Name: _____

Directions: **Work only on this sheet** (on both sides, if needed). MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

Important note: Remember that in problems calling for R code, you are allowed to use any built-in R function, e.g. **choose()**, **sum()**, **combn()** etc.

IMPORTANT NOTE: All questions refer to CUDA/NVIDIA GPUs.

1. (15) Fill in the blank: Having a lot of threads helps achieve _____ hiding.

2. (15) Suppose the variable **n** is our problem size, and we wish to check whether the total number of threads evenly divides this quantity. Write one line of code, to be run on the device, that sets the **bool** variable **evenlydivides** (assumed previously declared) to **true** if this condition holds, **false** otherwise. Assume we only use the "x dimension" in grids and blocks.

3. Consider the mutual-outlinks problem in Section 5.8, but changed as follow:

- Line 14 is now

```
i = me;
```

- Line 19 is now blank.
- Variables such as **totth** will now be ignored. (They would be removed, but let's keep the specs simple here.)

Answer the questions below. Assume the program won't ever be run with any partially-filled blocks; the total number of threads will be evenly divisible by the number of blocks.

(a) (20) One more line in the program would need to be changed. State which one, and what the new version of the line would look line. **Note! Put your answer on just ONE line in your submitted electronic file.** Sample answer line:

```
change line 45 to:  i = 168;
```

(b) (15) Which line, already a drain on execution speed, would become even more of a drain, and why?

4. (20) Consider static versus dynamic scheduling of loop operations. Our CUDA examples, such as Line 14 in the mutual-outlink example (original version, not changed as in Problem 3), have used static scheduling. Explain in one line why dynamic scheduling generally would not be a good choice for our GPU programming.

5. (15) Consider the primes-finding code in Section 5.9, specifically the function **sieve()**. This question concerns the issue of possible bank conflicts between thread 0 and thread 1. Fill in the blank with a mathematical condition involving **chunk**: There will be no bank conflicts as long as _____.

Solutions:

1. latency

2.

```
evenlydivides = (n % (gridDim.x * blockDim.x) == 0);
```

3.a

```
change line 50 to:  dim3 dimGrid(n/192,1);
```

3.b Line 20. It would be executed more often, thus more of a drain.

4. It would be difficult to implement dynamic scheduling without inducing a large degree of thread divergence. Also, dynamic scheduling requires atomic access to a work queue, which is slow on our GPUs.

5. In Line 74 the two threads will be simultaneously accessing items that are **chunks** words apart in memory. Those will be in separate banks as long as **chunk % 32** is not 0.