Comp. Arch. Test 2 Name:_____

1. Some characteristics of a "good" parallel program are correctness (i.e., gives the correct answer), performance, and scalability (i.e., speedup continues to improve as the number of processors increases).

a) Explain why correctness of a parallel program (e.g., pthread's program) is more difficult than the corresponding sequential program.

actions by pthreads need to be coordinated, e.g.s,
- updating of shared variable
- synchronization at a barrier

A sequential pgm has nothing to coordinate with.

8

b) When designing a parallel program, we focus on data parallelism instead of task parallelism since "task parallelism does not scale well with the number of processors, P. Explain why task parallelism does not scale well with P.

- each task needs a separate program instead of decomposing data among processors.
- tasks might have dependencies so they cannot all work simultaneously.

7

2. Some categories of performance loss in parallel programs are:
- Parallelization overheads: communication, synchronization, computation, memory
- Non-parallelizable code - Amdahl's law
- Idle processors - load balancing, memory-bound computations
- Contention for shared resources

Work (and corresponding data) can be allocated statically to threads/processes or dynamically via a work queue.

a) What type of performance loss are you trying to avoid by using dynamic allocation via a work queue? (explain)

Idle processors

7

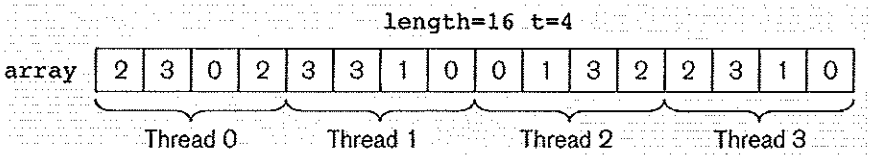
b) What new overhead might you introduce to manage a work queue to dynamically allocate work?

potentially adds:
- synchronization overhead + comm. to access work queue
- contention for shared resource - work queue

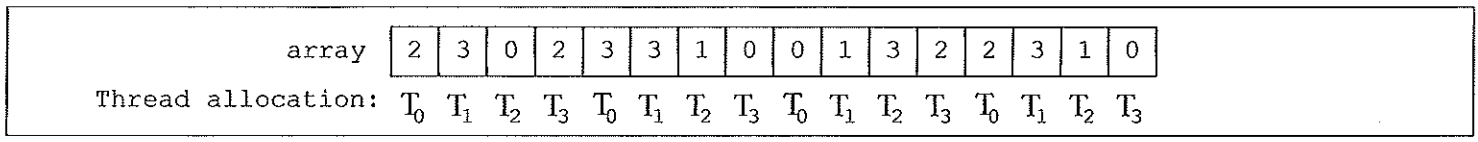
8

30

3. My 1D array summation program used a block allocation of the array to threads:



Instead consider using a cyclic allocation of the array to threads:



Compared to block allocation, do you think that cyclic allocation would increase execution time, decrease execution time, or have no impact? (Justify your answer)

Increase execution time with cyclic allocation due to increased hit rate. (cyclic can only use one value per cache line loading.

10

4. Consider the given 1D array summation thread code which uses a block allocation, but has each thread update a global variable sum.

```
void  * threadPartialSum(void * args) {

  int i, threadId;
  int start_index;
  int end_index;
  float * myArray;

  start_index = ((RANGE *) args)->start_index;
  end_index = ((RANGE *) args)->end_index;
  threadId = ((RANGE *) args)->threadId;
  myArray = ((RANGE *) args)->arrayToSum;

  pthread_mutex_lock(&updateSumLock);
  for (i=start_index; i <= end_index; i++) {
    sum += myArray[i];      // update global sum
 } /* end for (i */
  pthread_mutex_unlock(&updateSumLock);

} // end threadPartialSum
```

a) Would this code give the correct answer? (Justify your answer) Yes, since only one thread accesses the global sum at a time.

8

b) Could this code deadlock? (Justify your answer)

NO; since only one mutex to "hold" so no circular wait

8

c) Suggest ways to improve this code.

Only one thread doing summation at a time, so no parallel computation being achieved, so many ways better.
- move mutex to around "sum += myArray[i];"
- in loop, sum into a local variable, either after loop copy local sum to myArray[i]. -- no need for mutex

9

35

5. Consider a pthread solution for the *Red/Blue computation* which simulates two interactive flows:
- an *n* x *n* board is initialized so cells have one of three colors: red, blue, or white, where white indicates empty cells
- in the first half step of an iteration, any red color can move right one cell if the cell to the right is empty (white)
- in the second half step of an iteration, any blue color can move down one cell if the cell below is empty (white) (the case where red vacates a cell during the first half of an iteration and blue moves into it during the second half of an iteration is allowed)
- the board wraps around to the opposite side (i.e., left-to-right; top-to-bottom) when colors reach the edge
- viewing the board as overlaid with *t* x *t* tiles (t evenly divides n), the computation terminates if any tile's colored squares are more than *c*% one color (i.e., *c*% of tile blue or *c*% of tile red)

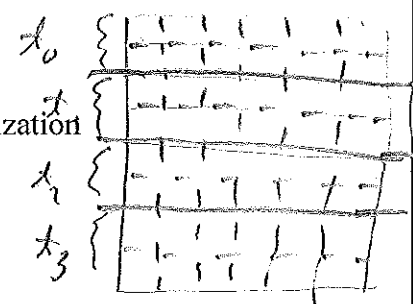
a) How would you assign work to each pthread (i.e., decompose the board so each pthread was responsible for updating a portion of it)?

If t << n, we might decompose by rows of tiles.

10 Certainly blocked decomposition

Use two boards: Val = current; new = next 1/2 time step

tile ...



b) Describe a high-level algorithm for the function run by each pthread include synchronization

fill Board set global Done flag to false.

while (true) {

check threads tiles if any tile > c% use mutex to set Done=tru

barrier on all threads

if Done then break

Calculate updated red positions for rows a thread is

25 responsible for in new board (copy blues to/from two)

barrier on all threads

if thread id == 0

swap new & val pointers

end if

barrier on all threads

two boards, no race condition on blue moves.

Calculate ... duplicate for updating blue over partial columns

(Idea: White cells time i ⇒ time i+1

flip thread's role - row of board in mod to - column of board on blue.

35