Synchronization in Parallel Architectures

Outline:

- Performance Requirements
- Hardware / Software Alternatives and Trade-offs
- Types of Synchronization
- Mutual Exclusion
  - **efficient, scalable busy-waiting**
- Events
  - **point-to-point**
  - **barriers**

© Culler 1995

Motivation

- "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

- How is this cooperation orchestrated?
  - **syncronization operations**
- Two basic types
  - **Events**
  - **Mutual Exclusion**

© Culler 1995

Basic Types of Synchronization

- Events: enforce dependences, ensure order
  - **producers –> consumers**
  - **data, anti, and output dependences**
  - **point-to-point or group based (barrier)**
    - » **waiting for a "thread" to complete a step**
    - » **waiting for a "phase" of computation to complete**
  - **extra dependences to limit concurrency**

- Mutual Exclusion: enforce consistent use of resources
  - **no ordering requirements, just disjoint usage**
  - **access to shared data structures (e.g., task queue)**
  - **multiple-reader singler-writer (e.g., dictionary)**

© Culler 1995

Confluence of Architecture and O.S / Run-time SW

- Architecture side
  - **instruction set & memory system support**
    - » **test&set, swap, fetch&op, compare&swap, load-lock/store-cond**
  - **hardware barriers** (CM, Meiko, T3D)
    - » **global-OR, network scan operations**
  - **fetch&op combining** (Ultra, RP3)
    - » **exploit topology, avoid hot-spots**
  - **cached-based queuing locks (Goodman QOLB)**
  - **tagged memory** (HEP/Tera, Dataflow, Spur, J-machine, Alewife)
    - » **full / empty bit**
  - **multithreading** (HEP/Tera, Dataflow, Alewife)

- O.S. / run-time side
  - **sophisticated locking algorithms**
  - **low-contention scheduling / data structures**

© Culler 1995

Performance Requirements

- Low Latency
  - **Latency to communicate event or get exclusive access?**
    - » **Example tradeoff: cacheable vs. non-cacheable locks**
    - » **Example tradeoff: invalidate vs. update based wakeup**
- High Throughput:
  - **How many synch operations can be done per unit time?**
    - » **Example tradeoff: fetch-&-op done at memory vs. in cache**
    - » **Example tradeoff: degree of pipelining in memory system**
- Scalability
  - **How are latency and throughput affected as number of participating PEs increases?**
    - » **Example tradeoff: HW combining vs. SW combining**
- Low Storage Overhead:
  - **bits per lock, bits per access**

- Correctness: Causality, Deadlock, Starvation(?), ...

© Culler 1995

Mutual Exclusion

- Enforced by design in message passing models
  - **each node has exclusive to local address space**
  - **buffering in send/receive message passing**
  - **handler as critical section in Active Messages**

- Enforced by lock / unlock in shared memory
  - **use atomic read-modify-write hardware primitives on shared variables**
  - **suspend and reschedule on event**
  - **busy-waiting**
    - » **consumes processor and network bandwidth**
    - » **advantageous when scheduling is more expensive than expected wait and resources are not otherwise needed**
    - » **also where suspension is not an option (e.g., kernel)**

© Culler 1995

Efficient Busy Waiting

(Mellor-Crummey and Scott, "Algorithms for Scalable Synchronization on Shared-Memory Machines,", ACM TOCS, v. 9, No. 1, feb. 1991)

- HW Assumptions
  - **simple fetch&op support**
  - **ability to use local locks or cache locks**
- Five Locking Algorithms
  - **simple test-and-set lock**
  - **spin lock (test and test-and-set)**
  - **ticket lock**
  - **array-based queueing lock (Anderson, et al., Graunke &Thakker)**
  - **list-based queueing lock (MCS)**

© Culler 1995

Simple Test&Set Lock

```
while (test&set(L) == LOCKED) {}
    ... critical section ...
```

- **Cacheable**
  - **good if good locality on lock access (low contention)**
    - » **reduces both latency and lock duration**
      - **particularly if lock acquired in exclusive state**
    - » **many concurrent lock operations on different locks**
  - **not good for high-contention locks!**
- **Non-cacheable**
  - **read-modify-write at main memory**
  - **easier to implement**
  - **high latency**
  - **better for high-contention locks, but not good !!!**

© Culler 1995

Spin Locks (test and test-and-set)

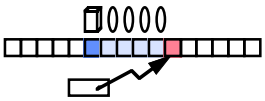
```
A:  while (L == LOCKED) {};   /* spin in cache */
    if test&set(L) == LOCKED goto A;
    ... critical section ...
```

- (–) Extra load and test on uncontended lock
- (+) Spin in cache contended lock
  - **releasing the lock invalidates or updates all the waiters**
- (–) All waiters grab for newly released lock
  - **O(p2) remote access**
  - **high contention on each**
  - "The simple lock algorithm should never be used on systems with support for cache coherence. Locks that may encounter contention should be protected with some form of collision avoidance." **Graunke & Thakker of Sequent, IEEE Computer, June 1990.**

© Culler 1995

Collision Avoidance

- Exponential Backoff
- Tickets
  - **two counters per lock (**next_ticket**,** now_serving**)**
  - **do fetch&inc on** next_ticket **then wait on** now_serving
  - FIFO order, lowish latency if **fetch&inc cacheable**
  - **not O(1) traffic per lock acquire**
- Array-based Queueing Locks



  - **fetch&inc give address on which to spin**
  - **every process spins on distinct location**
    - » **ensure in different cache lines or different processors**
  - **FIFO ordering, O(1) traffic per acquire with consistent caches**
  - **O(P) space per lock**
- List-based locks
  - **use compare&swap to build linked-lists in SW**
  - **locally allocated list per node to spin on**
  - **swap is sufficient, but lose FIFO property**

© Culler 1995

Performance of Spin Locks and Queueing Locks

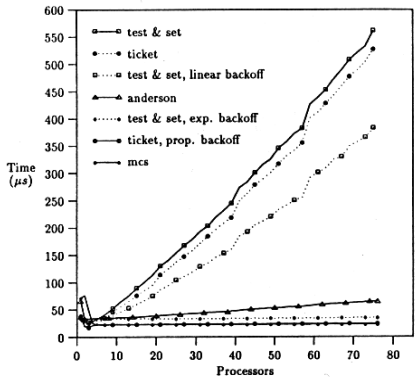


Fig. 15. Performance of spin locks on the Butterfly (empty critical section).

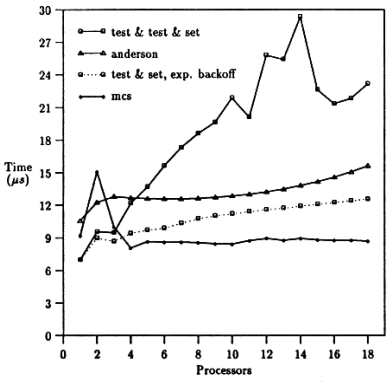


Fig. 17. Performance of spin locks on the Symmetry (empty critical section).

- With non-empty, yet small critical sections difference vanishes
  - **Anderson's Array-based lock comes out slightly ahead**

© Culler 1995

Recent Areas of Investigation

- Multi-protocol Synchronization Algorithms
  - **Reactive Synch. Alg., Lim & Agarwal, ASPLOS-VI**
  - **Adaptive Wait Mechanisms**
  - **Wait-free algorithms**
- Object-based synchronization
  - **Active Message handlers**
  - **Multipol (Yelick)**
  - **Flash ???**
- Integration with scheduling
- Multithreading
  - **what do you do while you wait?**

© Culler 1995

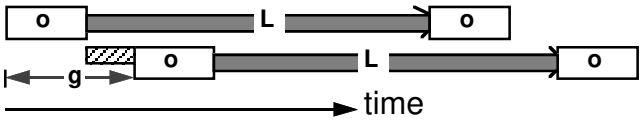
Barriers

- Enforce dependences among entire group between phases of computation
  - **fewer, more-powerful synchronization events**
- Hardware Approaches
  - **Global-OR, Global-AND, Reduction Tree, Scan Tree**
  - **Broadcast**
  - **Often pipelined**
  - **Difficult to support arbitrary subset of processors**
    - » **even harder with multiple processes per node**
  - **Composition with other communication operations is critical**
- Message Passing
  - **combining-trees, broadcast trees (including linear)**
- Shared Memory
  - **shared variables (centralized)**
  - **tree of variables**
  - **tournaments, dynamic trees, . . .**

© Culler 1995

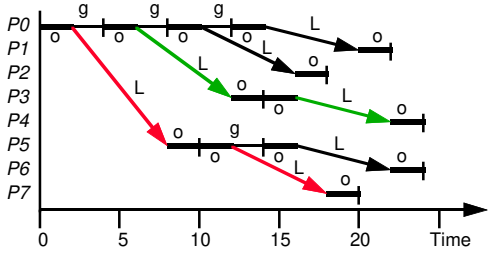
Optimal Broadcast & Reduction Under LogP

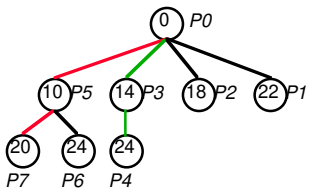
Model: Latency, Overhead, Gap



Optimal single item broadcast is an unbalanced tree

– shape determined by relative values of *L, o,* and *g.*





L=6, o=2, g=4, P=8

© Culler 1995

Barrier under shared memory

- Centralized
- Software combining tree
  - **each node is a variable in a separate memory module**
  - **leaf is a group of processors, including one "representative"**
    - » **last to arrive**
  - **write into one tree "on arrival"**
  - **read from second tree "on exit"**
  - **reverse sense for consecutive barriers**
    - » **cache coherent machines allow local spin, hard without CC**
- Dissemination Barrier
  - **log P rounds of synchronization**
  - **in round k, proc i synchronizes with proc (i+2^k) mod P**
    - » **can statically allocated flags to avoid remote spinning**
  - **like a butterfly network**
- Tournament Barrier
- MCS tree-based barrier

© Culler 1995

Tournament and MCS Barrier

- Tournament
  - **like combining tree, but representive node chosen statically**
    - » **no fetch&op needed**
  - **in round k, each proc i = 2^k mod 2^{k+1} sets flag for proc j = i - 2^k**
    - » **i drops out and j goes on to next round**
    - » **i waits for global completion flag**
  - **root sets a global variable, which all wait on**
    - » **can be replaced by wake-up tree**
- MCS
  - **tournament barrier with static allocation in wake-up tree, sense reversal, spins on local flag**
  - **pair of P-node trees**
    - » **fan-in 4 arrival tree and fan-in 2 wake-up tree**
  - **O(P) space, 2P-2 network transactions (theoretical min)**
  - **O(log P) network transactions on critcal path**

© Culler 1995

Performance of Barriers with local memory

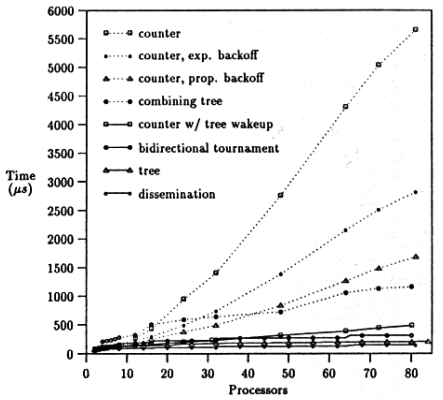


Fig. 19. Performance of barriers on the Butterfly.

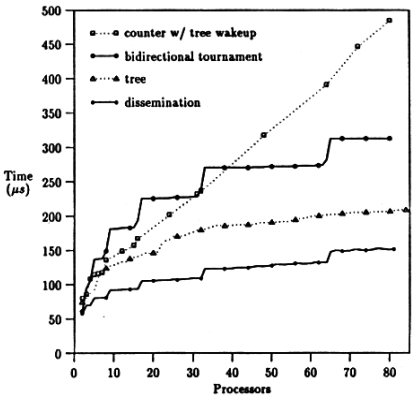


Fig. 20. Performance of selected barriers on the Butterfly.

© Culler 1995

Selected Barriers w/o local access

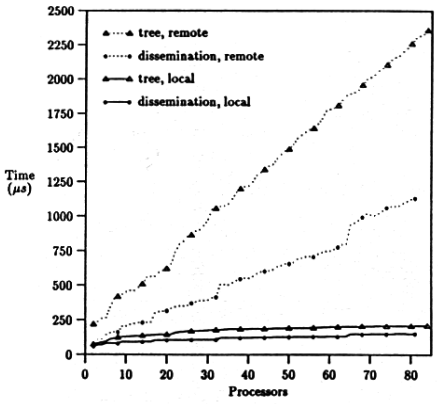


Fig. 22. Performance of tree and dissemination barriers with and without local access to shared memory.

© Culler 1995

Recent Directions

- Adaptive tree barriers
  - **late arrivals should be close to the root**
- Pipelined Scan Operations
- Hardware Support

© Culler 1995

Summary

- Synchronization operations in parallel architectures provide a rich interaction of:
  - **architecture & operating system**
  - **HW / SW trade-offs**
- Sophisticated HW support must be evaluated in the context of sophisticated synch. algorithms
  - **HW primitives determine which algorithms are applicable**
  - **HW performance determines which are effective**
- Everything has changed since MCS paper, yet little new data!!!
  - **architectures have advanced**
  - **programming models have advanced**
  - **networks have advanced**
  - **applications have advanced**
  - **processors and memory systems are radically different**

© Culler 1995