Hardware Synthesis of Weakly Consistent C Concurrency

John Wickerson
Imperial College London

a talk based on joint work with

Nadesh Ramanathan
Imperial College London

Shane T. Fleming
Imperial College London

George A. Constantinides
Imperial College London

S-REPLS @ Oxford

12 Jan 2017

Starting point

Concurrent languages with built-in support for strong/weak atomics

THE C++ PROGRAMMING LANGUAGE

OpenCL

✔

✔

?

CPUs

intel

ARM

GPUs

AMD

NVIDIA

FPGAs

XILINX

ALTERA

FPGAs

- Reconfigurable hardware

FPGAs

TECH

Intel Completes Acquisition of Altera

$16.7 billion deal underscores Intel CEO's plan to expand chip maker's business

By DON CLARK

Updated Dec. 28, 2015 9:27 p.m. ET

Intel Corp. on Monday completed its biggest-ever acquisition, part of Chief Executive Brian Krzanich's plan to use new tactics to expand the chip maker's business.

The $16.7 billion purchase of Altera Corp. makes Intel, known for microprocessors used in computers, the second-largest maker of

Xilinx Virtex 5 FPGA in Driverless Cars

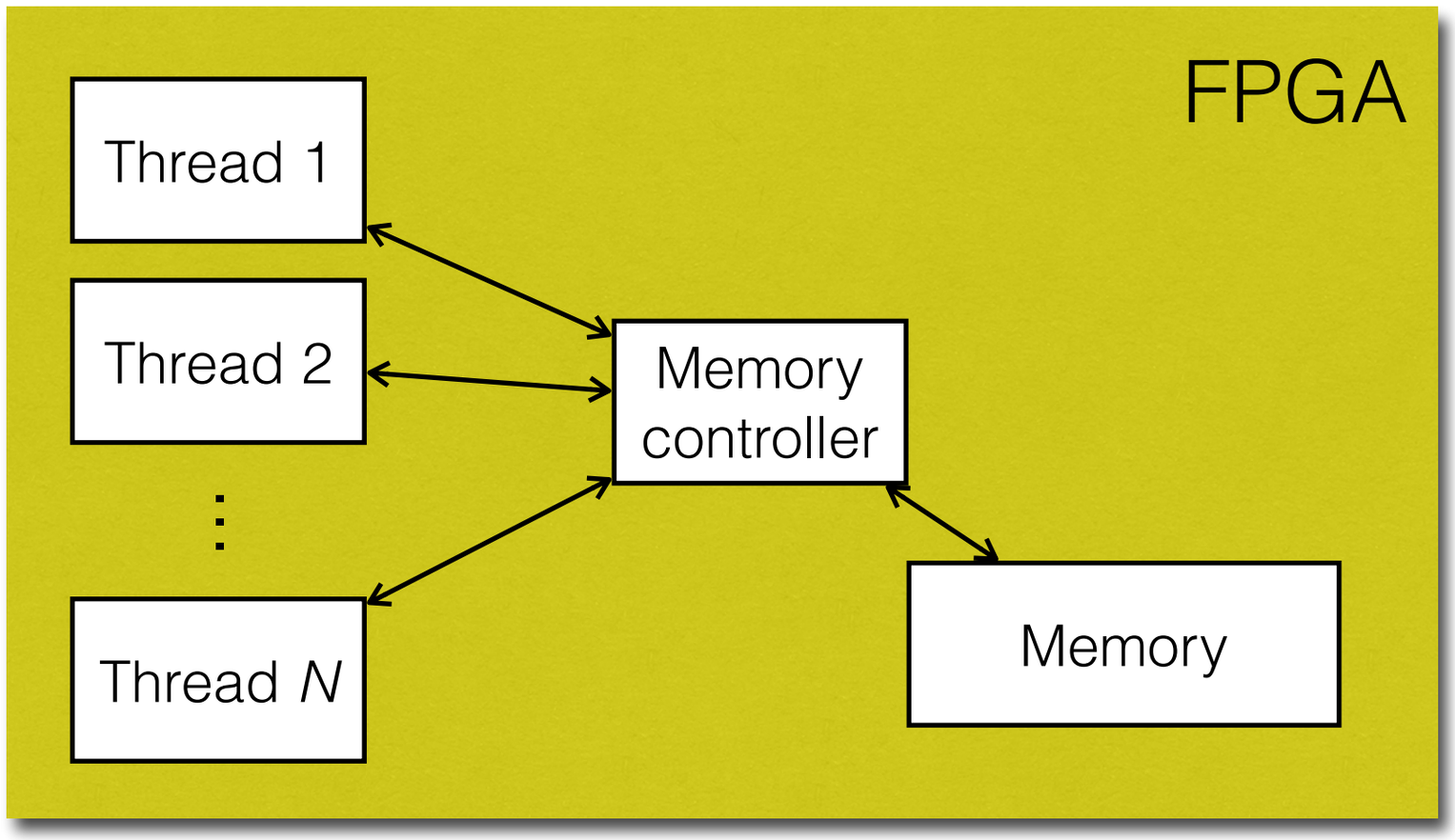
Posted by Nadeesha Thewarapperuma

Find me on:

Aug 4, 2015 11:00:00 AM

LegUp

- Open-source hardware synthesis tool developed at the University of Toronto since 2009
- Supports pthreads, and OpenMP (using locks)



Synthesis example

- Can we implement atomic stores/loads using just ordinary stores/loads?

```
int x=0; int y=0;
```

T1() {	T2() {
1.1 x=1;	2.1 if(y==1)
1.2 y=1;	2.2 r0=x;
}	}

assert(r0 ≠ 0)

Synthesis example

- Can we implement atomic stores/loads using just ordinary stores/loads?

```
int x=0; int y=0;
```

T1(int a) {	T2() {
1.1 x=a/3;	2.1 if(y==1)
1.2 y=1;	2.2 r0=x;
}	}

assert(r0 $\neq 0$)

```
                    int x=0; int y=0;
   T1(int a) {                          ||    T2() {
1.1 x=a/3;                              || 2.1 if(y==1)
1.2 y=1;                                || 2.2  r0=x;
   }                                    ||   }
                    assert(r0 ≠ 0)
```

Cycle:	1	2	3	4	5	···	35	36
1.1	ld a							
1.1			divide					
1.1								st x
1.2	st y							
2.1			ld y					
2.2			ld x					
2.2					slt y==1? x:null			

Implementing atomics

- **Unsound:** only respects RW/WR/WW dependencies

`r0=w;`
`r1=x;`
`r2=y.ld(ACQ);`
`r3=z;`

Implementing atomics

- **Unsound:** only respects RW/WR/WW dependencies

Cycle:	1	2
`r0=w;`	$\mathrm{ld}_{\mathtt{na}}$ w	
`r1=x;`	$\mathrm{ld}_{\mathtt{na}}$ x	
`r2=y.ld(ACQ);`	$\mathrm{ld}_{\mathtt{ACQ}}$ y	
`r3=z;`	$\mathrm{ld}_{\mathtt{na}}$ z	

Implementing atomics

- **SC:** all memory accesses strictly ordered

Cycle:	1	2	3	4	5	6	7	8
`r0=w;`	$\mathrm{ld}_{\mathtt{na}}$ `w`							
`r1=x;`			$\mathrm{ld}_{\mathtt{na}}$ `x`					
`r2=y.ld(ACQ);`					$\mathrm{ld}_{\mathtt{ACQ}}$ `y`			
`r3=z;`							$\mathrm{ld}_{\mathtt{na}}$ `z`	

Implementing atomics

- **SC-atomics:** all atomics are strictly ordered

Cycle:	1	2	3	4	5	6
`r0=w;`	$\mathrm{ld}_{\mathtt{na}}$ `w`					
`r1=x;`	$\mathrm{ld}_{\mathtt{na}}$ `x`					
`r2=y.ld(ACQ);`			$\mathrm{ld}_{\mathtt{ACQ}}$ `y`			
`r3=z;`					$\mathrm{ld}_{\mathtt{na}}$ `z`	

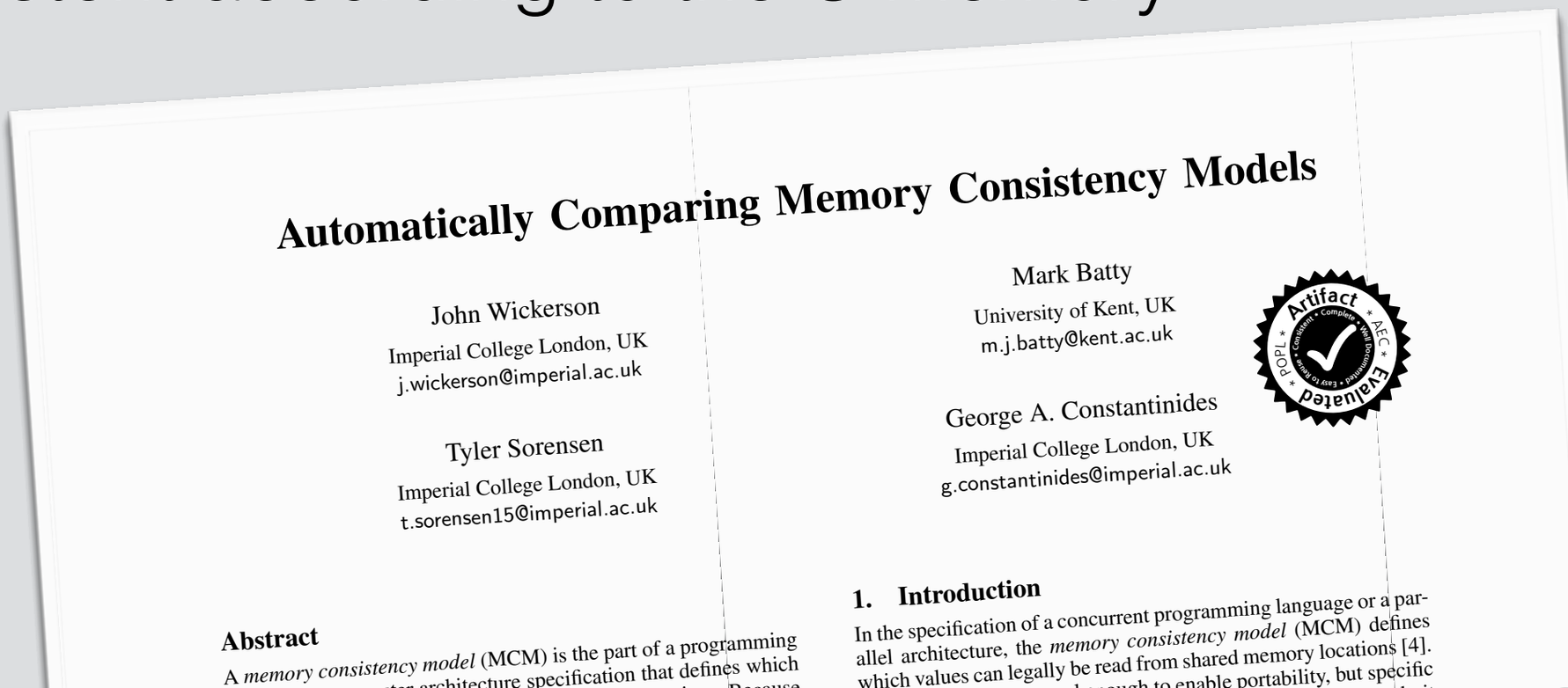
Implementing atomics

- **Weak-atomics:** acquires cannot move down; releases cannot move up; SCs cannot move at all

Cycle:	1	2	3	4
`r0=w;`	$\mathrm{ld}_{\mathtt{na}}$ `w`			
`r1=x;`	$\mathrm{ld}_{\mathtt{na}}$ `x`			
`r2=y.ld(ACQ);`	$\mathrm{ld}_{\mathtt{ACQ}}$ `y`			
`r3=z;`			$\mathrm{ld}_{\mathtt{na}}$ `z`	

Correctness

- We used the Alloy Analyzer to check that there is no execution (with ≤9 memory events) that:
  - is allowed by our scheduling constraints, but
  - is inconsistent according to the C memory model.



Automatically Comparing Memory Consistency Models

John Wickerson
Imperial College London, UK
j.wickerson@imperial.ac.uk

Tyler Sorensen
Imperial College London, UK
t.sorensen15@imperial.ac.uk

Mark Batty
University of Kent, UK
m.j.batty@kent.ac.uk

George A. Constantinides
Imperial College London, UK
g.constantinides@imperial.ac.uk

Abstract

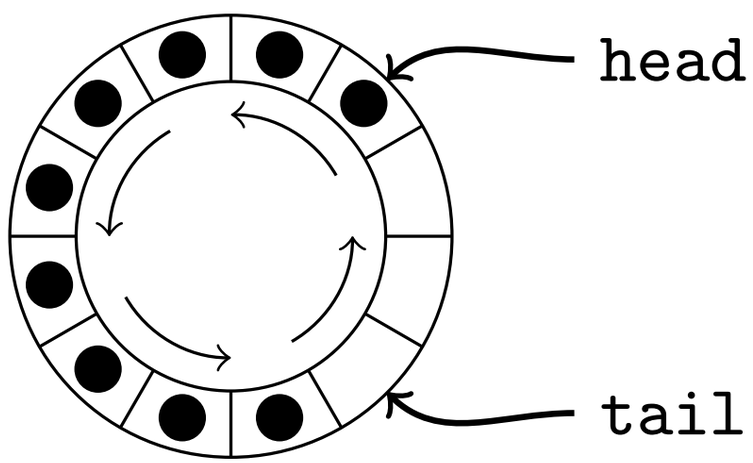
A *memory consistency model* (MCM) is the part of a programming

1. Introduction

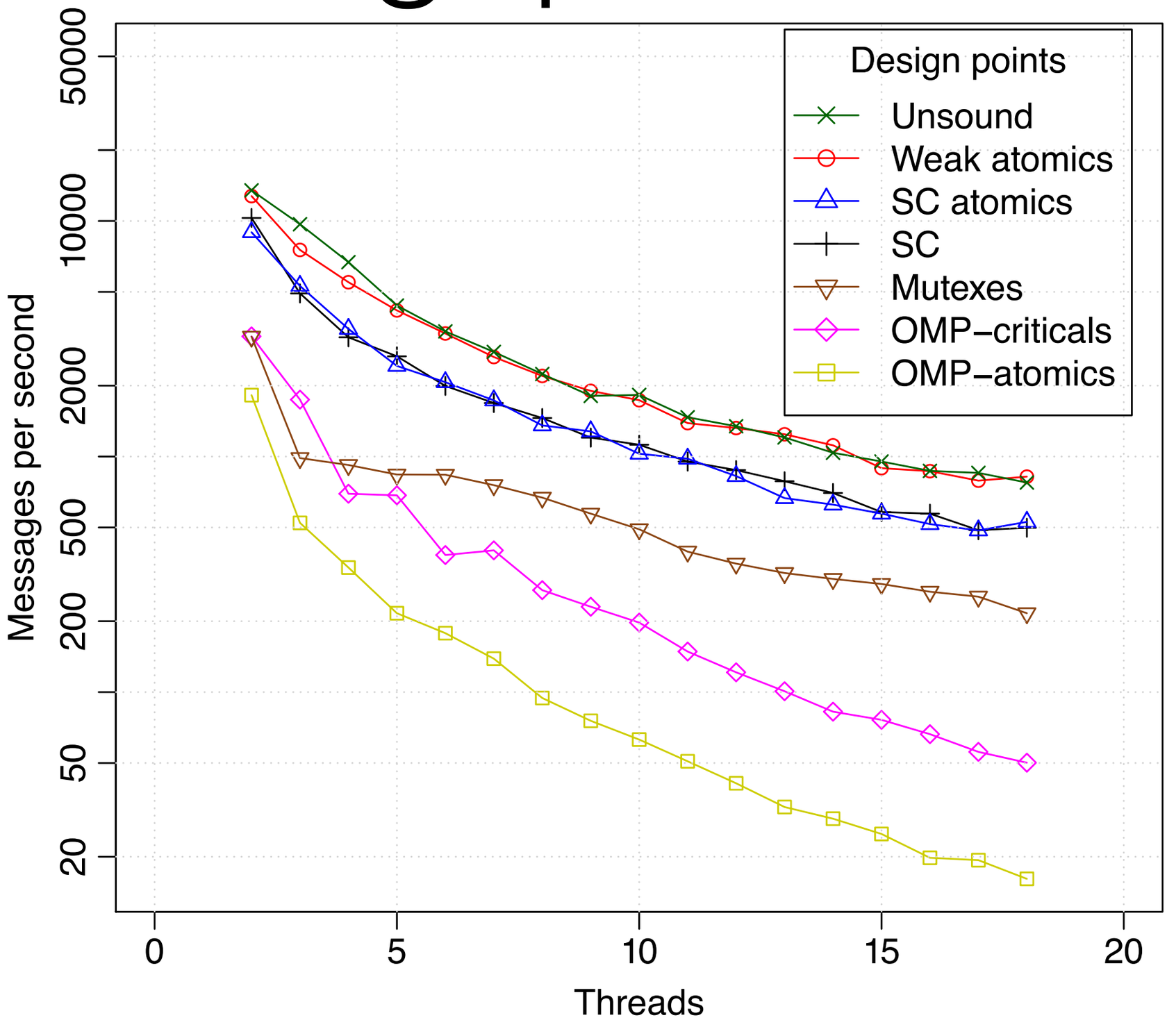
In the specification of a concurrent programming language or a parallel architecture, the *memory consistency model* (MCM) defines which values can legally be read from shared memory locations [4].

Case study: circular FIFO

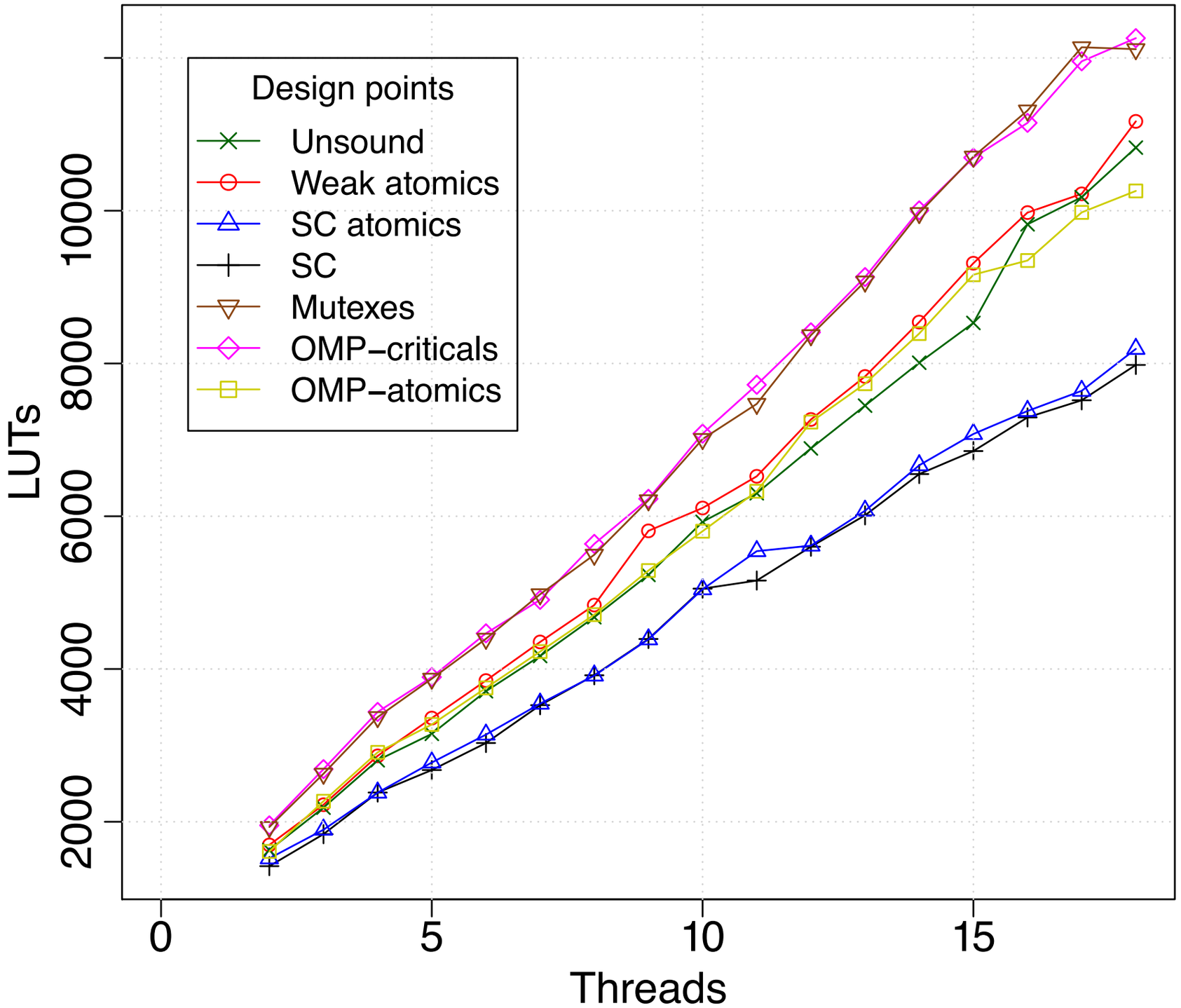
```
                    atomic_int tail=0; head=0;
                     int arr[SIZE]; res[MSGS];
1.1  while(prod<MSGS) {                    ||  while(cons<MSGS) {                    2.1
1.2   chead = head.ld(ACQ);                ||   ctail = tail.ld(ACQ);                2.2
1.3   ctail = tail.ld(RLX);                ||   chead = head.ld(RLX);                2.3
1.4   ntail = (ctail+1)%SIZE;              ||   nhead = (chead+1)%SIZE;              2.4
1.5   if(ntail != chead){                  ||   if(ctail != chead){                  2.5
1.6    arr[ctail] = prod                   ||    res[cons] = arr[chead];             2.6
1.7    tail.st(ntail,REL);                 ||    head.st(nhead,REL);                 2.7
1.8    prod++;                             ||    cons++;                             2.8
1.9   }                                    ||   }                                    2.9
1.10 }                                     ||  }                                    2.10
```



Throughput results



Circuit area results



Conclusion

- First implementation of weak atomics in a hardware synthesis tool
- Implementing atomics using scheduling constraints seems more efficient than using locks
- **Limitations:** no support for RMW operations; small and artificial benchmarks; only on-chip memory
- **Next steps:** add support for loop pipelining

Hardware Synthesis of Weakly Consistent C Concurrency

John Wickerson
Imperial College London

a talk based on joint work with

Nadesh Ramanathan
Imperial College London

Shane T. Fleming
Imperial College London

George A. Constantinides
Imperial College London

S-REPLS @ Oxford

12 Jan 2017