Stateless Model Checking for TSO and PSO

Parosh Aziz Abdulla Stavros Aronis Mohamed Faouzi Atig
Bengt Jonsson **Carl Leonardsson** Konstantinos Sagonas

Uppsala University, Sweden

MM'15 2015-02-24

Goals

Stateless Model Checking

- Find safety errors...
  - in given test case (fixed input program)
  - for all interleavings
  - for all reorderings (TSO/PSO)
- Works on real code in C/pthreads

Valid Test Case (in this presentation)

1. Terminates in bounded time
2. Nondeterminism: Interleavings, Reordering

Goals

Stateless Model Checking

- Find safety errors...
  - in given test case (fixed input program)
  - for all interleavings
  - for all reorderings (TSO/PSO)
- Works on real code in C/pthreads

Valid Test Case (in this presentation)

1. Terminates in bounded time
2. Nondeterminism: Interleavings, Reordering

Partial Order Reduction (SC)

```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

Executions

p:wx1 p:ry0 q:wy1 q:rx1	p:wx1 q:wy1 p:ry1 q:rx1	p:wx1 q:wy1 q:rx1 p:ry1
q:wy1 q:rx0 p:wx1 p:ry1	q:wy1 p:wx1 q:rx1 p:ry1	q:wy1 p:wx1 p:ry1 q:rx1

Partial Order Reduction (SC)

```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

Executions

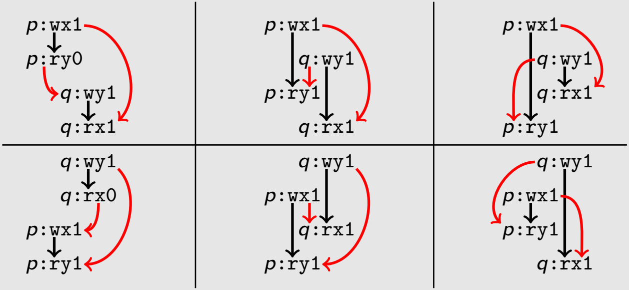
p:wx1	p:wx1	p:wx1
p:ry0	q:wy1	q:wy1
q:wy1	p:ry1	q:rx1
q:rx1	q:rx1	p:ry1
q:wy1	q:wy1	q:wy1
q:rx0	p:wx1	p:wx1
p:wx1	q:rx1	p:ry1
p:ry1	p:ry1	q:rx1

Partial Order Reduction (SC)

```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

Executions (Happens-Before)

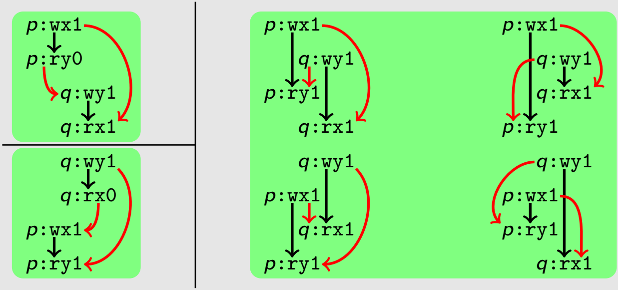


Partial Order Reduction (SC)

```
volatile int x = 0, y = 0;
```

p	*q*
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

Mazurkiewicz Traces ∼ Equivalence Classes over Executions



Stateless Model Checking with DPOR [Flanagan, Godefroid 2005]

Idea

- Explore one execution per Mazurkiewicz trace.
  - $\rightarrow$ Cover all observable behaviours.
- Keep only one execution in memory.
- Examine happens-before relation to find the next trace.

```
volatile int x = 0, y = 0;
```

p	*q*
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

p:`wx1`

p:`ry0`

q:`wy1`

q:`rx1`

`volatile int x = 0, y = 0;`

p	*q*
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

p:`wx1` → p:`ry0`

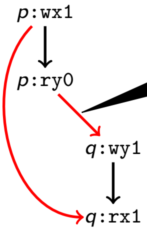
q:`wy1` → q:`rx1`

HB: Necessary Edges

- Program order (under SC)
- Thread spawning to child's first event
- ...

```
volatile int x = 0, y = 0;
```

p	*q*
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

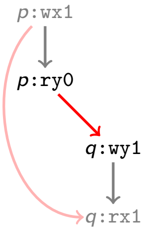


HB: Races

- Conflicting stores and loads.
- ...

`volatile int x = 0, y = 0;`

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

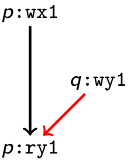


Reverse Races

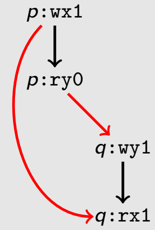
- Start from the end of the execution.

```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

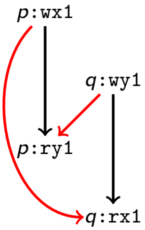


Previous Trace

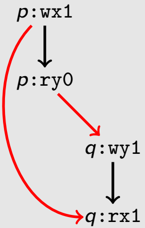


```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

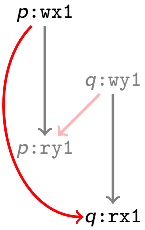


Previous Trace

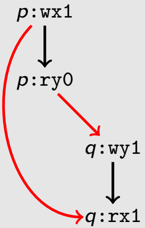


```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

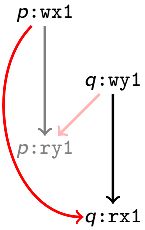


Previous Trace

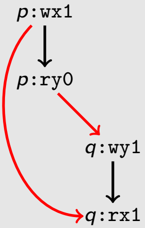


```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

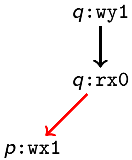


Previous Trace

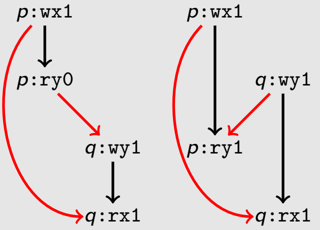


```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

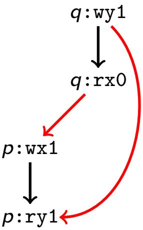


Previous Traces

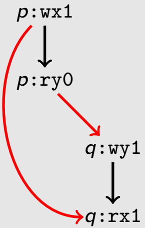


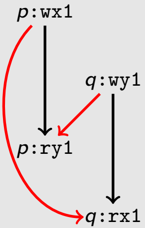
```
volatile int x = 0, y = 0;
```

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`



Previous Traces

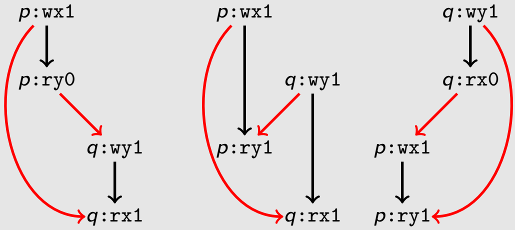




`volatile int x = 0, y = 0;`

p	q
`x = 1;`	`y = 1;`
`int a = y;`	`int b = x;`

All Traces



TSO

- Relaxes $W \to R$
- Store forwarding to own reads (ROWE)
- Operational semantics: store buffer per thread

Defining Traces for TSO

Extend Mazurkiewicz Traces to TSO

- Suitable equivalence classes
- Compatible with DPOR

Defining Traces for TSO

Extend Mazurkiewicz Traces to TSO

- Suitable equivalence classes
- Compatible with DPOR

Defining Traces for TSO

Extend Mazurkiewicz Traces to TSO

- Suitable equivalence classes
- Compatible with DPOR

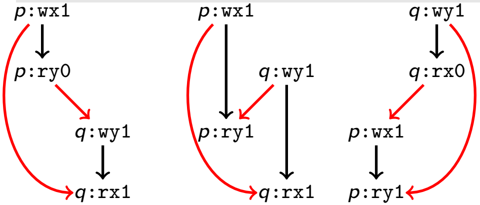
Shasha-Snir Traces

Capture observable order.

Defining Traces for TSO

Extend Mazurkiewicz Traces to TSO

- Suitable equivalence classes
- Compatible with DPOR



Shasha-Snir Traces

Same as Mazurkiewicz traces under SC!

Defining Traces for TSO

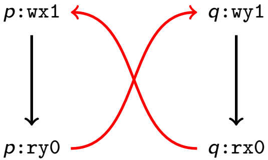
Extend Mazurkiewicz Traces to TSO

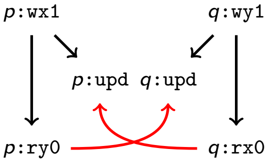
- Suitable equivalence classes
- Compatible with DPOR

$$
\begin{array}{ccc}
p\texttt{:wx1} & & q\texttt{:wy1} \\
\downarrow & & \downarrow \\
p\texttt{:ry0} & & q\texttt{:rx0}
\end{array}
$$

Shasha-Snir Traces

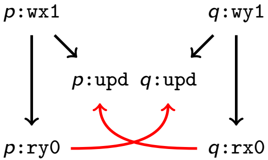
Not partial order.


p:wx1
q:wy1
p:ry0
q:rx0



Getting Rid of Cycles

- Operational semantics
- Ignores *which* events are reordered
- Canonical?

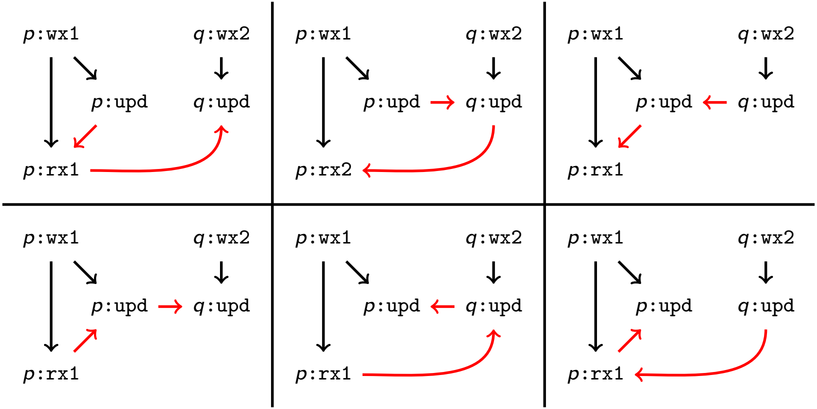


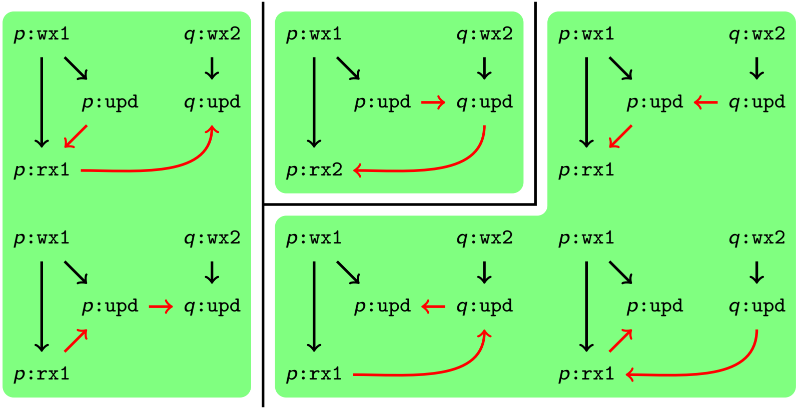
Getting Rid of Cycles

- Operational semantics
- Ignores *which* events are reordered
- Canonical?

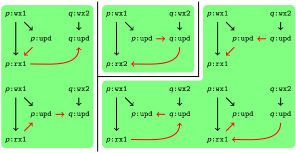
```
volatile int x = 0;
```

p	q
`x = 1;`	`x = 2;`
`int a = x;`	


p:wx1
q:wx2
p:upd
q:upd
p:rx1
p:wx1
q:wx2
p:upd
q:upd
p:rx2
p:wx1
q:wx2
p:upd
q:upd
p:rx1
p:wx1
q:wx2
p:upd
q:upd
p:rx1
p:wx1
q:wx2
p:upd
q:upd
p:rx1
p:wx1
q:wx2
p:upd
q:upd
p:rx1


p:wx1
q:wx2
p:upd
q:upd
p:rx1
p:rx2

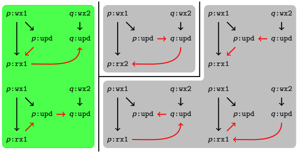
Solution: Chronological Traces



Two Rules

No order between load and update...

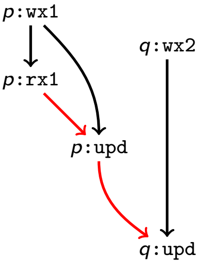
1. of the same thread.
2. when the update is "hidden" from the load.

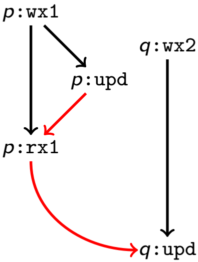


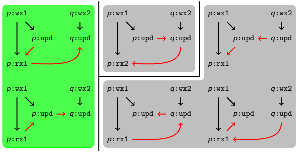
Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.





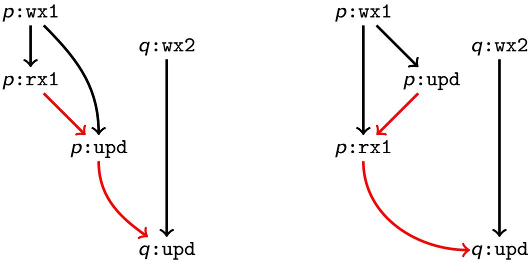


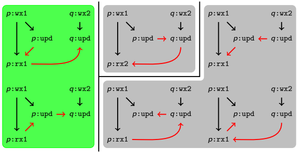
Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.

Solution: Chronological Traces

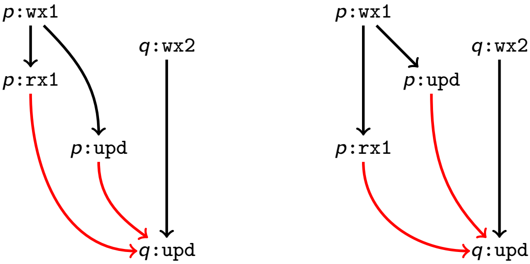


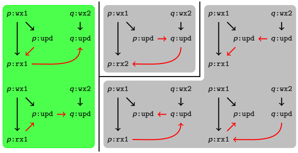


Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.

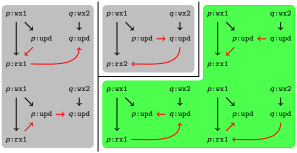




Two Rules

No order between load and update...

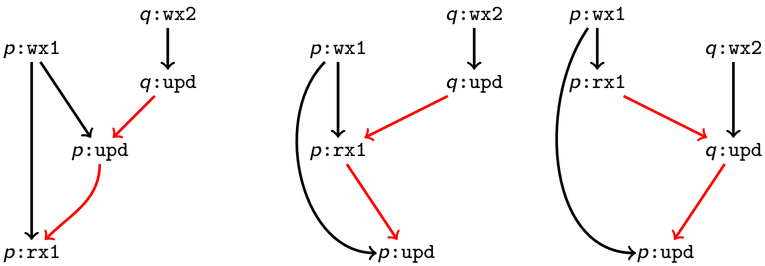
1. of the same thread.
2. when the update is “hidden” from the load.

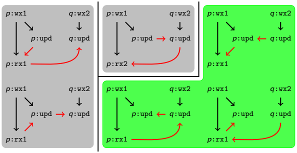


Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.

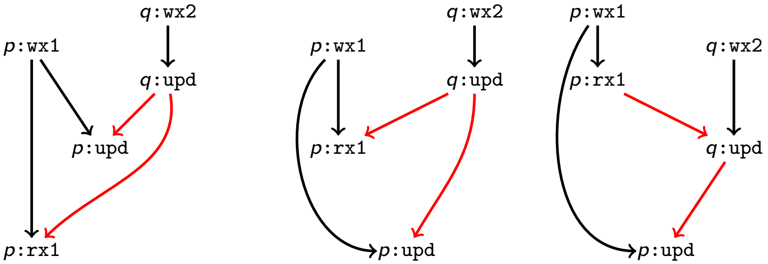


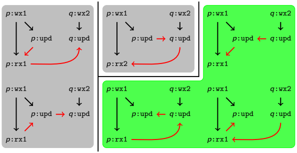


Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.



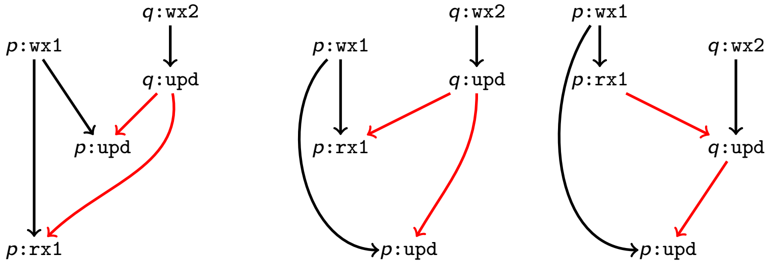


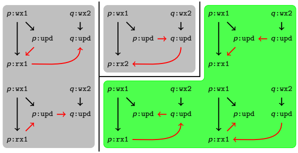
Two Rules

No order between load and update...

1. of the same thread.
2. when the update is “hidden” from the load.

Solution: Chronological Traces

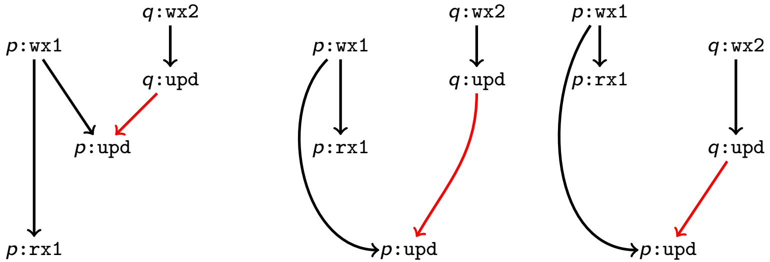


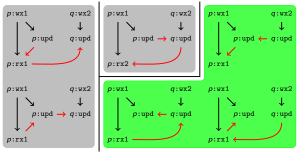


Two Rules

No order between load and update...

1. of the same thread.
2. when the update is "hidden" from the load.



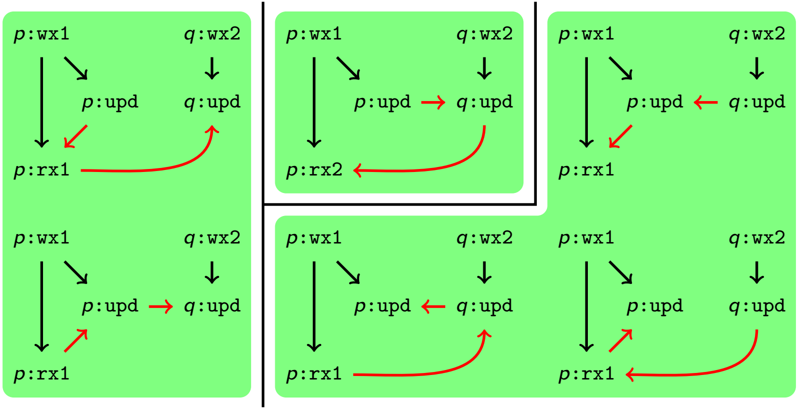


Two Rules

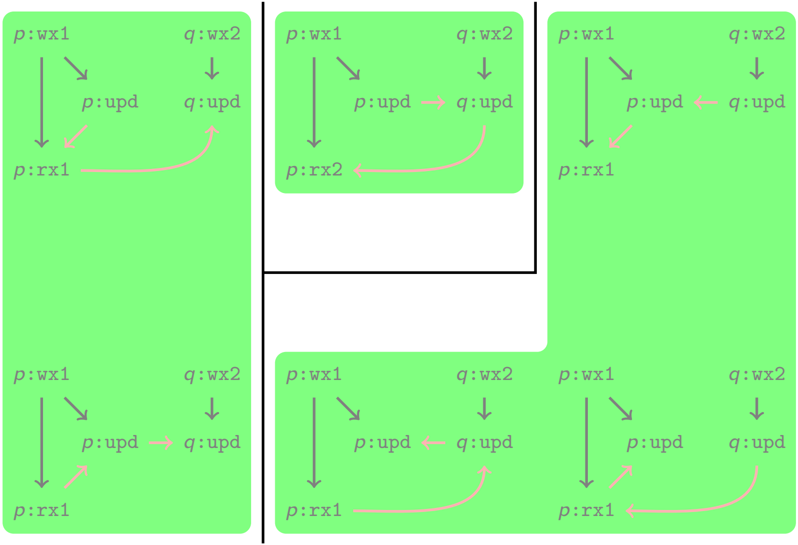
No order between load and update...

1. of the same thread.
2. when the update is “hidden” from the load.

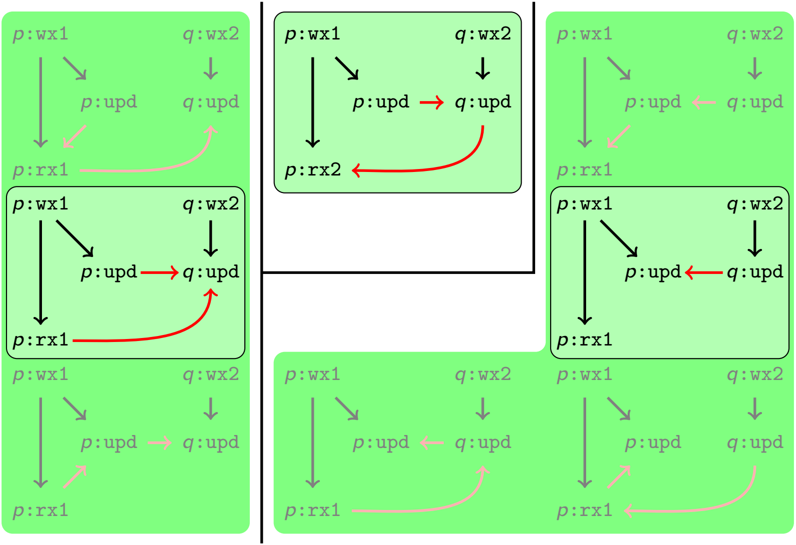
Solution: Chronological Traces



Solution: Chronological Traces



Solution: Chronological Traces



Our Contribution: Chronological Traces

- 1-to-1 with Shasha-Snir traces
- Compatible with DPOR
  - Executions under operational TSO/PSO semantics
  - Efficiently computable happens-before relation

Implementation: Nidhugg

Nidhugg

- Source-DPOR [Abdulla et al. 2014]
  - Near optimal.
  - Straight-forward.
- Works on C/Pthreads via LLVM IR.
- Runs on *one compilation* of test case.

Open source: `https://github.com/nidhugg/nidhugg`

Results: Time Consumption

	fence	LB	CBMC SC	CBMC TSO	CBMC PSO	goto-instrument SC	goto-instrument TSO	goto-instrument PSO	Nidhugg SC	Nidhugg TSO	Nidhugg PSO
apr_1.c	-	5	t/o	t/o	t/o	t/o	!	!	**5.88**	**6.06**	**16.98**
apr_2.c	-	5	t/o	t/o	t/o	!	!	!	**2.60**	**2.20**	**5.39**
dcl_singleton.c	-	7	5.95	31.47	*18.01	5.33	5.36	*0.18	**0.08**	**0.08**	***0.08**
dcl_singleton.c	pso	7	5.88	30.98	29.45	5.20	5.18	5.17	**0.08**	**0.08**	**0.08**
dekker.c	-	10	2.42	*3.17	*2.84	1.68	*4.00	*220.11	**0.10**	***0.11**	***0.09**
dekker.c	tso	10	2.39	5.65	*3.51	1.62	297.62	t/o	**0.11**	**0.12**	***0.08**
dekker.c	pso	10	2.55	5.31	4.83	1.72	428.86	t/o	**0.11**	**0.12**	**0.12**
fib_false.c	-	-	*1.63	*3.38	*3.00	***1.60**	***1.58**	***1.56**	*2.39	*5.57	*6.20
fib_false_join.c	-	-	*0.98	*1.10	*1.91	*1.31	*0.88	*0.80	***0.32**	***0.62**	***0.71**
fib_true.c	-	-	**6.28**	9.39	7.72	6.32	**7.63**	**7.62**	25.83	75.06	86.32
fib_true_join.c	-	-	6.61	8.37	10.81	7.09	5.94	5.92	**1.20**	**2.88**	**3.19**
indexer.c	-	5	193.01	210.42	214.03	191.88	70.42	69.38	**0.10**	**0.09**	**0.09**
lamport.c	-	8	7.78	*11.63	*10.53	6.89	t/o	t/o	**0.08**	***0.08**	***0.08**
lamport.c	tso	8	7.60	26.31	*15.85	6.80	513.67	t/o	**0.09**	**0.08**	***0.07**
lamport.c	pso	8	7.72	30.92	27.51	7.43	t/o	t/o	**0.08**	**0.08**	**0.08**
parker.c	-	10	12.34	*91.99	*86.10	11.63	~~9.70~~	~~9.65~~	**1.50**	***0.09**	***0.08**
parker.c	pso	10	12.72	141.24	166.75	11.76	10.66	10.64	**1.50**	**1.92**	**2.94**
peterson.c	-	-	0.35	*0.38	*0.35	0.18	*0.20	*0.21	**0.07**	***0.07**	***0.07**
peterson.c	tso	-	0.35	0.39	*0.35	0.19	0.18	~~0.56~~	**0.07**	**0.07**	***0.07**
peterson.c	pso	-	0.35	0.41	0.40	0.18	0.18	0.19	**0.07**	**0.07**	**0.08**
pgsql.c	-	8	19.80	60.66	*4.63	21.03	46.57	*296.77	**0.08**	**0.07**	***0.08**
pgsql.c	pso	8	23.93	71.15	121.51	19.04	t/o	t/o	**0.07**	**0.07**	**0.08**
pgsql_bnd.c	pso	(4)	**3.57**	**9.55**	**12.68**	3.59	t/o	t/o	89.44	106.04	112.60
stack_safe.c	-	-	44.53	516.01	496.36	45.11	42.39	42.50	**0.34**	**0.36**	**0.43**
stack_unsafe.c	-	-	*1.40	*1.87	*2.08	*1.00	*0.81	*0.79	***0.08**	***0.08**	***0.09**
szymanski.c	-	-	0.40	*0.44	*0.43	0.23	*0.89	*1.16	**0.07**	***0.13**	***0.07**
szymanski.c	tso	-	0.40	0.50	*0.43	0.23	0.23	~~2.48~~	**0.08**	**0.08**	***0.07**
szymanski.c	pso	-	0.39	0.50	0.49	0.23	0.24	0.24	**0.08**	**0.08**	**0.08**

Results: Time Consumption											
			CBMC			goto-instrument			Nidhugg		
	fence	LB	SC	TSO	PSO	SC	TSO	PSO	SC	TSO	PSO
apr_1.c	-	5	t/o	t/o	t/o	t/o	!	!	**5.88**	**6.06**	**16.98**
apr_2.c	-	5	t/o	t/o	t/o	!	!	!	**2.60**	**2.20**	**5.39**
dcl_singleton.c	-	7	5.95	31.47	*18.01	5.33	5.36	*0.18	**0.08**	**0.08**	***0.08**
dcl_singleton.c	pso	7	5.88	30.98	29.45	5.20	5.18	5.17	**0.08**	**0.08**	**0.08**
dekker.c	-	10	2.42	*3.17	*2.84	1.68	*4.00	*220.11	**0.10**	***0.11**	***0.09**
dekker.c	tso	10	2.39	5.65	*3.51	1.62	297.62	t/o	**0.11**	**0.12**	***0.08**
dekker.c	pso	10	2.55	5.31	4.83	1.72	428.86	t/o	**0.11**	**0.12**	**0.12**
fib_false.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	*6.20
fib_false_join.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	***0.71**
fib_true.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	86.32
fib_true_join.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**3.19**
indexer.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**0.09**
lamport.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	***0.08**
lamport.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	***0.07**
lamport.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**0.08**
parker.c	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	***0.08**
parker.c	pso	10	12.72	141.24	166.75	11.76	10.66	10.64	**1.50**	**1.92**	**2.94**
peterson.c	-	-	0.35	*0.38	*0.35	0.18	*0.20	*0.21	**0.07**	***0.07**	***0.07**
peterson.c	tso	-	0.35	0.39	*0.35	0.19	0.18	~~0.56~~	**0.07**	**0.07**	***0.07**
peterson.c	pso	-	0.35	0.41	0.40	0.18	0.18	0.19	**0.07**	**0.07**	**0.08**
pgsql.c	-	8	19.80	60.66	*4.63	21.03	46.57	*296.77	**0.08**	**0.07**	***0.08**
pgsql.c	pso	8	23.93	71.15	121.51	19.04	t/o	t/o	**0.07**	**0.07**	**0.08**
pgsql_bnd.c	pso	(4)	**3.57**	**9.55**	**12.68**	3.59	t/o	t/o	89.44	106.04	112.60
stack_safe.c	-	-	44.53	516.01	496.36	45.11	42.39	42.50	**0.34**	**0.36**	**0.43**
stack_unsafe.c	-	-	*1.40	*1.87	*2.08	*1.00	*0.81	*0.79	***0.08**	***0.08**	***0.09**
szymanski.c	-	-	0.40	*0.44	*0.43	0.23	*0.89	*1.16	**0.07**	***0.13**	***0.07**
szymanski.c	tso	-	0.40	0.50	*0.43	0.23	0.23	~~2.48~~	**0.08**	**0.08**	***0.07**
szymanski.c	pso	-	0.39	0.50	0.49	0.23	0.24	0.24	**0.08**	**0.08**	**0.08**

Robust

	Nidhugg		
	SC	TSO	PSO
stack_safe.c	**0.34**	**0.36**	**0.43**

Future Work

Ongoing Work: POWER

- More relaxed model
- Order enforced by complex event interaction
- #traces probably similar to under SC
- Techniques carry over to ARM, Alpha, etc.