From: Ravi Rajwar <ravi.rajwar@intel.com>
Subject: **TM workshop 4/8**
Date: March 28, 2005 7:52:45 AM PST
To: Brian Bershad <bershad@cs.washington.edu>

Dear Brian,

I am in the process of drawing up an agenda for the transactional memory workshop on 4/8.

I was wondering if you would be able to give a 10-15 minute talk on your perspective on the role of transactions for operating systems, including your past experiences.

I am putting aside a one-hour session on transactions and operating systems issues and would like you to give a short talk in that session.

Please let me know if you would be able to do this.

Thanks,

Ravi

The Role of Transactions for Operating Systems, including my Past Experiences

Brian Bershad
UW

MAKE and USE

- MAKE
  - OS can *implement* Transactional Memory for applications
- USE
  - OS can *use* Transactional Memory for its own purposes

MAKE all

- Library or Server-based systems
  - RVM
  - *Signal, page-get/set-state, thread-get/set-state*
- Kernel-based RVM
  - RHINO (on Spin) using extension technology
  - VINO as extension technology
  - *Signal, page-get/set-state, thread-get/set-state*
- Differ in performance and portability, but not function

MAKE some

- Compensate for missing hardware
  - No TestAndSet (MIPS)
    - Simulate the instruction (DEC Ultrix)
      - Trap, disable interrupts, read-modify-write, enable interrupts, RTI
      - Slow
    - Restartable sequences (Bershad & Redell 91)
      - Read-modify-write.
      - If the OS preempts within, PC is “adjusted” backwards.
        - » Works only on a uniprocessor
  - No CAS
    - Roll-out sequences (Bershad 93)
      - Lock, Read-compare-swap, Unlock.
      - If the OS preempts within, force unlock and adjust PC backwards or forwards, depending on point of incursion

USE

- Goal: OS Performs Atomic updates in the presence of concurrent activities
  - Eg, scheduling queue, fs buffer, etc.
- Non Goal: Recoverability
  - Although QuickSilver (86) is a notable exception

USE:Atomic Updates

- On a uniprocessor
  - Make all memory transactional by disabling interrupts
  - It's *ok* to do things like this inside the os
    - OS implements a VM, but doesn't need to run on one.
- Curiously,
  - Most potentially interruptible sequences are not interrupted.
  - Begin/End overhead dominates useful work
  - Under high load, lost data (missed interrupts) is common
    - Expensive B/E leads to high load sooner
  - Make it cheaper please
- Led to *Optimistic Synchronization* in the kernel (Stodolsky&Bershad 93)
  - Pretend interrupts are disabled
  - If an interrupt occurs, defer and disable

USE (Really)

- No Locks
  - Ability to withstand arbitrary delays w/o stalling
    - Eg, no more “page fault with lock held”
    - Cache misses
    - Processor failure
  - No priority inversion
    - Simplest case: acquiring a lock you already hold.
  - *Lock-free? Wait-free?* Who really cares?
- Examples
  - Synthesis (Massalin 91)
  - Cache Kernel (Greenwald & Cheriton 96)

USE: Lists, Versions, and DCAS

- There are lists and objects.
- Every list has a version number
- Version is incremented whenever the list changes
- Double-compare-and-swap (HW or SW)

"If neither the list nor the object have changed, change the list and the object.

```
int DCAS(int *addr1, int *addr2,
         int old1,   int old2,
         int new1,   int new2)
{
 <begin atomic>
 if ((*addr1 == old1) && (*addr2 == old2)) {
   *addr1 = new1;  *addr2 = new2;
   return(TRUE);
 } else {
   return(FALSE);
 }
 <end atomic>
}
```

USE: Exceptions

- What if the data structure/algorithm does not lend itself to DCAS?
  - *Herlihy's transform technique seems too difficult, and the others ones you can't understand*
- Abandon concurrency
  - Synchronous request/response to a serializing manager process

MAKE/USE: Conclusions

- MAKE
  - Low-end to high-end
  - Plenty of Tricks --solve the problem at its source
- USE
  - Intellectually interesting
  - A few successes (eg, opt spl)
  - Costs generally seemed greater than benefits
    - Hardware
      - missing or $$, especially under contention
    - Threads+locks got easier
      - Tools, language support
    - Depressing exceptions