CS 475 Parallel Computing
Instructor: Matthew Meyn
Student: **Khuong Luu** (**luukh**)

Project 6

1. **What machine did you run the programs on**

Due heavy loads on the rabbit server, eventually I decided to install everything on my local computer and run it by myself.

- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- Memory: 16GB, 2400Mhz
- Operating System: Ubuntu 16.04
- Kernel: 4.13.0-37-generic
- Number of Cores: 4
- Cache line size: 64 bytes
- **Compiler: gcc**

```
$ clinfo
Number of platforms                 1
  Platform Name                     NVIDIA CUDA
  Platform Vendor                   NVIDIA Corporation
  Platform Version                  OpenCL 1.2 CUDA 9.0.368
  Platform Profile                  FULL_PROFILE
  Platform Extensions               cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics cl_khr_fp64 cl_khr_byte_addressable_store cl_khr_icd
cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll
cl_nv_copy_opts cl_nv_create_buffer
  Platform Extensions function suffix   NV


  Platform Name                     NVIDIA CUDA
Number of devices                   1
  Device Name                       GeForce GTX 1060
  Device Vendor                     NVIDIA Corporation
  Device Vendor ID                  0x10de
  Device Version                    OpenCL 1.2 CUDA
```

```
Driver Version                  384.130
Device OpenCL C Version         OpenCL C 1.2
Device Type                     GPU
```

I use OpenCL version 1.1 from Kronos

```
#define CL_TARGET_OPENCL_VERSION 120
#include <CL/cl.h>
```

2. Multiply and Multiply-Add

2.1. Tables

Multiply program

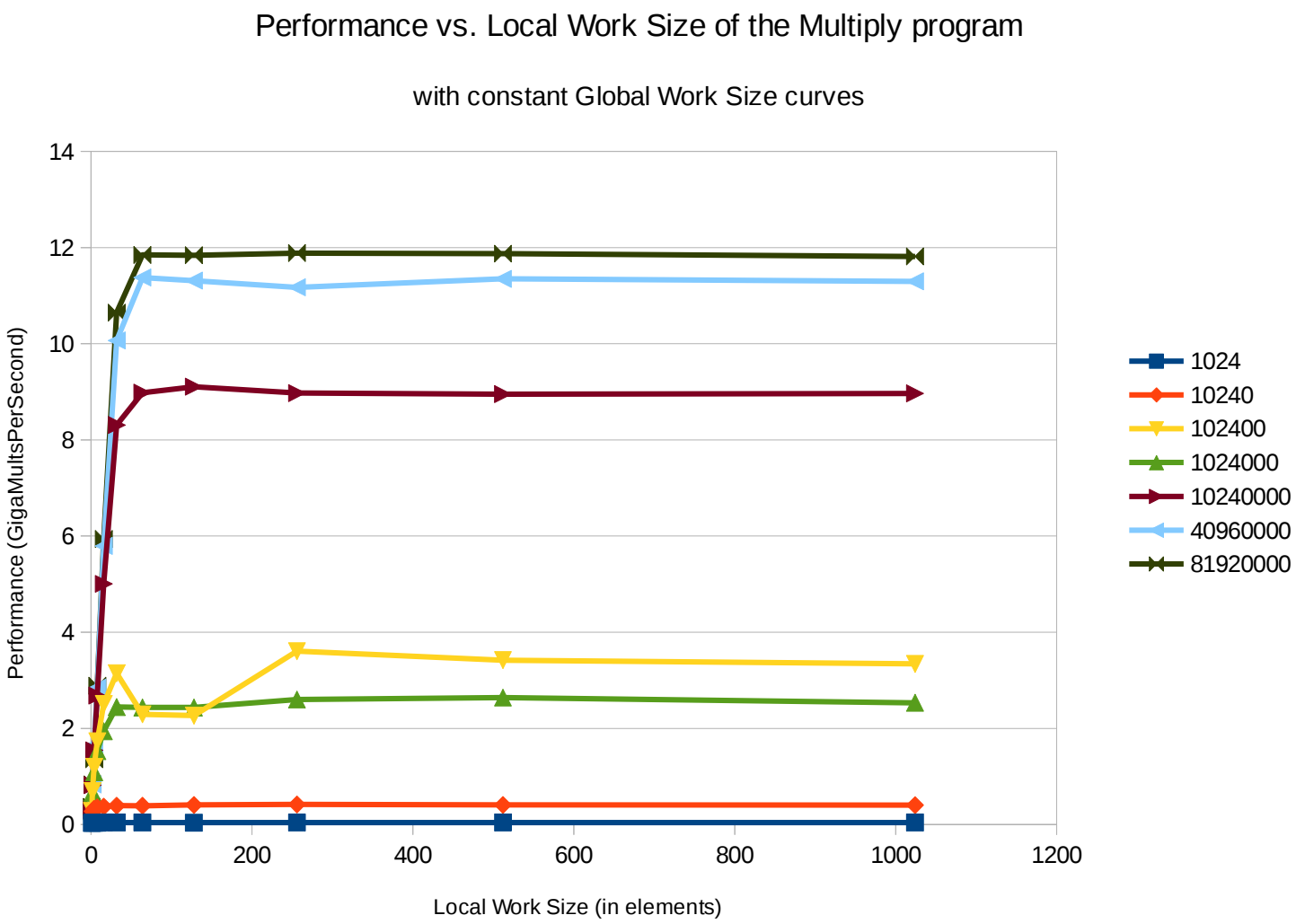
LOCAL_SIZE	1024	10240	102400	1024000	10240000	40960000	81920000
1	0.035	0.172	0.283	0.275	0.3	0.302	0.369
2	0.04	0.276	0.696	0.66	0.819	0.835	0.838
4	0.039	0.328	1.199	1.088	1.543	1.656	1.357
8	0.037	0.356	1.73	1.526	2.683	2.846	2.885
16	0.043	0.374	2.508	1.941	5.005	5.794	5.938
32	0.041	0.393	3.147	2.444	8.308	10.066	10.646
64	0.042	0.391	2.292	2.436	8.981	11.377	11.845
128	0.039	0.409	2.264	2.434	9.105	11.312	11.839
256	0.041	0.417	3.604	2.6	8.973	11.172	11.883
512	0.042	0.409	3.419	2.64	8.951	11.348	11.872
1024	0.042	0.403	3.342	2.533	8.965	11.297	11.814

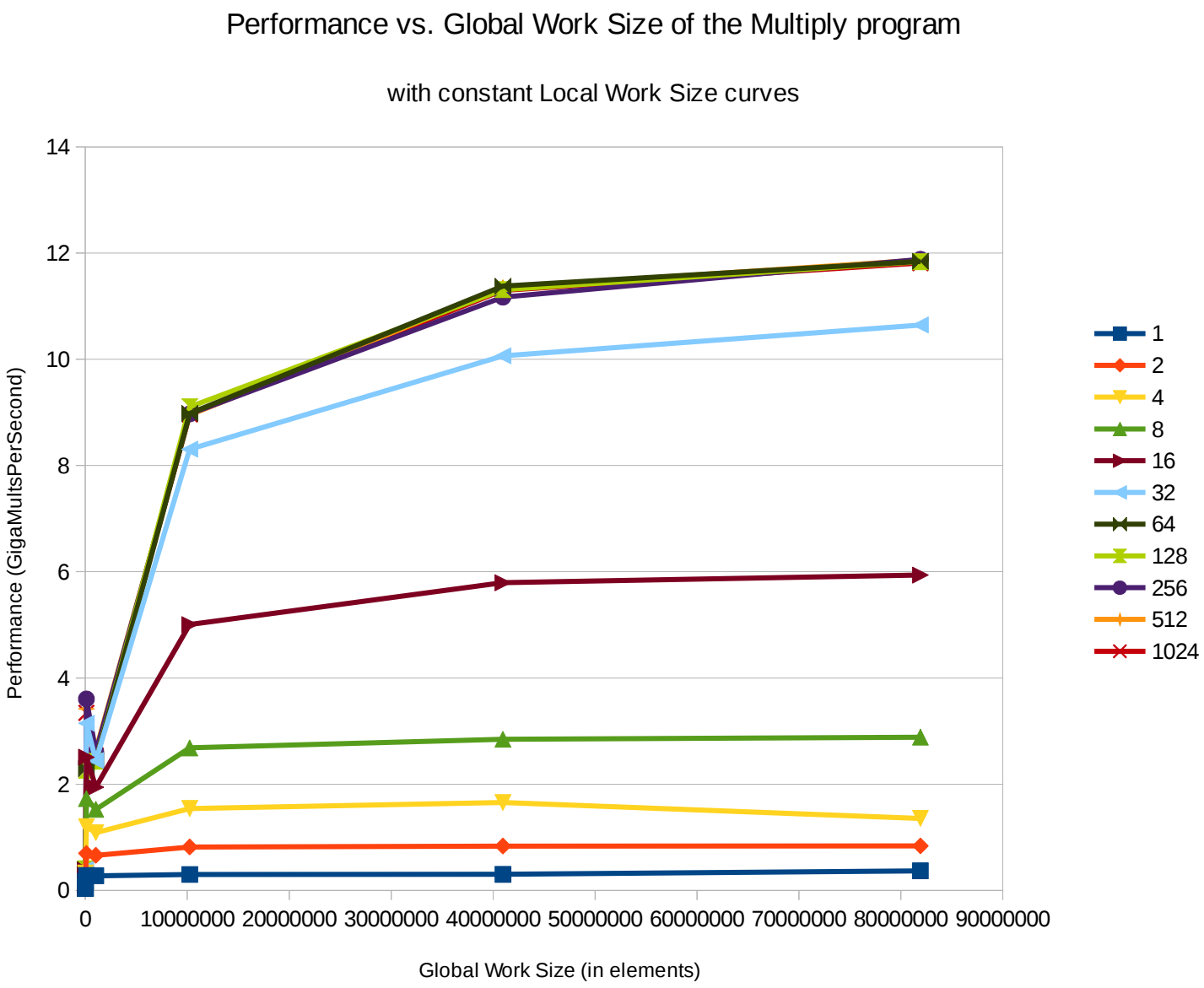
Multiply-Add program

LOCAL_SIZE	1024	10240	102400	1024000	10240000	40960000	81920000
1	0.037	0.207	0.38	0.367	0.414	0.414	0.415
2	0.039	0.274	0.699	0.666	0.818	0.83	0.837
4	0.039	0.328	1.188	1.102	1.584	1.651	1.662
8	0.039	0.361	1.836	1.508	2.795	3.042	3.076
16	0.041	0.367	2.537	2.058	4.987	5.588	5.761
32	0.04	0.262	2.996	2.395	6.903	8.262	8.492
64	0.041	0.401	3.254	2.437	6.811	8.639	9.166
128	0.039	0.402	3.303	2.433	7.269	8.785	9.141
256	0.036	0.4	3.445	2.364	7.272	8.69	9.12
512	0.042	0.414	3.372	2.272	7.347	8.779	9.131
1024	0.042	0.413	3.306	2.312	7.27	8.733	9.109

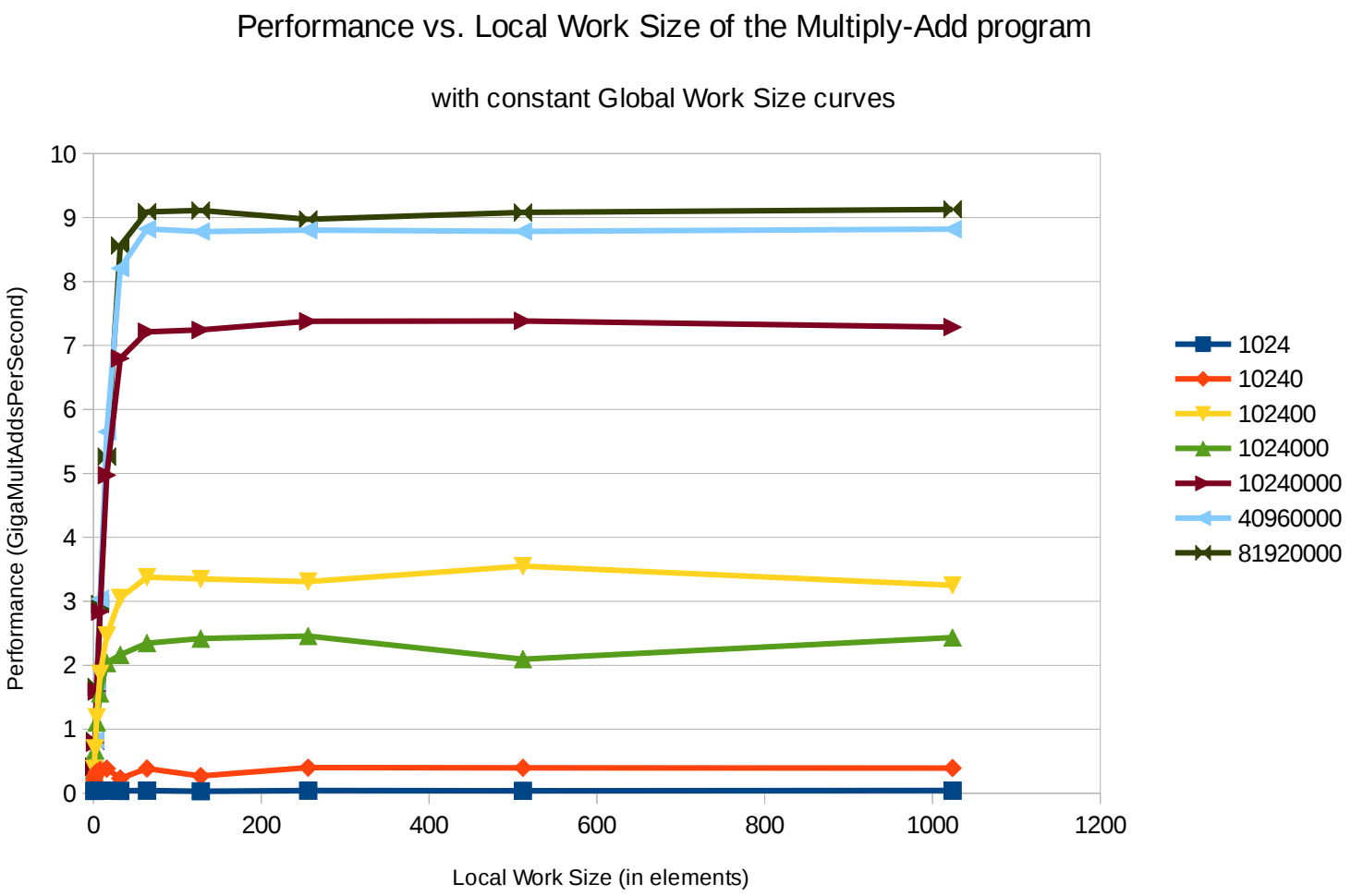
2.2. Graphs

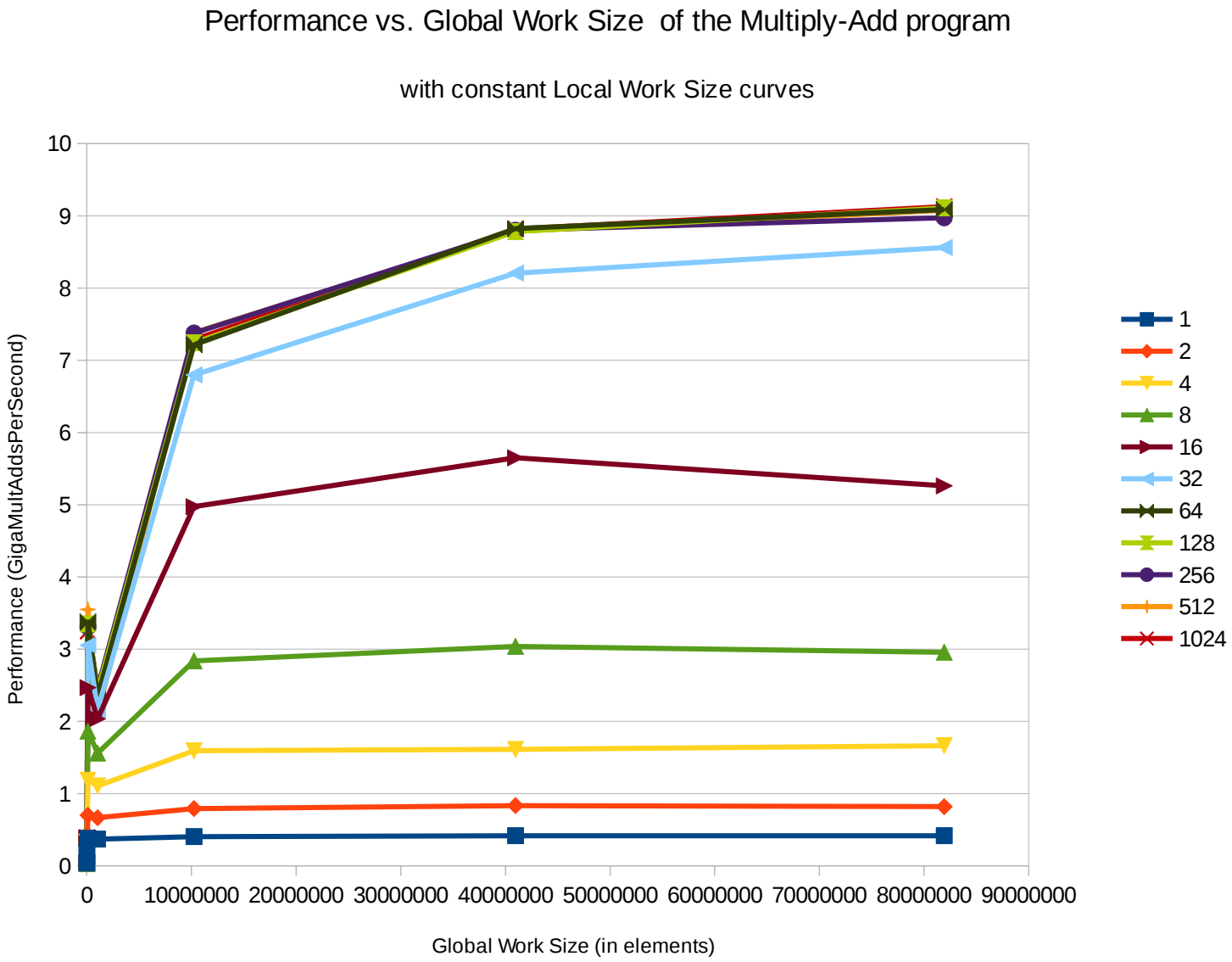
Multiply program




Performance vs. Global Work Size of the Multiply program
with constant Local Work Size curves
Performance (GigaMultsPerSecond)
14
12
10
8
6
4
2
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000
80000000
90000000
Global Work Size (in elements)
1
2
4
8
16
32
64
128
256
512
1024

Multiply-Add Program




Performance vs. Global Work Size of the Multiply-Add program
with constant Local Work Size curves
Performance (GigaMultAddsPerSecond)
10
9
8
7
6
5
4
3
2
1
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000
80000000
90000000
Global Work Size (in elements)
1
2
4
8
16
32
64
128
256
512
1024

3. Pattern and Explanation for Multiply and Multiply-Add program

3.1. Patterns in each of the 2 programs

Pattern 1: Given the constant local sizes, the bigger the global size, the significantly better the performance until the performance converges

Explanation: Given the constant local size, the bigger the global size leads to significantly more work groups being used, which in turn increase the parallelization of the program, which in turn certainly increase the performance significantly unlike in pattern 2 which not necessary increase the parallelization. until we reach the optimal number of work groups of the hardware. This also explain why increase in global work size is more effective than increase in local work size.

Pattern 2: Given the constant global size, increase in local size leads to increase in performance but only for the increases of smalls local sizes (1, 2, 4, 8, 16, 32 , 64). After that, the performance doesn't increase much if not to say not at all since the local size of 64.

Explanation: This is hard to explain but rather it tell us a meaningful and useful conclusion that the optimal value for local size for this kind of task and hardware is 64 (times the size of a float)

3.2. Different between the Multiply program and Mutiply-Add program

Difference: The performance of the Multiply program is higher than that of the Multiply-Add program. Given the same local size and global size, the performance of the Multiply program is of 3 Giga-Multiplies per Second bigger than the performance of the Multiply-Add program.

Explanation: because the kernel of the later has to do (one) more task compared to the former: Adding.

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

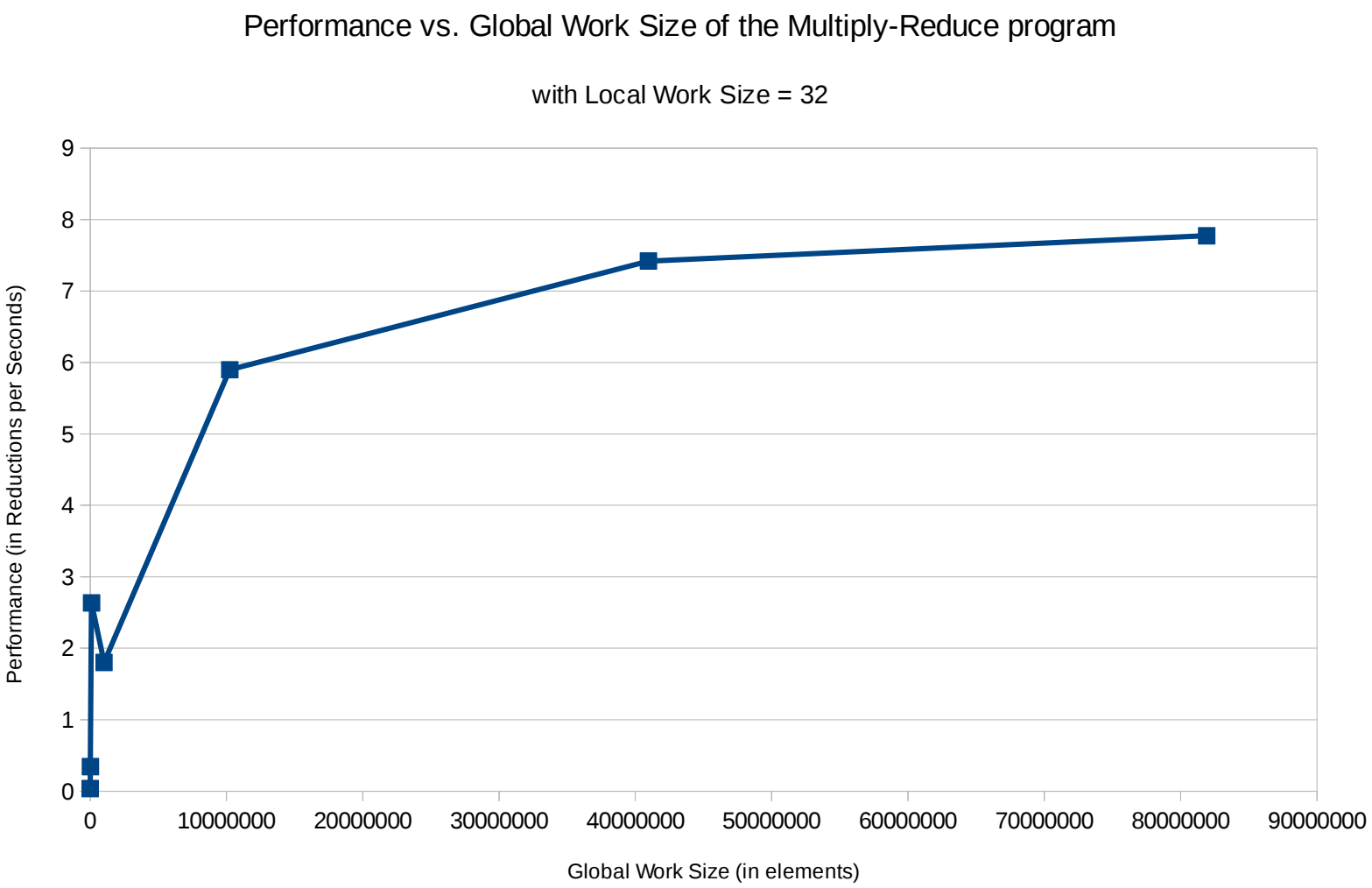
From the patterns and their explanation above, we have learned that increasing the global work size is more effective than increasing the local work size; and there is a optimal value for local work size that we need to find out in order to allocate the local size that use the GPU the most effective way.

4. Benchmark and Analyze Multiply Reduce program

4.1. Table

Global Work Size	1024	10240	102400	1024000	10240000	40960000	81920000
Performance	0.036	0.344	2.635	1.802	5.897	7.418	7.772

4.2. Graph



4.3. Patterns and Explanations

Pattern 1: Increase in global work size leads to increase in performance, until the performance start converging at global work size of 80M.

Explanation: This is the same explanation as with the Multiply program: The increase in Global Work Size leads to more number of work groups being use and thus increase the parallelization of the program, which certainly leads to better performance until we reach the optimal number of work groups.

Pattern 2: The performance of this program is less than the performance of the Multiply and the Multiply-Add programs
Explanation: This is because the kernel of this program has to perform more work than the kernel in the other two programs.

4.4. What does this mean for proper use of GPU computing?

The fact that more busy kernel code has the same performance pattern as the less busy kernel code gives us a lesson that we should do as much work in each work-groups as possible before passing data to/from the host. Data passing is expensive.