Optimization techniques

David Celný

Department of Physical Chemistry, UCT Prague

celnyd@vscht.cz

September 2, 2021

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Overview

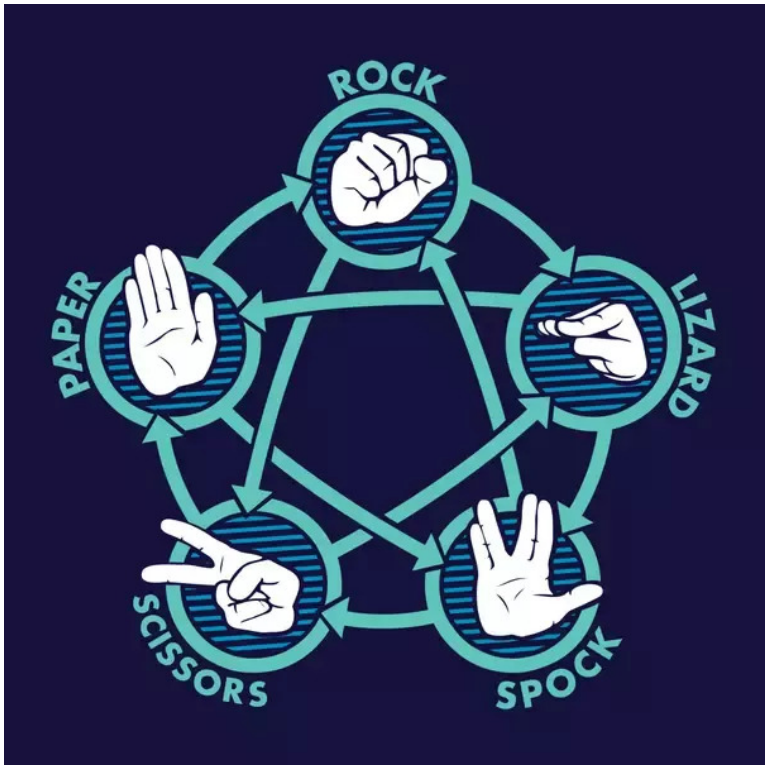
Techniques

Memory pattern

- coalesced accesses
- shared memory conflicts
- data race
- Array of Structures vs Structure of Arrays

Computation

- sp, dp, sfu
- thread divergence
- Instruction Level Parallelism
- computationally bound



	"Fermi"	"Fermi"	"Kepler"	"Kepler"	"Maxwell"	"Pascal"	"Volta"	"Turing"	"Ampere"
Tesla GPU	**GF100**	**GF104**	**GK104**	**GK110**	**GM200**	**GP100**	**GV100**	**TU104**	**GA100**
Compute Capability	2.0	2.1	3.0	3.5	5.3	6.0	7.0	7.0	8.0
Streaming Multiprocessors (SMs)	16	16	8	15	24	56	84	72	128
FP32 CUDA Cores / SM	32	32	192	192	128	64	64	64	64
FP32 CUDA Cores	512	512	1,536	2,880	3,072	3,584	5,376	4,608	8,192
FP64 Units	-	-	512	960	96	1,792	2,688	-	4,096
Tensor Core Units							672	576	512
Threads / Warp	32	32	32	32	32	32	32	32	32
Max Warps / SM	48	48	64	64	64	64	64	64	64
Max Threads / SM	1,536	1,536	2,048	2,048	2,048	2,048	2,048	2,048	2,048
Max Thread Blocks / SM	8	8	16	16	32	32	32	32	32
32-bit Registers / SM	32,768	32,768	65,536	65,536	65,536	65,536	65,536	65,536	65,536
Max Registers / Thread	63	63	63	255	255	255	255	255	255
Max Threads / Thread Block	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024
Shared Memory Size Configs	16 KB	16 KB	16 KB	16 KB	96 KB	64 KB	Config	Config	Config
	48 KB	48 KB	32 KB	32 KB			Up To	Up To	Up To
			48 KB	48 KB			96 KB	96 KB	164 KB
Hyper-Q	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic Parallelism	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Unified Memory	No	No	No	No	No	Yes	Yes	Yes	Yes
Pre-Emption	No	No	No	No	No	Yes	Yes	Yes	Yes
Sparse Matrix	No	No	No	No	No	No	No	No	Yes

Figure: Comparison of the TESLA GPU series capabilities.
@ https://www.nextplatform.com/2020/05/28/diving-deep-into-the-nvidia-ampere-gpu-architecture/

coalesced global mem. access

- consecutive threads access consecutive memory blocks
- depends on the memory cache lane size *(128byte)*
- single transaction should serve multiple cores
- both global memory read write

conclusion

- no holes
- ordered
- aligned *(32byte,128byte)*

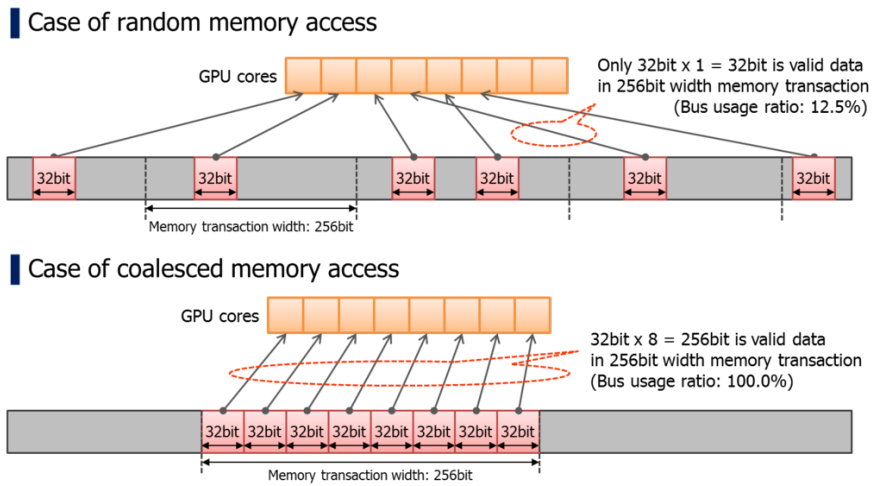


Figure: Example of un-coalesced memory access.
@ https:\\kaigai.hatenablog.com\entry2016\11\17\070708

conflict-less shared mem. access

- priority access to shared memory banks
- banks size is limited *(4,8byte)*
- use stride to ensure non-conflict
- helps with coalesced access to global mem.

conclusion

- either each access own bank
- all access the same bank

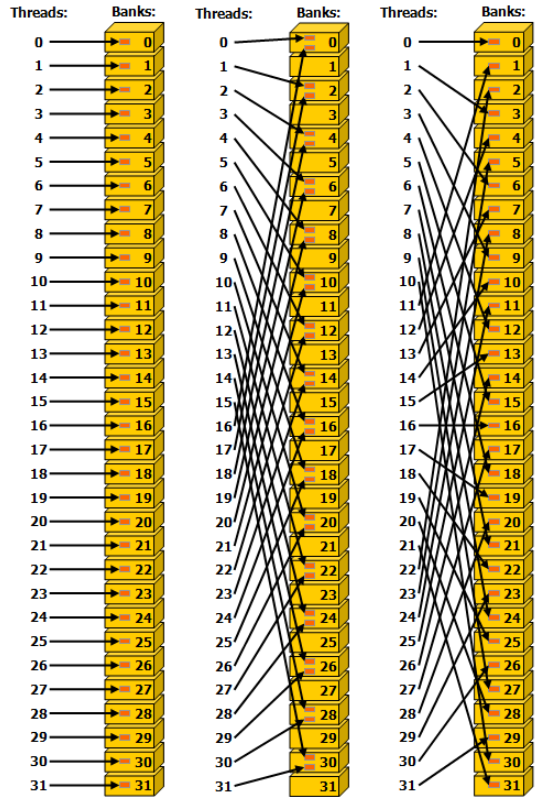


Figure: Example of strided shared mem. access.
© docs.nvidia.com

data race

- two or more threads change same memory space
- produce nondeterministic result
- use barriers *__syncthreads()*
- use the atomic operations

  (force serialize, slower)

conclusion

- organize algorithm better
- otherwise barriers
- atomics are slower convenience

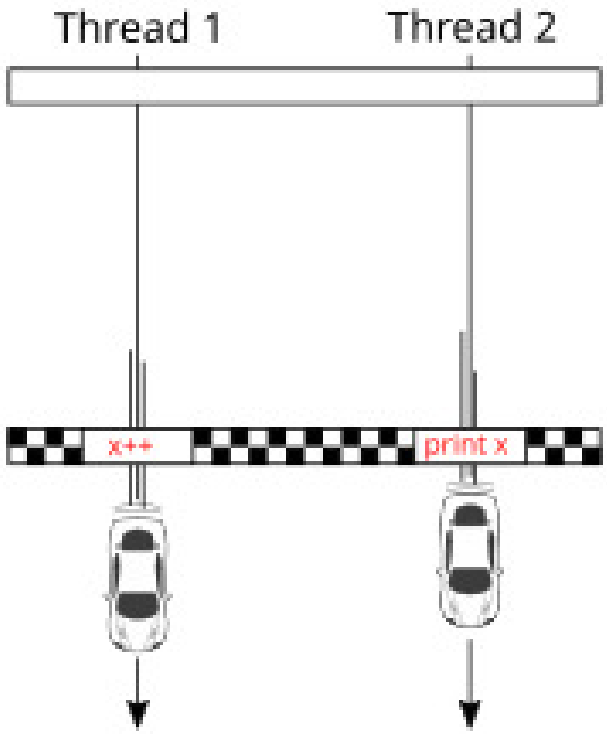


Figure: Data race visualization.
@ https://programming.guide/go/data-races-explained.html

SoA vs AoS

- how are data organized in memory
- depend on problem type and data sizes
- type use enforces parallelization strategy *(vertical, horizontal)*
- need to satisfy coalesced access and solve the problem

conclusion

- analyze problem and pick type
- use corresponding strategy
- for GPU default use SoA

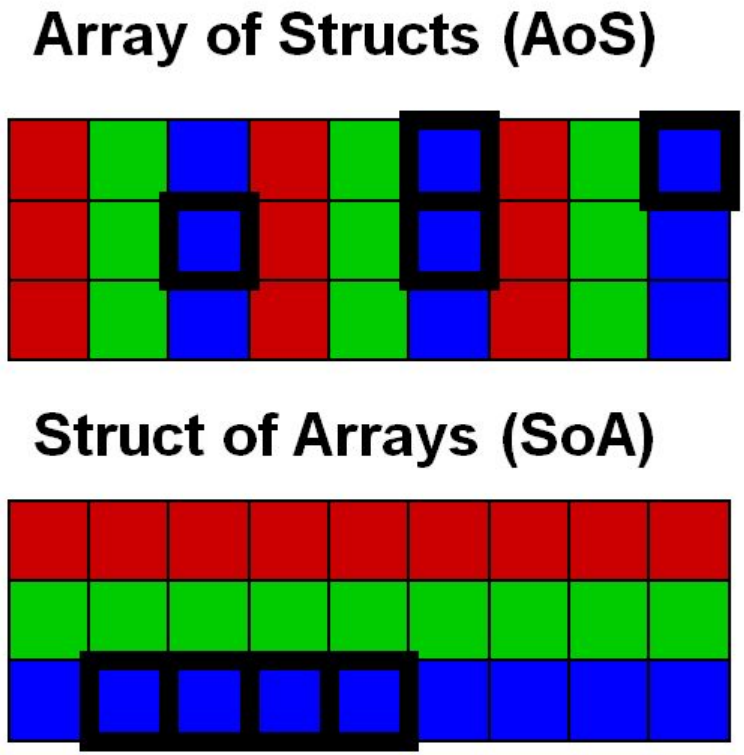


Figure: AoS and SoA stored in memory.
© https://asc.ziti.uni-heidelberg.de/en/node/18


NVIDIA Before
NVIDIA
RADEON
GRAPHICS
AMD
NVIDIA Now
NVIDIA
RTX2080Ti
NVIDIA
RTX3070
MemeZila.com

Varied speed for varied tasks

- different count of unit by purpose
- throughput can be limited by utilization
- shift to spread the tasks over units
- perform "enough" computation per mem. load

conclusion

- int/float is cheap
- dont misuse special functions *(sin,exp,log)*
- better to over-calculate

Figure: Ampere SM diagram.
@ docs.nvidia.com

thread divergence

- branching in the kernel code
- can't concurrently execute different instructions
- forces branch sequencing
- all branches are evaluated
- new architectures can handle short conditions

conclusion

- no conditions
- or short and simple

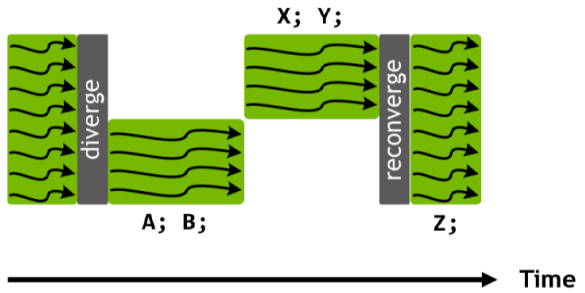


Figure: Pascal and earlier thread div.
@ https://www.peterstefek.me/shader-branch.html

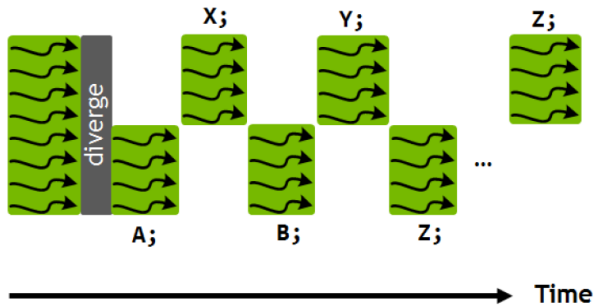


Figure: Volta and later thread div.
@ https://www.peterstefek.me/shader-branch.html

ILP

- more work done per core
- registers are expensive for thread indexes
- more calculation per memory load
- automatized by loop unroll macro in code
- useful for enough parallel or decrease occupancy

conclusion

- not use immediately
- use with caution

  (may worsen)
- complicates code

  (loops in kernel)

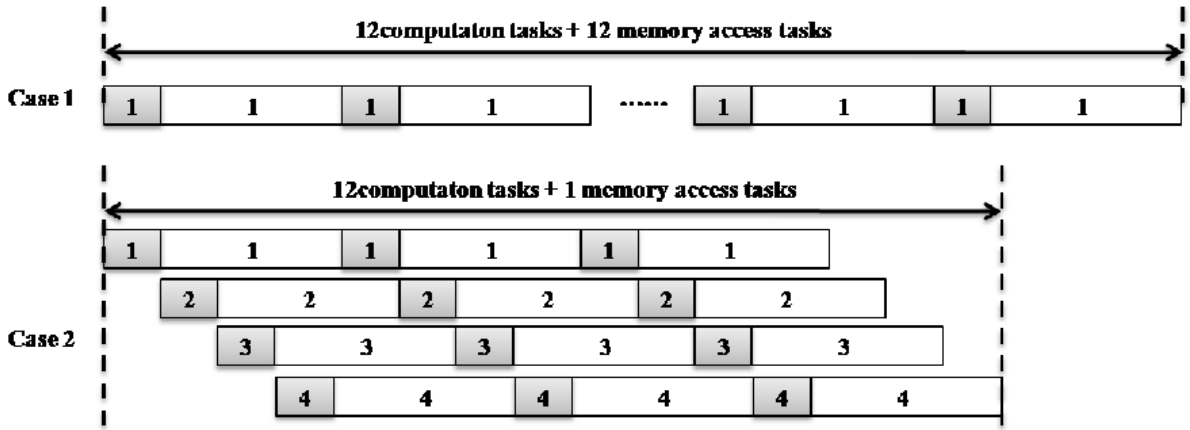


Figure: Cheng, Luo and R. Suda. "An execution time prediction analytical model for GPU with instruction-level and thread-level parallelism awareness." (2011).

Summary

- beware hardware dependence
- beware coalesced accesses
- beware data races
- beware branches in code
- beware utilization of units

References

Gerassimos Barlas (2015)
Multicore and GPU Programming: An Integrated Approach
Elsevier publishers ISBN: 978-0-12-417137-4

Thomas Sterling, Matthew Anderson & Maciej Brodowicz (2018)
High Performance Computing: Modern Systems and Practices
Elsevier publishers ISBN: 978-0-12-420158-3

Jason Sanders & Edward Kandrot (2011)
CUDA by Example: An introduction to General-Purpose GPU programming
Addison-Wesley ISBN-10: 978-0-13-138768-3

List of Nvidia graphics processing units (cited 2021)
https:\\en.wikipedia.org\wiki\List_of_Nvidia_graphics_processing_units

- overview image [Cit. 02.09.2021]. Available from https:\\me.me\i\gtx-1070-vs-a-relationship-gi-geforce-runs-pubg-still-1bf254a73db84c2dac0db220d25802f4
- techniques image [Cit. 02.09.2021]. Available from https:\\static.wixstatic.com\media\903056_39aa9523c70a428684be9744580b0b1bm̃v2.p
- computation image [Cit. 02.09.2021]. Available from https:\\memezila.com\wp-content\Nvidia-before-vs-Nvidia-now-meme-6521.png

Practical session

1. start/setup your development tool
2. get the source code
3. follow the **instructions in code**
   - if unsure → first think about it
   - if still lost → "google" it
   - if can't find → ask about it *(personally or mail)*
4. make sure your code **compile**
5. make sure your code **work**
6. send your code to me *(celnyd@vscht.cz)*

Until next time