To investigate how the running time varies with matrix size and how the versions that use shared memory and non-shared memory compare in terms of wall time.

Jonah Njenga

LASIGMA – RET 2012 - Teachers

East Baton Rouge Public School System

July 13, 2012

Acknowledgment

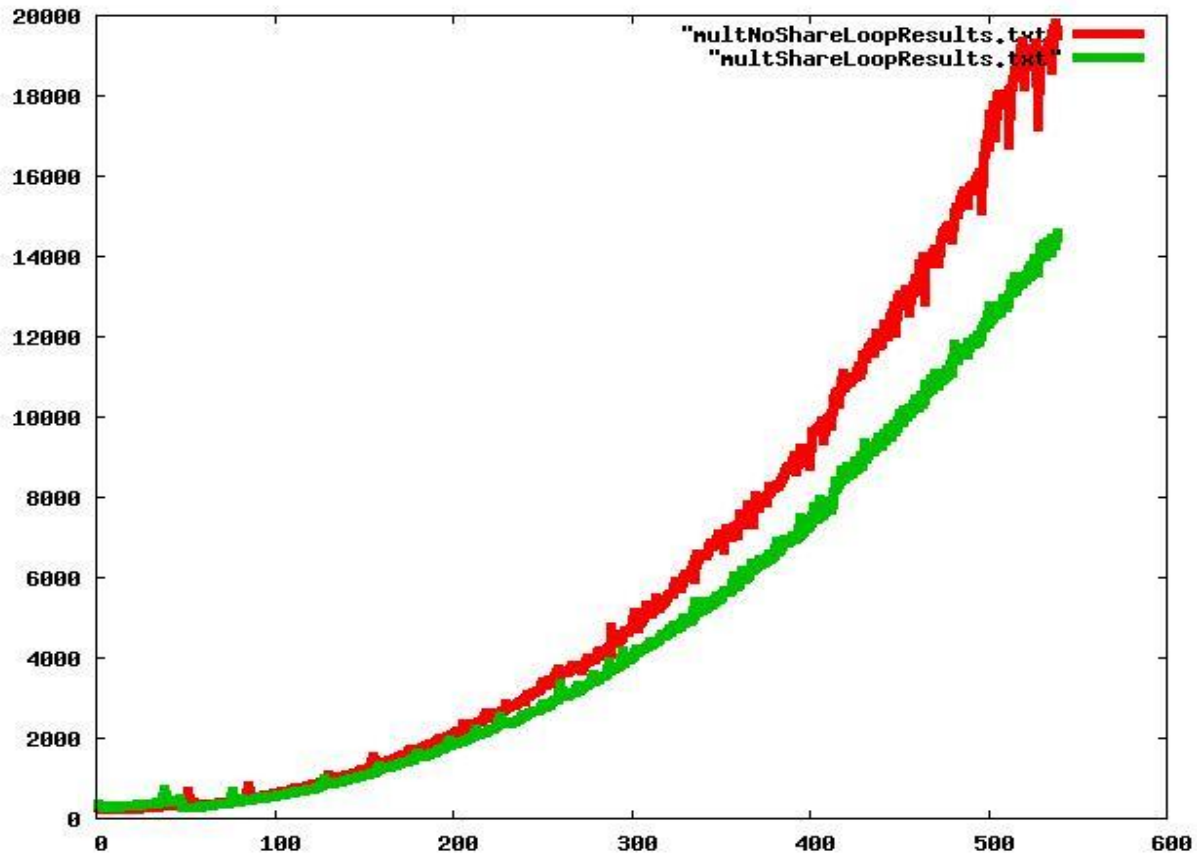
This summer research experience was supported by the National Science Foundation under the NSF EPSCoR Cooperative Agreement No. EPS-1003897 with additional support from the Louisiana Board of Regents.

Abstract

In this activity, an example of Matrix Multiplication was used to study the basics of GPU computing in the CUDA environment.
The use of CUDA and the nvcc compiler made it possible to write programs in C which was run on an ordinary host computer (Linux).

Matrix multiplication was implemented on a CUDA-enabled graphics card.
Programs were run to see how running time varied with the matrices sizes.
Comparison of running time between shared memory version and not shared memory version.

Gnuplot was used to plot running time against dimensions of the matrices.


"multNoShareLoopResults.txt"
"multShareLoopResults.txt"
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
100
200
300
400
500
600

Introduction

Matrix definition

A **matrix** (plural, **matrices**) is a rectangular array of numbers.
We can abbreviate the array as $S = [s_{ij}]$.
The size of a matrix is the number of rows by the number of columns. Thus, S is a i $\times$ j matrix.

Importance:

- Useful and fundamental mathematical objects in scientific computation.

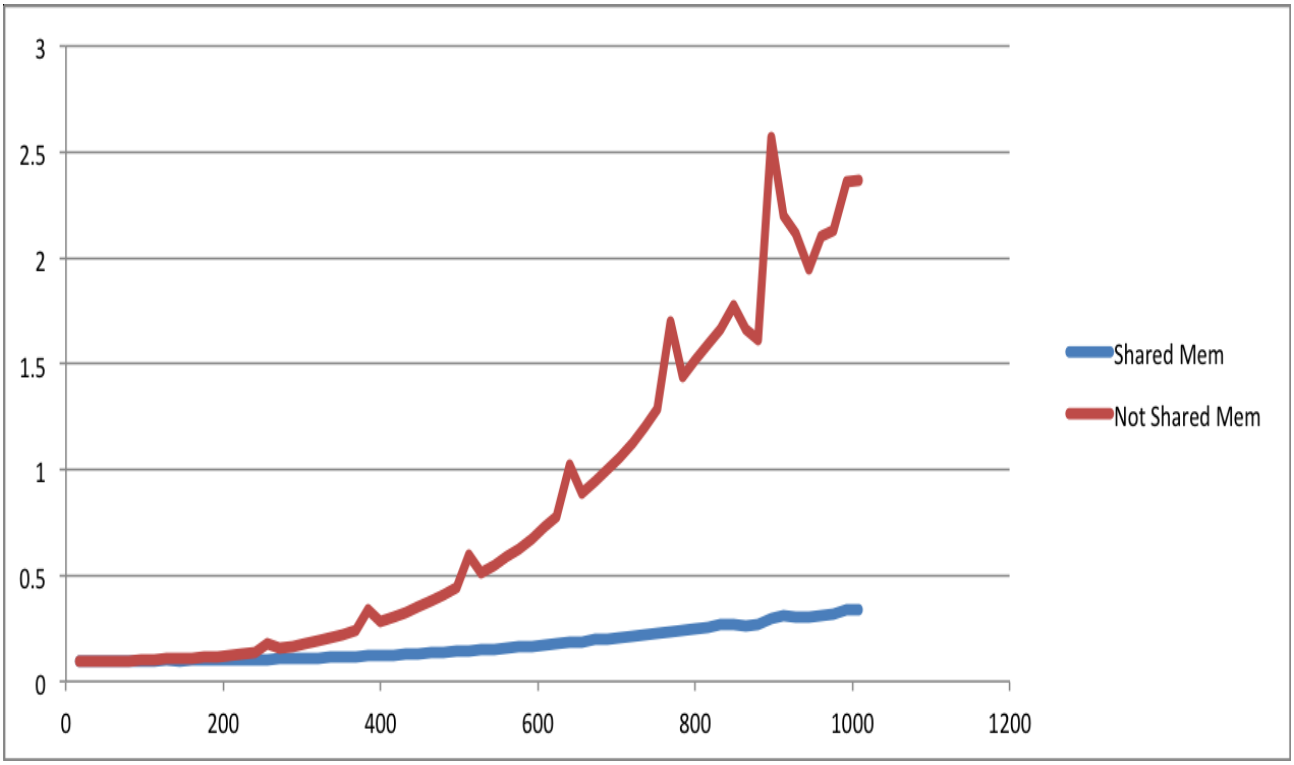
Applications include:
- Computer graphics (3D)
- solving systems of equations
- DNA sequence comparison
- Modeling electrical circuits or computer networks

Matrices Operations:
Addition, subtraction, multiplication and, division.
The ability to multiply matrices allows modeling of many problems.

There is need to optimize the compute capability, which can be described as the general computing power of GPUs.

4 multiprocessors, 32 cores

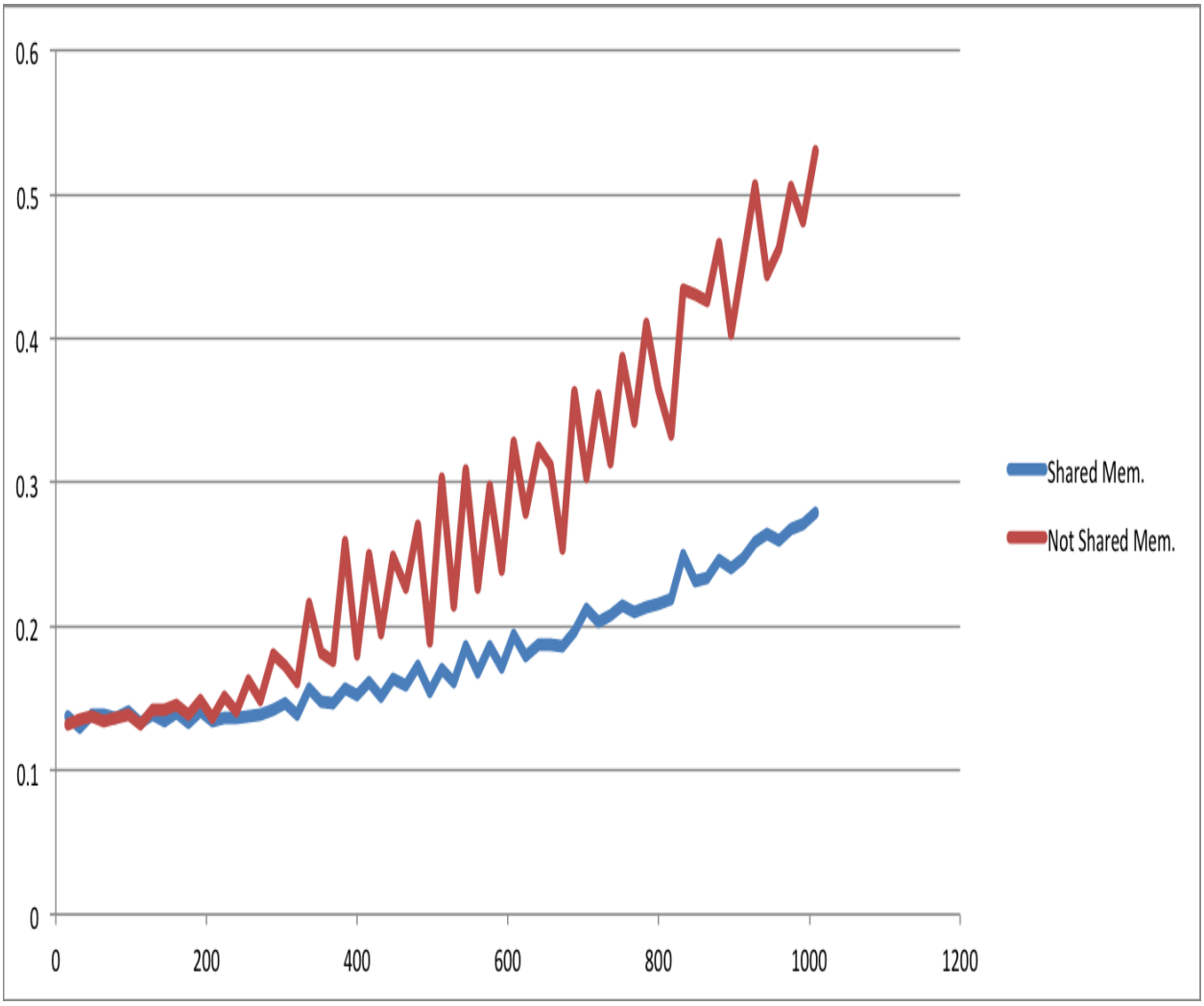
- Dramatic difference in running times
- For matrices of size 1000 X 1000, a factor of seven improvement
- Ratio seems to be increasing as matrix size increases.



Robert Hochberg, April 5, 2012

6 multiprocessors, 48 cores

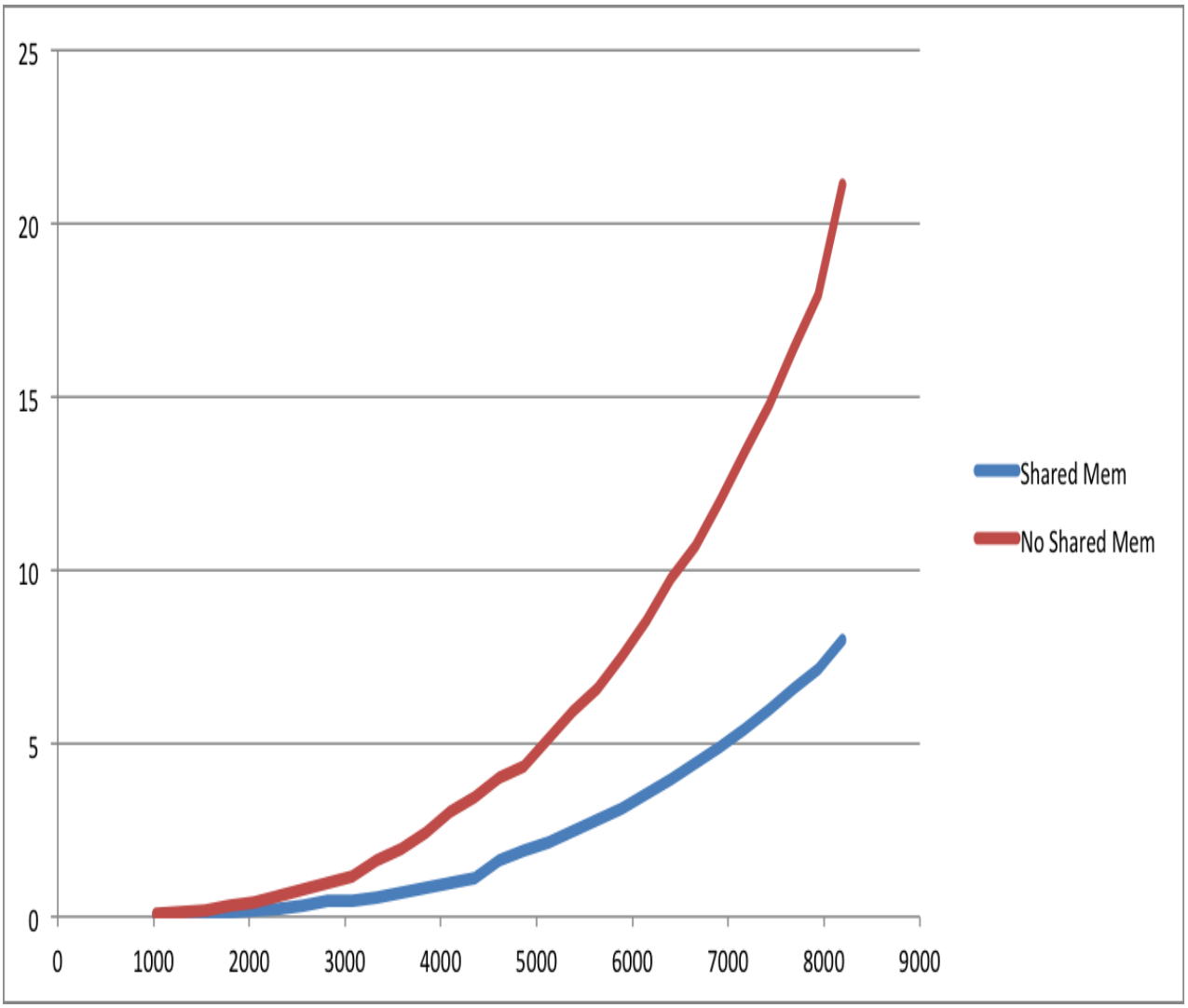
-Ratio approaches 2 as matrix size approaches 1000.
-Increase in multiprocessor enhances improvement in non-shared memory version
- Improvement in the shared memory version due to better coalesce memory access.



Robert Hochberg, April 5, 2012

14 multiprocessors, 448 cores

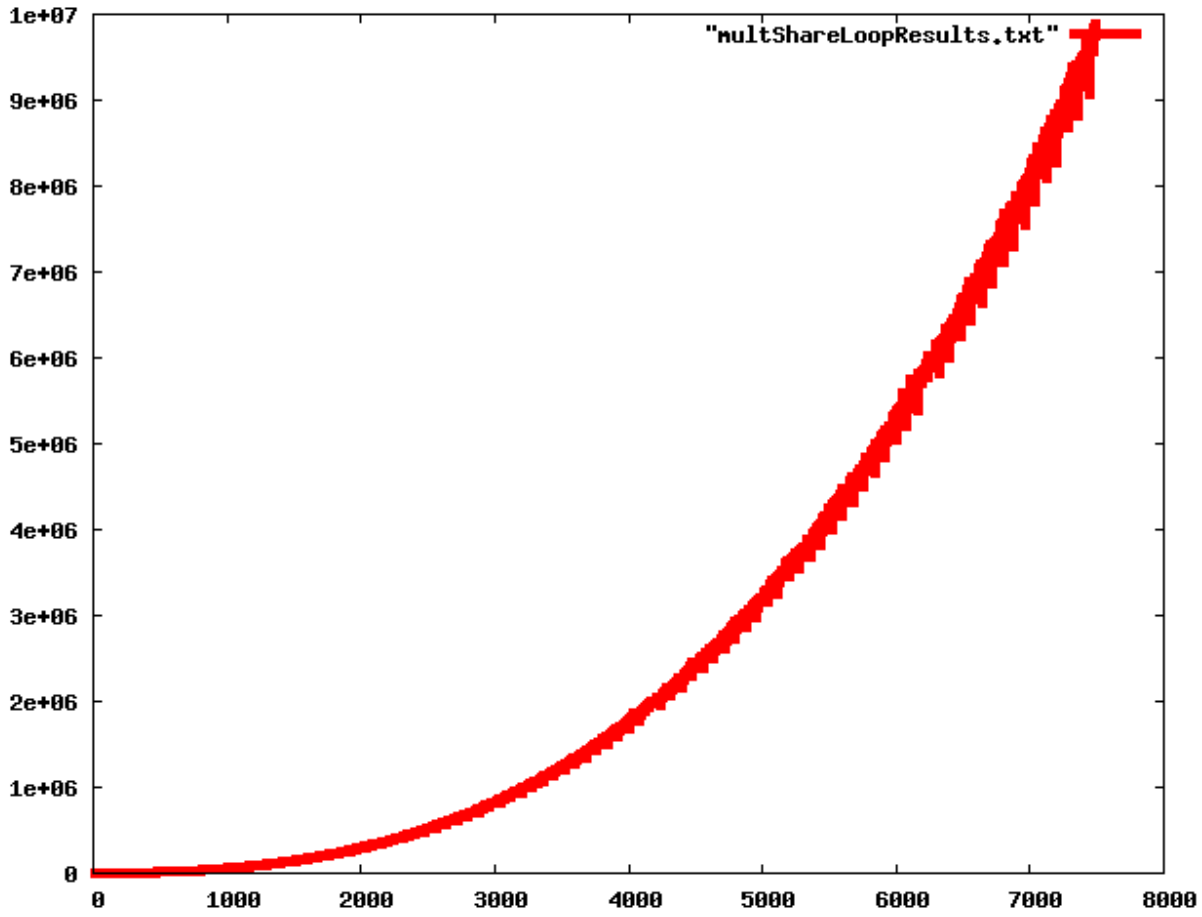
- Ratio of about 1.7
- Location of card affects performance.
- For matrices of size 8000 x 8000, the shared memory Version is faster by a factor of 2.6.



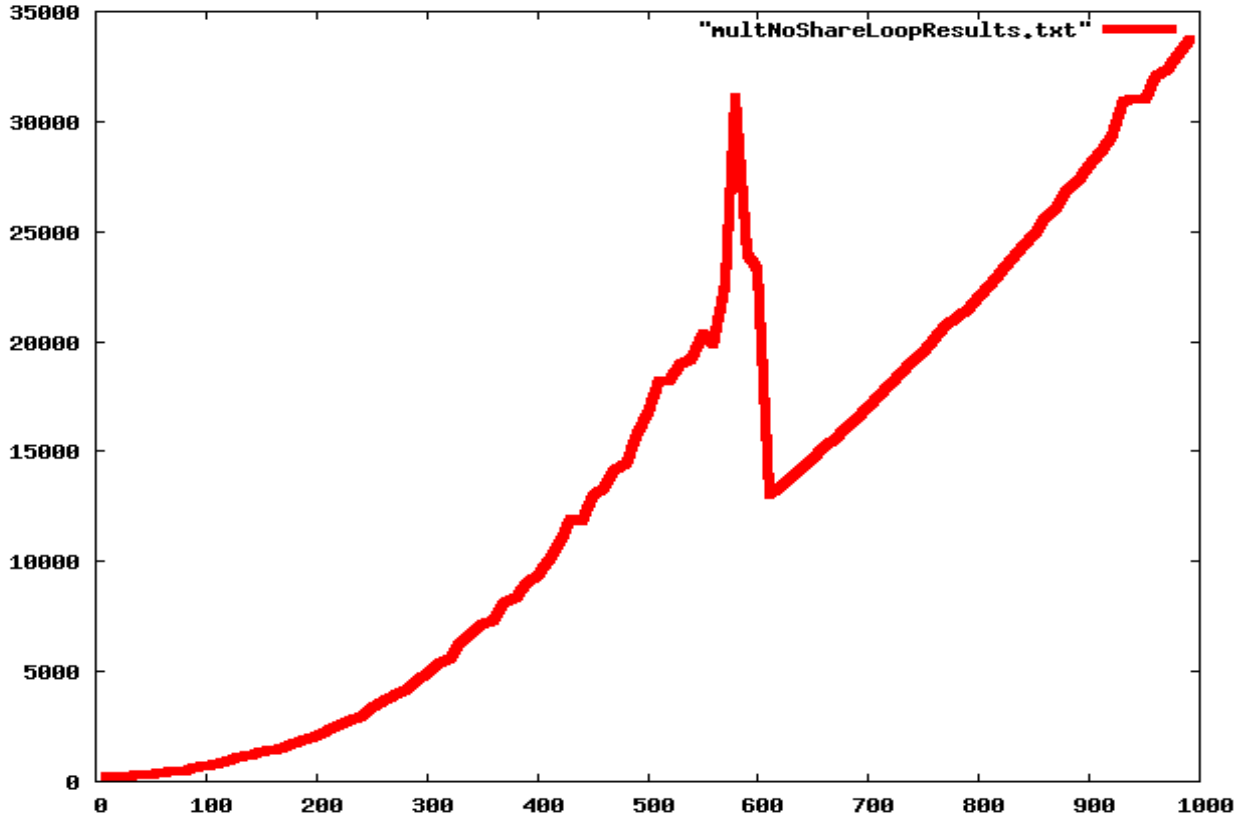
Robert Hochberg, April 5, 2012

Results and Discussion

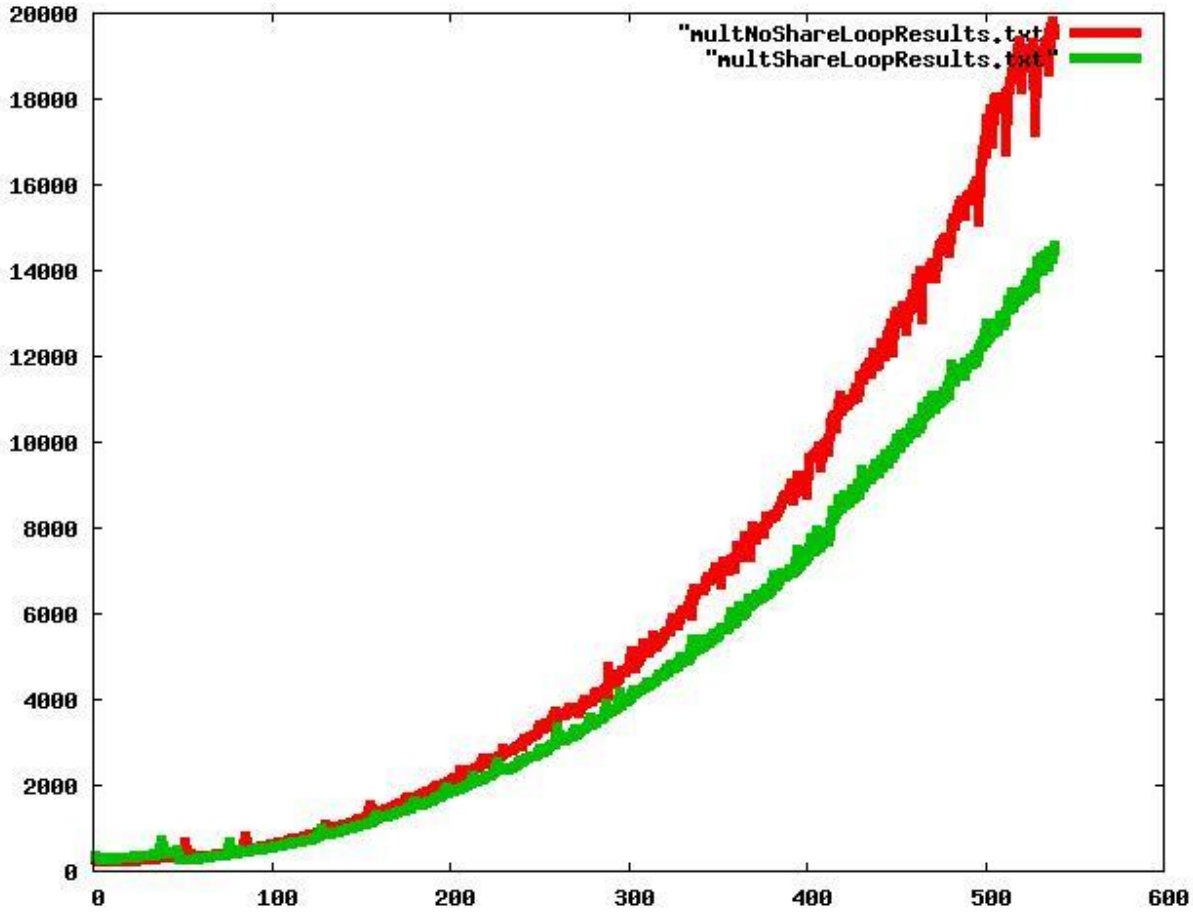
a) MultShare memory version



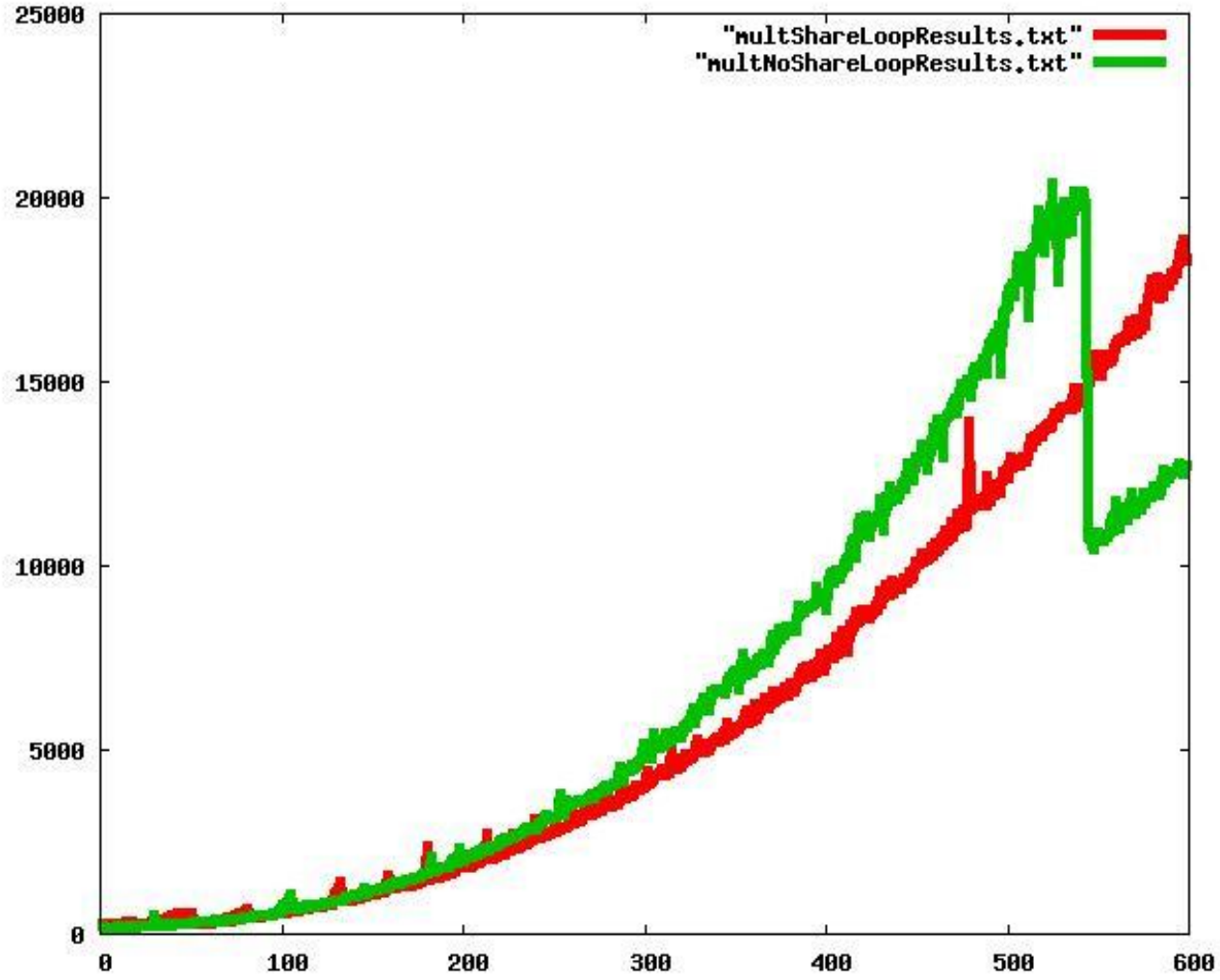
b) Mult-NoShare Memory version



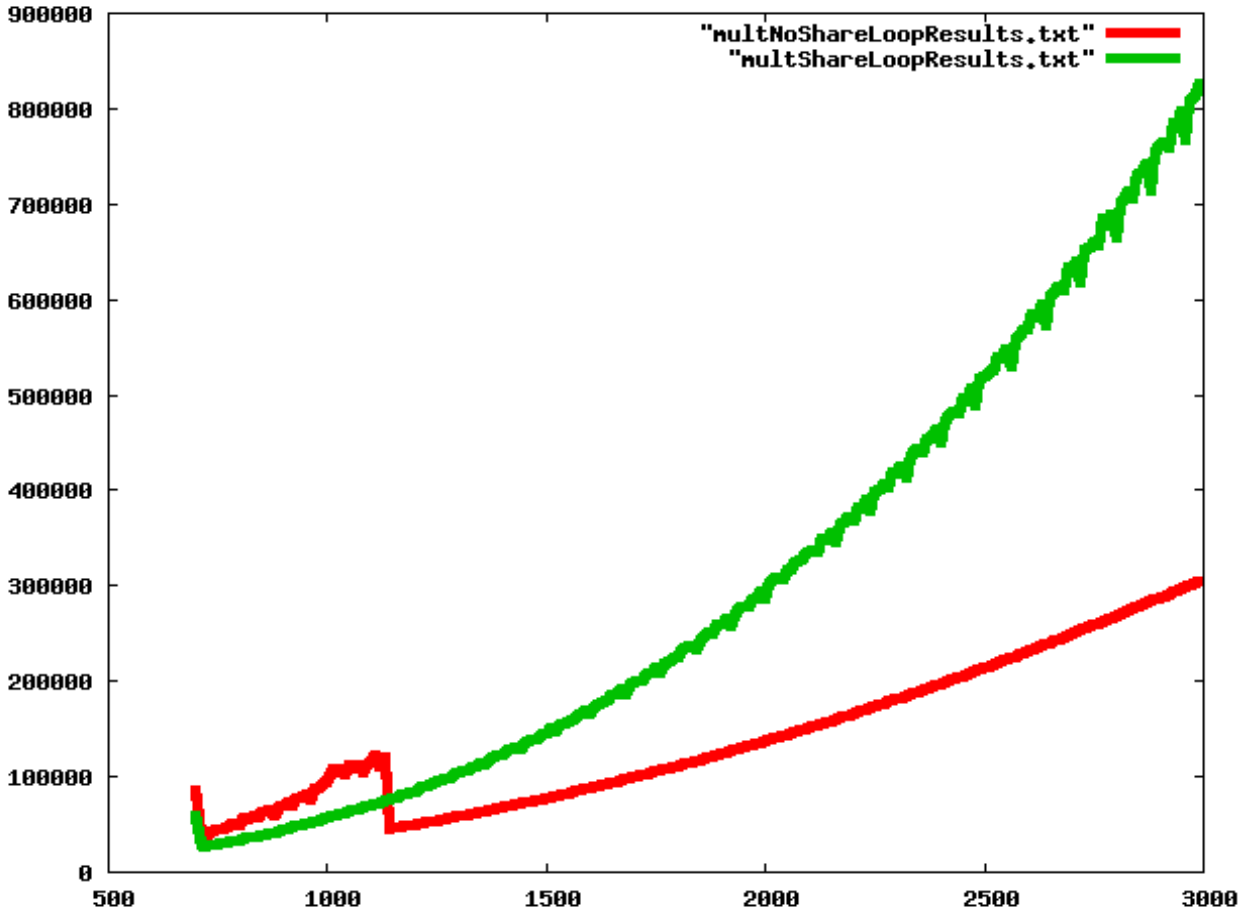
c) Multi-plot : Share/No-Share Comparison (N<600)



d) Multi-plot : Share/No-Share Comparison (N> 600)



e) Multi-plot : Share/No-Share Comparison (N> 3000)



Conclusion

- As the size of a matrix increases the running time increases.
- Share memory version have are generally faster than the non-shared memory version.
- The advantage of share memory is evident when the matrix size increases.

Reference

Matrix Multiplication with CUDA | A basic introduction
to the CUDA programming model, Robert Hochberg, April 5, 2012