GPU-beschleunigtes Sortieren

Am Beispiel QuickSort auf einer Nvidia Grafikkarte

Von Michael Schmidt und Kevin Büchele

Gliederung

1. Erklärung QuickSort
2. CUDA Hardwaremodell
3. Erklärung des Algorithmus`
4. Beispiel-Diagramme

1. QuickSort

- Finden eines Pivotelements
- Zu sortierendes Array mit Pivot als „Seperator“ spalten
- Rekursiv wird links vom Pivot und rechts vom Pivot nach selben Prinzip sortiert
- Am Ende werden alle Teile zusammengefügt (trivialerweise ist danach kein Sortieren mehr notwendig)

2. CUDA Hardware

Definition:

- parallele Berechnungsarchitektur
- deutliche Steigerung der Rechenleistung

In anderen Worten:

- bietet Schnittstelle für Programmierern, um die Rechenleistung der GPUs mit einzubeziehen
- insgesamt eine signifikante Steigerung der Rechenleistung zuerreichen.

Weitere Eigenschaften (Nachteile)

- Meistens in C programmiert
- Eher untypische Datentypen (9 Bit, 12 Bit)

  → Häufige Emulierung durch Software notwendig
- Höhere Latenzzeiten durch Anbindung mittels PCI(e)

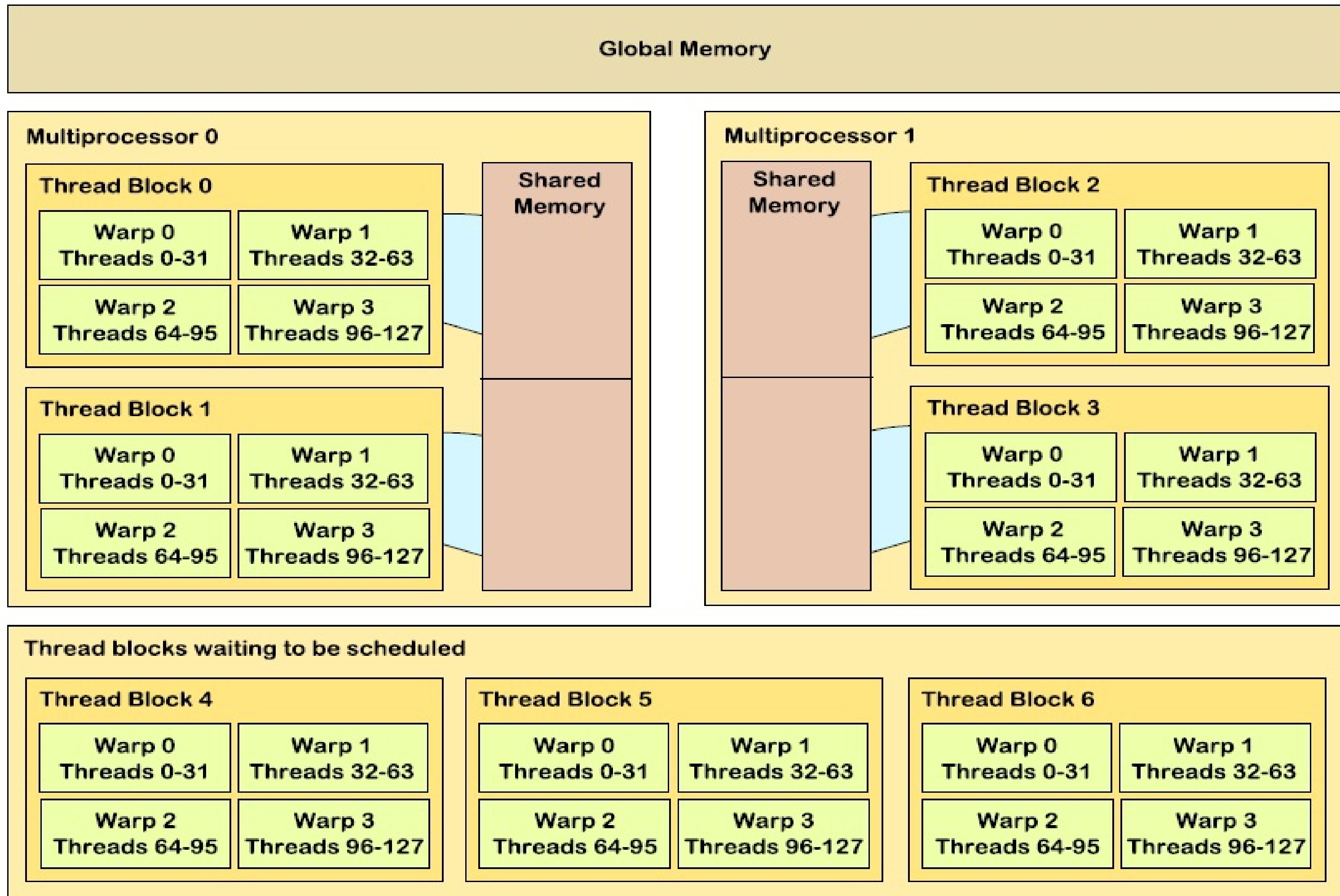
Technische Eigenschaften der unterstützenden Grafikkarten

- 16 Multiprozessoren mit jeweils 8 Prozessoren
- Jeder Multiprozessor unterstützt bis zu 768 Threads
- 16KiB (= 2048 Byte) lokaler Speicher vorhanden mit maximal 8192 verfügbaren Registern

Scheduling

- Threads zusammengefasst in Threadblocks
- Abhängig von Anzahl der Register, benötigter Speicherplatz, können einzelne Threadblocks zum gleichen Multiprozessor zugeordnet werden
- Warps fassen sequenziell/gleichzeitig arbeitende Threads zusammen
- Es kann immer nur ein Warp pro Multiprozessor ausgeführt werden

Grafische Darstellung der Hardware



Leistung von CUDA

ISV	BESCHREIBUNG	VORTEILE DES GRAFIKPROZESSORS
MathWorks MATLAB®	Datenparallele Berechnungen (MATLAB PCT, MDCS)	Deutliche Beschleunigung für mehr Produktivität für Studenten, Wissenschaftler und Techniker
Jedox Palo	Excel-Erweiterung mit OLAP für Planung und Analyse	20- bis 40-fache Beschleunigung: Entscheidungsfindung innerhalb von Stunden statt Tagen
ParStream	Beschleunigung von Datenbanken und Datenanalyse mit Grafikprozessoren	10-fache oder noch größere Beschleunigung: Milliarden von Datensätzen werden in Sekunden durchsucht
Tanay Data Analytics von Fuzzy Logix	Datenbank-integrierte Analytik-Engine	Beschleunigung von Finanzsimulationen, Data-Mining und statistischen Verfahren mit Grafikprozessoren
GPU-Quicksort	Hoch optimierter Quicksort-Algorithmus	10-fache Beschleunigung: Über 16 Millionen Gleitkommazahlen werden in weniger als 0,5 s sortiert
AccelerEyes Jacket für MATLAB	Nahtlose Beschleunigung von MATLAB mit Jacket™	Bis zur 100-fachen Beschleunigung je nach Anwendung
Wolfram Mathematica	Analysen in symbolischer Mathematik (Mathematica)	Deutliche Leistungssteigerungen bei unterschiedlichen Anwendungen, unter anderem für lineare Algebra, Bildbearbeitung, Finanzsimulationen und Fourier-Transformationen

3. Der Algorithmus beim Sortieren

Phase 1

- Eingegebene Zahlen werden in gleich so unterteilt, sodass es gleich viele Blöcke wie Treads gibt

  → ermöglicht später synchronisiertes Arbeiten

  → Partitionierung

Algorithm 2 GPU-Quicksort (First Phase GPU Kernel)

function GQSORT($blocks, d, \hat{d}$)
var global $sstart, send, oldstart, oldend, blockcount$
var block local $lt, gt, pivot, start, end, s$
var thread local $i, lfrom, gfrom, lpivot, gpivot$

$(s_{start \to end}, pivot, parent) \leftarrow blocks_{blockid}$ ▷ Get the sequence block assigned to this thread block.
$lt_{threadid}, gt_{threadid} \leftarrow 0, 0$ ▷ Set thread local counters to zero.

$i \leftarrow start + threadid$ ▷ Align thread accesses for coalesced reads.
for $i < end, i \leftarrow i + threadcount$ **do** ▷ Go through the data...
  if $s_i < pivot$ **then** ▷ counting elements that are smaller...
    $lt_{threadid} \leftarrow lt_{threadid} + 1$
  if $s_i > pivot$ **then** ▷ or larger compared to the pivot.
    $gt_{threadid} \leftarrow gt_{threadid} + 1$

$lt_0, lt_1, lt_2, \ldots, lt_{sum} \leftarrow 0, lt_0, lt_0 + lt_1, \ldots, \sum_{i=0}^{threadcount} lt_i$ ▷ Calculate the cumulative sum.
$gt_0, gt_1, gt_2, \ldots, gt_{sum} \leftarrow 0, gt_0, gt_0 + gt_1, \ldots, \sum_{i=0}^{threadcount} gt_i$

if $threadid = 0$ **then** ▷ Allocate memory in the sequence this block is a part of.
  $(seq_{sstart \to send}, oseq_{oldstart \to oldend}, blockcount) \leftarrow parent$ ▷ Get shared variables.
  $lbeg \leftarrow FAA(sstart, lt_{sum})$ ▷ Atomic increment allocates memory to write to.
  $gbeg \leftarrow FAA(send, -gt_{sum}) - gt_{sum}$ ▷ Atomic is necessary since multiple blocks access this variable.

$lfrom = lbeg + lt_{threadid}$
$gfrom = gbeg + gt_{threadid}$

$i \leftarrow start + threadid$
for $i < end, i \leftarrow i + threadcount$ **do** ▷ Go through data again writing elements
  if $s_i < pivot$ **then** ▷ to the their correct position.
    $\neg s_{lfrom} \leftarrow s_i$ ▷ If s is a sequence in d, $\neg s$ denotes the corresponding
    $lfrom \leftarrow lfrom + 1$ ▷ sequence in $\hat{d}$ (and vice versa).
  if $s_i > pivot$ **then**
    $\neg s_{gfrom} \leftarrow s_i$
    $gfrom \leftarrow gfrom + 1$

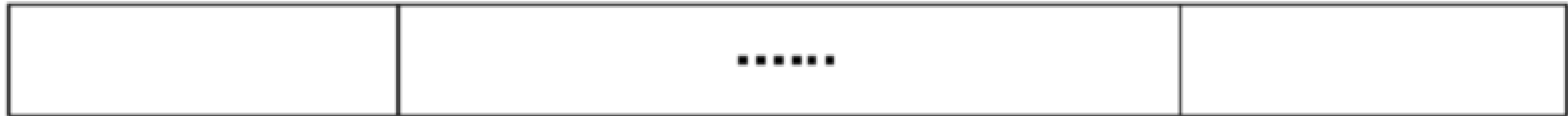
if $threadid = 0$ **then**
  if $FAA(blockcount, -1) = 0$ **then** ▷ Check if this is the last block in the sequence to finish.
    for $i \leftarrow sstart, i < send, i \leftarrow i + 1$ **do** ▷ Fill in pivot value.
      $d_i \leftarrow pivot$
    $lpivot \leftarrow median(\neg seq_{oldstart}, \neg seq_{(oldstart+sstart)/2}, \neg seq_{sstart})$
    $gpivot \leftarrow median(\neg seq_{send}, \neg seq_{(send+oldend)/2}, \neg seq_{oldend})$
    result $\leftarrow$ **result** $\cup \{(\neg seq_{oldstart \to sstart}, lpivot)\}$
    result $\leftarrow$ **result** $\cup \{(\neg seq_{send \to oldend}, gpivot)\}$

Phase 2

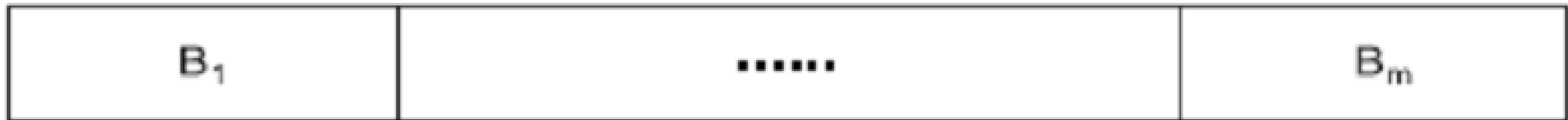
- Jeder Thread bekommt nun seinen eigenen Block zugewiesen

  → Thread-Synchronisation nicht mehr nötig
- Läuft ausschließlich auf dem GPU
- Falls nötig, weiteres Partitionieren

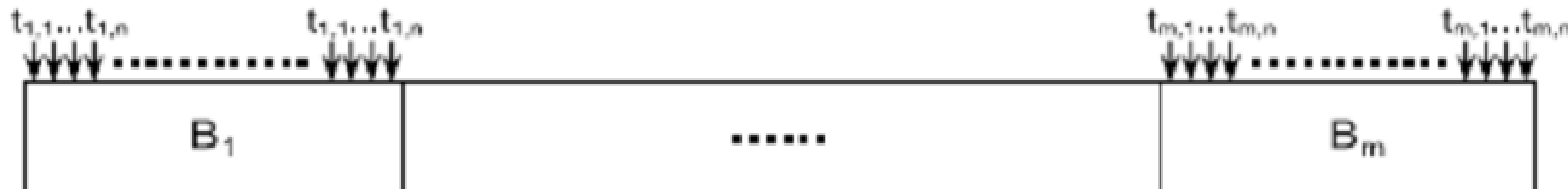
Partitionierung



(a) The sequence to be sorted is divided into *m* equally sized sections



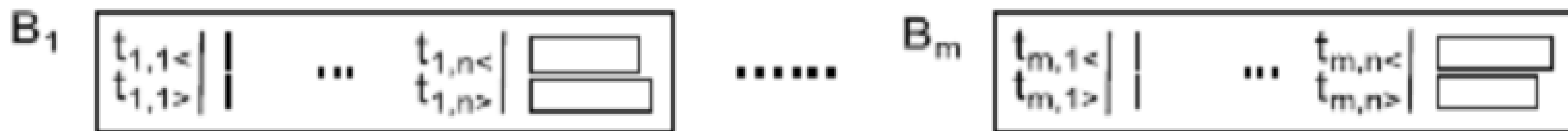
(b) Thread blocks are assigned to the sections



(c) Each thread block goes through its assigned section...

Partitionierung (cont‘d)

...keeping track of the number of elements above and below the pivot

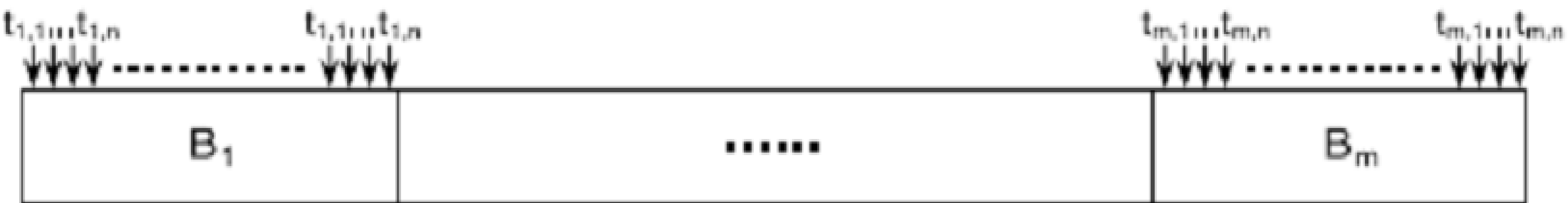


(d) Each thread block calculates the cumulative sum of the two types of elements seen

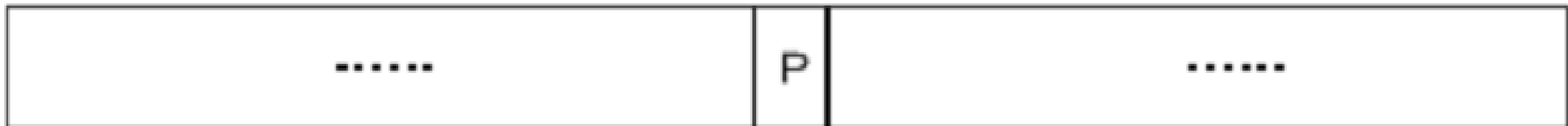
(e) A cumulative sum of each thread blocks total is calculated

Partitionierung (cont'd)

(f) Each thread uses the cumulative sums to find out where to write



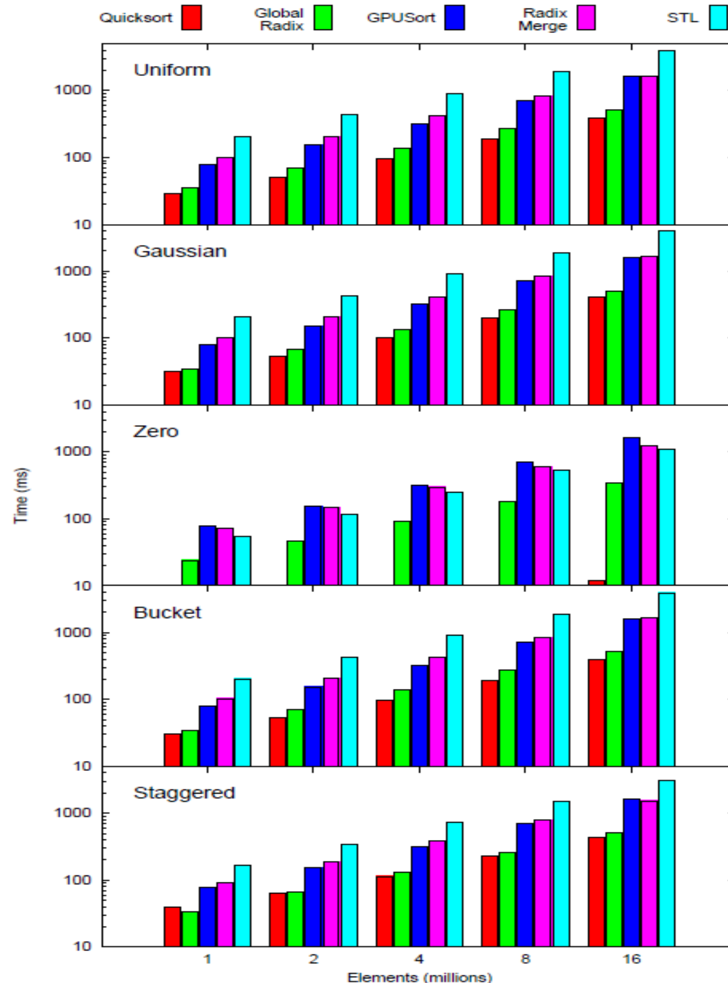
(g) Each thread block goes through its assigned data, writing it to the auxiallary array



(h) Block B_m fills the gap between the left and right subsequence with the pivot value

4. Beispiele

8800 GTX



= Standard Template Library

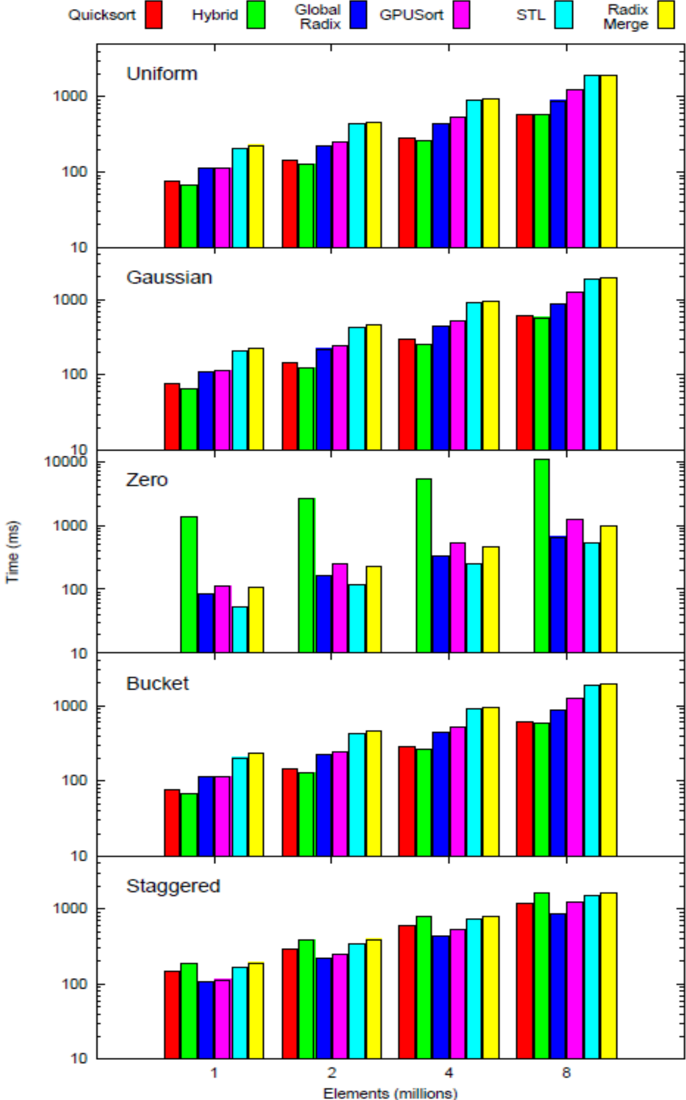
Zufällige Zahlen zwischen 0 und 2^{31}

Durchschnitt von 4 zufällig gewählten Zahlen aus dem Uniform-Teil

Eine zufällige Konstante

Einteilung in Blöcke mit verschiedenen Werten

8600GTS



Quellen

http://www.nvidia.com/object/cuda_home_new.html

http://www.cse.chalmers.se/research/group/dcs/gpuqsortdcs.html