OpenCL

Seminar Programmiersprachen im Multicore-Zeitalter
Universität Siegen
Tim Wiersdörfer
tim.wiersdoerfer@student.uni-siegen.de

Abstract: In diesem Dokument wird ein grundlegender Einblick in das relativ neue Programmier-Modell, für Multicore-Prozessoren, OpenCL (Open Computing Language) gewährt. Und auf deren Funktionsweise, in Verbindung mit verschiedenen Plattformen, wie zum Beispiel CPUs und GPUs eingegangen. Des Weiteren wird sich in diesem Dokument mit den signifikantesten Eigenschaften der, mit diesem Modell verbundenen, eigenen Programmiersprache OpenCL C und dessen Vor- und Nach-Teile befasst.

Inhaltsverzeichnis

1 Was ist OpenCL?

OpenCL steht für “Open Computing Language” und ist ein offenes Standard-Modell zur parallelen und plattform-unabhängigen Programmierung. Die bedeutet es kann zu der Programmierung von unterschiedlicher heterogenen Plattformen zum Einsatz kommen, dabei ist es unbedeutend ob es sich dabei um CPUs[1], GPUs[2], DSPs[3] oder Cell Prozessoren[4] handelt. Die einzige Voraussetzung ist, dass die entsprechenden Systeme OpenCL, sowie die dazugehörige Programmiersprache „OpenCL C“ unterstützen. Hierzu abstrahiert OpenCL von der realen Hardware, in dem drei abstrakte Modelle eingeführt werden:

- Das Plattform-Modell
- Das Ausführungs-Modell
- Das Speicher-Modell

Diese drei Modelle beschreiben den Aufbau und die Funktionsweise der Hardware, welche OpenCL Programme ausführen können.

1.1 Entwicklung von OpenCL

Ursprünglich wurde OpenCL von der Firma Apple entwickelt, welche 2008 die Zusammenarbeit mit den Firmen NVIDIA, Intel, AMD und IBM einging und die „OpenCL Working Group“ bildete, mit dem Ziel diese auf Hardware unterschiedlicher Hersteller zu implementieren.

Am 8. Dezember 2008 wurde die Spezifikation für OpenGL 1.0 von der Khronos Group[5] als standardisierte Programmierschnittstelle für GPU Computing veröffentlicht. Darauf folgte am 16. November 2011 die bis zum jetzigen Zeitpunkt aktuelle Spezifikation 1.2 mit Verbesserungen, welche zu der Vorgängerversion 1.0 abwärtskompatibel bleiben.

Des Weiteren wurde OpenCL erstmals am 28. August 2009 von Apple mit dem Betriebssystem Mac OS X 10.6 (Snow Leopard) auf den Markt gebracht, bei welchem die dazugehörigen Programme, die sogenannten Kernel, zur Laufzeit auf verschiedenen vorhandenen OpenCL-fähigen Geräten verteilt werden können.

[1] CPU: Central Processing Unit, ist die zentrale Verarbeitungseinheit eines Computers
[2] GPU: Graphics Processing Unit, dient zur Berechnung der Bildschirmausgabe auf einem Computer
[3] DSP: Digital Signal Processor, dient der Bearbeitung von digitalen Signalen.
[4] Cell Prozessoren: sind Prozessoren, welche IBM gemeinsam mit Sony und Toshiba entwickelt hat und welche für paralleles Rechnen prädestiniert sind.
[5] Khronos Group: „ist ein im Jahr 2000 gegründetes Industriekonsortium, das sich für die Erstellung und Verwaltung von offenen Standards im Multimedia-Bereich auf einer Vielzahl von Plattformen und Geräten einsetzt.“ Zitat Wikipedia

1.2 Warum OpenCL?

Auf Grund von technischen Schwierigkeiten, die Leistung von Prozessoren durch eine höhere Taktrate zu steigern und den daraus resultierenden Mehrkosten, wurden sowohl für CPUs, als auch für GPUs, Multicore-Prozessoren[6] entwickelt, welche häufig kostengünstiger sind. Dies hat jedoch zur Folge, dass auch die Programmiersprachen diese neue Eigenschaften der Prozessoren unterstützen müssen um auch bei rechenintensiven Anwendungen eine gute Performance zu gewährleisten. Und genau hier findet OpenCL ihren Platz. Denn unter Ausnutzung dieser heterogenen Plattform[7] und der Unterstützung verschiedener Geräte und Architekturen durch portablen Code[8], kann mit OpenCL nicht nur auf Multicore-CPUs, sondern, und vor allem, auch auf Multicore-GPUs zurückgreifen.

Im Vergleich zu CPUs bringen GPUs verschiedene Vor- und Nachteile mit sich. CPUs besitzen nur wenige ALUs, in den meisten Fällen 1-4, haben jedoch einen sehr komplexen Befehlssatz und sind auf das Ziel optimiert sequenzielle Aufgaben schnellstmöglich Abzuarbeiten. GPUs hingegen besitzen mehrere Hunderte und damit eine wesentliche größere Anzahl an ALUs, welche auf eine massive Parallelisierung spezialisiert sind. Jedoch einen begrenzteren Befehlssatz aufweisen als CPUs.

Die Herausforderung welche OpenCL also zu bewältigen hat, sind die großen Unterschiede in den Programmieransätzen von Multicore-CPUs und –GPUs. Im Besonderen bei den GPGPU[9] Programmiermodellen, welche über sehr komplexe Speicherhierarchien, sowie Vektor-Operationen verfügen welche allesamt Plattform-, Hersteller- und Hardware-spezifisch sind, ist es schwer diese unter einem gemeinsamen Sprach-Standard zu vereinen.

2 Das Plattform-Modell

Das OpenCL Plattform-Modell stellt das abstrakte Grundgerüst eines OpenCL-Systems dar, welches jedoch von jeder unterschiedlichen Hardware auf eine andere Art und Weise umgesetzt wird. Grundsätzlich besteht ein OpenCL-System aus einem Host[10] und mehreren OpenCL Devices (im englischen auch „compute device), siehe Abbildung 1. OpenCL schreibt vor, dass ein solches Device aus einer oder mehreren unabhängig voneinander arbeitenden Recheneinheiten (englisch „compute unit“, kurz „CU“, siehe Abbildung 1) aufgebaut ist. Diese CU sind bei Mehrkernprozessoren die verfügbaren Kerne, welche zusammengefasst die CPU ergeben und bei einer Grafikkarte und deren GPU sind es die Shader[11]. Des Weiteren besteht eine solche Recheneinheit oder auch Compute Unit, aus ein oder mehreren ausführenden Elementen (englisch „processing element“, kurz „PE“, siehe Abbildung 1).

[6] Multicore-Prozessor: ist ein Mikroprozessor mit mehr als einem vollständigen Hauptprozessor auf einem einzigen Chip.
[7] Heterogene Plattform: ist eine Plattform von unterschiedlichen physikalischen Eigenschaften
[8] Portabler Code: bezeichnet man Programm-Code, welcher Betriebssystem-unabhängig verwendet werden kann
[9] GPGPU: General Purpose Computation on Graphics Processing Unit, bezeichnet die Verwendung von Grafikprozessoren für Aufgaben über deren ursprünglichen Aufgabenbereich hinaus
[10] Host: bezeichnet in diesem Fall, den zentralen Verwaltungsknoten hinter dem Multicore-Systems
[11] Shader: sind kleine Recheneinheiten in aktuellen Grafikchips

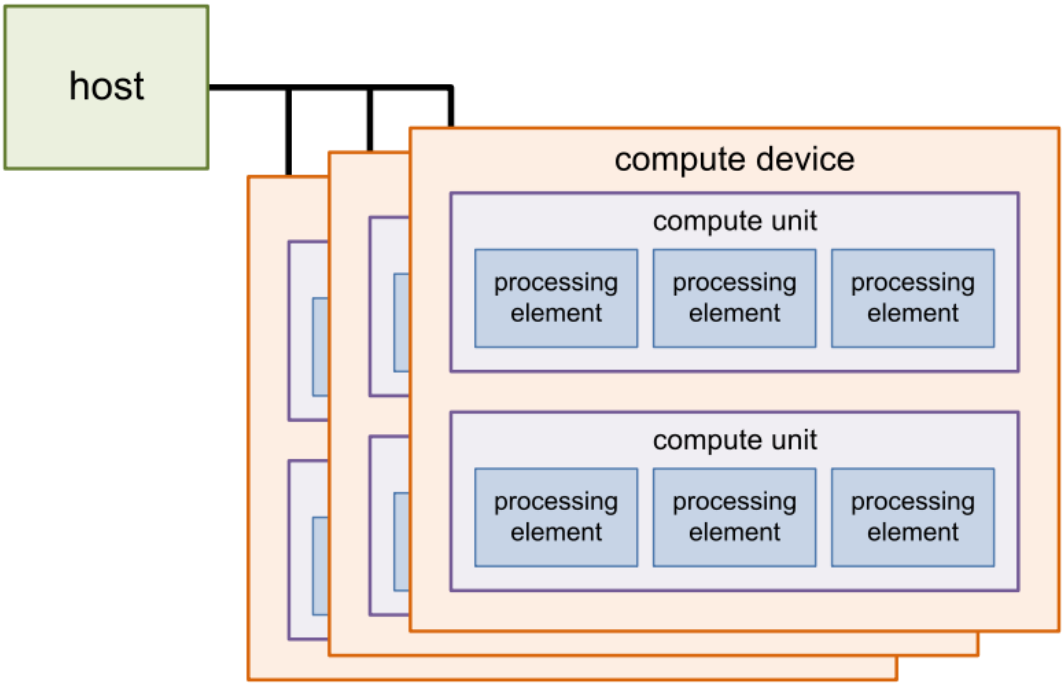

Abbildung 1: Graphische Darstellung des Plattform-Modells

Der Host verteilt dabei die Kernel[12] zur Laufzeit an die verfügbaren Geräte. Hierbei existieren zwei verschiedene Arten von Kernel:

- **OpenCL-Kernel:**

  Dies ist eine Funktion, welche auf einem OpenCL-Device ausgeführt wird und welche in der OpenCL-eigenen Programmiersprache OpenCL C geschrieben wurde. Und werden erst zur Laufzeit vom OpenCL-Compiler[13] übersetzt und daraufhin von einem OpenCL-Device ausgeführt. Dies hat zur Folge, dass zur Entwicklungszeit nicht bekannt sein muss, auf welcher Hardware das Programm ausgeführt werden soll und ist somit Plattform-unabhängig.

- **Native Kernel:**

  Hierbei handelt es sich ebenfalls um eine Funktion, welche auf einem OpenCL-Device ausgeführt werden kann, jedoch sind diese optional und implementierungsspezifisch.

Betrachtet man dieses Modell in einem konkreten System, so wird man feststellen, dass diese einzelnen Rollen von unterschiedlicher Hardware übernommen werden können. Dieses wird im Folgenden an Hand von zwei Beispielen verdeutlicht:

[12] Kernel: englisch für Kern, bezeichnet einen elementaren Bestandteil eines Systems
[13] OpenCL-Compiler: ist ein Programm zum Übersetzen von OpenCL Quelltext in Maschinensprache

1. **Ein System besteht aus einer CPU und einer GPU:**

   Den Host bildet in diesem Fall die CPU, zusammen mit dem Hauptspeicher. Diese beiden führen die OpenCL-Anwendung aus und steuern die Arbeit des Devices, welches in diesem Fall die GPU ist. Die Rolle der Compute Unit wird nun von einem SIMD-Core im GPU übernommen und ein Processing Element steht für einen Streaming-Processor[14].

2. **Ein System besteht aus einer Multicore-CPU:**

   Im diesem Szenario führt ebenfalls die CPU, zusammen mit dem Hauptspeicher, die OpenCL-Anwendung aus und bildet damit den Host. Jedoch wird in diesem Fall ebenfalls von der CPU die Rolle des Devices übernommen, d.h. jeder Kern der CPU steht nun für eine Compute Unit, welche aus einem einzelnen Processing Element besteht.

3 Das Ausführungs-Modell

Bei der Ausführung einer OpenCL-Anwendung wird zwischen zwei verschiedenen Teilen unterschieden:

- Der sogenannte *Kernel*, dessen mehrere Instanzen parallel auf einem OpenCL-Device ausgeführt werden
- Das Host-Programm

Diese beiden Anwendungsteile, werden im Folgenden tiefergehend und umfassender beschrieben.

3.1 Kernel

Ein Kernel ist eine Funktion, welche von einem OpenCL-Device auf parallele Art und Wiese ausgeführt wird. Hierzu werden mehrere Instanzen eines Kernels gestartet, eine solche Instanz nennt man Work-Item (dies entspricht in etwa dem Begriff von einem Thread). Ein jedes solcher Work-Item wird eine *globalID* zugewiesen und wird immer sequentiell ausgeführt.

3.2 Host-Programm

Das Host-Programm wird auf dem Host ausgeführt, wo es zur Laufzeit, die Kernel auf die verschiedenen, angeschlossenen OpenCL-Devices, so verteilt, dass diese auf dem am besten passenden Device ausgeführt werden. Hierbei kann die Auswahl auf Basis der folgenden Gründe getroffen werden:

- Maximale Anzahl von Recheneinheiten
- Höchste Taktfrequenz
- Größter Speicher
- Und viele weitere gerätespezifischen Eigenschaften

[14] Streaming-Processor; englisch für einen Prozessor, welcher in der Lage ist Datenströme zu verarbeiten

Des Weiteren ist das Host-Programm dafür zuständig alle Kernelinstanzen zu verwalten und zu steuern. Hierzu definiert dieser beim Start eines Kernels, wie viele Instanzen, bzw. Work-Items, des Kernels gestartet werden sollen. Zusätzlich werden die Work-Items in sogenannte Work-Groups unterteilt, hierbei gilt, dass alle Work-Groups in einem Programm dieselbe Größe haben und, dass jede eine eigene *workGroupID* erhält. Da jedes Work-Item von einem Processing Element eines OpenCL-Devices ausgeführt wird, und da alle Work-Items aus einer Work-Group von einer Compute Unit gemeinsam ausgeführt werden, ist es daher nur Work-Items aus derselben Work-Group möglich miteinander zu kommunizieren.

3.3 Warteschlange (englisch „command-queue“)

Um einen möglichst effizienten Programmfluss zu gewährleisten, werden Kernelinstanzen nicht direkt auf einem OpenCL-Device gestartet, sondern in eine Warteschlange für dieses Device eingereiht. Diese Warteschlange nennt sich command-queue und wird von dem Host-Programm genutzt, um Kernel-Ausführungen, Speicher-Operationen und Synchronisations-Operationen durchzuführen. Die Warteschlange wird daraufhin, automatisch, von dem entsprechenden OpenCL-Device abgearbeitet, was bedeutet, wenn ein Befehl abgearbeitet ist, wird sofort das nächste aus der Warteschlange bearbeitet.

4 Das Speicher-Modell

In OpenCL wird der vorhandene Speicher in verschiedene logische Bereiche eingeteilt, sodass, ein OpenCL-Device einen, von dem Host (Hostspeicher, englisch „host memory“ siehe Abbildung X) logisch getrennten, eigenen Speicher besitzt. Dies hat zur Folge, dass Daten, welche in einem Kernel zur Verfügung stehen sollen, explizit zu dem entsprechenden Device kopiert werden müssen. Bei dem Hostspeicher handelt es sich hierbei also um den regulären Arbeitsspeicher des Leitprogramms, auf den ein OpenCL-Kernel keinen direkten Zugriff hat, siehe Abbildung 2.

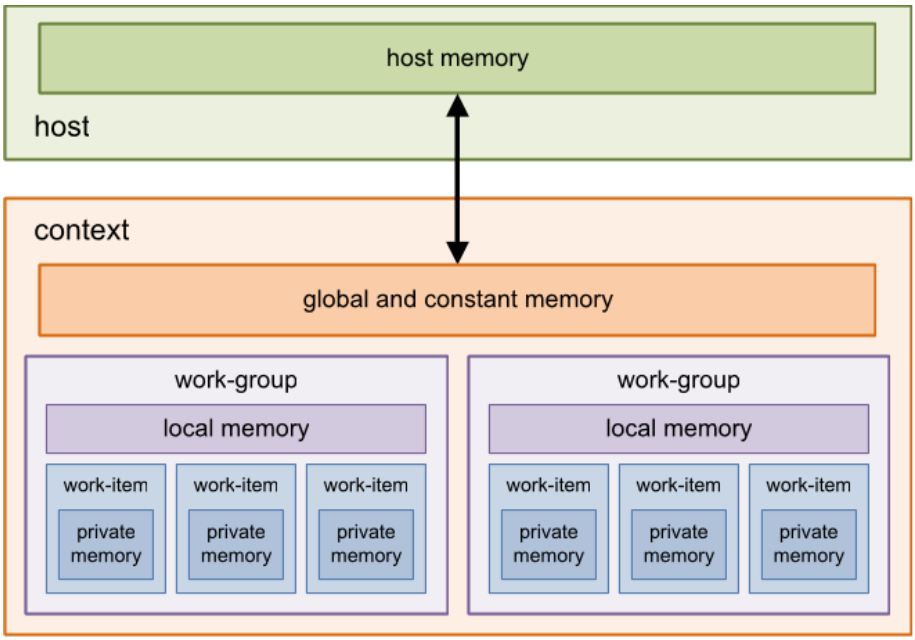

Abbildung 2: Darstellung des OpenCL Speicher-Modells

Wie auf der Abbildung 2 zu sehen ist, unterscheidet OpenCL zusätzlich zwischen vier weiteren Speicherbereichen innerhalb eines OpenCL-Devices:

- **Globaler Speicher (englisch „global memory“)**

  Hierbei handelt es sich um den größten Speicher, welcher vergleichbar ist, mit dem Arbeitsspeicher. Alle Work-Items in allen Work-Groups haben darauf Lese- und Schreibzugriff.

- **Konstanter Speicher (englisch „constant memory“)**

  Dies ist ein Bereich des globalen Speichers, der sich darin von dem restlichen Speicher unterscheidet, dass die Kernel-Instanzen, bzw. die Work-Items in allen Work-Groups aus diesem Speicher ausschließlich Lesen dürfen.

- **Lokaler Speicher (englisch „local memory“)**

  Dieser, in der Regel 16 kiB großer, Speicher ist lokal, ausschließlich für eine Work-Group. Alle Work-Items dieser Work-Group haben auf diesen Zugriff und können somit über diesen Speicher Daten austauschen und sich in der Ausführung synchronisieren.

- **Privater Speicher (englisch „private memory“)**

  Hierbei handelt es sich um den privaten Speicher eines Work-Items, bzw. einer Kernelinstanz. Andere Work-Items oder das Leitprogramm können auf den Inhalt dieses Speichers nicht zugreifen

Diese Speicherstruktur hat zur Folge, dass Daten, welche auf dem Device zur Verfügung gestellt werden sollen, vom Entwickler dieser Anwendung explizit vom Host aus dorthin kopiert werden müssen. Hierzu ist zu beachten, dass das Host-Programm ausschließlich Zugriff auf die Daten im globalen und konstanten Speicher hat und, dass der Host auf den lokalen oder privaten Speicher absolut keine Lese- bzw. Schreib-Rechte hat. Will man also im Kernel Daten verarbeiten, muss der Host diese zuvor in den globalen, bzw. konstanten Speicher kopieren, bevor der Kernel Zugriff auf diese hat, um sie entweder gleich an Ort und Stelle zu verarbeiten oder sie in den eigenen, privaten oder lokalen, Speicher kopieren kann.

5 Aufbau einer OpenCL Anwendung

Auf Grund der zuvor beschriebenen logischen Trennung des Speichers zwischen Host und Device, müssen, für den Ablauf des Programmes notwenigen, Daten zwischen deren Speichern ausdrücklich ausgetauscht werden. Hierdurch ergibt sich zwangsläufig der folgende, typische Aufbau einer OpenCL-Anwendung:

```
int main (void) {

   1. Initialisierung von OpenCL

   2. Kernel-Quellcode kompilieren

   3. Reservieren des globalen und konstanten Speichers auf dem
      Device und Daten zum Device kopieren.

   4. Kernelinstanzen ausführen

   5. Daten zurück auf den Host kopieren

}
```

5.1 OpenCL C

Eine solche OpenCL-Anwendung wird, wie zuvor schon erwähnt, in der eigenen Programmiersprache OpenCL C verfasst, welche im Grunde auf der Programmiersprache C basiert. Diese hat jedoch, sowohl einige Einschränkungen als auch zusätzliche Konstrukts.

Zu den signifikantesten Einschränkungen von OpenCL C im Vergleich zu einem „normalen“ C Programm gehört, dass Fehlen von Standardfunktionen, wie zum Beispiel printf() oder malloc(). Des Weiteren sind dynamische Arrays genauso wenig möglich, wie Funktionszeiger, da auf einigen OpenCL-Devices, wie z.B. den GPUs kein vollständiges Betriebssystem, mit den entsprechenden Funktionen, wie z.B. einer dynamischen Speicherverwaltung, zur Verfügung stehen. Darüber hinaus, sind ebenfalls keine Rekursionen erlaubt, da um eine Solche zu ermöglichen, eine Rücksprungadresse der Funktion gespeichert werden müsste, diese jedoch nicht von allen OpenCL-Devices unterstützt wird.

Neben den Einschränkungen gibt es in OpenCL C jedoch auch noch zusätzliche Programmkonstrukts, welche die Programmierer beachten. Hierzu gehören unter anderem so genannte Qualifier[15], Schlüsselworte, die man vor eine Funktions-Deklaration zusätzlich angibt. Funktionen, welche auf einem Device ausgeführt werden und von einem Host aufgerufen werden, also die so genannten Kernel, werden mit dem Qualifier __kernel deklariert. Diese unterliegen des Weiteren der Bedingung, dass sie den Rückgabewert void haben müssen. Ein Beispiel hierfür wäre:

```
__kernel void foo (int bar, int baz);
```

Neben Funktionen, können in OpenCL C auch Variablen mit einem von vier Qualifier markiert werden

- __global
- __constant
- __local
- __private

[15] Qualifier: hierbei handelt es sich um eine Zusatzinformation (Meta-Data)

Diese entsprechen den vier Speicherbereichen aus dem OpenCL Speicher-Modell: globaler, konstanter, lokaler und privater Speicher. Sollte der Qualifier nicht explizit angegeben sein, so wird der private Speicherbereich vorausgesetzt. Beispiel:

```
// Variable i wird im privaten Speicher abgelegt
int i;
// Array mit 4 float Variablen im lokalen Speicher
__local float x[4];
// Zeiger p zeigt auf eine Variable im globalen Speicher
__global int *p;
```

Des Weiteren stellt OpenCL C, ähnlich wie, die Standard C Bibliothek, eine Reihe von eingebauten Funktionen bereit, welche die Folgenden umfassen:

- Mathematische Funktionen (sin(), cos(), ...)
- Funktionen zur Synchronisation
- Atomare (d.h. unteilbare) Funktionen (diese werden jedoch nicht von allen OpenCL-Devices unterstützt
- Funktionen zum Zugriff auf die globale und lokale ID eines Work-Items

Um den Unterschied zwischen einer Funktion, geschrieben in traditionellen Programmiersprachen und dem in der Multicore-unterstützenden Sprache OpenCL C zu verdeutlichen, folgt hier der Programm-Code für eine Berechnung des Skalar-Produktes zweier Vektoren, in beiden Methoden:

Standard ISO C99

```
// Skalarprodukt
void dot_product (const float *a,
                  const float *b,
                  const int n,
                  float *c)
{
      for (int i = 0; i < n; ++i)
            c[i] = a[i] * b[i];
}
```

OpenCL C

```
// Skalarprodukt
__kernel void dot_product (   __global const float4 *a,
                              __global const float4 *b,
                              __global float *c)
{
      int gid = get_global_id(0);
      c[gid] = a[gid] * b[gid];
}
```

6 Performance & Fazit

OpenCL ist ein Plattform-übergreifendes Programmiermodell zur Entwicklung von Multicore-Anwendungen für verschiedene parallele Prozessor Architekturen. Auf Grund dessen, dass bei der OpenCL CPU Implementierung für jeden logischen Kern ein eigener Thread zur Berechnung des Kernels gestartet wird und die Work-Items automatisch von OpenCL an die Threads verteilt werden, ist eine massive Performance-Steigerung im Vergleich zu der traditionellen sequentiellen Abarbeitung der Berechnungen zu erwarten. Dabei ist die OpenCL Implementierung auf GPU-Basis noch um ca. 1,6 Mal schneller, als die parallele Implementierung auf der CPU.

Da zu erwarten ist, dass auch in der Zukunft, die Anzahl der Kerne in den CPUs und GPUs der Computer weiter ansteigen wird und die Taktrate sich im Vergleich dazu eher konstant halten wird, ist die Multicore-Nutzung für aktuelle Programme von großer Bedeutung. Des Weiteren sind die Grafikprozessoren, in den meisten Systemen, bei der Nutzung von nicht-grafisch-lastigen Anwendungen kaum Ausgelastet und bieten somit eine weitere und vor allem, in der parallelen Nutzung, sehr leistungsstarke Ressource, auf die zurückgegriffen werden sollte.

Aus diesen Gründen ist OpenCL ein Programmiermodell, welches sich, mit großer Wahrscheinlichkeit, in der Zukunft, bei vielen rechenintensiven Anwendungen durchsetzen wird.

Literaturverzeichnis

[GO11] Gorlatch, Sergei & Steuwer, Michel, OpenCL – ein Programmieransatz für GPU und CPU, Westfälische Wilhelms-Universität Münster, 2011

[AP10] Apelt, Nicolas / Zöllner Christian, OpenCL, Multi-Core Architectures and Programming (Seminar), Hardware-Software-Co-Design, Universität Erlangen-Nürnberg, 2010

[UB11] Universität Basel, Vorlesung OpenCL Grundlagen, Basel 2011

[NV12] http://www.nvidia.de/object/cuda_opencl_new_de.html

[WI12] http://en.wikipedia.org/wiki/OpenCL

[ZD11] http://www.zdnet.de/news/41558030/spezifikation-von-opencl-1-2-veroeffentlicht.htm

[KH11] http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf