# **Protokoll 3: Rechnen auf GPUs**

**Gruppe 9: Marcel Beyer, Martin Carnein, Franz Lehmann**

## Aufgabe 1:

Die maximale Leistung der gegebenen GPU berechnet sich wie folgt:

Aus der offiziellen Produktbeschreibung kann man einen Peak FP64 Performance von 7,8 TFLOPs entnehmen. Dies stimmt also mit dem berechneten Wert mit Boost überein.

## Aufgabe 2:

Für die Bestimmung der Anzahl der double precision floating point Operationen, müssen drei Metriken erfasst werden:

*smsp\_\_sass\_thread\_inst\_executed\_op\_dadd\_pred\_on.sum -> Anzahl double precision floating point Additionen*

*smsp\_\_sass\_thread\_inst\_executed\_op\_dmul\_pred\_on.sum -> Anzahl double precision floating point Multiplikationen*

*smsp\_\_sass\_thread\_inst\_executed\_op\_dfma\_pred\_on.sum -> Anzahl double precision floating point FMA Operationen*

Diese müssen nach folgender Gleichung zusammengerechnet werden:

*Anzahl floating point Operationen =*

*smsp\_\_sass\_thread\_inst\_executed\_op\_dadd\_pred\_on.sum + smsp\_\_sass\_thread\_inst\_executed\_op\_dmul\_pred\_on.sum + smsp\_\_sass\_thread\_inst\_executed\_op\_dfma\_pred\_on.sum \* 2*

## Aufgabe 3:

void openmp\_matrix\_mul(const float\* mat1, const float\* mat2, float\* mat\_out, int n) {  
#pragma omp target data map(to:n,mat1[0:n\*n],mat2[0:n\*n]) map(from:mat\_out[0:n\*n])  
 {  
#pragma omp target teams distribute parallel for collapse(2)  
 for (int i = 0; i < n; i++) {  
 for(int j = 0; j < n; j++) {  
 mat\_out[i\*n+j] = mat1[i\*n] \* mat2[j];  
 for (int k = 1; k < n; k++) {  
 mat\_out[i\*n+j] += mat1[i\*n+k] \* mat2[k\*n+j];  
 }  
 }  
 }  
 }  
}

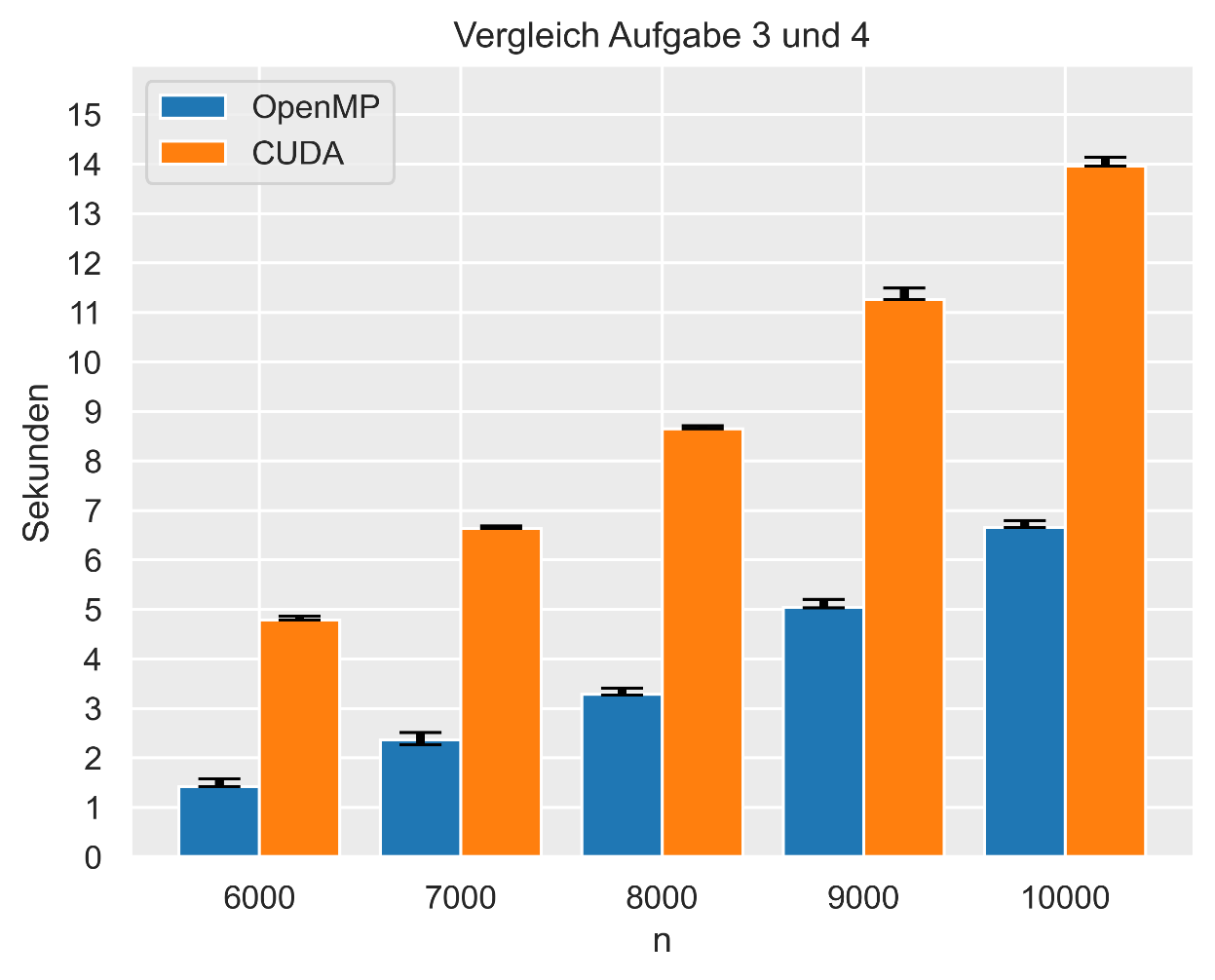
zu Aufgabe c: Die GFLOPs Rate für die größte Matrixgröße (n = 10000) beträgt 292.621 GFLOPs.

## Aufgabe 4:

Aufgabe c : 142.200 GFLOPs bei n = 10000

## zu Aufgabe 3b und Aufgabe 4b:

Die Laufzeiten wurden gemessen und in folgendem Barplot gegenübergestellt:



## Aufgabe 5:

Vergleich der Speichertransfers:

Ohne Prefetching benötigen wir 42,1 Sekunden für Speichertransfers. Mit Prefetching sind es nur 18,6 Sekunden.

Vergleich der Kernellaufzeit:

Die Kernellaufzeit dauert bei beiden Methoden etwa gleich lang. Was auch Sinn macht, da die Rechenzeit des Kernels nicht von den Speichertransfers beeinflusst wird, solange die Daten auf denen gerechnet wird, vor der Berechnung auf der GPU vorliegen.

Die Daten für die Vergleiche haben wir aus folgenden Nvidia Nsight Systems Daten entnommen:



## Aufgabe 6:

Bei der Performance-Analyse mit Nsight Compute wird deutlich, dass die theoretisch mögliche Auslastung der Streaming Multiprozessoren sowohl mit OpenMP als auch mit CUDA bei weitem nicht erreicht wird. Man sieht z. B. auch, dass von theoretisch 64 Warps pro SM bei CUDA nur ca. 30 und beim OpenMP nur ca. 48 genutzt werden.

Hinzu kommt bei CUDA, dass auch die Speicherbandbreite nur zu ca. 30% ausgenutzt wird. Das lässt darauf schließen, dass die CUDA-Implementierung wahrscheinlich latency-bound ist. Die OpenMP-Implementierung nutzt hingegen 65% der Speicherbandbreite aus, wodurch diese Implementierung eher memory-bound ist.

Zusammen führt das dazu, dass nur ein Bruchteil der theoretische Peak Performance erreicht wird.

Folgende Daten aus Nsight Systems/Compute dienen uns als Referenz für unsere Analyse:

