Project 6

Brandon Lee

# Machine Specifications

I ran this project on rabbit through my 2015 MacBook Pro.

Multiply Table and Graphs

\*\*\*The values in the following tables of this document are represented in MegaMultiplies Per Second, MegaMultiply-Adds Per Second, and MegaMultiply-Reductions Per Second respectively.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Multiply Table | 8 Work Items | 32 Work Items | 64 Work Items | 128 Work Items | 256 Work Items | 512 Work Items |
| 1024 Array Elements | 29.512 | 43.748 | 59.054 | 60.66 | 42.088 | 48.29 |
| 10240 Array Elements | 324.616 | 435.986 | 443.866 | 354.915 | 393.543 | 459.605 |
| 81920 Array Elements | 1788.373 | 3017.756 | 3330.487 | 4657.986 | 3105.383 | 3692.752 |
| 655360 Array Elements | 2742.884 | 7848.247 | 11057.196 | 12387.955 | 12382.572 | 12230.973 |
| 1536000 Array Elements | 2921.257 | 9693.42 | 14338.925 | 15257.621 | 15395.718 | 13081.135 |
| 4096000 Array Elements | 2994.251 | 11149.978 | 16141.49 | 18150.718 | 17938.477 | 17409.415 |
| 5120000 Array Elements | 2992.953 | 10472.896 | 16546.126 | 18315.018 | 17052.172 | 16725.467 |
| 8000000 Array Elements | 3499.357 | 11650.48 | 16292.051 | 17887.848 | 17195.204 | 17105.092 |

Multiply-Add Table and Graphs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Multiply-Add Table | 8 Work Items | 32 Work Items | 64 Work Items | 128 Work Items | 256 Work Items | 512 Work Items |
| 1024 Array Elements | 45.07 | 55.589 | 55.033 | 44.013 | 55.883 | 41.279 |
| 10240 Array Elements | 389.25 | 473.088 | 321.144 | 461.095 | 302.02 | 445.721 |
| 81920 Array Elements | 1601.752 | 3320.632 | 2987.927 | 2920.812 | 3220.126 | 3032.502 |
| 655360 Array Elements | 2645.653 | 7699.339 | 9763.859 | 9855.927 | 9737.89 | 9527.797 |
| 1536000 Array Elements | 2824.345 | 8553.577 | 11264.466 | 12024.425 | 12202.582 | 11339.559 |
| 4096000 Array Elements | 2895.86 | 9684.52 | 12737.309 | 13913.752 | 13829.802 | 13404.94 |
| 5120000 Array Elements | 2900.953 | 9373.461 | 12819.486 | 13260.538 | 14047.333 | 13959.36 |
| 8000000 Array Elements | 3375.905 | 9888.678 | 12617.023 | 13795.553 | 13747.901 | 14319.544 |

Multiply and Multiply-Add Commentary

What patterns are you seeing in the performance curves?

*Multiply*

I’m noticing that in the Multiply graphs, specifically in terms of performance versus local work size, as we increase the array size, we observe that as we increase the input array size, the greater the number of work items in a work group, the better the performance. The performance difference between each of these work group sizes starts out negligible, but quickly increases followed by a plateau of performance.

When observing performance versus global work size, as we increase the number of items per work group, the greater the array size (global work size), the better the performance. We see that this increase in difference starts out minuscule, but as the global work size gets bigger, we see an exponential increase for certain sizes, followed by a plateau in performance.

*Multiply-Add*

In terms of performance versus local work size, I’m seeing that the trend follows a very similar path as performance versus local work size for multiply. We observe that as the greater the number of work items per work group, as the array size increases, the performance increases as well, followed by plateau. One notable difference between this pattern and Multiply’s however, is that the margins of performance between the different local work sizes are much smaller than in Multiply’s local work sizes.

Regarding performance versus array size, we observe a similar pattern as well. The overall pattern between Multiply and Multiply-Add remain very similar, with a smaller margin of performance difference between the global work sizes of Multiply-Add versus Multiply.

Why do you think the patterns look this way?

*Local Work Size Graphs*

As an explanation for why smaller local work sizes appear to have lower performance, some of these work group sizes are so ridiculously small that it is difficult to yield a good performance as there are more processing units than 8, which results in wasted performance. We see that as we increase the work group size to 32 and above, we utilize more of the processing units, which gives us better performance in return.

*Global Work Size Graphs*

There’s definitely an observable sweet spot of work group sizes. However, the performance differences in the graphs are more observable in the greater data set sizes as there isn’t really enough work to get done in order to effectively measure performance. Array sizes of under 100,000 and under seem to not perform nearly as well as array sizes greater than 100,000. This can be due to the lack of a large enough data set to justify all the overhead of setting this parallelism up.

What is the performance difference between doing a Multiply and doing a Multiply-Add?

I don’t think that there’s too much of a performance difference between multiply and multiply-add. If we’re being picky, we can observe that the margin of performance difference between multiply-add is a bit smaller. Specifically, we can see that a work group size of 32 in multiply-add is closer to the work groups above it than in the multiply graph. This may be due to the fact that we do a bit more processing in multiply-add than in just multiply.

What does that mean for the proper use of GPU parallel computing?

This means that if we want to effectively utilize proper GPU parallel computing, we must acknowledge that the needs to be a sufficient data set to utilize our parallel computing with, along with a sufficient work group size. This is to ensure that we utilize all the processing units we can so that we don’t bottleneck ourselves. Additionally, having enough work to do can also potentially impact our performance when considering these parameters.

Reduction Tables and Graphs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reduction Table | 32 Work Items | 64 Work Items | 128 Work Items | 256 Work Items |
| 1024 Array Elements | 16.064 | 13.565 | 17.994 | 15.909 |
| 10240 Array Elements | 164.163 | 149.99 | 139.166 | 250.122 |
| 81920 Array Elements | 931.852 | 1134.312 | 1215.556 | 1514.037 |
| 655360 Array Elements | 4030.727 | 6946.126 | 7293.285 | 6480.949 |
| 1536000 Array Elements | 4686.942 | 7062.101 | 8865.084 | 8089.66 |
| 4096000 Array Elements | 4650.719 | 7592.679 | 10319.304 | 9378.943 |
| 5120000 Array Elements | 4788.354 | 7737.201 | 11123.06 | 9576.824 |
| 8000000 Array Elements | 5270.623 | 9334.214 | 13606.669 | 11707.426 |

Reduction Commentary

What pattern are you seeing in this performance curve?

I’m observing that there appears to be a rapid performance increase at around array size 655360. This is followed by a slow but steady increase as the array size gets larger. Interestingly, the 128 work group size appears to be more effective in performance over 256.

Why do you think the pattern looks this way?

I think the pattern that performance rapidly jumps then plateaus is a result of the data set size not being adequately large enough in the smaller data sets to compensate for all of the overhead. However, as we give larger and larger array sizes to process, we observe that these work groups begin to be more effectively utilized, resulting in an increase in performance.

In terms of local work size 128 being the best performer, I’m not quite sure why this is the sweet spot over 256 local work size. However, a possible explanation for this is due to the fact that this is a very parallelizable problem. As such as we increase the work group size past a certain threshold, the number of processing units being effectively utilized tend to reach a certain optimal point.

What does that mean for the proper use of GPU parallel computing?

From the reduction results, we observe that the larger the data set, the better performance we get out of parallelizing the problem. This trend also shows that greater local work group sizes *usually* tend to yield better performance, but not quite always. All in all, this means that for proper use of GPU parallel programming, one must consider how large their data set is along with how large their local work group is as well if they want the most optimal results.