Karlsruhe Institute of Technology (KIT) WS 2016/2017
Institute of Computer Science and Engineering (ITEC)
Chair for Computer Architecture and Parallel Processing (CAPP)

Prof. Dr. Wolfgang Karl
Prof. Dr. Götz Alefeld
M. Sc. Thomas Becker
Dipl.-Math. Markus Hoffmann

Softwarepraktikum Parallele Numerik

Projekt 2 — Partielle Differentialgleichungen und Cuda

Cuda

Aufgabe 1:
Schreiben Sie ein Programm, welches einige Eigenschaften verfügbarer GPUs[1] ausgibt. Hierfür wird mit `cudaGetDeviceProperties` durch die CUDA-API eine nützliche Bibliotheksfunktion bereit gestellt. Dieser Funktion wird ein `struct cudaDeviceProp` übergeben, in der alle Eigenschaften und Parameter der GPUs gespeichert werden.

Aufgabe 2:
In dieser Aufgaben sollen Sie ein Programm schreiben, welches die verfügbare Datentransferrate zwischen dem Host-System und der GPU, sowie die interne Speicherbandbreite der GPU testet. Gehen Sie dabei folgendermaßen vor:
Allokieren Sie zwei Integer-Arrays im Speicher des Host-Systems, sowie zwei Arrays von gleicher Größe im Speicher der GPU (mittels `cudaMalloc`). Kopieren Sie innerhalb des Host-RAMs die Werte des ersten Arrays in das zweite Array. Kopieren Sie anschließend das Array vom Host-System mittels `cudaMemcpy` in den Speicher der Grafikkarte, dann vom einen GPU-Array in das andere, ebenfalls mittels `cudaMemcpy`, und zuletzt wieder zurück in den Arbeitsspeicher des Systems.
Messen Sie jeweils den Zeitverbrauch und berechnen Sie die Transferrate. Variieren

[1] Eine grundlegende Dokumentation zu CUDA ist unter `http://docs.nvidia.com/cuda/index.html` abrufbar. Weitere Dokumentation und Beispielprogramme finden Sie im zugehörigen SDK oder unter `i82sn02:/opt/cuda/sdk/`. Machen sie sich mit den grundlegenden CUDA-Direktiven und Bibliotheksfunktionen vertraut. Beachten Sie insbesondere die CUDA-eigenen Funktionen zur Zeitmessung.
Bitte nehmen Sie Rücksicht auf andere Benutzer auf diesem Rechner, die ggfs. auch Messungen durchführen. Messen Sie für alle Aufgaben die reine Kernel-Laufzeit als auch die Laufzeit inklusive Speichertransfers.

Sie außerdem die Größe des Arrays. Welche Transferraten erreichen Sie? Welche Unterschiede beobachten Sie?
Erweitern Sie nun Ihr Programm um einen CUDA-Kernel, welcher sämtliche Werte eines Arrays im GPU-Speicher um eine Konstante X erhöht.
Vergleichen Sie ihre CUDA-Version mit einer parallelen Version auf dem Host-System. Welchen Speedup erreichen Sie? Variieren Sie Array-Größe sowie die Blockgröße zwischen 4, 8, 16, 24 und 32. Überprüfen Sie jeweils das Ergebnis. Welche Rolle spielen hierbei die SIMD-Register?

Parallelisierung

Aufgabe 3:
Im Rahmen des letzten Projektes haben Sie an dieser Stelle eine paralle Version des Gauß-Seidel-Verfahrens entwickelt.

a) Setzen Sie ihre Entwicklung nun auch mit Hilfe von CUDA um und vergleichen Sie die Ergebnisse und den Speed-Up mit den Resultaten aus Aufgabe 4 des ersten Projektes.

b) Informieren Sie sich über asynchrone Parallelisierungsmethoden[2]. Diese eignen sich, um auf GPUs noch einmal deutliche Beschleunigungen zu erreichen. Verbessern Sie ihre Version des Gauß-Seidel-Verfahrens mit Hilfe der gefundenen Techniken und zeigen Sie die Vorteile dieser Techniken damit auf. Gibt es auch Nachteile? Welchen Einfluss haben hierbei die Warp-Größen der GPU?

c) Erörtern Sie, ob die von ihnen verwendete asynchrone Methodik auch für Multi-Core-, Many-Core- oder MIC-Architekturen geeignet ist. Wo liegen die Unterschiede in der Anwendung zur Verwendung auf GPUs? Lässt sich daraus ein "Hardwarefavorit" ermitteln?

[2] Mögliche Stichworte: Approximate Computing, asynchrone Parallelisierung, relaxierte Parallelisierung.

Aufgabe 4:
Gegeben seien zwei Algorithmen zur Bestimmung der unvollständigen LU-Zerlegung (ILU) einer symmetrischen Matrix $A \in \mathbb{R}^{n\times n}$ mit $A \approx LU$. In der Indexmenge S_U seien alle die Indexpaare (i,j) hinterlegt, für die $u_{ij} \neq 0$ gilt.

Algorithmus 1 (ILU nach Fixpunktiteration)

Wähle Startmatrix $U^0 = u_{ij}^0 \in \mathbb{R}^{n\times n}$ zur gegebenen Indexmenge S_U.

for $m = 0, 1, \ldots$ until convergence
  for $(i,j) \in S_U$

$$s^m = a_{ij} - \sum_{k=1}^{i-1} u_{ki}^m u_{kj}^m$$

if $i \neq j$ **then**

$$u_{ij}^{m+1} = \frac{s^m}{u_{ii}^m}, \qquad l_{ji}^{m+1} = u_{ij}^{m+1}$$

else

$$u_{ii}^{m+1} = \sqrt{s^m}, \qquad l_{ii}^{m+1} = u_{ii}^{m+1}$$

Algorithmus 2 (ILU nach Gauß)

for $j = 1$ **to** n

$$l_{jj} = \sqrt{a_{ii} - \sum_{k=1}^{j-1} l_{ik}^2}, \qquad u_{jj} = l_{jj}$$

for $i = j + 1$ **to** n

if $a_{ij} = 0$ **then**

$$l_{ij} = 0, \qquad u_{ji} = 0$$

else

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}, \qquad u_{ji} = l_{ij}$$

a) Welcher der beiden Algorithmen eignet sich aus Sicht der Mathematik besser zur Bestimmung der ILU-Zerlegung? Welcher Algorithmus lässt sich besser parallelisieren? Welche Methodik führt beim entsprechend weniger gut parallelisierbaren Verfahren zum wahrscheinlich besten Ergebnis bezüglich Laufzeit und Skalierbarkeit? Welche Speichermethodik ziehen Sie für L und U in Betracht und warum?

b) Implementieren Sie zunächst Algorithmus 2 für die Matrix A aus Aufgabe 4 des ersten Projektes. Verwenden Sie diesen, um eine Indexmenge S_U für Algorithmus 1 zu finden. Implementieren Sie im Anschluss auch Algorithmus 1 und parallelisieren Sie beide Verfahren mit Hilfe von CUDA. Decken sich Ihre Evaluationsergebnisse mit den Annahmen aus Aufgabenteil a)? Begründen Sie.

c) Ohne Implementierung: Wäre eine Umsetzung der Parallelisierung mit Hilfe von OpenMP geeigneter? Woran machen Sie ihre Entscheidung fest?

Partielle Differentialgleichungen

Aufgabe 5:
Vorkonditionierung stellt eine wichtige Methode dar, um Krylow-Unterraumverfahren deutlich zu verbessern. Eine Vorkonditionierungsmethodik ist gegeben durch eine Kombination aus ILU-Zerlegung und einem geeigneten Löser für triangläre Gleichungssysteme, beispielsweise dem Gauß-Seidel-Verfahren. Dabei wird die ILU-Zerlegung genutzt, um die folgenden beiden Gleichungssysteme aus A zu erstellen:

$$L\hat{y} = r_m$$
$$Uy = \hat{y},$$

wobei y das Produkt aus Vorkonditionierungsmatrix B und Residuum r_m des m-ten Schrittes des Krylow-Unterraumverfahrens beschreibt. Diese beiden Gleichungssysteme werden nun mit dem Gauß-Seidel-Verfahren gelöst.

a) Stellen Sie aus den bisherigen Entwicklungen beider Projekte ein möglichst vollständig parallelisiertes Verfahren zusammen, welches das in Aufgabe 5 des ersten Projektes gegebene Poisson-Problem mit Hilfe von Finiten Differenzen und vorkonditioniertem Krylow-Unterraumverfahren löst. Der lokale Fehler ϵ_i darf 10^{-5} nicht überschreiten. Bewerten Sie die Qualität ihres Verfahrens und stellen Sie die Ergebnisse[3] anschaulich dar.

b) Ist es notwendig, alle verwendeten Verfahren bis zur angegebenen Genauigkeit zu bringen? Welche können früher abgebrochen werden, welche nicht? Wann[4] kann abgebrochen werden und warum?

c) Aus Aufgabenteil a) ergibt sich bereits eine Aufteilung einzelner Verfahrenskomponenten auf verschiedene Recheneinheiten und Beschleuniger. Ist diese Aufteilung sinnvoll? Verwenden Sie die Aufteilung, die ihnen am zeitsparendsten scheint und setzen Sie diese um. Optimieren Sie ihre Version so weit wie möglich und vergleichen Sie diese optimale Verfahrensversion mit dem Aufgabenteil a).

(Optional) **Aufgabe 6:**
Die Lattice-Boltzmann-Methode (LBM) (oder auch Gitter-Boltzmann-Methode) ist eine numerische Technik für die Simulation von Strömungen. Sie kann beispielsweise dazu verwendet werden, um die inkompressible, zeitabhängige Navier-Stokes-Gleichung numerisch zu lösen. Die LBM basiert auf der Berechnung einer stark vereinfachten Teilchen-Mikrodynamik. Das heißt, es wird eine Simulation auf der Teilchenebene durchgeführt. Die Wechselwirkung der mikroskopischen Teilchen wird durch die Boltzmann-Gleichung beschrieben. Aufgrund der internen Struktur (geringer Speicher- und Rechenbedarf je Zelle) eignet sich das Verfahren u.a. zur Berechnung von Strömungen in komplexen Geometrien.
Als Anwendungsfall wurde ein einfaches Modell gewählt. Hierbei handelt es sich um einen mit einer Flüssigkeit gefüllten Würfel. Während der Simulation wird nun der Deckel des Würfels zur Seite geschoben, was dazu führt, dass die Flüssigkeit in Rotation versetzt wird.

[3] Sowohl die errechneten Ergebnisse des Poisson-Problems als auch die Bewertungsergebnisse.
[4] Gemeint ist die temporale (nach wievielen Schritten, nach welcher Laufzeit, o.ä.), nicht die causale (nach welchen Kriterien kann das beurteilt werden) Form von wann.

Ziel dieser Aufgabe ist die Parallelisierung der Lattice-Boltzmann-Methode mittels CUDA[5].

a) Wie groß darf der Würfel maximal werden, damit die Daten vollständig im Grafikspeicher gehalten werden können?

b) Was müssen Sie bei größeren Würfeln beachten?

c) Welchen Speed-Up erreichen Sie, wenn Sie das gegebene Programm und ihre Parallelisierung möglichst effizient gestalten?

Version: 19. Dezember 2016

[5]

- Programmaufruf `./do_lbm < input.dat > output.txt`
- Führen sie ihre Laufzeitmessungen mit unterschiedlicher Kantenlänge (z.b. 10, 50, 100, ...), sowie einer unterschiedlichen Anzahl an Zeitschritten durch (Standard sind 100 Zeitschritte). Bitte beachten Sie, dass Lattice-Boltzmann-Methoden mit jeder Dimension skalieren, d.h. eine Verdoppelung der Kantenlänge des Würfels hat eine 8-fach längere Laufzeit zur Folge.
- Verwenden Sie die bekannten Programme zur Identifikation der Engpässe.
- Für die Verifikation genügt ein Würfel mit einer Kantenlänge von 5 und 100 Zeitschritten. Aktivieren sie hierfür die Testausgabe in *main.c*
- Verifikation der Korrektheit mittels *diff* (`diff output_ser.txt output_par.txt`)