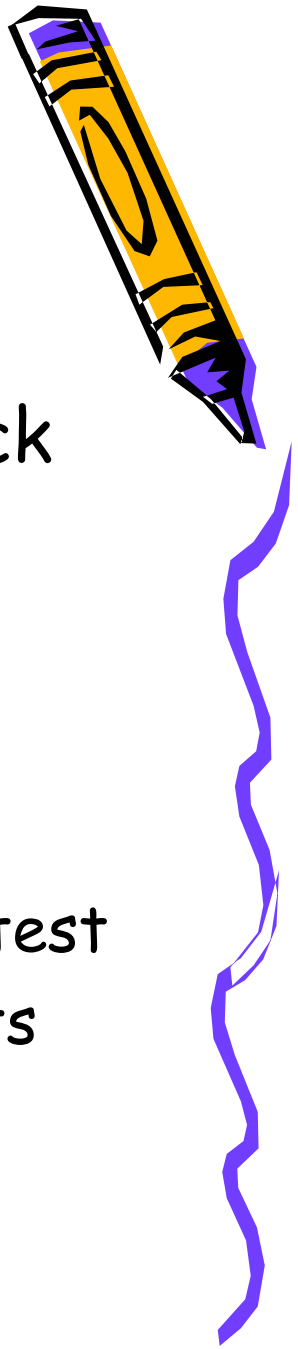
Parallel Nondeterminator

He Yuxiong, Wang Junqing

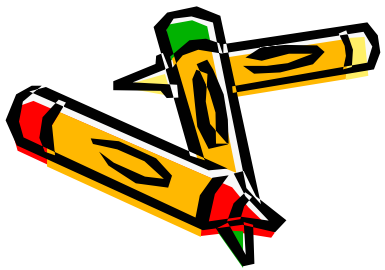
Objective



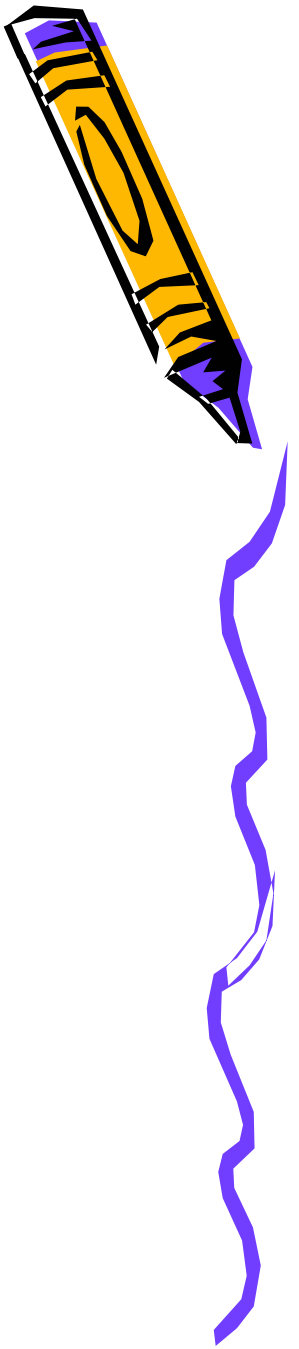
Develop Parallel Nondeterminator to check for determinacy races during the parallel execution of a Cilk program.

Two main tasks:

- Develop efficient algorithm for concurrency test
- Evaluate its performance through experiments



Outline

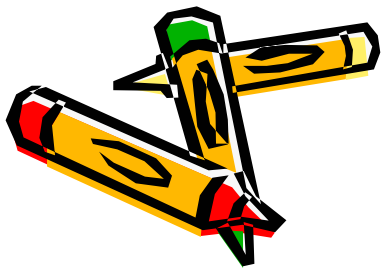


Algorithm: modified NR Labeling (MNR)

- » make it applicable to Cilk
- » correct and systematic proof
- » improve the performance

Simulation:

- » Performance analysis
- » Simulator design and implementation
- » Experimental results



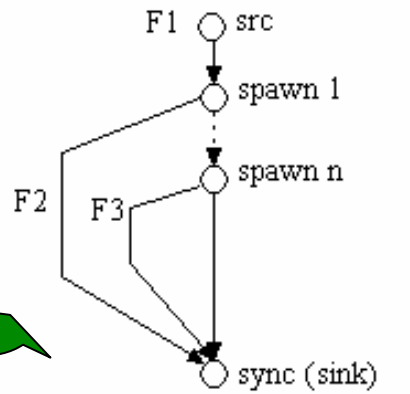
Cilk POEG

Cilk POEG is the partial order execution graph of Cilk Program.

- Vertex – sync, spawn operation
- Edge – thread

Cilk POEG Unit

- basic constructive block

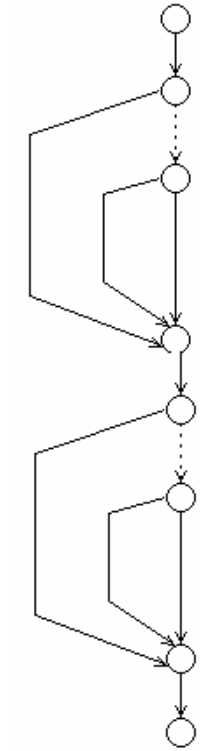


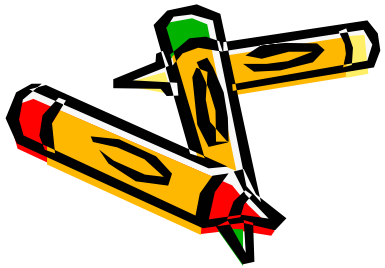
Cilk POEG

Cilk POEG is the recursive composition of series of Cilk POEG units

Series of Cilk POEG units

series composition of Cilk POEG units with an extended sink node

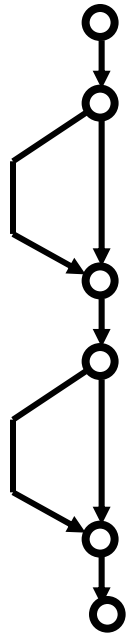


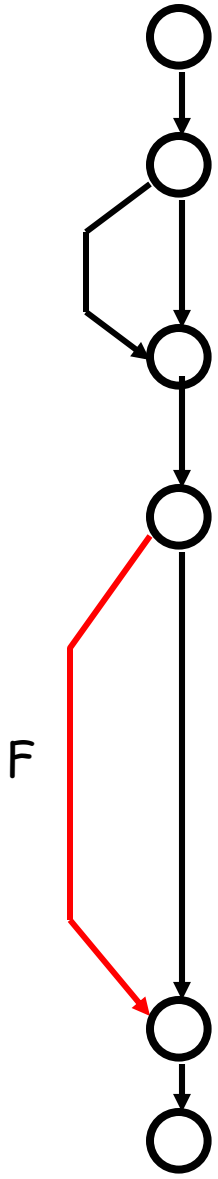


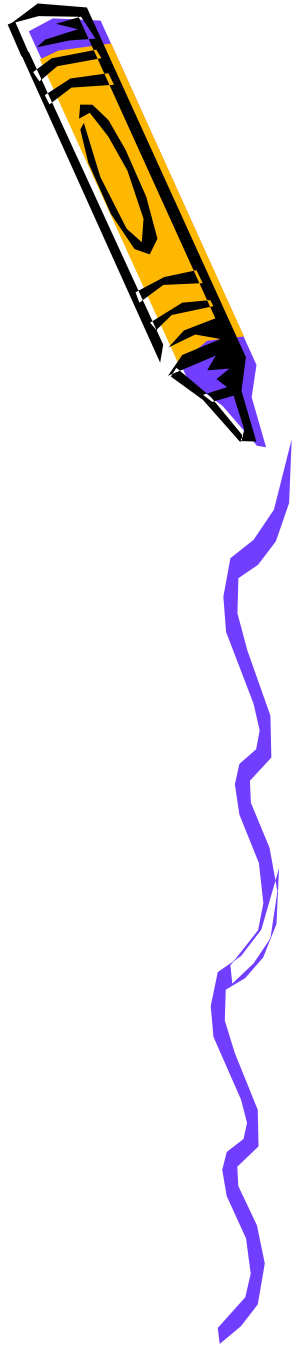
Cilk POEG

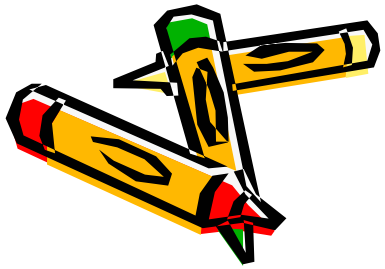
Recursive Composition

replace any basic function F by the series of cilk POEG units





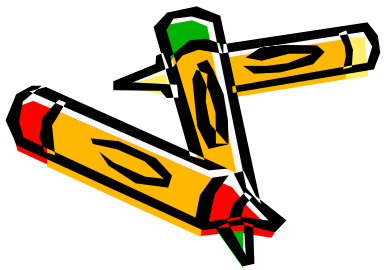


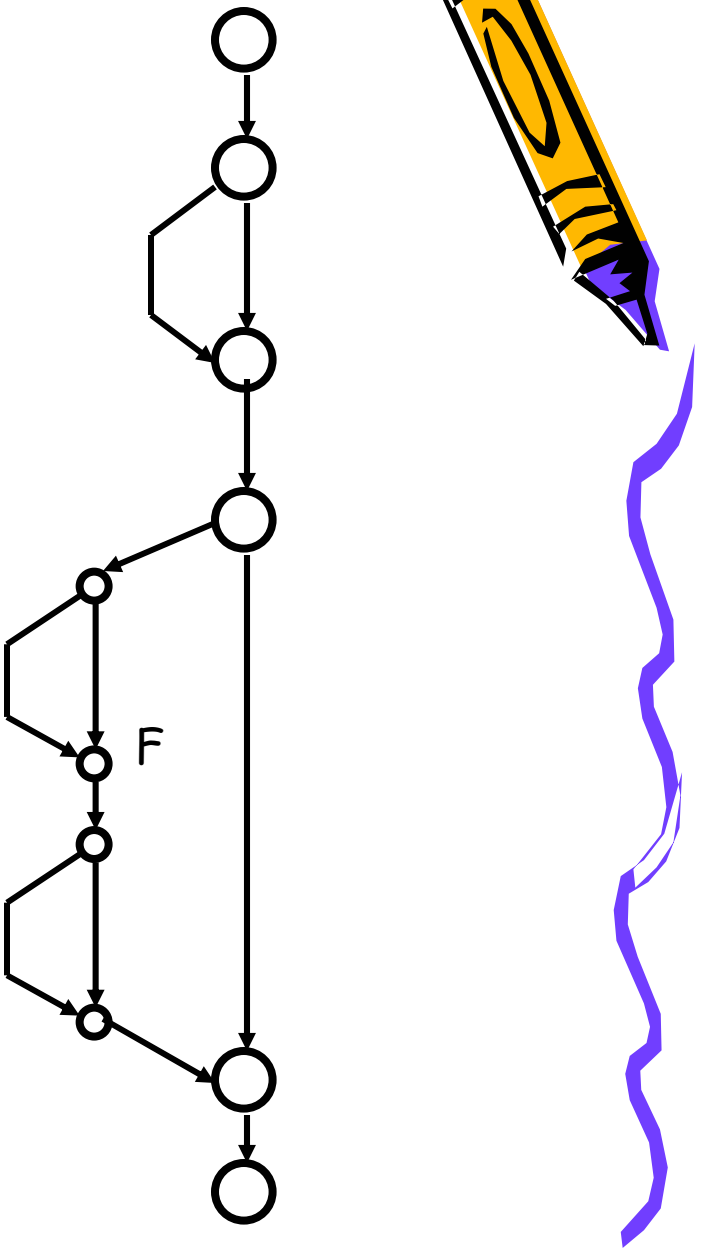


Cilk POEG

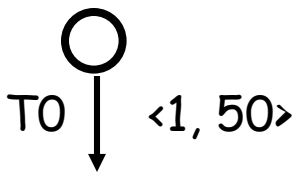
Recursive Composition

replace any basic function F with the series of cilk POEG units

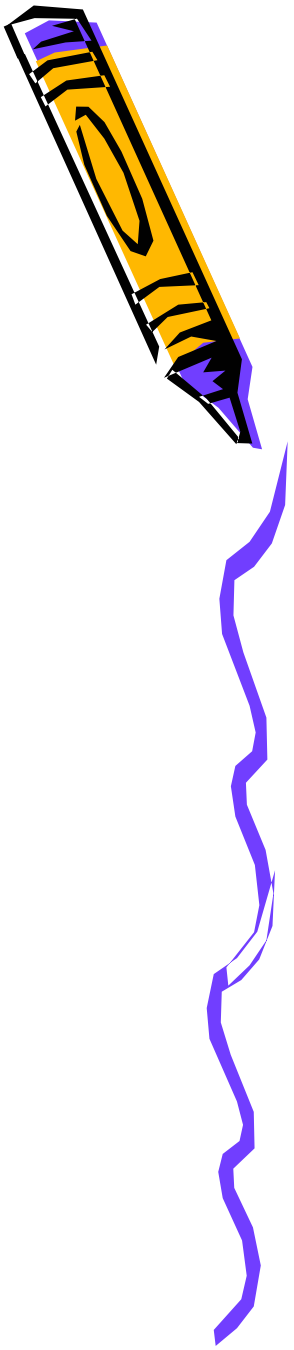


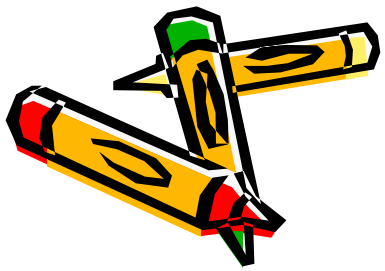


MNR Labeling

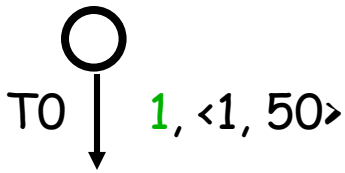


Basic Label:
NR(T0)=<1, 50>





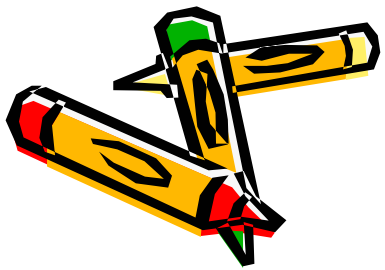
MNR Labeling



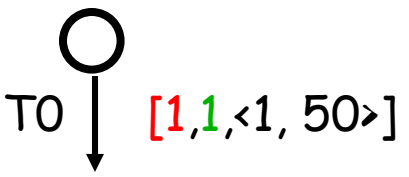
Basic Label:
NR(T0)=<1, 50>

Sync counter:
Sc(T0)=1

Syn counter of T:
the number of the sync threads of T in the critical path from initial thread to T



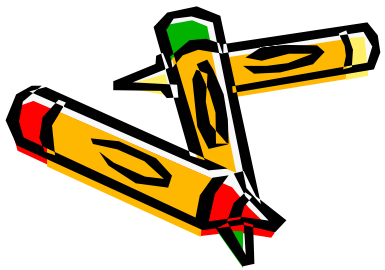
MNR Labeling



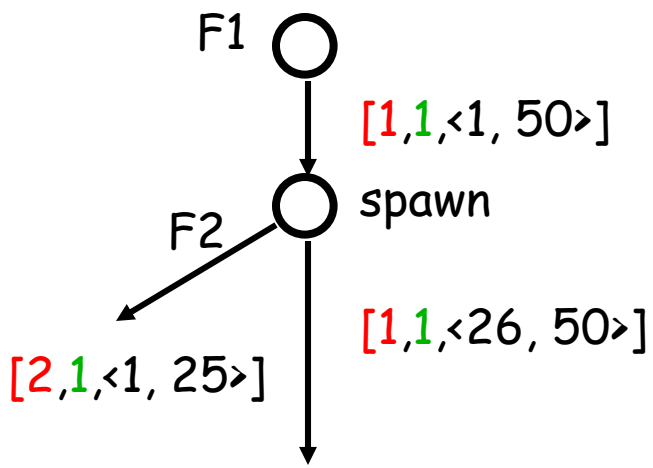
Basic Label:
NR(T0)=<1, 50>

Sync counter:
Sc(T0)=1

Recursive depth+1:
Rd(T0)=1



MNR Labeling



Spawn Operation

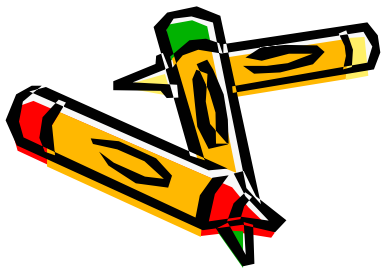
Child:

(1) Divide and inherit the first half of the range in parent thread

(2) Update recursive depth

Parent:

(1) Inherit the second half of the range in the parent thread before spawn



MNR Labeling

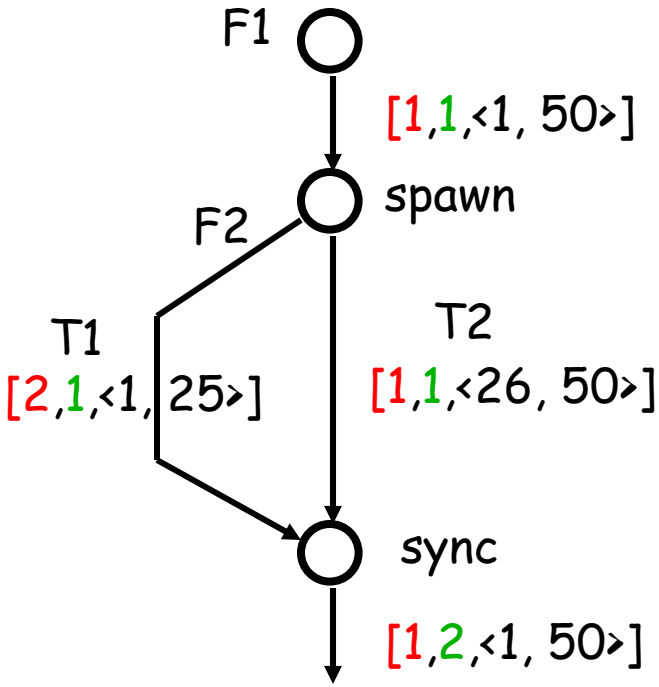
Sync Operation

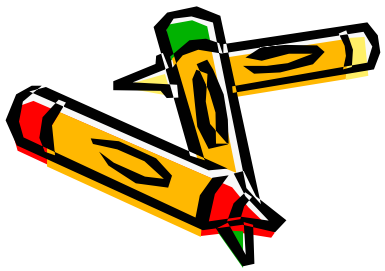
Child:
(1) Return sync count

Parent:
(1) Restore the initial range
(2) Update sync count

Concurrency Test

$Ti||Tj \Leftrightarrow NR(Ti) \cap NR(Tj) = \varepsilon$

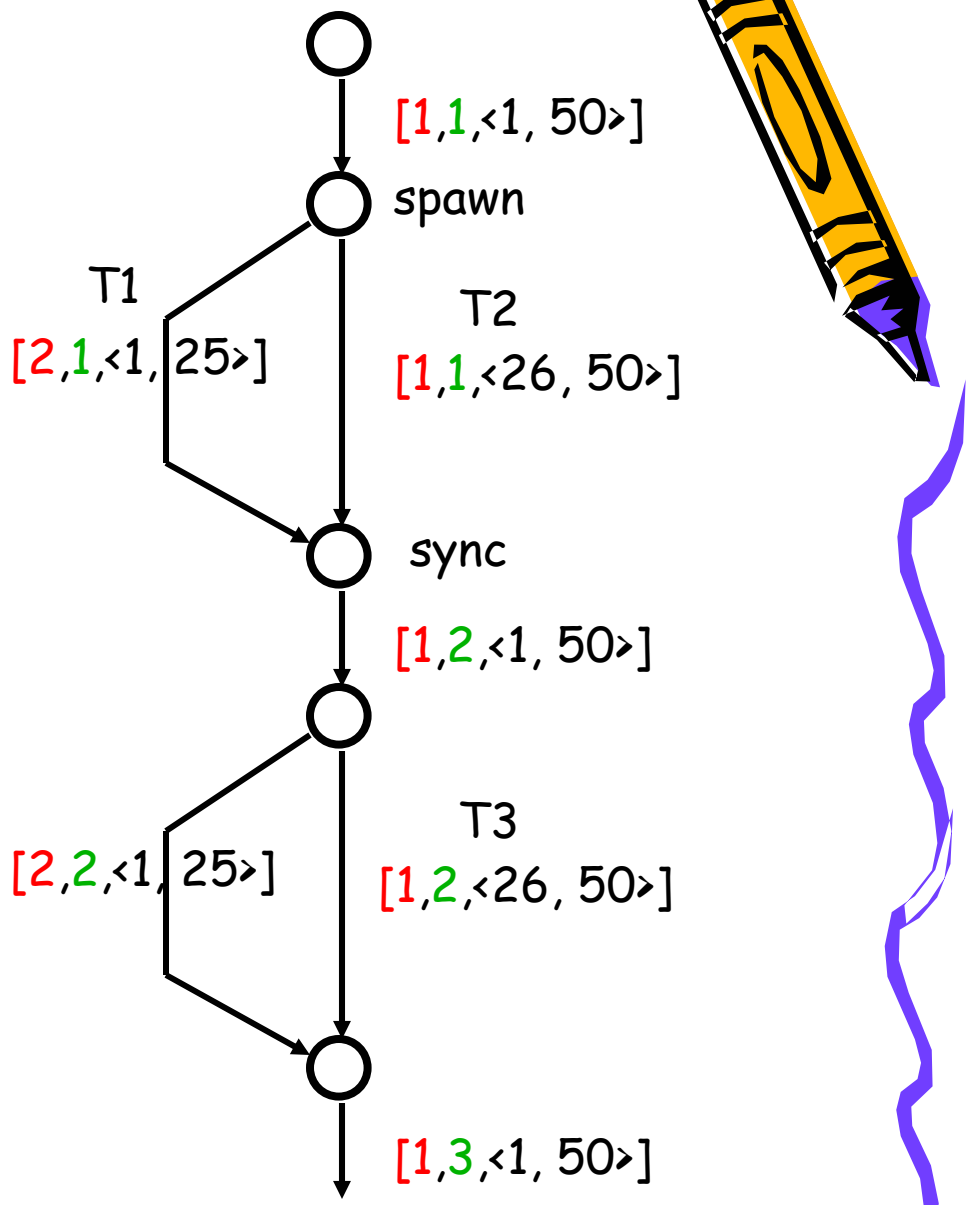


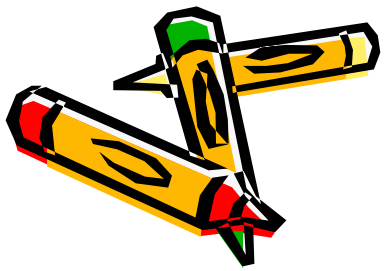


MNR Labeling

Concurrency Test

$Ti||Tj \overset{?}{\Leftrightarrow} NR(Ti) \cap NR(Tj) = \varepsilon$

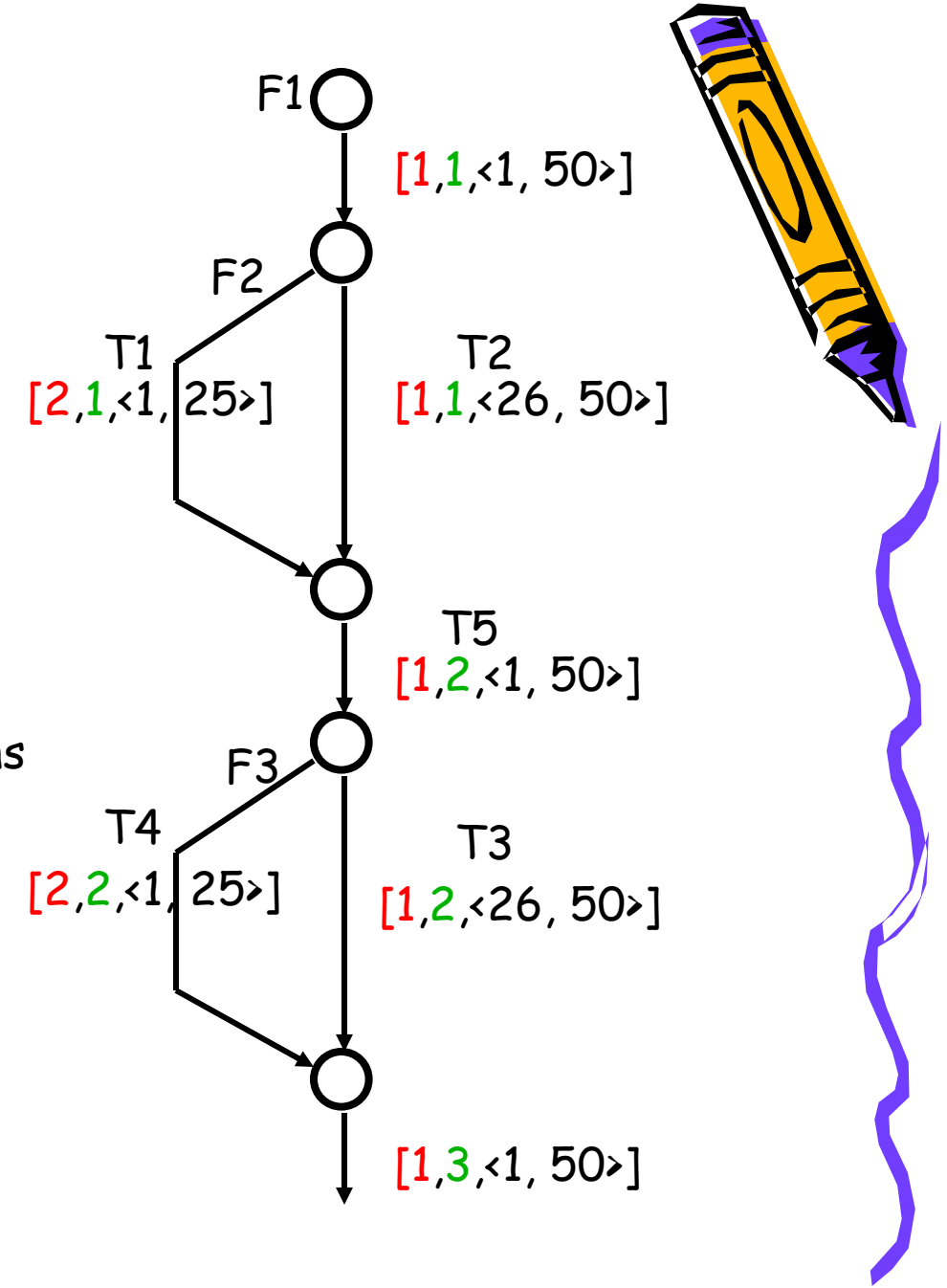


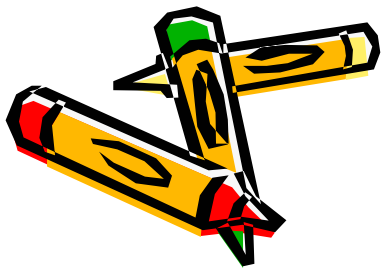


MNR Labeling

History OH(T):
Include one-way roots of the thread T

One-way root of T is the most recent sync thread happened before T in its ancestor functions and its own function

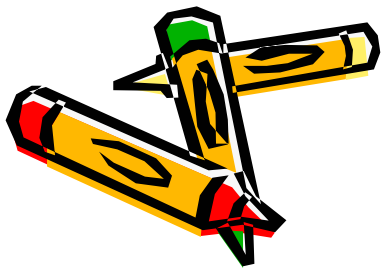


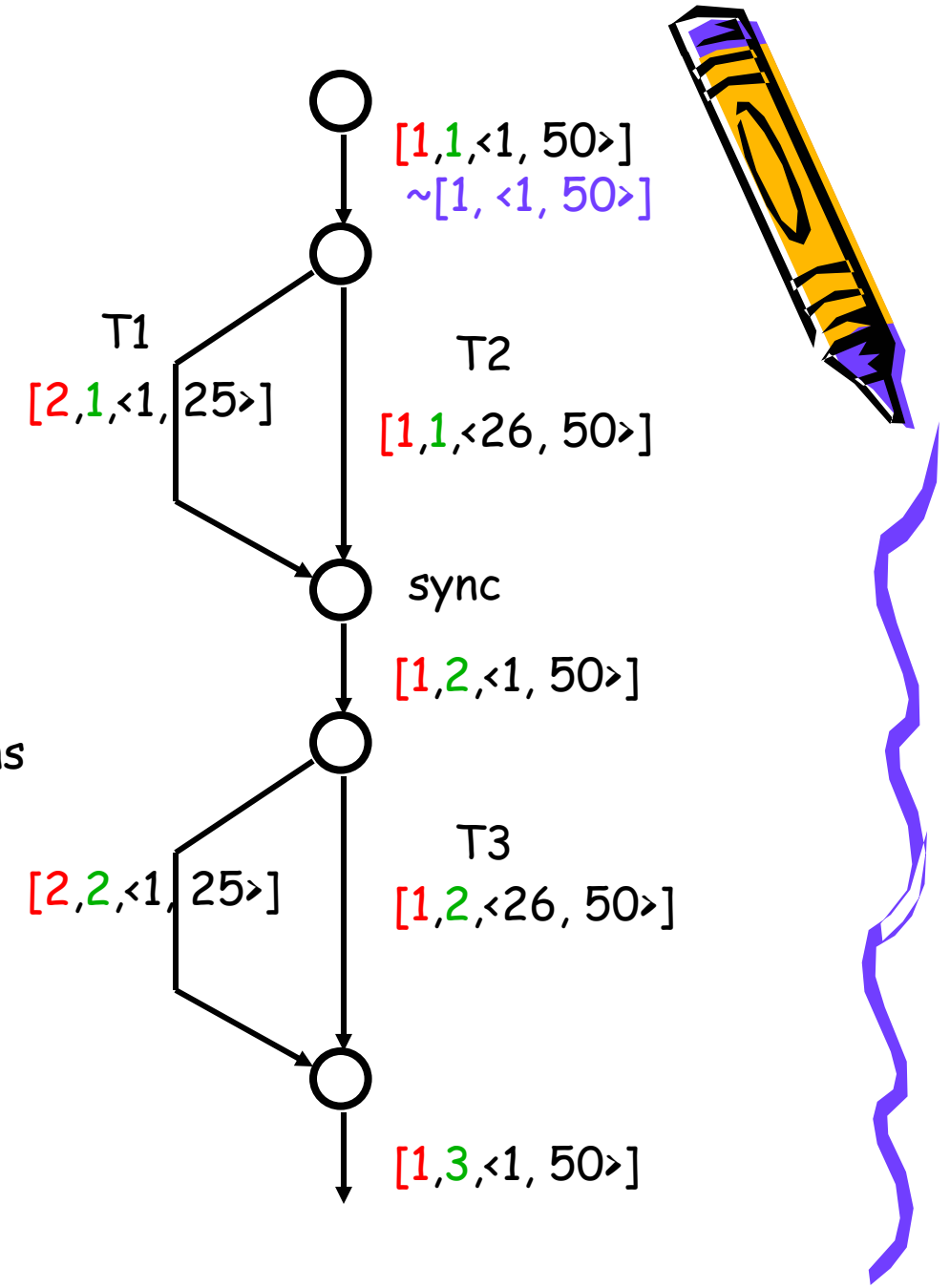


MNR Labeling

History OH(T):
Include one-way roots of the thread T

One-way root of T is the most recent sync thread happened before T in its ancestor functions and its own function





MNR Labeling

Spawn Operation

Child:

(1) Divide and inherit the first half of the range in parent thread

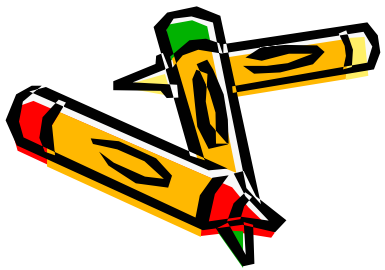
(2) Update recursive depth

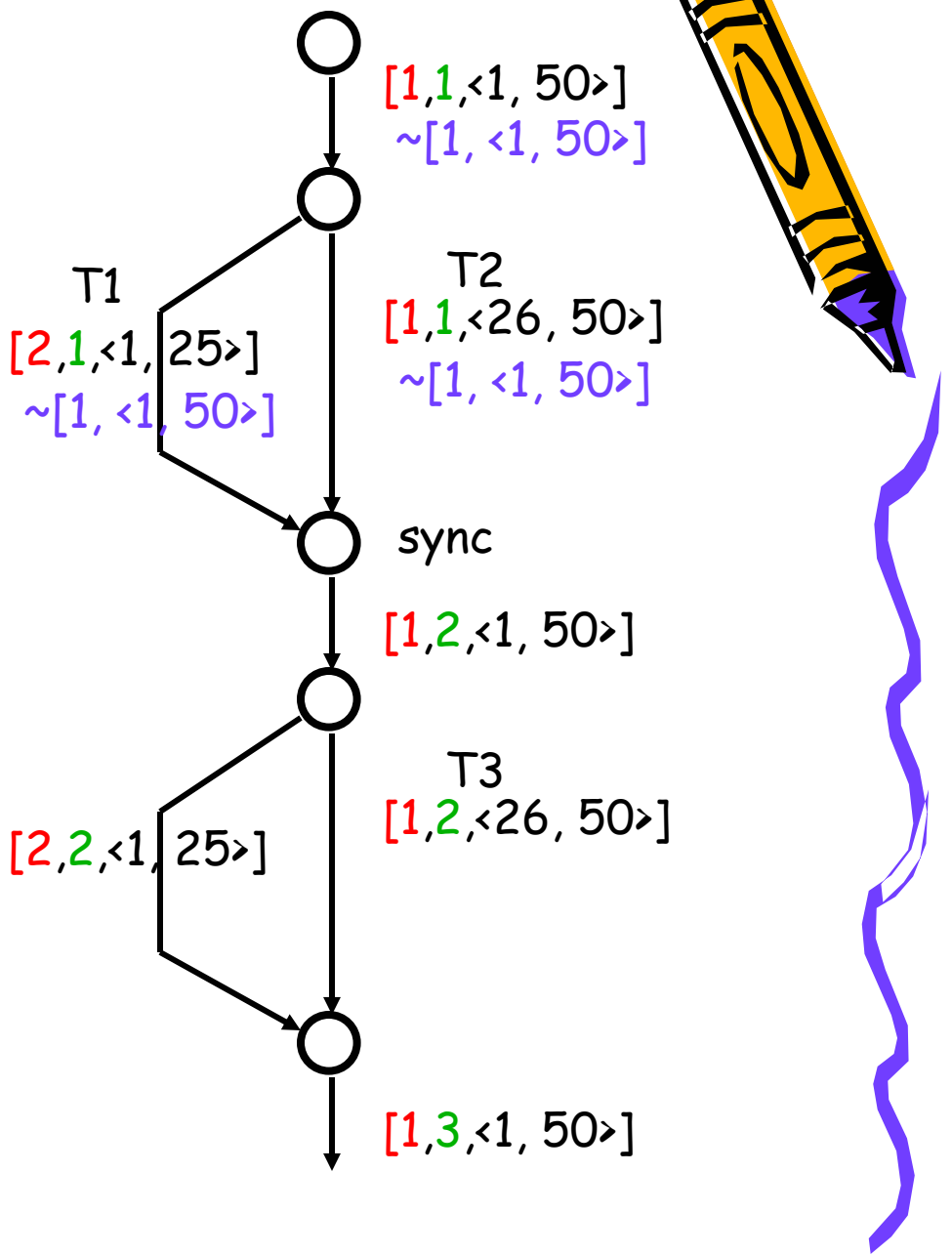
(3) Copy OH(parent) to child

Parent:

(1) Inherit the second half of the range in the parent thread before spawn

(2) OH does not change





MNR Labeling

Sync Operation

Child:

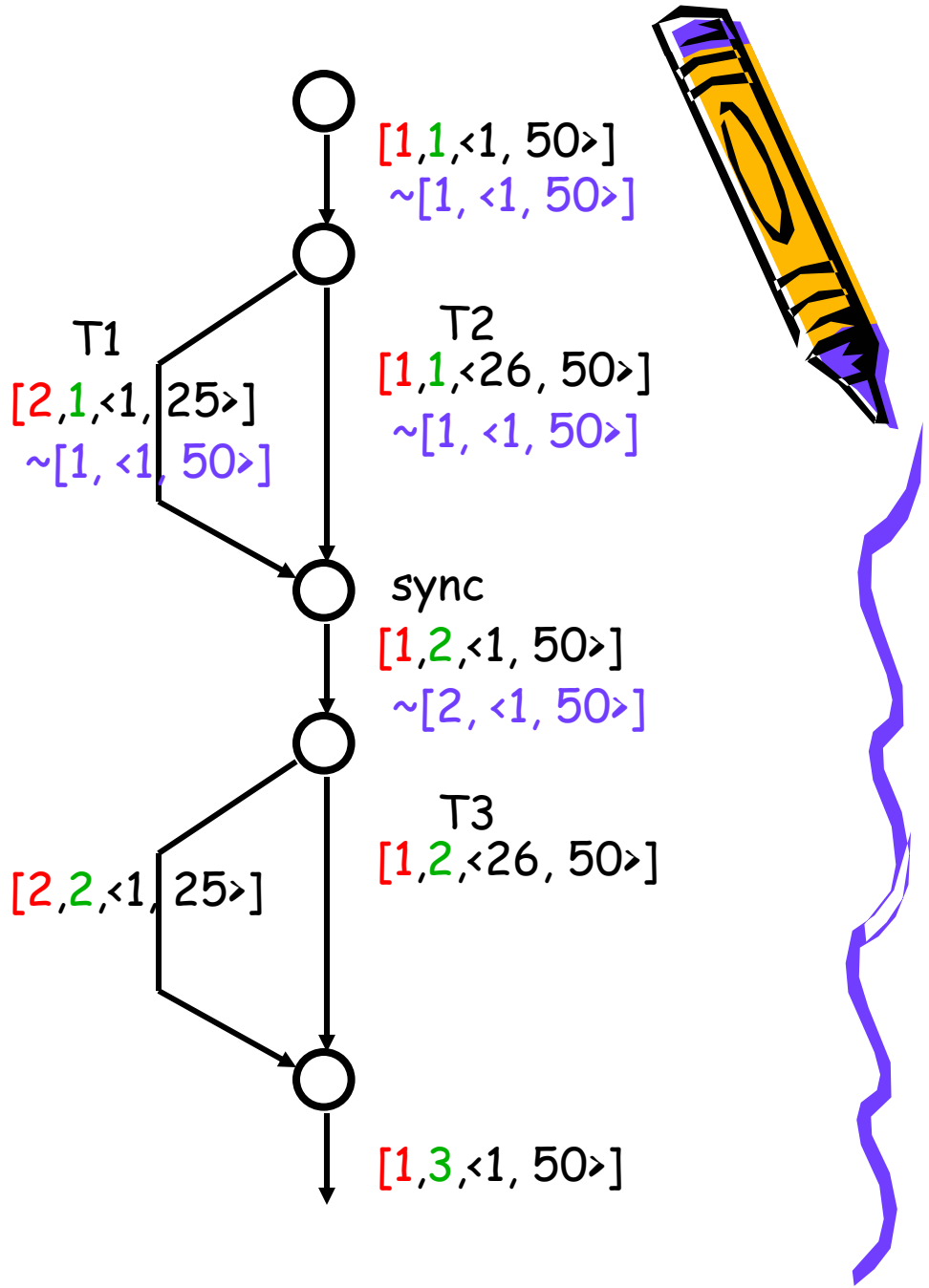
(1) Return sync count

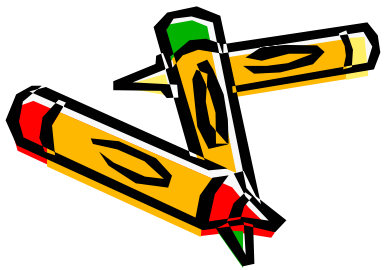
Parent:

(1) Restore the initial labeling

(2) Update sync count

(3) Update OH





MNR Labeling

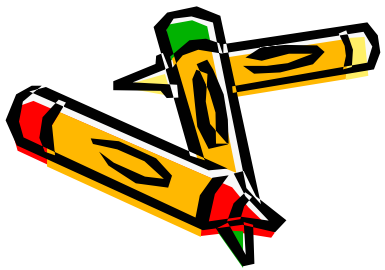
Concurrency Test

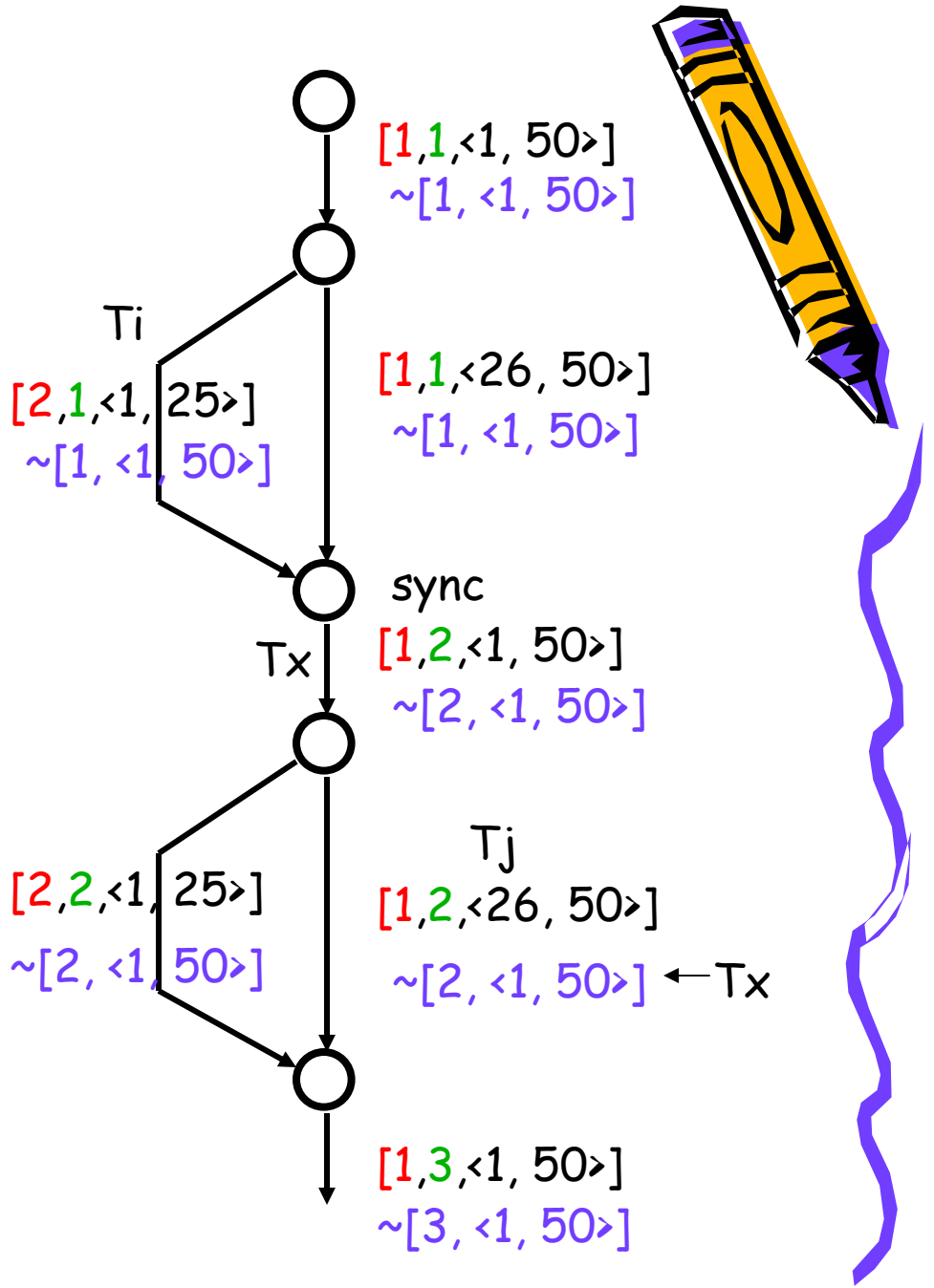
if sc(Ti) = sc(Tj)
  Ti||Tj ⇔ NR(Ti) ∩NR(Tj) = ε

if sc(Ti) < sc (Tj)
  Ti||Tj ⇔ NR(Ti) ∩NR(Tx) = ε

Tx: nearest one-way root of Tj
  from Ti.

  the thread whose sync counter is
  the smallest one in OH(Tj) such that
  it is greater than sync counter of Ti





Main Theory for MNR

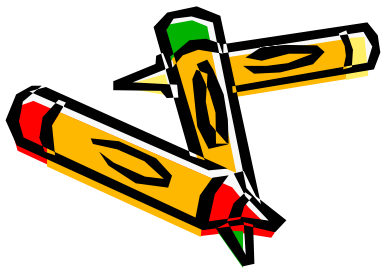
Theorem 1:

Given two threads Ti and Tj in Cilk POEG

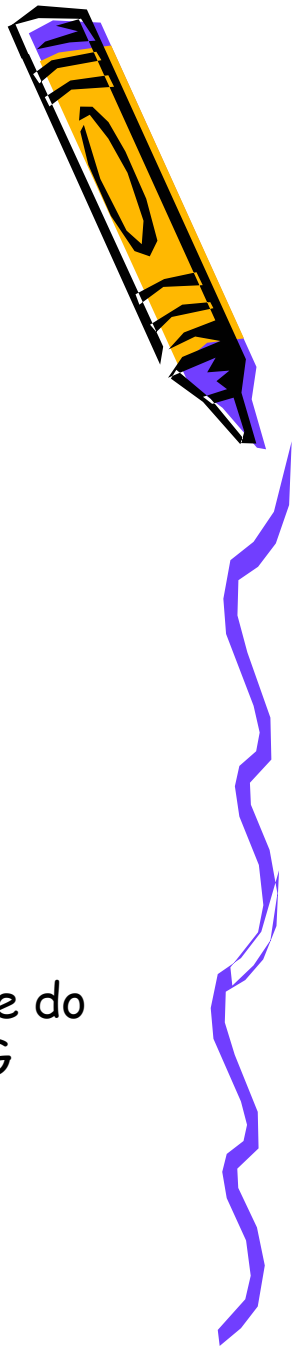
$$Ti||Tj \iff \begin{cases} NR(Ti) \cap NR(Tj) = \varepsilon & \text{if } sc(Ti) = sc(Tj) \\ NR(Ti) \cap NR(Tx) = \varepsilon & \text{if } sc(Ti) < sc(Tj) \\ NR(Tj) \cap NR(Ty) = \varepsilon & \text{if } sc(Tj) < sc(Ti) \end{cases}$$

Proof: (=>)

lemma: $Ti||Tj \Rightarrow NR(Ti) \cap NR(Tj) = \varepsilon$



Proof of Main Theorem



Proof: (=> Continued)

lemma: Ti||Tj => NR(Ti) ∩NR(Tj) = ε

proof by induction:

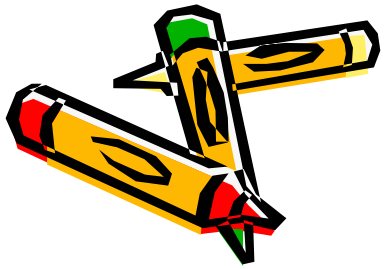
base step:

(1) true in Cilk POEG unit

(2) true in series of Cilk POEG units

induction step:

suppose lemma is true in Cilk POEG G, prove it is still true in G' which is the Cilk POEG after we do recursive composition on any basic function of G



Main Theory for MNR

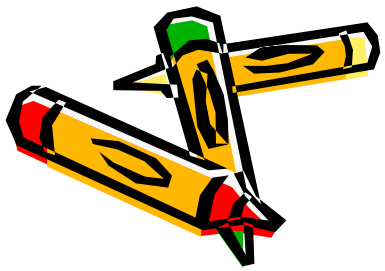
Theorem 1:

Given two threads Ti and Tj in Cilk POEG

$$Ti||Tj \iff \begin{cases} NR(Ti) \cap NR(Tj) = \varepsilon & \text{if } sc(Ti) = sc(Tj) \\ NR(Ti) \cap NR(Tx) = \varepsilon & \text{if } sc(Ti) < sc(Tj) \\ NR(Tj) \cap NR(Ty) = \varepsilon & \text{if } sc(Tj) < sc(Ti) \end{cases}$$

Proof: (=>)

lemma: $Ti||Tj \Rightarrow NR(Ti) \cap NR(Tj) = \varepsilon$



Proof of Main Theorem

Proof: (<=)

Case 1: sc(Ti) = sc(Tj)

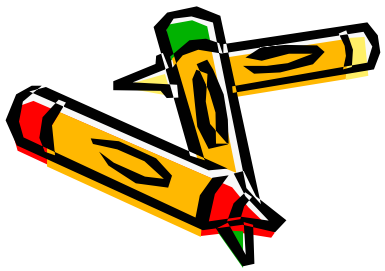
prove by contradition

(a) if Ti-> Tj or Tj-> Ti, match Ti and Tj with the threads in one-way cilk POEG

(b) Prove

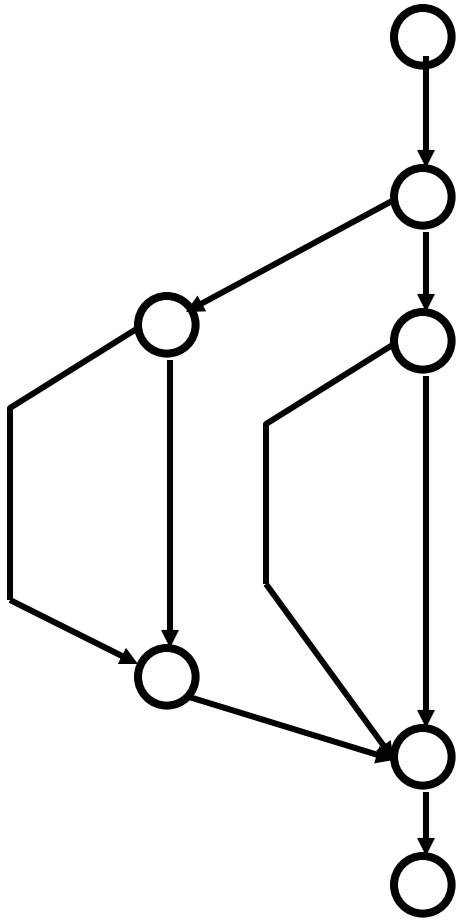
(Ti -> Tj) v (Tj -> Ti) => NR(Ti) ∩NR(Tj) != ε

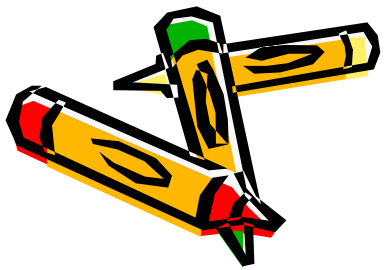
in one-way Cilk POEG



One-way Cilk POEG

Recursive composition
of Cilk POEG units





Proof of Main Theorem

Proof: (<=)

Case 1: sc(Ti) = sc(Tj)

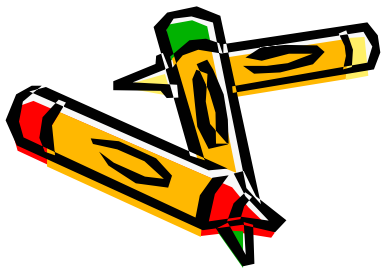
prove by contradition

(a) if Ti-> Tj or Tj-> Ti, match Ti and Tj with the threads in one-way cilk POEG

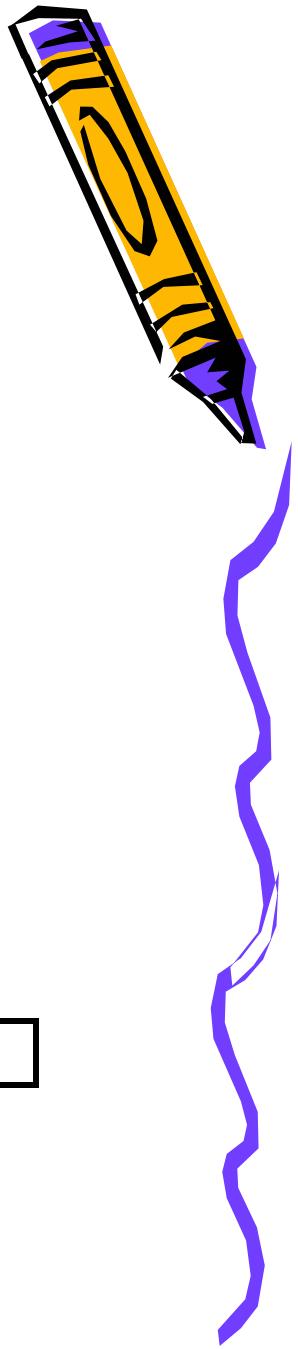
(b) Prove

(Ti -> Tj) v (Tj -> Ti) => NR(Ti) ∩NR(Tj) != ε

in one-way Cilk POEG



Proof of Main Theorem



Proof: (<= Continued)

Case 2: sc(Ti) < sc(Tj)

Lemma:

Ti -> Tj => NR(Ti) ∩NR(Tx) != ε

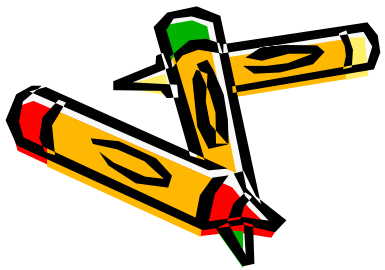
proof:

based on Gs=sub-graph(LCA(Ti, Tj)), Ti and Tj are

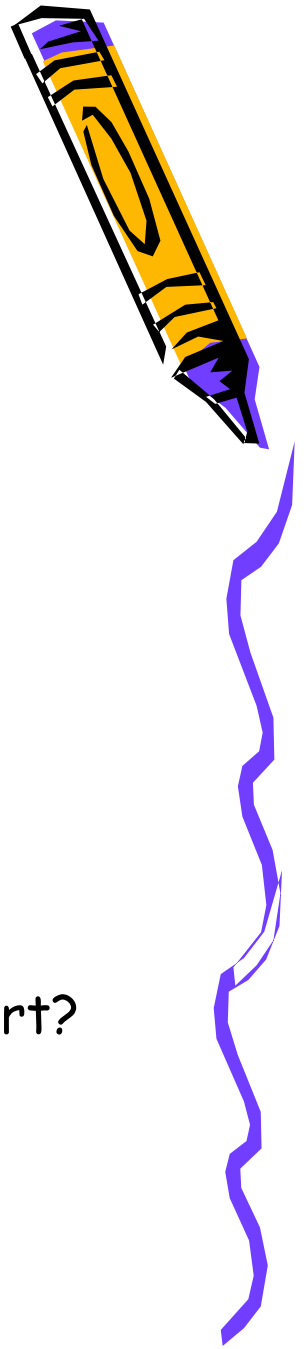
(a) in the same expanded Cilk POEG unit in Gs

(b) in different expanded Cilk POEG units in Gs

Case 3: similar way with case 2 □



Variable Monitoring Algorithm

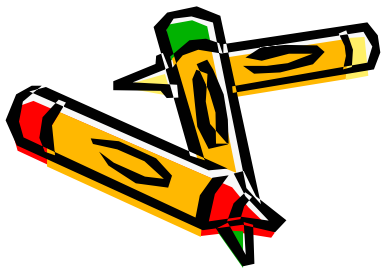


Tasks:

- Report data race
- Update the access history entries

Issues:

- How many history entries per variable?
- How to update the history entries?
- Does it guarantee correct and complete race report?



Variable Monitoring Algorithm

How many history entries per variable?

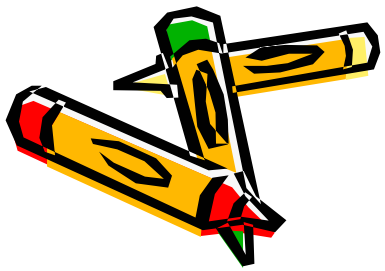
- two read entries, one write entry

How to update the history entries?

- introduce left-of operator: $Ti \prec Tj \Leftrightarrow Ti.\alpha < Tj.\alpha$
- prove left-of operator establishes a canonical ordering of relatives

Theorem 2: Given three threads Ti, Tj and Tk, Ti||Tj||Tk

Ti < Tj < Tk => LCA(Ti, Tk) = LCA(Ti, Tj, Tk)



Variable Monitoring Algorithm

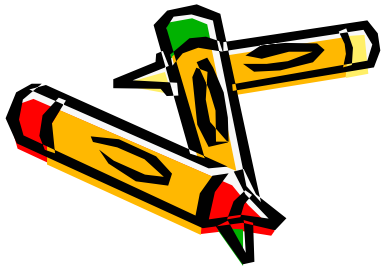
Does it guarantee correct and complete race report?

Theorem 3-6:

At least one data race will be reported if there are any conflicting, logically concurrent access in a shared variable.

Theorem 7:

Data race on a variable will only be reported if there exists conflicting, logically concurrent access.



Performance

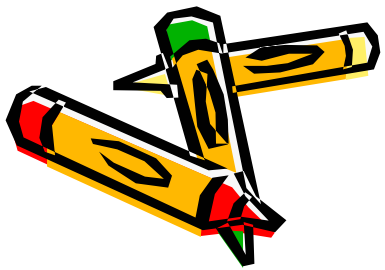
V: # of shared variables

T: maximum parallelism

N: depth of recursion

P: # of processors

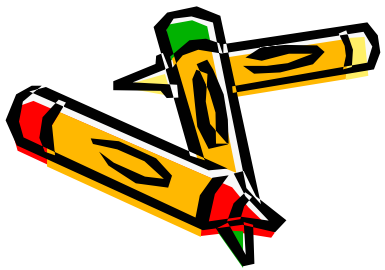
- Time Complexity
  - label updates
    -- for each spawn: O(N)
    -- for each sync: O(1)



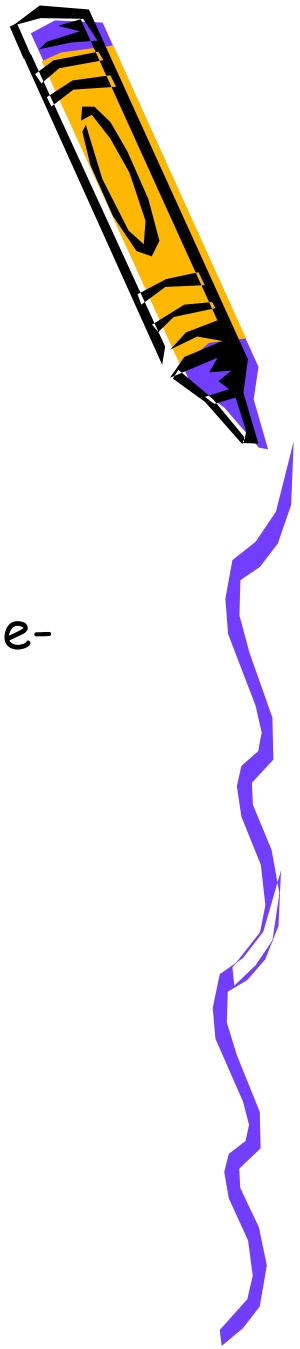
Performance (cont.)

- access history check
  -- for each shared variable access: O (lgN)

- Space Complexity
  - variable access history
    -- for each variable: O(1)
  - labels
    -- for each thread: O(N)
  -- in total: O (V+min (NP, NT))



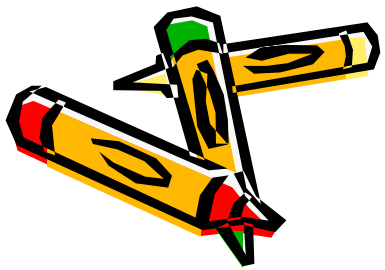
Simulation



- Generator
  - -- generate files:

    two files under the same specification, one is no race-checking cilk program, the other is for MNR Nondeterminator.

  - -- requirements:
    - random enough
    - general case



Simulation (cont.)

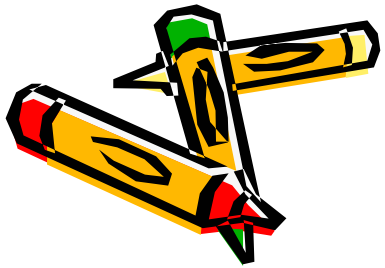
-- specifications:

- # of shared variables
- depth of recursion
- # of functions

For each function:

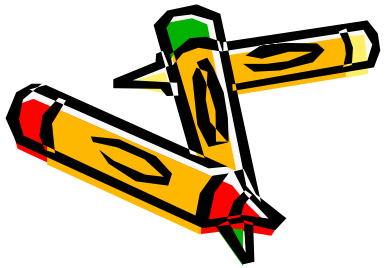
- # of syncs
- # of spawns per sync
- range of calculation delay

- Simulator Runtime Support
  - -- labeling
  - -- readcheck / writecheck operation



Testing Introduction

- Correctness
  - Sample results



```
**************************
Write-Read data race
@ var_9 in line:504
 with line:891
**************************
**************************
Write-Write data race
@ var_29 in line:529
 with line:869
**************************
**************************
Read-Write data race
@ var_29 in line:844
 with line:869
**************************
....

MNR Nondterminator
```

```
-----------------------------------
Data race @ 0x3ce5c
   `var_9' (eg1_s.cilk:408) with
   `var_9' (eg1_s.cilk:717)
-----------------------------------
-----------------------------------
Data race @ 0x33174
   `var_29' (eg1_s.cilk:428) with
   `var_29' (eg1_s.cilk:680)
-----------------------------------
-----------------------------------
Data race @ 0x33174
   `var_29' (eg1_s.cilk:428) with
   `var_29' (eg1_s.cilk:700)
-----------------------------------
....

Serial Nondeterminator in Cilk 5.2.1
```

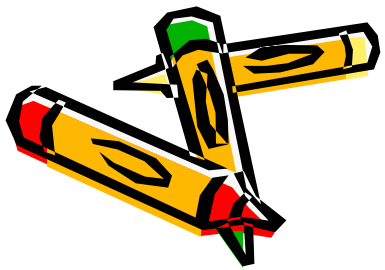
Testing Introduction (cont.)

- Result analysis

```
cilk int main(…) {
    …
spawn function_13(…);
…;
spawn function_23(…);
…;
sync;
…;
}
```

```
cilk void function_13(…){
    …
    writecheck(&hist[8],9, flabel, 504);
    var_9=234;
    …
}

cilk void function_23(…){
    …
    spawn function_13(…);
    int tmp;
    readcheck(&hist[8], 9, flabel, 891);
    tmp=var_9;
    …
}
```



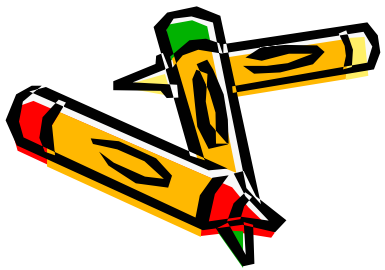
Testing Introduction (cont.)

- Result analysis

```
cilk int main(…) {
    …
spawn function_13(…);
…;
spawn function_23(…);
…;
sync;
…;
}
```

```
cilk void function_13(…){
    …
    writecheck(&hist[8],9, flabel, 504);
    var_9=234;
    …
}

cilk void function_23(…){
    …
    spawn function_13(…);
    int tmp;
    readcheck(&hist[8], 9, flabel, 891);
    tmp=var_9;
    …
}
```



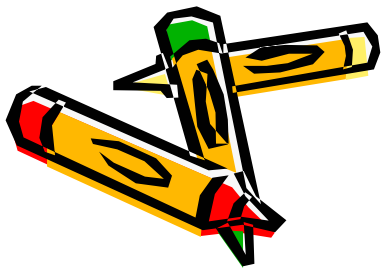
Testing Introduction (cont.)

- Result analysis

```
cilk int main(…) {
    …
spawn function_13(…);
…;
spawn function_23(…);
…;
sync;
…;
}
```

```
cilk void function_13(…){
    …
    writecheck(&hist[8],9, flabel, 504);
    var_9=234;
    …
}
```

```
cilk void function_23(…){
    …
    spawn function_13(…);
    int tmp;
    readcheck(&hist[8], 9, flabel, 891);
    tmp=var_9;
    …
}
```



Testing Introduction (cont.)

- Result analysis

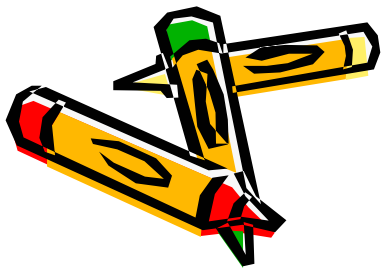
```
cilk int main(…) {
    …
spawn function_13(…);
…;
spawn function_23(…);
…;
sync;
…;
}
```

```
cilk void function_13(…){
    …
    writecheck(&hist[8],9, flabel, 504);
    var_9=234;
    …
}
```

```
cilk void function_23(…){
    …
    spawn function_13(…);
    int tmp;
    readcheck(&hist[8], 9, flabel, 891);
    tmp=var_9;
    …
}
```

WR Race

```
**********
Write-Read data race
@ var_9 in line:504
 with line:891
**********
```



Testing Introduction (cont.)

- Effectiveness

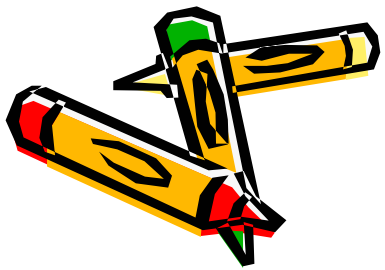
  -- We try to illustrate the performance of our MNR Nondeterminator from 4 main experiments

  - recursion
  - calculation delay
  - # of spawn per sync
  - critical path & parallelism

  -- all the tests are conducted on ygg

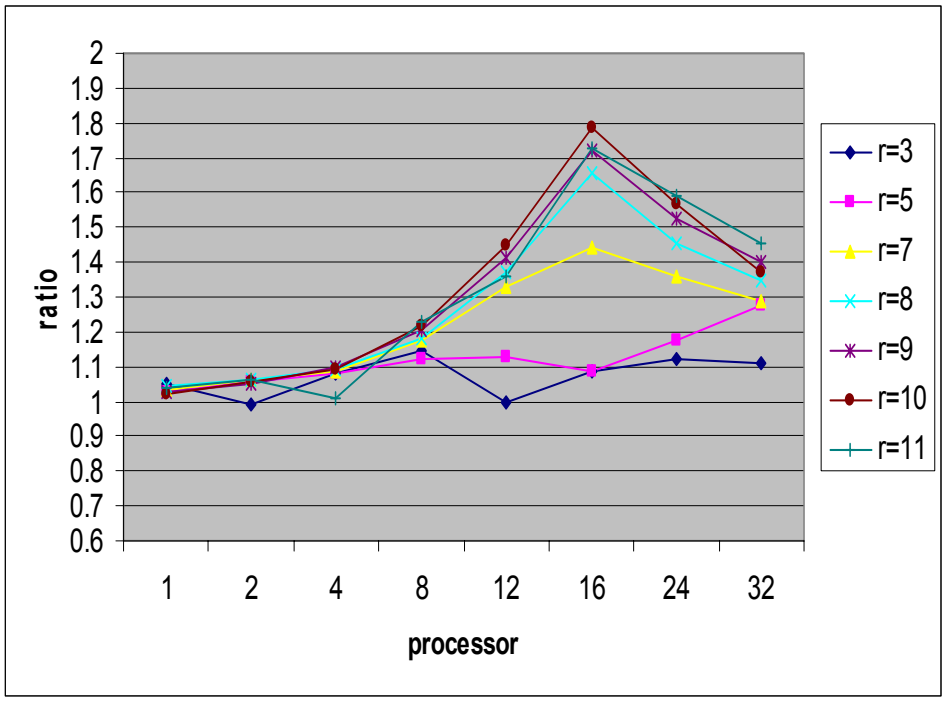
Performance Ratio:

$$\frac{\text{execution time in MNR Nondeterminator}}{\text{corresponding cilk program without race-checking}}$$



Testing (1)

-- change of recursion



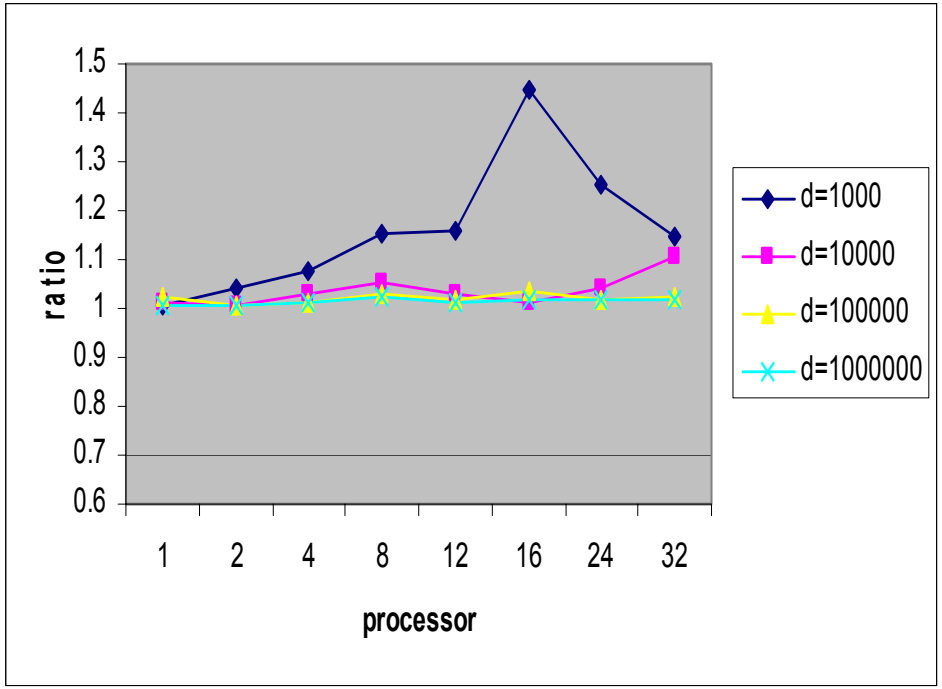
- Analysis

  - Range from 1 to 1.8, mainly restricted in the interval of 1 to 1.5

  - Relative performance drops with the increase of recursion level when the # of proc is large

Testing (2)

-- change of calculation delay

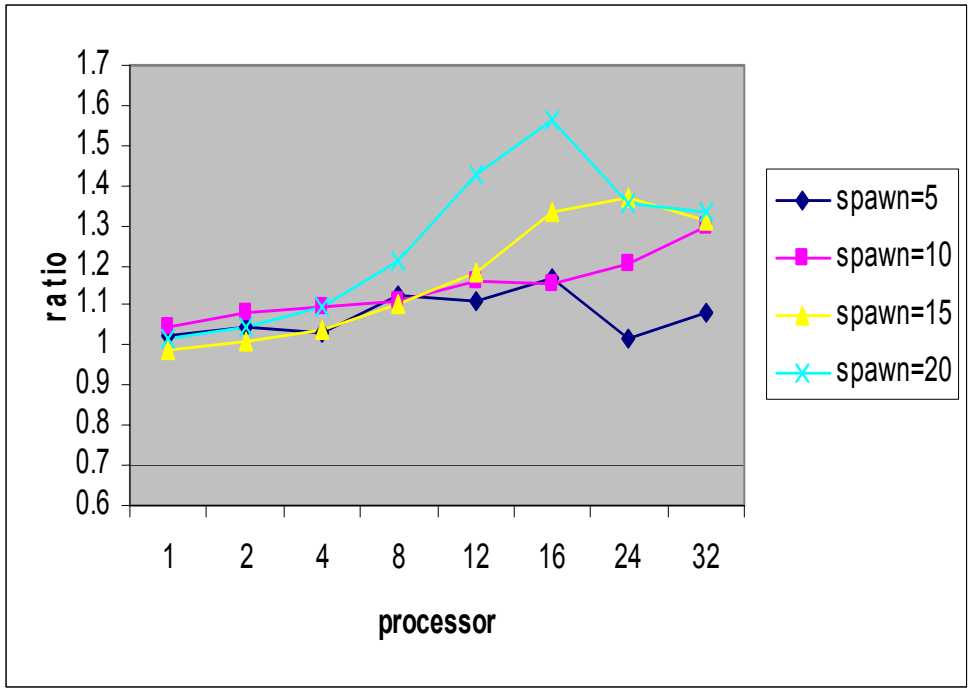


- Analysis

  - When d=1000, the performance ratio is worst
  - With the increment of calculation delay, the performance improves, the ratio is close to 1

Testing (3)

-- change of spawn per sync

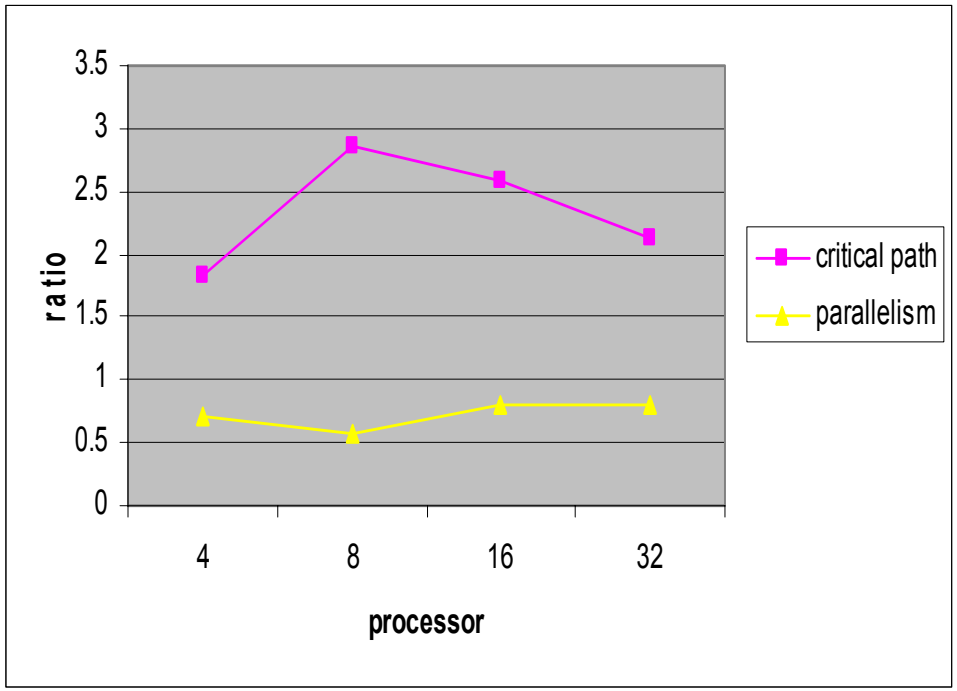


- Analysis

  - A little fluctuation around 16 proc
  - With the increment of spawn number, the lock overhead increases

Testing (4)

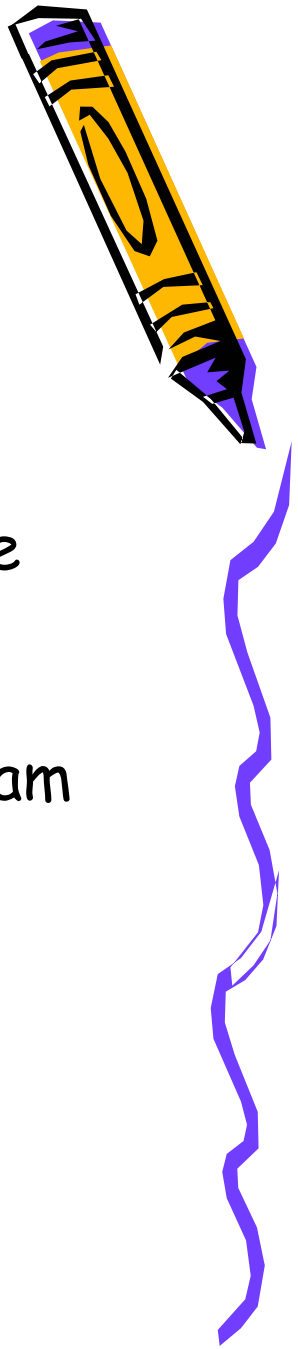
-- test on critical path & parallelism



- Analysis

➢ Critical path ranges from 2 to 3, meanwhile parallelism ranges from 0.5 to 1

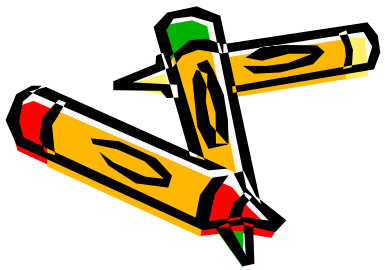
Testing Summary



- Analysis

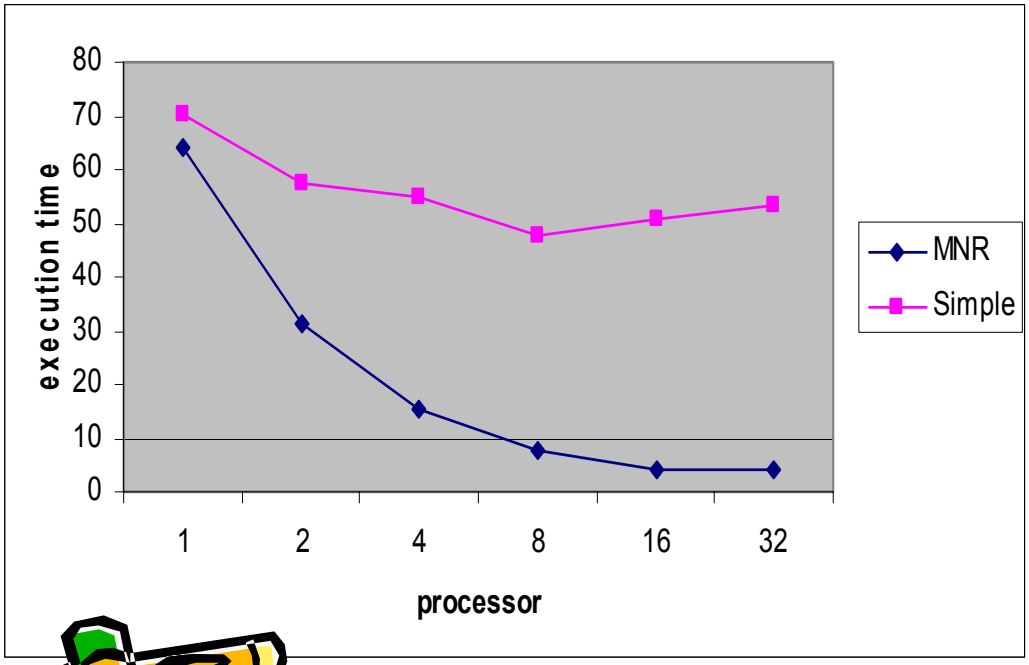
➢ From the above 4 tests, we could find that the execution time of our MNR Nondeterminator ranges generally from 1 to 1.8, compared with the corresponding no race-checking cilk program based on our testcases.

➢ Unknown phenomenon that the performance decreases when the # of proc reaches 16



Testing Summary (cont.)

- Bad case



- Analysis

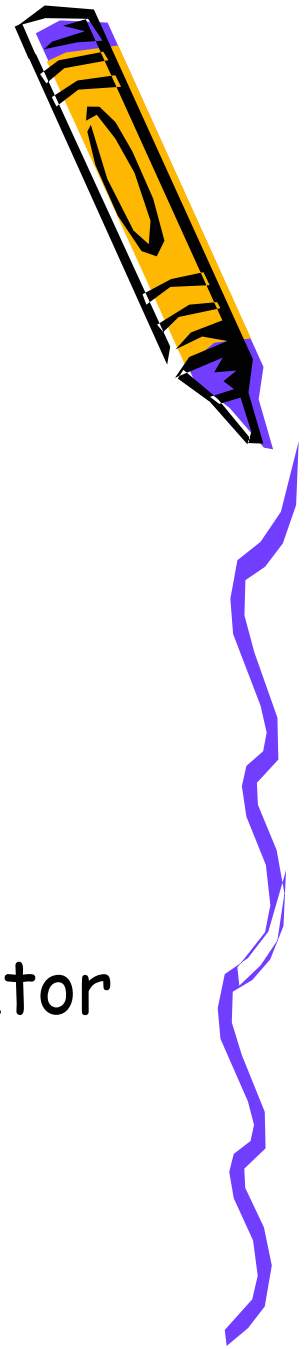
➢Testcase specification

➢Performance of 32 proc is only 2/3 of that of 1 proc

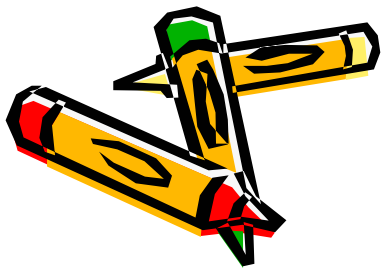
➢Lock/unlock overhead.

➢Most accesses will be held on by the locks

Testing Summary (cont.)



- Further testing
  - more testing data
  - benchmark case
  - comparison with serial Nondeterminator



Conclusions

Achievements

--MNR Labeling

time complexity:

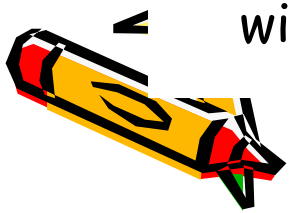
O(1) for each sync, O(N) for each spawn

O(lgN) for worst case of access check

space complexity: O(V + min (NT, NP))

-- Performance

with the change of the parameters, such as recursion, spawn number, calculation delay and processor number, the ratio to the no race-checking cilk program is generally within the range from 1 to 1.8 based on our testcases



Questions ?