JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

DezSys08 - GPGPU

Systemtechnik-DezSys
5BHIT 2015/16

Michael Weinberger

Version 1.0
Begonnen am 15. Januar 2016
Beendet am 23. Januar 2016

Note:
Betreuer: Borko/Micheler

Inhaltsverzeichnis

1 Einführung

1.1 Ziele

Die Aufgabe beinhaltet eine Recherche über grundsätzliche Einsatzmöglichkeiten für GPGPU. Dabei soll die Sinnhaftigkeit der Technologie unterstrichen werden. Die Fragestellungen sollen entsprechend mit Argumenten untermauert werden. Im zweiten Teil der Arbeit soll der praktische Einsatz von OpenCL trainiert werden. Diese können anhand von bestehenden Codeexamples durchgeführt werden. Dabei wird auf eine sprechende Gegenüberstellung (Benchmark) Wert gelegt. Die Aufgabenstellung soll in einer Zweiergruppe bearbeitet werden.

1.2 Aufgabenstellung

Informieren Sie sich über die Möglichkeiten der Nutzung von GPUs in normalen Desktop-Anwendungen. Zeigen Sie dazu im Gegensatz den Vorteil der GPUs in rechenintensiven Implementierungen auf [1Pkt].
Gibt es Entwicklungsumgebungen und in welchen Programmiersprachen kann man diese nutzen [1Pkt]?
Können bestehende Programme (C/C++ und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür [1Pkt]?
Gibt es transcompiler und wie kommen diese zum Einsatz [1Pkt]?

Präsentieren Sie an einem praktischen Beispiel den Nutzen dieser Technologie. Wählen Sie zwei rechenintensive Algorithmen (z.B. Faktorisierung) und zeigen Sie in einem aussagekräftigen Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware gegenüber dem Ausführen auf einer CPU bringt (OpenCL).

Punkteschlüssel:

Auswahl und Argumentation der zwei rechenintensiven Algorithmen (Speicher, Zugriff, Rechenoperationen) [0..4Pkt]
Sinnvolle Gegenüberstellung von CPU und GPU im Benchmark [0..2Pkt]
Anzahl der Durchläufe [0..2Pkt]
Informationen bei Benchmark [0..2Pkt]
Beschreibung und Bereitstellung des Beispiels (Ausführbarkeit) [0..2Pkt]

1.3 Links

OpenCL-Examples von René Hollander & Paul Kalauner [1]

2 Möglichkeiten GPU Desktopanwendungen, Vorteil GPU bei rechenintensiven Implementierungen

GPGPU (General Purpose Computation on Graphics Processing Unit) ist eine Programmier-Schnittstelle mit der Möglichkeit, allgemeine Berechnungen von der GPU ausführen zu lassen. Die ursprüngliche Aufgabe von Grafikchips ist, den Bildschirm mit Pixel zu füllen. Inzwischen eignen sich die umfangreichen Befehlssätze der GPUs auch für allgemeine Berechnungen, dies ist ein vergleichsweise neuer Trend in der Computertechnik, der darauf abzielt freie Rechenleistung konsequent auszunutzen.
Eine CPU besitzt wenige, im Serverbereich bis zu 16, Kerne die für jedmögliche Art von Berechnungen ausgelegt sind. Eine GPU besitzt mehrere Hundert, heute sogar schon bis zu über 1536 Kerne die parallel arbeiten. Ein Vorteil, bestimmte komplexe Berechnungen hierauf auszulagern erscheint klar sichtbar. Bei der reinen Rechenleistung ist wie auf dem Diagramm ersichtlich die

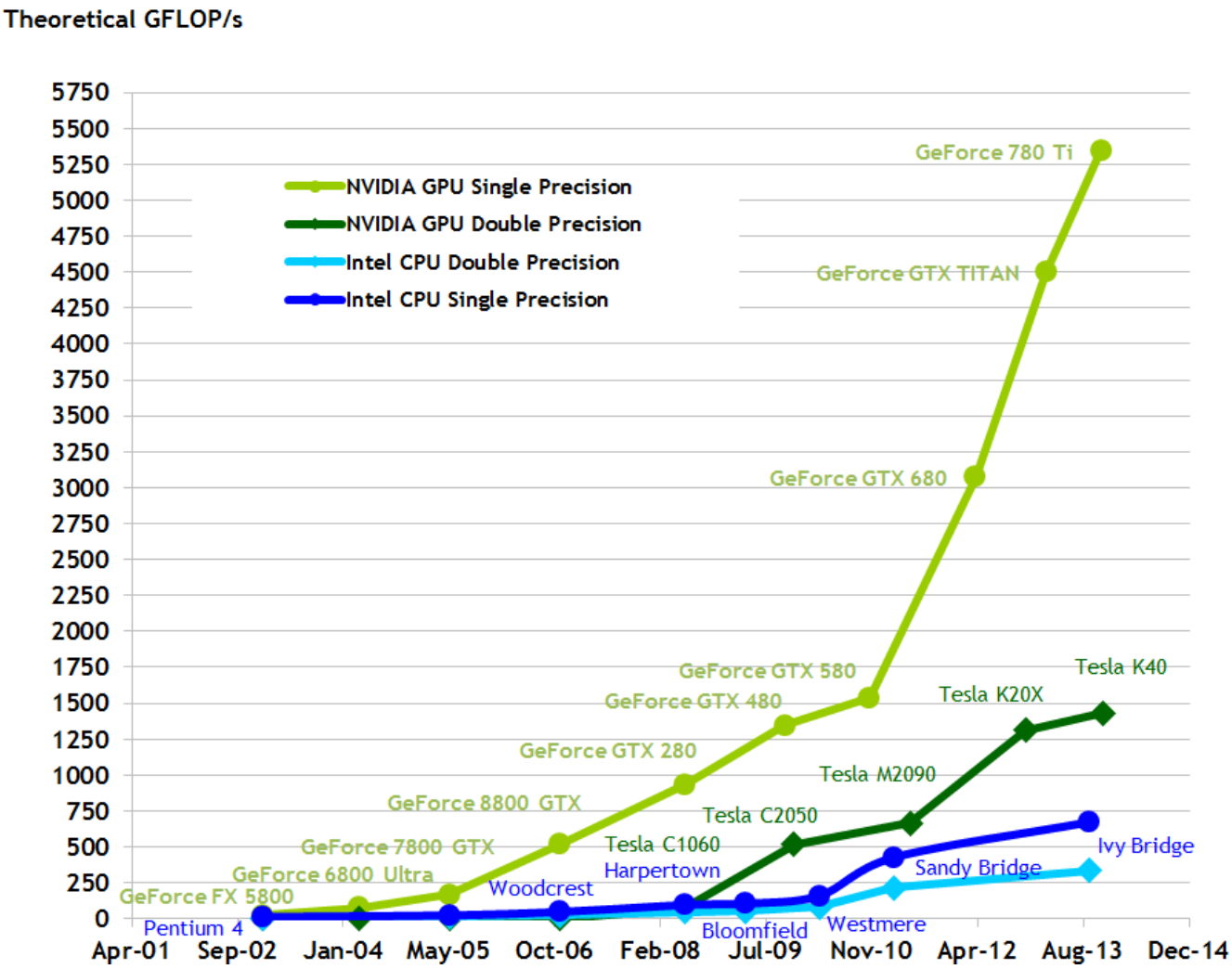

Abbildung 1: GPU vs CPU, reine Rechenleistung [2]

GPU weit vorne. Die Rechenleistung ist in FLOPS (Floating Point Operations), die theoretisch mögliche Spitzenleistung, angegeben.

GPGPU wird verwendet um Rechenaufwendige Applikationen zu beschleunigen. Besonders aufwendig sind vor allem die folgenden Anwendungsbereiche Simulationswissenschaften, Medizin, Deep Learning.

Grafikkarten sind auf massive Parallelität ausgelegt, dies stellt aber auch Entwickler vor neue Herausforderungen. Es können je nach Anzahl an programmierbaren Shadereinheiten über 1000 Berechnungen zugleich in einem Takt ausgeführt werden. [3] [4]

3 Entwicklungsumgebungen, Programmiersprachen

3.1 CUDA SDK

Die CUDA Entwicklertools umfassen drei wichtige Komponenten: Den aktuellsten CUDA-Treiber, ein komplettes CUDA-Toolkit sowie CUDA SDK-Codebeispiele. In der Entwicklung bindet man ein Programm an den GPU-Hersteller Nvidia und somit an das Vorhandensein von Nvidia-Hardware. Es stellt eine umfassende Entwicklungsumgebung bereit, um hauptsächlich in C/C++ geschriebene Programme mithilfe der GPU zu beschleunigen. Es existieren aber auch Wrapper für die Sprachen Perl, Java, Python, Ruby, Fortran und .NET. [5]

3.2 OpenCL

OpenCL (Open Computing Language) ist ein offenes, lizenzfreies und plattformübergreifendes Framework, um Systeme aus einer Kombination von CPUs, GPUs und anderen Prozessoren zu programmieren. OpenCL wurde ursprünglich von Apple entwickelt, und war der erste Standard, welcher nahezu perfekt für heterogene Systeme ausgelegt war. Eine namhafte Implementierung mithilfe von OpenCL ist etwa der APP SDK von AMD. Die Host-API ist in OpenCL C/C++ definiert, es gibt außerdem Third-Party-APIs für Python, Java und .NET. [4]

4 Bestehende Programme (C/C++ und Java) auf GPUs nutzen und Grundvoraussetzungen

Dies ist nur durch bestimmte Toolkits, wie etwa GPSME [6] möglich, das einem bestimmte Vorgänge abnimmt. Ansonsten ist dieser Schritt nicht ohne erheblichen Mehraufwand möglich, die Logik müsste bspw. in OpenCL C erneut geschrieben werden, was auch ein komplettes Redesign der Algorithmen zur parallelen Lauffähigkeit auf mehreren Kernen bzw. Threads mit sich zieht.

5 Transcompiler und deren Einsatz

Nach einer Recherche ist mir kein solcher aufgefallen, der direkt source-to-source operiert. Wie im vorherigen Punkt beschrieben würde dies wahrscheinlich auch nur eingeschränkt Sinn machen, wenn z.B. Code in C ohne die benötigte Auslegung auf Parallelität zu OpenCL C-Code generiert wird. Außerdem sind in OpenCL C, welche im Grunde auf der Syntax von ISO C99 basiert, zusätzliche Datentypen und Funktionen enthalten zur parallelen Verarbeitung, die einem in C abgehen. Umso weiter entfernt scheint nun die Vorstellung eines Transcompilers, der hier alle Umstände berücksichtigt, und aus einer eingeschränkten Umgebung Code sinngemäß erweitern und verändern kann.
Zur generellen Erklärung:
Ein Transcompiler ist ein spezieller Compiler, der den Sourcecode eines Programmes in eine andere Programmiersprache überrträgt. [7]

6 Praktisches Beispiel: Benchmark

6.1 Auswahl und Argumentation der Algorithmen

6.1.1 Pi berechnen auf beliebig viele Stellen mit GPUPI

Die hier verwendete Bailey-Borwein-Plouffe-Formel ist eine Summenformel zur Berechnung der n-ten Nachkommazahl der Kreiszahl Pi. Der Gesamtaufwand beläuft sich auf die komplexe mathematische Berechnung.

6.1.2 CompuBenchCL

Die folgenden Tests werden im Benchmark berücksichtigt:

- Gesichtserkennung
  Basiert auf dem Viola-Jones-Algorithmus, analysiert Bilder auf Gesichter, um biometrische Daten zu sammeln.
- TV-L1 Optical Flow
  Dieses Berechnungsmodell wird etwa in der Vidokompression verwendet.
- Ocean Surface Simulation
  Testet den FFT-Algorithmus anhand von Meereswellen.
- Particle Simulation - 64k
  Simuliert viele einzelne Partikel in einem Würfel und deren Bewegungen.
- T-Rex
  Ein Kurzfilm als Echtzeitberechnung.
- Video Composition
  Stellt verschiedene Videofilter gleichzeitig dar.
- Bitcoin Mining
  Berechnen des Bitcoin-Algorithmus, die Hersteller werden sich freuen!

6.2 Gegenüberstellung CPU/GPU

sadf

6.3 Anzahl der Durchläufe

sadf

6.4 Informationen bei Benchmark

Systemspezifikationen:

- Acer Aspire V3-772
- Windows 8.1 64-bit
- Ausgabe 1920x1080p
- Intel Core i7-4702MQ @ 2.20Ghz, 4 Kerne, 8 logische Prozessoren
- NVIDIA GeForce GTX 760M & Intel HD Graphics 4600 mit aktuellen Treibern (23.01.2016)
- CL API Intel OpenCL & NVIDIA CUDA
- 16 GB RAM, 2 x 8 GB DDR3 1600 Mhz
- Toshiba-HDD, 1 TB
- Intel-SSD, 120 GB
- 100%, Netzbetrieb

6.5 Beschreibung, Bereitstellung des Beispiels

sadf

Literatur

[1] Kalauner Hollander. Opencl example. `https://github.com/ReneHollander/opencl-example.git`. zuletzt besucht: 15.01.2016.

[2] Nvidia. Cuda toolkit documentation. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#simt-`. zuletzt besucht: 23.01.2016.

[3] Elektronik-Kompendium.de. Gpgpu - general purpose computation on graphics processing unit. `http://www.elektronik-kompendium.de/sites/com/1405281.htm`. zuletzt besucht: 23.01.2016.

[4] Kalauner Hollander. Gpgpu - ausarbeitung. `https://elearning.tgm.ac.at/mod/resource/view.php?id=45929`. zuletzt besucht: 23.01.2016.

[5] Nvidia. Cuda toolkit. `https://developer.nvidia.com/cuda-toolkit`. zuletzt besucht: 23.01.2016.

[6] http://www.gp sme.eu/. A general toolkit for "gputilisation" in sme applications. `http://www.gp-sme.eu/`. zuletzt besucht: 23.01.2016.

[7] http://www.computerhope.com/jargon/t/transcompiler.htm. Transcompiler. `http://www.computerhope.com/jargon/t/transcompiler.htm`. zuletzt besucht: 23.01.2016.

Listings

Abbildungsverzeichnis