

Insert Title here



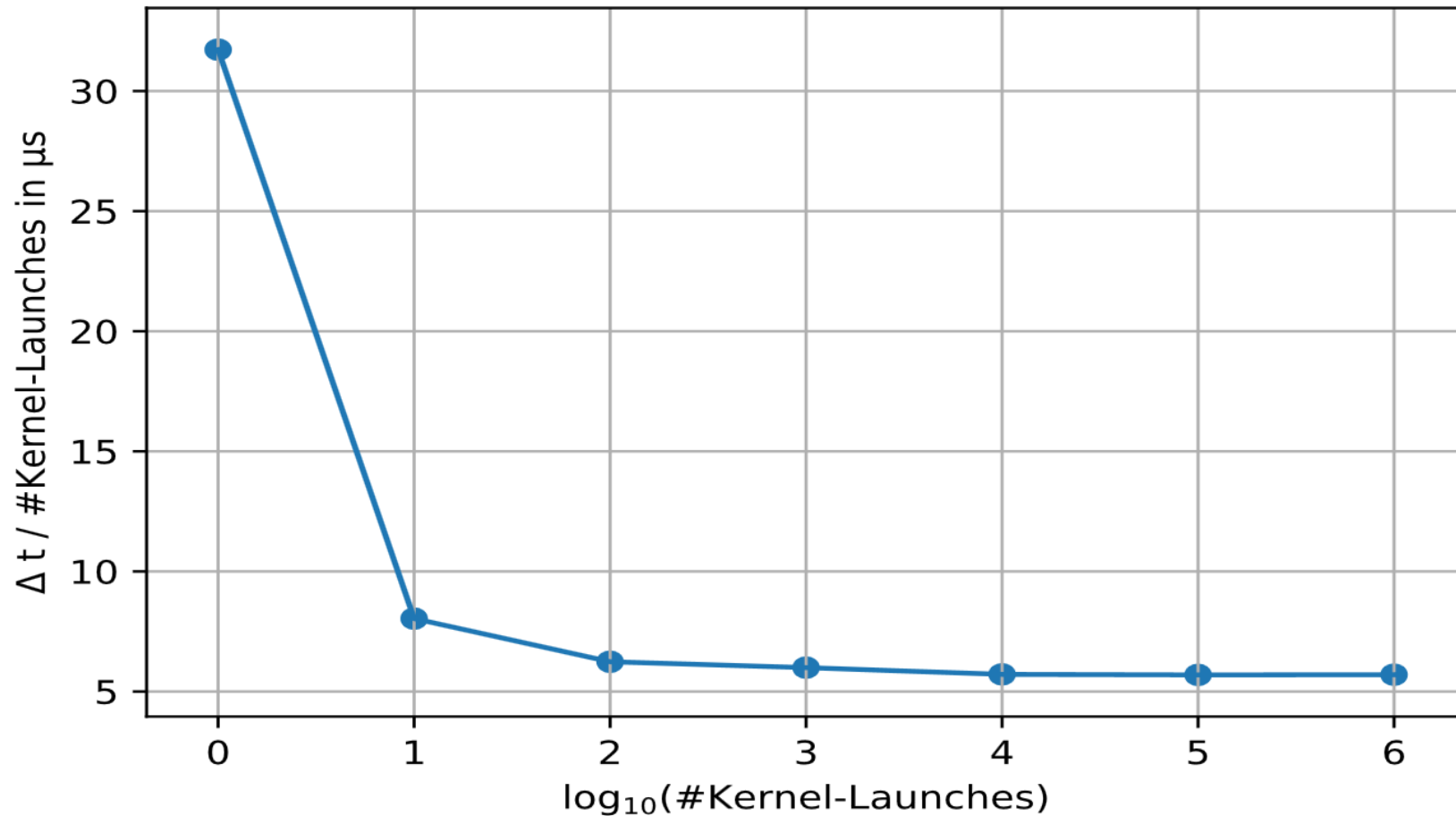
Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

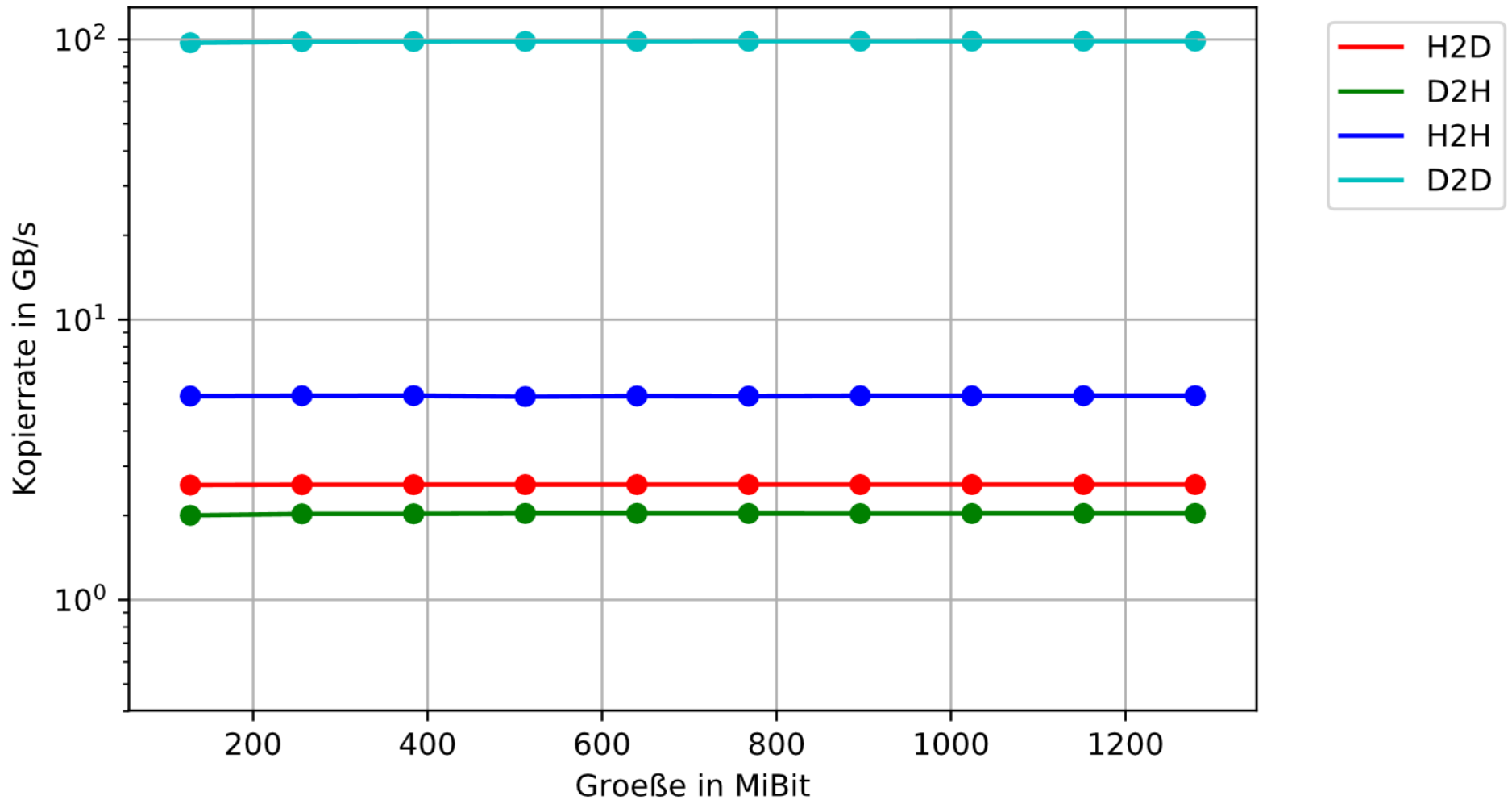
GTX 1070

- .Cores 1920
- .Basistaktung 1506 MHz
- .Standard-Speicherkonfiguration 8 GB GDDR5
- .Speicherbandbreite 256 GB/s

Empty Kernel – Startup Cost



Datenkopiererrate CudaMemcpy



Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

Copy Kernel

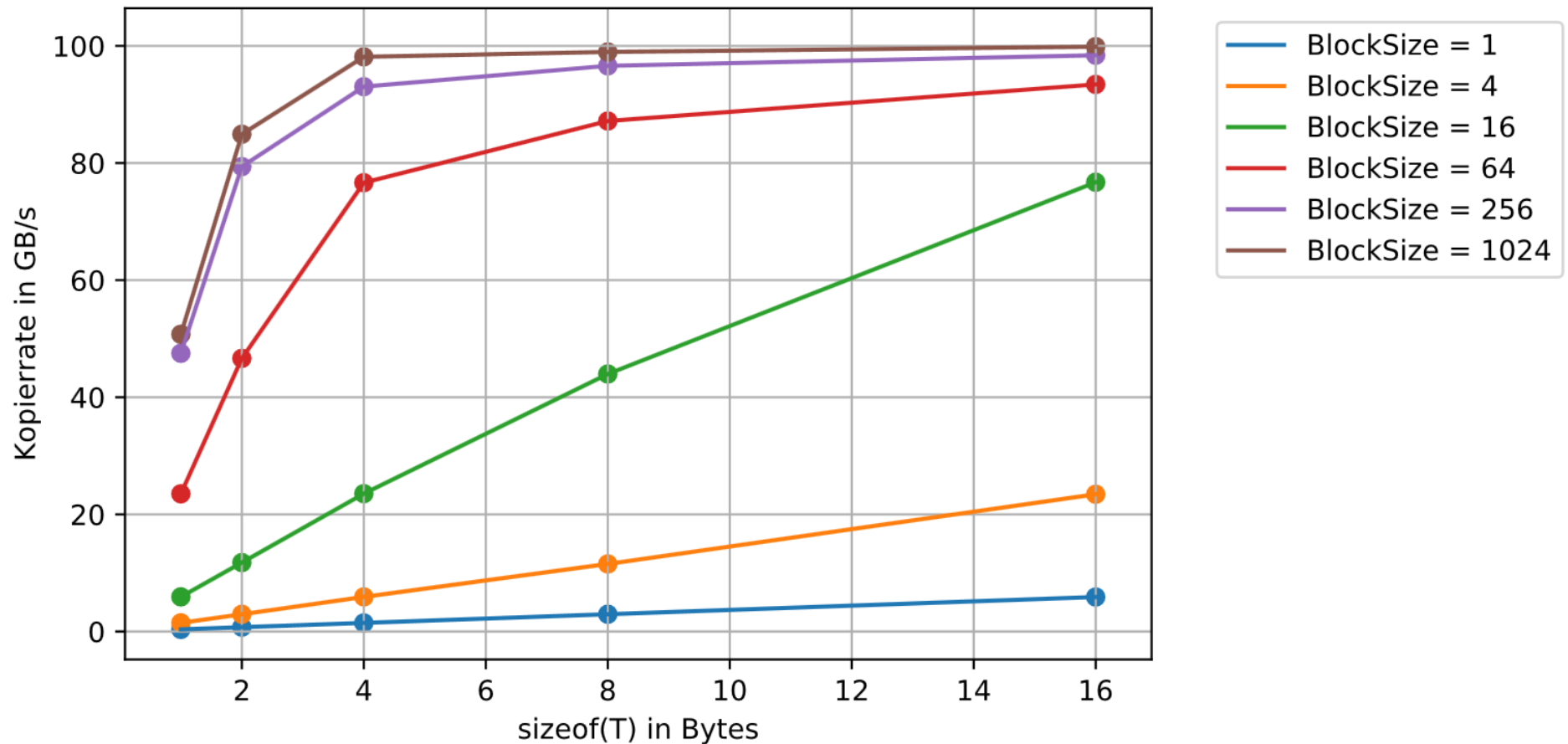
```
//Kernel definition
template<typename T>
__global__
void copyKernel(T* out, T* in) {
    unsigned id = threadIdx.x + blockIdx.x * blockDim.x;

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

.Parameter:

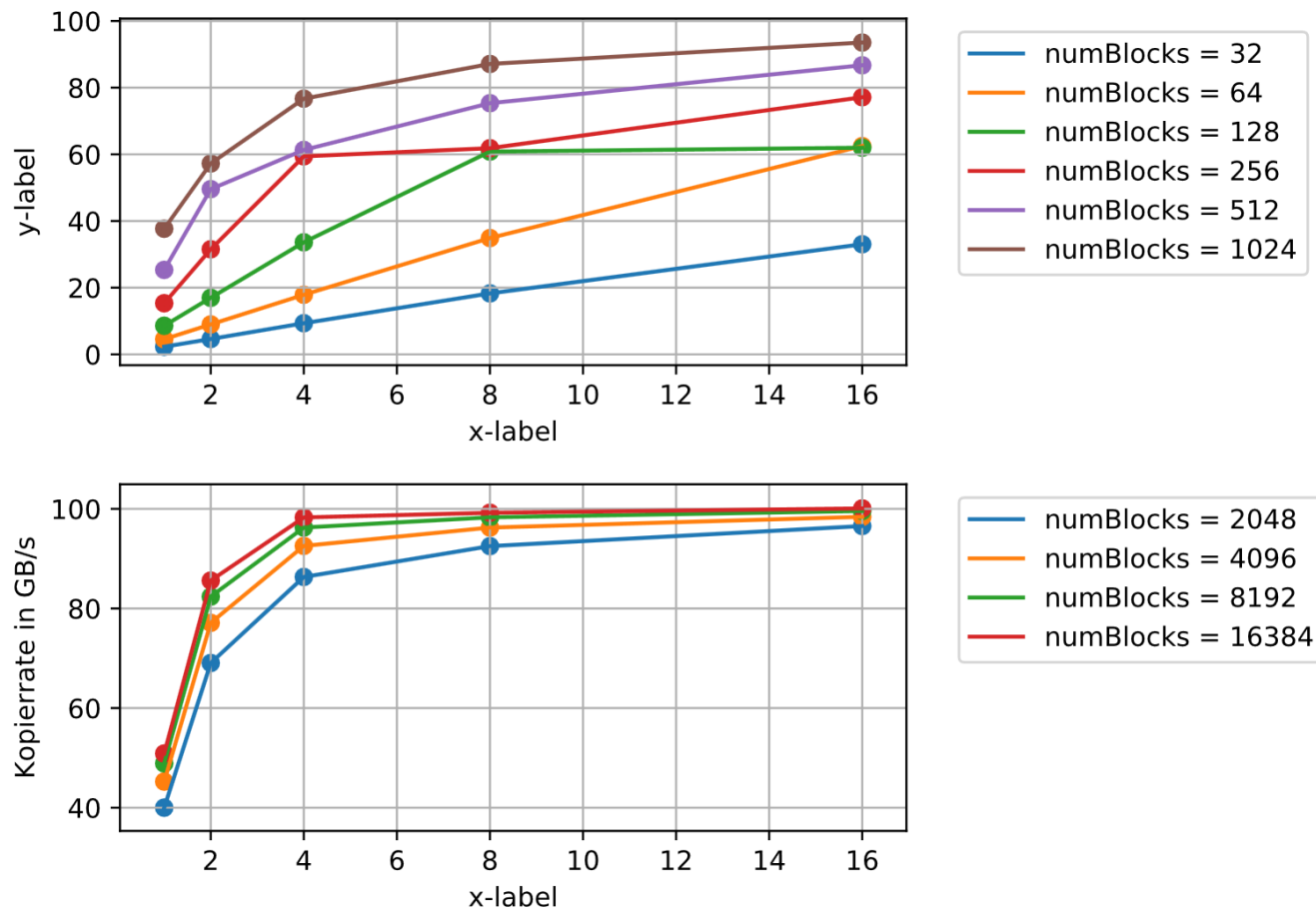
- ❑ Blockgröße
- ❑ Anzahl der Blöcke
- ❑ Zugriffstypen T (zB. char, int...)

Copy Kernel – BlockSize & sizeof(T)



Copy Kernel – Number of Blocks

.BlockSize = 1024



Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

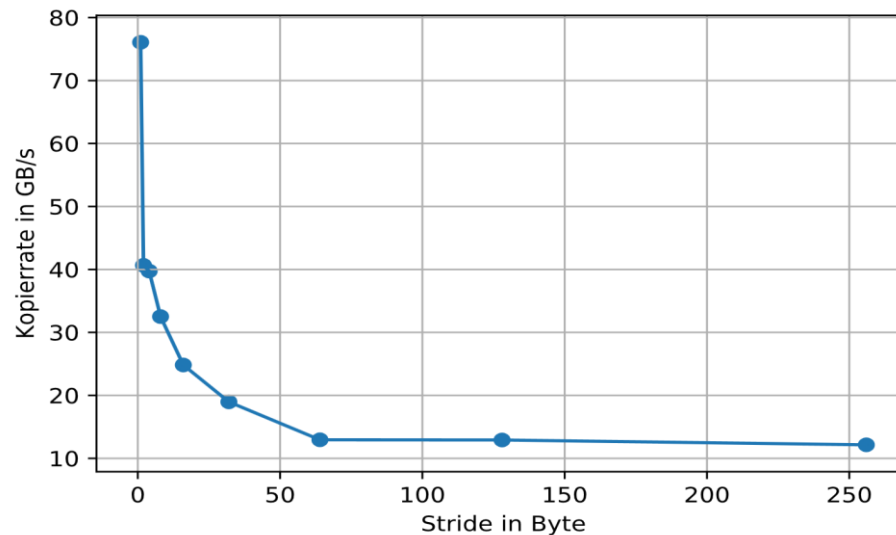
Strided Access

```
//Kernel definition
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];
}
```

- Zugriff nicht mehr auf jedes Element konsekutiv hintereinander, sondern auf jedes N-te Element
- Bei elementarem Zugriff sollte sich die Kopierrate nicht ändern (Übersprunge Elemente werden nicht mitgezählt)

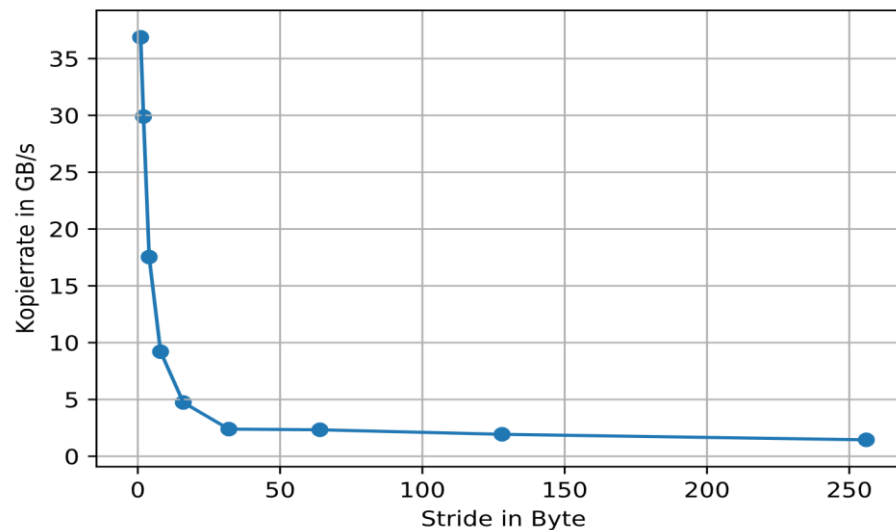
Strided Access

- .Blockgröße = 128 Threads
- .Blockanzahl = 2048
- .Zugriffstyp: int4 (sizeof(int4) = 16)
- .Stride = 1,2,4,8,16,32,64,128,256



Strided Access II

- .Blockgröße = 256 Threads
- .Blockanzahl = 8192
- .Zugriffstyp: char (sizeof(char) = 1)
- .Stride = 1,2,4,8,16,32,64,128,256



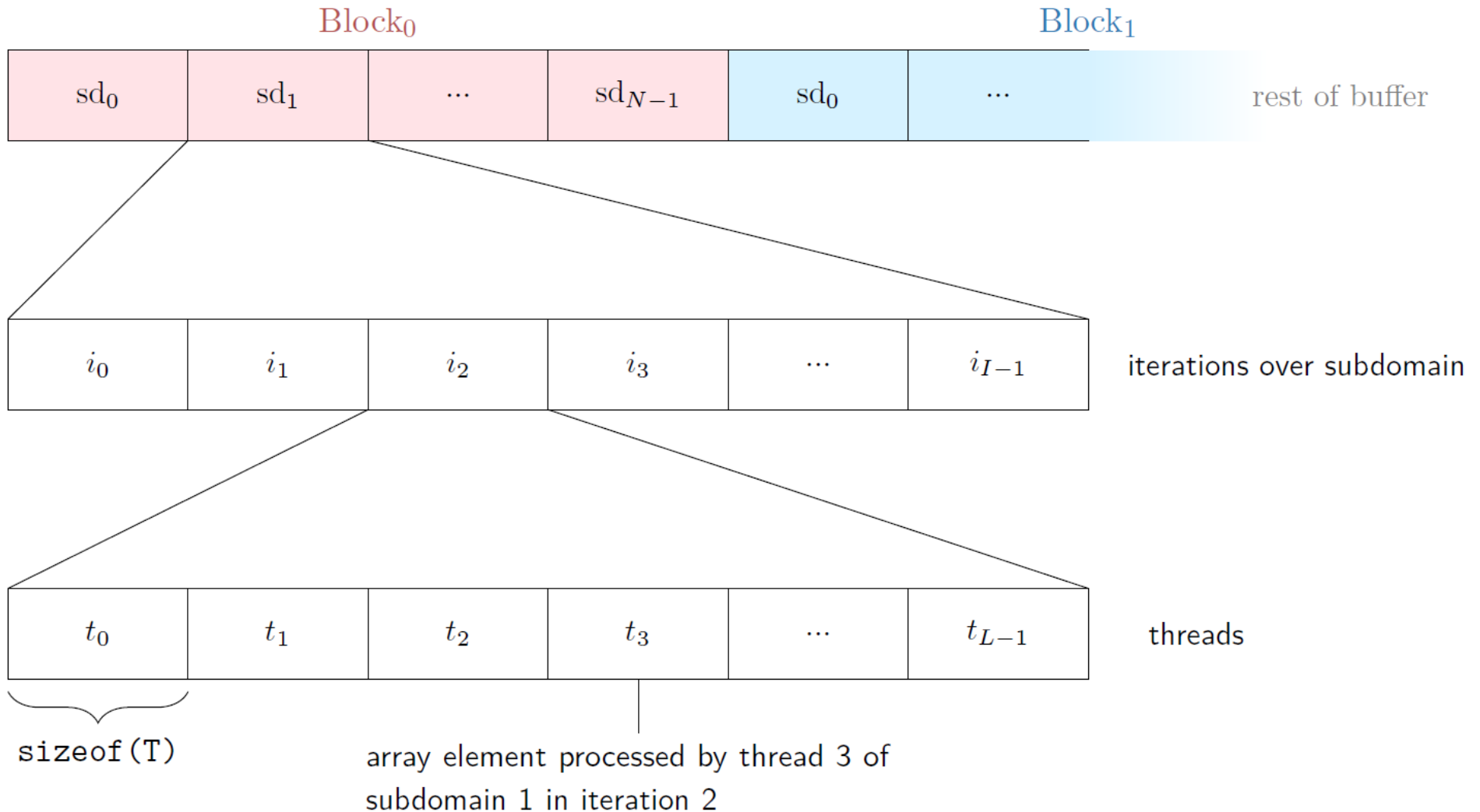
Strided Access: Beobachtungen

- Zwei fast gleiche Werte jeweils bei $\text{sizeof}(T) * \text{stride} = 32$ und 64
- Ab $\text{sizeof}(T) * \text{stride} = 1024$ wenn überhaupt nur noch vernachlässigbares Nachlassen der Kopiertrate

Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

Offset Access – Zugriffsmuster



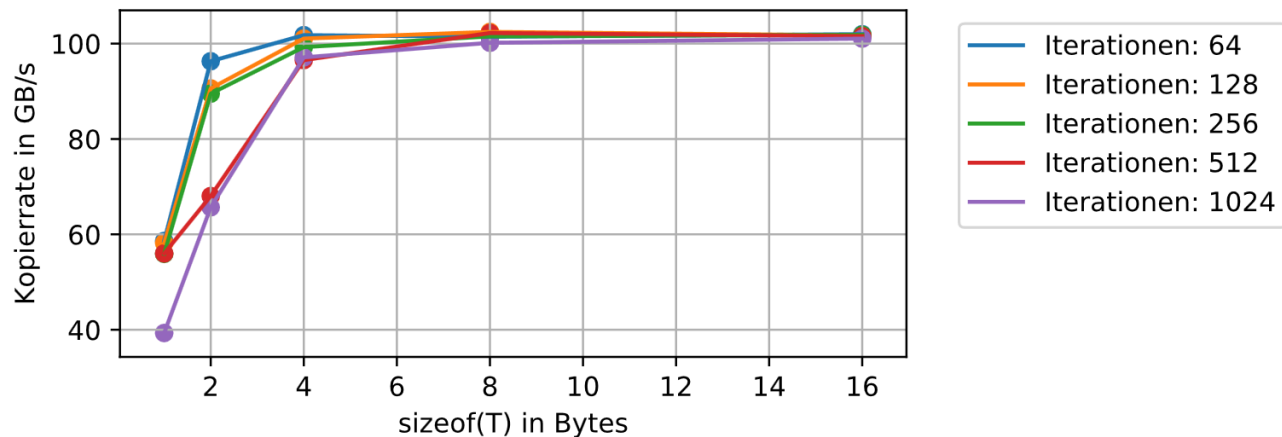
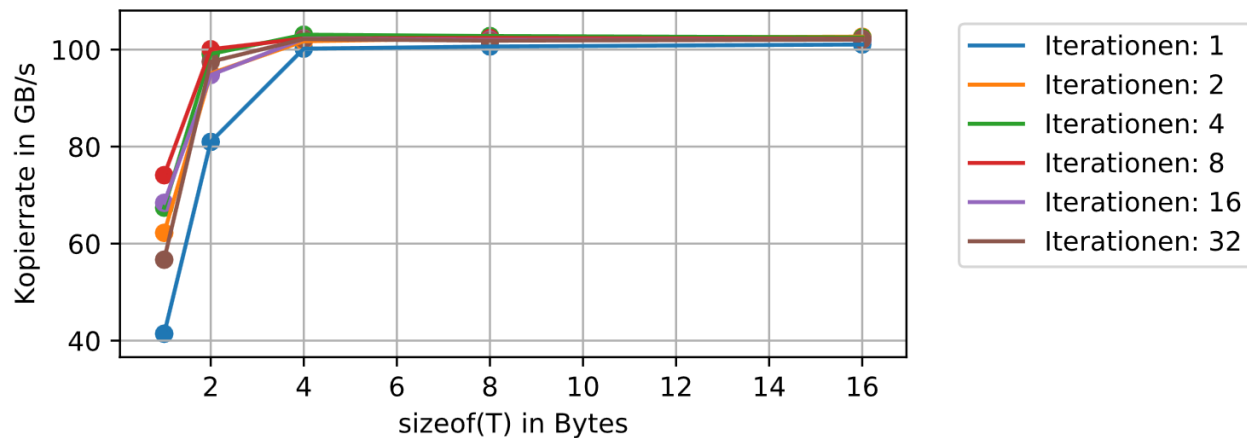
Offset Access

```
//Kernel definition
template<typename T>
__global__
void offtKernel (T* out,
                 T* in,
                 const unsigned int sd_size,
                 const unsigned int block_size,
                 const unsigned int I,
                 const unsigned int L)
{
    const unsigned int sd_id = static_cast<int> (threadIdx.x / L); //automatically rounded down in int arithmetics
    const unsigned int id = threadIdx.x - sd_id * L;
    const unsigned int sd_start = blockIdx.x * blockDim.x * I + sd_id * L * I;

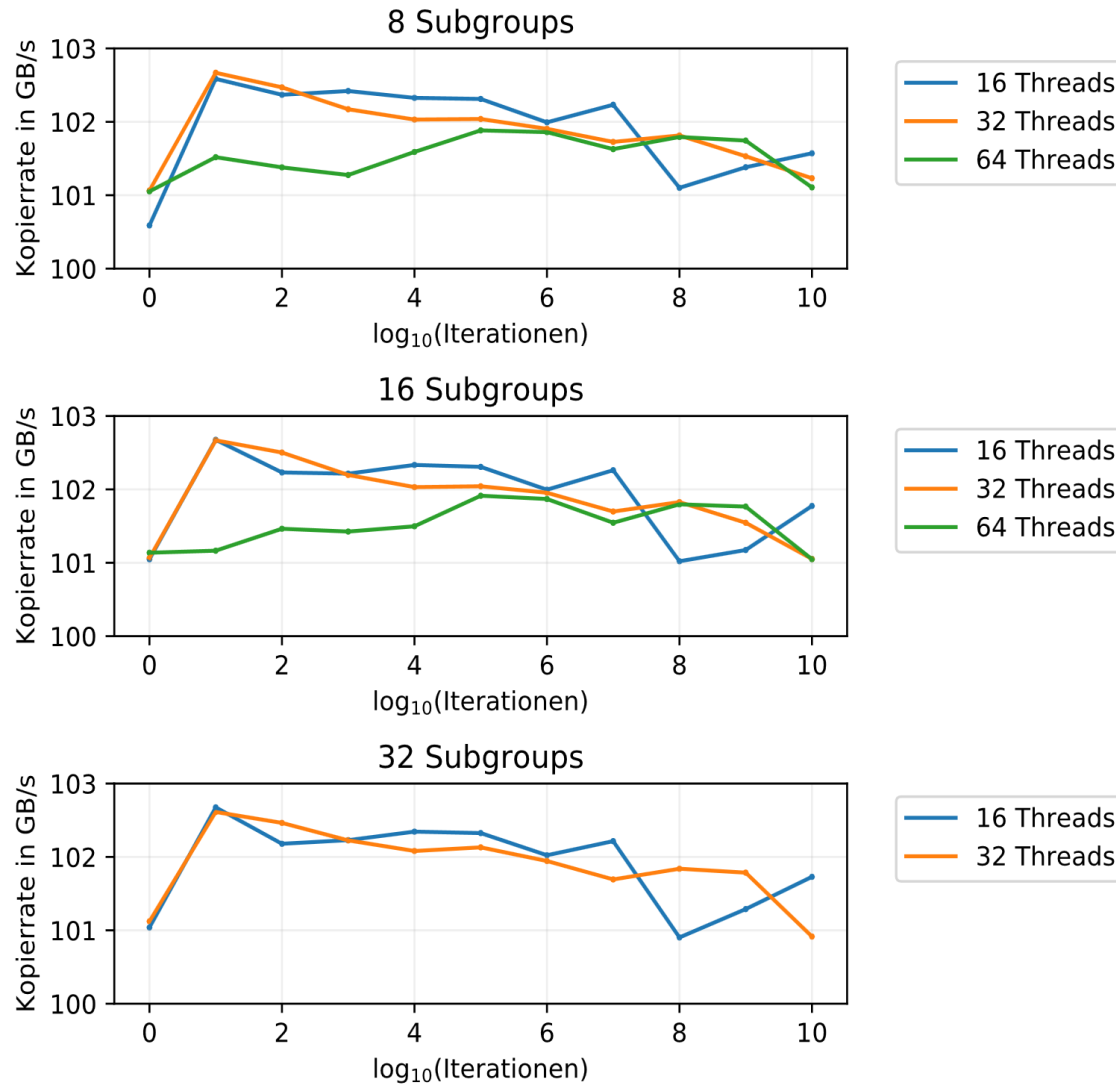
    for (unsigned int i = 0; i < I; i++)
    {
        const unsigned int el_id = sd_start + i * L + id;
        ((T*) out)[el_id] = ((T*) in)[el_id];
    }
}
```

Typ und Iterationen

- 32 Threads pro Subdomäne, 16 SD pro Block

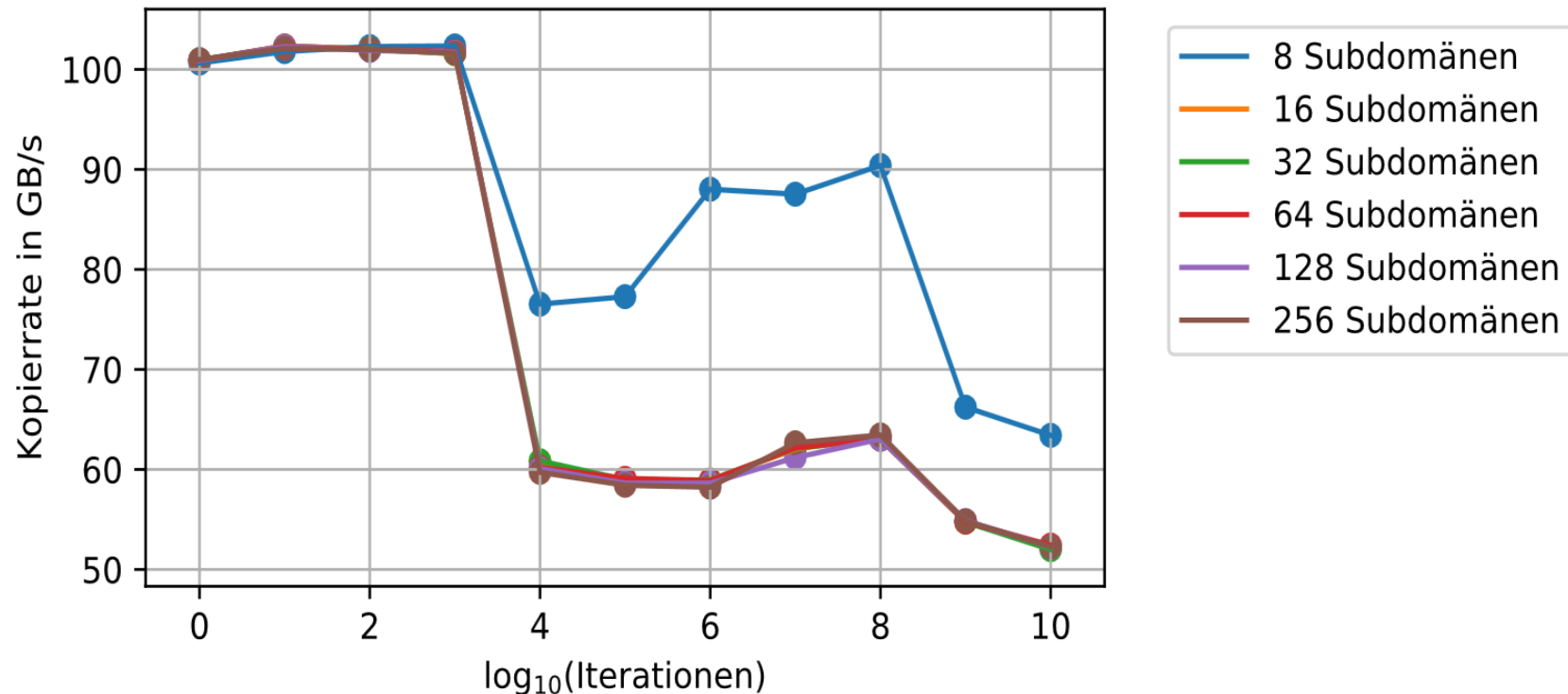


Subgroup Anzahl pro Block



Variiere Subdomänen pro Block

.4 Threads pro SD, sizeof(T) = 16



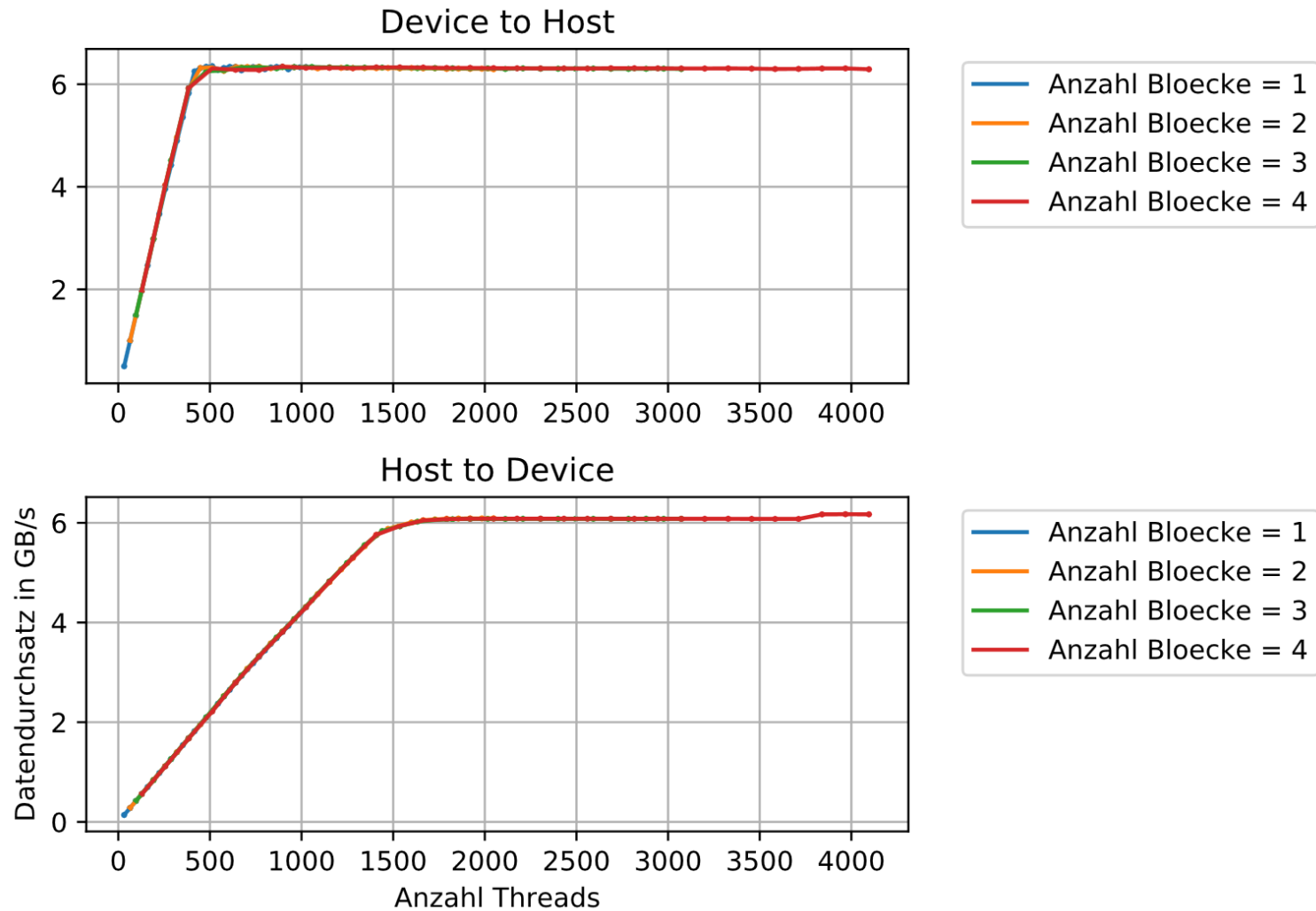
Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

Zugriffsmuster: Copy Kernel

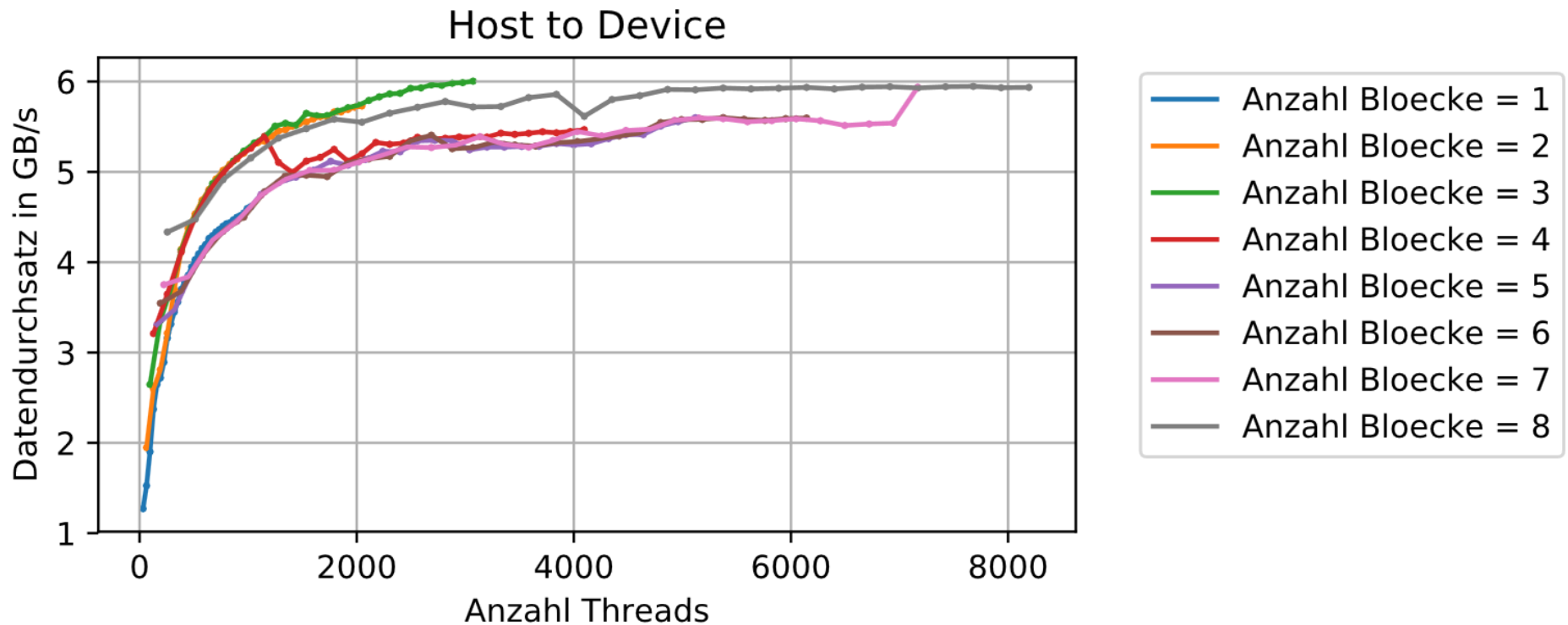
```
//Kernel definition
template<typename T>
__global__
void copyKernel (T* out,
                 T* in,
                 const unsigned int N)
{
    const unsigned int id = threadIdx.x + blockIdx.x * blockDim.x;
    for (unsigned int i= id; i < N; i = i + blockDim.x * gridDim.x)
    {
        const unsigned el_id = i;
        ((T*) out)[el_id] = ((T*) in)[el_id];
    }
}
```

CudaMalloc und CudaMallocHost



Unified Memory

.CudaMallocManaged + memset + CudaMemset



Gliederung

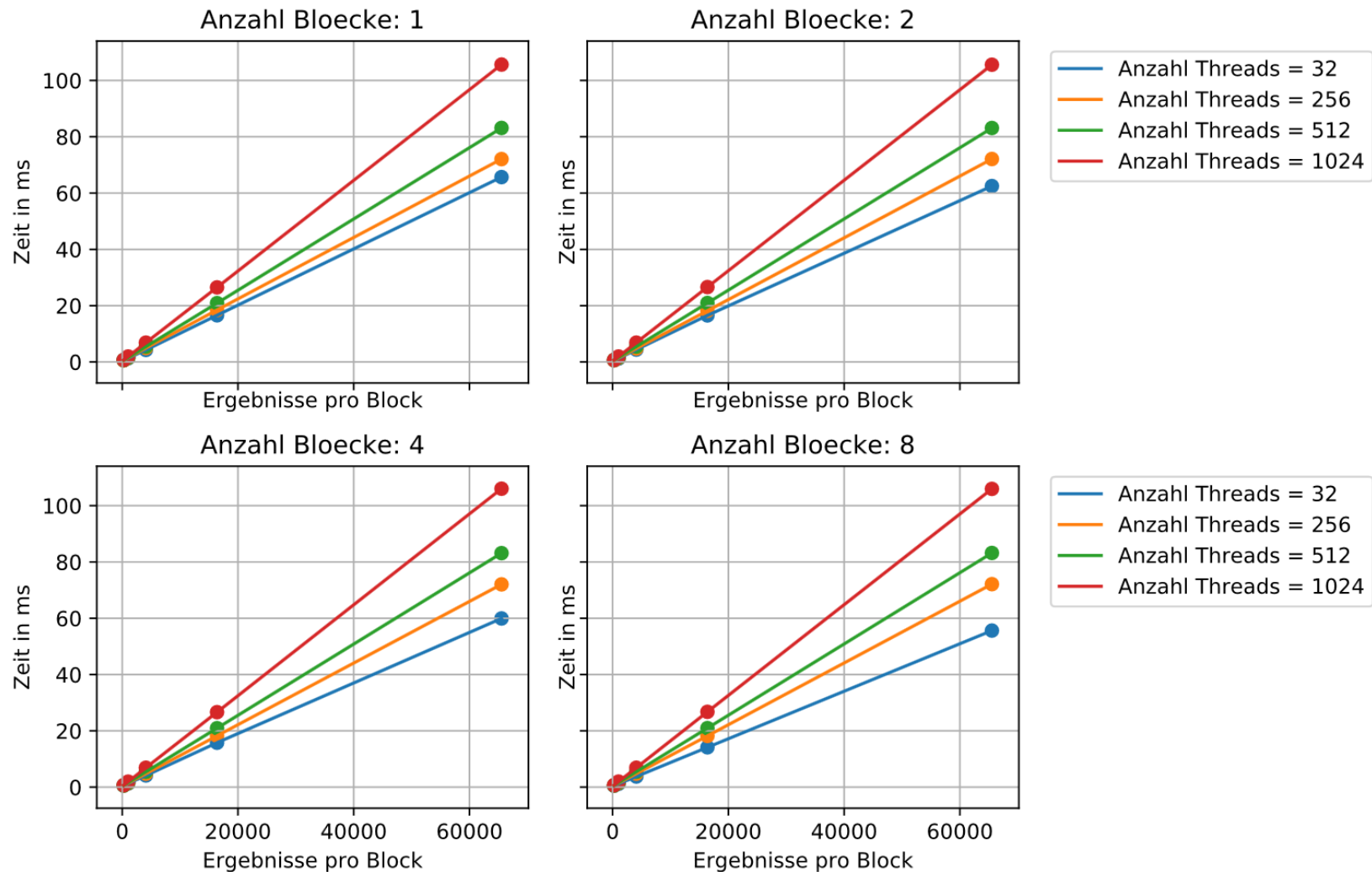
- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Kommunikation zwischen Threads

Kommunikation -Aufgabenstellung

- Was genau tat dein Code nochmal? Kann man das überhaupt in einem Satz zusammenfassen?
- Kurz gesagt ist ja die Idee darauf hinzuweisen, dass natürlich Effizienz verloren geht, wenn die Kernel aufeinander warten müssen statt einfach vor sich hin zu arbeiten

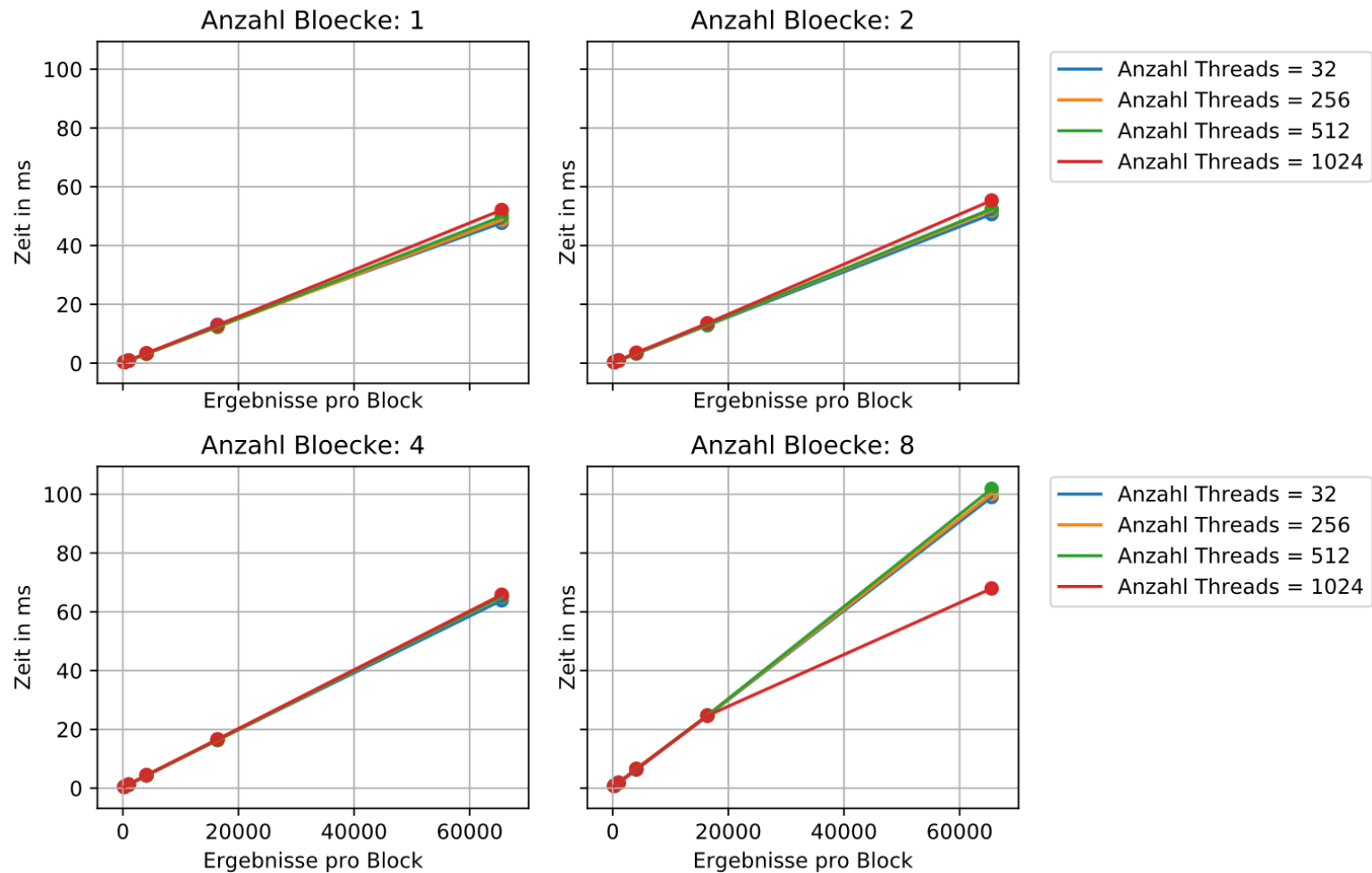
Kommunikationsmessung: Laufzeit ohne Kommunikation

•Variation Ergebnisse pro Block: 256,1024,4096,16384,65536



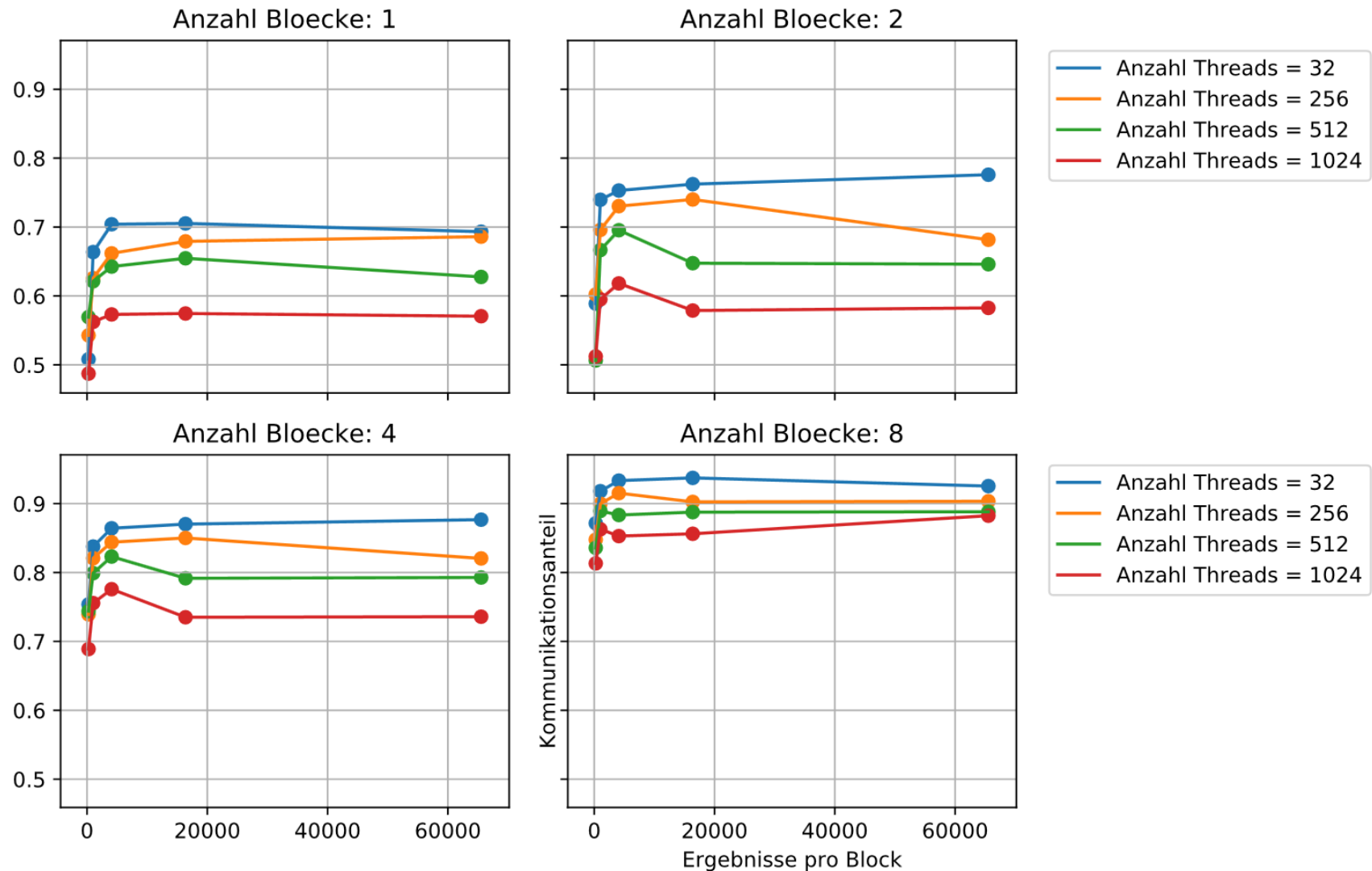
Kommunikationsmessung: Laufzeit mit Kommunikation

•Variation Ergebnisse pro Block: 256,1024,4096,16384,65536



Anteil der Kommunikation an der Gesamtzeitdauer

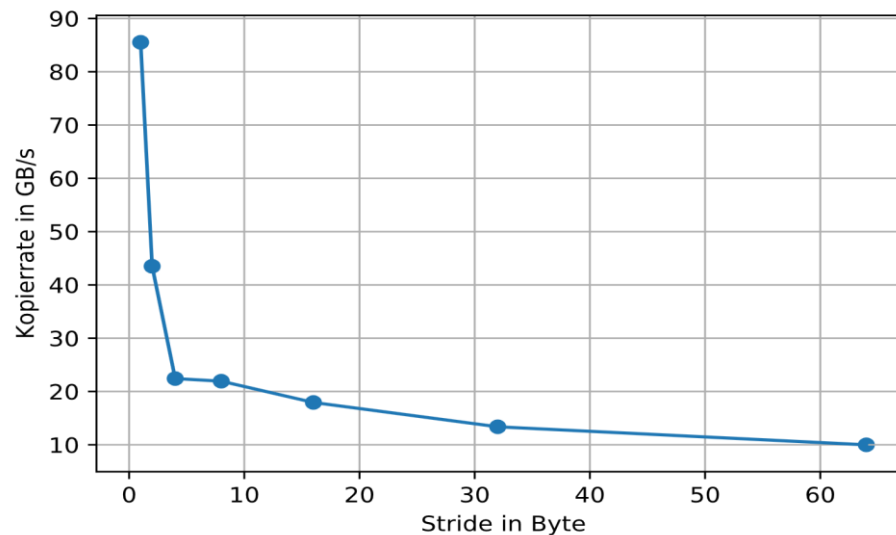
•Variation Ergebnisse pro Block: 256,1024,4096,16384,65536



Ende der Präsentation –
Fragen?

Zusatz - Strided Access III

- .Blockgröße = 256 Threads
- .Blockanzahl = 4096
- .Zugriffstyp: int2 ($\text{sizeof}(\text{int2}) = 8$)
- .Stride = 1,2,4,8,16,32,64



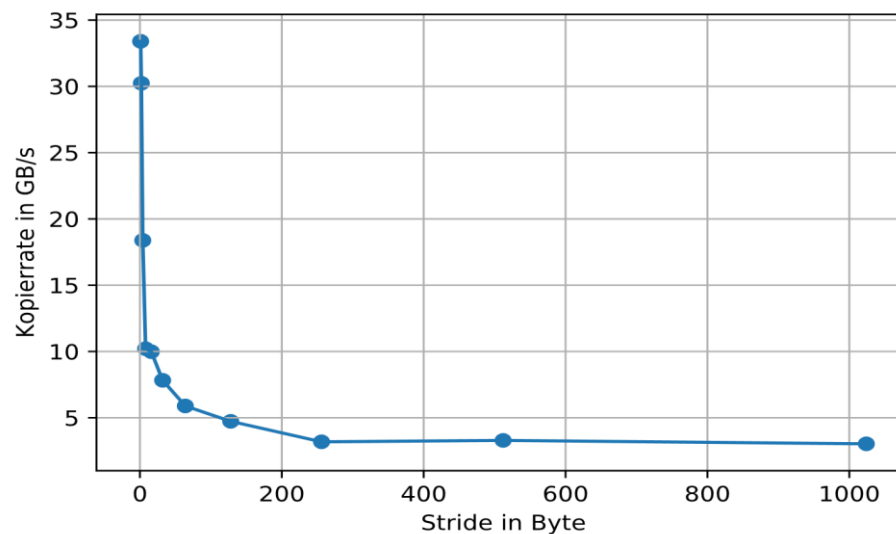
Zusatz - Strided Access IV

.Blockgröße = 128 Threads

.Blockanzahl = 2048

.Zugriffstyp: int (sizeof(int) = 4)

.Stride = 1,2,4,8,16,32,64,128,256,512,1024



Quellen

Bild GTX 1070: <https://upload.wikimedia.org/wikipedia/commons/6/62/NVIDIA-GTX-1070-FoundersEdition-FL.jpg>

Spezifikationen GTX 1070: <https://www.nvidia.com/de-de/geforce/products/10series/geforce-gtx-1070/>