Disk Benchmark:

The disk benchmark program operates by reading or writing a constant stream of bytes to or from the disk with a specified block size. The writing is done to a generic file on the disk called testfile and uses the fwrite function. It also reduces computational overhead by utilizing the buffer parameter of the write call. The writing operation benchmark was tested against the dd program in order to verify that the program is within a reasonable range of the expected result.

For the read test, the program reads from a file of a specific size generated by the program and then reads it using the fread function and specifying the buffer size.

For both tests we open the file in binary mode, this eliminates the need to initialize the data prior to writing. Also, we malloc enough memory in order to read or write the entire file without loops. While this puts a limit on the max size of the test we can run, it provides a more pure measurement. Also, we don't need to run tests were the file to read would be larger than the available memory as that shouldn't happen in practice. Finally, even with a physical upper bound we can still run large enough tests to get valuable results.

Network Benchmark:

The network benchmark is in part based off of iperf in terms of design and structure. Run without specifying server or client mode will perform the benchmark with separate processes over the localhost. However, in order to benchmark the actual network connection the program will be run on the server in server mode, then on the client machine, the program will initiate a connection to the server and proceed to write data over the network. The program can be run in either TCP or UDP mode to test the transfer rates of either protocol. It also performs a pretest to determine the network latency of the connection in a similar manner to the ping command.

The program has the ability to write with a specified buffer size in order to test ideal settings of the link. Also, two forms of tests exist in the program. The default is to transfer as much data as possible in a certain amount of time (default 10 seconds) and measure the data transmitted. Alternatively a specific amount may be entered and the program will time the duration of the transfer of the set amount of data.

We use loops for the writing as the overhead of the loop is negligible compared to the network latencies and thus won't be a significant influence of deviation in the results. The results were also compared against results from iperf in order to establish that the application was measuring correctly and getting reasonable results.

CPU Benchmark:

This benchmark's structure is very simple. It uses a separate set of utility functions to create the desired number of threads, and feed each thread an entry function to execute. This test have two entry functions, they are just summation over a set of integers and floats, respectively. After the threads terminate, it use their durations as basis for the calculation of GFLOPS/GIOPS. The summation is done up to ten million in order to get a reasonable result.

Memory Benchmark:

This benchmark incorporates three different tests while varying block sizes and thread numbers. The first one tests the efficiency of memcpy function, By using 16 blocks, it circularly copy the current block's contents into the next block (if current block is 15 then the next block is 0), this process is done ten thousand times. The second one tests sequential access memory, it simply sequentially accesses every block, this process is done ten thousand times. The third one tests random access memory, within the range of 32 blocks, it randomly chose every next one so cache is not possible, again this process is done ten thousand times. After the tests the program collects the number of bytes processed and the duration and outputs the throughput and the latency.

GPU Benchmark:

The gpu benchmark tests the speed of arithmetic operations on the gpu as well as the memory speed. This means that it is broken into two parts. The first part is calculating the number of IOPs and FLOPs the GPU can perform. To do this, we initialize 2 arrays full of integers for iops and doubles for flops. Then we have a gpu kernel routine to sum the arrays. We do this summation of 1000 elements 1000000 times to make sure to saturate the gpu cores and amortize out the idle time while the cores are being populated with work as well as when they start completing. For the memory benchmark, we allocate space on the host and the device. Then at various block sizes we do about 1000000 memcpys, from the host to the device, device to host, and device to device to test on chip memory speed. By doing lots of memcpys, we amortize out the time taken in the loop to get a good measurement of how much throughput the gpu memory has.