**CS/ECE 758: PROGRAMMING MULTICORE PROCESSORS
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON**

Prof. Mark D. Hill
Assistant Jason Power

Examination
In-Class
Monday, October 29, 2012
Weight: 25%

 1:15 minutes.

**CLOSED BOOK**, etc., but one cheat sheet allowed (two-sided 8.5x11 page).

The exam is two-sided and has **EIGHT** pages, including two blank pages at the end.

Plan your time carefully, since some problems are longer than others.

NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **KEY** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **ProblemNumber** | **MaximumPoints** | **ActualPoints** |
| 1 | 12 |  |
| 2 | 12 |  |
| 3 | 12 |  |
| 4 | 12 |  |
| 5 | 12 |  |
| Total | 60 |  |

**Problem 1: Math, etc. (12 points)**

* *(3 points)* Consider a job that is 1/8 sequential and 7/8 perfectly parallel. What is the execution time speed up from moving from 1 processor core to (i) 4, (ii) 8, or (iii) an infinite number of cores?
1. *1 / [1/8 + (7/8)/4] = 32/11 = 2.91*
2. *1 / [1/8 + (7/8)/8] = 64/15 = 4.27*
3. *1/[1/8 + 0] = 8*
* *(4 points)* Consider a CILK-like job that executes one task that spawns four tasks that each spawn three tasks and then magically completes. Assume that each task takes five time units. Write expressions for the job’s execution time on (i) 1 core, (ii) 6 cores, (iii) 12 cores, or (iv) an infinite number of cores.
* *(i) 5\*[1+4+12] = 5\*17 = 85*
* *(ii) 5\*[1+1+2] = 5\*4 = 20*
* *(iii) 5\*[1+1+1] = 5\*3 = 15*
* *(iv) 5\*[1+1+1] = 5\*3 = 15*
* *(5 points)* For many practical problems using linear systems of equations, most matrix values are zero. Give an example of one way to represent such a *sparse matrix* wherein zero-valued elements are not explicitly stored.

*See textbook or Wikipedia, but for a 2D matrix, most representations have three 1D arrays that, for non-zero elements, give values, column index, and row index.*

**Problem 2: Locks (12 points)**

Recall that the synchronization primitive compare and swap (CAS) stores a new value in an address “a” if and only if “a” is equal to an old value:

Boolean CAS(word \*a, word old, word new):
atomic{
 t:= (\*a == old);
 if (t) \*a := new;
 return t;
}

Assume that 0 means “unlocked” and non-0 means “locked.”

(a) Using CAS and simple loads, stores, etc., implement a very simple lock(word \*L) subroutine that returns only when lock L is obtained.

Lock: while (!CAS(\*L,0,1)) {} /\* returns when L was 0 \*/

* Implement unlock(word \*L).
* Unlock: \*L = 0 /\* CAS unnecessary \*/
* Discuss ways to improve the performance of lock(word \*L) vs. your answer to part (a).

*See Michael Scott’s Chapter 4: test-and-test-and-set, exponential backup, ticket locks, MCS locks, blocking, etc.*

**Problem 3: GPUs (12 points)**

The *CUDA programming model* seeks to enable general-purpose programming on GPUs, such as the NVidia GeForce 8800.

* Describe the *advantages* of CUDA/GPU programming relative to writing programs for the host (x86) processor(s).
* *Provides access to a vast number of threads on cheap hardware*
* *Software and hardware manages thread at low overhead.*
* *Well suited to many data parallel applications are or are close to being SIMD.*
* Describe the *disadvantages* of CUDA/GPU programming relative to writing programs for the host (x86) processor(s).
* *Must be almost-completely data parallel (c.f., Amdahl’s Law).*
* *High-level sharing across all thread only via shared/group memory.*
* *Thread’s can’t dynamically react to or synchronize with recent activity of other threads.*
* *No promise of backward or forward compatibility.*
* *Programmer must confront many low-level detail, especially about memory sizes and locations.*
* List and briefly describe four different optimizations that that can improve the performance of general-purpose GPU programs.
* *Create data-parallel code*
* *Manage global memory bandwidth*
* *Minimize host-GPU transfers*
* *Seek memory access that coalysce*
* *Minimize global memory access*
* *Reduce branch divergence with a warp*

**Problem 4: Transaction Memory, etc. (12 points)**

* Programming with locks can *deadlock* while programming with transaction memory cannot. Why? Consider presenting the conditions necessary for deadlock.

*Deadlock requires four necessary conditions: (1) more than one lock, (2) hold & wait, (3) no revocation of lock holders, and (4) cyclic dependences.*

*To eliminate deadlock, lock-based program usually try to establish (partial) order of locks avoid (4) cyclic dependences.*

*Many TM systems implicitly get read/write locks on data. They eliminate deadlock by aborting one or more transactions on conflict that revokes the implicit locks, thereby avoiding (3) no revocation of lock holders, and (4) cyclic dependences.*

* Some hardware transaction memory systems allow only *bounded* transactions, while others enable *unbounded* transactions. Which is easier to program? Which is easier to implement in hardware? Why?

*It is easier to program with unbounded TM systems where correctness does not depend on the programmer tracking write or read set size, especially in the present of opaque library calls.*

*Bounded TM is easier to implement in hardware, which has natural bounded structures such as write buffers, caches, and physical memory.*

* For database transactions, what are the key ideas in *Bayer and Schkolnick’s “Concurrency of Operations on B-Trees”?*

About concurrent operations on B-trees (as the title suggests):

*0. Single Lock*

*1. Pessimistic Writer: spider lock for read and simple write*

*2. Optimistic Writer: unchanged reader while writer (i) tries read locks at non-leaf levels, (ii) Gets write lock on leaf; if safe does update, (iii) If not, releases all locks and does protocol 1*

*3. Balanced Writers: unchanged reader while writer (i) grabs alpha (intent) locks and (ii) converts alpha locks to write locks as necessary*

*4. Generalized Solution -- …*

**Problem 5: Reader (12 points)**

* *(4 points)* How does the *Sun Niagara* chip (Kongetira, et al.) differ from most multicore chips today, such as those you used in homework assignments 1-4?

*Uses very simple (6-stage pipeline) in-order cores*

*Had 8 cores back then which is now starting to happen*

*4-way multithreading (vs. more-common 2-way)*

*Seeks to tolerate latency more than reduce it*

*Optimized for throuput of server workloads*

*(b) (4 points)* What are the key arguments of *Edward Lee’s “The Problem with Threads”?*

*Threads are a poor model: low-level, non-deterministic, bug-prone, deadlock possible.*

*Laments programs prefer familiar syntax even with greatly-changed semantics*

*Calls for better abstractions; gives some of his own examples.*

* *(4 points)* What are the key ideas of *Gagan Gupta and Gurindar S. Sohi’s, “Dataflow Execution of Sequential Imperative Programs on Multicore Architectures”?*

*Advocates decorating an object-oriented serial program with read-write set on object so that a custom run-time that object execute methods in parallel, still preserving serial semantics, while getting performance comparable to pthreads. Inspired by out-of-order superscalar cores, but taken to a coarser grain more appropriate to multicore chips.*

**Scratch Sheet 1 of 2 (in case you need additional space for some of your answers)**

**Scratch Sheet 2 of 2 (in case you need additional space for some of your answers)**