Prof. Dr. Gerhard Zumbusch
Wissenschaftliches Rechnen I – WS 2022/23

Thema: GPU

Ein Quasi-Standard für Berechnungen auf Grafikkarten (GPU) ist die Programmiersprache CUDA, die C/C++ um ein heterogenes Programmiermodell erweitert. Die Berechnung wird aufgeteilt zwischen Host und Grafikkarte(n), zwischen denen Kommunikation explizit programmiert werden muss. Ebenso muss die Datenverteilung auf Host und auf GPU Speicher explizit organisiert werden.
Verwenden Sie dazu folgende CUDA-Befehle für die Kommunikation zwischen Host und Prozessen

```
//host code
// Allocate vectors in device memory
float* d_A;
cudaMalloc(&d_A, size);
// Copy vectors from host memory to device memory
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
// Invoke kernel
MyKernel<<<NoProcesses, NoThreadsPerProcess>>>(d_A, ...);
// Copy result from device memory to host memory
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
// Free device memory
cudaFree(d_A);
```

und für den Code auf der Grafikkarte Ausdrücke wie etwa

```
__global__ void MyKernel(...)
unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
```

Aufgabe 6

Benutzen Sie ihr serielles Programm (Jacobi Iteration) um dieses nun mit CUDA zu vektorisieren/parallelisieren.
Die Umsetzung gestaltet sich so: Ein Kernel-Aufruf entspricht einem Jacobi-Iterationsschritt. Jeder Thread bearbeitet nur einen Gitterpunkt oder einen Teil der Gitterpunkte, jeweils in einfacher Rechengenauigkeit.

a) Verwenden Sie 2000 Iterationen und 1024×1024 innere Gitterpunkte um die Laufzeiten des Programms zu ermitteln.

b) Versuchen sie das Programm zu optimieren bezüglich Speicherlayout. Verwenden sie dazu die Ideen der vorhergehenden Aufgabe zur Vektorisierung mit SIMD Instruktionen angewandt auf die Vektorlänge 32 floats.

   Welche Aufteilung ist CUDA Threads und Prozesse ist günstig?

c) optional: Schreiben sie eine Multi-GPU Version des Programms. Verwenden sie zwei GPUs, die jeweils ihren eigenen Speicherbereich haben und beide einzeln vom Host gesteuert werden.