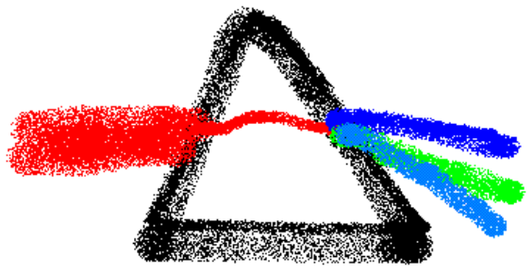
An Efficient Static Trace Simplification Technique for Debugging Concurrent Programs

Jeff Huang, Charles Zhang HKUST

{smhuang, charlesz}@cse.ust.hk

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY



Problem

- **Debugging concurrent programs is challenging**

  - **Concurrency bug is hard to reproduce**
    - *Heisenbugs*
    - Record & replay: (LEAP, FSE2010)

  - **Concurrency bug is hard to understand**
    - Too many events (ER, ISSTA2007)
    - Too many context switches

Problem

- **Debugging concurrent programs is challenging**
  - **Concurrency bug is hard to reproduce**
    - *Heisenbugs*
    - Record & replay: (LEAP, FSE2010)
  - **Concurrency bug is hard to understand**
    - Too many events (ER, ISSTA2007)
    - **Too many context switches**

Motivating Example

dynamic instructions:

thread 1

thread 2

thread 3

trace:

Motivating Example

dynamic instructions:

thread 1

thread 2

thread 3

trace:

Motivating Example

dynamic instructions:

thread 1

thread 2

thread 3

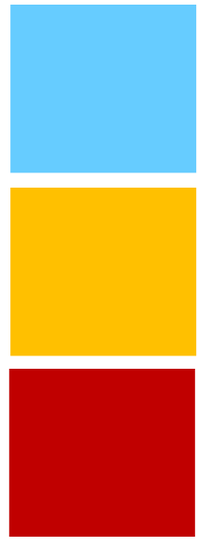
trace:

A	B	C	D	E	F	G	H	I	J

#cs=9

Motivating Example

dynamic instructions:



thread 1

thread 2

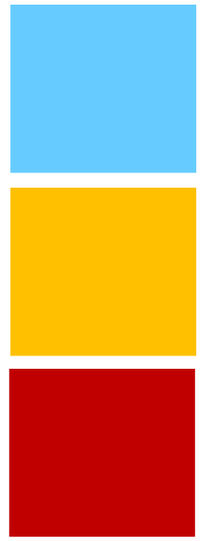
thread 3

trace:

#cs=9

Motivating Example

dynamic instructions:



thread 1

thread 2

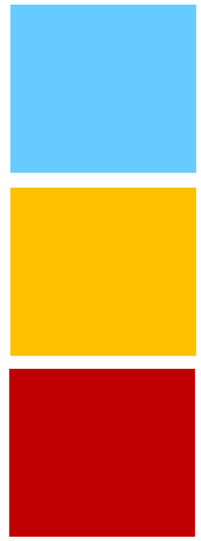
thread 3

trace:

- **Can we reduce the number of context switches? (#cs)?**

Motivating Example

dynamic instructions:



thread 1

thread 2

thread 3

trace:

- **Can we reduce the number of context switches? (#cs)?**

YES!

Motivating Example

#cs

trace: A B C D E F G H I J 9

trace': A F I B D H C E G J 2

Challenge

#cs

trace: A B C D E F G H I J 9

trace': A F I B D H C E G J 2

☝ **Can the simplified trace still manifest the bug?**

State of the Art

- **Tinertia** (Jalbert and Sen, FSE 2010)
  - A *dynamic* approach
  - A *black-box* approach

- **SimTrace** (Huang and Zhang, SAS 2011)
  - A *static* approach
  - A *white-box* approach

Outline

- Motivation
- **Problem Definition**
- SimTrace Technique
- Results
- Conclusion

Definitions

- **Event (a) :** a dynamic instruction performed by a thread

- **Trace (A = [ai]) :** a sequence of events

- **Property (Ω) :** a certain property over the program state
  - **Ω (A) = true** *if the program state driven by the trace A satisfies the property* **Ω**

Definitions

- **Trace Equivalence : A'** is equivalent to **A** iff
  - **A'** and **A** are reshuffles of each other
  - For any property **Ω, Ω (A)<-> Ω (A')**

- **EQ(A)** : the set of all traces that are equivalent to **A**

Definitions

- **Context Switch (CS):** a context switch occurs if two *consecutive* events in the trace is performed by different threads
  - Let $u_i = 1$ if thread(a_i)≠thread(a_{i-1}); otherwise, $u_i = 0$.

- **Number of Context Switches in a trace**
  - **#CS (A)** = $\sum u_i$, $i = 1, 2, \ldots, |A|$

Context Switch Minimization

- **Input: A**
- **Output: A'** s.t. **minimize #CS(A)**

A' in **EQ(A)**

Tinertia (Jalbert and Sen, FSE'10)

- **The context switch minimization problem is NP-Hard**

- **A dynamic black-box approach:**
  - ***Static Consolidation*:** merge and move
  - ***Dynamic Validation*:** deterministic replay

Tinertia (Jalbert and Sen, FSE'10)

- **Hard to scale to large traces**
  - To reduce one context switch, it requires at least one runtime validation
  - Dynamic re-execution is slow
- **O(n^3)** - n is the trace size

Our observation

- **Can we get rid of the dynamic validation part?**
  - *A completely static approach is more scalable*
- **A static white-box approach**
  - Consider the event dependence
  - Make sure all dependences are respected
  - The property **Ω** is guaranteed to be satisfied

Modeling of Dependence Relation

- Let **a_i** happened before **a_j** in the trace
- **a_j is dependent on a_i** *iff*
  - *Internal dependence* (by the same thread)
  - *Remote dependence* (by different threads)
    - fork-join, wait-notify, unlock-lock
    - **write-read, read-write, write-write**
- Write as **a_i -> a_j**

Modeling of Dependence Relation

- Let **a_i** happened before **a_j** in the trace
- **a_j is dependent on a_i** *iff*
  - *Internal dependence* (by the same thread)
  - *Remote dependence* (by different threads)
    - fork-join, wait-notify, unlock-lock
    - **write-read, read-write, write-write**
- Write as **a_i -> a_j**

This is a ***strict*** *partial order*!

A Theorem of Trace Equivalence

Any rescheduling of the events in a trace respecting the dependence relation generates an equivalent trace.

- Proof (*main insight*): by respecting the dependence relation, every event in the rescheduled trace reads or writes the same value on the program state as its corresponding event in the input trace.

SimTrace –
Efficient Static Trace Simplification

1. **Dependence Graph Construction**
   - Model all dependency relations

2. **Static Dependence Graph Simplification**
   – A graph merge algorithm

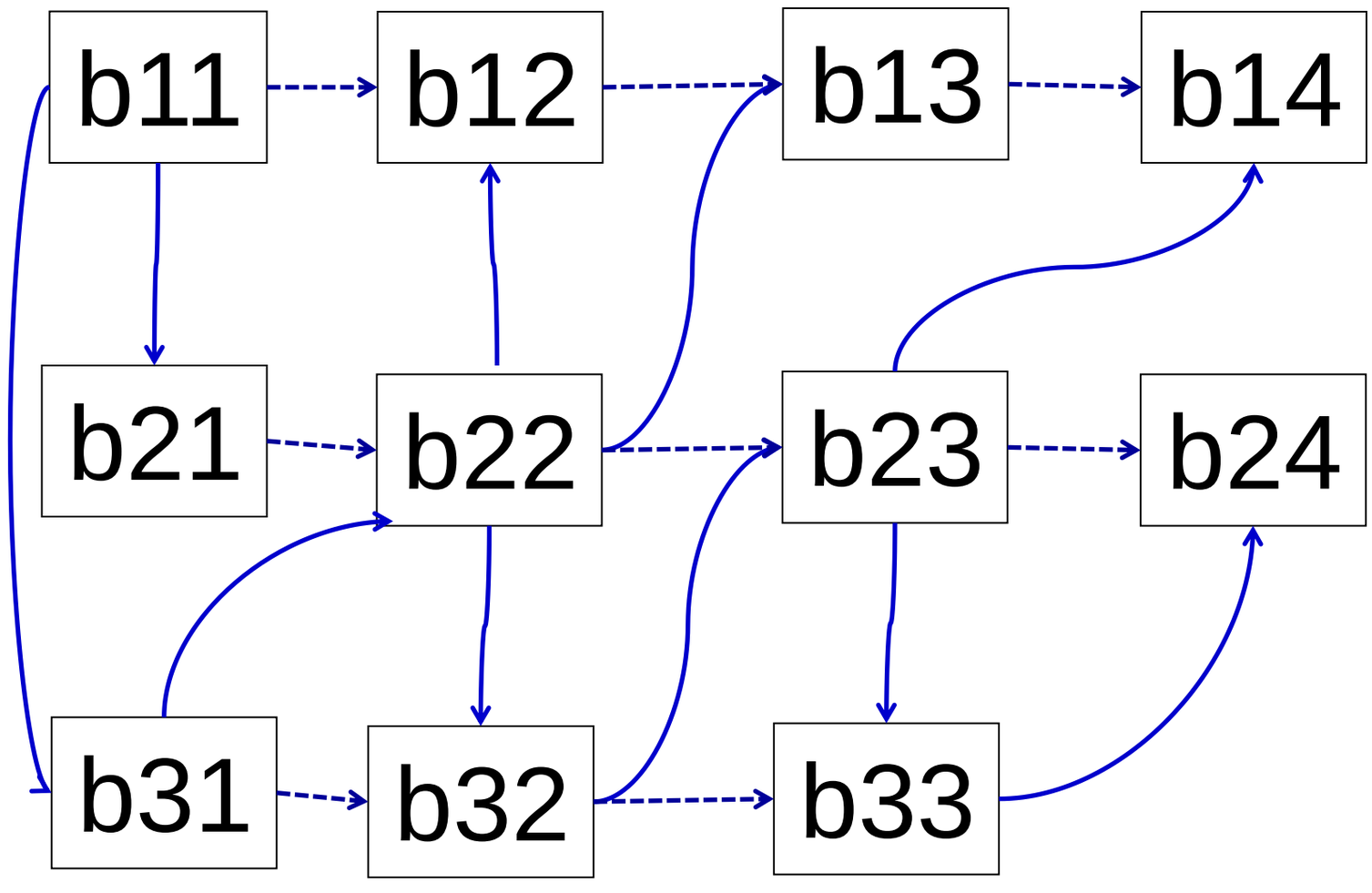
Dependency Graph Construction

- **DAG (V, E)**
- **V**: event in the trace
- **E**: dependence relations between events
- Two types of edges:
  - **Dashed edge**: internal dependence
  - **Solid edge**: remote dependence

Dependency Graph Simplification

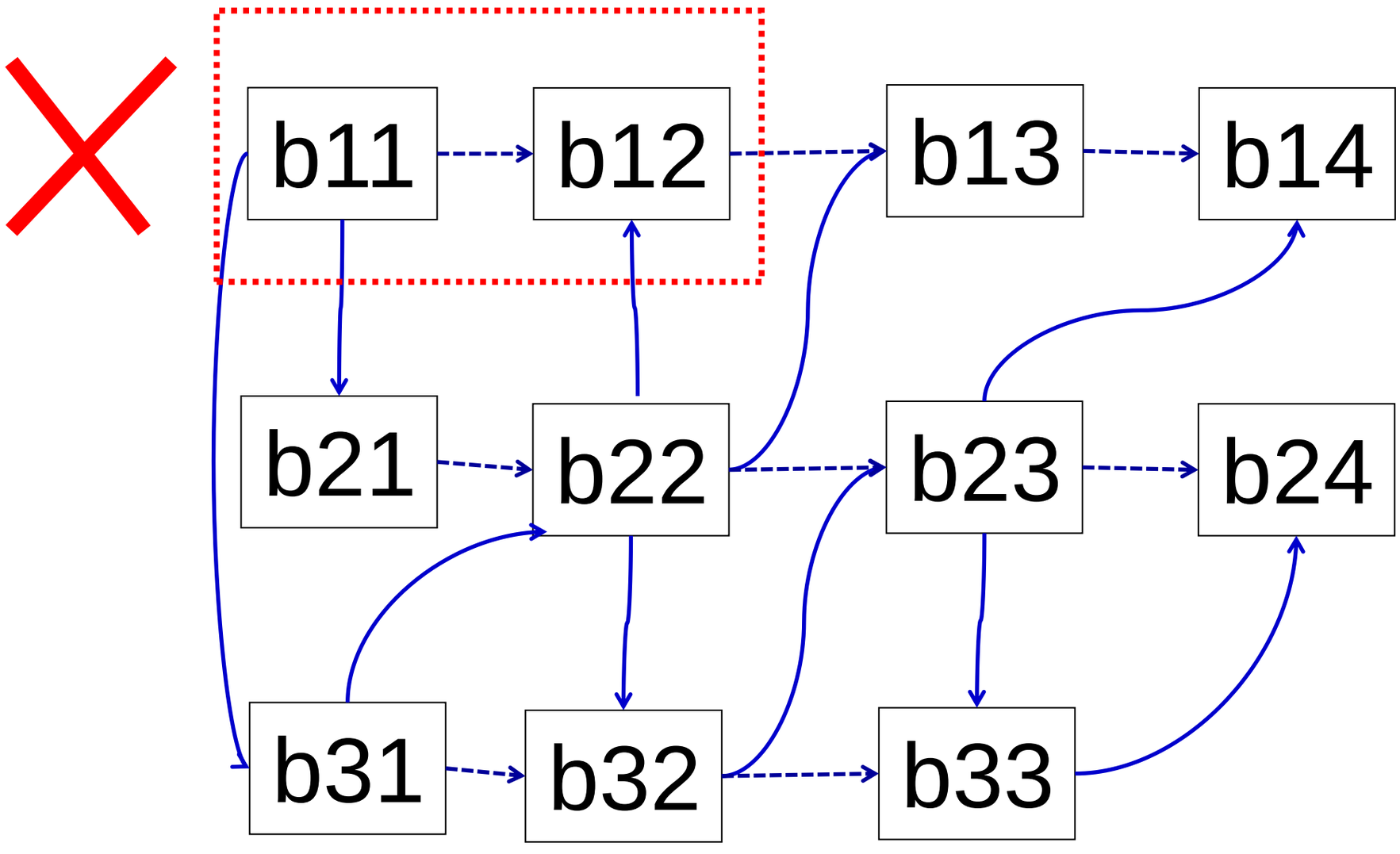
- **Merge nodes connected by dashed edges** *if the* ***merging condition*** *is satisfied:*
  - *Are they reachable in the graph without using the dashed edge connecting them?*
- **Repeat for each dashed edge**

Example



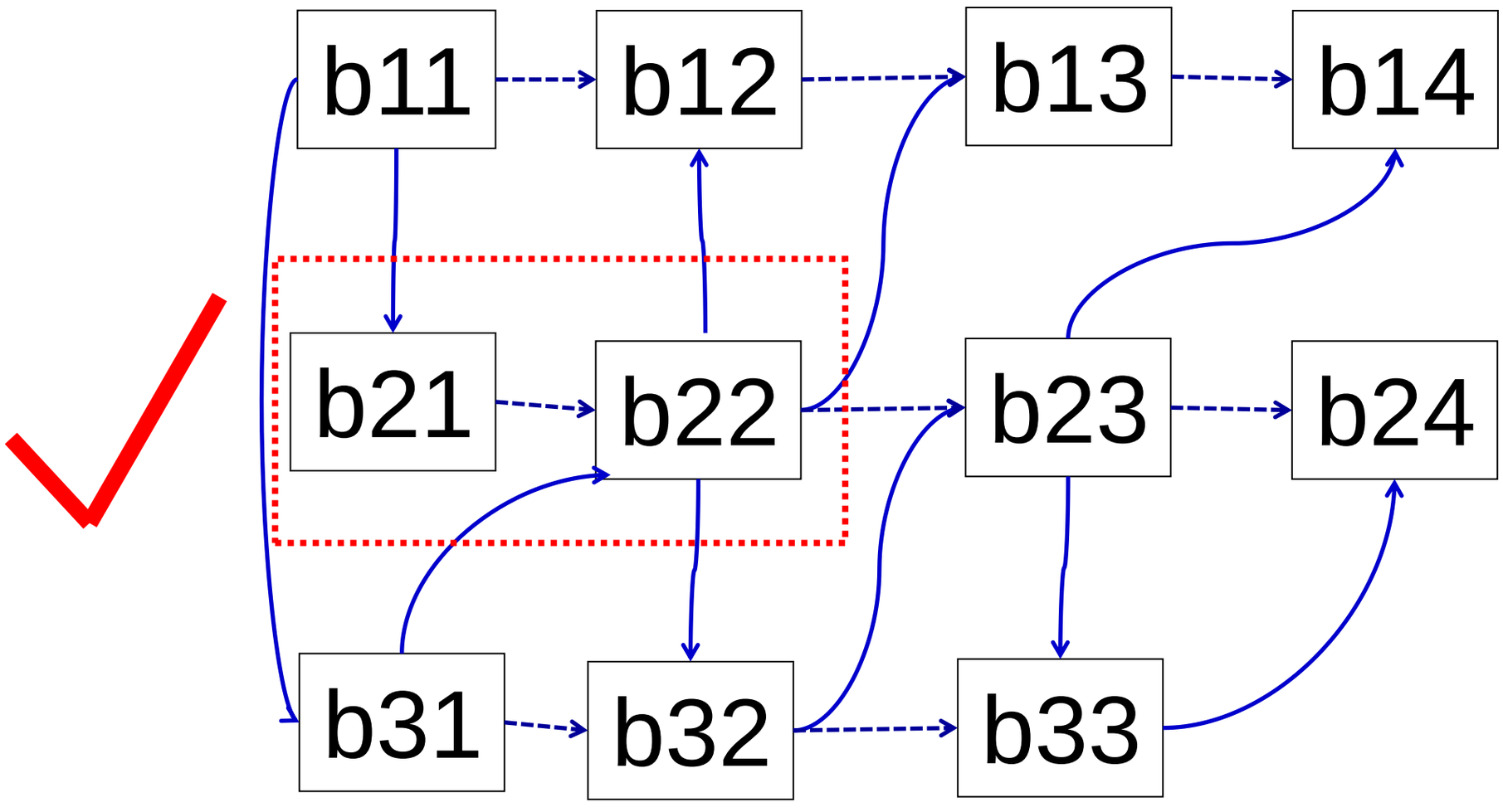
b_{ij}: the j-th node of thread t_i

Example – failed merge



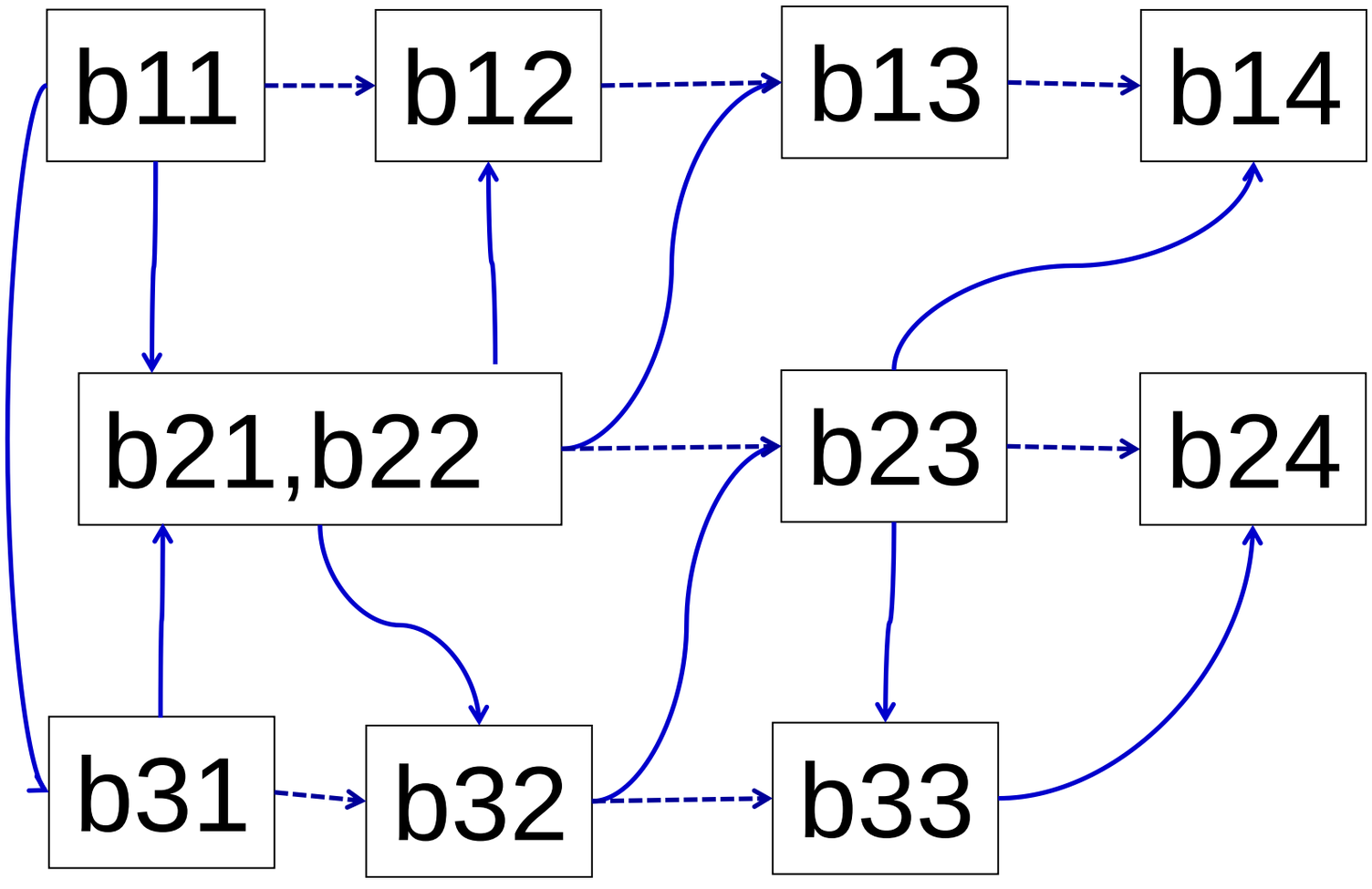
b_{ij}: the j-th node of thread t_i

Example – successful merge



b_{ij}: the j-th node of thread t_i

Example – new graph after merge



b_{ij}: the j-th node of thread t_i

Characteristics of Graph Simplification

- Each successful merge operation reduces one context switch

- The time complexity is O(|V|^2)
  - #dashed edges merge operations O(|V|)
  - Testing reachability can be done in sub-linear time < O(|V|)

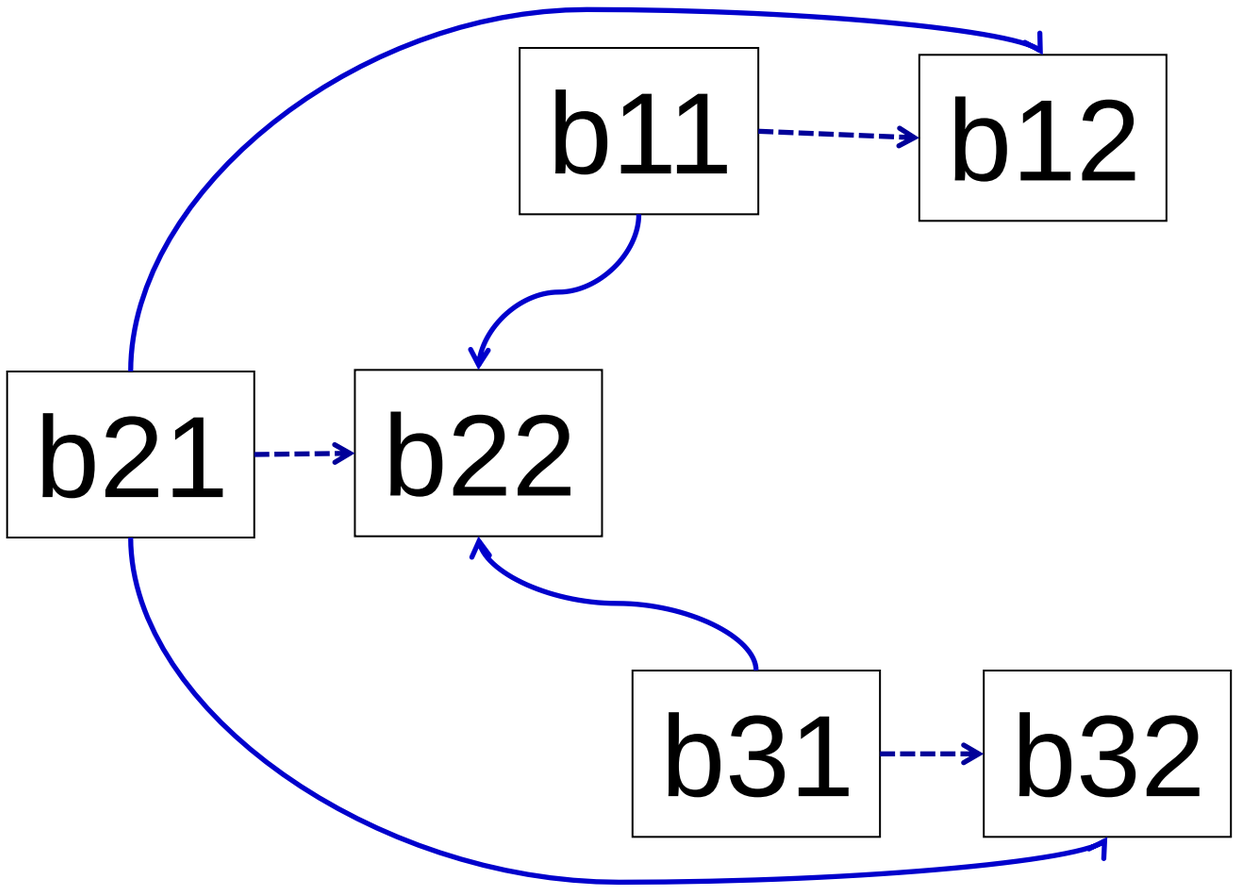
Characteristics of Graph Simplification

- Each successful merge operation reduces one context switch

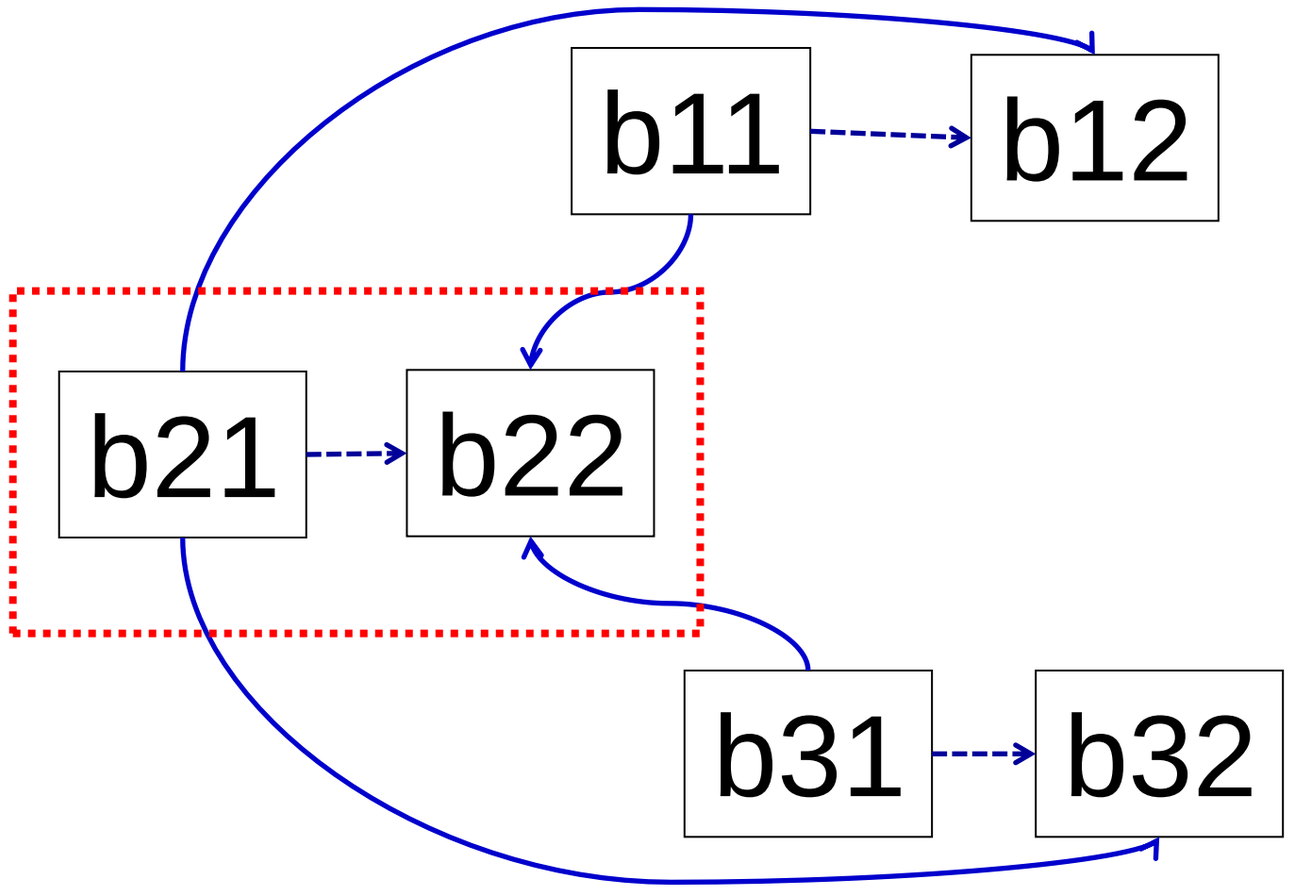
- The time complexity is O(|V|^2)
  - #dashed edges merge operations O(|V|)
  - Testing reachability can be done in sub-linear time < O(|V|)

- **Does not guarantee global optimal simplification**
  - Guarantee *the local optimum* specific to the chosen evaluation order of the dashed edges

Why this problem is hard

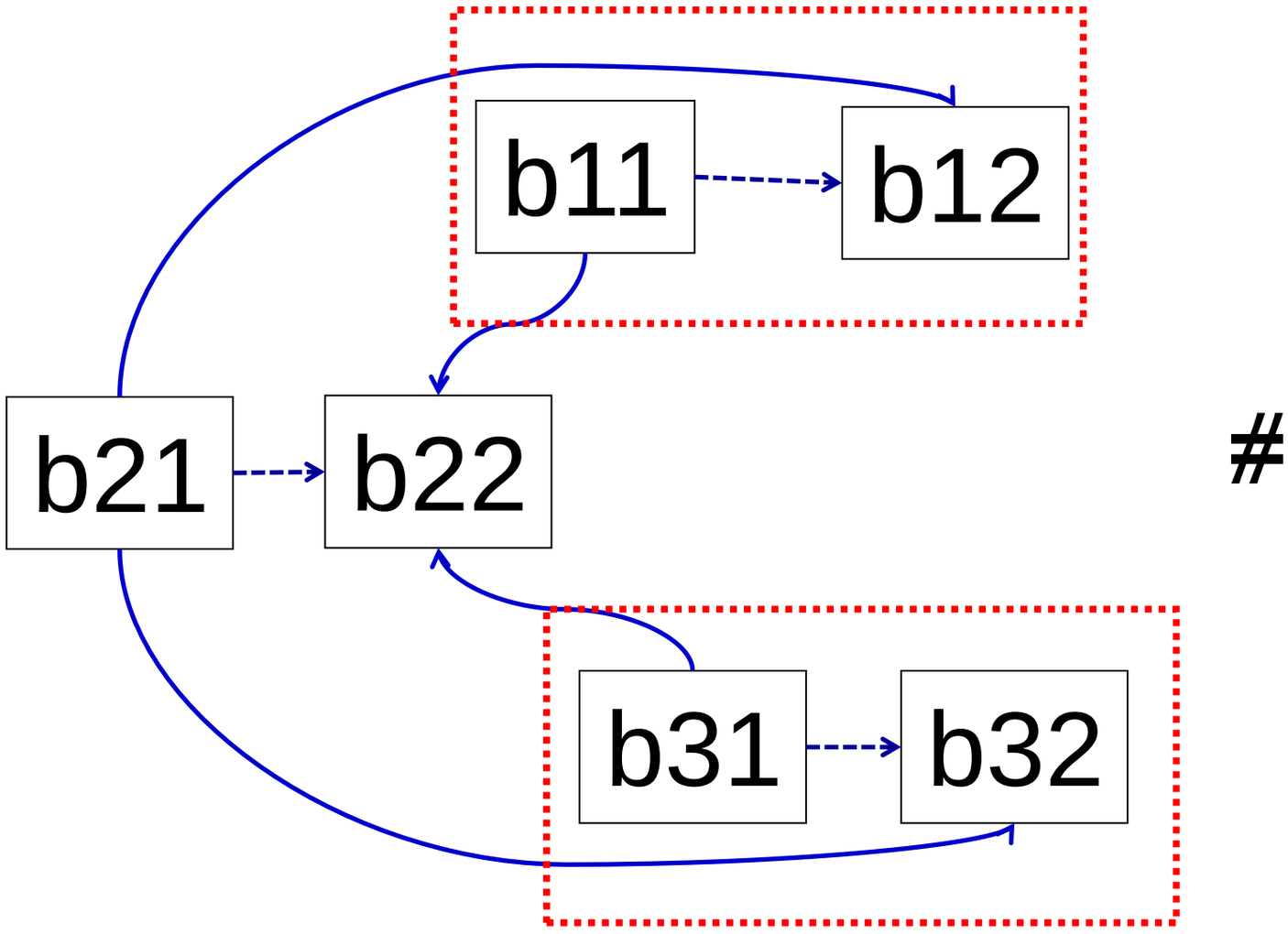


A naïve merge order



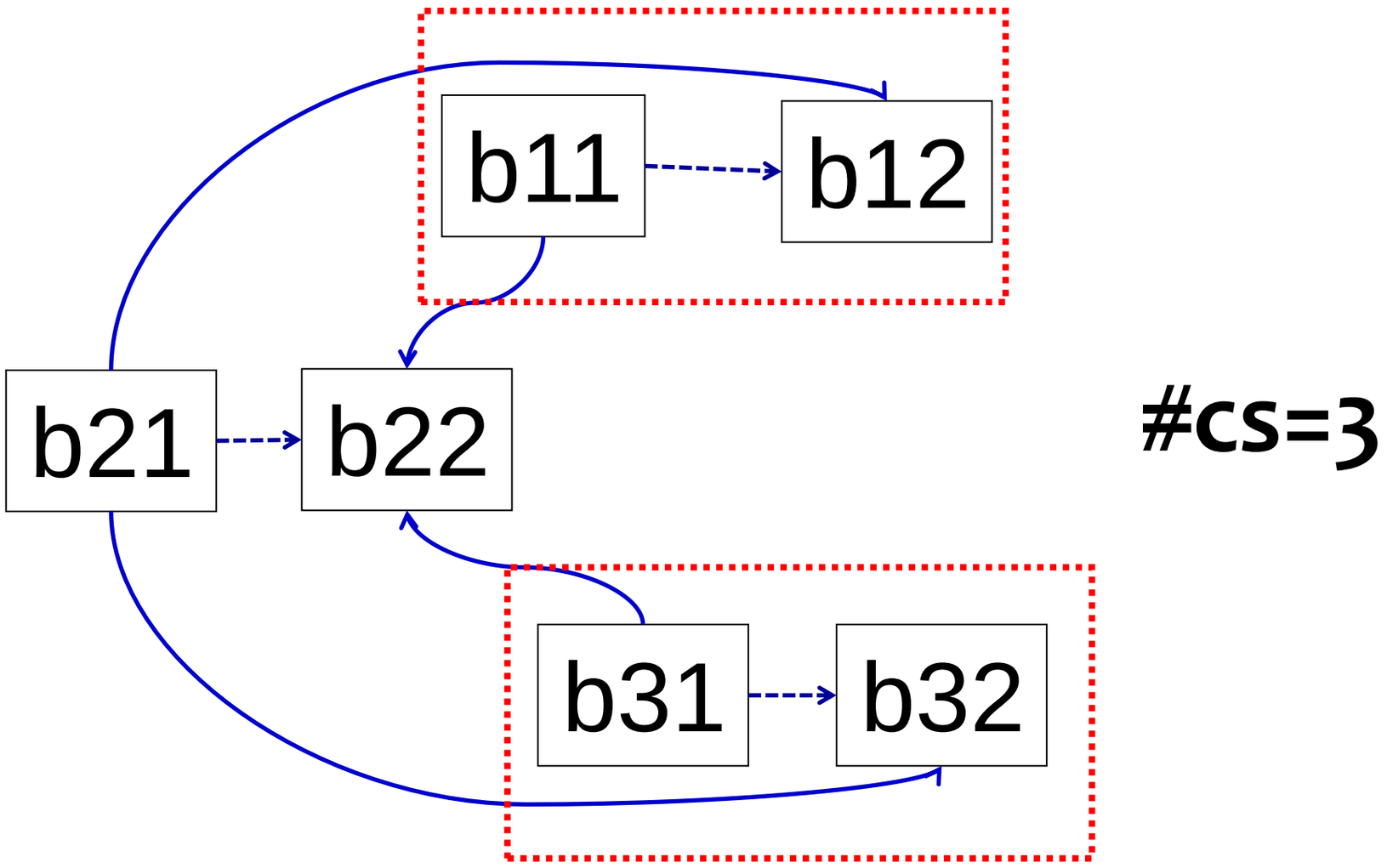
#cs=4

The optimal result



#cs=3

The optimal result



#cs=3

The evaluation order of the dashed edges matters!!

Overall SimTrace algorithm

- Input: trace
- Output: equivalent trace with reduced #CS

- Preprocess: dependence graph construction
- Loop:
  - 1. ***Randomly*** pick a dashed edge and test the reachability
  - 2. Merge and compact graph if not reachable
- A topological sort of the simplified graph

Overall time complexity is **O(n + |V|^2)**

Experiments

- A tool for Java programs
  - http://www.cse.ust.hk/prism/simtrace

- Evaluate trace simplification **effectiveness** & **efficiency**
  - 8 popular multithreaded programs with known bugs
  - ~ 570KLOC
  - 2.53GHz Intel Core 2 Duo processor
  - Windows 7, 4GB memory
  - JDK 1.6.0_10

Results

Program	LOC	Thread	SV	Trace Size	Time	Old Ctxt	New Ctxt	Reduction
Philosopher	81	6	1	**131**	**6.0ms**	51	18.0	**65%**
Bubble	417	26	25	**1,493**	**23.0ms**	454	163.0	**71%**
Elevator	514	4	13	**2,104**	**8.0ms**	80	14.0	**83%**
TSP	709	5	234	**636,499**	**149.0s**	9,272	1,337.0	**86%**
Cache4j	3,897	4	5	**1,225,167**	**592.0s**	417	33.0	**92%**
Weblench	35,175	3	26	**11,630**	**57.0ms**	156	24.0	**85%**
OpenJMS	154,563	32	365	**376,187**	**38.0s**	96,643	11402.0	**88%**
Jigsaw	381,348	10	126	**19,074**	**130.0ms**	2,396	65.0	**97%**

All data are averaged over 50 runs for each subject

Results

Program	LOC	Thread	SV	Trace Size	Time	Old Ctxt	New Ctxt	Reduction
Ph								65%
								71%
E								83%
								86%
C								92%
W								85%
O								88%
Jigsaw	381,348	10	126	**19,074**	**130.0ms**	2,396	65.0	**97%**

Simplification Effectiveness

Simtrace: **65% ~ 97%**

All data are averaged over 50 runs for each subject

Results

Program	LOC	Thread	SV	Trace Size	Time	Old Ctxt	New Ctxt	Reduction
Ph								65%
								71%
E								83%
								86%
C								92%
W								85%
O								88%
Jigsaw	381,348	10	126	**19,074**	**130.0ms**	2,396	65.0	**97%**

Simplification Effectiveness

Simtrace: 65% ~ 97%

Tinertia: 32% ~ 97%

All data are averaged over 50 runs for each subject

Results

Program	LOC	Thread	SV	Trace Size	Time	Old Ctxt	New Ctxt	Reduction
Phi								65%
								71%
E								83%
								86%
C								92%
W								85%
O								88%
								97%

Simplification Efficiency & Scalability

SimTrace: >1M events **<600s**

SimTrace: 1500 events **~ 23ms**

All data are averaged over 50 runs for each subject

Results

Program	LOC	Thread	SV	Trace Size	Time	Old Ctxt	New Ctxt	Reduction
Phi								65%
								71%
E								83%
								86%
C								92%
W								85%
O								88%
								97%

Simplification Efficiency & Scalability

SimTrace: >1M events **<600s**

SimTrace: 1500 events **~ 23ms**

Tinertia: 1500 events **> 750s**

All data are averaged over 50 runs for each subject

Contributions

- **A theorem of trace equivalence**
  - A strict modeling dependence relation

- **A graph model of the context switch simplification problem**

- **An efficient static trace simplification algorithm**

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Jeff Huang, Charles Zhang HKUST

{smhuang, charlesz}@cse.ust.hk