Inhalt

1. Deckblatt
2. Eidesstaatliche Erklärung
3. Vorwort
4. Kurzfassung
5. Einleitung
   * Zielsetzung
     + Was zeigt dieses Dokument?
       - GPGPU Beschleunigung mit OpenCL
       - Beispielimplementierungen von Scan, Sort und Matrix Mul.
     + Was soll bei meiner Arbeit rauskommen?
       - Welche Algorithmen profitieren von GPU Beschleunigung
         * In konkreten Fällen?
         * Immer?
       - Mehraufwand einer GPU Implementierung im Vergleich mit CPU?
     + Wie komme ich zu diesem Ziel?
       - Algorithmen ausgewählt
       - Bestehende Implementierungen eingebunden
       - Teilweise Algorithmen selbst implementiert
       - Referenzimplementierungen für die CPU erstellt
       - Benchmarks auf verschiedener Hardware
   * Motivation
     + Warum ist dieses Ziel wichtig? (für alle?)
     + Was ließe sich dadurch verbessern?
       - Um wie viel ließe es sich verbessern?
     + Wem würde das helfen?
       - Speziellen Unternehmen
       - Spezielle Domänen/Anwendungsgebiete?
       - Allen?
   * Kapitelübersicht
6. Einführung in OpenCL
   * Historischer Rückblick
     + Entwicklung von Grafikhardware
     + Shader Programmierung (als erste Möglichkeit eingeschränkt für GPU's zu programmieren, Focus auf Grafikberechnungen)
     + GPGPU Ansätze (Missbrauch von Shadern für allgemeine Probleme, vgl. Sort mit GLSL Vertex Program)
     + Technologien (Cg, CUDA, OpenCL, DirectCompute)
   * Was ist OpenCL?
     + Offener Industriestandard (keine Implementierung/Bibliothek)
     + Khronos (spezifiziert API)
     + Vendors (implementieren API)
     + Komponenten
       - API Spezifikation
       - OpenCL C Language
   * Anforderungen
     + GPU (and CPU) Hardware
     + Treiber
     + SDK (Header, Bibliotheken (C/C++))
     + Bindings für andere Programmiersprachen
   * Hardware Architekturen
     + Grafikkarten-/Prozessorarchitektur
     + Unterschiede/Gemeinsamkeiten
   * API
     + Aufbau eines OpenCL Programs
     + Abstraktion der Hardware in OpenCL
       - Platforms, Devices, CommandQueues, Buffer, Programs, Kernels
     + Minimalistisches Codebeispiel (ohne Kernel)
   * Kernel execution
     + Was ist ein Kernel?
     + Wie werden Kernel ausgeführt?
       - Work groups/items
         * Hardware threads/Synchronisierung
       - Wahl der richtigen work group size
         * Auswirkungen auf Performance
         * Konsequenzen für die Programmierung
   * Memory model
     + Speicherbereiche auf der GPU
       - Global, local, constant, private
       - Performance
       - Wie werden diese angesprochen?
       - Bedeutung für die Kernel Programmierung
       - Generelle Designentscheidungen
         * Paralleler Speicherzugriff
         * Shared memory zum Cachen/Synchronisieren
         * Bench conflicts
     + Datentransfer
       - Wie werden Daten zur GPU befördert?
   * Begriff von Hardwareherstellern (AMD, NVIDIA)
7. State of the Art
   * Welche Programme/Produkte/Unternehmen setzen OpenCL ein?
   * Wie weit ist OpenCL verbreitet?
   * Anwendungsgebiete
     + Von OpenCL
     + Von GPGPU Beschleunigung
     + WebCL
8. Implementierung
   * Matrix multiplication
     + Problembeschreibung (Multiplizieren von zwei quadratischen Matrizen)
     + CPU Implementierung
       - Laufzeitverhalten (Diagramm)
     + Naive GPU Implementierung
       - Ansatz (Aufteilung der Elemente auf Work items)
       - Kernel code/host code, mit Erklärung
       - Laufzeitverhalten (Diagramm)
     + (Optimierte GPU Implementierung)
       - Möglichkeiten zur Verbesserung
         * Mehrere Elemente per work item
         * Caching in local memory
         * Texture unit
         * Vector instructions
       - Evt. Kernel code/host code, mit Erklärung
       - Auswirkungen
         * Mehraufwand beim Entwickeln
         * Auswirkungen auf verschiedener Hardware
       - Laufzeitverhalten (Diagramm) und Vergleich mit naiver Implementierung
         * OpenCL Implementierung auf CPU
     + Vergleich mit bestehenden GPGPU und CPU Implementierungen
       - clAmdBlas
       - cuBlas
       - Intel Math Kernel Library (MKL)
       - AMD Core Math Library (ACML)
       - (NVIDIA OpenCL Samples)
       - Gemeinsames Diagramm
     + Conclusion
       - Was läuft wann besser?
       - Ab welcher Problemgröße?
       - Auf welcher Hardware?
   * Parallel prefix sum (scan)
     + Problembeschreibung (lineares Problem, schlecht parallelisierbar)
     + CPU Implementierung
       - Laufzeitverhalten (Diagramm)
     + Naive GPU Implementierung (kurz)
       - Extrem imperformant (Warum?)
     + Welche Möglichkeiten lineares Problem zu parallelisieren? (Optimierte GPU Implementierungen)
       - Tree based (GPU Gems)
         * Work efficiency?
       - Kernel code/host code, mit Erklärung
       - Laufzeitverhalten (Diagramm)
         * Genauere Betrachtung (Upload, Scan, Download)

Übertragungsdauer vs. Berechnung

* + - Vergleich mit bestehenden GPGPU und CPU Implementierungen
      * Apple
      * libCL
      * (AMD APP SDK Samples)
      * NVIDIA OpenCL Samples
      * Gemeinsames Diagramm
    - Conclusion
      * Warum lineare Probleme trotzdem auf einer GPU notwendig sind
        + Building blocks für andere Algorithmen (z.B. RadixSort)
      * Zukünftige Hardwareentwicklungen
        + GPU und CPU am selben Chip (vgl. Sandy Bridge)

Übertragung in GPU Speicher entfällt

* + Sorting
    - Problembeschreibung
    - CPU Implementierung (nur Aufzählung)
      * Laufzeitverhalten (Diagramm)
    - Ansätze für GPU Implementierungen
      * Sorting networks (Bitonic)
      * RadixSort
    - Implementierungen
      * Bitonic Sort (Bealto)
        + Schrittweise Verbesserungsmöglichkeiten
        + Jeweils mit Kernel code/host code, und Erklärung
        + Laufzeitverhalten
        + n \* n log²(n), warum aber trotzdem schneller?
      * Radix sort (integer only)
        + Wirkt leicht parallelisierbar, Probleme?

Abarbeitungsreihenfolge von work items

* + - * + Kernel code/host code, mit Erklärung
        + Laufzeitverhalten (Diagramm)
    - Vergleich mit bestehenden GPGPU und CPU Implementierungen
      * libCL
      * (AMD APP SDK Samples)
      * (NVIDIA OpenCL Samples)
      * clpp (OpenCL Data Parallel Primitives Library)

1. Weiterführende Benchmarks
   * Vergleiche auf verschiedener Hardware
   * Vergleiche auf verschiedenen Betriebssystemen
2. Conclusion
   * Wo kann OpenCL sinnvoll eingesetzt werden?
     + Wo ist der Mehraufwand einer GPU Implementierung leistbar?
   * Zukünftige technologische Entwicklungen/Trends