GPUs

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Johannes Coym
E-Mail-Adresse:	4coym@informatik.uni-hamburg.de
Matrikelnummer:	6693524
Studiengang:	Wirtschaftsinformatik

Hamburg, den 18.03.2017

Inhaltsverzeichnis

1 Einleitung

GPUs sind Prozessoren, welche dazu optimiert sind, Grafiken zu berechnen und diese an einen oder mehrere Monitore auszugeben. Da sich jedoch viele andere Berechnungen auf ähnliche Art durchführen lassen, wie grafische Berechnungen, gibt es seit einigen Jahren die Möglichkeit die GPUs hierfür mitzuverwenden. In den folgenden Kapiteln wird deshalb behandelt, was GPUs von CPUs unterscheidet und wie dieser Unterschied zur Beschleunigung von Software nutzbar ist.

Das zweite Kapitel beginnt hierfür damit, was GPUs von CPUs unterscheidet, wobei auf die jeweiligen Vor- und Nachteile eingegangen wird und zum Ende des Kapitels gezeigt wird, was diese Unterschiede leistungstechnisch bedeuten. Im dritten Kapitel wird dann auf einige Technologien eingegangen, welche einem die Möglichkeit bieten die Rechenleistung der GPUs zu nutzen. Zum Abschluss folgt zunächst ein kurzes Fazit zu GPUs und den verschiedenen Technologien und zuletzt folgt noch einen kurzer Ausblick, wie GPUs sich weiterentwickeln werden und was noch von Ihnen zu erwarten ist.

2 Vergleich mit CPUs

2.1 Vor- und Nachteile gegnüber CPUs

CPUs und GPUs unterscheiden sich schon grundlegend in der Architektur. Während CPUs darauf ausgelegt sind jede mögliche Aufgabe, die ihnen gestellt wird, möglichst gut zu bewältigen, sind GPUs ausschließlich für grafische Berechnungen optimiert und für sonstige Aufgaben entweder ungeeignet oder sehr langsam. Aus diesem Grund haben GPUs auch eine deutlich höhere Anzahl an Recheneinheiten als CPUs, jedoch laufen diese mit einem niedrigeren Takt. Als Beispiel sind im Folgenden die Rechenkerne und der Takt von einer aktuellen Desktop-CPU, dem Intel Core i7-7700k, einer aktuellen Server-CPU, dem Intel Xeon E5-2699 v4 und einer GPU für High Performance Computing, der Nvidia Tesla P100:

- Intel Core i7-7700k: 4 Rechenkerne(8 virtuelle Rechenkerne), bis zu 4,5 GHz
- Intel Xeon E5-2699 v4: 22 Rechenkerne(44 virtuelle Rechenkerne), bis zu 3,6 GHz
- Nvidia Tesla P100: 3584 Shader Processing Units(SPUs), bis zu 1,48 GHz

Hierbei sieht man, dass die GPU durch ihre deutlich höhere Anzahl von Kernen den niedrigeren Takt kompensieren kann und eine deutlich höhere Leistung als die CPUs erreicht, aber die CPUs haben hier den Vorteil, dass durch ihre Architektur jeder virtuelle Rechenkern etwas anderes ausführen kann und hierdurch einige verschiedene Aufgaben gleichzeitig bewältigt werden können. Bei GPUs lässt sich die volle Leistung jedoch nur mit hochgradig parallelisierbaren Anwendungen rausholen, da die 3584 SPUs einer Tesla P100 zum Beispiel in 56 Streaming Multiprocessors(SM), die jeweils 64 SPUs enthalten, unterteilt sind. Diese 64 SPUs, die in einem SM zusammengefasst sind, sind hierbei jedoch lediglich dazu fähig eine Aufgabe zur Zeit zu bewältigen, weshalb serielle Berechnungen auf CPUs viel schneller berechnet werden können.

Aus diesem Grund setzen Programme auch nicht darauf komplett auf der GPU ausgeführt zu werden, stattdessen bietet sich die Möglichkeit an leistungsintensive, hochgradig parallelisierbare Berechnungen auf die GPU zu verlagern, damit die CPU für serielle Berechnungen freigegeben wird. Ein weiterer Nachteil der GPUs ist hierbei jedoch, dass die Programmteile, welche auf der GPU ausgeführt werden sollen, je nach verwendeter Technologie, eventuell komplett neu geschrieben werden müssen, was ein enormer Aufwand sein kann.

Abbildung 2.1: Aufbau eines SM bei der Tesla P100, `http://cdn.wccftech.com/wp-content/uploads/2016/04/gp100_SM_diagram.png`

2.2 Leistung gegenüber CPUs

Um zu zeigen, wie groß der Leistungsvorteil von GPUs im Vergleich zu CPUs bei parallelisierbaren Berechnungen ist, sind im Folgenden Leistungswerte von den CPUs und der GPU aus dem letzten Beispiel bei Double Precision Float Berechnungen(FP64), welche bei High Performance Computing sehr wichtig sind.

- Intel Core i7-7700k: 0,2 TFLOPS
- Intel Xeon E5-2699 v4: 1,4 TFLOPS [Lin17]
- Nvidia Tesla P100: 5,3 TFLOPS [P1017]

Da der Energieverbrauch jedoch auch ein wichtiger Faktor ist, bietet es sich hierbei auch an, zu vergleichen, wie groß die Leistungsunterschiede sind, wenn die Leistung in Relation zum Stromverbrauch betrachtet wird:

- Intel Core i7-7700k: 2,2 GFLOPS/Watt
- Intel Xeon E5-2699 v4: 9,66 GFLOPS/Watt
- Nvidia Tesla P100: 17,68 GFLOPS/Watt

Hierbei sieht man, dass die GPU auch unter Einbeziehung des Energieverbrauchs noch deutlich schneller ist, jedoch ist der Unterschied deutlich niedriger, nachdem die CPUs weniger Strom verbrauchen.

3 Technologien

3.1 OpenCL

Um diesen Geschwindigkeitsvorteil nutzen zu können, gibt es verschiedene Technologien um dies in seinem Quellcode zu verwenden. Eine von diesen Technologien ist OpenCL, welche ursprünglich von Apple entwickelt wurde und seit 2008 standadisiert ist. OpenCL funktioniert dabei jedoch nicht nur mit GPUs, sondern grundsätzlich mit jeder Art von Prozessoren, wie zum Beispiel CPUs oder Soundprozessoren.

Um OpenCL in seinen Programmen zu verwenden bietet OpenCL dabei 2 eigene Programmiersprachen „OpenCL C“ und „OpenCL C++“. OpenCL C basiert dabei auf dem Standard C99 und OpenCL C++ basiert auf dem Standard C++14, jedoch erweitern sie diese dabei aber um ein paar Datentypen, schränken sie aber gleichzeitig in manchen Bereichen auch ein. Desweiteren bietet OpenCL, zumindest unter Windows, auch noch die Möglichkeit direkt auf Objekte der Grafikschnittstellen OpenCL und DirectX zuzugreifen, um diese direkt im OpenCL Code zu verwenden.

OpenCL hat jedoch das Problem, dass es sehr schwierig ist als Anfänger in die Programmiersprachen reinzukommen, da sie kompliziert zu verwenden ist. Ein weiteres Problem dabei ist, dass vorhandener Code in C oder C++ auch nicht direkt in OpenCL verwendet werden kann, sondern zunächst angepasst oder komplett neu geschrieben werden muss, damit er auf der GPU lauffähig ist.

Als Beispiel für die unterschiedlichen Technologien folgt nun am Ende jedes Abschnitts ein Ausschnitt, wie Quellcode für eine „Single-Precision A*X Plus Y“(SAXPY) Berechnung auf der GPU aussieht. Der erste solche Ausschnitt hierfür ist in OpenCL C und enthält das Kopieren der Daten in den Grafikspeicher und Starten der Operation, wobei, im Vergleich mit den noch kommenden Technologien, sichtbar wird, inwiefern OpenCL schwerer anzuwenden ist.

```
1  __kernel void SAXPY (__global float* x, __global float* y,
       ↪ float a)
2  {
3      const int i = get_global_id (0);
4      y [i] += a * x [i];
5  }
6  ...
7  // Create memory buffers on the device for each vector
8  cl_mem A_clmem = clCreateBuffer(context,
       ↪ CL_MEM_READ_ONLY,VECTOR_SIZE * sizeof(float), NULL,
       ↪ &clStatus);
9  cl_mem B_clmem = clCreateBuffer(context,
       ↪ CL_MEM_READ_ONLY,VECTOR_SIZE * sizeof(float), NULL,
       ↪ &clStatus);
10 cl_mem C_clmem = clCreateBuffer(context,
       ↪ CL_MEM_WRITE_ONLY,VECTOR_SIZE * sizeof(float), NULL,
       ↪ &clStatus);
11 ...
12 size_t global_size = VECTOR_SIZE;
13 size_t local_size = 64;
14 clStatus = clEnqueueNDRangeKernel(command_queue, kernel, 1,
       ↪ NULL, &global_size, &local_size, 0, NULL, NULL);
```

Listing 3.1: SAXPY OpenCL C [Ope17]

3.2 OpenACC

Eine weitere dieser Technologien ist OpenACC, welches genauso, wie OpenCL, eine offene Schnittstelle zur Beschleunigung von Programmen mit Hilfe von anderen Prozessoren ist. OpenACC unterstützt hierbei im Gegensatz zu OpenCL lediglich GPUs und keine sonstigen Prozessoren.

Die Syntax von OpenACC ist jedoch sehr ähnlich zu der von OpenMP, welches zur Beschleunigung von Programmen auf mehreren CPU-Kernen dient, und lässt sich auch zusammen mit dieser im gleichen Programm verwenden. Hierfür müssen nur im bereits vorhandenen Quallcode mit kurzen pragma-Anweisungen markiert werden, damit der Compiler weiß, dass er diesen Codeteil für eine Verwendung auf der GPU kompilieren soll. Verfügbar ist OpenACC hierbei für die Programmiersprachen C, C++ und Fortran, benötigt jedoch derzeit einen speziellen Compiler und diese Compiler sind kostenpflichtig.

Auf der OpenACC-Website gibt es zu OpenACC bei Klimaberechnungen auch einen Leistungstest, bei dem 4 verschiedene Klimaberechnungen zunächst auf einem Intel Sandy Bridge Prozessor mit 8 Kernen und MPI ausgeführt wurde und anschließend noch auf lediglich einem CPU-Kern und einer Nvidia K20X mit OpenACC ausgeführt

wurde. [ACC17] Das Ergebnis hierbei war, dass die Berechnungen auf der CPU mit MPI 57,6 Sekunden gedauert haben, die Berechnungen mit OpenACC auf der GPU jedoch lediglich 13,7 Sekunden gedauert haben, was etwa die 4,2-fache Berechnungsgeschwindigkeit bedeutet. Was hierbei aber noch zu beachten ist, dass eine K20X noch einen deutlich höheren Stromverbrauch(235 W) hat, als eine Intel Sandy Bridge 8-Kern CPU(115-135 W) und zusätzlich noch einen Prozessorkern benötigt. Somit ist der Geschwindigkeitsvorteil unter Einbeziehung des Energieverbrauchs nur noch etwa beim Faktor 2.

Im Folgenden iist nun noch eine SAXPY-Operation in C, welche eine einzelne pragma-Anweisung enthält, welche dafür sorgt, dass dieser Code auf der GPU ausgeführt wird.

```
1 void saxpy(int n, float a, float * restrict x, float *
    ↪ restrict y)
2 {
3     #pragma acc kernels
4     for (int i = 0; i < n; ++i)
5     y[i] = a*x[i] + y[i];
6 }
7 ...
8 // Perform SAXPY on 1M elements
9 saxpy(1<<20, 2.0, x, y);
```

Listing 3.2: SAXPY OpenACC [SAX17]

3.3 CUDA

Die dritte Technologie, die in diesem Kapitel vorgestellt wird, ist CUDA, welches eine Sammlung von Schnittstellen und Tools zur Verwendung von Nvidia GPUs für nichtgrafische Anwendungen ist. Die Technologie ist hierbei exklusiv auf Grafikprozessoren von Nvidia beschränkt und kann nicht für GPUs anderer Hersteller oder sogar anderer Prozessoren verwendet werden.

CUDA enthält hierfür 4 Möglichkeiten, mit denen Programme mit Hilfe der GPU Beschleunigen werden können. Hierfür gibt es sowohl spezielle Bibliotheken, welche bestimmte Funktionen mit der GPU beschleunigen können, als auch eine Implentation von OpenACC, um bestimmte Codeabschnitte zu beschleunigen, sowie eigene CUDA C/C++ Programmiersprachen für parallele Programmierung und ein eigenes SDK für Deep Learning. [CUD17a]

Die eigenen Programmiersprachen „CUDA C“ und „CUDA C++“ bieten hierbei eine ähnliche Funktionalität, wie Open CL C/C++, jedoch sind diese für Nvidia GPUs schneller als OpenCL, da diese hierfür speziell optimiert sind, desweiteren sind die CUDA Programmiersprachen deutlich leichter zu verwenden, da diese auch nicht so komplex für diverse Prozessorarten lauffähig sein müssen und sind so auch zum Einstieg deutlich leichter.

Verwendung finden die Schnittstellen und Tools von CUDA bereits in hunderten Anwendungen und zu den Anwendungsbereichen hierfür zählen zum Beispiel Datenwissenschaft, Maschinelles Lernen, Physik und auch Klimaberechnungen.[CUD17b]

Das letzte Beispiel ist nun eine SAXPY Operation in CUDA C++, welcher, genauso wie der OpenCL C Abschnitt, das Kopieren der Daten in den Grafikspeicher und den Start der Operation enthält, hierbei jedoch übersichtlicher als OpenCL ist.

```
1  __global__ void saxpy(int n, float a, float * restrict x,
       ↪ float * restrict y)
2  {
3      int i = blockIdx.x*blockDim.x + threadIdx.x;
4      if (i < n) y[i] = a*x[i] + y[i];
5  }
6  ...
7  int N = 1<<20;
8  cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
9  cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);
10
11 // Perform SAXPY on 1M elements
12 saxpy<<<4096,256>>>(N, 2.0, x, y);
```

Listing 3.3: SAXPY CUDA C++ [SAX17]

4 Fazit

Letztendlich ist OpenACC als rein Compiler-basierte Schnittstelle mit Abstand am leichtesten zu verwenden und kann hierbei auch mit anderen Technologien, wie zum Beispiel OpenMP im gleichen Programm verwendet werden. Desweiteren muss hierfür der Quellcode nicht neu geschrieben werden und es können auch sehr einfach lediglich bestimmte Teile mit der GPU beschleunigt werden.

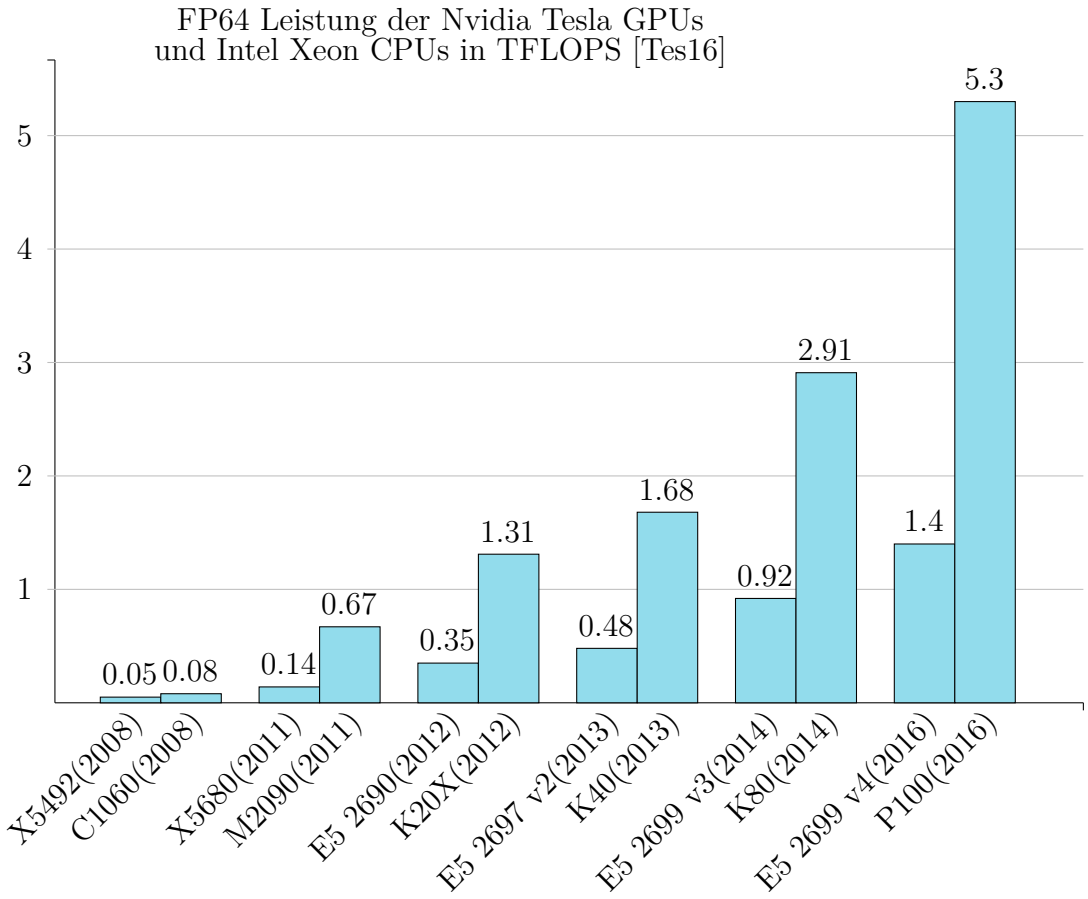
Jedoch haben CUDA und OpenCL hierbei auch ihre Daseinsberechtigung, da sie eine deutlich hardwarenähere Unterstützung als OpenACC bieten und sich so auch besser optimieren lassen. Der Nachteil bei diesen ist jedoch, dass sie durch diese Hardwarenähe deutlich schwieriger zu verwenden sind als OpenACC und außerdem voraussetzen, dass der gesamte Quellcode umgeschrieben werden muss. Unter diesen beiden Technologien ist CUDA jedoch eindeutig, die leichtere zu verwenden, was aber auch daran liegt, dass CUDA auch nur für GPUs von Nvidia verwendbar ist, während OpenCL mit jedem Prozessor funktioniert.

Am Ende mussfür ein Projekt immer eine Entscheidung getroffen werden, was die Voraussetzungen sind und welches Prinzip dafür am besten geeignet ist. Ist bereits Quellcode vorhanden, welcher beschleunigt werden soll, indem er auf der GPU ausgeführt wird, ist häufig OpenACC die beste Lösung, sofern keine Vorraussetzung ist durch diverse Optimierungen am GPU-Code weitere Leistung rauszuholen. Wenn noch kein Quellcode vorhanden ist oder den vorhandenen Quellcode für die GPUs dann auch möglichst gut optimieren will, dann sollte die Entscheidung darauf basieren auf welchen GPUs das Programm lauffähig sein soll. Lautet die Antwort auf diese Frage, dass dies nur auf Nvidia GPUs der Fall sein sollte, dann ist CUDA die bessere Technologie für einen, da diese leichter zu verwenden ist und für Nvidia GPUs optimiert ist. Ansonsten bleibt noch OpenCL übrig, da einem diese Technologie die meisten Möglichkeiten bietet um die Programme zu optimieren und auf allen GPUs lauffähig zu machen.

5 Ausblick

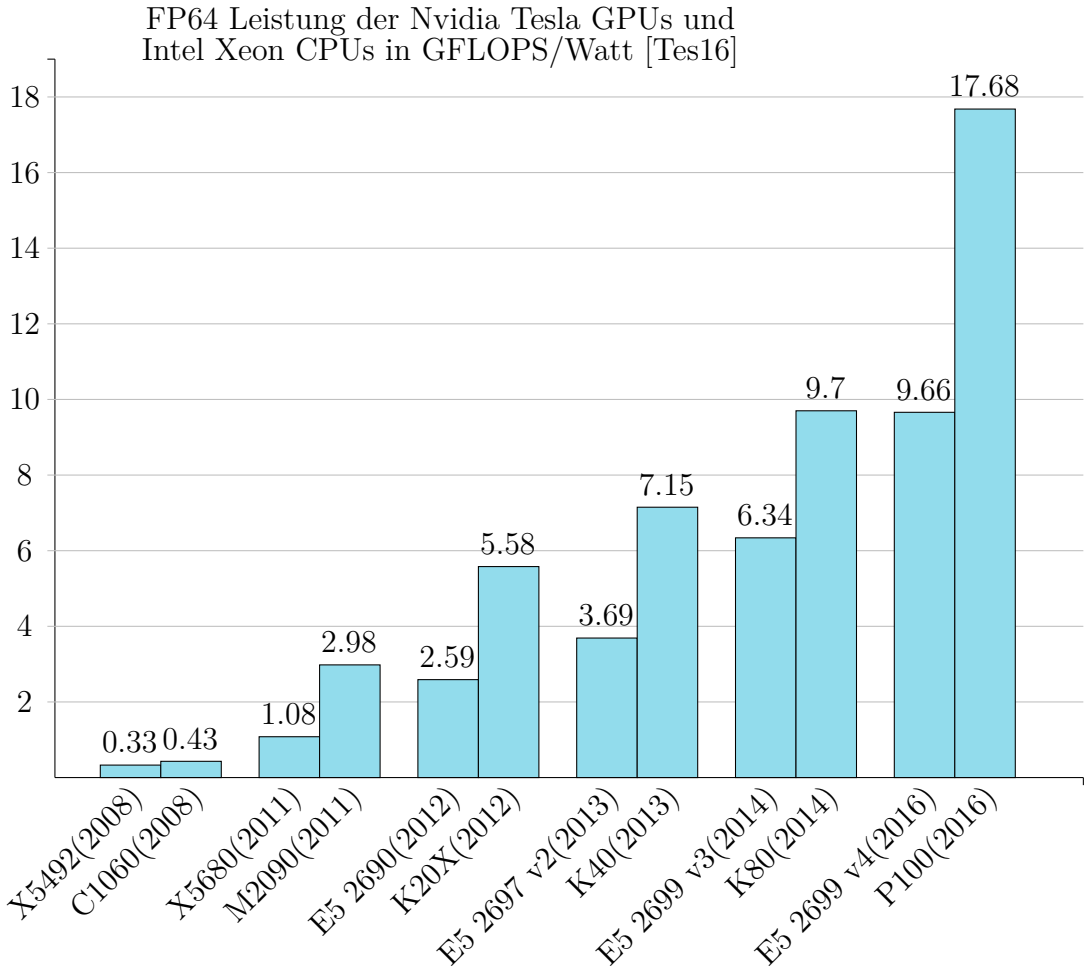
Um zu sehen, wie sich die GPUs für Berechnungen noch entwickeln können, lohnt es sich die Leistung der letzten Generationen von GPUs und CPUs zu betrachten. Hierfür wird in der folgenden Grafik die Leistung einiger Nvidia Tesla GPUs seit 2008 betrachtet, im Vergleich zu den in den selben Jahren erschienenen Intel Xeon CPUs.

Wie anhand der Graphen gut sichtbar ist, haben die GPUs zwischen 2008 und 2011 deutlich größere Leistungsfortschritte als die CPUs gemacht, jedoch waren die folgenden Leistungsfortschritte nur noch größtenteils proportional zu denen der CPUs, teilweise sogar schlechter.



Aktuell haben die GPUs zwar knapp die vierfache Leistung der CPUs, jedoch ist hier auch wieder nicht der höhere Stromverbrauch betrachtet, deshalb folgt eine weitere Grafik, welche die Leistung pro Watt Verbrauch der selben Prozessoren gegenüberstellt.

Den Stromverbrauch einzubeziehen verringert hierbei den Vorsprung der GPUs deutlich und gerade in den letzten Generationen haben die CPUs eher wieder auf die GPUs aufgeholt, nachdem die CPU Generation von 2016 hier auch schon mit den GPUs von 2014 mithalten kann. Dennoch schaffen die GPUs es derzeit noch fast die doppelte Leistung beim identischen Stromverbrauch zu erzeugen, auch wenn der Vorsprung nicht mehr so deutlich ist, wie ohne Einbezug des Stromverbrauchs.



Mit diesem Vorteil in der Energieeffizienz kann es für bestimmte Berechnungen, die leicht mit einer GPU beschleunigt werden können, ein großer Vorteil sein zusätzlich zu den CPUs auch GPUs für die Berechnung zu verwenden. In der näheren Zukunft wird sich die Entwicklung wohl auch ähnlich fortsetzen, womit die GPUs weiter, für manche Gebiete, Anwendung finden werden.

Jedoch könnten die GPUs noch Konkurrenz in diesen Gebieten durch Many-Core-

CPUs, wie die Intel Xeon Phis bekommen, welche im Gegensatz zu den GPUs sogar in der Lage sind ohne weitere CPU lauffähig zu sein. Bei diesen CPUs muss sich jedoch erst noch zeigen, wie diese sich etablieren, jedoch zeigen die aktuellen Modelle schon, dass sie mit etwa 14,3 GFLOPS/Watt schon nah an die Energieeffizienz der GPUs herankommen. [Xeo17]

Literaturverzeichnis

[ACC17] Weather, climate and ocean modeling. http://www.openacc.org/content/applications/Weather, 22.02.2017.

[CUD17a] Accelerated computing. https://developer.nvidia.com/accelerated-computing, 22.02.2017.

[CUD17b] Gpu computing anwendungen. http://www.nvidia.de/object/gpu-computing-applications-de.html, 22.02.2017.

[Lin17] Measuring performance of intel broadwell processors. http://en.community.dell.com/techcenter/high-performance-computing/b/general_hpc/archive/2016/03/31/measuring-performance-of-intel-broadwell-processors-with-high-performance-computing-benchmarks, 11.01.2017.

[Ope17] An example of opencl program. https://www.packtpub.com/mapt/book/Application-Development/9781849692342/1/ch01lvl1sec12/An+example+of+OpenCL+program, 26.02.2017.

[P1017] Tesla p100. http://www.nvidia.de/object/tesla-p100-de.html, 11.01.2017.

[SAX17] Six ways to saxpy. https://devblogs.nvidia.com/parallelforall/six-ways-saxpy/, 26.02.2017.

[Tes16] Nvidia tesla. https://en.wikipedia.org/wiki/Nvidia_Tesla, 30.11.2016.

[Xeo17] Intel veröffentlicht xeon phi mit bis zu 7 teraflops. https://www.golem.de/news/knights-landing-intel-veroeffentlicht-xeon-phi-mit-bis-zu-7-teraflops-1606-121642.html, 27.02.2017.

Abbildungsverzeichnis