ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

26. Juli 2019
Blatt 9

Parallele Programmierung mit C++ (SS 2019)

Abgabe bis zum kein Abgabetermin

Lernziele:

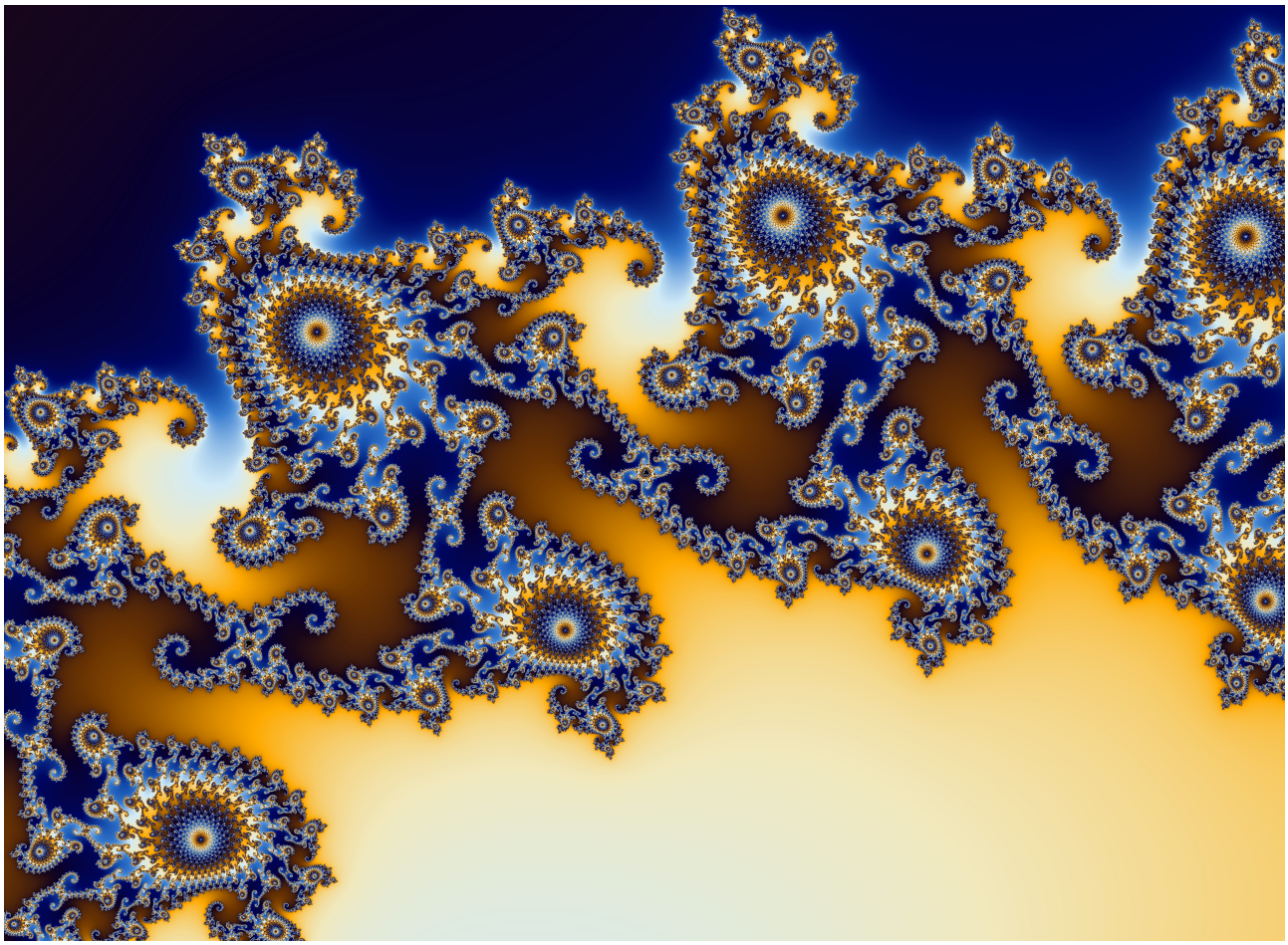
- Verwendung abgebildeten Speichers bei CUDA zur Performance-Verbesserung
- Anpassung der Datenstruktur, so dass der GPU-seitige Zugriff auf den abgebildeten Speicherbereich effizient möglich ist

Aufgabe 10: Mandelbrot Reloaded

Die grafische Visualisierung der Mandelbrot-Menge eignet sich auch zur Parallelisierung mit Hilfe von Grafikkarten. Insbesondere lohnt es sich, für den dabei zu füllenden Pixelbuffer *mapped memory* zu verwenden, der sowohl auf der CPU- als auch der GPU-Seite zugreifbar ist.

Es sind hier allerdings einige Anpassungen sinnvoll, damit dies effizient umgesetzt werden kann:

- Bislang haben wir pro Pixel bei einem *GdkPixbuf* drei Bytes belegt (für die Farben rot, grün und blau). Solche Zugriffe auf „krumme“ Bytezahlen sind aber ineffizient und führen zu Konflikten. Es gibt auch eine Variante des *GdkPixbuf* mit vier Bytes pro Pixel, wenn beim Anlegen der *has_alpha* auf *TRUE* gesetzt wird. Dann kommt noch als viertes Byte ein Alpha-Kanal hinzu, der hier einfach auf 255 gesetzt werden sollte.
- Die Übertragung der 32 Bit für ein Pixel sollte in genau einer Schreiboperation erfolgen und nicht verteilt über mehrere Instruktionen, die nur die einzelnen Bytes füllen.
- Darüberhinaus ist es sinnvoll, wenn jede Zeile eines *GdkPixbuf* auf einer Cache-Line-Boundary beginnt. Die Cache-Lines auf der Grafikkarte der Livingstone belegen 128 Bytes. Das lässt sich umsetzen, indem bei der Spezifikation des *rowstride* ein entsprechend höherer Wert angegeben wird.



Hinweise: Den Speicher für den *GdkPixbuf* sollten Sie zuerst mit *cudaHostAlloc* unter Verwendung des Parameters *cudaHostAllocMapped* anlegen. Der zurückgelieferte Zeiger passt für die CPU-Seite, den für die GPU-Seite passenden Zeiger erhalten Sie mit *cudaHostGetDevicePointer*. In diese Speicherfläche können Sie dann mit *gdk_pixbuf_new_from_data* einen *GdkPixbuf* anlegen.

Wenn Sie die Datenstruktur mit den Konfigurationsparametern aus der Kommandozeile der GPU zur Verfügung stellen möchten, lässt sich das folgendermaßen erreichen. Sei *Parameters* der entsprechende Datentyp, dann können Sie eine entsprechende Datenstruktur im *constant memory* anlegen:

```
__constant__ Parameters p;
```

Später können Sie dann die Datenstruktur auf der CPU-Seite zur GPU kopieren:

```
struct Parameters hp; // parameters on the host
// ...
CHECK_CUDA(cudaMemcpyToSymbol, p, &hp, sizeof(Parameters));
```

Da Sie keine abschließende Kopieraktion von *device to host memory* haben, müssen Sie bedenken, das Sie selbst *cudaDeviceSynchronize* nach dem Aufruf der Kernel-Funktion aufrufen müssen, um auf deren Ende zu warten.

Links zu Manualseiten:

- CUDA-Funktionen zur Speicherverwaltung
- Anlegen eines *GdkPixbuf* in bereits belegten Speicher

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

```
tar cvf mandelbrot.tar *.cu *.?pp [mM]akefile
submit pp 10 mandelbrot.tar
```

Viel Erfolg!