Inhalt (65.5)

1. Deckblatt
2. Eidesstaatliche Erklärung
3. Vorwort
4. Kurzfassung
5. Einleitung (3)
   * Zielsetzung (2)
     + Was zeigt dieses Dokument? (1)
       - GPGPU Beschleunigung mit OpenCL
       - Beispielimplementierungen von Scan, Sort und Matrix Mul.
     + Was soll bei meiner Arbeit rauskommen? (0.5)
       - Welche Algorithmen profitieren von GPU Beschleunigung
         * In konkreten Fällen?
         * Immer?
       - Mehraufwand einer GPU Implementierung im Vergleich mit CPU?
     + Wie komme ich zu diesem Ziel? (0.5)
       - Algorithmen ausgewählt
       - Bestehende Implementierungen eingebunden
       - Teilweise Algorithmen selbst implementiert
       - Referenzimplementierungen für die CPU erstellt
       - Benchmarks auf verschiedener Hardware
   * Motivation (1)
     + 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 (14.5)
   * Historischer Rückblick (2)
     + 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? (1.5)
     + Offener Industriestandard (keine Implementierung/Bibliothek)
     + Khronos (spezifiziert API)
     + Vendors (implementieren API)
     + Komponenten
       - API Spezifikation
       - OpenCL C Language
   * Anforderungen (1)
     + GPU (and CPU) Hardware
     + Treiber
     + SDK (Header, Bibliotheken (C/C++))
     + Bindings für andere Programmiersprachen
   * Hardware Architekturen (2.5)
     + Grafikkarten-/Prozessorarchitektur (2)
     + Unterschiede/Gemeinsamkeiten (0.5)
   * API (2.5)
     + Aufbau eines OpenCL Programs (1)
     + Abstraktion der Hardware in OpenCL (1.5)
       - Platforms, Devices, CommandQueues, Programs, Kernels, Buffers
     + Minimalistisches Codebeispiel (ohne Kernel und ohne Buffers) (1)
   * Kernel execution (2.5)
     + Was ist ein Kernel? (0.5)
     + Wie werden Kernel ausgeführt? (2)
       - Work groups/items (0.75)
       - Hardware threads/Synchronisierung (0.75)
       - Wahl der richtigen work group size (0.5)
         * Auswirkungen auf Performance
         * Konsequenzen für die Programmierung
   * Memory model (2.5)
     + Speicherbereiche auf der GPU (1.5)
       - Global, local, constant, private
       - Performance
       - Wie werden diese angesprochen?
       - Bedeutung für die Kernel Programmierung
     + Generelle Designentscheidungen (0.5)
       - Paralleler Speicherzugriff
       - Shared memory zum Cachen/Synchronisieren
       - Bench conflicts
     + Datentransfer (0.5)
       - Wie werden Daten zur GPU befördert?
   * Begriff von Hardwareherstellern (AMD, NVIDIA) (1, Anhang)
7. State of the Art (1.5)
   * (Welche Programme/Produkte/Unternehmen setzen OpenCL ein?)
   * (Wie weit ist OpenCL verbreitet?)
   * Anwendungsgebiete (1)
     + Von OpenCL
     + Von GPGPU Beschleunigung
     + WebCL (0.5)
8. Implementierung (41.5)
   * Matrix multiplication (12)
     + Problembeschreibung (Multiplizieren von zwei quadratischen Matrizen) (1)
     + CPU Implementierung (1)
       - Laufzeitverhalten (Diagramm)
     + Naive GPU Implementierung (3)
       - Ansatz (Aufteilung der Elemente auf Work items) (0.5)
       - Kernel code/host code, mit Erklärung (2)
       - Laufzeitverhalten (Diagramm) (0.5)
     + Optimierte GPU Implementierung (5)
       - Möglichkeiten zur Verbesserung (1)
         * Mehrere Elemente per work item
         * Caching in local memory
         * Texture unit
         * Vector instructions
       - Evt. Kernel code/host code (Ausschnitte), mit Erklärung (3)
       - (Auswirkungen)
         * Mehraufwand beim Entwickeln
         * Auswirkungen auf verschiedener Hardware
       - Laufzeitverhalten (Diagramm) und Vergleich mit naiver/CPU Implementierung (1)
         * OpenCL Implementierung auf CPU
     + Vergleich mit bestehenden GPGPU und CPU Implementierungen (1)
       - clAmdBlas
       - cuBlas
       - Intel Math Kernel Library (MKL)
       - AMD Core Math Library (ACML)
       - (NVIDIA OpenCL Samples)
       - Gemeinsames Diagramm
     + Conclusion (1)
       - Was läuft wann besser?
       - Ab welcher Problemgröße?
       - Auf welcher Hardware?
   * Parallel prefix sum (scan) (10)
     + Problembeschreibung (lineares Problem, schlecht parallelisierbar) (1)
     + CPU Implementierung (1)
       - Laufzeitverhalten (Diagramm)
     + Naive GPU Implementierung (kurz) (2)
       - Extrem imperformant (Warum?)
     + Welche Möglichkeiten lineares Problem zu parallelisieren? (Optimierte GPU Implementierungen) (4)
       - Tree based (GPU Gems) (1)
         * Work efficiency?
       - Evt. Kernel code/host code, mit Erklärung (2)
       - Laufzeitverhalten (Diagramm) (1)
         * Genauere Betrachtung (Upload, Scan, Download)

Übertragungsdauer vs. Berechnung

* + - Vergleich mit bestehenden GPGPU und CPU Implementierungen (1)
      * Apple
      * libCL
      * (AMD APP SDK Samples)
      * NVIDIA OpenCL Samples
      * Gemeinsames Diagramm
    - Conclusion (1)
      * Warum lineare Probleme trotzdem auf einer GPU notwendig sind
        + Building blocks für andere Algorithmen (z.B. RadixSort)
  + Sorting (19.5)
    - Problembeschreibung (1)
    - CPU Implementierung (nur Aufzählung) (1)
      * Laufzeitverhalten (Diagramm)
    - Ansätze für GPU Implementierungen (17.5)
      * Bitonic Sort (Bealto) (5)
        + Ansatz (1)
        + Kernel code/host code, und Erklärung (2)
        + Schrittweise Verbesserungsmöglichkeiten (2)
        + Laufzeitverhalten (Diagramm) (1)

n \* n log²(n), warum aber trotzdem schneller?

* + - * Radix sort (integer only) (6.5)
        + Ansatz (1)
        + Wirkt leicht parallelisierbar, Probleme? (1)

Abarbeitungsreihenfolge von work items

* + - * + Kernel code/host code, mit Erklärung (4)

(AMD Variante)

(NVIDIA Variante)

* + - * + Laufzeitverhalten (Diagramm) (0.5)
      * (MergeSort) (6)
    - Vergleich mit bestehenden GPGPU und CPU Implementierungen (1)
      * libCL
      * (AMD APP SDK Samples)
      * (NVIDIA OpenCL Samples)
      * clpp (OpenCL Data Parallel Primitives Library)

1. Weiterführende Benchmarks (3)
   * Vergleiche auf verschiedener Hardware
   * Vergleiche auf verschiedenen Betriebssystemen
2. Conclusion (2)
   * Wo kann OpenCL sinnvoll eingesetzt werden? (1)
     + Wo ist der Mehraufwand einer GPU Implementierung leistbar?
   * Zukünftige technologische Entwicklungen/Trends (1)
     + GPU und CPU am selben Chip (vgl. Sandy Bridge)
       - Übertragung in GPU Speicher entfällt