Ausganssituation

Die GPU Implementierung der Fraktale der letzten Aufgabenstellung wird jetzt für die Optimierungsaufgabenstellung herangezogen.

Die folgende Abbildung zeigt, die Ausganssituation ohne Optimierung, mit double als value data type. Die Tests werden jeweils mit zweier Potenzen für die block size bis 32 durchgeführt.

```
--------------------------------------------------
Device Metadata:
--------------------------------------------------
Device: GeForce GTX 770M
Compute capability: 3.0
Arch: Kepler
Cores: 192
Global memory: 3072 MB
Shared memory per block: 48 KB
Shared memory per multiprocessor: 48 KB
Max threads per block: 1024
--------------------------------------------------


################################################################################
Start GPU tests 'GPLS'
################################################################################
name: GPU-GPLS | block_size: 2 | millis: 1898
name: GPU-GPLS | block_size: 4 | millis: 713
name: GPU-GPLS | block_size: 8 | millis: 442
name: GPU-GPLS | block_size: 16 | millis: 587
name: GPU-GPLS | block_size: 32 | millis: 1282
################################################
Ended GPU tests 'GPLS'
################################################
```

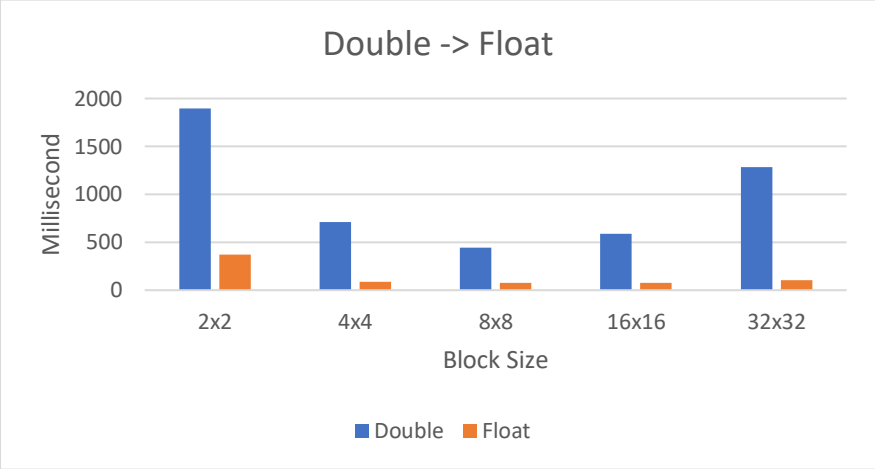
Double value type -> float value type

Als erste Optimierung werden die Variablen, die jetzt double als Datentyp verwenden, auf float umgestellt.

```
--------------------------------------------------
Device Metadata:
--------------------------------------------------
Device: GeForce GTX 770M
Compute capability: 3.0
Arch: Kepler
Cores: 192
Global memory: 3072 MB
Shared memory per block: 48 KB
Shared memory per multiprocessor: 48 KB
Max threads per block: 1024
--------------------------------------------------


################################################################################
Start GPU tests 'GPLS'
################################################################################
name: GPU-GPLS | block_size: {2,2} | millis: 374
name: GPU-GPLS | block_size: {4,4} | millis: 85
name: GPU-GPLS | block_size: {8,8} | millis: 77
name: GPU-GPLS | block_size: {16,16} | millis: 78
name: GPU-GPLS | block_size: {32,32} | millis: 104
################################################
Ended GPU tests 'GPLS'
################################################
```

Wie in der vorherigen Abbildung ersichtlich, wirkt sich diese Optimierung massiv positiv auf die Performance aus.



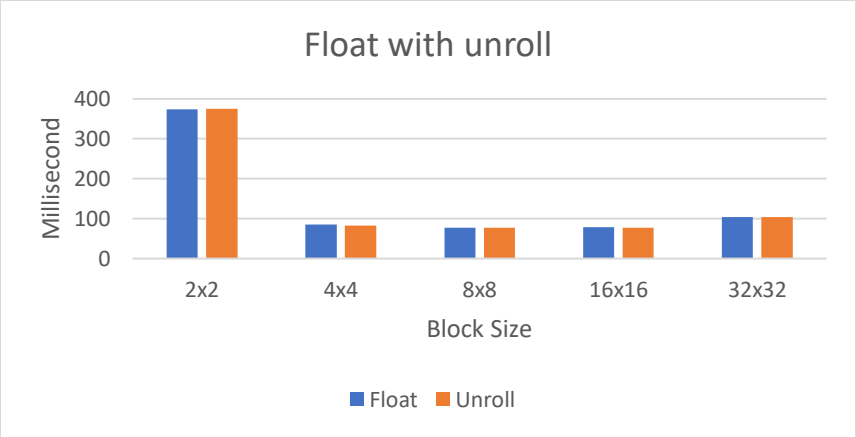
Double stellt zwar mehr Genauigkeit dar und hat eine größere Domain, jedoch wird dadurch auch die Geschwindigkeit für die Genauigkeit geopfert. Bei Float opfert man etwas Genauigkeit für die Performance.

#pragma unroll_completely

Dieses Makro weist den Compiler an, die Schleife auszurollen und sie so zu parallelisieren. Jedoch hat diese Optimierung im Gegensatz zu Float nichts gebracht.

```
-----------------------------------------------------
Device Metadata:
-----------------------------------------------------
Device: GeForce GTX 770M
Compute capability: 3.0
Arch: Kepler
Cores: 192
Global memory: 3072 MB
Shared memory per block: 48 KB
Shared memory per multiprocessor: 48 KB
Max threads per block: 1024
-----------------------------------------------------


################################################################################
Start GPU tests 'GPLS'
################################################################################
name: GPU-GPLS | block_size: {2,2} | millis: 375
name: GPU-GPLS | block_size: {4,4} | millis: 82
name: GPU-GPLS | block_size: {8,8} | millis: 77
name: GPU-GPLS | block_size: {16,16} | millis: 77
name: GPU-GPLS | block_size: {32,32} | millis: 104
################################################
Ended GPU tests 'GPLS'
################################################
```

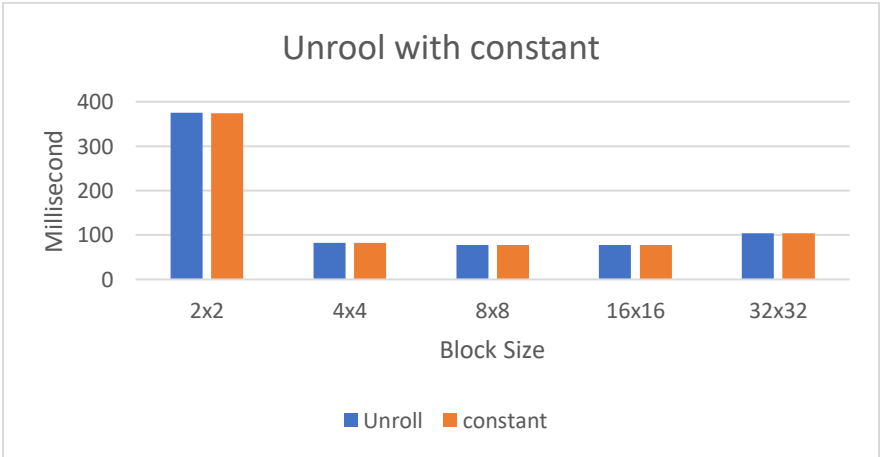


RGB Map in constant memory

Als nächstes wurde versucht das RGB Array in den konstant Memory zu kopieren und nicht das Array als Argument dem Kernel zu übergeben. Diese Optimierung hat leider auch keine Verbesserung erwirkt.

```
-------------------------------------------------------
Device Metadata:
-------------------------------------------------------
Device: GeForce GTX 770M
Compute capability: 3.0
Arch: Kepler
Cores: 192
Global memory: 3072 MB
Shared memory per block: 48 KB
Shared memory per multiprocessor: 48 KB
Max threads per block: 1024
-------------------------------------------------------


################################################################################
Start GPU tests 'GPLS'
################################################################################
name: GPU-GPLS | block_size: {2,2} | millis: 375
name: GPU-GPLS | block_size: {4,4} | millis: 82
name: GPU-GPLS | block_size: {8,8} | millis: 77
name: GPU-GPLS | block_size: {16,16} | millis: 77
name: GPU-GPLS | block_size: {32,32} | millis: 104
################################################
Ended GPU tests 'GPLS'
################################################
```

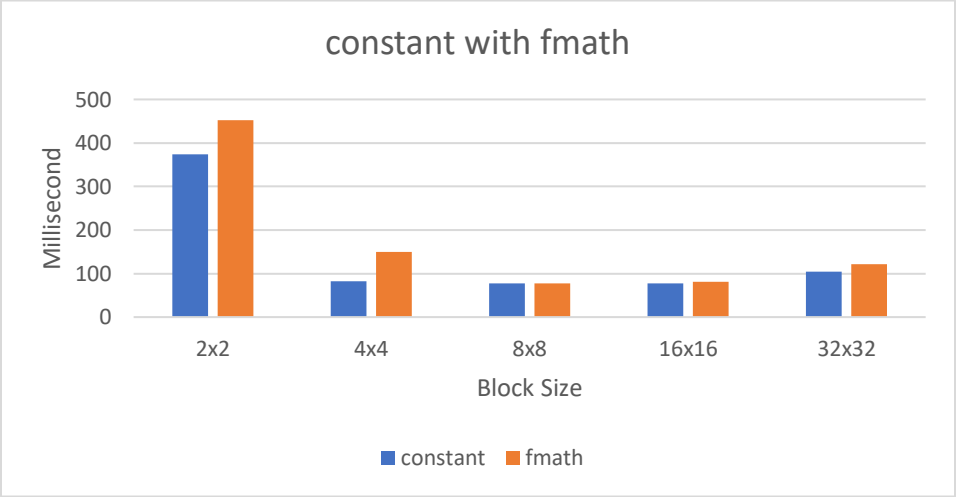


Cuda fast math

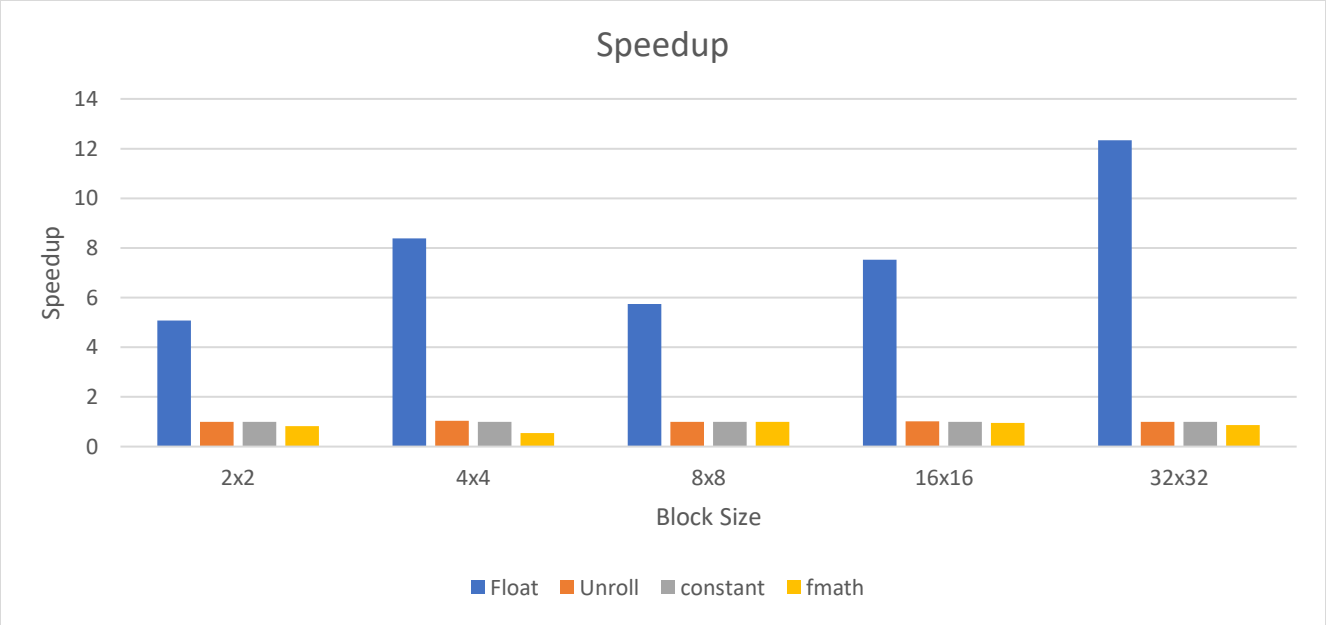
Als nächstes wird versucht die arithmetischen Operationen zu optimieren, indem die Cuda FAST Math API verwendet wird.

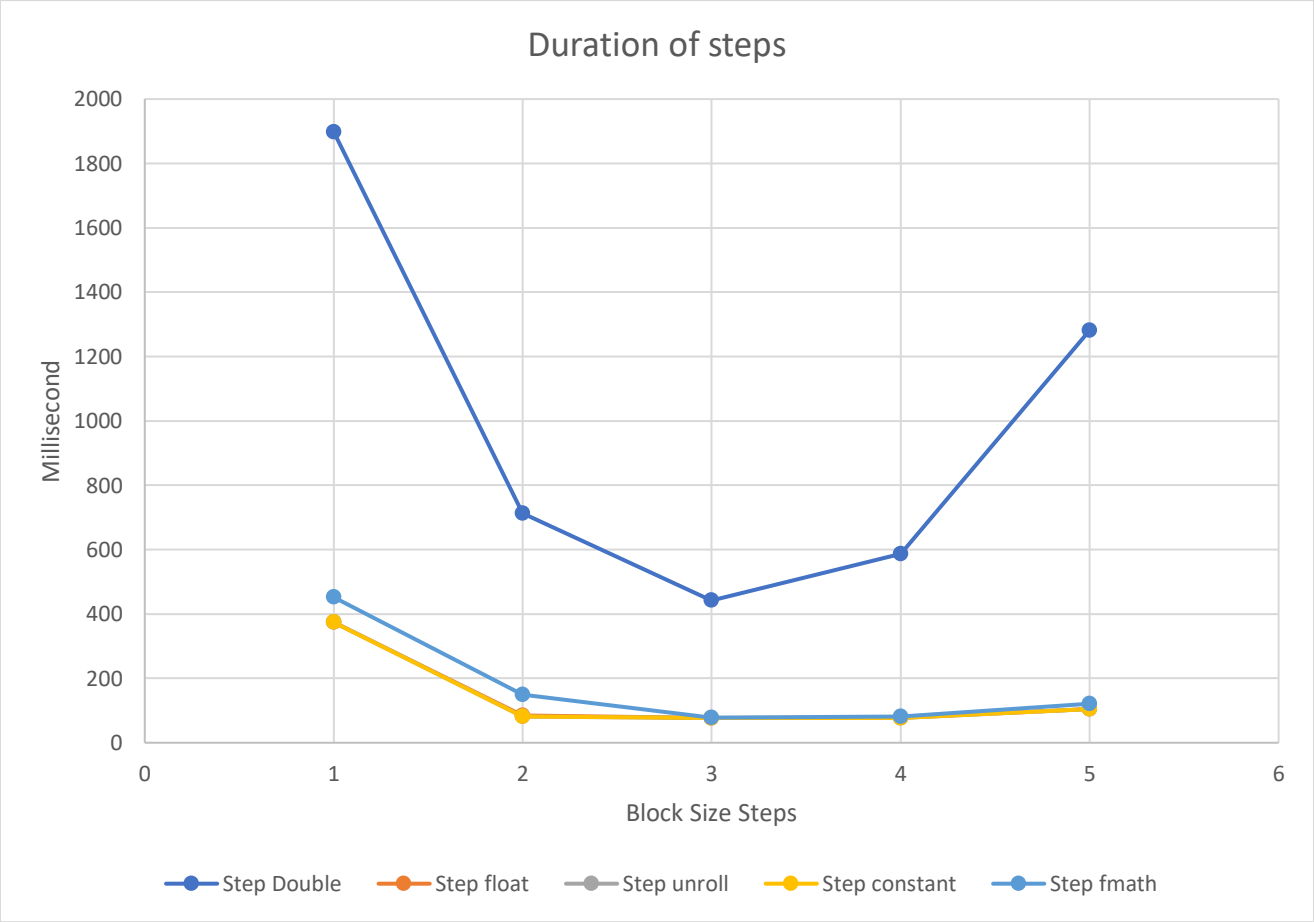
```
-----------------------------------------------------
Device Metadata:
-----------------------------------------------------
Device: GeForce GTX 770M
Compute capability: 3.0
Arch: Kepler
Cores: 192
Global memory: 3072 MB
Shared memory per block: 48 KB
Shared memory per multiprocessor: 48 KB
Max threads per block: 1024
-----------------------------------------------------


################################################################################
Start GPU tests 'GPLS'
################################################################################
name: GPU-GPLS | block_size: {2,2} | millis: 452
name: GPU-GPLS | block_size: {4,4} | millis: 150
name: GPU-GPLS | block_size: {8,8} | millis: 78
name: GPU-GPLS | block_size: {16,16} | millis: 81
name: GPU-GPLS | block_size: {32,32} | millis: 121
################################################
Ended GPU tests 'GPLS'
################################################
Press any key to continue
```



In diesem Fall hat sich sogar gezeigt, dass die Verwendung von der CUDA Math API sogar die Laufzeit verschlechtert.





Die Optimierung von double auf float hat am meisten Performanceverbesserungen gebracht. Die anderen Optimierungen wirken sich nicht aus. Wieder erwarten hat die Verwendung der Math API sogar eine Verschlechterung der Performance bewirkt.