Hardware:
Hardware specifications:
GTX 745 NVIDIA
GPU name: GM107
CUDA Cores - 384
Base Clock speed - 1033
Memory Clock - 1.8 Gbps DDR3
Memory Size: 4 GB
PCI Express 3.0

Report:

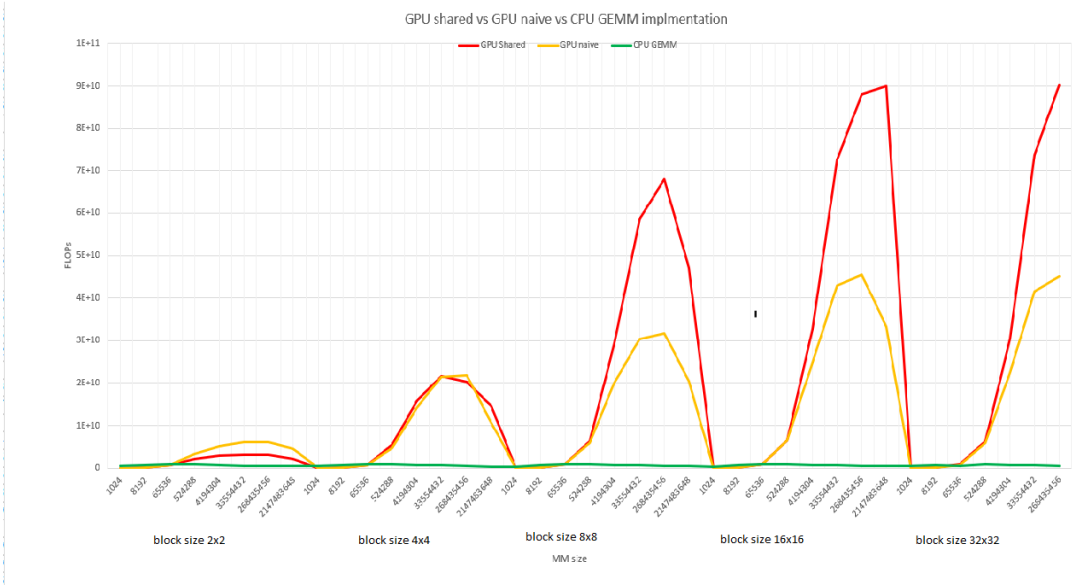
Work was done on Windows and Linux machines for testing, but all data reported on was collected using the Linux labs. I used the GPUs to be able to parallelize the matrix multiplication. I used 3 functions to do the process of data collection, CPU implementation, GPU shared, and GPU naive. The combination of threads and blocks was calculated with respect to the size of the problem space, which increments in powers of 2.

Once I have allocated the space of the array, values were initialized with random values for comparison purposes. The allocations of host and device were made, and after the clock would be recorded using Cuda's function of cudaEvent_t for the calculation of memoryCopy and the execution of the kernel.

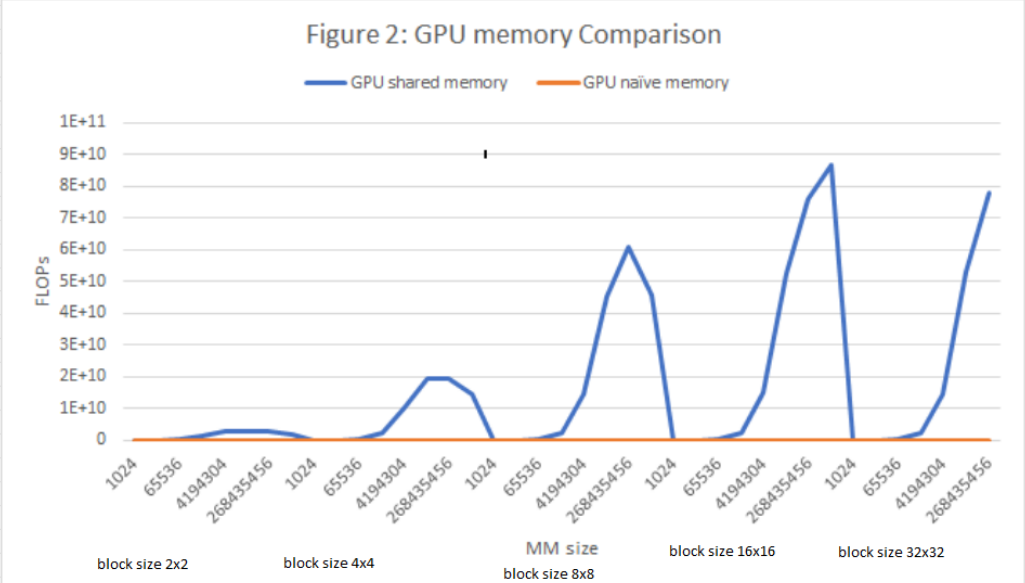
I compiled the program using nvcc compiler using the command: nvcc --gpu-architecture=sm_50 -std=c++11 <filename>.cu . I also used scripts for testing, but hardcoded values for ease of use. The one boundary case to be aware of is whenever a value is larger than the hardware arch, as well as when the size is not equal to a power of 2, which the values may be off. I tried to negate this by accounting for it with the execution configuration, but there could be some occurrences of errors.

I was trying to calculate the residual using BLAS but I was running into errors pertaining the library and the operation of compiling it with the Linux lab, so I tried multiple options, but went with using an iteration of my GEMM to be able to get a host completion of a matrix multiplication configuration. In order to calculate the residual, I collected the sum of the possible error that could occur between the device and host versions and averaged them by the size. The condition is met if the size is squared for the majority of properties.

Figure 1:



My first graph shows the relationship between the shared blocked, naive, and CPU implementations and their speeds. It is interesting to see how the allocation in memory and how they are affected by the increases in size and the importance of locality affects the FLOP of the dot product. I believe that the reason behind getting these results has to do with the way memory is allocated and is specified in the execution call. When implementing the shared blocked memory version, I chose to have a precondition that makes the size of the arrays equal to a dimension that is of base 2. This is to stay in its squared dimension which is a formal makeup of the hardware architecture that I am using for the assignment. Understand that there is an early trend that shows closer values as the size of the block and the size of the array is smaller than the naive and blocked version are similar. I believe this to be so, because of the low movement of data since there is not a large amount of allocated data being used to compute the information. The communication bandwidth is lowered but as the effective size gets closer to infinity, the values of the FLOP meet a higher differentiation in comparison. This can also be seen through the memory comparison in Figure 2 which shows this dynamic as the size of the gets bigger.



This shows the contrast in speed with the comparison to the On-chip location of the shared memory, and the off-chip operation, which is done on the global scope provide the visualization that as the block size increase so does the intensity of the GPU for the shared version of the computation.

OUTPUT:

```
Size of M: 8 ,Size of N: 8 , Size of K: 8 , Milliseconds for Memory recorded: 0.146464 ,
matrix size 8x8 , threads 2x2  GPU
Size of M: 8 ,Size of N: 8 , Size of K: 8 , Milliseconds for Memory recorded: 0.045216 ,
matrix size 8x8 , threads 2x2  GPU
Size of M: 8 ,Size of N: 8 , Size of K: 8 , Milliseconds for Memory recorded: 0.001024 ,
matrix size 8x8 , threads 2x2  GPU
CORRECT
average of residual:
0

C:\Users\Samuel Debesai\source\repos\gputest\x64\Debug\gputest.exe (process 6096) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
Size of M: 512 ,Size of N: 512 , Size of K: 512 , Milliseconds for Memory recorded: 14.408704 ,
matrix size 512x512 , threads 16x16  GPU
Size of M: 512 ,Size of N: 512 , Size of K: 512 , Milliseconds for Memory recorded: 14.051392 ,
matrix size 512x512 , threads 16x16  GPU
Size of M: 512 ,Size of N: 512 , Size of K: 512 , Milliseconds for Memory recorded: 0.001024 ,
matrix size 512x512 , threads 16x16  GPU
CORRECT
average of residual:
0

C:\Users\Samuel Debesai\source\repos\gputest\x64\Debug\gputest.exe (process 18712) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
Size of M: 32 ,Size of N: 32 , Size of K: 32 , Milliseconds for Memory recorded: 0.197760 ,
matrix size 32x32 , threads 16x16  GPU
Size of M: 32 ,Size of N: 32 , Size of K: 32 , Milliseconds for Memory recorded: 0.079488 ,
matrix size 32x32 , threads 16x16  GPU
Size of M: 32 ,Size of N: 32 , Size of K: 32 , Milliseconds for Memory recorded: 0.001024 ,
matrix size 32x32 , threads 16x16  GPU
CORRECT
average of residual:
0

C:\Users\Samuel Debesai\source\repos\gputest\x64\Debug\gputest.exe (process 16444) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

DATA:

Block size	size	GPU(n) time	GPU(s) time	CPU time	GPU(n) FLOP	GPU(s) FLOP	CPU FLOP	GPU(s) MEM ms	GPU(s) MEM time	GPU(n) MEM ms	GPU(n) MEM time	GPU(s) mem FLOP	GPU(n) mem FLOP
4	1024	0.000085184	0.000083616	0.000002176	12021036.81	12246460.01	470588235.3	0.202912	0.000202912	0.171552	0.000171552	5046522.6	8.36E-05
4	8192	0.000079392	0.000070304	0.000010144	103184199.9	116522530.7	807570977.9	0.214912	0.000214912	0.176096	0.000176096	38117927	7.03E-05
4	65536	0.000097568	0.000086976	0.000073408	671695637.9	753495217.1	892763731.5	0.294176	0.000294176	0.190176	0.000190176	222778201	8.70E-05
4	524288	0.000246016	0.00016304	0.000576192	2131113424	3215701668	909918915.9	0.397184	0.000397184	0.26832	0.00026832	1.32E+09	0.0001630 4
4	419430 4	0.001428672	0.000814464	0.005282592	2935806119	5149772120	793985982.6	1.581824	0.001581824	0.921472	0.000921472	2.65E+09	0.0008144 6
4	33554432	0.010700608	0.005499808	0.053315742	3135750043	6101018799	629353184.3	10.86928	0.01086928	5.640864	0.005640864	3.09E+09	0.0054998 1
4	268435456	0.086442627	0.043777023	0.5107005	3105359767	6131880096	525622074	87.078178	0.087078178	44.225697	0.044225697	3.08E+09	0.0437770 2
4	2147483648	1.062792114	0.484636505	4.838290527	2020605554	4431122348	443851735.7	1063.367432	1.063367432	488.251343	0.488251343	2.02E+09	0.4846365 1

16	1024	0.000083488	0.000071392	0.000002048	12265235.72	14343343.79	500000000	0.20576	0.00020576	0.171456	0.000171456	4976671.9	7.14E-05
16	8192	0.000072128	0.000072064	0.00001024	113575865.1	113676731.8	800000000	0.201376	0.000201376	0.168512	0.000168512	40680121	7.21E-05
16	65536	0.000080896	0.000079712	0.000073248	810126582.3	822159775.2	894713848.8	0.21568	0.00021568	0.179776	0.000179776	303857567	7.97E-05
16	524288	0.000097152	0.000111136	0.000575936	5396574440	4717535272	910323369.3	0.235648	0.000235648	0.203776	0.000203776	2.23E+09	0.00011114
16	4194304	0.000265696	0.0002952	0.005293152	15786101409	14208346883	792401956.3	0.417152	0.000417152	0.402048	0.000402048	1.01E+10	0.0002952
16	33554432	0.001550368	0.001569888	0.051183617	21642882206	213737744435	655569769.5	1.740992	0.001740992	1.696736	0.001696736	1.93E+10	0.00156989
16	268435456	0.01325152	0.012321728	0.510722046	20256955881	217855365774	525599899.4	13.827584	0.013827584	12.790976	0.012790976	1.94E+10	0.01232173
16	2147483648	0.147692505	0.200769913	5.010881348	14540234442	106962423660	428564058.7	148.683655	0.148683655	201.727997	0.201727997	1.44E+10	0.20076991
64	1024	0.000103392	0.000069632	0.00000272	9904054.472	14705882.35	376470588.2	0.218048	0.000218048	0.169536	0.000169536	4696213.7	6.96E-05

64	8192	0.0000712	0.000071808	0.000010208	115056179.8	114081996.4	802507837	0.20336	0.00020336	0.168928	0.000168928	40283242	7.18E-05
64	65536	0.00007296	0.000072064	0.000073376	898245614	909413854.4	893153074.6	0.203648	0.000203648	0.171264	0.000171264	321810182	7.21E-05
64	524288	0.000083136	0.00009088	0.000572416	6306389530	5769014085	915921288	0.218304	0.000218304	0.1992	0.0001992	2.40E+09	9.09E-05
64	419404	0.00014256	0.00020864	0.005261024	29421324355	20103067485	797240993.4	0.283488	0.000283488	0.314592	0.000314592	1.48E+10	0.00020864
64	33554432	0.000572352	0.00110752	0.049633343	58625517164	30296908408	676046181.3	0.742368	0.000742368	1.252288	0.001252288	4.52E+10	0.00110752
64	26843456	0.003946016	0.008478976	0.503546631	68026955795	31658947496	533089568	4.400256	0.004400256	8.907488	0.008907488	6.10E+10	0.00847898
64	21474836 48	0.045621346	0.104847389	3.943130615	47071904630	20481994530	544613876	46.666882	0.046666882	106.073853	0.106073853	4.60E+10	0.10484739
256	1024	0.0000728	0.000071904	0.000004	14065934.07	14241210.5	256000000	0.265824	0.000265824	0.17216	0.00017216	3852172.9	7.19E-05
256	8192	0.000071104	0.00007136	0.000010176	115211521.2	114798206.3	805031446.5	0.208384	0.000208384	0.171872	0.000171872	39312039	7.14E-05

256	65536	0.000073632	0.000070848	0.000073216	890047805.3	925022583.6	895104895.1	0.20256	0.00020256	0.17024	0.00017024	323538705	7.08E-05
256	524288	0.00008208	0.0000824	0.000587936	6387524366	6362718447	891743319	0.219808	0.000219808	0.200032	0.000200032	2.39E+09	8.24E-05
256	4194304	0.000128064	0.00016912	0.005278816	32751624188	24800756859	794553930.3	0.28176	0.00028176	0.273216	0.000273216	1.49E+10	0.00016912
256	33554432	0.000461984	0.00078176	0.051409664	72631156057	42921653704	652687245.7	0.635808	0.000635808	0.924	0.000924	5.28E+10	0.0007817 6
256	268435456	0.003050688	0.005888704	0.519179199	87991776281	45584810512	517038156.6	3.525824	0.003525824	6.312032	0.006312032	7.61E+10	0.0058887
256	2147483648	0.023868065	0.064636383	4.07297876	89973093671	33224068989	527251374.1	24.788544	0.024788544	65.515907	0.065515907	8.66E+10	0.0646363 8
1024	1024	0.000085728	0.000081792	0.000001984	11944755.51	12519561.82	516129032.3	0.218464	0.000218464	0.16928	0.00016928	4687271.1	8.18E-05
1024	8192	0.000072416	0.000069152	0.000010112	113124171.5	118463674.2	810126582.3	0.202784	0.000202784	0.1688	0.0001688	40397665	6.92E-05
1024	65536	0.000072224	0.000080032	0.00010512	907399202.5	818872451	623439878.2	0.203872	0.000203872	0.178528	0.000178528	321456600	8.00E-05

1024	524288	0.00008304	0.000090432	0.000600896	6313680154	5797593772	872510384.5	0.223968	0.000223968	0.195296	0.000195296	2.34E+09	9.04E-05
1024	419404	0.00013856	0.000187456	0.00526752	30270669746	22374871970	796257821.5	0.282944	0.000282944	0.292832	0.000292832	1.48E+10	0.0001874 6
1024	33554432	0.000454016	0.0008056	0.052437824	73905835918	41651479643	639889862.7	0.629312	0.000629312	0.9448	0.0009448	5.33E+10	0.0008056
1024	268435456	0.002977664	0.005933184	0.510623901	90149679749	45243069488	525700922.9	3.43008	0.00343008	6.283712	0.006283712	7.83E+10	0.0059331 8