Q1:

Python does not natively support parallel processing. This is because the python interpreter uses a “Global Interpreter Lock” which restricts the ability for Python to process multiple threads simultaneously, even in an environment with multiple processers. Additionally, many Python functions are dependent upon the GIL. Because of this, it would be very difficult and laborious for a programmer to write a parallel program, and bypass the GIL (otherwise the GIL would bottleneck the threads, and at the point, one might as well write sequential programs.) However, their do exist Python Libraries which allow programmers to write and execute parallel threads within a process.

Q2: R, similarly, does not natively support parallelism. R, does not by default, access all availalble processors, nor does it calculate in thread. In native R, however, it is easy to run functions which may seem parallel, using the “Apply” family of functions. Essentially, a programmer, using “apply”, “lapply”, or “sapply”, can apply a specific function to a structured set of data, without using a “for each” loop. However, this is not truly parallel, as each computation still occurs within one processor.

Q3: Python does have available libraries which support parallelism. On Python.org, the library, in chapter 17, references two available libraries; threading.py and multithreading. Threading.py, however does not bypass the GIL, so while it allows you to write a parallel program, logically, it does not truly operate in parallel and users will not see the result. However, the multithreading process allows for concurrent access and updating of the data using a “concurrency” module, and bypasses the GIL. This allows for a truly parallel program.

Q4: Similar to Python, while R does not natively support parallelism, it does have packages which will allow a programmer to fully use all available cores on their machine. These libraries include doMC, doSnow, and doParalell. With these libraries, R has many ways to execute parallel code, such as the aforementioned Lapply, Parlapply, and “for-each” style parallelism.

Q5: Using GPU’s allows users to take advantage of parallel programs, as GPU’s often have many available processors. For python, pyCuda is a library which has API’s and kernals available for using CUDA to access the GPU with a python code. Additionally, CUDA can be used in conjunction with R to access the GPU processors.