

# Big Picture?

## Transactional Memory: Architectural Support for Lock-Free Data Structures

Maurice Herlihy  
Digital Equipment Corporation  
Cambridge Research Laboratory  
Cambridge MA 02139  
herlihy@crl.dec.com

J. Eliot B. Moss  
Dept. of Computer Science  
University of Massachusetts  
Amherst, MA 01003  
moss@cs.umass.edu

*CSE 240B*

*Dean Tullsen*

*CSE 240B*

*Dean Tullsen*

## Terms

- Lock-free
- Priority inversion
- Convoying
- Serializability
- Atomicity

*CSE 240B*

*Dean Tullsen*

## Primitives

- Ltx
- ST
- Validate
- Commit

*CSE 240B*

*Dean Tullsen*

## How is it implemented?

## What would you do if only a single cache?

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## Coherence

## Let's try

Name	Access	Shared?	Modified?
INVALID	none	—	—
VALID	R	Yes	No
DIRTY	R, W	No	Yes
RESERVED	R, W	No	No

Table 1: Cache line states

Name	Kind	Meaning	New access
READ	regular	read value	shared
RFO	regular	read value	exclusive
WRITE	both	write back	exclusive
T.READ	trans	read value	shared
T.RFO	trans	read value	exclusive
BUSY	trans	refuse access	unchanged

Table 3: Bus cycles

Name	Meaning
EMPTY	contains no data
NORMAL	contains committed data
XCOMMIT	discard on commit
XABORT	discard on abort

Table 2: Transactional tags

try, it first searches for an EMPTY entry, then for a NORMAL entry, and finally for an XCOMMIT entry. If the XCOMMIT entry is DIRTY, it must be written back. Notice that XCOMMIT entries are used only to enhance performance. When a ST tentatively updates an entry, the old value must be retained in case the transaction aborts. If the old value is

Lw A

LT A

LT A

ST A

commit

LT A

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

# Code

```

typedef struct {
    Word deqs;
    Word enqs;
    Word items[QUEUE_SIZE];
} queue;

unsigned queue_deq(queue *q) {
    unsigned head, tail, result;
    unsigned backoff = BACKOFF_MIN;
    unsigned wait;
    while (1) {
        result = QUEUE_EMPTY;
        tail = LTX(&q->enqs);
        head = LTX(&q->deqs);
        /* queue not empty? */
        if (head != tail) {
            result =
                LT(&q->items[head % QUEUE_SIZE]);
            /* advance counter */
            ST(&q->deqs, head + 1);
        }
        if (COMMIT()) break;
        /* abort => backoff */
        wait = random() % (01 << backoff);
        while (wait--);
        if (backoff < BACKOFF_MAX)
            backoff++;
    }
    return result;
}
    
```

Figure 2: Part of Producer/Consumer Benchmark

```

void list_enq(entry* new) {
    entry *old_tail;
    unsigned backoff = BACKOFF_MIN;
    unsigned wait;

    new->next = new->prev = NULL;

    while (TRUE) {
        old_tail = (entry*) LTX(&Tail);
        if (VALIDATE()) {
            ST(&new->prev, old_tail);
            if (old_tail == NULL) {
                ST(&Head, new);
            } else {
                ST(&old_tail->next, new);
            }
            ST(&Tail, new);
            if (COMMIT()) return;
            wait = random() % (01 << backoff);
            while (wait--);
            if (backoff < BACKOFF_MAX)
                backoff++;
        }
    }
}
    
```

Figure 3: Part of Doubly-Linked List Benchmark

Dean Tullsen

CSE 240B

# Results

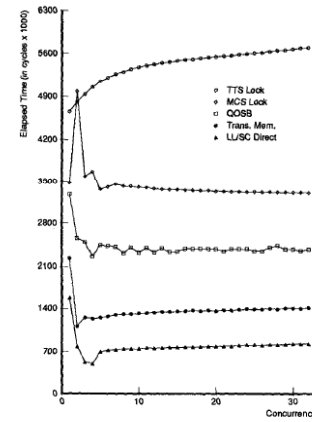


Figure 4: Counting Benchmark: Bus and Network

CSE 240B

Dean Tullsen

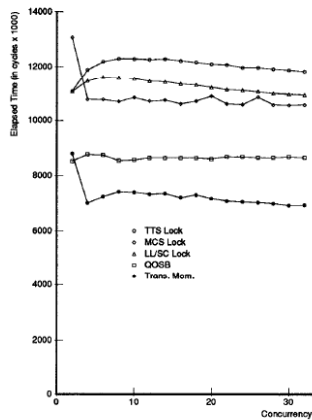
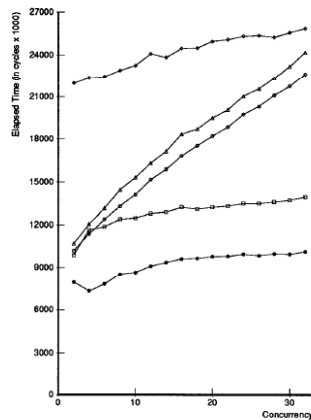


Figure 5: Producer/Consumer Benchmark: Bus and Network



Dean Tullsen

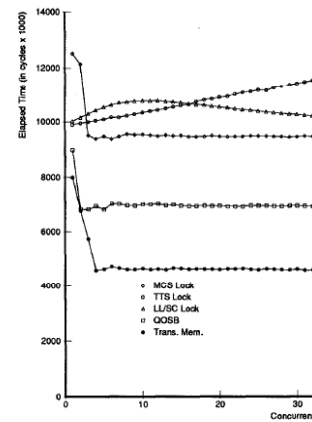


Figure 6: Doubly-Linked List Benchmark: Bus and Network

CSE 240B

Dean Tullsen

# Big Picture?

## Transactional Memory Coherence and Consistency

Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg,  
Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun  
Stanford University  
{lance, viewong, broccoli, bdc, johnd, elektrik, mkprabhu, wijayah}@stanford.edu,  
christos@ee.stanford.edu, kunle@stanford.edu

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## How different than herlihy TM?

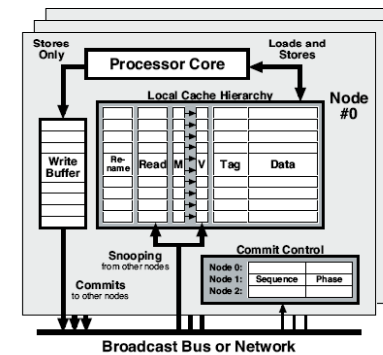


Figure 1: A sample 3-node TCC system.

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

# Ordering

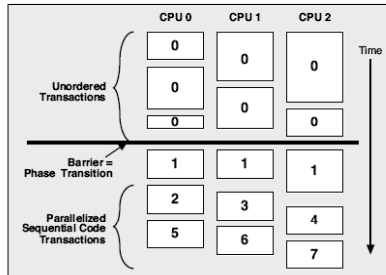


Figure 2: Timing illustration of how transactions (numbered blocks) running on three different processors are forced to commit by phase number sequence.

# Let's try

Lw A  
 LT A  
  
 LT A  
 ST A  
 commit  
  
 LT A

# Double Buffering

- Why?

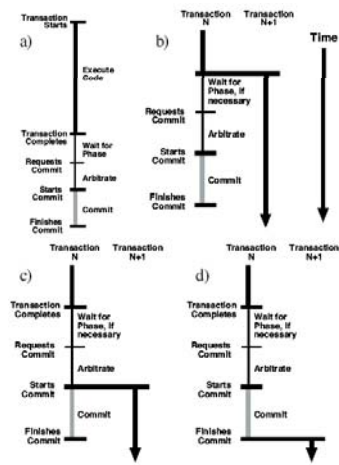


Figure 3: The effect of double buffering: a) a sample transaction timeline, b) double buffering of all speculative state, c) double buffering for write buffer but not read bits in cache, and d) pure single buffering.

# Results

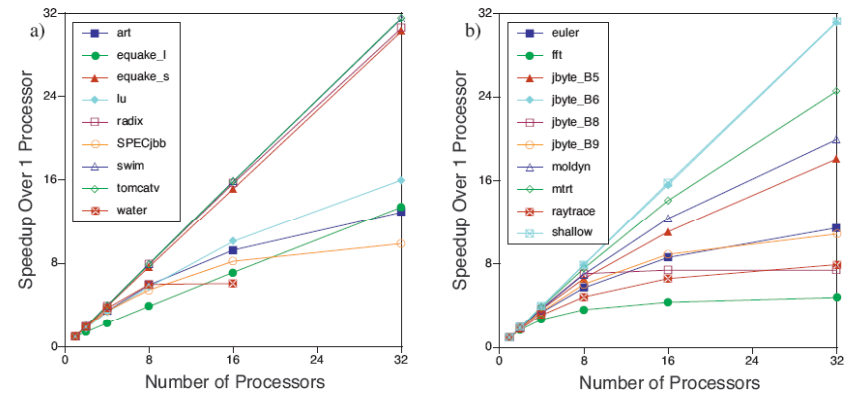


Figure 4: Speedups for varying numbers of processors with our manually parallelized benchmarks (a) and Java benchmarks with automated parallelism (b) on a perfect TCC system with 1 IPC processors, no memory delays, and  $\infty$  commit bandwidth.

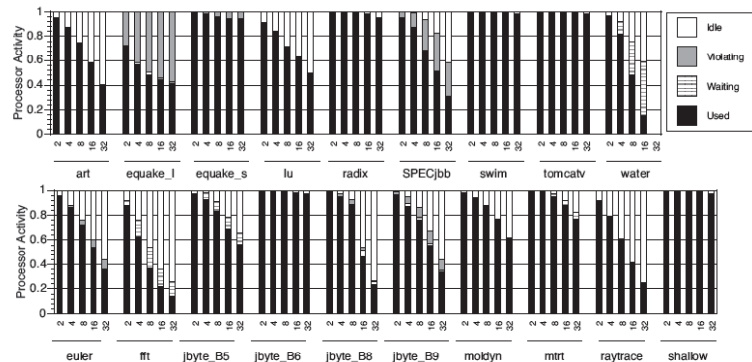


Figure 5: Distribution of execution time on the perfect TCC system's processors between useful work, violated time (failed transactions), waiting time (load imbalance in parallel code), and idle time (time waiting during sequential code).

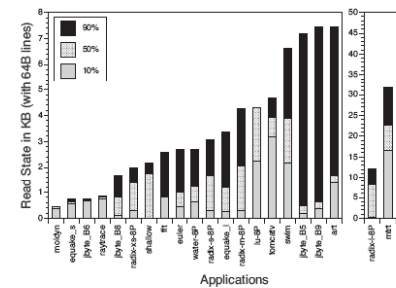


Figure 6: State read by individual transactions with store buffer granularity of 64-byte cache lines. We show state required by the smallest 10%, 50%, and 90% of iterations.

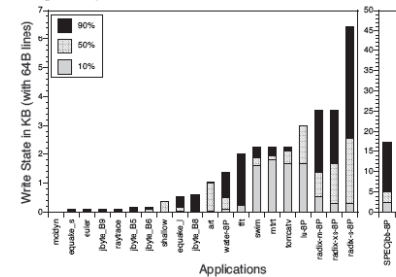


Figure 7: Same as Fig. 6, but for write state.

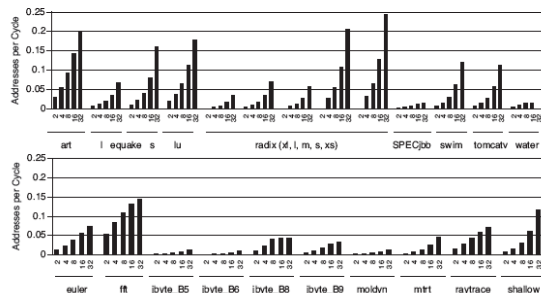


Figure 8: Number of 64 byte cache line addresses broadcast per cycle in our perfect system with 1 IPC,  $\infty$  bus bandwidth, and no cache misses. This indicates essentially the *maximum* snoop rate per avg. processor IPC that could be expected.

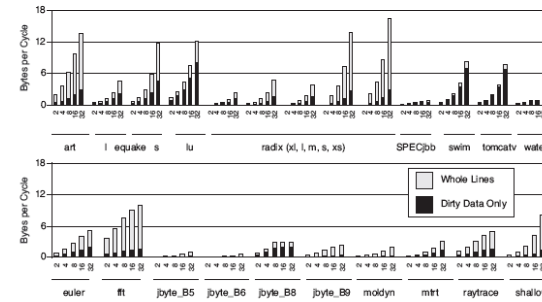


Figure 9: Average bytes per cycle broadcast by a 1 IPC system with infinite bus bandwidth, no cache misses, and TCC with an update protocol. 4 bytes of "address overhead" per 64-byte cache line are assumed.

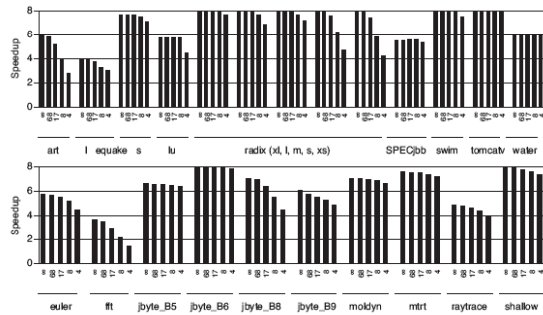


Figure 11: The effect of finite commit bus bandwidth (4, 8, 17, and 68 bytes per cycle) on the speedup of 8 processor systems writing 68-byte (4B header + 64B data) cache lines during commit (or 17, 9, 4, or 1 cycle per committed line).