CUDA

Superpočítač do každej domácnosti

Univerzita Pavla Jozefa Šafárika v Košiciach

Ladislav Mikeš

27.02.2018 (v02)

Overview

Výpočtový výkon

$$FLOPS = IPC * Freq * N_{thread}$$

Výpočtový výkon

CPU	GHz	T	GFLOPS	$
Intel Core i9-7980XE	3.87	36	562	$1920
Intel Core i7-8700K	4.57	12	223	$340
AMD EPYC 7551	2.99	64	654	$3800
AMD Ryzen 7 1800X	3.82	16	236	$350

Cyklus

```
for i=1:N
        data[i] = compute(i);
```

Výpočtový výkon GPU

+ Veľké množstvo výpočtových jednotiek (na výpočet geometrie)
– Nízky výkon na jednotku
+ Vysoká priepustnosť pamäte
– Obmedzená veľkosť pamäte
+ Cenovo efektívne
– Cenovo efektívne

Výpočtový výkon

CPU	GHz	T	GFLOPS	$
Intel Core i9-7980XE	3.87	36	562	$1920
Intel Core i7-8700K	4.57	12	223	$340
AMD EPYC 7551	2.99	64	654	$3800
AMD Ryzen 7 1800X	3.82	16	236	$350
Nvidia 1080 Ti	1.48	3584	10609	"$700"
Nvidia 1050 Ti	1.29	768	1981	"$139"

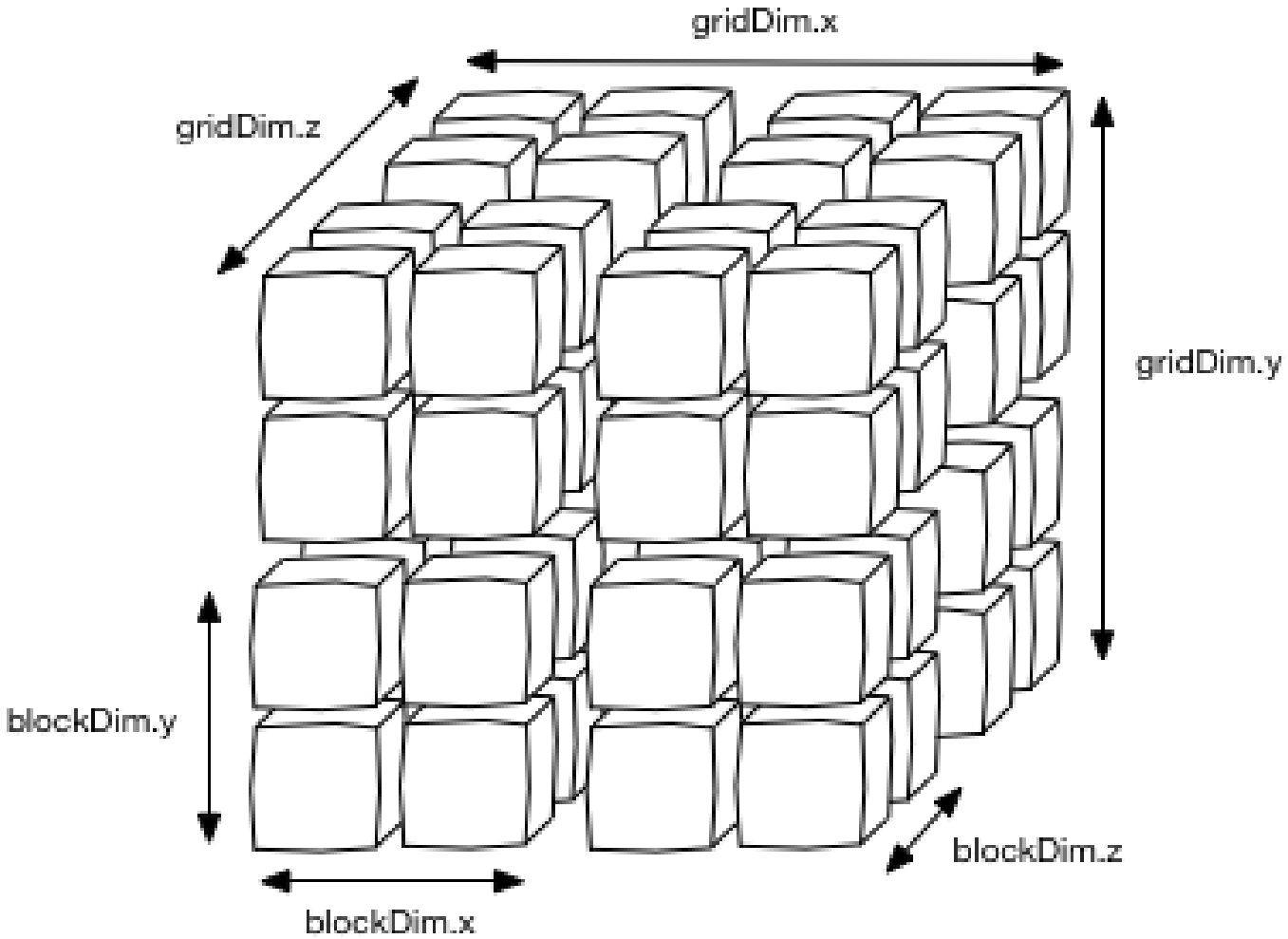
CUDA

- Compute Unified Device Architecture
- nadstavba C (owrapované aj do iných)
- používa vlastný (pred-)kompilátor nvcc
- vstavané funkcie s prefixom cuda (cudaMemcpy)
- vlastné funkcie pre host (CPU) a/alebo device (GPU)
- paralelizovaný kód (compute(i)) - kernel

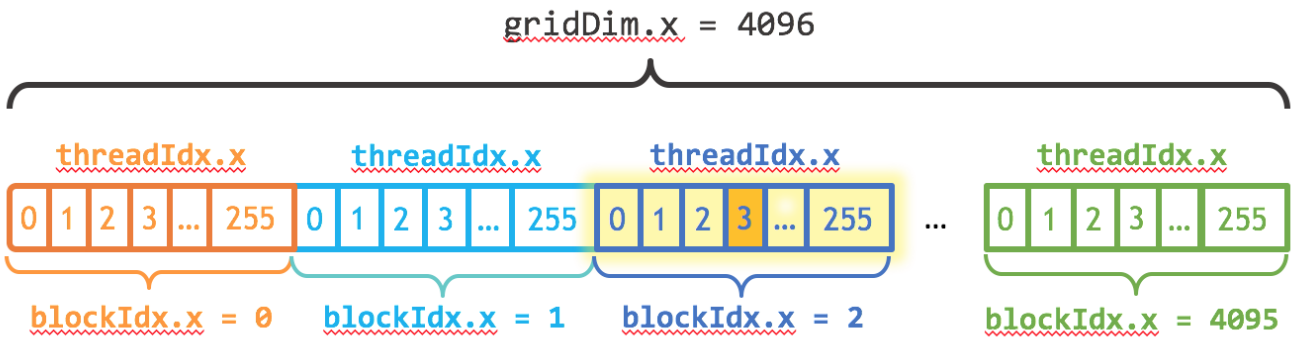
Hello word

cudaGetDeviceCount(*int)
cudaGetDeviceProperties(*cudaDeviceProp, idx);

Indexovanie - compute(i)



Indexovanie



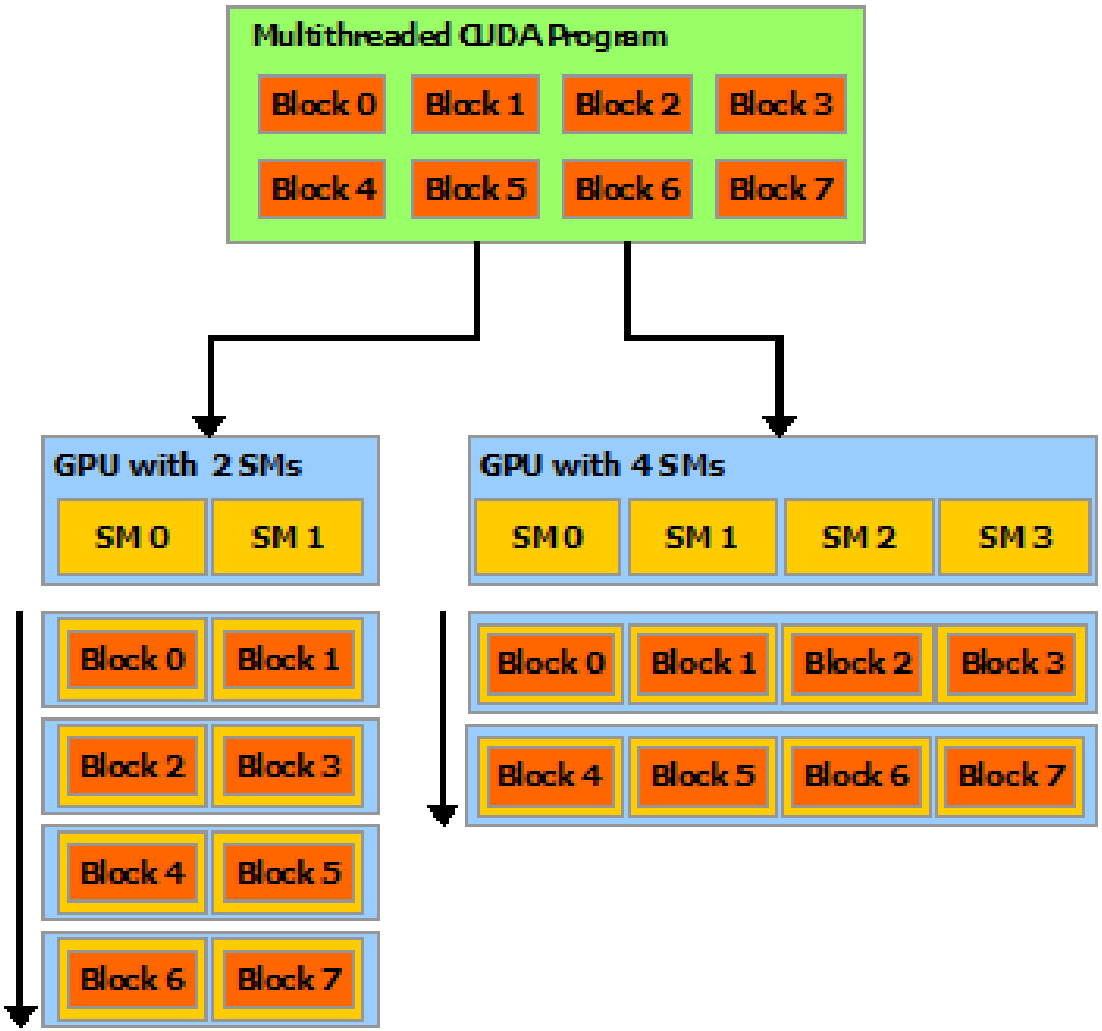

```
index = blockIdx.x  * blockDim.x + threadIdx.x

index =    (2)      *   (256)    +     (3)      = 515
```

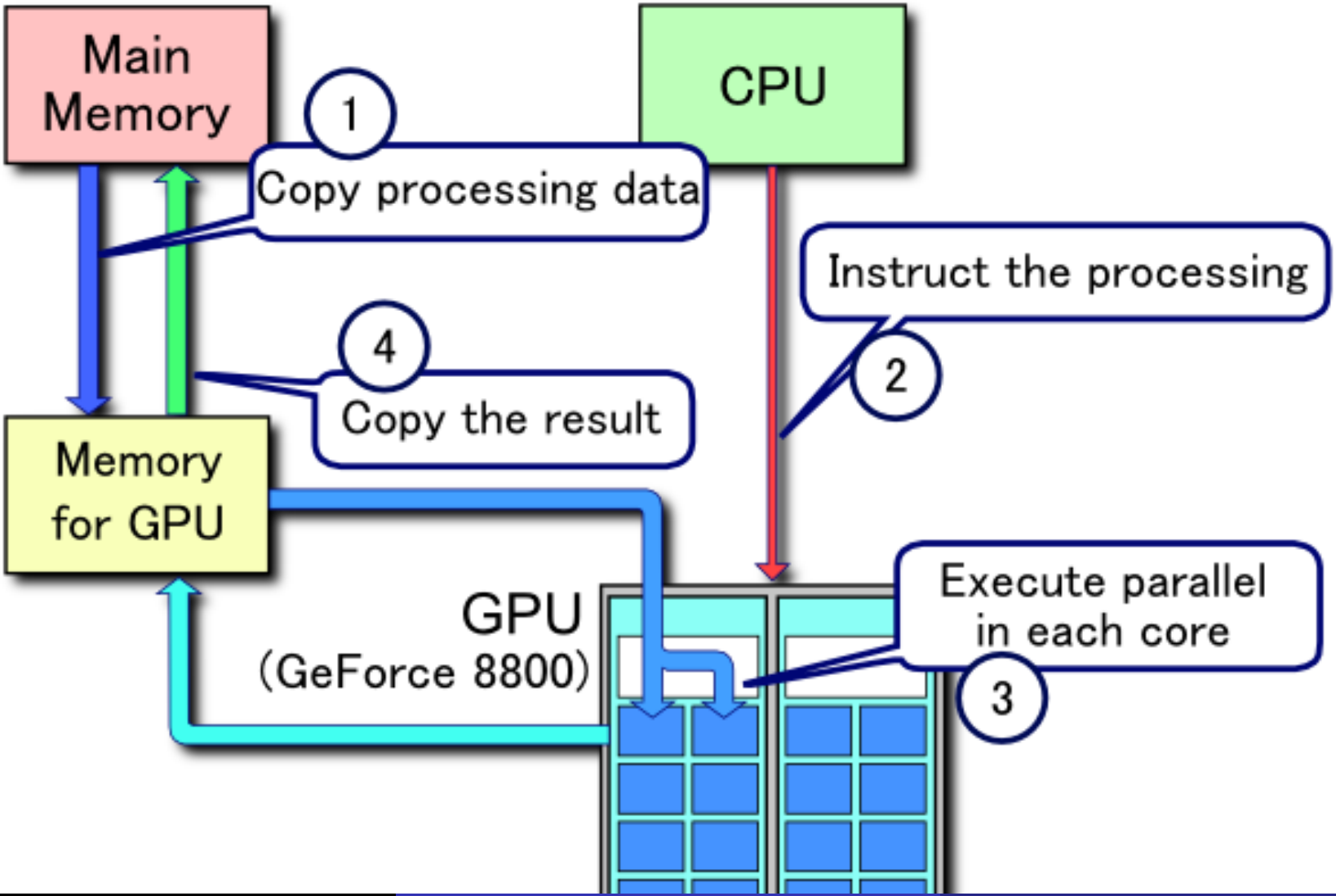
Indexovanie - limity

Technical specifications	Compute capability (version)														
	1.0	1.1	1.2	1.3	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2
Maximum dimensionality of grid of thread blocks	2				3										
Maximum x-dimension of a grid of thread blocks	65535				$2^{31} - 1$										
Maximum y-, or z-dimension of a grid of thread blocks	65535														
Maximum dimensionality of thread block	3														
Maximum x- or y-dimension of a block	512				1024										
Maximum z-dimension of a block	64														
Maximum number of threads per block	512				1024										
Maximum number of resident blocks per multiprocessor	8					16				32					

Granularita výpočtu



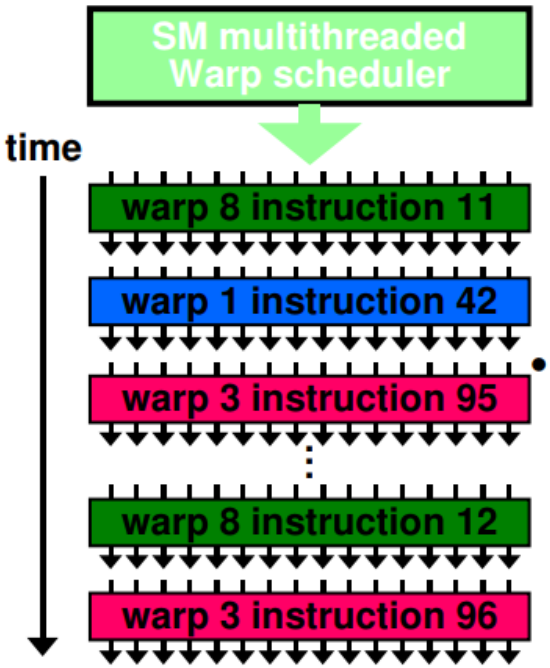
Beh programu



GP106(GL) : GTX 1060 / Quadro P2000

SM : 10
Stream Processors (CORES) : 1280 (128/SM)
Warp size : 32
Blocks per SM (6.1) : 32
Warps per SM (6.1) : 64
Threads per SM (6.1) : 2048
Elementarnou skupinou threadov je warp, kazdy blok sa paralelne vykonava rozvrhovanim warpov na SM
G512 B512 - blok ma 16 warpov po 32 ... maximalne 4 bloky na SM, 13 blokov sekvencne

SM Warp Scheduling



- SM hardware implements zero-overhead Warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution on a prioritized scheduling policy
  - All threads in a Warp execute the same instruction when selected
- 4 clock cycles needed to dispatch the same instruction for all threads in a Warp in G80
  - If one global memory access is needed for every 4 instructions
  - A minimal of 13 Warps are needed to fully tolerate 200-cycle memory latency

Hello world

cudaMalloc() - alokuje pamat na GPU
cudaMemcpy() - kopiruje data medzi RAM a pamatou GPU
kernel<<<gdim,bdim>>>() - spusti paralelny kod
cudaDeviceSynchronize() - synchronizacna bariera
cudaMemcpy() - kopirovanie dat spat
cudaFree() - uvolnenie pamate

Testovanie spravnosti a casu

1. vygenerovat vstupy
2. vypocitat 'spravne' vysledky na CPU + zmerat cas
3. vypocitat vysledok na GPU: cisty cas alebo realny, vratane prenosov
4. porovnat vysledky

Caveats - (Ne)presna matematika

Pouzitim prepinaca -use_fast_math pri kompilacii je mozne dosiahnut lepsiu rychlost za cenu znizenej presnosti.

Caveats - Vetvenie

Kedze vsetky vlakna warpu vykonavaju tu istu instrukciu, pri vetveni sa musia navzajom pockat.
Tomuto je idealne sa vyhnut, niekedy sa da if(a){b=x;else b=y;} nahradit b=a*x+(!a)*y

Caveats - Pinned memory

Pri kopirovani medzi host a device musi engine kontrolovat suvislost zdrojovej pamate
Ak sa pouzije "pinned" pamat (vynutene kontinualna), H2D/D2H sa zrychli
napr. cudaMallocHost((void**)&h_buffer,sizeof(float)*size) a cudaFreeHost(h_buffer)

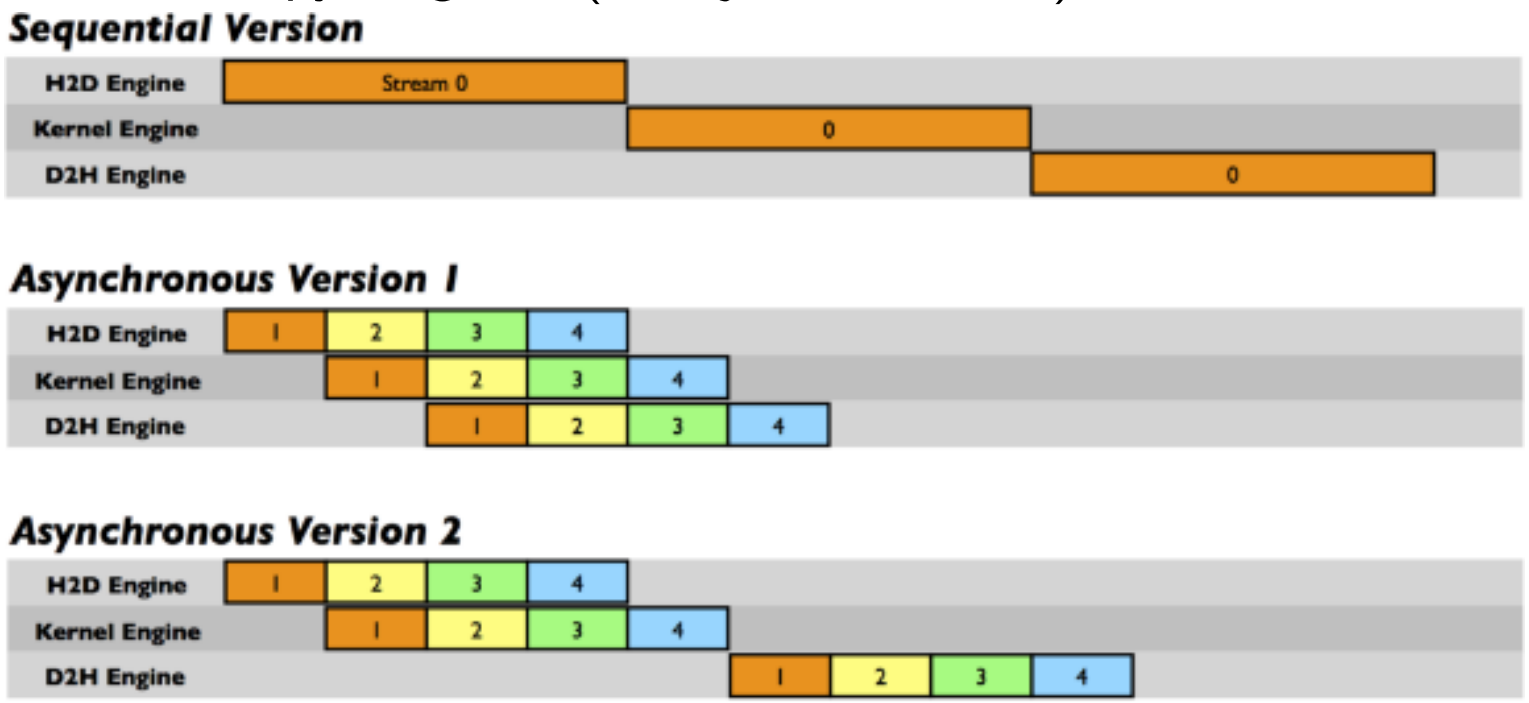
Caveats - DP

- sucasne CPU su 64-bitove, vedia pracovat s dvojitou presnostou bez vacsej ujmy
- GPU ma standardne 32-bitove jadra, 64-bitovy vykon sa dosiahne primiesanim urciteho mnozstva FP64 jadier, 1:32 u hernych, 1:3 - 1:2 u Titan, Tesla

Caveats - Pay 2 Win

Profesionalne karty maju niektore vyhody oproti hracskym

- Lepsi FP64 vykon
- (default) compute mod - moze dlho nereagovat
- Podpora ECC pamate
- Dalsi "Copy Engine" (ale aj $\geq$ GTX 9xx)



Úlohy

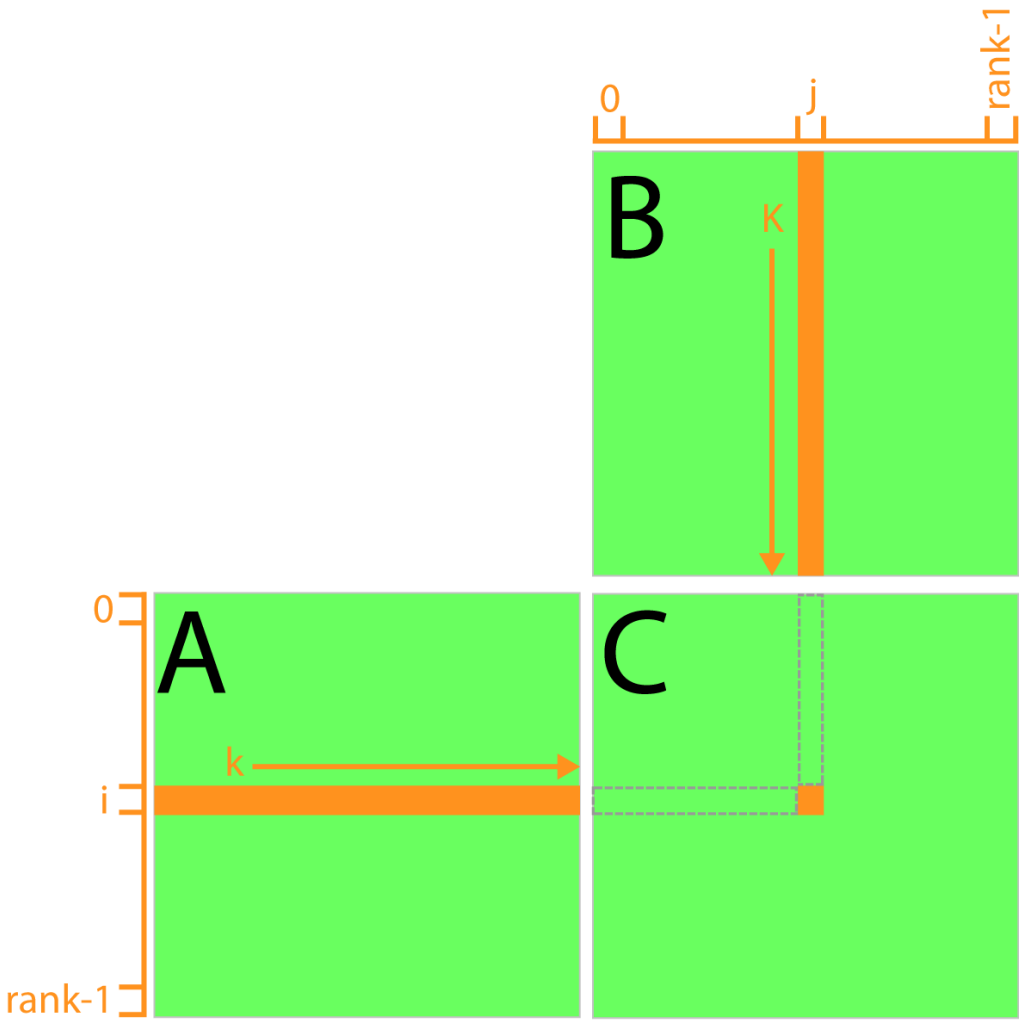
1. Ziskajte pristup ku kontu studentX , zmente heslo a vytvorte subor obsahujuci Vase meno [vnutri skolskej siete] ssh studentX@158.197.31.56 -p 222
2. spustenie programu cez curun ./main
3. Vyskusajte vplyv prepinaca -O3 na cas behu vypoctu na CPU
4. Vyskusajte vplyv prepinaca -DDOUBLE_PRECISION na cas behu vypoctu na CPU a GPU
5. Vyskusajte vplyv pouzitia cudaMallocHost((void**)&d_a,SOF*size); namiesto a=new FLOATING[size]; a cudaFreeHost(d_a); namiesto delete a; (ditto pole device)

Úlohy

6. Vyskusajte vplyv pouzitia zlozitejsej funkcie (SIN(src[i])+COS(src[i]))*factor na cas behu vypoctu na GPU, prip. optimalizacie, napr cachovanie src[i] do premennej a SICO
7. Vyskusajte vplyv prepinaca -use_fast_math na cas behu vypoctu na GPU (pri sin+cos)
8. Upravte program tak, aby pocital Hadamardov sucin (po elementoch) 2 vektorov (c[i]=a[i]*b[i])
9. Upravte predchadzajuci program a optimalizujte kod na pocitanie funkcie: if (b[i]>0.5) c[i]=SIN(a[i])*COS(b[i]); else c[i]=COS(a[i])*SIN(b[i]);
10. Najdite optimalnu velkost bloku a gridu (povodne<<<512,512>>>) pre predchadzajuci problem za pouzitia nvprof: curun nvprof ./main

A few hours later ...

Nasobenie matic



Caveats - Nahodny pristup k pamati

Pristup k pamati je obmedzenim najma u jednoduchsich vypoctov, pre urychlenie sa naraz pracuje s cachovanym blokom. Urychlenie nastane za podmienky, ze thready pristupuju k castiam toho isteho bloku.
Pri nasobeni matic nasobime riadok stlpcom (a[i][k] * b[k][j]), skusme to dekonstruovat

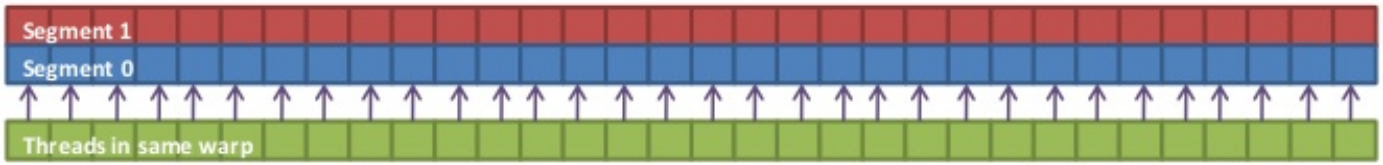
- Riadok s riadkom: a[i][k] * b[j][k]
- Stlpec so stlpcom: a[k][i] * b[k][j]

Pristup po stlpcoch pojde pomaly!

Pristup k pamati

Coalescing Examples

Simple, Stride-1:

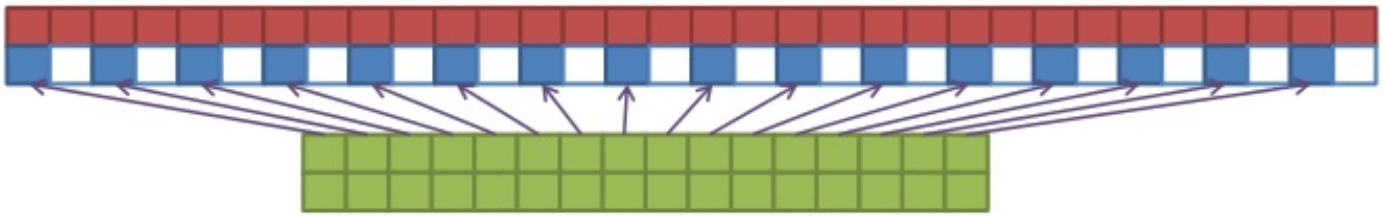


Every thread accesses memory within same 128B-aligned memory segment, so the hardware will coalesce into 1 transaction.

Pristup k pamati

Will This Coalesce?

Stride-2, half warp:

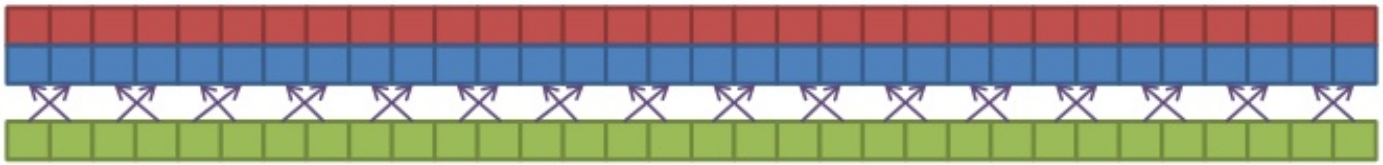


Yes, but..

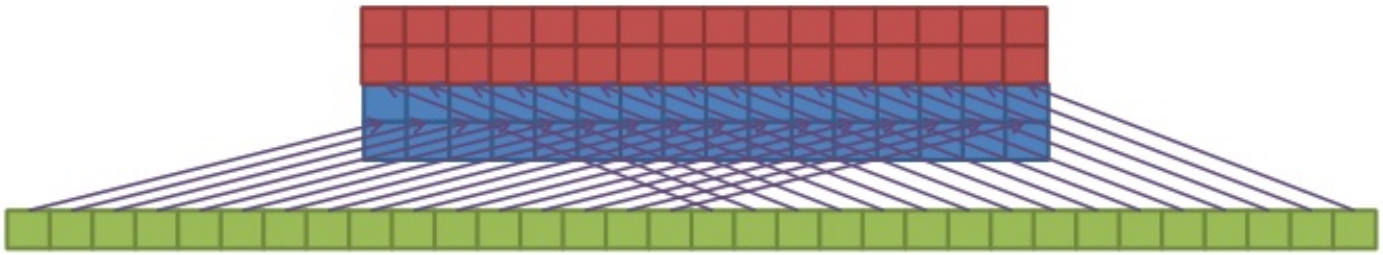
- Half of the memory transaction is wasted.
- Poor utilization of the memory bus.

Pristup k pamati

Will This Coalesce?



Yes! Every thread is still accessing memory within a single 128B segment and segment is 128B aligned.



No. Although this is stride-1, it is misaligned, accessing 2 128B segments. 2 64B transactions will result.

Zdieľaná pamäť

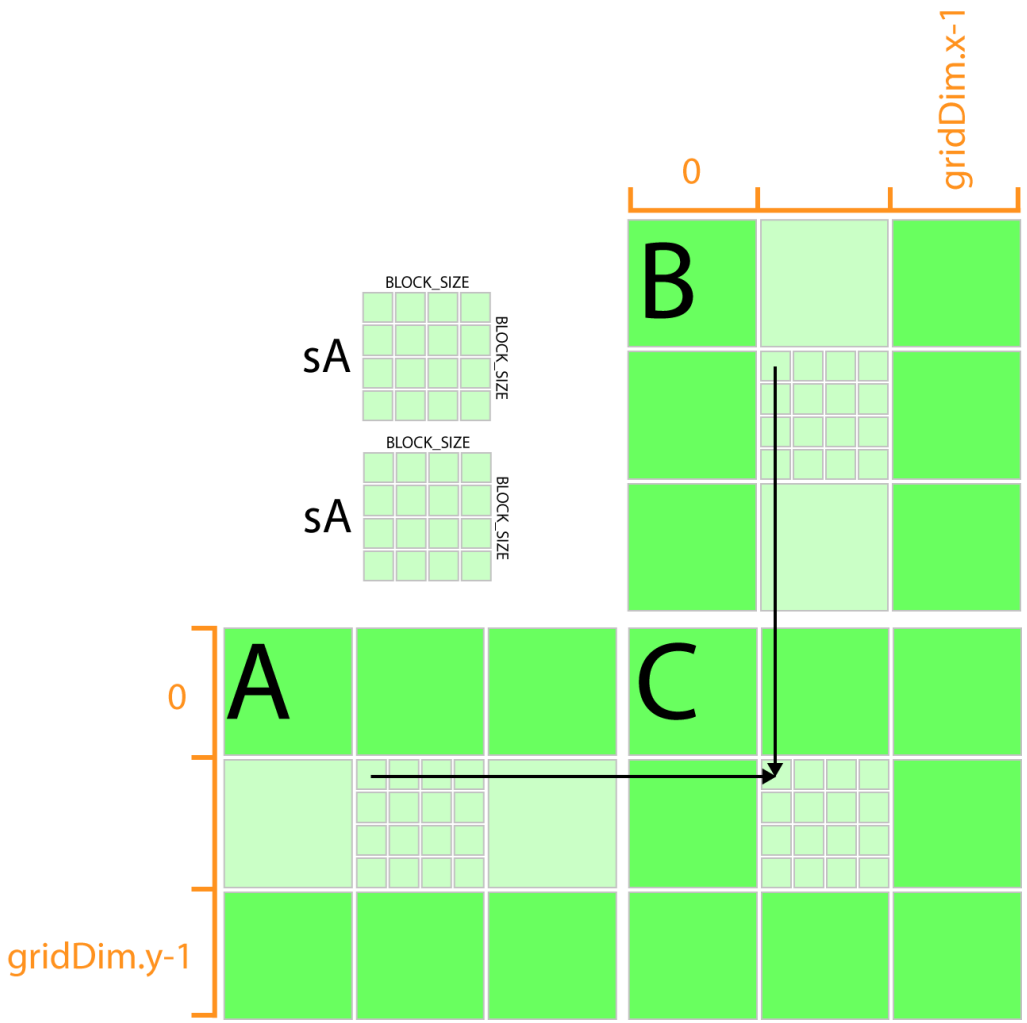
Kazdy SM ma okrem registrov aj obmedzenu (48kB) pamat, zdielanu threadmi v bloku.
Staticka velkost v kode @compile time:

- __shared__ float buff[1024];

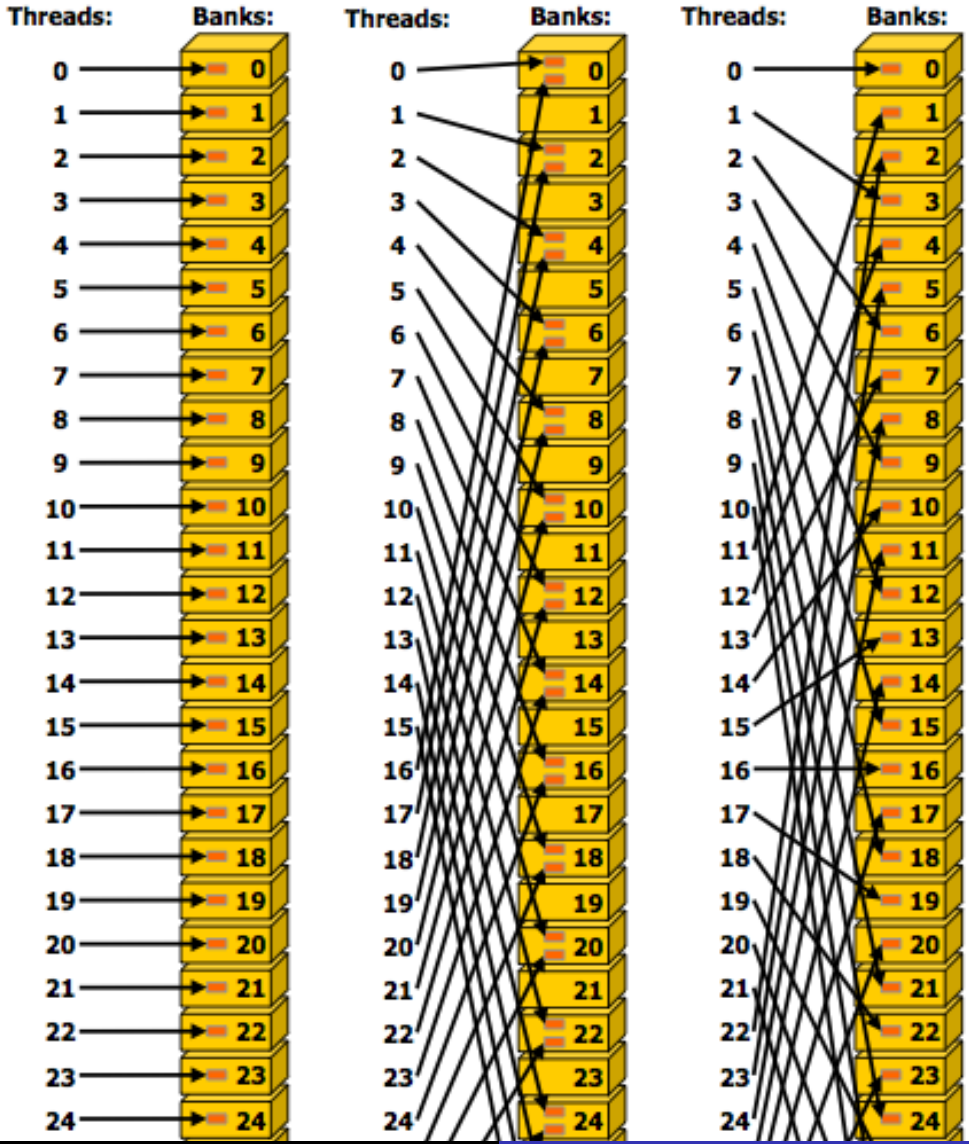
Dynamicka velkost @runtime:

- extern __shared__ float buff[];
- kernel<<<gdim,bdim,1024*sizeof(float)>>>

Nasobenie matic - dlazdicky



Caveats - Bank conflicts (32-bit/cycle x 32banks)



Pri konflikte pamatovych baniek (u roznych 32-bit slovach) dochadza k sekvencnemu pristupu