ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

14. Juli 2017
Blatt 11

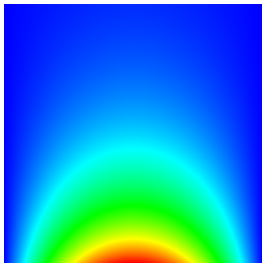
Parallele Programmierung mit C++ (SS 2017)

Abgabe bis zum 20. Juli 2017, 10:00 Uhr

Lernziele:

- Parallelisierte globale zweidimensionale Aggregierung auf grafischen Prozessoren

Aufgabe 14: Jacobi Reloaded



Das Jacobi-Verfahren (siehe dazu das 5. und 7. Übungsblatt) eignet sich bekanntlich sehr gut zur Parallelisierung. Dennoch ist die Umsetzung auf einem Grafikprozessor nicht trivial. Auf GPUs können wir nur innerhalb eines Blocks synchronisieren. Wir benötigen aber eine globale Synchronisierung, da jeder Block am Rand auch von den Nachbarblöcken abhängt. Ebenso bieten uns die GPUs keine Unterstützung zur Aggregierung über den gesamten Bereich an. Normalerweise wird das Jacobi-Verfahren jedoch fortgeführt, bis die betragsmäßig größte Veränderung im letzten Schritt unter eine vorgegebene Schranke fällt (etwa 10^{-6}).

Für diese Aufgabe steht Ihnen wieder eine Vorlage zur Verfügung. Bei dieser können Sie auf der Kommandozeile die Zahl der zu rechnenden Iterationen angeben und den Namen der zu erzeugenden PNG-Datei. Sie ist bereits parallelisiert und enthält drei Kernel-Funktionen, eine für die Initialisierung, eine für den Jacobi-Schritt und eine für das Füllen des Pixelpuffers. So sieht die Initialisierung und der Aufruf der Jacobi-Schritte aus:

```
/* prepare matrices A and B for the Jacobi iterations
   where A is initialized to A_0 */
hpc::cuda::GeMatrix<T, Index> A_dev(N, N, hpc::cuda::RowMajor);
hpc::cuda::GeMatrix<T, Index> B_dev(N, N, hpc::cuda::RowMajor);
auto A = A_dev(0, 0, N, N);
auto B = B_dev(0, 0, N, N);
dim3 block_dim(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid_dim(GRID_SIZE, GRID_SIZE);
```

```
init_matrix<<<grid_dim, block_dim>>>(A);

/* run the given number of iterations;
   iterations is guaranteed to be even */
while (iterations > 0) {
   jacobi_iteration<<<grid_dim, block_dim>>>(A, B); --iterations;
   jacobi_iteration<<<grid_dim, block_dim>>>(B, A); --iterations;
}
```

Durch die Sequenz der Kernel-Aufrufe wird eine globale Synchronisierung erzwungen. Das ist vergleichbar mit einem Aufruf von *__syncthreads()* auf globaler Ebene, nur wird dies durch den Neustart einer Kernel-Funktion deutlich teurer. Dies lässt sich hier leider nicht vermeiden. Zu beachten ist hier, dass A und B nur in der GPU leben und niemals zum Hauptspeicher kopiert werden.

Was hier jedoch fehlt, ist ein sinnvolles Abbruchkriterium, das mit einer Fehlerschranke arbeitet. Im Rahmen dieser Aufgabe sollen Sie die vorhandene Lösung so erweitern, dass Sie auf der GPU das globale Maximum der Abweichung bestimmen und zur CPU kommunizieren, so dass das Hauptprogramm die oben gezeigte Schleife abbrechen kann, wenn dieses klein genug ist.

Zu beachten ist hierbei, dass das globale Maximum ausschließlich auf der GPU zu berechnen ist. Hierfür empfiehlt sich die Vorgehensweise, die ab der Folie 360 im eindimensionalen Fall vorgestellt wird. Im Rahmen dieser Aufgabe ist es nur etwas aufwendiger, da Sie zweidimensional arbeiten müssen und die Aggregation global erfolgen muss. Sie können dies zuerst blockweise tun und mit Hilfe einer weiteren Kernel-Funktion die Werte der einzelnen Blocks aggregieren. Das klappt unter der Voraussetzung, dass $GRID\_SIZE \times GRID\_SIZE$ in einen Block passt, wovon Sie bei dieser Aufgabe ausgehen können (ansonsten wären noch mehr Zwischenschritte notwendig).

Sie werden dabei feststellen, dass trotz der Parallelisierung die Feststellung des globalen Maximums der Veränderungen durchaus nennenswert Rechenzeit frisst und es sich daher nicht lohnt, dies nach jeder Iteration durchzuführen. Stattdessen ist es lohnenswert, erst nach n Iterationen das Abbruchkriterium zu überprüfen, wobei es mit einigen Messungen möglich sein sollte, ein halbwegs geeignetes n zu bestimmen.

Verpacken Sie all Ihre Quellen wiederum mit *tar* in ein Archiv und reichen Sie dies ein:

```
tar cvf jacobi.tar jacobi.cu *.?pp [mM]akefile
submit pp 14 jacobi.tar
```

Dies ist das letzte Übungsblatt dieser Vorlesung. Die Beispiellösung wird in der letzten Vorlesung am 20. Juli besprochen. Weitere Übungsstunden finden nicht mehr statt.

Viel Erfolg!