TreadMarks

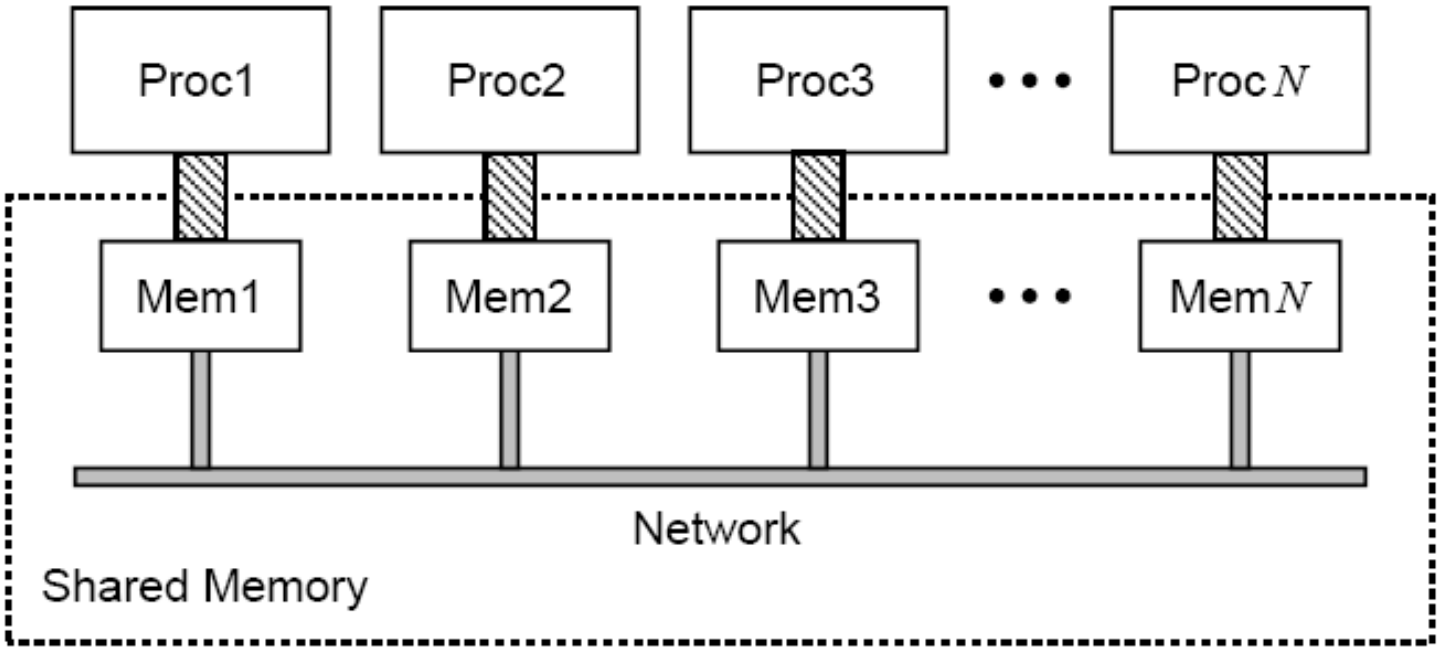
Andrew Hoskins

April 2, 2013

Outline

- DSM Motivation
- TreadMarks API
- Jacobi/TSP
- Sequential Consistency
- Release Consistency
- Single/Multiple Writer Protocols
- Conclusion

Distributed Shared Memory Motivation



- Globally shared virtual memory
- No communication detail burden
- Low cost

API

$Tmk\_startup(int\ argc, char**\ argv)$
$Tmk\_exit(int\ status)$

$Tmk\_malloc(unsigned\ size)$
$Tmk\_free(char*\ ptr)$

$Tmk\_barrier(unsigned\ id)$
$Tmk\_lock\_acquire(unsigned\ id)$
$Tmk\_lock\_release(unsigned\ id)$

Jacobi

```
length = M / Tmk_nprocs;
begin = length * Tmk_proc_id;
end = length * (Tmk_proc_id+1);

for( number of iterations ) {

    for( i=begin; i<end; i++ )
        for( j=0; j<N; j++ )
            scratch[i][j] = (grid[i-1][j]+grid[i+1][j]+
                             grid[i][j-1]grid[i][j+1])/4;

    Tmk_barrier(1);

    for( i=begin; i<end; i++ )
        for( j=0; j<N; j++ )
            grid[i][j] = scratch[i][j];

    Tmk_barrier(2);

}
```

TSP

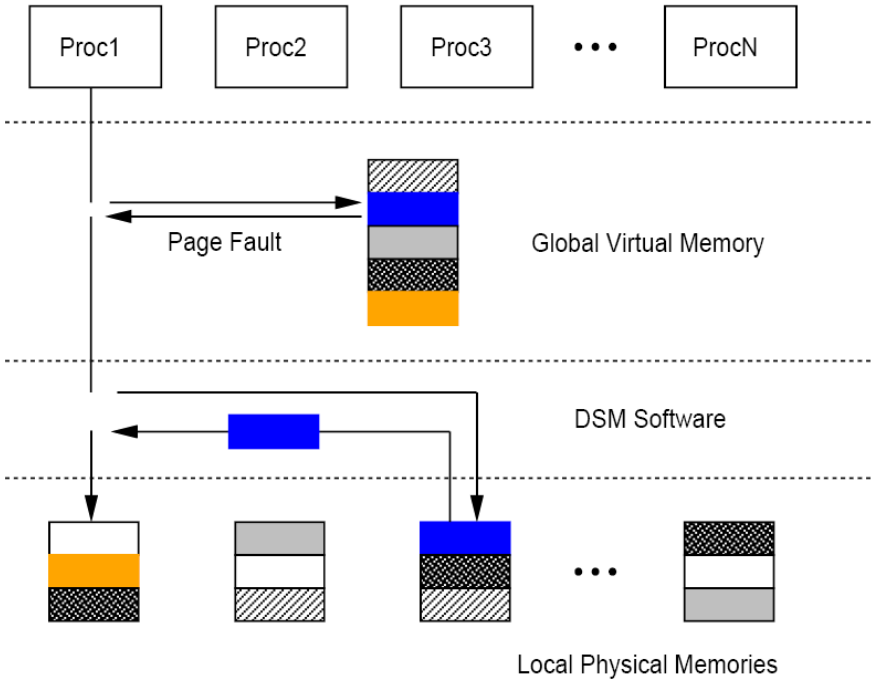
```
Tmk_barrier(0);

while( true ) do {
    Tmk_lock_acquire(queue_lock_id);
        if( queue is empty ) {
            Tmk_lock_release(queue_lock_id);
            Tmk_exit();
        }
        Keep adding to queue until a long,
        promising tour appears at the head;
        Path = Delete the tour from the head;
    Tmk_lock_release(queue_lock_id);

    length = recursively try all cities not on Path,
             find the shortest tour length

    Tmk_lock_acquire(min_lock_id);
        if (length < *Shortest_length)
            *Shortest_length = length;
    Tmk_lock_release(min_lock_id);
}
```

Sequential Consistency



Virtual memory page protection used to detect shared memory accesses

Sequential Consistency

Pros

- Simple

Cons

- False Sharing
- Ping-Pong Effect
- Unnecessary Communication Overhead

Release Consistency

- Correct parallel program should have no data races
- Data races should be prevented with synchronization
- Memory updates can be delayed until synchronization

This provides the appearance of sequential consistency given the exclusive use of the TreadMarks synchronization primitives

Release Consistency

```
length = M / Tmk_nprocs;
begin = length * Tmk_proc_id;
end = length * (Tmk_proc_id+1);

for( number of iterations ) {

    for( i=begin; i<end; i++ )
        for( j=0; j<N; j++ )
            scratch[i][j] = (grid[i-1][j]+grid[i+1][j]+
                             grid[i][j-1]grid[i][j+1])/4;

    Tmk_barrier(1);

    for( i=begin; i<end; i++ )
        for( j=0; j<N; j++ )
            grid[i][j] = scratch[i][j];

    Tmk_barrier(2);

}
```

Release Consistency

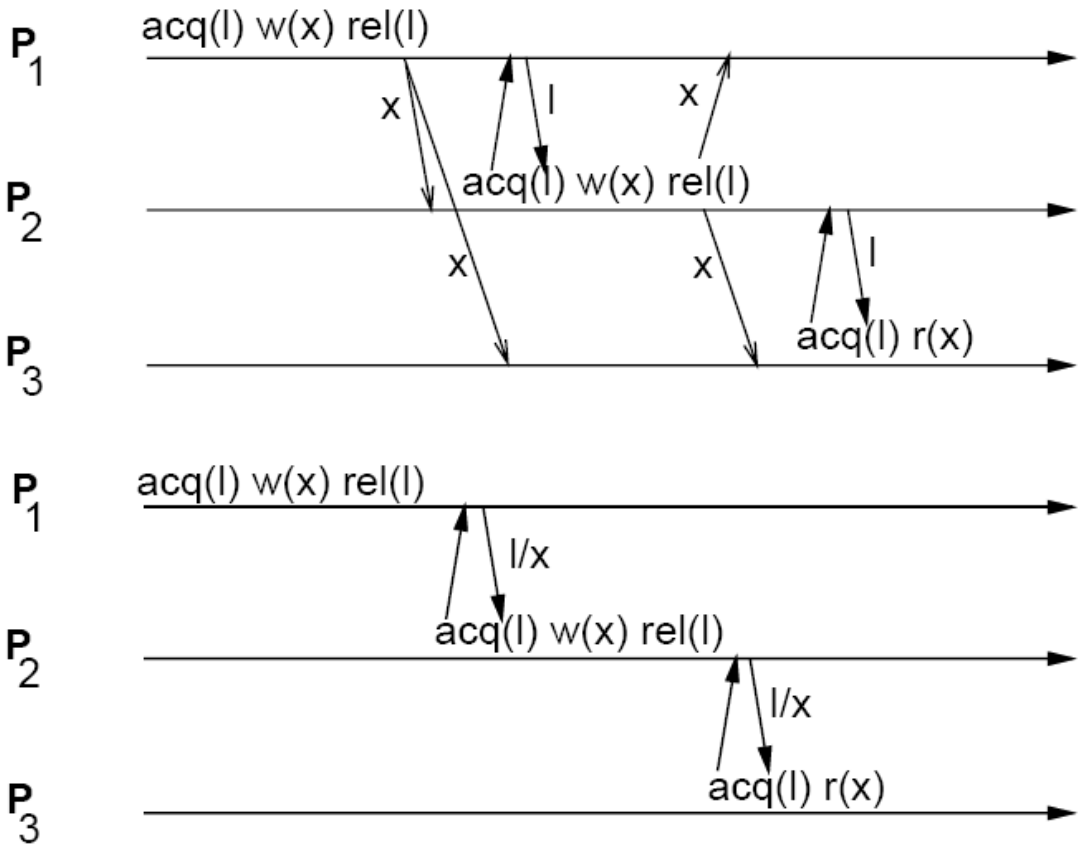
```
Tmk_barrier(0);

while( true ) do {
    Tmk_lock_acquire(queue_lock_id);
        if( queue is empty ) {
            Tmk_lock_release(queue_lock_id);
            Tmk_exit();
        }
        Keep adding to queue until a long,
        promising tour appears at the head;
        Path = Delete the tour from the head;
    Tmk_lock_release(queue_lock_id);

    length = recursively try all cities not on Path,
             find the shortest tour length

    Tmk_lock_acquire(min_lock_id);
        if (length < *Shortest_length)
            *Shortest_length = length;
    Tmk_lock_release(min_lock_id);
}
```

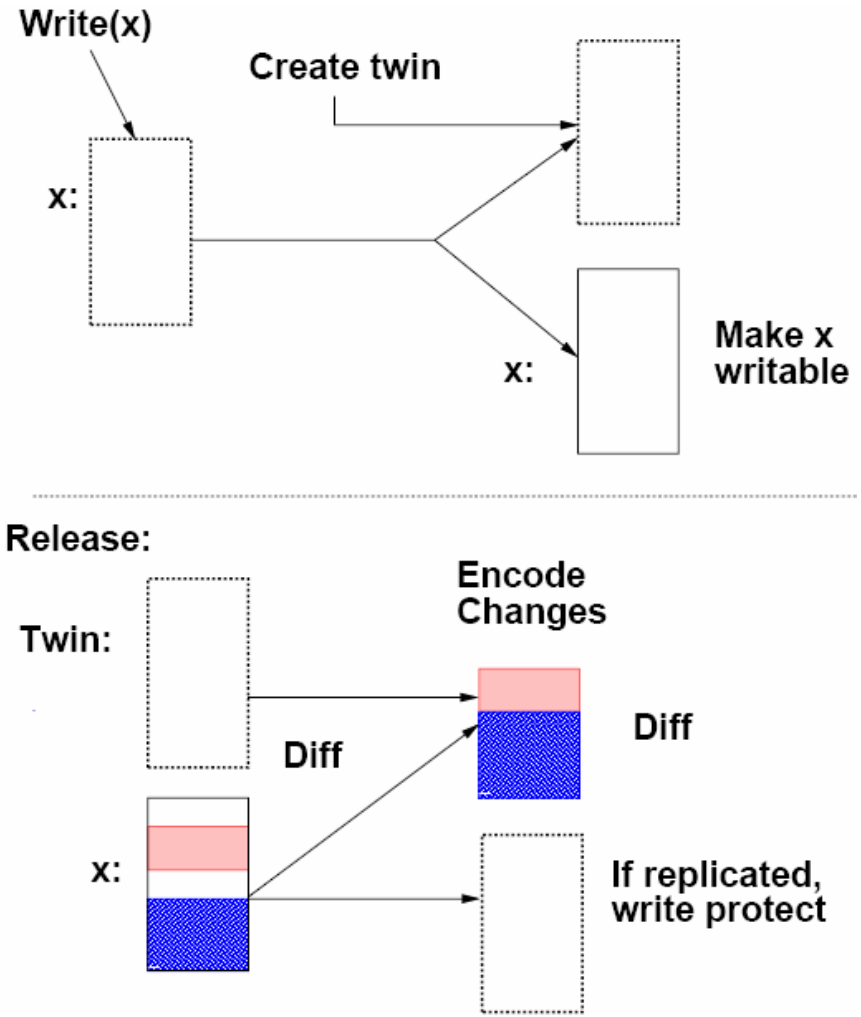
Release Consistency



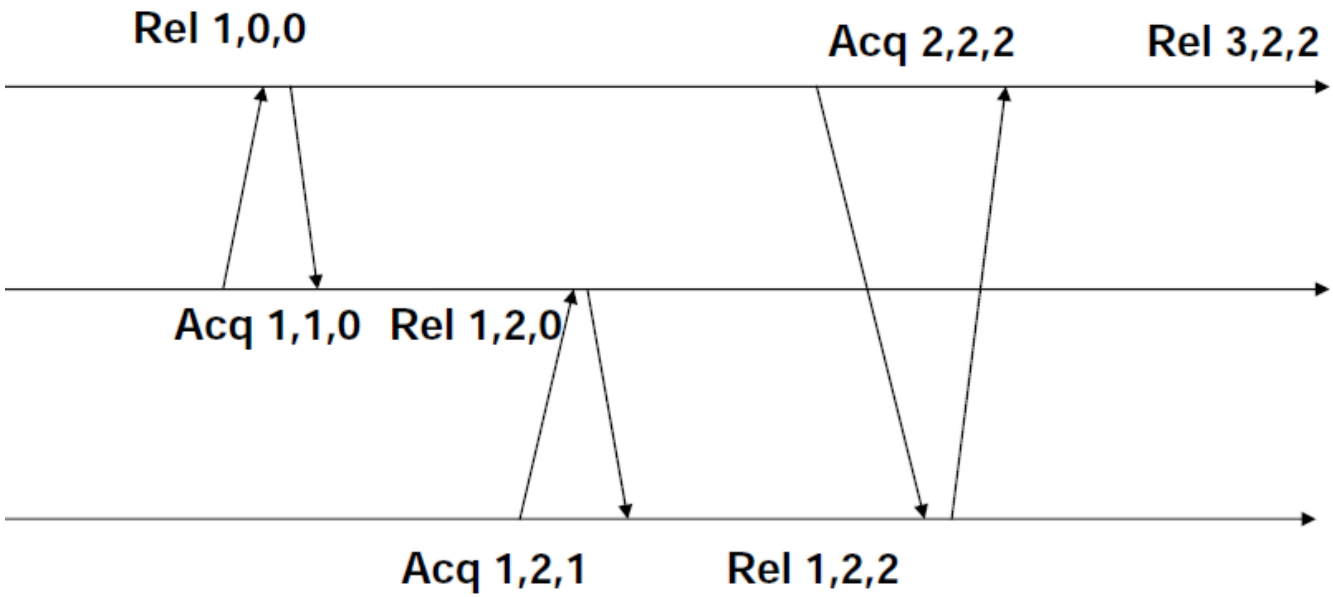
Single Writer Protocols

- Multiple reader
- Single writer
- Suffers from false sharing more-so than hardware systems

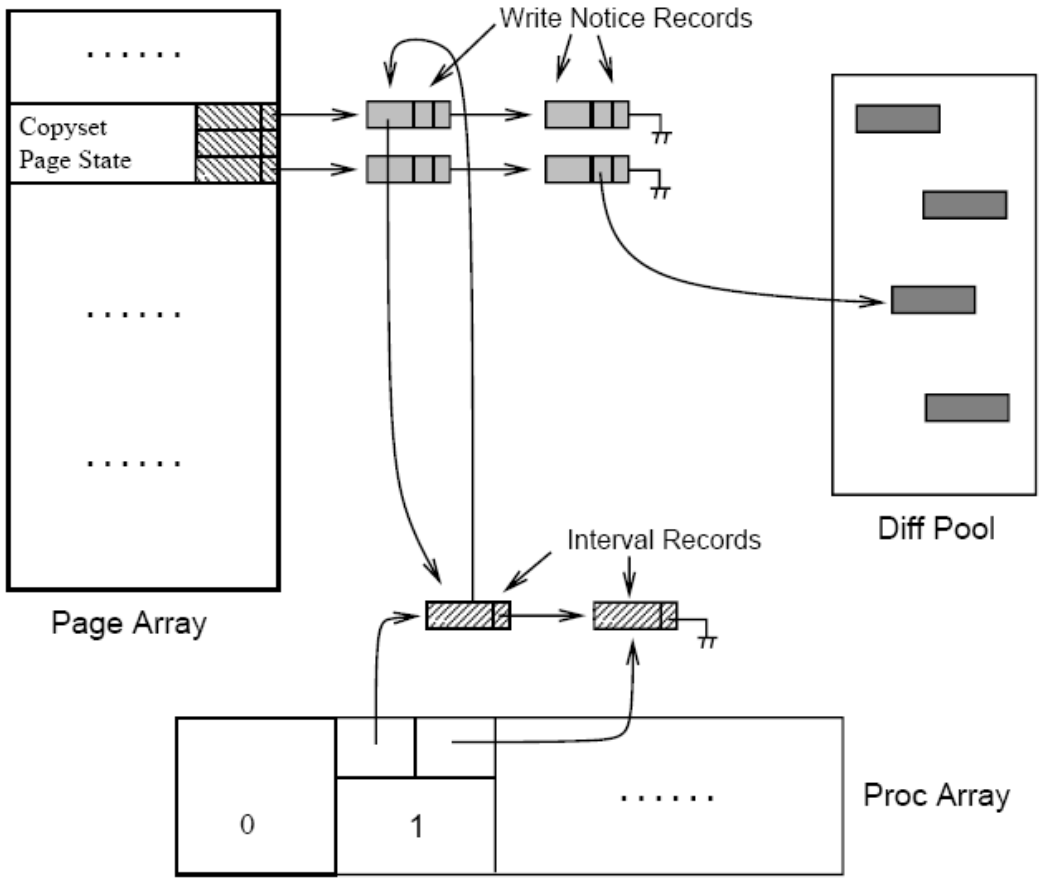
Multiple Writer Protocols



Timestamps



Implementation



Conclusion

- Lazy release consistency minimizes the need to push updates to other processors and allows updates to piggyback on lock acquisitions
- Multiple writer protocols minimize false sharing and reduce update size
- Requires no kernel or compiler support
- Not good for applications with a high frequency of communication and synchronization