ECE 56300 Spring 2017 Exam 2

All questions are worth 5 points.
For isoefficiency questions, do not worry about breaking costs down to t_c, t_w and t_s.

Question 1. Innovative Big Machines has developed special hardware that allows their machines to do reductions on four numbers at a time. Thus the complexity of a reduction's communication cost is $log_4 P$ instead of the typical $log_2 P$. Thanks to special Time Travel Logic (TTL), the cost of doing the three recurrence operations at each step (e.g., +, *, etc.) is 0. What is the isoefficiency of a program that does n^2 operations and a reduction?

A simple solution is:

$Tp = n^2/P + \log_4 P$

$To = P\,Tp = n^2 + P \log_4 P - n^2$
$= P \log_4 P$ which is the isoefficiency relation

Question 2. Time Industries, the developers of the Time Travel Logic, are unable to deliver, and so reduction is redesigned. The new reduction has P0 send its partial sum to P1, which adds its partial sum to P0's, and then sends this sum to P2, and so forth. What is the isoefficiency relation of this reduction? Is it better or worse, in terms of scalability, than the isoefficiency relation of Question 1?

Solution:

The recurrence takes O(P-1) time, which is close to O(P) time for large P. This is because of the sequential nature of the communication. Therefore,

$Tp = n^2/P + P$

$To = P\,Tp = n^2 + P^2 - n^2$
$= P^2$ which is the isoefficiency relation

Question 3. Bob is given the job to write a program that will get a speedup of 3.8 on 4 processors. He makes it 95% parallel, and goes home dreaming of a big pay raise. Using Amdahl's law, and assuming the problem size is the same as the serial version, and ignoring communication costs, what speedup will Bob actually get?

Solution (using Amdahl's Law):

Speedup = $1/(f - (1 - f)/P)$

$= 1/(.05 - .95/4)$ **(1)**

$= 3.478$

Full credit was given if you set up the fraction (3) correctly.

Question 4. You need to decide which of two parallel programs to use. You will run on 1024 processors, but you currently only have 16 processors to measure performance. You collect the following information, where Ψ is speedup and e is the experimentally determined serial fraction.

Program A

p	2	4	8	16
Ψ	1.88	3.12	4.5	5.8
e	0.06	0.09	0.11	.12

Program B

P	2	6	8	16
Ψ	1.79	2.9	4.3	5.7
e	0.12	0.12	0.12	0.12

Which program will give the best performance on 1024 processors? Explain your answer in ~30 words or less.

Solution (Karp-Flatt):

Program B is best because its sequential overhead is not growing, and therefore going to larger numbers of processors it will have better speedups.

Question 5. A parallel program is run on 1, 2, 4, 8, 16 and 32 cores, and we get the following speedups and efficiency.

P	2	4	8	16	32	64
Ψ	1.5	4.2	7.6	15.12	30.24	60.48
E	0.75	1.05	0.94	0.92	0.90	0.88

The speedup for 4 processors is greater than 4, and the efficiency is greater than 1. The amount of computation is the same as in the sequential version. We can conclude that the super-linear speedup is the result of (circle the correct answer.)

(a) More cache misses because of poor locality
(b) Fewer cache misses because of good locality
(c) Fewer cache misses because of more cache available on four cores than two, and sufficiently good locality to take advantage of this.
(d) Good programming, and the program would have gotten a super-linear speedup even if processors did not have caches.

Question 6. A programmer has parallelized 99% of a program, but there is no value in increasing the problem size, i.e., the program will always be run with the same problem size regardless of the number of processors or cores used. What is the expected speedup on 20 processors? Show your work.

Solution (using Amdahl's Law):

1/(.01 + (1-.01)/20
= 16.81
Again, full credit was given for setting up the fraction.

Question 7. A programmer has parallelized a program such that s, the amount of time the program spends in serial code, is .01. Moreover, the the problem size can expand as the number of processors increases. What is the expected speedup on 20 processors? Show your work.

Solution (using Gustavson-Barsis):

Speedup = P + (1-P) s
= 20 – 19 * 0.01 = 19.81
Full credit was given for setting up the problem.

An OMP program contains the following code:

```
void search(struct node* root, int findValue,
int level) {
  if (root == NULL) return;

  if (root->data == findValue) found++;

   if (level < 1) {
     seqSearch(root→left, key);
     seqSearch(root→right, key);
   } else {
     #pragma omp task
      search(key, self->left, level+1);

      #pragma omp task
      uniSearch(key, self->right, level+1);
}
```

Question 8. **found** is a global variable. Is there a race on the increment? **Yes.**

Question 9. Assume you are not sure about the answer to Question 8, and you want to be sure that there is not race. What choices do you have to ensure there is no race? Circle all that are true.

(a) Use an OpenMP *critical* pragma
(b) On processors where the increment operation can be done using an atomic hardware increment, use an OpenMP *atomic* pragma
(c) There cannot be a race because the increment is not contained within a *#pragma omp parallel* or a *#pragma omp task* inside the function.
(d) The odds of two threads updating **found** at the same time is small, so there is no need to worry about it.

Questions 10 – 14 use the following code.

Other answers may be correct.
Let me know if you think you lost points for a correct answer

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void main() {

  int c1 = 0;
  int c2 = 0;
  int i;

  omp_set_num_threads(4);

  #pragma omp parallel
  {
    #pragma omp atomic
    C1++;
    printf("par: %d\n", omp_get_thread_num( )); //L1

    #pragma omp master
    {
      #pragma omp critical
      {
      c2++;
      printf("master: %d\n", omp_get_thread_num( )); //L2
      }
    }

    #pragma omp single
    {
      #pragma omp critical
      {
      printf("single: %d\n", omp_get_thread_num( )); //L3
      c2++;
      }
    }
  }
  printf("c1: %d, c2: %d\n", c1, c2); //L4
}
```

Question 10. What value(s) are printed for the/a thread number in ***L1***?
(a) 0
(b) 0, 1, 2 or 3
(c) 0, 1, 2 and 3

Question 11. What value(s) are printed for the/a thread number in ***L2***?
(a) 0
(b) 0, 1, 2 or 3
(c) 0, 1, 2 and 3
(b) is also correct since although the master thread is usually 0, it is implementation dependent and only must always the be same thread.

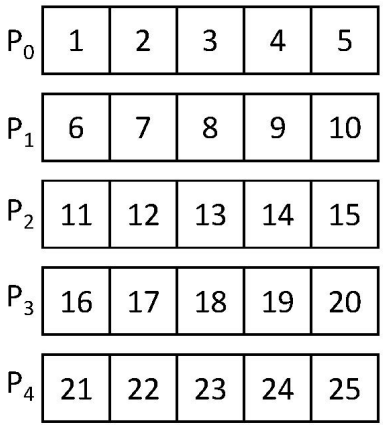
Question 12. What value(s) are printed for the/a thread number in ***L3***?
(a) 0
(b) 0, 1, 2 or 3
(c) 0, 1, 2 and 3

Question 13. What value(s) are printed for the c1 in ***L4***?
(a) 0
(b) 1
(c) 2
(d) 3
(e) 4
(f) some other value

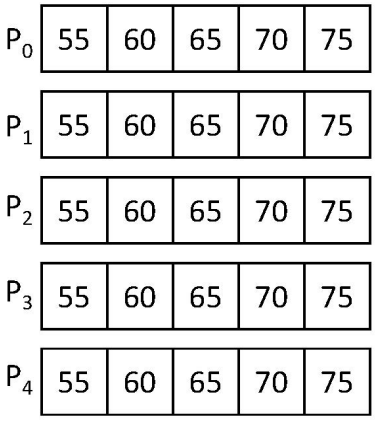
Question 14. What value(s) are printed for the c2 in ***L4***?
(a) 0
(b) 1
(c) 2
(d) 3
(e) 4
(f) some other value

Question 15. Bob gets a job programming MPI programs. His program only uses *SSend* and *Srecv* communication primitives and does not deadlock. He later changes these to non-blocking Isend and Irecvs, with a correct and valid Wait instruction immediately after each Isend and Irecv. Can his program deadlock from communication. Answer "yes" or "no".

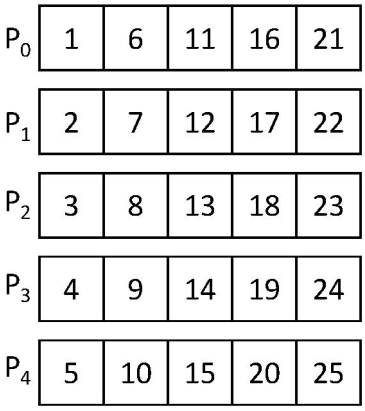
Question 16 – 19. The processes in an MPI program have data that looks Figure (a) below. For Questions 16 through 19 give the name (you do not need to give the full function call or arguments) of the collective communication operation that leads to the data being communicated as shown for the corresponding question.



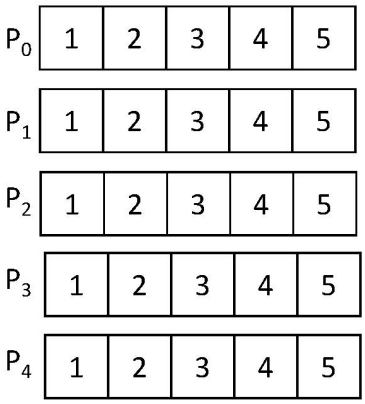
(a)



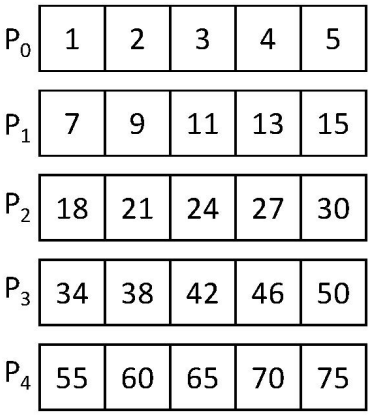
Question 16



Question 17



Question 18



Question 19

16. AllReduce
17. AlltoAll
18. Bcast
19. Scan