Übungsblatt 2

04.05.2018 / B. Leder

Wissenschaftliches Rechnen III / CP III

Aufgabe 2.1: *Methode der Konjugierten Gradienten in C*

Implementieren Sie die Methode der Konjugierten Gradienten (Algorithmus 1) in C. Die Matrix A soll dem diskreten Laplace-Operator mit Dirichlet-Randbedingungen in zwei Dimensionen entsprechen ($A \sim -\Delta$). In der (ersten) Vorlesung wurde gezeigt, dass die Poisson-Gleichung auf einem Gitter mit insgesamt $(N_x + 2) \times (N_y + 2)$ Gitterpunkten (Rand- und innere Punkte, $i = 0, 1, \ldots, N_x + 1$, $j = 0, 1, \ldots, N_y + 1$) als Gleichungssystem mit $n = N_x \cdot N_y$ Unbekannten und genauso vielen Gleichungen geschrieben werden kann:

$$Ax = b$$

mit $A \in \mathbb{R}^{n \times n}$ und $x, b \in \mathbb{R}^n$. Die rechte Seite, b, enthält die Information über die Randbedingungen, die Komponenten von x sind die Lösung/Unbekannten an den inneren Punkten $0 < i < N_x + 1$, $0 < j < N_y + 1$

$$u_{i,j} \equiv x_{(j-1)N_x+i}$$

Die Matrix-Vektor-Multiplikation $w = Av$ soll nun als Funktion implementiert werden

```
void laplace_2d(double *w, double *v)
```

Dafür ist es hilfreich, alle Vektoren um die Randpunkte zu erweitern $x, b, v, w, \cdots \in \mathbb{R}^m$, $m = (N_x + 2) \cdot (N_y + 2)$. Dirichlet-Randbedingungen werden genau dadurch erreicht, dass alle Vektoren am Rand Null gesetzt werden: $x_{(j-1)N_x+i} = 0$ für $i = 0$, $j = 0$, $j = N_y + 1$ oder $i = N_x + 1$, und dort nicht durch `laplace_2d` geändert werden. Dann kann man zeigen, dass der Algorithmus 1 die Randpunkte nicht ändert und im Inneren zur gesuchten Lösung konvergiert.

Zu dieser Übungsaufgabe finden Sie die Datei `cg.c` auf der Website (kompilieren mit: `nvcc cg.c -lm`, wegen `sqrt`). Dort sind bereits die Einteilung in Rand- und innere Punkte, Initialisierung und Ausgabe von Vektoren und das Norm-Quadrat ($||v||^2$) implementiert.

1) Implementieren Sie die Matrix-Vektor-Multiplikation $w = Av$. Verifizieren Sie ihr Programm mit Hilfe der Matlab-Routine `laplace.m`.
2) Implementieren Sie zunächst notwendige Funktionen von Vektoren: Skalarprodukt, Addition, etc.
3) Fügen Sie alles zum Algorithmus 1 zusammen. Verifizieren Sie ihr Programm mit Hilfe Ihrer Matlab-Version.

10 Punkte

Aufgabe 2.2: *Array-Addition auf der GPU*

Ausgehend von einem Programm zur Array-Addition auf dem Host, `addArrayHost.cu` auf der Website, implementieren Sie eine Kernel-Funktion, die genau eine Addition ausführt. Benutzen Sie diese, um die Array-Addition auf dem Device zu berechnen und verifizieren Sie das Ergebnis.

Messen Sie Latenz (in ms) und Bandbreite (in GB/s) für das Kopieren zwischen Host und Device. Bestimmen Sie den Durchsatz (in Gflops) für die Berechnung auf dem Host und auf dem Device. Benutzen Sie hierzu die Funktion `seconds`, die die Zeit in Sekunden zurück gibt. Bestimmen Sie Messwerte für ansteigende Array-Größen und extrahieren Sie Latenz, Bandbreite und Durchsatz aus dem asymptotischen Verhalten für große Array-Größen.

10 Punkte