

Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors

Seminar Talk

<http://www.hpc.rwth-aachen.de/teaching/seminar/cthpc/>

Angelika Schwarz

... based on work of Nathan Bell and Michael Garland

<http://dx.doi.org/10.1145/1654059.1654078>

January 27, 2015

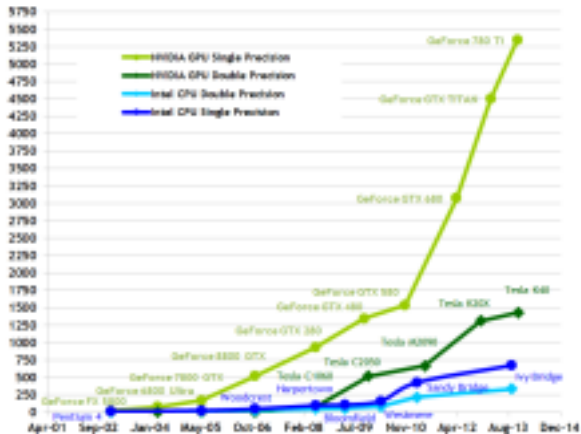
Motivation

Compute

$$\begin{pmatrix} y \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{pmatrix} := \begin{pmatrix} \blacksquare & & & & & & & \\ & \blacksquare & & & & & & \\ & & \blacksquare & & & & & \\ & & & \blacksquare & & & & \\ & & & & \blacksquare & & & \\ & & & & & \blacksquare & & \\ & & & & & & \blacksquare & \\ \blacksquare & & & & & & & \\ \blacksquare & & & & & & & \blacksquare \end{pmatrix}^A \cdot \begin{pmatrix} x \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{pmatrix}$$

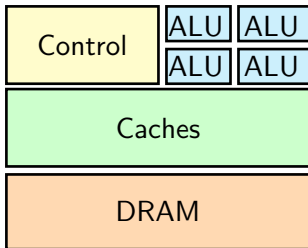
Why GPUs?

Theoretical GFLOPs

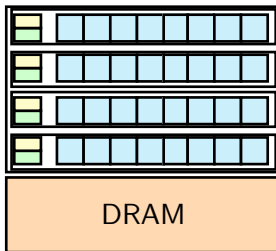


Source: NVIDIA CUDA C Programming Guide Version 6.5

GPU vs. GPU



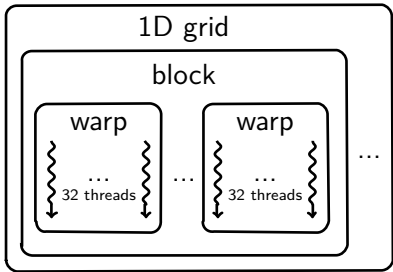
- small number of cores
- relatively large amount of memory per core
- memory hierarchy



- large number of cores
- massive hardware multithreading
- power efficient
- small cache sizes, hide latency by multithreading

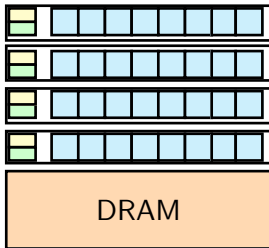
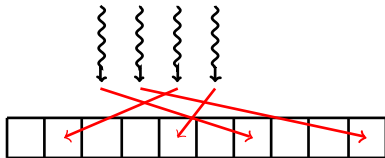
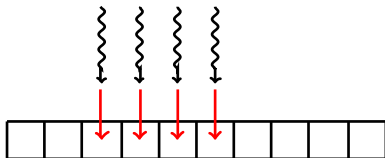
Source: NVIDIA C Programming Guide Version 6.5, adapted

GPUs in short - SIMT



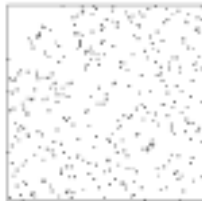
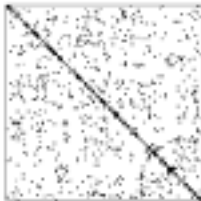
- partitioning into warps
- a warp executes one common instruction at a time
- full efficiency when all threads of a warp take the same execution path

GPUs in short - Coalescing



- coalescing of global memory accesses
- ⇒ GPU program gets close to peak memory bandwidth

Sparse $\neq$ Sparse



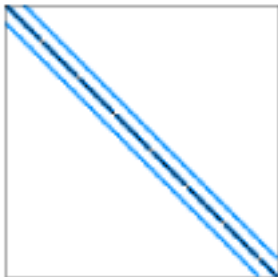
Source: Florida Matrix Collection (rdb450l, bcsprw05, GD98_c)

Agenda

- Compute $\mathbf{y} := \mathbf{A} \cdot \mathbf{x}$
 - $\mathbf{x}, \mathbf{y}$ dense
 - $\mathbf{A}$ sparse
 - sparsity pattern: structured $\leftrightarrow$ unstructured
- GPU excels if
 1. all threads of a warp follow the same execution path
 2. data access is coalesced
 3. load imbalance is avoided

Challenge: map potentially highly irregular matrices onto GPU architecture s.t. requirements 1. - 3. are met

Diagonal Format



Diagonal Format

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 7 \\ 3 & 0 & 0 & 8 \end{pmatrix}$$

Diagonal Format

$$\begin{pmatrix}
 * & * & * & 1 & 0 & 0 & 0 & * & * & * \\
 * & * & * & 0 & 4 & 5 & 0 & * & * & * \\
 * & * & * & 0 & 0 & 6 & 7 & * & * & * \\
 * & * & * & 3 & 0 & 0 & 8 & * & * & *
 \end{pmatrix}$$

Handwritten red annotations above the matrix: $3 \dots -1 \ 0 \ 1$. Red arrows indicate the diagonal elements: from (1,4) to (4,4), from (2,5) to (4,6), and from (3,7) to (4,8).

values:

$$[* * * 3 | 1 \ 4 \ 6 \ 8 | 0 \ 5 \ 7 \ *] \xrightarrow{0}$$

offsets:

$$[-3 \qquad \qquad 0 \qquad \qquad 1 \qquad \qquad]$$

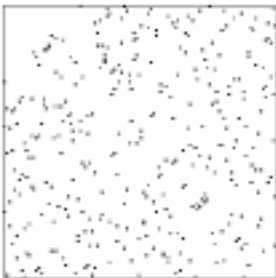
Diagonal Format – Parallelisation

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 7 \\ 3 & 0 & 0 & 8 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

values: $\begin{bmatrix} * & * & * & 3 & | & 1 & 4 & 6 & 8 & | & 0 & 5 & 7 & * \end{bmatrix}$

offsets: $\begin{bmatrix} -3 & 0 & 1 \end{bmatrix}$

Coordinate Format



Coordinate Format

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 7 \\ 3 & 0 & 0 & 8 \end{pmatrix}$$

(1,2,5)

```
row_indices: [0 1 1 2 2 3 3]
col_indices: [0 1 2 2 3 0 3]
values:      [1 4 5 6 7 3 8]
```

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 7 \\ 3 & 0 & 0 & 8 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

row_indices: [0 1 1 2 2 3 3] → shared memory

col_indices: [0 1 2 2 3 0 3]

values: [1 4 5 6 7 3 8]

$\downarrow$ $\downarrow$
 $A_{ij} \cdot x_j \dots$

→ shared memory

$$\sum_j A_{ij} \cdot x_j$$

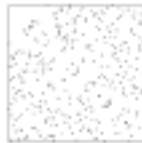
Coordinate Format – Parallelisation (2)

		^{warp}																				
lane		0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7				
row_indices		[0	0	0		1	1	1		2	2		2	2	2		3	3		4	4	4]
$A_{ij} \cdot x_j$	local result	[3	4	2		6	1	2	1	5		2	6	1	8	1	5	0	1]			
segmented reduction																						
$\sum_j A_{ij} \cdot x_j$	sums	[3	7	9		6	7	9	1	6	2	8	9		8	9	5	5	6]			

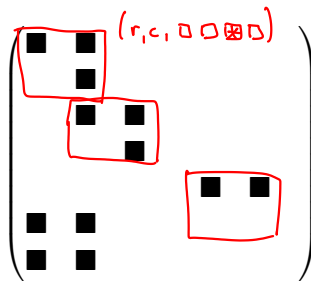
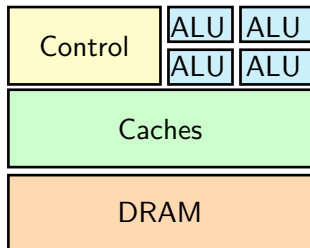
y_0 (points to 3, 7, 9)
 y_1 (points to 6, 7, 9)
+ (points to 1, 6, 2, 8, 9)

Summary Matrix Formats

DIA [ELL] [CSR] [HYB] COO



Outlook: Sparse Matrix-Vector Product on CPUs



- memory hierarchy $\rightarrow$ data locality
- reduce indexing overhead by tiling
- block matrix formats

Summary

- Flavours of sparsity
 - structured vs. unstructured
 - different matrix format account for distinct sparsity patterns
 - matrix formats discussed here: diagonal format, coordinate format
- Design goals for GPU programming
 1. all threads of a warp should follow the same execution path
 2. coalesced data access
 3. work balance