

Parallele Zugriffsmuster auf GPUs



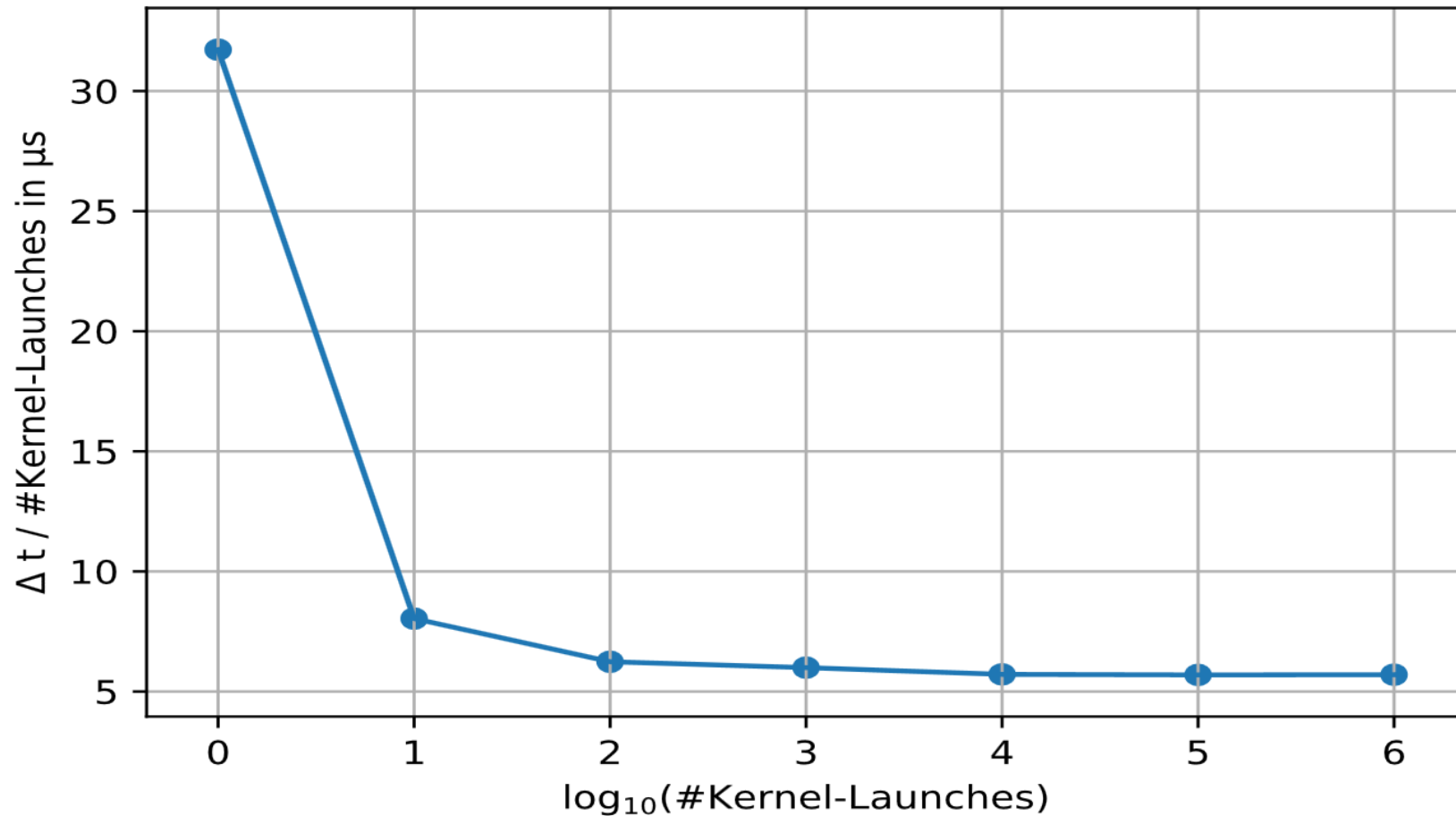
Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Producer-Consumer-Verhältnis

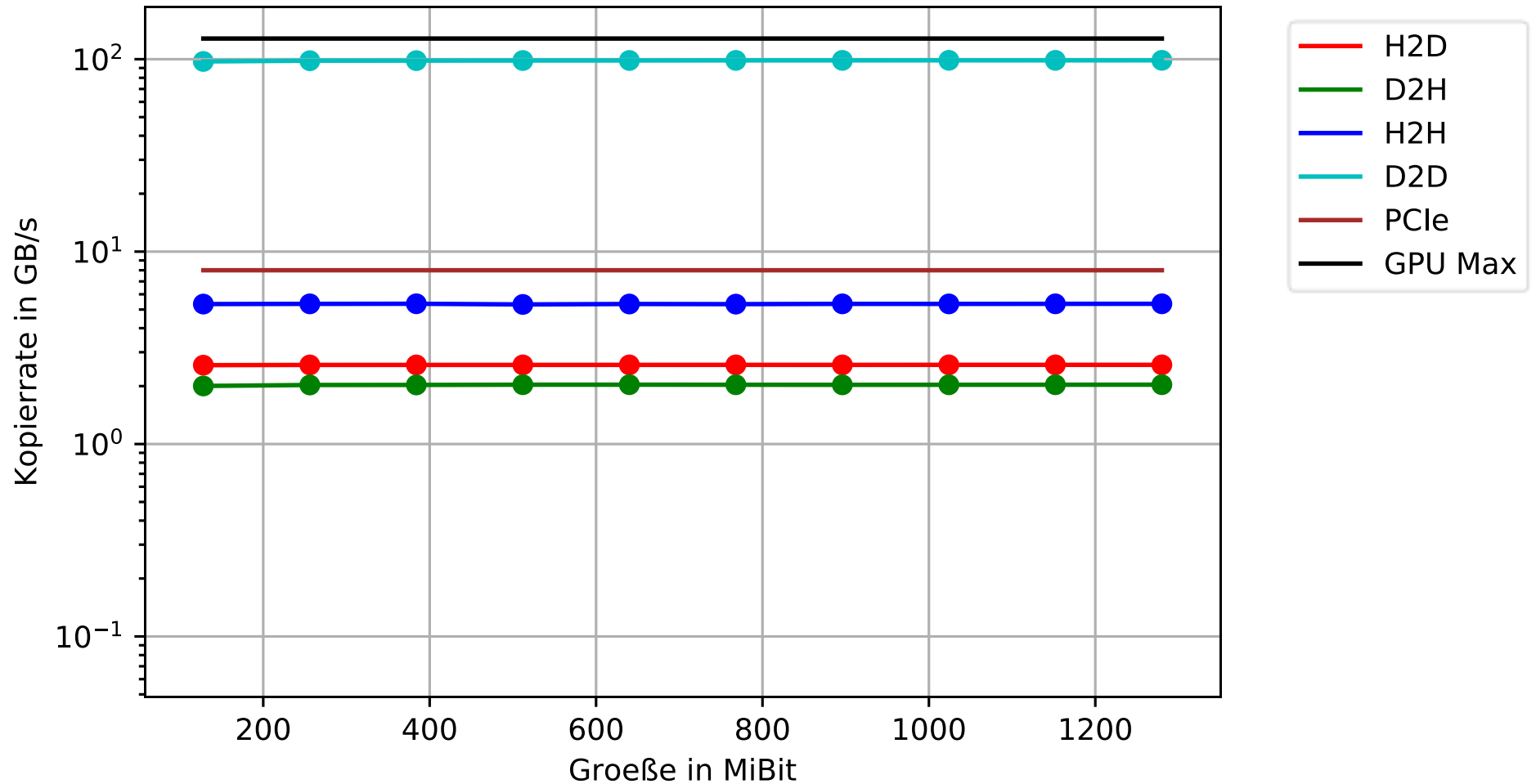
GTX 1070

- 1920 Cores, Warp: 32 Threads
- Maximale Taktung laut „nvidia-smi“: 1708 MHz
- Speicherkonfiguration 8 GB GDDR5
- Herstellerangabe Durchsatz 256 GB/s
- PCIe 3.0 fähig, in der Messung wurde aber nur PCIe 2.0 x16 verwendet
- Maximale Kopiertrate PCIe 2.0 x16: 8.0 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) Producer-Consumer-Verhältnis

Copy Kernel

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

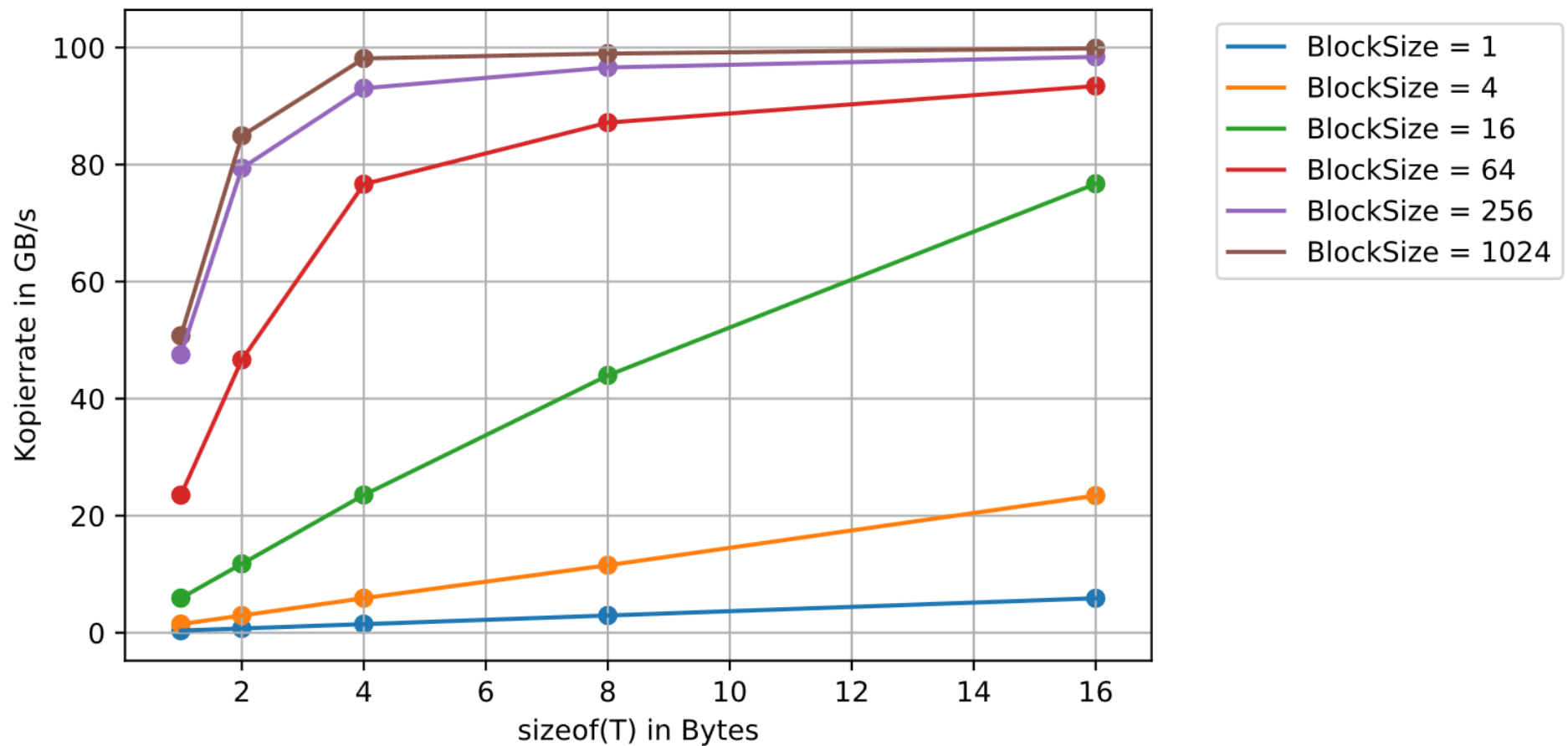
    out[id] = in[id];
}
```

Parameter:

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

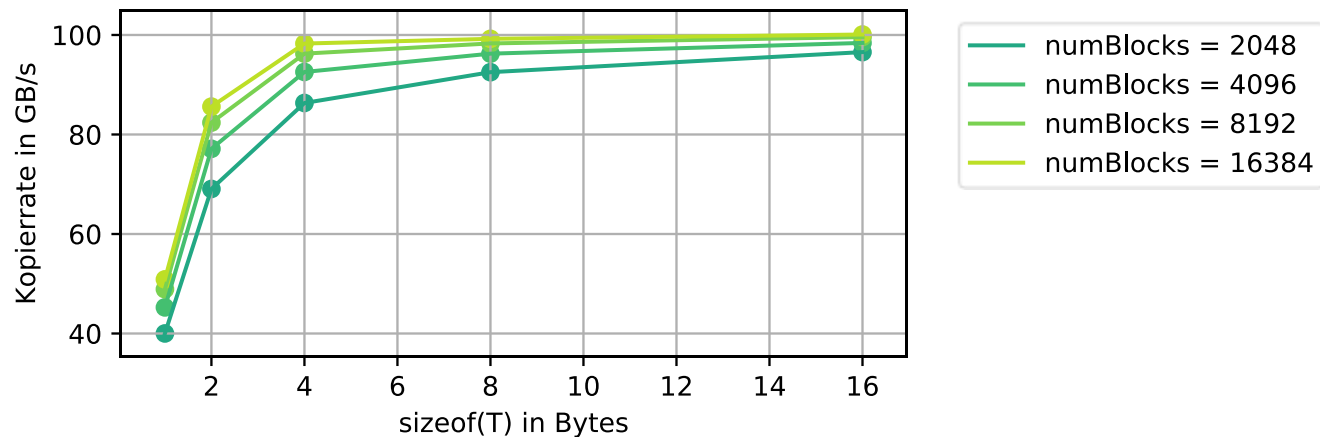
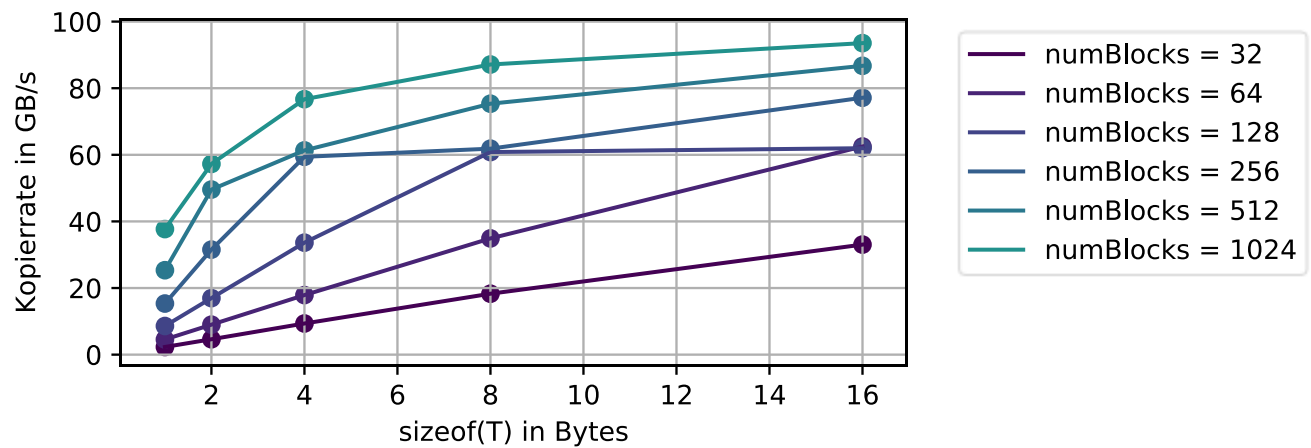
Copy Kernel – BlockSize & sizeof(T)

Anzahl Blöcke = 16384



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) Producer-Consumer-Verhältnis

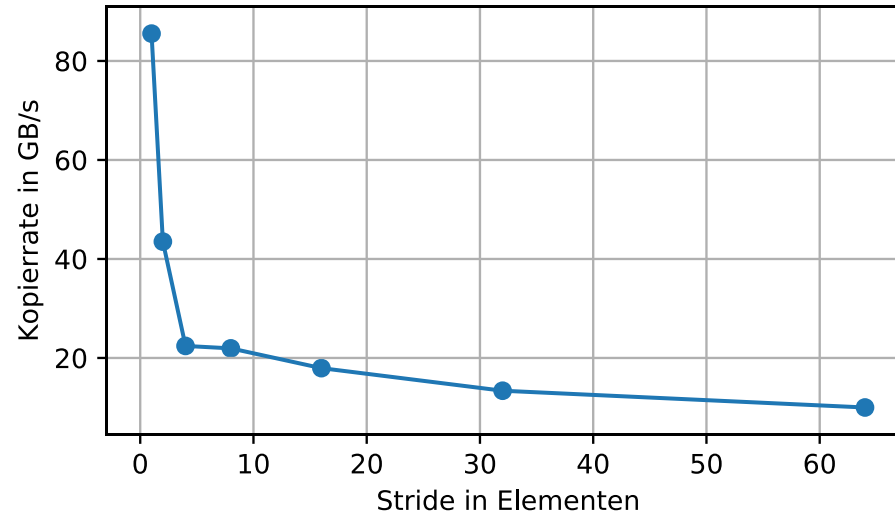
Strided Access

```
//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 (Wenn übersprungene Elemente nicht mitgezählt werden)
- Cache-Effekte sind zu erwarten

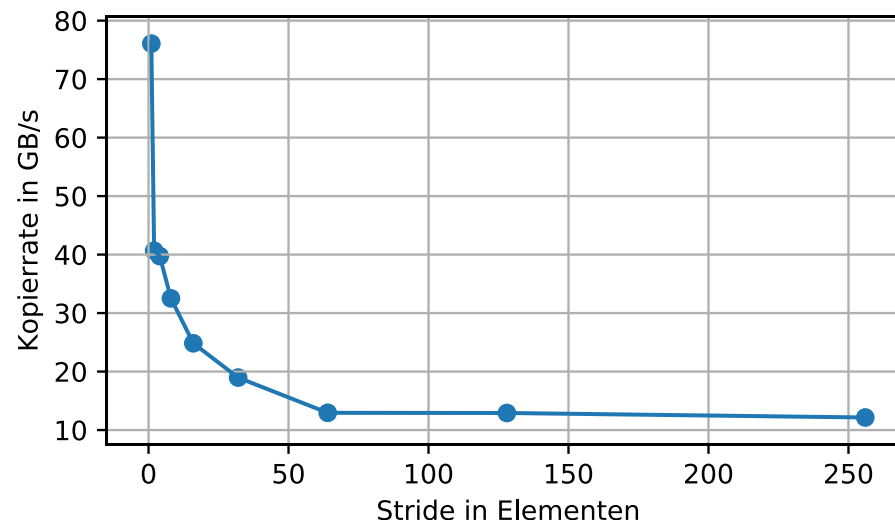
Strided Access I

- Blockgröße = 256 Threads
- Blockanzahl = 4096
- Zugriffstyp: int2 (sizeof(int2) = 8)
- stride = 1,2,4,8,16,32,64



Strided Access II

- Blockgröße = 128 Threads
- Blockanzahl = 2048
- Zugriffstyp: int4 ($\text{sizeof}(\text{int4}) = 16$)
- stride = 1, 2, 4, 8, 16, 32, 64, 128, 256



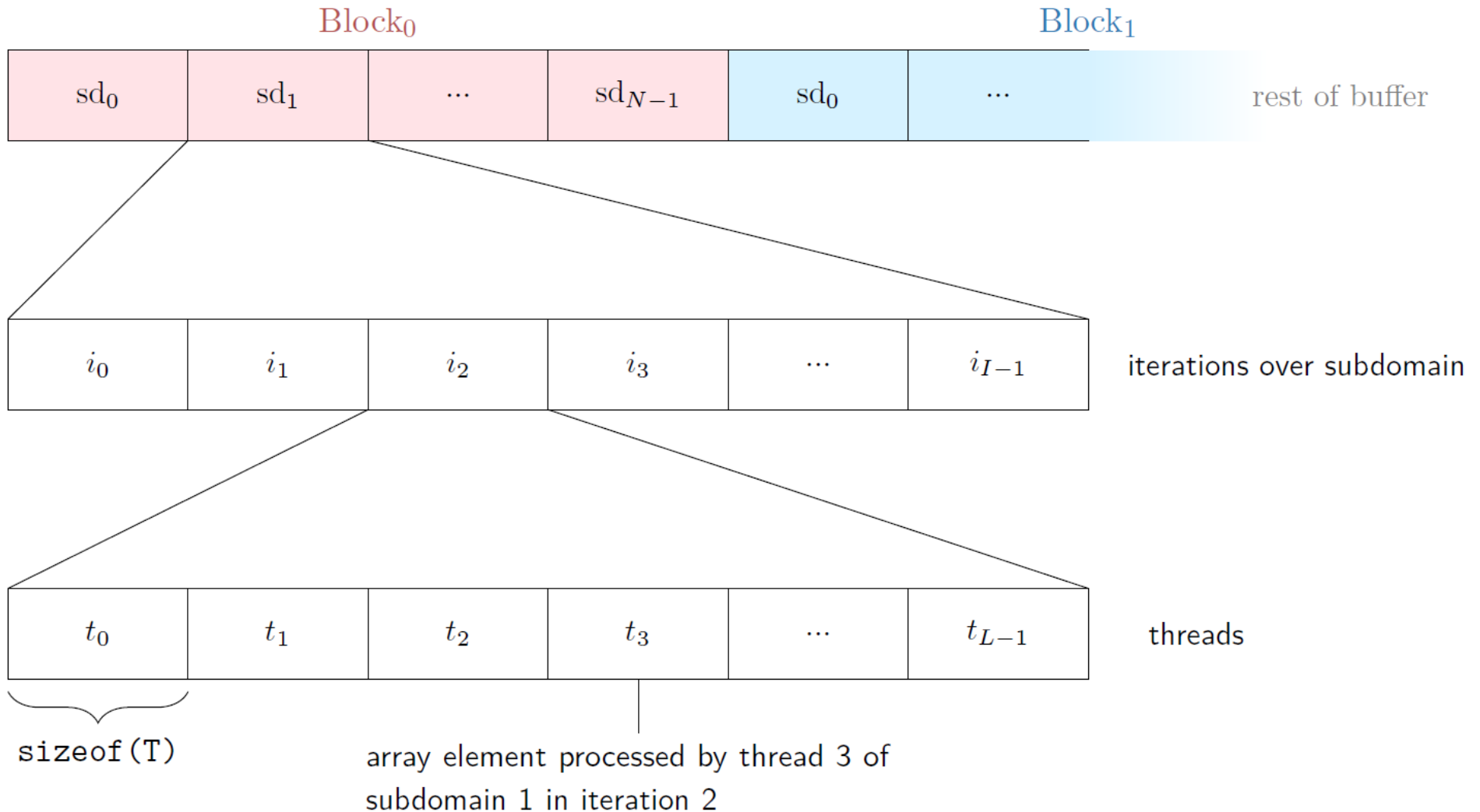
Strided Access: Beobachtungen

- Zwei fast gleiche Werte jeweils bei $\text{sizeof}(T) * \text{stride} = 32$ und 64
- Ab $\text{sizeof}(T) * \text{stride} = 2048$ wenn überhaupt nur noch vernachlässigbares Nachlassen der Kopierrate

Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Producer-Consumer-Verhältnis

Offset Access – Zugriffsmuster



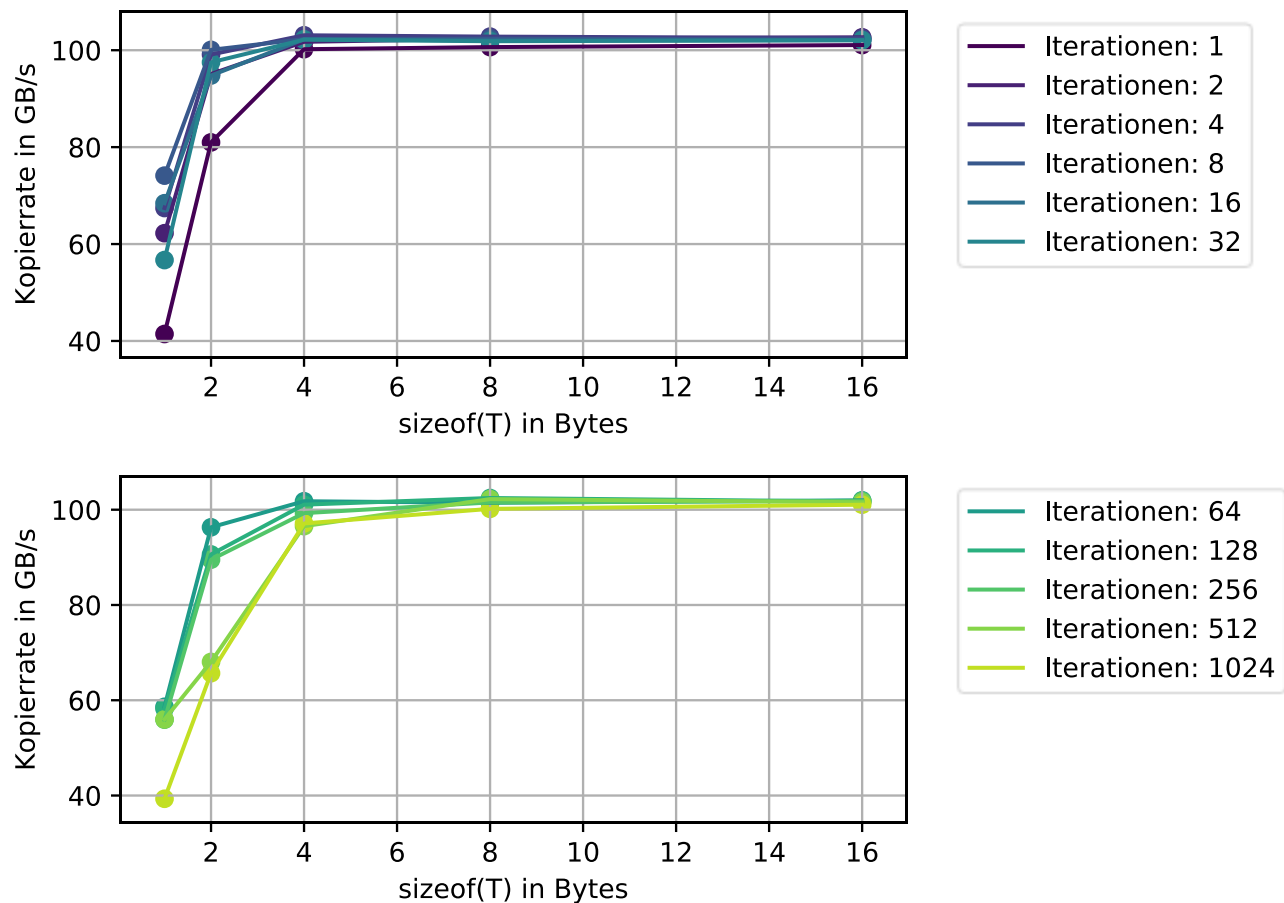
Offset Access

```
//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];
    }
}
```

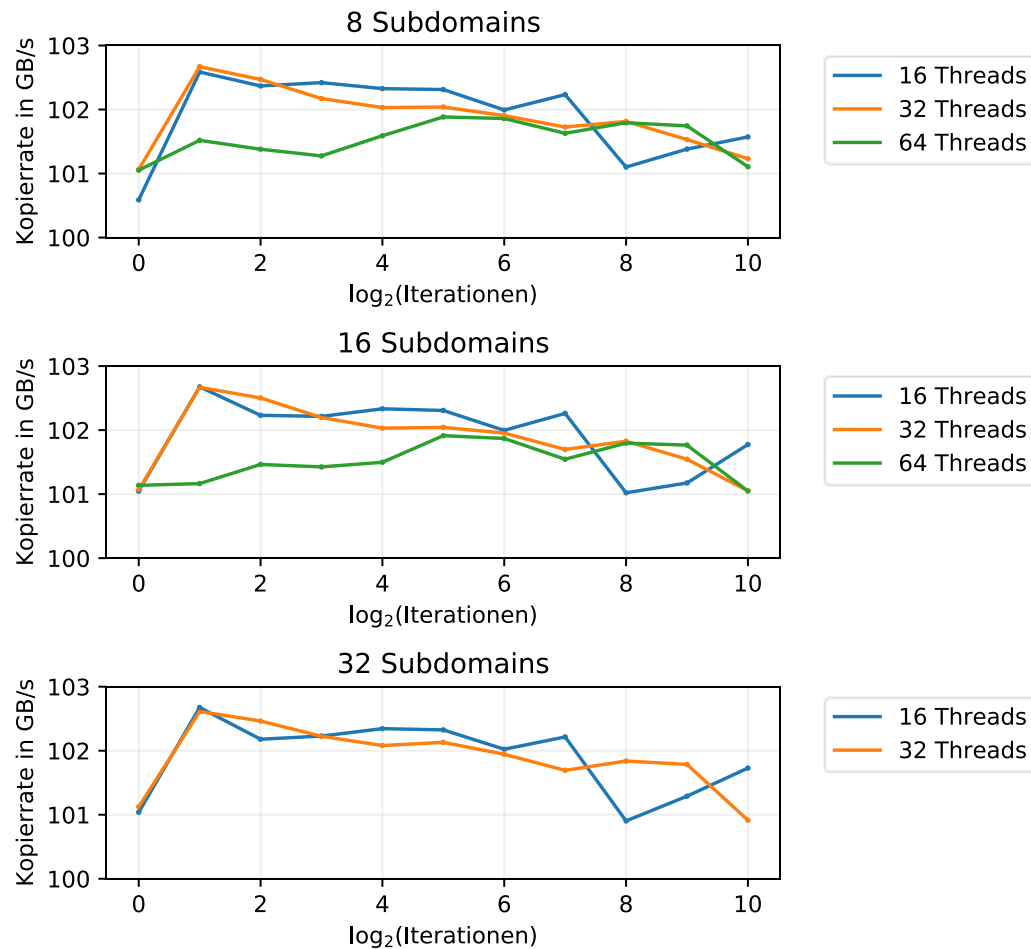
Typ(T) und Iterationen

32 Threads pro Subdomäne, 16 SD pro Block



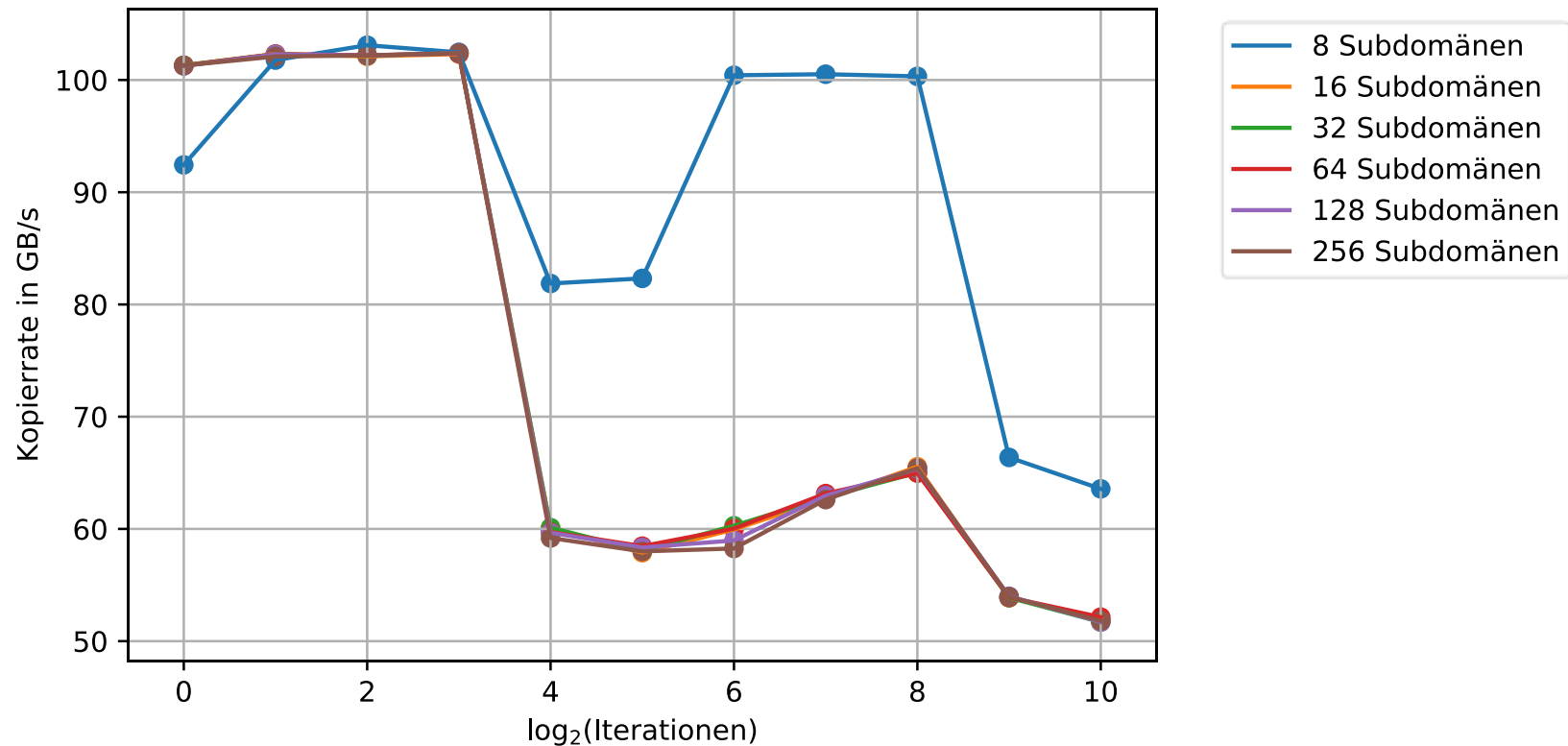
Subdomains pro Block

Zugriffstyp: int4



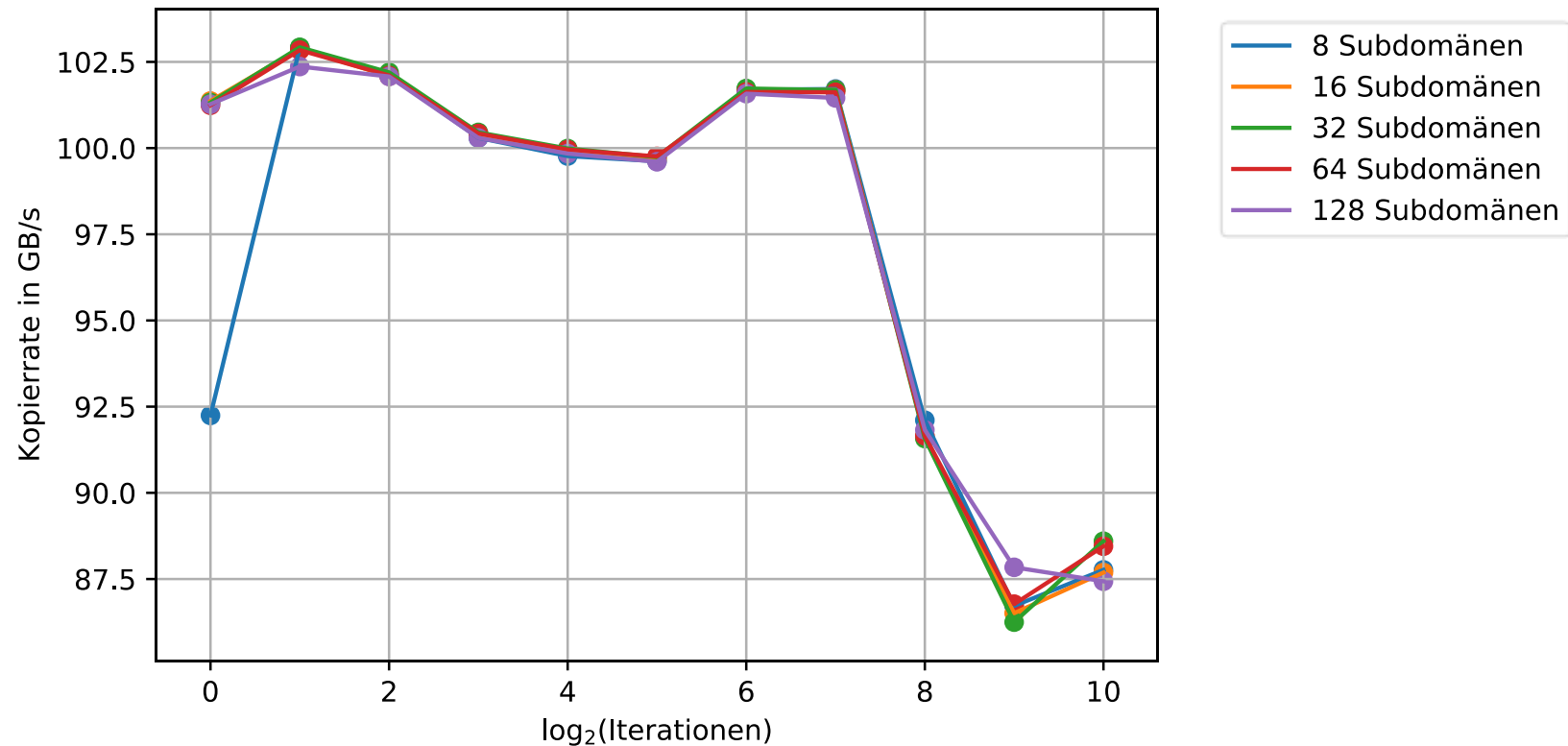
Variiere Subdomänen pro Block

4 Threads pro SD, sizeof(T) = 16



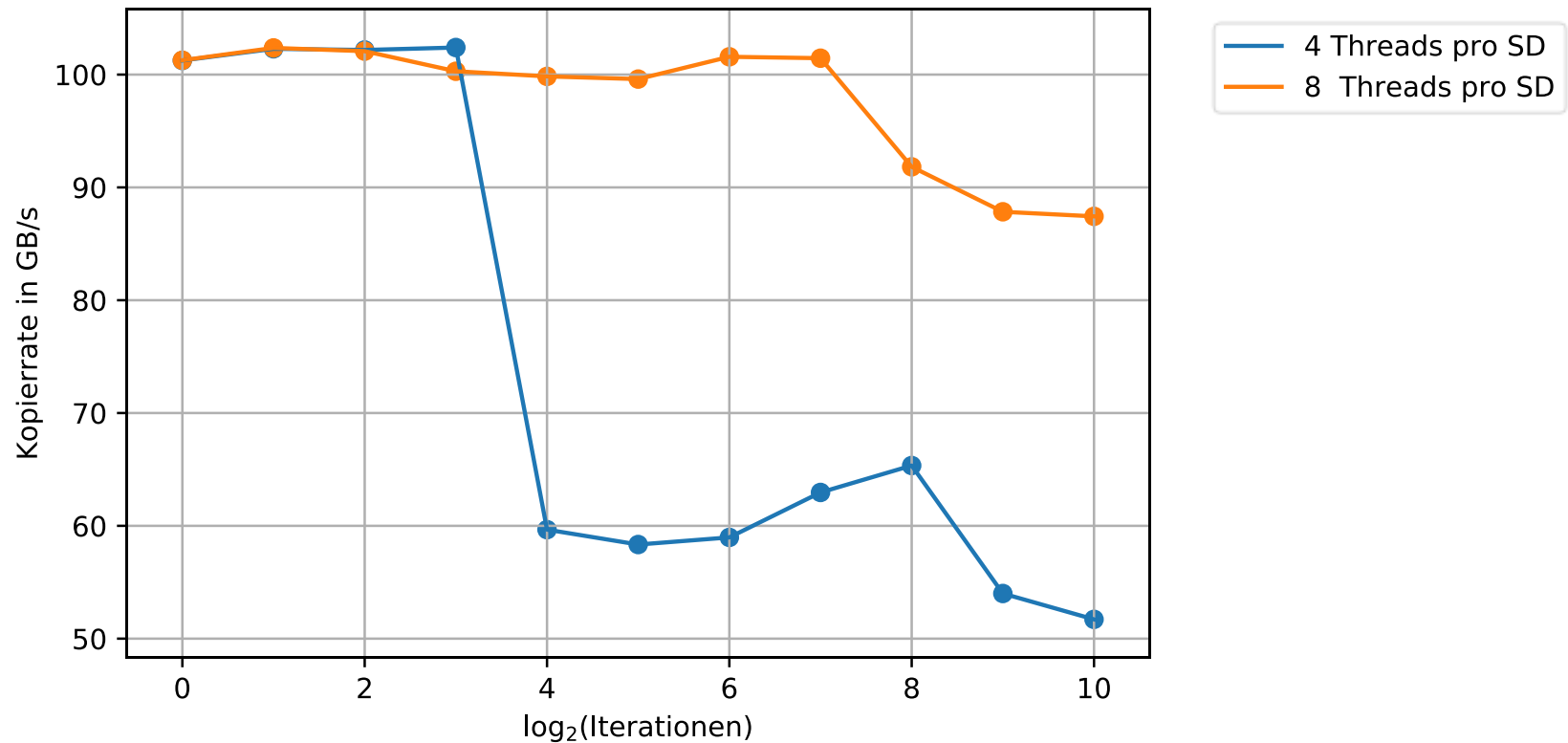
Variiere Subdomänen pro Block

8 Threads pro SD, sizeof(T) = 16



Direkter Vergleich

sizeof(T) = 16, 128 Subdomänen



Offset Access: Beobachtungen

- Im Detail komplexe Zusammenhänge
- Obwohl der Warp, d.h. die 32 zusammen ausgeführten Threads, auseinandergerissen wird, kann das Kopieren trotzdem effizient sein
- Es zeigt sich ein deutlicher Zusammenhang mit der L2-Cache Line-Size (128 Byte)

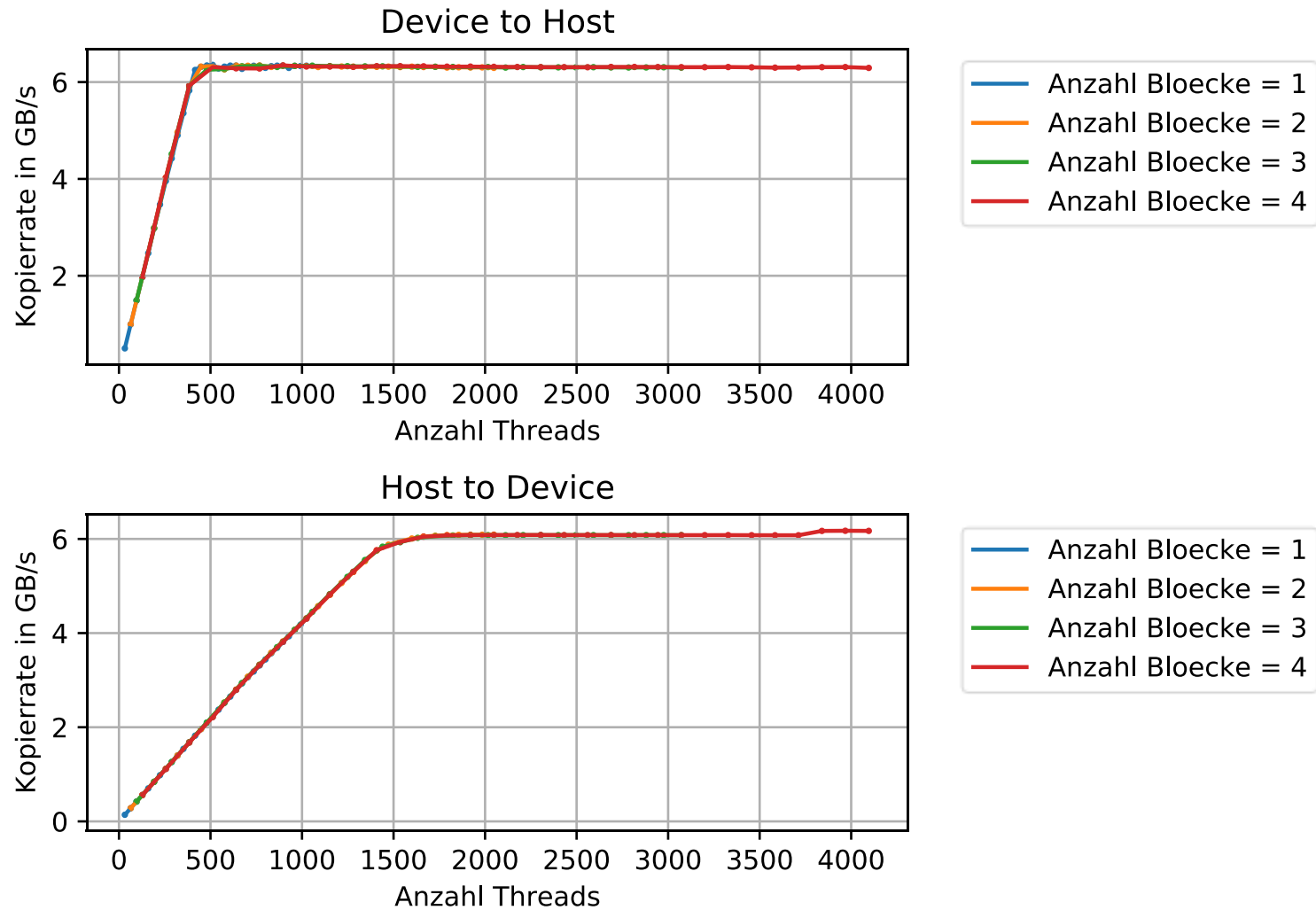
Gliederung

- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Producer-Consumer-Verhältnis

Zugriffsmuster: Copy Kernel

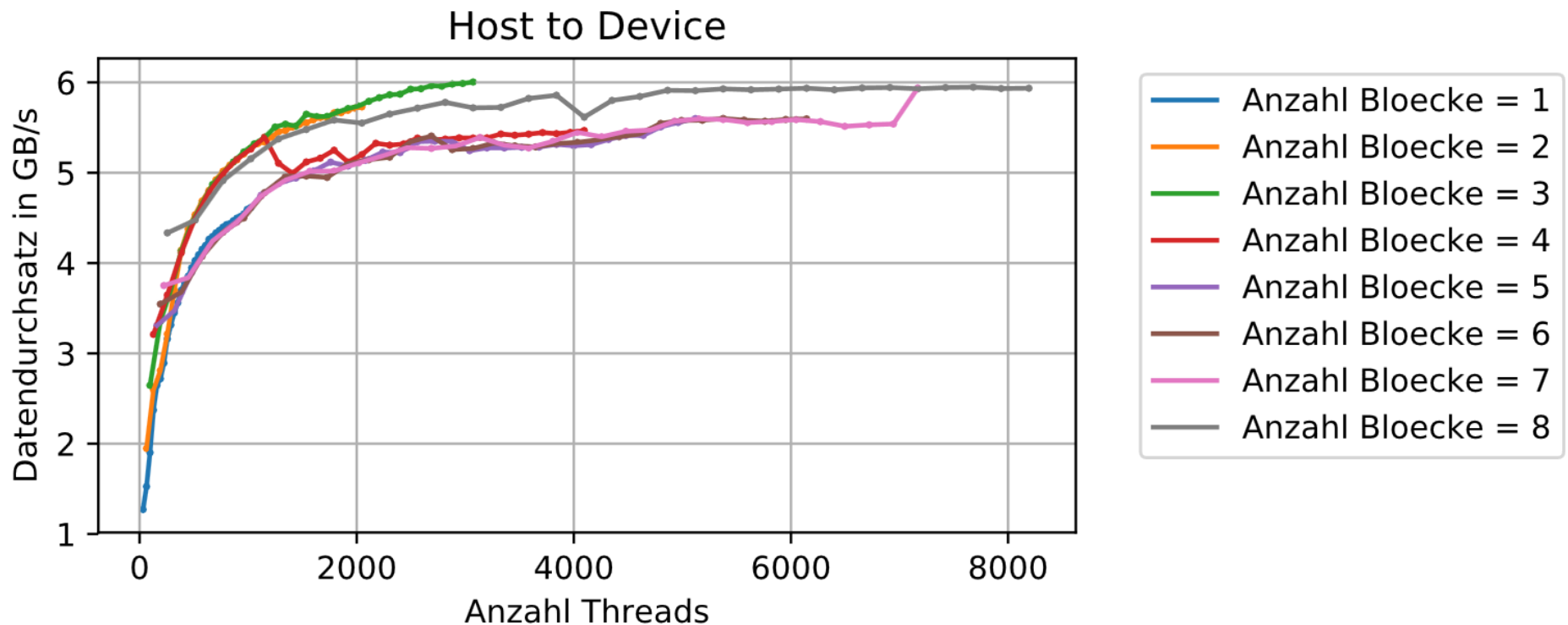
```
//Kernel definition
template<typename T>
__global__
void copyKernel (T* out,
                 T* in,
                 const unsigned int N)
{
    const unsigned int id = threadIdx.x + blockIdx.x * blockDim.x;
    for (unsigned int i= id; i < N; i = i + blockDim.x * gridDim.x)
    {
        const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

CudaMalloc() und CudaMallocHost()



Unified Memory

CudaMallocManaged() & memset & CudaMemset



Gliederung

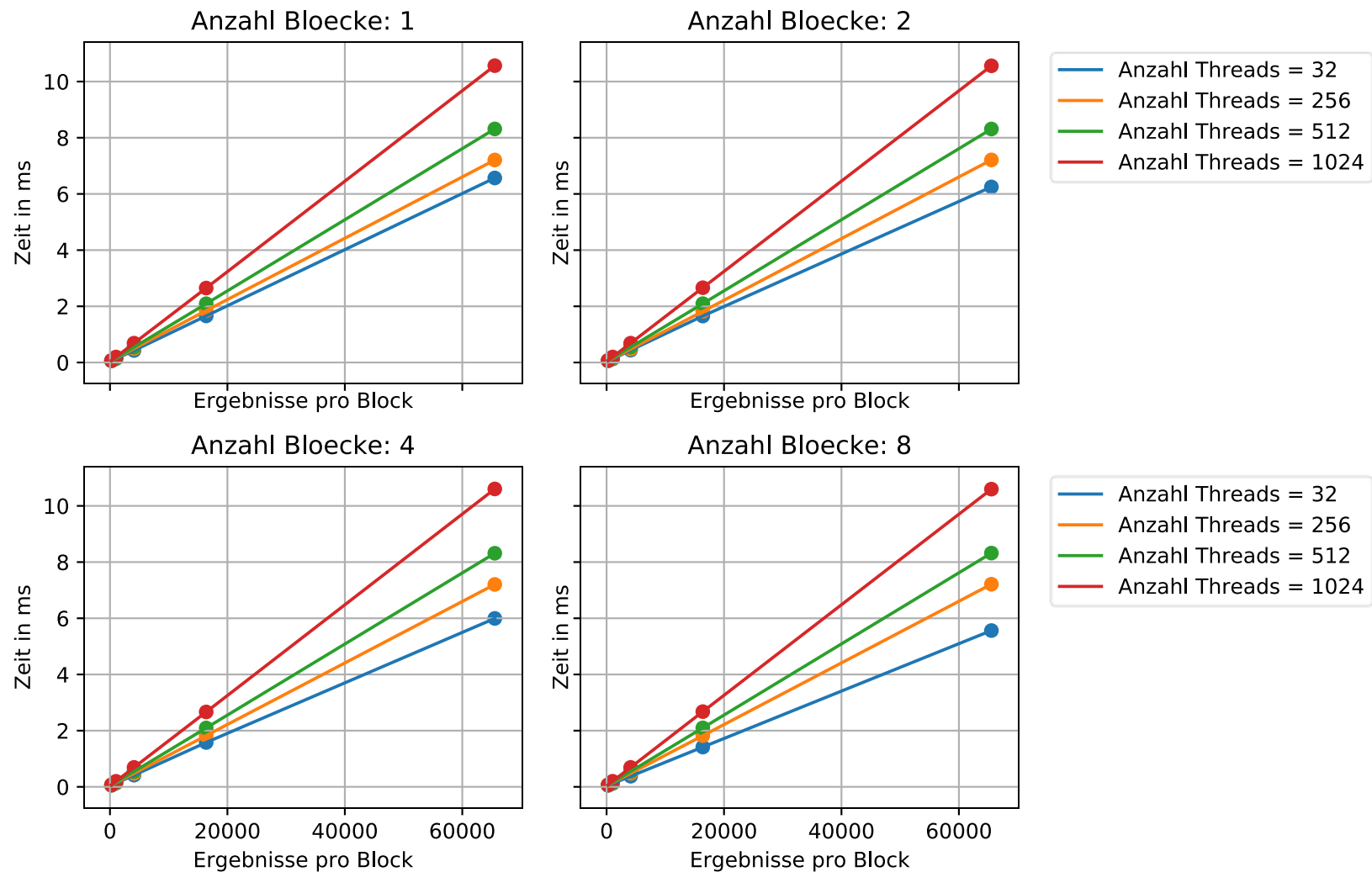
- 1) Allgemeines
- 2) Zugriffsmuster: Copy Kernel
- 3) Zugriffsmuster: Strided Access
- 4) Zugriffsmuster: Offset Access
- 5) Allokation: Standard und UnifiedMemory
- 6) Producer-Consumer-Verhältnis

Producer-Consumer-Verhältnis

- Zwei Kernel, die miteinander kommunizieren, d.h. der Consumer muss auf den Producer warten
- Producer berechnet Durchschnitt eines Arrays
- Consumer nimmt Durchschnitt und schreibt ihn vielfach in Array gleicher Größe
- In der Praxis recht komplexer Code, daher hier nur grob umrissen

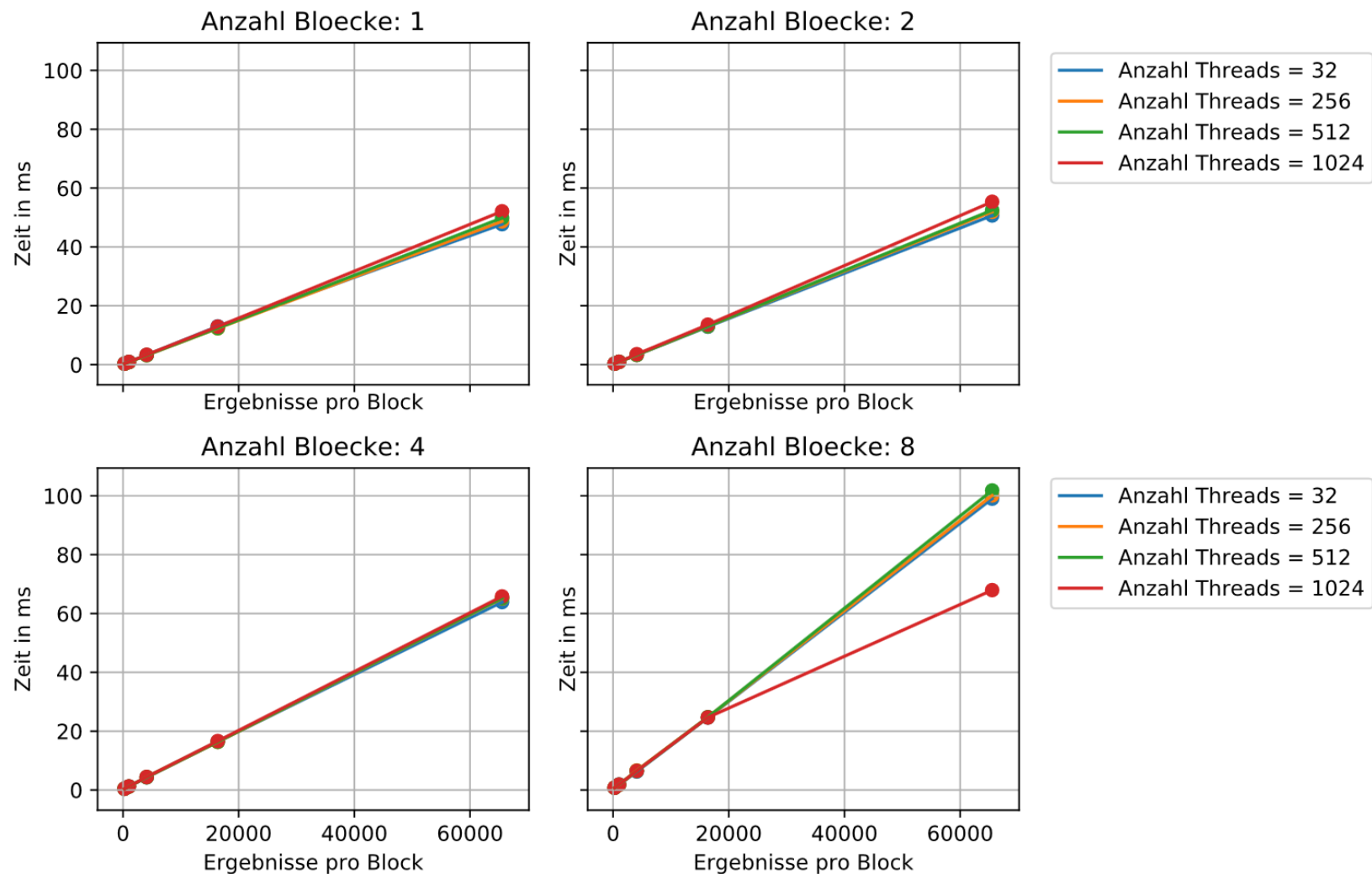
Kommunikationsmessung: Laufzeit ohne Kommunikation

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



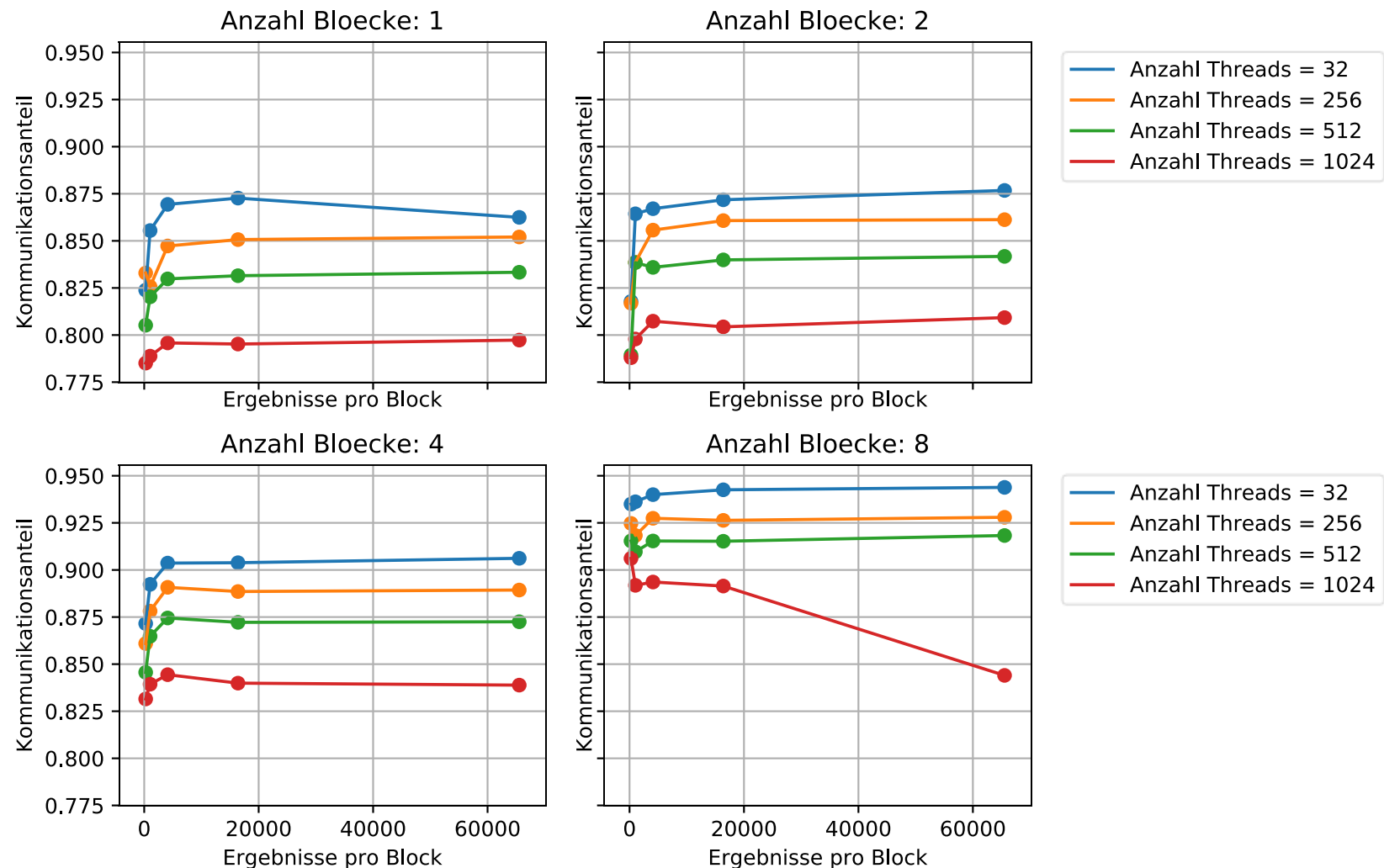
Kommunikationsmessung: Laufzeit mit Kommunikation

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



Anteil der Kommunikation an der Gesamtzeitdauer

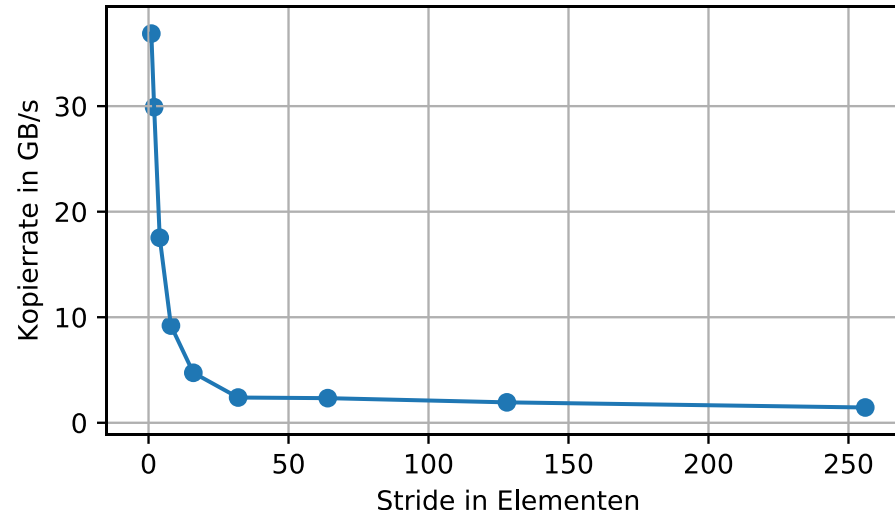
Variation Ergebnisse pro Block: 256, 1024, 4096, 16384, 65536



Ende der Präsentation –
Fragen?

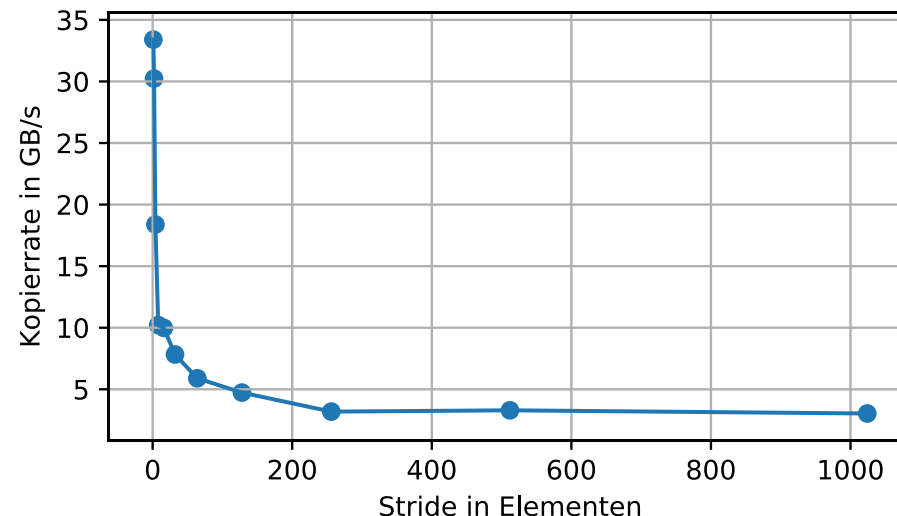
Zusatz - Strided Access III

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



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>

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

Spezifiizaktionen PCIe 2.0 x16 : https://en.wikipedia.org/wiki/PCI_Express