ECE 563 Second Midterm Spring 2018

This exam is open notes, open books. No electronics are allowed except for computers used to take this exam at a remote testing location.

Calculations only need to be carried out to two significant digits. Show your work. For problems that you only need to calculate and answer but not draw conclusions, you can set the problem up and not do the arithmetic.

Name (printed):

Signed:

By signing you are stating that you have neither given nor received help from others during this exam, and that you understand that if you have given or received help from others you will receive a grade of zero on the exam. In any problem needing calculations, answers only need to be compute to two digits.

Question 1.

You are working at MegaSim, a company that does computer simulations. You currently execute a program that is 95% parallel on a machine with 100 cores, and execute it using a 100 process MPI program. Your manager would like to buy a machine that has 1000 cores, and will have 1000 MPI processes executing.

(a) What is the calculated speedup on the 100 core machine and the 1000 core machine, running the same problem size on both machines?

S_{100} = 1/(f + (1-f)/P) = 1/(.05 + .95/100) = 1/(.05 + .0095) – 1/.06 ~= 16

S_{1000} = 1/(.95 + .95 / 1000) = 1/(.05 + .00095) = 1/.051 ~= 19

(b) What is the efficiency on a 100 core machine and the 100 core machine?

E_{100} = 16/100 – 16%

E_{1000} = 19/1000 = 1.9%

(c) Does buying the 1000 core machine to run your program cost effective assuming it costs 10 times what the 100 node machine cost?

The 1000 core machines is 10X more expensive and provides ~ 1.16X better performance. It is not cost effective.

(d) You benchmark the program on the machine. Is the speedup obtained from actually running the program more or less than predicted by Amdahl's law? Explain why.

Less, because Amdahl's Law does not take into account communication overhead.

Question 2.

After the evaluation in Question 1, your manager tells you that the size of the problems to be solved by your program will increase by large amounts. With larger problem sizes, your program is still 95% parallel.

(a) Will the speedup of your program executing on the hardware increase or decrease?

With larger problems sizes, the speedup should increase.

(b) You determine that the *fraction of a parallel execution that is sequential* is 0.05. What is the predicted speedup on the 1000 core machine in Question 1?

S_{1000} = P + (1-P) s = 1000 + (1-1000) * .05 = 1000 + -999 * .05 ~= 950.05

(c) Again, you actually benchmark your program on a 1000 node machine, and again its speedup differs from the calculated speedup of (b). Is it more or less, and why?

Less, because like Amdahl's Law, communication and other parallel overheads are not taken into account.

Question 3 (Karp Flatt)

(a) A program is run on 4 processors with a speedup of 3, 10 processors with a speedup of 3.3, and 100 processors for a speedup of 4. What is *e* for 4, 10 and 100 processors?

E_4 = (1/S – 1/P) / (1 – 1/P) = (.33 - .25)/(1 - .25) = .08 / .75 = 0.106

e_{10} = (.3 - .1) / (1 - .1) = .2 / .9 = .22

e_{100} = (.25 - .01) / (1 - .01) = .24 / .99 = 0.24

(b) Is the parallel overhead increasing, decreasing or staying the same with increasing numbers of processors? What are the implications of this for scaling to even larger machines?

It is increasing. Efficiency will decrease increasingly rapidly as we move to larger machines.

Question 4. A program has an overhead of nP^2, where P is the number of processors. The work $W = n^4$, i.e., n elements requires n^4 time steps to compute in a sequential program.

(a) What is T_P, the parallel time on P processors?

$T_P = n^4 / P + n P^2$

(b) What is $P\ T_P$, the total time P processors are occupied in a parallel execution?

$P\ T_P = n^4 - n P^3$

(c) What is $P\ T_P - T_O$? **(this was a typo, and should have been $P\ T_P - T_1$. Below is the answer if there was no typo. I gave credit to everyone for this.)**

$P\ T_P - T_O = n^4 - n P^3 - n^4 = n P^3$

(d) Show the result of setting $W = P\ T_P - T_O$ and eliminating any n terms from the right hand side.

$W = n^4 = n P^3$; $n^3 = P^3$

(e) What is the isoefficiency relationship resulting from (d)?

We need to get the left hand side to W, or n^4, so we raise both sides to the 4/3 power. $n^4 = P^4$

(f) Let E be the efficiency when $n=2$, $P=4$. Given your answer to (e), what should the work $W = n^4$ be when $P=8$ to maintain the same efficiency?

Work is $2^4 = 16$, initial overhead is P^4 or 256. When P is doubled, the overhead increases by a factor of 16 (e.g., 8^4 is 16 times larger than 4^4). Thus the work needs to increase by a factor of 16.

(g) Given your answer to (f), What should n be with P=8, i.e., how large is the data?

For W to increase by a factor if 16, n needs to increase by $16^{1/4}$, or 2. Thus n doubles, leading to a 16X increase in work.

Question 5.

(a) Circle all that are overheads when executing a program on a GPU:

(I) Cache misses on the host processors
(ii) Transferring data from the host to the GPU
(iii) Transferring data from the GPU to the host
(iv) Loads from global memory
(v) Control divergence
(vi) warp scheduling

(b) True or **false:** All if statements in a GPU program lead to control divergence.
(c) **True** or false: Different threads in a block can communicate via *shared* memory.
(d) **True** or false: Threads in different blocks can communicate via *shared* memory.
(e) What the benefit of *memory coalescing*? **It allows all fetched data to be used, increasing memory bandwidth.**

(f) Given a program with limited parallelism (e.g., 64 way parallelism) would a GPU or a multi-core with vector units be the best target to execute this in terms of utilization of the processor? **A multicore, not enough parallelism to get sufficient performance from a GPU.**

Question 5, continued.

(g) Answer each of the following:

(I) Given a 24 x 24block of threads, how many warps will there be in the block?

24^2 / 32 = 576 / 32 = 18

(I) If an SM (streaming multiprocessor) can take up to 1536 threads, how many of these blocks can be scheduled on the SM

1536 / 576 = 2

(iii) Which would more fully occupy the SM, 8 x 8 blocks of threads, 16 x 16 blocks of threads, or 24 x 24 blocks of threads? Assume a maximum of 8 blocks can be scheduled to the SM. From this alone, what would be the best block size?

8x8: 1536 / 64 = 24, 64 * 8 = 512, 1024 of the 1536 threads not utilized.
16 x 16: 1536 / 256 = 6, 256 * 6 = 1536, fully utilized – the best.
24 x 24: 1536 / 576 = 2, 576 * 2 = 384 of the 1536 threads not utilized.

(h) Building on question (h): Each 8 x 8 block uses 2K of shared memory. Each 16 x 16 block uses 8K of memory. Each 24 x 24 block uses 18K of memory. The shared memory on this GPU has 24K of memory. Show which block size(s) best utilizes the available threads in a GPU with this additional constraint.

8 of the 8 x 8 blocks can execute at a time in an SM with this amount of memory. 64 x 8 = 512 threads utilized.

3 of the 16 x 16 blocks can execute at a time in an SM with this amount of memory. 256 x 3 = 768 threads.

1 of the 24 x 24 blocks can execute at a time in an SM with this amount of memory.. 24 x 24 is 576 threads.

16x 16 best utilize the GPU given the constraints of (i) and (h). Note that algorithmic considerations might make one or the other of these better, e.g., with matrix multiply, larger blocks are better because the increase data reuse, and thus 16 x 16 blocks would be preferred.