Hints for debugging GPU programs

- The CUDA calls return error status. Remember to check the error status and report any errors after every call – this will help you localize where an error occurred. The kernel call itself does not return error status; you nonetheless should check for errors on this too (as shown in the skeleton code for the labs).
- Use *printf*() statements. CUDA does support *printf*(). Note that printing is done from the CPU rather than the GPU; thus, CUDA buffers the *printf*() output and returns it to the CPU for printing. CUDA does not directly support C++ printing with **cout**. Note we've given you a simple function *print_array*() that may be helpful.
- Some GPUs are quite slow at double-precision arithmetic. Be sure to use **float** rather than **double**.
- CUDA has a memory checker that will catch most accesses to illegal memory. Turning it on can be very helpful. For each memory error, it will tell you whether the access was a read or write, and whether it involved GPU DRAM, shared memory or both. This detail is usually a pretty good clue as to where in your code the problem is. In principle, you can use the memory checker by, instead of running *./a.out*, running *cuda-memcheck a.out*. However, it turns out that *cuda-memcheck* was recently retired and replaced by a more powerful (and complicated) tool called the compute sanitizer. While we're figuring out how to use it, the simplest solution is just to revert back to an earlier version of CUDA. So you *run.sh* would now read

```
source /etc/profile.d/modules.sh
module load cuda/11.0
module load gcc/11.2.0
cuda-memcheck ./a.out > results.out
```

- If your program runs way too slowly, the job manager might evict it for exceeding its CPU allocation. This might be because of your code. However, it is more often simply because you've forgotten to compile with **-O2**.