Provably-efficient GPU algorithms

Nodari Sitchinava
University of Hawaii

Joint work with Volker Weichert

July 1, 2014
GPU computing

- GPGPU – graphics cards for computation
- Hundreds of cores
- Thousands of threads

Nodari Sitchinava – Provably-efficient GPU algorithms
The Challenge

- How to write efficient program?
- How to design algorithms?
- What makes algorithms efficient on GPUs?
GPUs

Global Memory ($n \leq 3$GB)

Shared (Local) Memory ($M \approx 48$ kB)

Registers ($\approx 32$ kB)

Nodari Sitchinava – Provably-efficient GPU algorithms
GPUs

Global Memory ($n \leq 3\text{GB}$)
- Coalesced accesses ($w = 32$)

Shared (Local) Memory ($M \approx 48 \text{kB}$)

Registers ($\approx 32 \text{kB}$)
GPUs

**Global Memory** \((n \leq 3\text{GB})\)
- Coalesced accesses \((w = 32)\)
- Hyperthreading

**Shared (Local) Memory** \((M \approx 48\text{ kB})\)

**Registers** \((\approx 32\text{ kB})\)
GPUs

**Global Memory** $(n \leq 3\text{GB})$
- Coalesced accesses $(w = 32)$
- Hyperthreading

**Shared (Local) Memory** $(M \approx 48\text{ kB})$
- Memory banks

**Registers** $(\approx 32\text{ kB})$
GPUs

Global Memory ($n \leq 3\text{GB}$)
- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48\text{ kB}$)
- Memory banks

Registers ($\approx 32\text{ kB}$)
GPUs

- **Global Memory** \((n \leq 3\text{GB})\)
  - Coalesced accesses \((w = 32)\)
  - Hyperthreading

- **Shared (Local) Memory** \((M \approx 48\text{kB})\)
  - Memory banks

- **Registers** \((\approx 32\text{kB})\)
  - Private to each thread

Nodari Sitchinava – Provably-efficient GPU algorithms
GPUs

Global Memory ($n \leq 3\text{GB}$)
- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48\text{ kB}$)
- Memory banks

Registers ($\approx 32\text{ kB}$)
- Private to each thread
- Fastest if addressable at compile time
GPU: PRAM model?
GPU: PRAM model?

Parallel scan throughput
- Coalesced accesses: 165GB/s
- Random access: 5GB/s

PRAM Model

Nodari Sitchinava – Provable-efficient GPU algorithms
GPU: PRAM with local caches and blocked accesses?

- Access to global memory in contiguous blocks
Parallel External Memory (PEM) Model (SPAA ’08)

**PEM**: A simple model of locality and parallelism

- Explicit cache replacement
- Up to $P$ block transfers $= 1$ I/O
- Block-level CREW access
**Parallel External Memory (PEM) Model (SPAA ’08)**

**PEM**: A simple model of locality and parallelism

- Explicit cache replacement
- Up to $P$ block transfers $= 1$ I/O
- Block-level CREW access

**Complexity**: Parallel I/O complexity – number of *parallel* block transfers
$w = 32$ cores per cache

PEM model has only one core per cache
Model

- $w$ cores/threads per cache
- Always load data into shared memory of size $M$
- Access global memory in blocks of size $B = \Theta(w)$
Branch divergence

SIMD algorithms
- E.g. sorting networks for sorting in shared memory
Bank conflicts

What about bank conflicts?
Effect of bank conflicts on runtime

![Graph showing the effect of bank conflicts on runtime. The graph compares the runtime of kernel 3, kernel 3 bank conflicts, and an improved kernel 3 runtime. The x-axis represents the number of colors, and the y-axis represents the runtime in milliseconds. The graph shows a significant increase in runtime with an increase in bank conflicts.](image-url)
GPU: PEM with multiprocessors and bank conflict free processing

Model

- \( w \) cores/threads per cache
- Access global memory in blocks of size \( B = \Theta(w) \)
- Always load data into shared memory of size \( M \)
- Design SIMD algorithms with no bank conflicts
Bank Conflict Free Computations
# Modeling bank conflicts

## Matrix view of shared memory

- Column-major layout in $w \times (M/w)$ matrix
- One thread per row
- Convert column-to row-major to process columns

Conversion for square matrices is a matrix transposition

| Memory Bank 0 | A[0] | A[8] | A[16] | A[24] | A[32] | A[40] | A[48] | A[56] |
|---------------|------|------|-------|-------|-------|-------|-------|-------|
| Memory Bank 1 | A[1] | A[9] | A[17] | A[25] | A[33] | A[41] | A[49] | A[57] |
| Memory Bank 2 | A[2] | A[10]| A[18] | A[26] | A[34] | A[42] | A[50] | A[58] |
| Memory Bank 3 | A[3] | A[11]| A[19] | A[27] | A[35] | A[43] | A[51] | A[59] |
| Memory Bank 4 | A[4] | A[12]| A[20] | A[28] | A[36] | A[44] | A[52] | A[60] |
| Memory Bank 5 | A[5] | A[13]| A[21] | A[29] | A[37] | A[45] | A[53] | A[61] |
| Memory Bank 6 | A[6] | A[14]| A[22] | A[30] | A[38] | A[46] | A[54] | A[62] |
| Memory Bank 7 | A[7] | A[15]| A[23] | A[31] | A[39] | A[47] | A[55] | A[63] |
Example: Bank Conflict Free Sorting

**ShearSort [Sen et al. 1986]**

Repeat \(\log(M/w)\) times:
- Sort columns in alternate order
- Sort rows in increasing order

**Result:** Sorted matrix in column-major order
Complexity

If $M = w^2$

$O((\text{sort}(w) \cdot \log w)$ time to sort $M$ elements

Using a sorting network to sort each row/column

$O(w \log^3 w)$ time
Sorting network vs. ShearSort

- ShearSort with register optimization
- ShearSort
- Odd-Even Transposition Sort

Nodari Sitchinava – Provably-efficient GPU algorithms
Bank Conflict Free Sorting (Basecase)
Bank Conflict Free Sorting (Basecase)

![Graph showing runtime vs input size for different sorting algorithms.](image-url)

- thrust mergesort
- thrust mergesort (base case 1024)
- mergesort/shearsort (base case 1024)
- mergesort/shearsort (base case 8192)

Nodari Sitchinava – Provably-efficient GPU algorithms
Bank Conflict Free Merging

Merging two sorted streams

- Load $\frac{M}{2}$ items from each stream
- Repeat:
  - Sort $M$ items
  - Output $\frac{M}{2}$ smallest items
  - Assign the rest $\frac{M}{2}$ items to the stream with the larger largest element
  - Load $\frac{M}{2}$ items from the other stream

Complexity

$O\left(\frac{N}{M} \cdot \text{sort}(M)\right)$ time
# Bank Conflict Free Sorting (Full)

## BCFMergeSort
- Sort $N/M$ runs of $M$ elements
- Repeatedly merge pairs of streams until $2P$ streams remain
- Using distribution, partition streams into $P$ buckets
- Repeatedly merge pairs of streams within each bucket

## Complexity
- $O\left(\frac{N}{Pw} \cdot (1 + \log^2(w) \cdot \log(N/w))\right)$ time
- $O(1 + \log(N/M))$ parallel scans
Bank Conflict Free Sorting (Full)

![Graph showing runtime per element vs input size for different sorting algorithms.](image)

- warpsort
- thrust
- BCFMergesort
- BCFMergesortPB

Nodari Sitchinava – Provably-efficient GPU algorithms
Conclusions

GPU Model

- Access global memory in blocks of size $B = \Theta(w)$
- Cache of size $M$ (matrix)
- $w$ cores/threads per cache
- Design SIMD algorithms that process items in cache by rows or columns
- Optimize by using registers
Future work

- Private registers of Kepler architecture as rows of the matrix
- Tradeoff between number of “processors” and cache size
- Design more algorithms
- Automatic scheduling of threads to cores
Thank you!