Zusammenfassung

This work introduces a way to generate “plants”, espe

1 Einführung und Allgemeines

In diesem Praktikum sollen verschiedene Werte gemessen werden.

Grundlage CPU (Host) und GPU (Client)

Einige relevante Spezifikationen der Grafikkarte sind:

- 1920 Cores, wovon ein Warp jeweils 32 Threads umfasst
- Maximale Taktung laut „nvidia-smi“: 1708 MHz
- Speicherkonfiguration: 8 GB GDDR5 RAM
- Herstellerangabe für den Durchsatz: 256 GB/s
- PCIe 3.0 fähig, in der Messung wurde allerdings nur PCIe 2.0x16 verwendet. Die maximale Kopierrate von PCIe 2.0x16 entspricht 8.0GB/s

Grundlage CPU (Host) und GPU (Client)

Als Vergleichsmaßstab für die unterschiedlichen Kernel ist zuerst die mit Standard-Kopieroperatoren erziehlbare Kopierrate relevant. Abbildung 1 zeigt die für `CudaMalloc()` gemessenen Kopierraten und die vom Hersteller angegebenen Maximalwerte sowie die Spezifikation für den verwendeten PCIe.

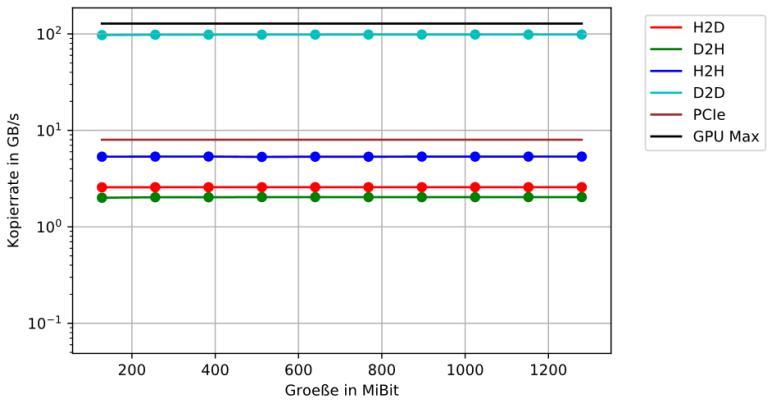

Abbildung 1: Kopierrate von CudaMemcpy()

Bei der tatsächlichen Durchführungen der Messungen ist es wichtig, zu beachten, dass jeweils der erste Kernel Aufruf eines CUDA Programms eine wesentlich längere Zeit benötigt als darauf folgende Aufrufe des selben Kernels. Abbildung 2 zeigt eine Beispielmessung für einen Empty Kernel, dh. einen Kernel, der zwar aufgerufen wird, dann auf der GPU allerdings keine Arbeit verrichtet. Um dieser Verzerrung entgegenzuwirken, muss vor jeder Laufzeitmessung für einen Kernel dieser einmal als „Warmup“ aufgerufen, aber dann aus der eigentlichen Messung herausgerechnet werden.

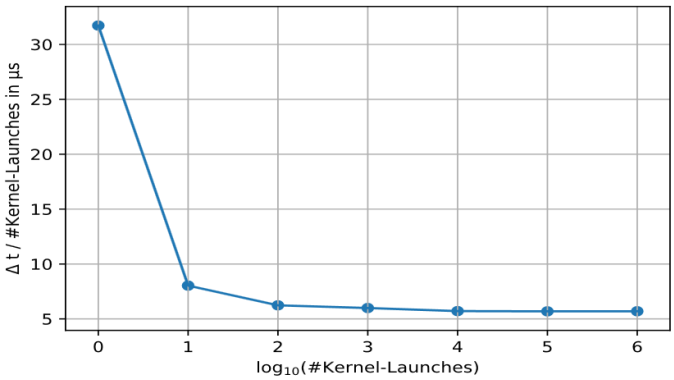

Abbildung 2: Startup Cost eines Empty Kerneles

2 Copy Kernel

Der „Copy Kernel“, der jeweils ein Element eines Buffers in die entsprechende Position eines anderen Buffers kopiert, lässt sich durch folgenden CUDA-Quellcode beschreiben:

```
template<typename T>
__global__
void copyKernel(T* out, T* in) {
        unsigned id = threadIdx.x + blockIdx.x * blockDim.x;

        out[id] = in[id];
}
```

Obwohl die Funktion als Input nur Zeiger auf ein Quell-und ein Ziel-Buffer nimmt, gibt es doch drei implizite Parameter. Diese treten auch bei allen folgenden Messungen auf:

1. Anzahl der Blöcke, in die die einzelnen Threads von der GPU gruppiert werden
2. Die Größe dieser Blücke
3. Zugriffstyp `T` (zB. `char`, `int`)

Abbildung 3 und Abbildung 4 zeigen die Auswirkungen von Variationen über die drei Parameter auf die durch den Copy Kernel erzielte Kopierrate. Dabei zeigt sich, dass sowohl für die Anzahl der Blöcke, für die Blockgröße als auch für `sizeof(T)` ein jeweils höherer Parameter zu einer besseren Kopierrate führt. Für hinreichend hohe Werte wird eine Sättigung nahe des Ergebnisses aus Abbildung 1 erzielt. Interessant ist außerdem, dass die Abbildungen sich mit steigendem `sizeof(T)` je linearer verhalten, je geringer Blockgröße bzw. Blockanzahl sind.

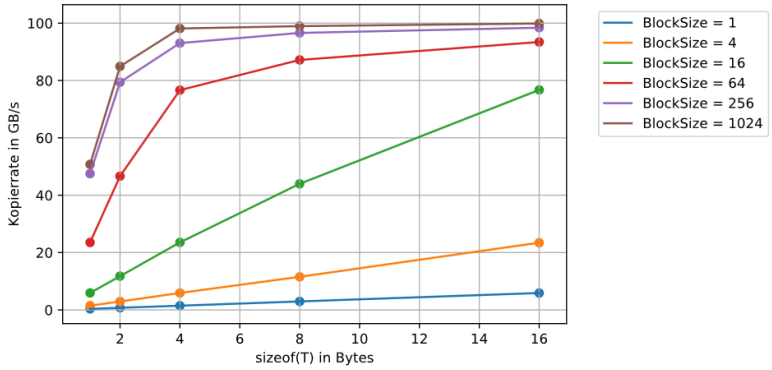

Abbildung 3: Variation über `BlockSize` und `sizeof(T)`, `numBlocks` = 16384

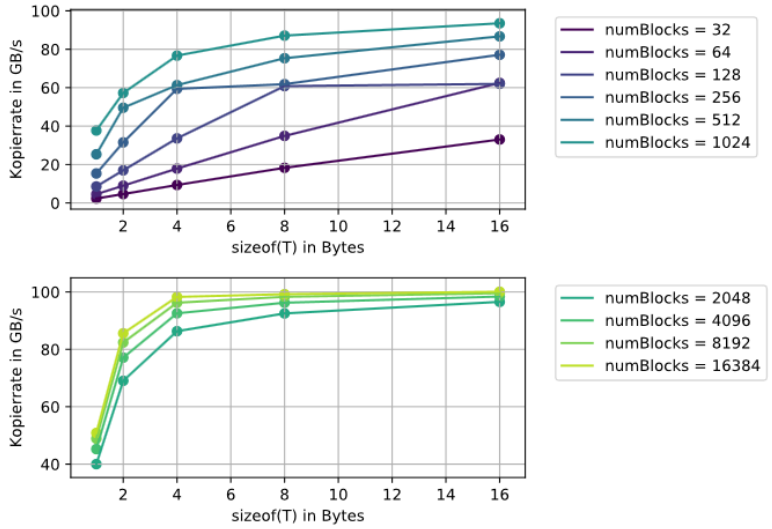

Abbildung 4: Variation über `numBlocks` und `sizeof(T)`, `BlockSize` = 1024

3 Strided Access

Der Strided Access ist wie folgt definiert:

```
template<typename T>
__global__
void copyKernel(T* out, T* in, int stride) {
        unsigned id = threadIdx.x + blockIdx.x * blockDim.x;

        out[id*stride] = in[id*stride];
}
```

Das heißt, der Zugriff erfolgt nicht mehr auf jedes Element konsekutiv hintereinander, sondern nur auf jedes N-te Element. Zusätzlicher Parameter ist dann der `stride`, der festlegt, wie viele Buffer-Einträge übersprungen werden. Bei elementarem Zugriff, wie er zum Beispiel in einer Turingmaschine definiert ist, sollte sich keine Änderung der Kopierrate ergeben, sofern die nicht bearbeiteten Elemente nicht mitgerechnet werden. Beim der Messung zeigt sich aber, dass deutliche Cashe-Effekte auftreten. Bei Messungen zeigt sich ein nahezu exponentieller Abfall der Kopierrate mit steigendem `stride`, für den die beiden Messungen Abbildung 5 und Abbildung 6 beispielhaft stehen. Bei `sizeof(T)` $\cdot$ `stride` $= 32$ und `sizeof(T)` $\cdot$ `stride` $= 64$ ergeben sich jeweils zwei fast gleiche Werte, und ab `sizeof(T)` $\cdot$ `stride` $= 2048$ fällt die Kopierrate wenn überhaupt nur noch vernachlässigbar ab (bei diesen beiden Messungen wie auch bei allen anderen).

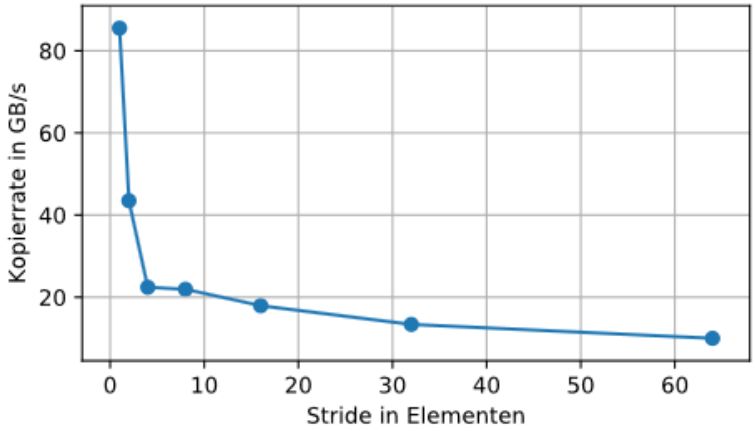

Abbildung 5: Strided Access: Variation über `stride`, `BlockSize` $= 256$, `numBlocks` $=$ 4096, `T`: `int2` (`sizeof(int2)` $= 8$)

Literatur

[1] Wikimedia: “Hundsrose.jpg”, commons.wikimedia.org/wiki/File:Hundsrose.jpg, viewed on 01/13/19

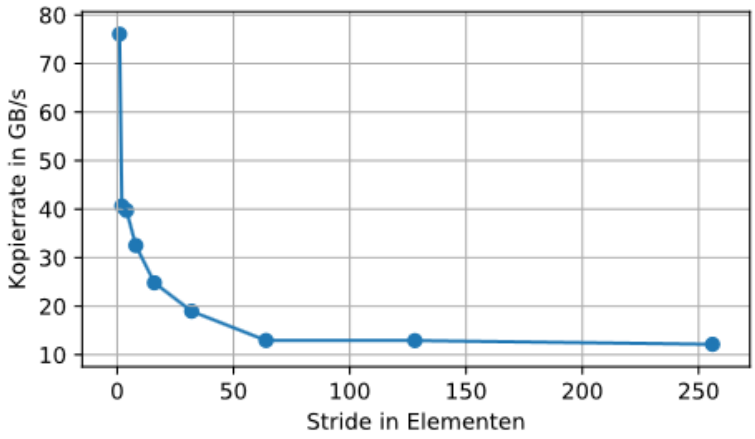

Abbildung 6: Strided Access: Variation über `stride`, `BlockSize` = 128, `numBlocks` = 2048, `T`: `int4` (`sizeof(int4)` = 16)

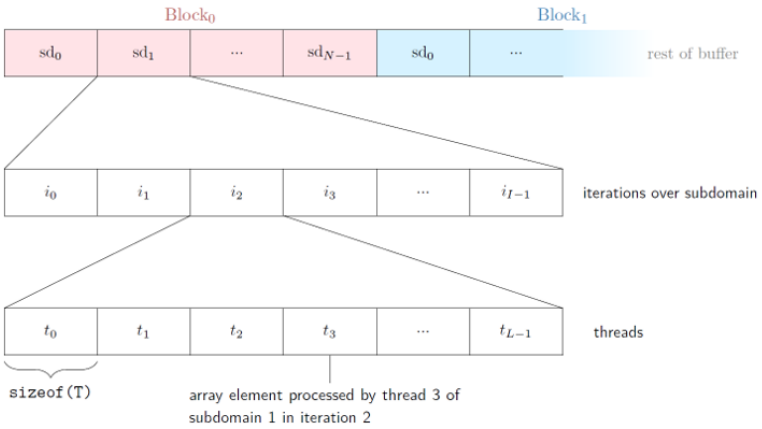

Abbildung 7:

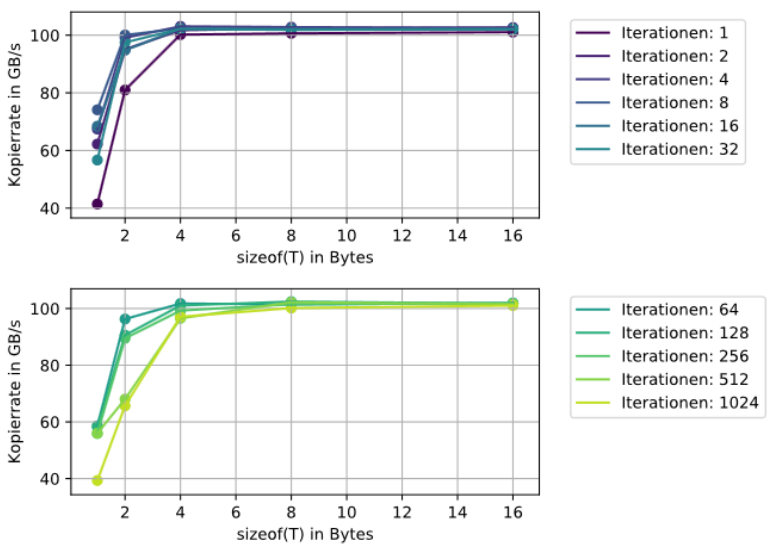

Abbildung 8:

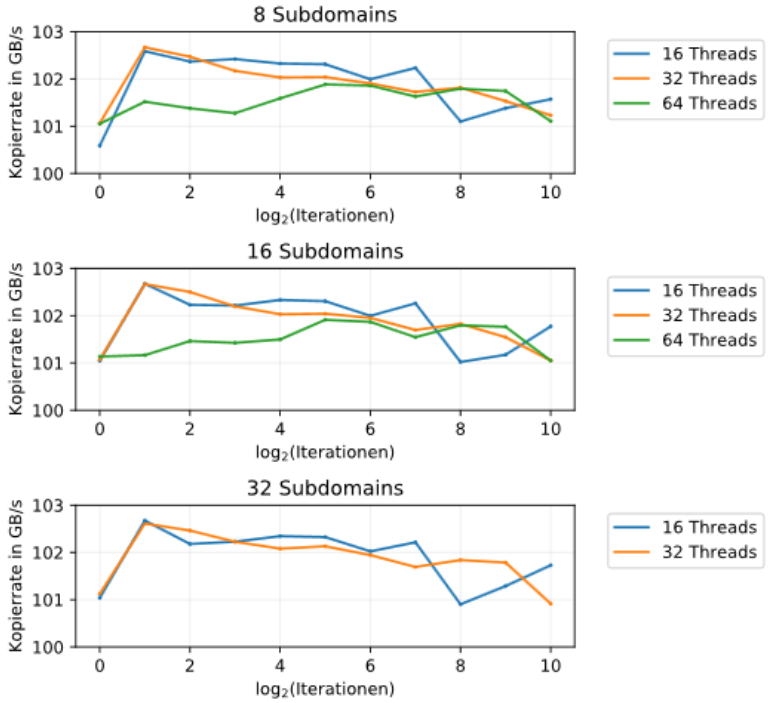

Abbildung 9:

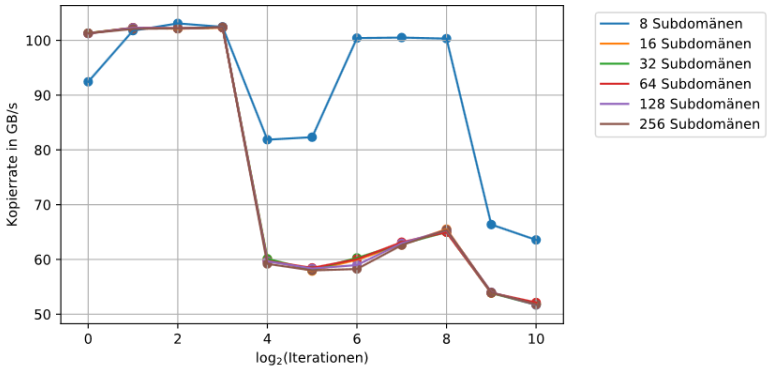

Abbildung 10:

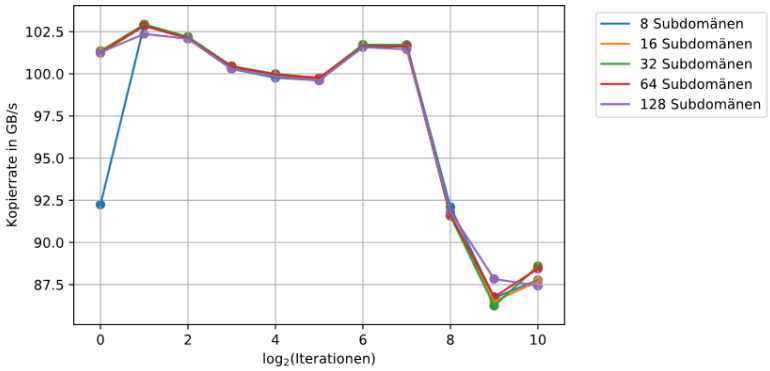

Abbildung 11:

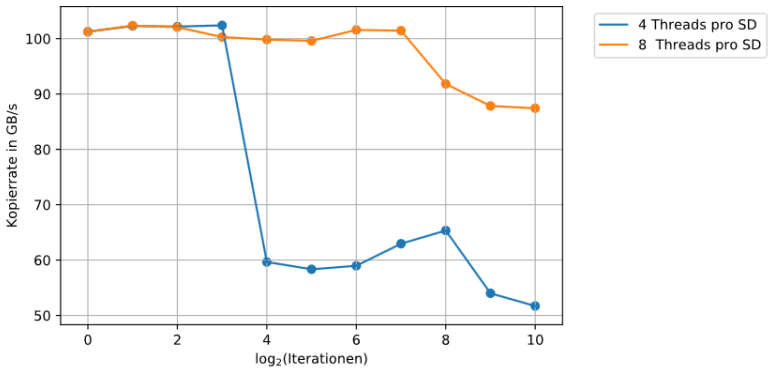

Abbildung 12:

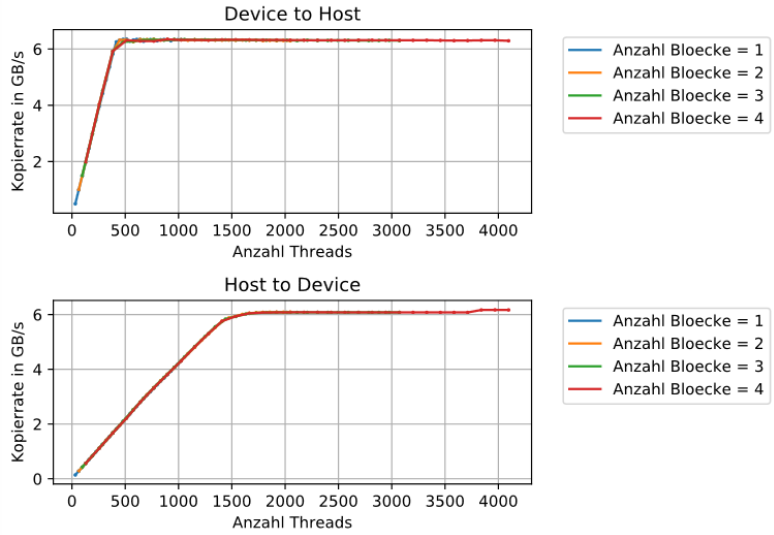

Abbildung 13:

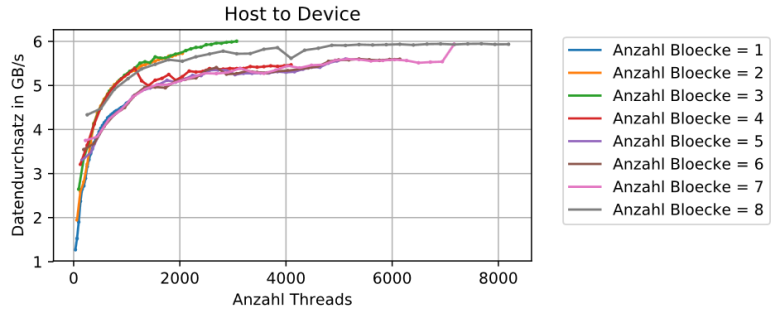

Abbildung 14:

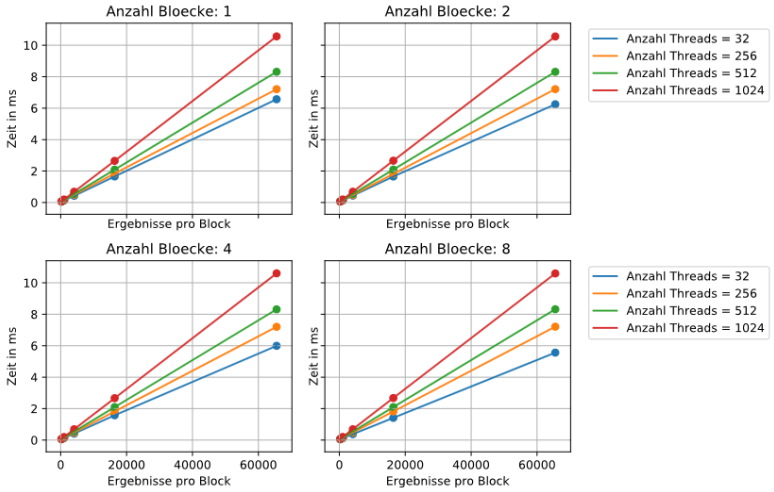

Abbildung 15:

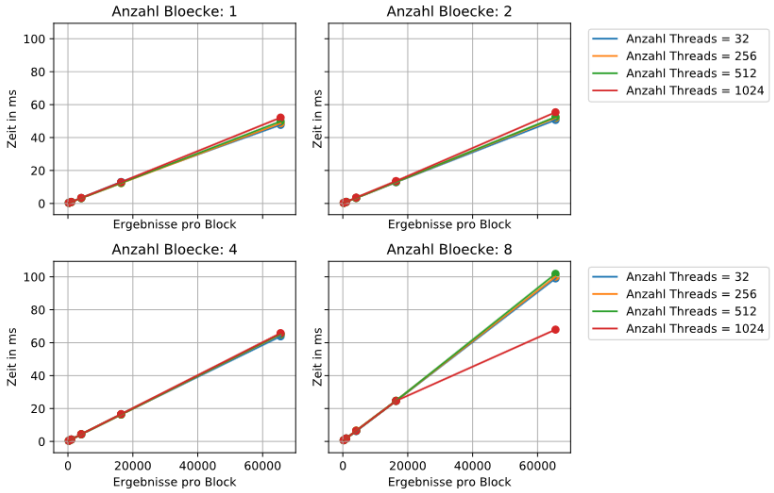

Abbildung 16:

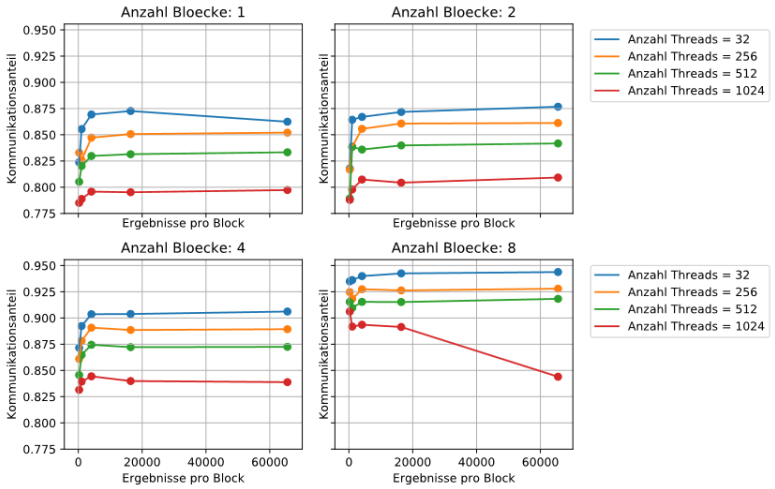

Abbildung 17:

[2] Wikimedia: “Baum im Sossusvlei.jpg”, commons.wikimedia.org/wiki/File:Baum_im_Sossusvlei.jpg, viewed on 01/13/19

[3] “Make Something Unreal”, unrealengine.com/en-US/what-is-unreal-engine-4, viewed on 01/13/19

[4] Minecraft Wiki: “Tree’, minecraft.gamepedia.com/Tree, viewed on 01/13/19

[5] Wikipedia: “Spore’, en.wikipedia.org/wiki/Spore_(2008_video_game), viewed on 01/13/19

[6] Cell Lab website, cell-lab.net/, viewed on 01/13/19

[7] Eco website, strangeloopgames.com/eco/, viewed on 01/13/19

[8] SpeedTree website, store.speedtree.com/, viewed on 01/13/19

[9] UE API Documentation (api.unrealengine.com/): “UInstancedStaticMeshComponent”, viewed on 01/13/19

[10] UE Documentation (docs.unrealengine.com/en-us/): “Blueprint Spline Components Overview”, viewed on 01/13/19

[11] Johan Knutzen, “Generating Climbing Plants Using L-Systems”, viewed on 01/13/19