TSE Extra Assignment

Tim Ackermans & Dominic Voets

Introduction

For this assignment we were told to define our own matrix calculation on the GPU and CPU. We decided to use the Gaussian Elimination method. This method is also used in calculating the determinant of a square matrix and also to solve a problem that involves multiple equations with multiple variables. This calculation can only be performed on square matrices. Below there is an example of how Gaussian Elimination works.

Gaussian Elimination Example

This is an example which shows how Gaussian Elimination works within a 4x4 matrix.
Our starting Matrix

2	6	8	9
3	5	7	1
9	3	2	1
5	7	6	8

Divide the 1st row by 2

1	3	4	9/2
3	5	7	1
9	3	2	1
5	7	6	8

Multiply the 1st row by 3

3	9	12	27/2
3	5	7	1
9	3	2	1

5	7	6	8

Subtract the 1st row from the 2nd row and restore it

1	3	4	9/2
0	-4	-5	-25/2
9	3	2	1
5	7	6	8

Multiply the 1st row by 9

9	27	36	81/2
0	-4	-5	-25/2
9	3	2	1
5	7	6	8

Subtract the 1st row from the 3rd row and restore it

1	3	4	9/2
0	-4	-5	-25/2
0	-24	-34	-79/2
5	7	6	8

Multiply the 1st row by 5

5	15	20	45/2
0	-4	-5	-25/2
0	-24	-34	-79/2

5	7	6	8

Subtract the 1st row from the 4th row and restore it

1	3	4	9/2
0	-4	-5	-25/2
0	-24	-34	-79/2
0	-8	-14	-29/2

Restore the 1st row to the original view

2	6	8	9
0	-4	-5	-25/2
0	-24	-34	-79/2
0	-8	-14	-29/2

Divide the 2nd row by -4

2	6	8	9
0	1	5/4	25/8
0	-24	-34	-79/2
0	-8	-14	-29/2

Multiply the 2nd row by -24

2	6	8	9
0	-24	-30	-75
0	-24	-34	-79/2

0	-8	-14	-29/2

Subtract the 2nd row from the 3rd row and restore it

2	6	8	9
0	1	5/4	25/8
0	0	-4	71/2
0	-8	-14	-29/2

Multiply the 2nd row by -8

2	6	8	9
0	-8	-10	-25
0	0	-4	71/2
0	-8	-14	-29/2

Subtract the 2nd row from the 4th row and restore it

2	6	8	9
0	1	5/4	25/8
0	0	-4	71/2
0	0	-4	21/2

Restore the 2nd row to the original view

2	6	8	9
0	-4	-5	-25/2
0	0	-4	71/2

0	0	-4	21/2

Subtract the 3rd row from the 4th row

2	6	8	9
0	-4	-5	-25/2
0	0	-4	71/2
0	0	0	-25

Results

Example of our algorithm

Original matrix 8x8:

```
Matrix:
6.000000 9.000000 8.000000 5.000000 9.000000 2.000000 4.000000 1.000000
8.000000 3.000000 9.000000 3.000000 8.000000 7.000000 8.000000 6.000000
8.000000 9.000000 4.000000 1.000000 1.000000 7.000000 6.000000 1.000000
5.000000 8.000000 7.000000 6.000000 9.000000 6.000000 3.000000 1.000000
3.000000 1.000000 7.000000 5.000000 9.000000 2.000000 8.000000 4.000000
3.000000 7.000000 3.000000 4.000000 7.000000 3.000000 4.000000 8.000000
3.000000 2.000000 6.000000 6.000000 2.000000 7.000000 4.000000 8.000000
3.000000 4.000000 8.000000 5.000000 5.000000 3.000000 6.000000 7.000000
```

By the CPU 8x8:

```
Matrix:
6.000000 9.000000 8.000000 5.000000 9.000000 2.000000 4.000000 1.000000
0.000000 -9.000000 -1.666667 -3.666667 -4.000000 4.333333 2.666667 4.666667
0.000000 0.000000 -6.111111 -4.444444 -9.666666 2.888889 -0.222222 -1.888889
0.000000 0.000000 0.000000 1.454546 0.896970 4.687878 -0.193940 0.351515
0.000000 0.000000 0.000000 0.000000 -0.500000 -3.062498 5.000000 0.250000
0.000000 0.000000 0.000000 0.000000 0.000000 -19.312468 30.499971 10.249998
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 -5.211864 22.445736
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 61.861504
```

By Kernel 8x8:

```
Matrix:
6.000000 9.000000 8.000000 5.000000 9.000000 2.000000 4.000000 1.000000
0.000000 -9.000000 -1.666667 -3.666667 -4.000000 4.333333 2.666667 4.666667
0.000000 0.000000 -6.111111 -4.444445 -9.666667 2.888889 -0.222222 -1.888889
0.000000 0.000000 0.000000 1.454545 0.896970 4.687878 -0.193940 0.351515
0.000000 0.000000 0.000000 0.000000 -0.500002 -3.062500 5.000000 0.250000
0.000000 0.000000 0.000000 0.000000 0.000000 -19.312439 30.499907 10.249994
0.000000 0.000000 0.000000 0.000000 0.000000 0.000004 -5.211872 22.445732
0.000000 0.000000 0.000000 0.000000 0.000000 0.000010 0.000000 61.861404
```

Timing table

Matrix Size (X & Y)	CPU Time(ms)	Kernel 1	Kernel 2
4	0.0333	611.96	630.18
8	0.132	641.28	613.02
16	0.970	719.84	690.79
32	7.927	669.20	550.92
64	64.25	697.64	584.36
128	542.18	788.71	576.77
256	2636.68	1542.33	538.56
350	7940.86	5217.65	541.27
512	22552.36	16070.07	CL_OUT_OF_RESOURCES[1]

[1] Kernel 2 uses more memory than kernel 1, therefore it cannot be used to calculate a 512x512 matrix

In this timing table it is clear that CPU is the quickest with small matrices, but when the size of the matrix exceeds 16x16 the time it takes for the CPU to calculate the matrix multiplies with 8 with every step the matrix increases in size. Further it is clear that Kernel 2 is a lot more efficient than Kernel 1.