**Project #6**

**OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce**

**CHI CHIEH WENG**

[**wengchic@oregonstate.edu**](mailto:wengchic@oregonstate.edu)

**Machine:** DGX Systems

**Array Multiply**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Global Dataset Size** | **Local Work Size** | | | | | | | |
|  | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| 1024 | 0.021 | 0.023 | 0.023 | 0.023 | 0.022 | 0.018 | 0.024 |
| 2048 | 0.046 | 0.036 | 0.046 | 0.046 | 0.044 | 0.045 | 0.045 |
| 4096 | 0.092 | 0.07 | 0.093 | 0.092 | 0.091 | 0.092 | 0.089 |
| 8192 | 0.115 | 0.176 | 0.184 | 0.188 | 0.184 | 0.148 | 0.182 |
| 16384 | 0.342 | 0.355 | 0.281 | 0.375 | 0.365 | 0.292 | 0.374 |
| 32768 | 0.651 | 0.704 | 0.701 | 0.569 | 0.571 | 0.564 | 0.716 |
| 65536 | 1.105 | 1.27 | 1.35 | 1.359 | 1.415 | 1.111 | 1.453 |
| 131072 | 1.834 | 2.047 | 2.293 | 2.186 | 2.799 | 2.879 | 2.878 |
| 262144 | 1.116 | 1.28 | 1.354 | 1.411 | 1.424 | 1.079 | 1.437 |
| 524288 | 1.837 | 1.741 | 2.039 | 2.718 | 2.809 | 2.849 | 2.873 |
| 1048576 | 2.653 | 3.556 | 4.317 | 5.062 | 5.338 | 5.065 | 5.361 |
| 2097152 | 3.372 | 5.094 | 6.91 | 6.927 | 9.509 | 9.394 | 8.862 |
| 4194304 | 3.629 | 6.375 | 9.24 | 7.707 | 13.373 | 14.652 | 11.998 |
| 8388608 | 4.175 | 7.384 | 11.93 | 16.997 | 21.145 | 20.007 | 21.188 |

**Max Performance:** 21.188 GigaMultiplications/Second with Global Dataset Size of 8,388,608 and Local Work Size of 512

**Array Multiply – Add**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Global Dataset Size** | **Local Work Size** | | | | | | | |
|  | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| 1024 | 0.021 | 0.023 | 0.023 | 0.023 | 0.023 | 0.018 | 0.023 |
| 2048 | 0.035 | 0.044 | 0.044 | 0.035 | 0.035 | 0.044 | 0.035 |
| 4096 | 0.09 | 0.088 | 0.09 | 0.092 | 0.089 | 0.089 | 0.091 |
| 8192 | 0.173 | 0.179 | 0.178 | 0.18 | 0.178 | 0.177 | 0.177 |
| 16384 | 0.338 | 0.351 | 0.298 | 0.362 | 0.289 | 0.364 | 0.368 |
| 32768 | 0.638 | 0.669 | 0.68 | 0.573 | 0.713 | 0.707 | 0.715 |
| 65536 | 0.925 | 1.259 | 1.083 | 1.371 | 1.158 | 1.388 | 1.442 |
| 131072 | 1.804 | 2.206 | 2.076 | 2.146 | 2.8 | 2.877 | 2.902 |
| 262144 | 2.383 | 3.57 | 4.484 | 5.105 | 5.367 | 5.436 | 4.327 |
| 524288 | 1.829 | 2.242 | 2.499 | 2.682 | 2.785 | 2.731 | 2.789 |
| 1048576 | 2.627 | 3.629 | 4.501 | 2.94 | 5.15 | 5.121 | 5.185 |
| 2097152 | 3.373 | 4.481 | 6.996 | 5.154 | 8.997 | 7.378 | 8.933 |
| 4194304 | 3.873 | 6.498 | 9.707 | 11.399 | 13.838 | 12.225 | 13.763 |
| 8388608 | 4.159 | 7.229 | 11.644 | 16.494 | 17.999 | 17.939 | 17.74 |

**Max Performance:** 17.999 GigaMultiplications/Second with Global Dataset Size of 8,388,608 and Local Work Size of 128

**What patterns are you seeing in the performance curves?**

In "Array Multiply" and "Array Multiply – Add", as the global workload and local workload increase, the performance will continue to improve.

**Why do you think the patterns look this way?**

First of all, when using GPU data parallelism, as the global workload increases, the performance will also improve. Secondly, usually the more threads in each workgroup, the higher the performance. If the number of threads is less than 32, the performance will be greatly reduced because the number of threads in the warp is 32. If 8 or 16 threads are used, many threads will remain idle.

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

The maximum performance for OpenCL Multiply was 21.188 MegaMultiplies/Second with global work size of 8,388,608 and local work size of 512. The maximum performance for OpenCL Multiply-Add was 17.999 MegaMultiplies-Adds/Second with global work size of 8,388,608 and local work size of 128.

The performance of the multiply-add operation may be lower because the third array must be copied from the host to the device, and additional add operations must be performed.

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

From the figure, we can observe that to correctly use GPU parallel computing, only consider using a global workload of 2,097,152 or greater, and a benchmark with a local workload of 32 or greater to find a local work that can always provide the best performance. the amount.

**Multiply+Reduce application**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Global Dataset Size** | **Local Work Size** | | | | | | | |
|  | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| 1024 | 0.022 | 0.023 | 0.018 | 0.022 | 0.022 | 0.018 | 0.022 |
| 2048 | 0.044 | 0.041 | 0.045 | 0.045 | 0.046 | 0.044 | 0.045 |
| 4096 | 0.087 | 0.088 | 0.09 | 0.07 | 0.089 | 0.09 | 0.089 |
| 8192 | 0.172 | 0.177 | 0.182 | 0.139 | 0.176 | 0.178 | 0.18 |
| 16384 | 0.331 | 0.355 | 0.343 | 0.297 | 0.306 | 0.286 | 0.357 |
| 32768 | 0.488 | 0.686 | 0.717 | 0.704 | 0.574 | 0.702 | 0.733 |
| 65536 | 1.114 | 1.21 | 1.329 | 1.358 | 1.414 | 1.108 | 1.137 |
| 131072 | 1.751 | 2.154 | 2.491 | 2.627 | 2.805 | 2.847 | 2.708 |
| 262144 | 0.871 | 1.241 | 1.351 | 1.375 | 1.431 | 1.427 | 1.387 |
| 524288 | 1.819 | 2.253 | 2.52 | 2.685 | 2.634 | 2.81 | 1.35 |
| 1048576 | 2.653 | 3.656 | 4.286 | 5.067 | 4.212 | 5.287 | 4.196 |
| 2097152 | 3.376 | 4.574 | 6.22 | 8.607 | 9.063 | 7.758 | 9.526 |
| 4194304 | 3.815 | 6.331 | 9.619 | 12.406 | 14.764 | 14.858 | 14.802 |
| 8388608 | 4 | 7.354 | 11.821 | 17.058 | 20.784 | 21.165 | 19.003 |

**Max Performance:** 21.165 GigaMultiplications/Second with Global Dataset Size of 8,388,608 and Local Work Size of 256

**What pattern are you seeing in this performance curve?**

Like Array Multiply and Array Multiply-Add, as the scale of global work and local work has increased, performance has continued to improve.

**Why do you think the pattern looks this way?**

The reason is the same as explained in the previous section. GPU parallelism usually provides the best performance under a larger data set size and a larger local workload (above 32).

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

As in the previous part, if the data set size is greater than 2,097,152 and the number of threads is 32 or greater. However, theoretically the best performance sort will be Array Multiply> Array Multiply – Add> Multiply+Reduce application, therefore, I will choose to use Array Multiply.