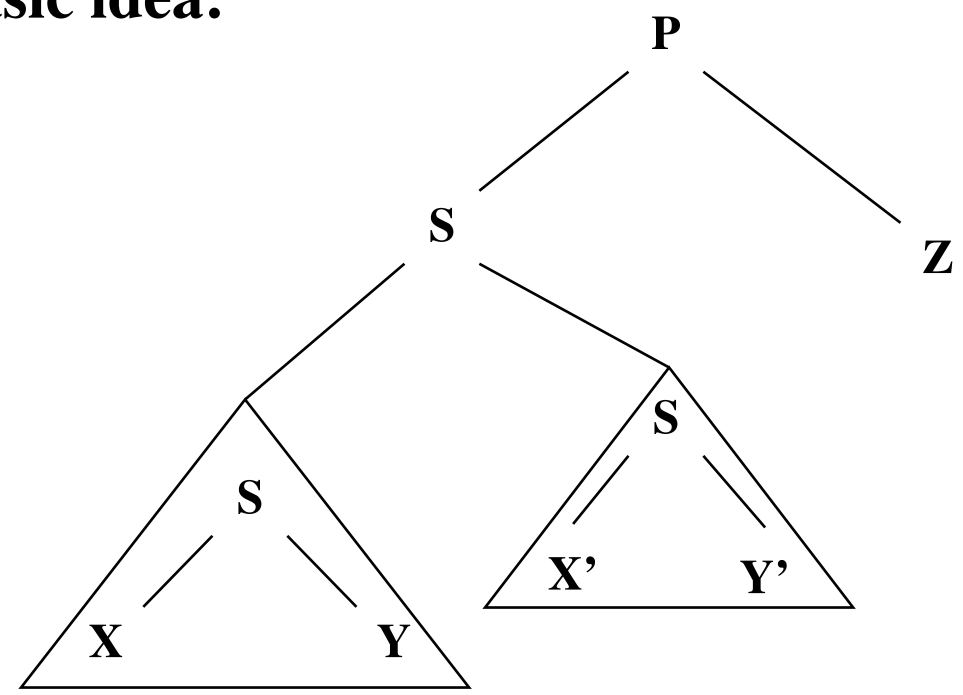
TERD Algorithm

```
write memory location l by Transaction T Procedure F:
      If (trans-id-write[l]!=T)
            trans-id-read[l] ← T
            trans-id-write[l] ← T
            Eval-Write (l,T,F)

  Eval-Write (l,T,F)
        // check and report determinacy race
        // update record (shadow spaces)
```

TERD Algorithm

Basic idea:



TERD Algorithm

T : serial execution time

v : number of shared locations being monitored

α : 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*

Programs	Serial (no T.D)	Serial (with T.D.)	Slowdown
Fib (30)	3.1 sec	9.6 sec	3.21
C.K. (5, 8)	2.2 sec	31.2 sec	14.18
L.U. (512x512)	1.1 sec	10.6 sec	9.63

Transactional Cilk Performance

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

T_∞ : critical path length, parallel execution ???

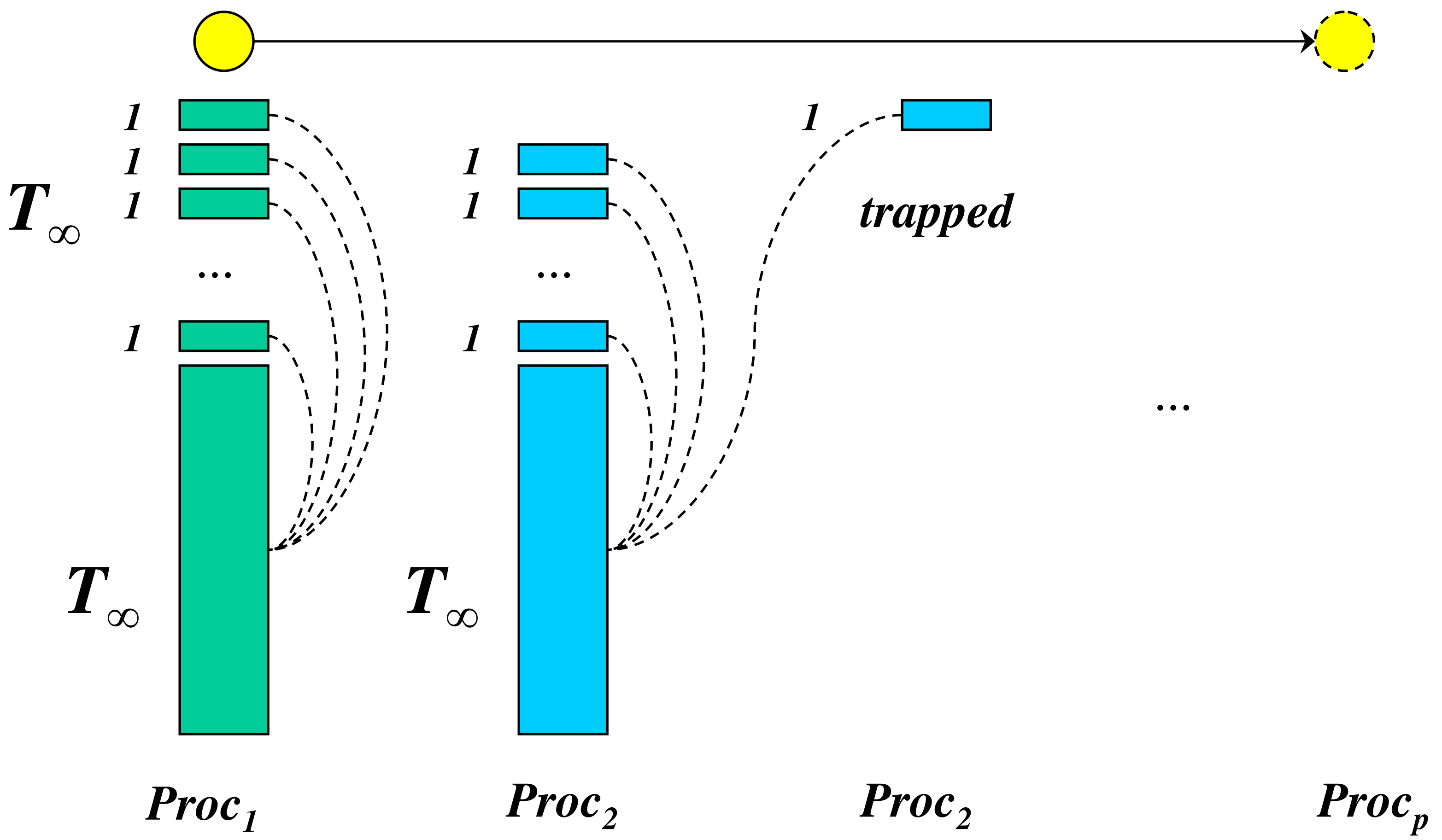
Best case: ***no abort/retry*, *or* *abort/retry does not affect T_P***

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

$T_1/P >> T_\infty$ → Linear Speedup ???

Randomized Work-Stealing

Linear Speedup: Impossible



Linear Speedup: Impossible

There exists a transactional Cilk program with T_1 as the serial execution time and T_∞ *as the minimum time required by the execution of infinite number of processor, where* T_∞ 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_∞ , $X_1 = 1$, $Y_1 = 0$

$$X_n = \begin{cases} X_n + 1 & 1-((p-2)/(p-1))^{p-Xn-Yn} \\ X_n & \text{otherwise} \end{cases}$$

$$Y_n = \begin{cases} Y_n & ((p-X_n)/(p-1))^{p-Xn-Yn} \\ Y_n + 1 & \text{otherwise} \end{cases}$$

Linear Speedup: Impossible

$$E[X_n] = -2n/p + n/8 + n^2/16p + n^2/4p^2$$

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

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

Note that, E[Xn] 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