Parallele Algorithmen mit OpenCL

Universität Osnabrück, Henning Wenke, 2013-07-03

Warp / Wavefront

- Feste Anzahl konsekutiver Threads einer GPU werden SIMD-artig ausgeführt
- Nvidia (derzeit): 32
- Nvidia nennt diese Gruppen „Warp“
- AMD Bezeichnung: „Wavefront“
- Streaming Multiprocessor einer Nvidia Kepler:
  - Max gleichzeitig existierende Warps: 64 ⇒ 2048 Threads
  - 4 Warp Scheduler
  - Jeder kann jeweils zwei unabhängige Anweisungen an zugeordnete Warps erteilen
- Half-Warp (Nvidia Bezeichnung)
  - Die ersten 16 oder…
  - … die hinteren 16 Threads eines Warps

Caches

- Einige Daten bleiben nach Verwendung in Caches
- Falls benötigte Daten noch im Cache, wird Zugriff auf Global Memory unnötig
- Gibt‘s bei Nvidia ab Fermi-Architektur (März 2010)
- Nvidia Kepler
  - L1: Max 48 KB je Streaming Multiprocessor, sehr schnell
  - L2: 512 KB Global, schnell
- Gerade bei unvorhersehbaren (datenabhängigen) Zugriffsmustern hilfreich
- Können Einsatz von Local Memory bei geringen Datenmengen überflüssig machen: Z.B. Nbody-System

Abschnitt

Speicherzugriffsmuster

Global Memory

- Für Global Memory DRAM verwendet
  - Hohe Latenz und recht geringe Bandbreite
  - Konsekutive Adressen, die Angefragte beinhaltend, werden parallel ausgelesen
- Szenario: Alle Threads eines Warps laden in einer Anweisung konsekutive Adressen
  - Wird erkannt und zu einem Zugriff kombiniert
  - „Coalesced Access“
  - Peak Performance (nur) so erreichbar

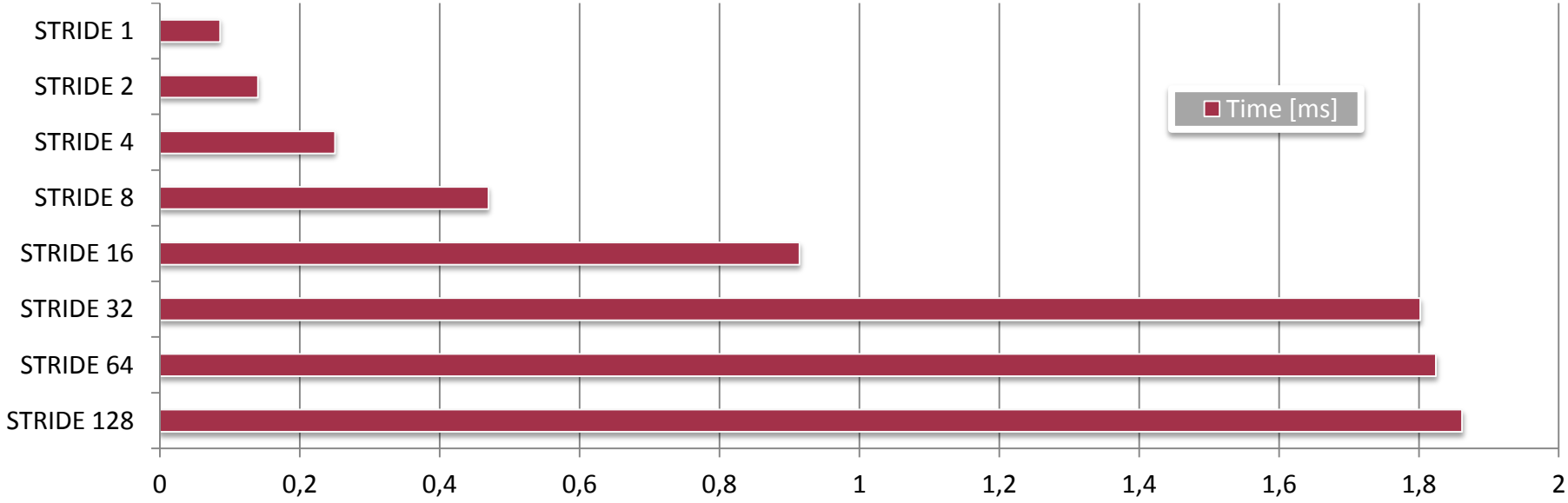
Beispiel: Strided VecAdd

- Addiere 2 n-komponentige float Vektoren, deren Werte je STRIDE Positionen im Speicher auseinander liegen
- Daten:
  - CLMem a & b: $n \cdot STRIDE \cdot 4$ Byte groß
  - CLMem c: $n \cdot 4$ Byte groß
- Miss Zeit für verschiedene STRIDE Werte: 1, 2, 4, 8, 16, 32, 64, 128

```
#define SRTRIDE 32                    // oder: 1, 2, 4, 8, 16, 64, 128
kernel void stridedVecAdd(            // Erzeuge n work-items
 global float* a, global float* b, global float* c){
    int id = get_global_id(0);
    c[id] = a[STRIDE * id] + b[STRIDE * id];
}
```

Auswertung

- Diagramm: Ergebnisse für GTX 670, n = 1048576 Work-Items
- Beobachtungen:
  - Bis STRIDE = 32 jeweils starke Verschlechterung der Laufzeit im Vergleich mit nächst geringerem STRIDE-Wert
  - Für STRIDE 32 wird 21-fache Laufzeit von STRIDE 1 benötigt
  - Danach geringe Änderung
- Hinweis: Verhalten auf CPU ähnlich



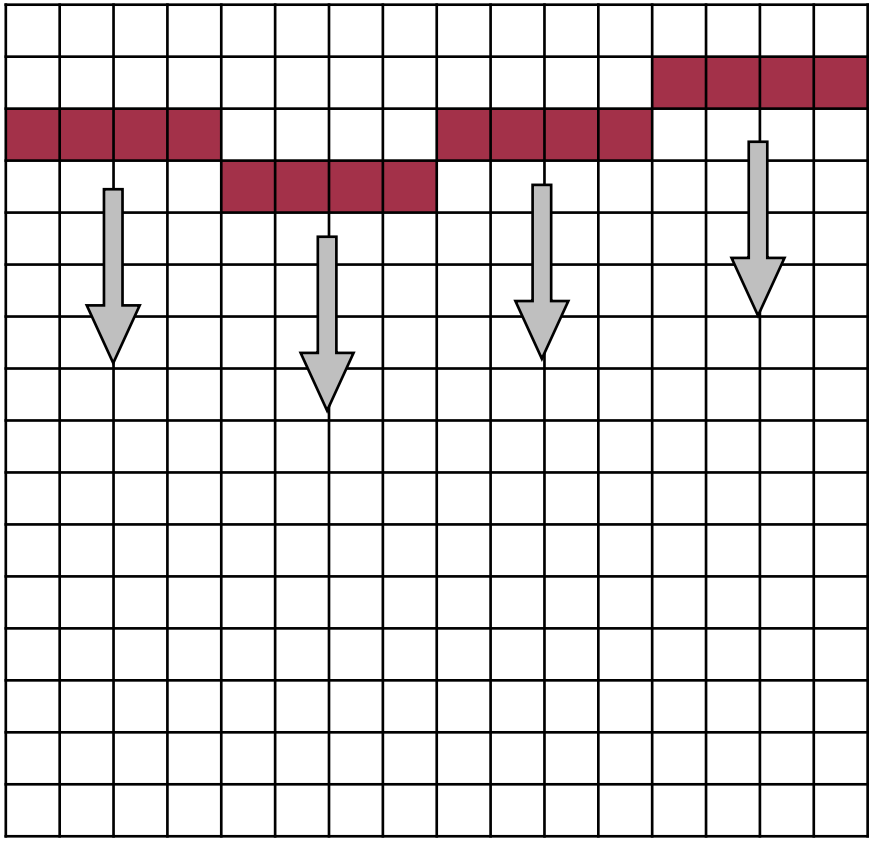
Beispiel: Row/Col-Wise Sequential OPs

- Gegeben: 2D-Raster aus floats mit Kantenlänge: MAT_SIZE
- Aufgaben:
  - A: Berechne je Zeile parallel & innerhalb der Zeilen sequentiell „etwas“ aus
  - B: Berechne je Spalte parallel & innerhalb der Spalte sequentiell „etwas“ aus
- „Etwas“ sei hier einfach Summe der Elemente

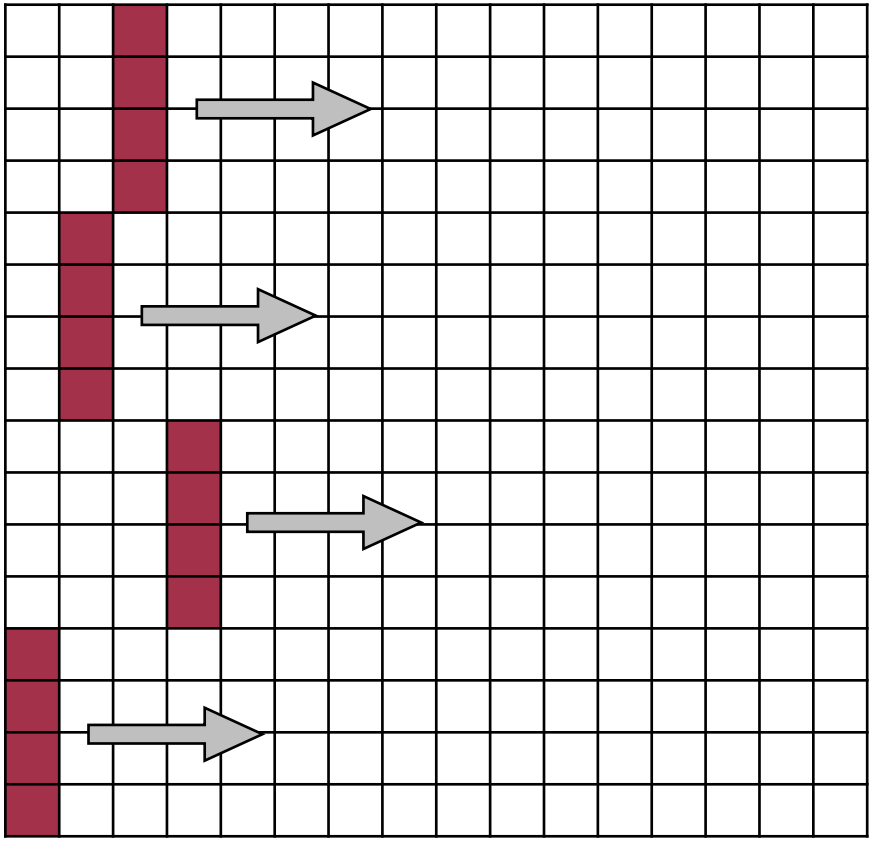
```
#define MAT_SIZE 16384
kernel void matSumRow(global float* A, global float* B){   // A
   int y = get_global_id(0);
   float result = 0;
   for(int x = 0; x < MAT_SIZE; x++)
      result += A[x + y * MAT_SIZE];
   B[y] = result;
}

kernel void matSumCol(global float* A, global float* B){   // B
   int x = get_global_id(0);
   float result = 0;
   for(int y = 0; y < MAT_SIZE; y++)
      result += A[x + y * MAT_SIZE];
   B[x] = result;
}
```

Skizze zu einem Zeitpunkt (Nvidia)

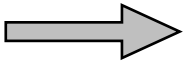
`matSumCol`



`matSumRow`



Durch einen Warp geladene Elemente (hier nur 4 statt 32 dargestellt)



Arbeitsrichtung über verschiedene Zeitschritte

Auswertung

- Sei MAT_SIZE 16384
  - 16384 Work-Items
  - Jedes Work-Item führt 16384 sequentielle Schritte aus
- Ergebnisse für GPU (Nvidia GTX 670)
  - `matSumRow`: 328 ms
  - `matSumCol`: 9,6 ms
  - Faktor: 34
- Ergebnisse für CPU (Intel Core i7 2700k)
  - `matSumRow`: 124 ms
  - `matSumCol`: 2006 ms
  - Faktor: 0,06

Sonstiges

- Zugriffe auf Constant Memory (Cuda)
  - Zugriffe eines Half-Warps auf gleiche Adresse zusammengefasst
  - Kann sehr gut gecached werden
- Zugriffe auf Local Memory
  - Ungünstige Zugriffsmuster können Zugriffe serialisieren
  - Stichwort: „Bank Conflict“
- Asynchrones Kopieren zwischen Local- & Global Memory
  - `event_t async_work_group_copy(…)`
  - `event_t async_work_group_strided_copy(…)`
  - `wait_group_events(event_t ev)`
- Verwendung zu vieler Ressourcen (z.B. Register) je Work-Item / Work-Group kann mögliche Parallelität einschränken

Literatur zu Optimierung

- CUDA C Programming Guide, Version 4.2
- Programming Massively Parallel Processors: A Hands-on Approach, Second Edition
- NVIDIA GeForce GTX 680 (Whitepaper)
- … galt auch schon letzte Woche

Praxisteil

Projekte, Themen, Organisatorisches

Organisatorisches

- Gruppen aus mindestens zwei Personen
- Maximalgruppengröße themenabhängig
- Termine in zwei Blöcken:
  - Gruppe A: 19. August – 6. September
  - Gruppe B: 23. September – 11. Oktober
- Abschlusspräsentation
  - Am letzten Tag des jeweiligen Blocks
  - Teilnahme an eigenem Präsentationstermin genügt
  - Dauer: Möglichst genau 5 Minuten pro Person
- Terminprobleme bitte mit uns besprechen
- Doku: Gibt‘s auch

Projekte / Themen

- Finale Vergabe: Mittwoch, 10.7
- Vorschläge sammeln wir schon vorher
- Möglichst mit Bezug zur Studienrichtung
- Zunächst Thema festlegen, Aufgabenstellung finden wir dann…
- Beispiel, Algorithmus XYZ
  - Parallel formulieren & implementieren
  - Optimieren & Performance evaluieren
  - Ergebnisse vergleichen mit Angaben der Literatur
- Vorschläge noch bis Montag möglich
- Jede(r) sollte wenigstens eine Präferenz für ein Themengebiet nennen, z.B: „Computergrafik“, „Scientific Simulation“
- Manche Themen können mehrfach vergeben werden

Sammlung bisheriger Vorschläge

- Remote Rendering
- Simulation einer Schwarmintelligenz
- Real-Time Curve / Nurbs Tesselation (2x)
- Visualisierung eines Räuber-Beute Systems
- Neuronale Netze
- Niederschlag / Wasser Simulation
- „Knoten lösen“

Weitere Ideen

- Siehe Veranstaltung erste Woche
- Literatur: Paper oder GPU Computing Gems (Jade & Emerald Edition):
  - http://proquestcombo.safaribooksonline.com/book/electrical-engineering/computer-engineering/9780123859631
  - http://proquestcombo.safaribooksonline.com/book/-/9780123849885
- Reine Programmieraufgaben auch ok
- Beliebige Algorithmen & Untersuchung mit Fokus auf:
  - Distributed Memory
  - CPU / AMD Optimierung
  - Heterogenous Systems
- Dynamic Parallelism auf einer Nvidia Geforce GTX Titan