Lecture 17

Computing with Graphical Processing Units

Today's lecture

- MPI wrap up
  - Sorting
  - Processor Geometries
- Computing with GPUs

Announcements

Bucket sort

- Divide key space into equal subranges and associate a bucket with each subrange
- Each processor maintains p local buckets
  - Assigns each key to a local bucket (local bucket ~ $\mathbf{n/p^2}$ elements)
  - Routes the buckets to the correct owner
    Sorts all incoming data into a single bucket
- Poor worst case behavior for non-uniform key distribution

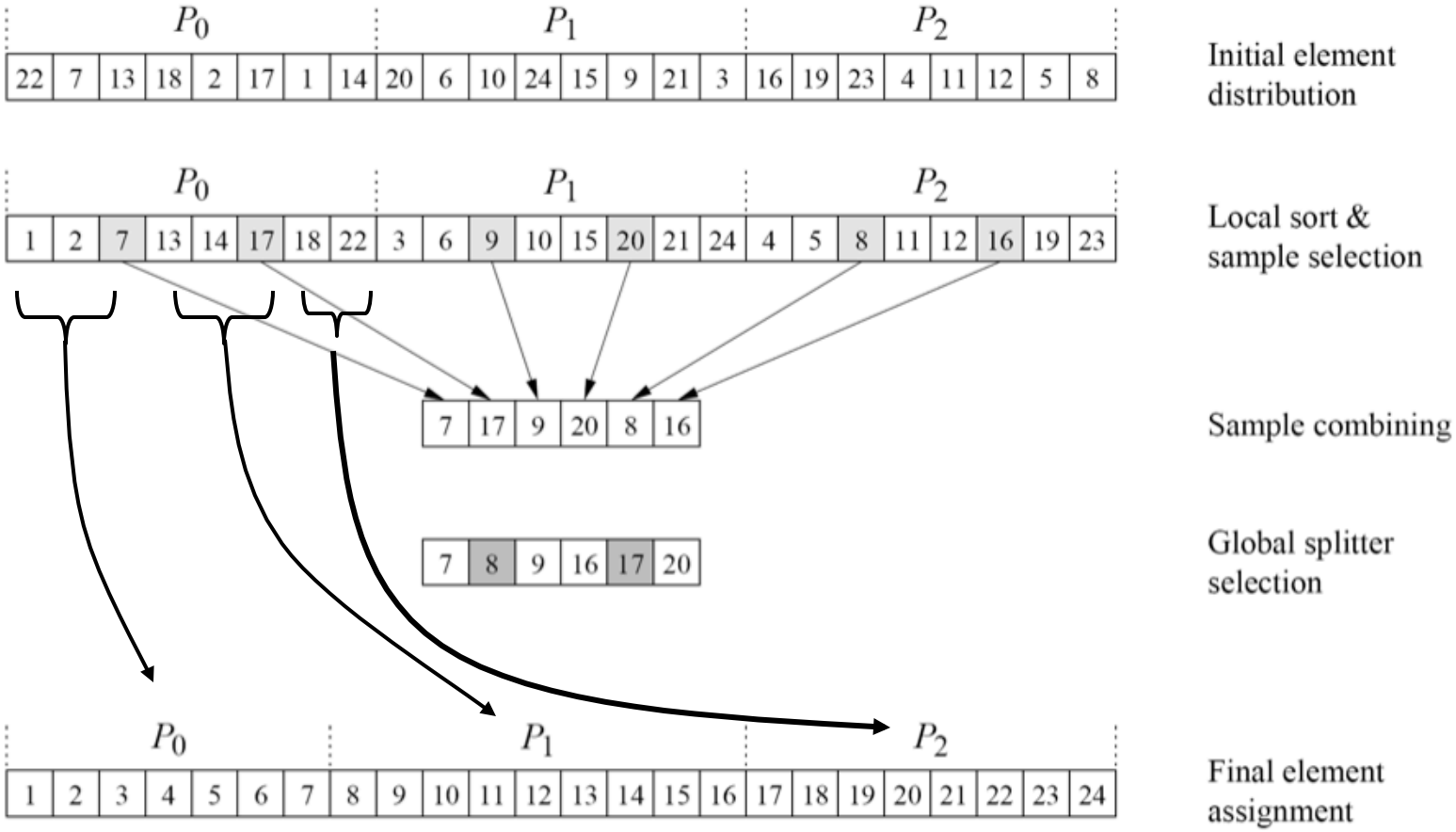


Wikipedia

Sample sort

- Overcomes poor worst case behavior
- Estimates distribution of the global key range
- Distributes key space non-uniformly over processors
- Samples keys to determine a set of *p-1* ***splitters*** that partition the key space into *p* disjoint intervals
- Once each processor knows the splitters, it can distribute its keys to the others accordingly

Alltoallv used in sample sort



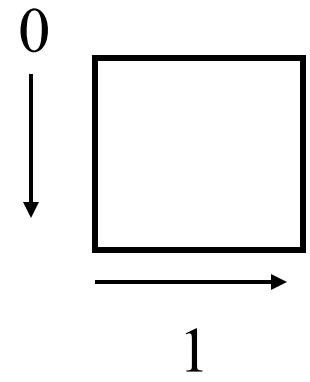
Introduction to Parallel Computing, 2nd Ed,, A.Grama, A.l Gupta, G. Karypis, and V. Kumar, Addison-Wesley, 2003.

Performance

- Assuming $n \geq p^3$ …
- $\mathbf{T_P} = \boldsymbol{O((n/p)\ lg\ n)}$
- If $\boldsymbol{s = p}$, each processor will will merge not more than
  $\boldsymbol{2n/p + n/s - p}$ elements
- If s > p, each processor will will merge not more than
  $\boldsymbol{(3/2)(n/p) - (n/(ps)) + 1 + d}$ elements
- Duplicates ***d*** do not impact performance unless $\boldsymbol{d = O(n/p)}$
- Tradeoff: increasing ***s*** …
  - Spreads the final distribution more evenly over the processors
  - Increases the cost of determining the splitters

Performance is sensitive to processor geometry (Aliev-Panfilov on Triton.sdsc.edu)

	Total			Without communication		
Geometry	GF/s	Time	Comm	GF/s	Time	
4 x 4	38.5	5.02	0.370	41.6	4.66	7.2% Communication
2 x 8	38.4	5.04	0.410	41.8	4.63	
8 x 2	38.1	5.07	0.376	41.2	4.70	
1 x 16	34.6	5.59	0.965	41.8	4.62	
16 x 1	33.0	5.87	1.00	39.8	4.87	
4x16	150.4	1.35	0.13	166.6	1.22	9.6% Communication
16x4	147.3	1.38	0.14	164.4	1.24	
8x8	146	1.4	0.17	165.9	1.23	
2x32	133.9	1.52	0.28	164.6	1.24	
1x64	130.9	1.56	0.32	164.2	1.24	
32x2	126.2	1.61	0.33	159.2	1.28	
64x1	119	1.71	0.33	147.8	1.38	

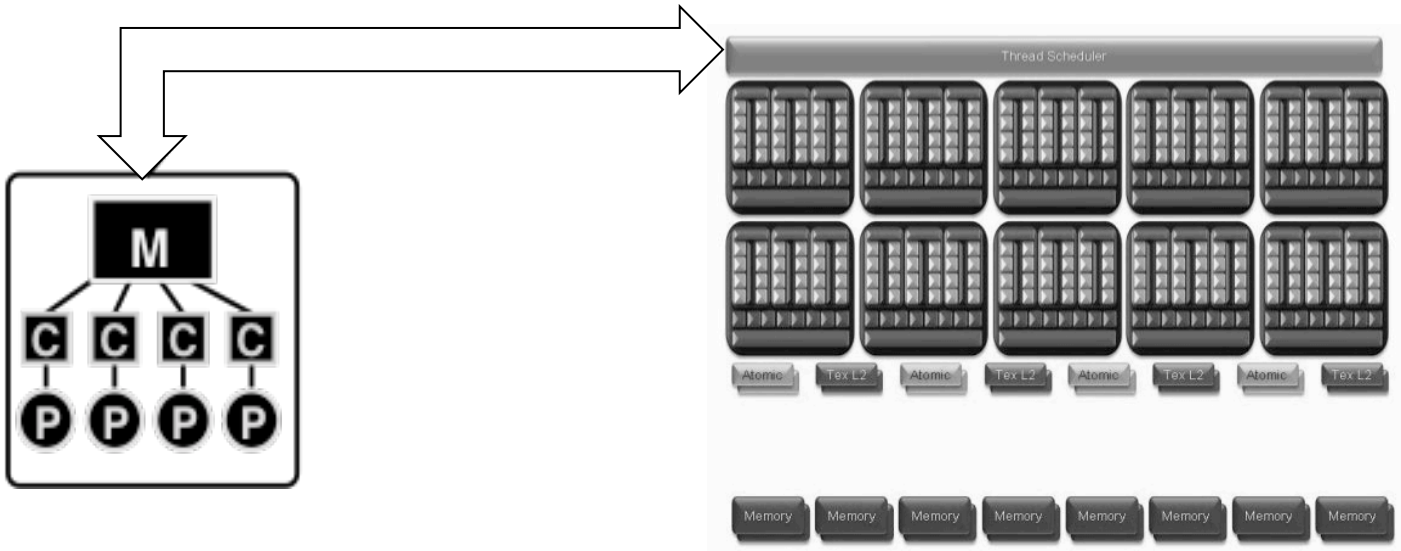


Computing with Graphical Processing Units (GPUs)

©2010 Scott B. Baden / CSE 160 / Winter 2010

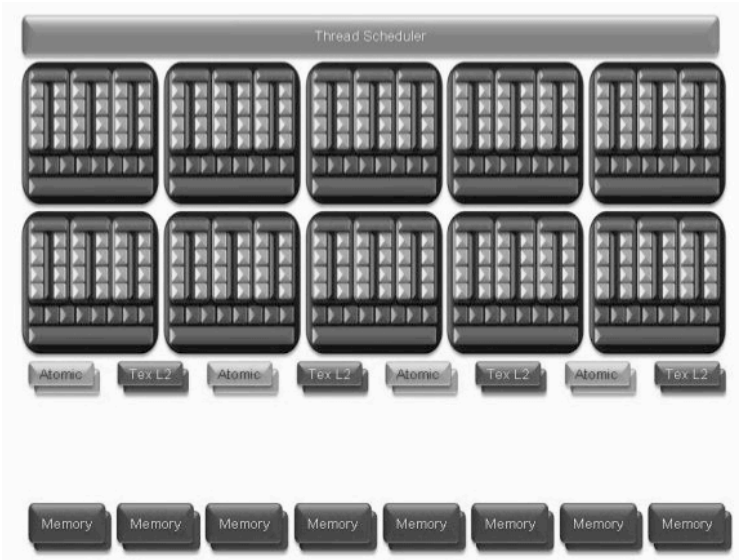
Heterogeneous processing

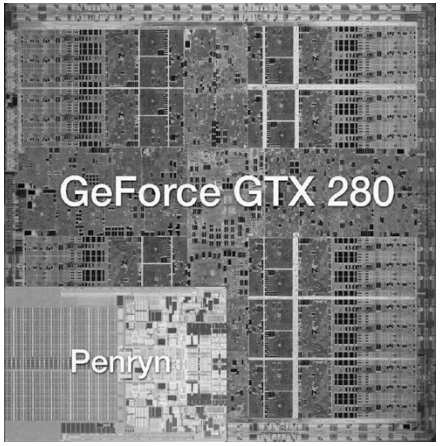
- Two types of processors: general purpose + accelerator
  - "host" and "device"
- Accelerator can perform certain tasks more quickly subject to various overhead costs



NVIDIA GeForce GTX 280

- Hierarchically organized clusters of streaming multiprocessors
  - 240 cores @ 1.296 GHz
  - Peak performance 933.12 Gflops/s
- SIMT parallelism
- 1 GB device memory (frame buffer)
- 512 bit memory interface @ 132 GB/s





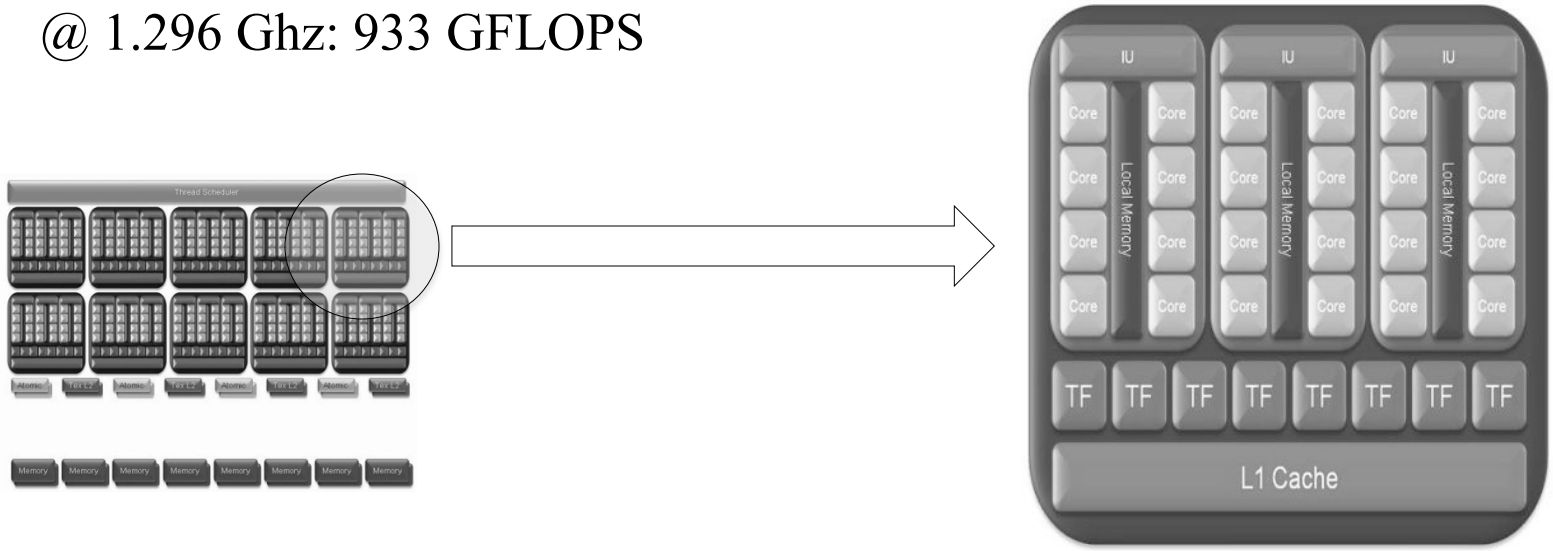
GTX 280: 1.4B transistors

Intel Penryn: 410M (dual core)

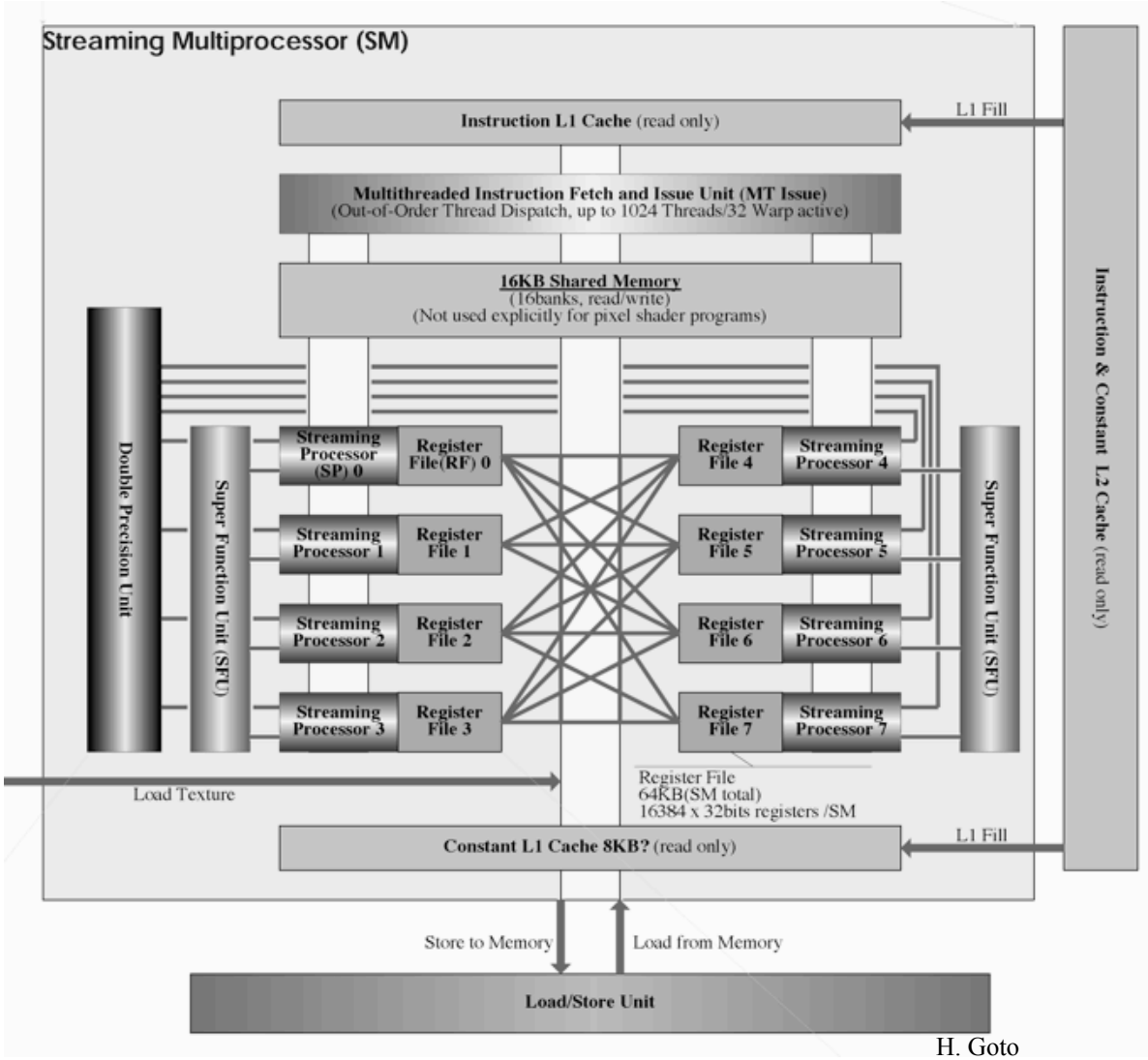
©2010 Scott B. Baden / CSE 160 / Winter 2010

Streaming processing cluster

- GTX-280 GPU
  10 clusters @ 3 streaming multiprocessors or *vector cores*
- Each vector core
  - 8 scalar cores: fused multiply adder + multiplier (FMA, 32 bits), truncate intermediate result
  - May complete 1 FMA + 1 multiply per cycle = 3 flops / cycle (SP)
  - 64- bit FMA + 2 super function units (2 FMAs)
  - Share local memory (16KB) and partition the registers (64KB)
- 3 flops/core/cycle * 240 cores = 720 flops/cycle
- @ 1.296 Ghz: 933 GFLOPS



Streaming Multiprocessor

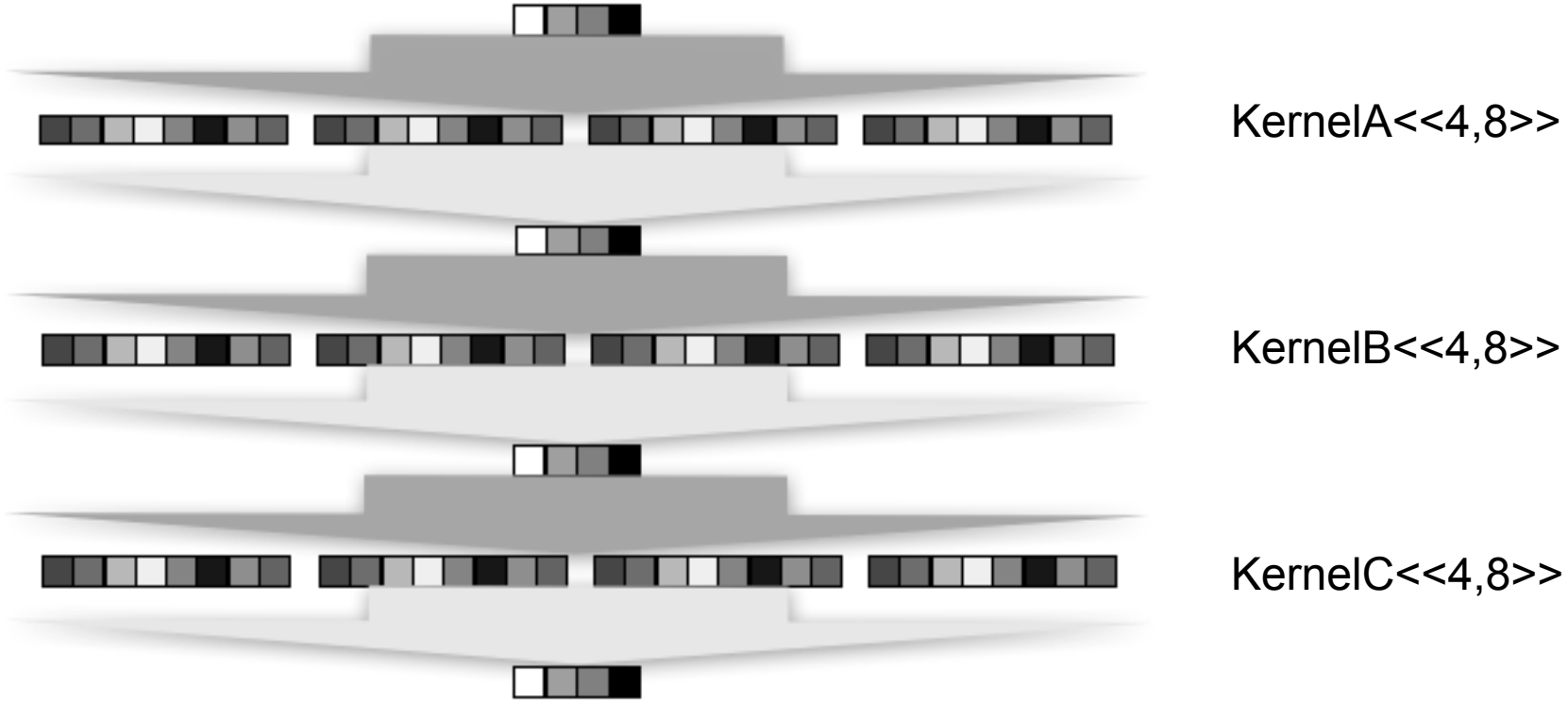


H. Goto

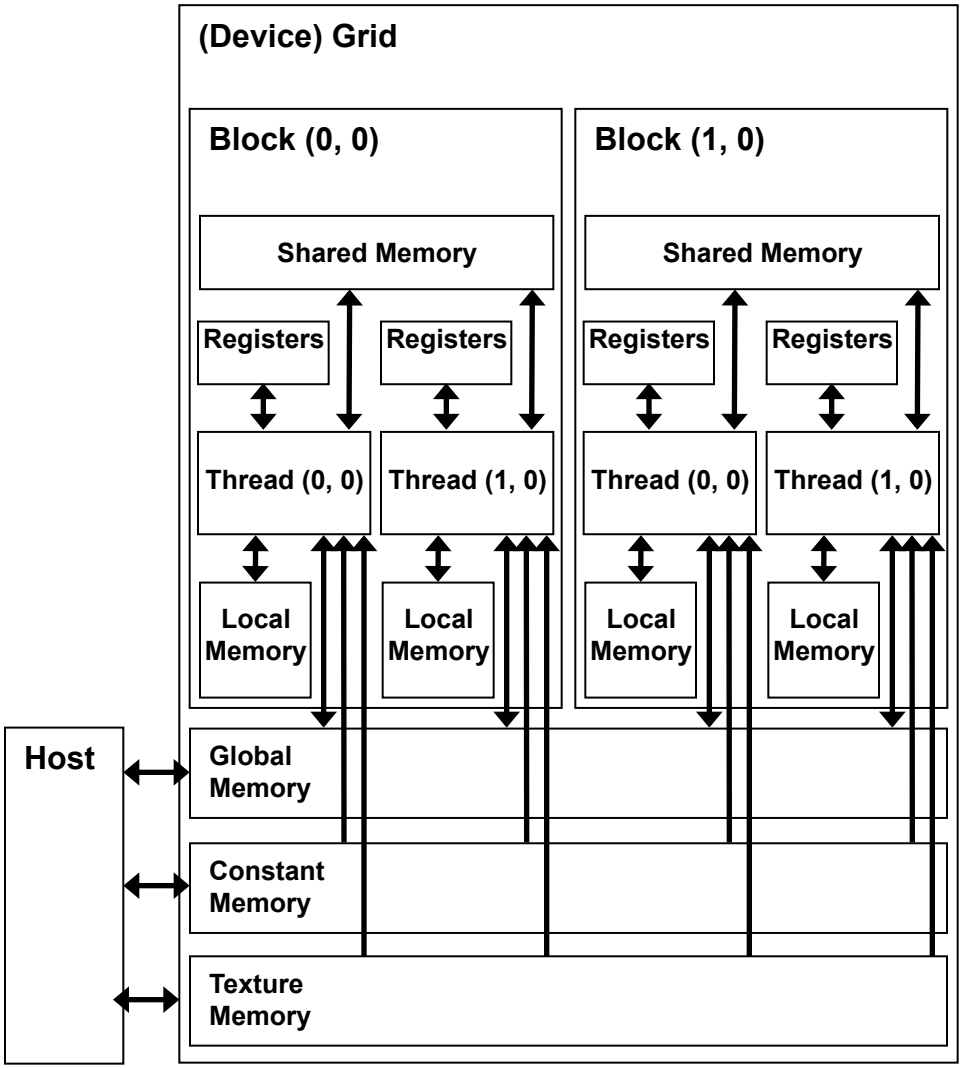
©2010 Scott B. Baden / CSE 160 / Winter 2010

Accelerator model

- Under control of the *host*, invoke sequences of multithreaded kernels on the *device* (GPU)
- Many lightweight threads
- CUDA: programming environment + C extensions



Memory Hierarchy

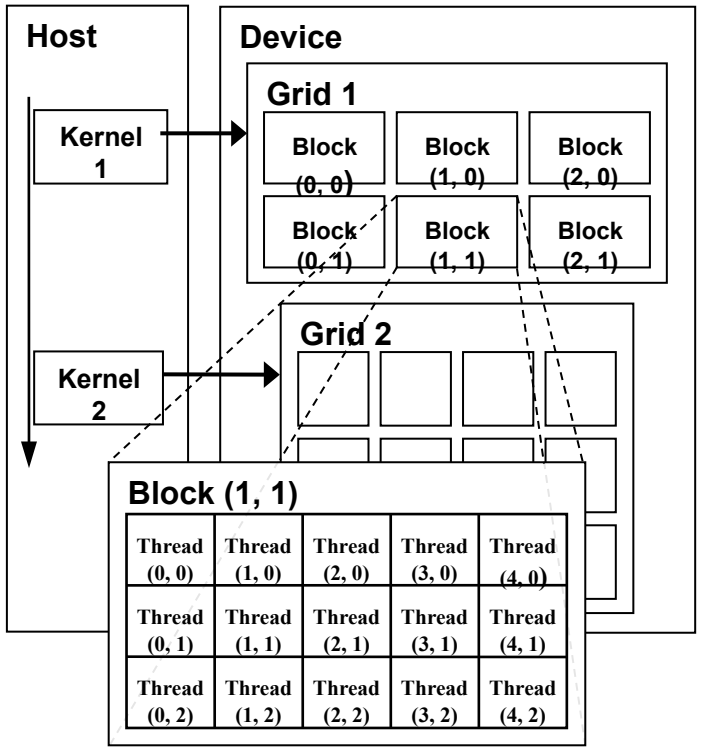


Name	Latency (cycles)	Cached
Global	DRAM – **100s**	No
Local	DRAM – 100s	No
Constant	1s – 10s – 100s	Yes
Texture	1s – 10s – 100s	Yes
Shared	1	--
Register	1	--

Courtesy DavidKirk/NVIDIA and Wen-mei Hwu/UIUC

Thread execution model

- Kernel call spawns virtualized, hierarchically organized threads
  - Grid ⊃ Block ⊃ Thread
- Thread Blocks
  - Subdivide a global index domain
  - Cooperate, synchronize, with access fast on-chip shared memory
  - Threads in different blocks communicate only through slow global memory
- Threads
  - Assigned to SM in units of blocks
  - Compiler re-arranges loads to hide latencies
- Global synchronization via kernel invocation

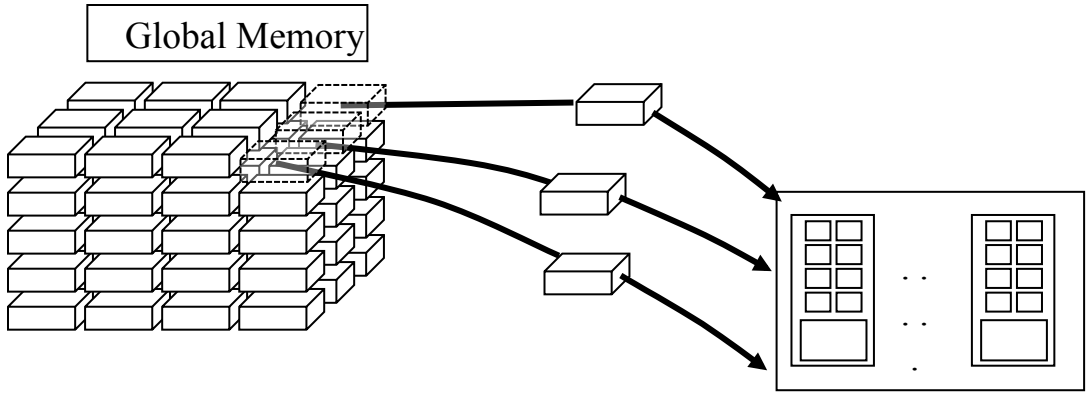


KernelA<<<2,3>,<3,5>>>()
Grid Block

DavidKirk/NVIDIA & Wen-mei Hwu/UIUC

Threads, blocks, grids and data

- Threads all execute same instruction (SIMT)
- A block may have a different number of dimensions (1d, 2d or 3d) than a grid (1d/2d)
- Each thread uniquely specified by block & thread ID
- Programmer determines the mapping of virtual thread IDs to global memory location
  - $\prod: Z^n \rightarrow Z^2 \times Z^3$
  - $\Theta(\Pi_\iota), \forall \Pi_\iota \in \Pi$



Coding example – Increment Array

Serial Code

```
void incrementArrayOnHost(float *a, int N){
  int i;
  for (i=0; i < N; i++) a[i] = a[i]+1.f;
}
```

```
#include <cuda.h>
__global__ void incrementOnDevice(float *a, int N){
   int idx = blockIdx.x*blockDim.x + threadIdx.x;
   if (idx<N) a[idx] = a[idx]+1.f;
}

incrementOnDevice <<< nBlocks, blockSize >>> (a_d, N);
```

Rob Farber, Dr Dobb's Journal

Managing memory

```
float *a_h, *b_h;          // pointers to host memory
float *a_d;                // pointer to device memory

cudaMalloc((void **) &a_d, size);

for (i=0; i<N; i++) a_h[i] = (float)i;  // init host data

cudaMemcpy(a_d, a_h, sizeof(float)*N,
                    cudaMemcpyHostToDevice);

int bSize = 4;
int nBlocks = N/bSize + (N%bSize == 0?0:1);
```

Transferring results + cleanup

```
incrementOnDevice <<< nBlocks, bSize >>> (a_d, N);

// Retrieve result from device and store in b_h
  cudaMemcpy(b_h, a_d, sizeof(float)*N,
                      cudaMemcpyDeviceToHost);
// check results
  for (i=0; i<N; i++) assert(a_h[i] == b_h[i]);

// cleanup
  free(a_h); free(b_h);
  cudaFree(a_d);
```


©2010 Scott B. Baden / CSE 160 / Winter 2010

CUDA language extensions

- Type qualifiers to declare device kernel functions
  __global__ void matrixMul(…)
- Kernel launch syntax
  matrixMul<<< grid, threads >>>(…)
- Keywords
  blockIdx threadIdx
- Runtime, e.g. storage allocation
  cudaMalloc cudaFree cudaMemCpy

Experiments - increment benchmark

- Total time: timing taken from the host, includes copying data to the device
- Device only: time taken on device only

N = 8388480, block size = 128, times in milliseconds

Reps = 10	100	1000	10000	
8.5	83	830	8300	Device time
29	100	850	8300	Kernel launch + data xfer
77	770	7700		Host
16	103			a[i] = 1 + sin(a[i]) - Device)
6900	23.6 sec			Sine function (Host)

Speedup

- How much of an improvement did our parallel algorithm obtain over the serial algorithm?
- S***peedup, S***

$$\frac{\text{Running time of the fastest program on conventional processors}}{\text{Running time on the accelerator}}$$

- Baseline: a multithreaded program

Bandwidth test results

./bandwidthTest

Device 0: Tesla C1060

Host to Device Bandwidth

Transfer Size (Bytes) Bandwidth(MB/s)

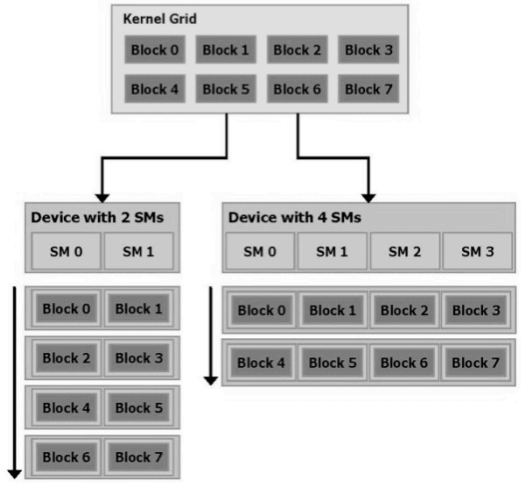
32M 3324.5

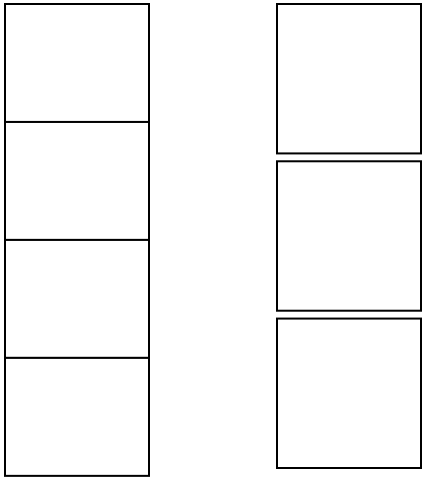
Device to Host Bandwidth

32M 2792.0

Dynamic behavior – resource utilization

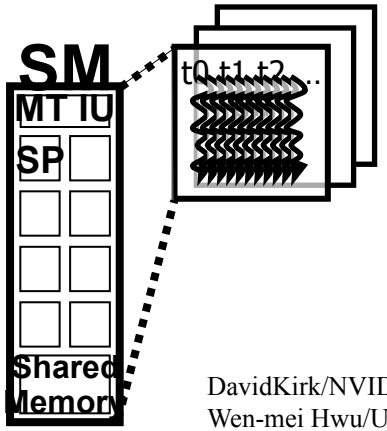
- Each vector core (SM):
  1024 thread slots and 8 block slots
- Hardware partitions slots into blocks at run time, accommodates different processing capacities
- Registers are split dynamically across all blocks assigned to the vector core
- A register is private to a single thread within a single block





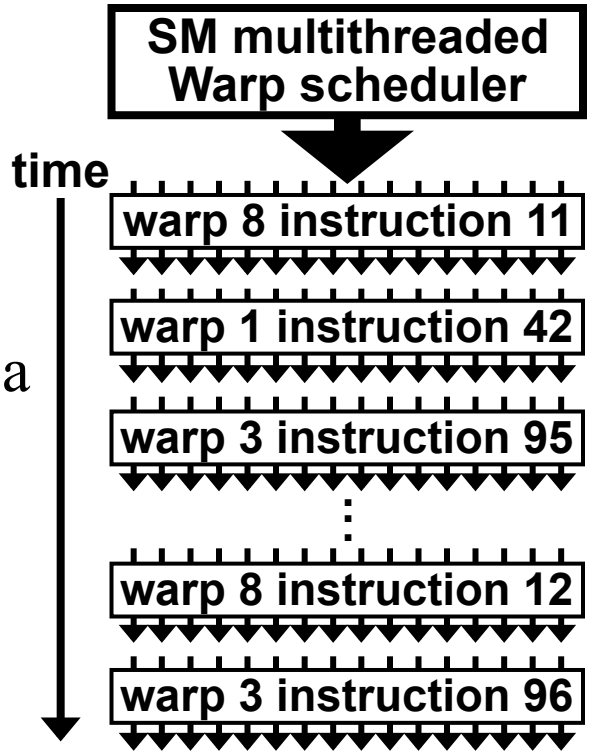
Courtesy DavidKirk/NVIDIA and Wen-mei Hwu/UIUC

Warp scheduling



DavidKirk/NVIDIA &
Wen-mei Hwu/UIUC

- Blocks are divided into *warps* of 32 (SIMD) threads which are ….
- Subidivided into schedulable units: 16 threads (→ 32 on Fermi)
- Warps are scheduled with zero overhead in hardware
- Scheduler finds an eligible warp: all operands are ready
  - Scoreboarding
  - Priorities
- Branches serialize execution within a warp

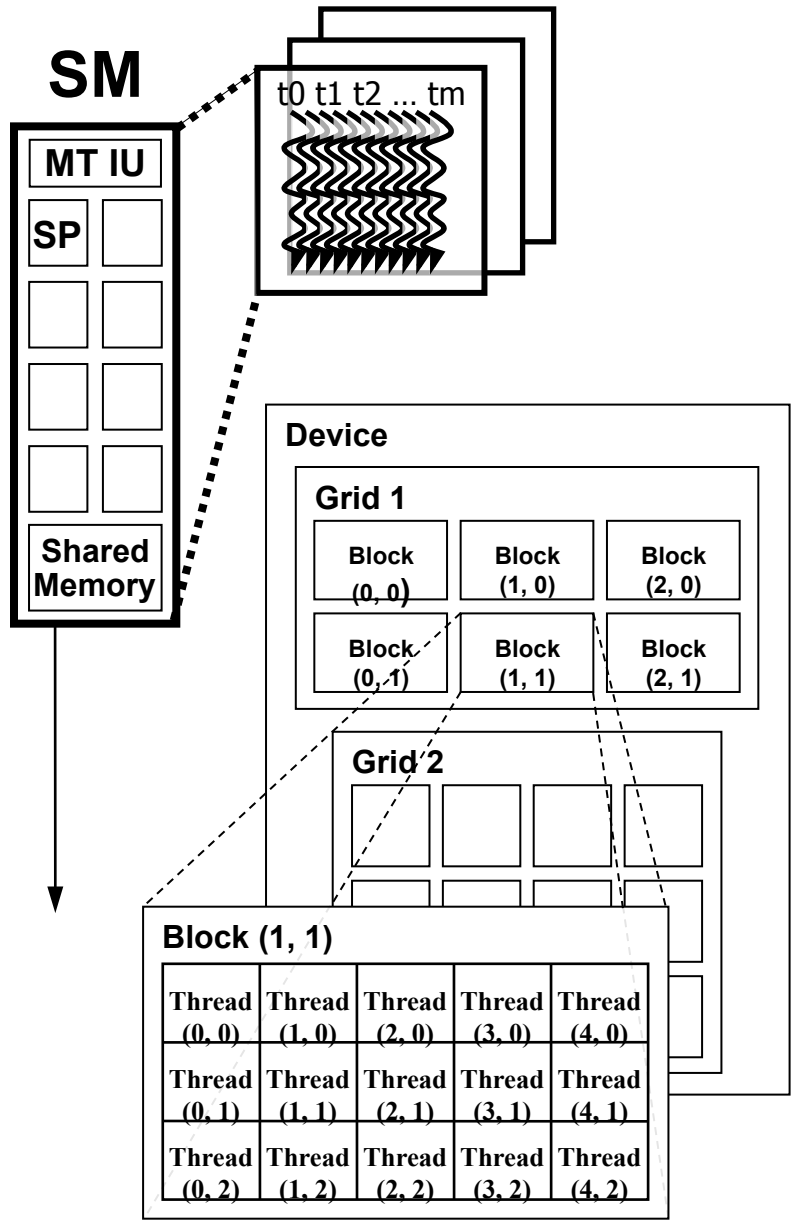


Running times of CUDA instructions

- To execute an instruction for all threads of a warp, the warp scheduler must issue the instruction over
  - 4 clock cycles for an integer or single-precision floating-point arithmetic instruction [throughput = 8]
  - 32 cycles for double-precision [1]
  - 16 cycles for single-precision floating-point transcendental [2]

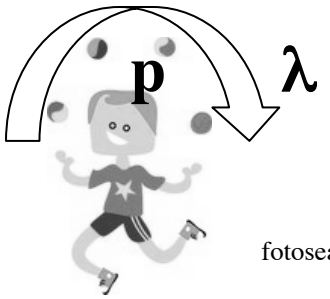
Constraints

- SM
  - Up to 8 resident blocks
  - Not more than 1024 threads
  - Up to 32 warps
- All threads in a warp execute the same instruction
  - All branches followed
  - Instructions disabled
  - Divergence, serialization
- Grid: 1 or 2-dimensional (64K-1)
- Blocks – 1,2, or 3- dimensional
  - ≤ 512 threads
  - Max dimensions: 512 , 512 , 64
  - Registers subdivided over threads
  - Synchronization among all threads in the block
- Synchronization only within block

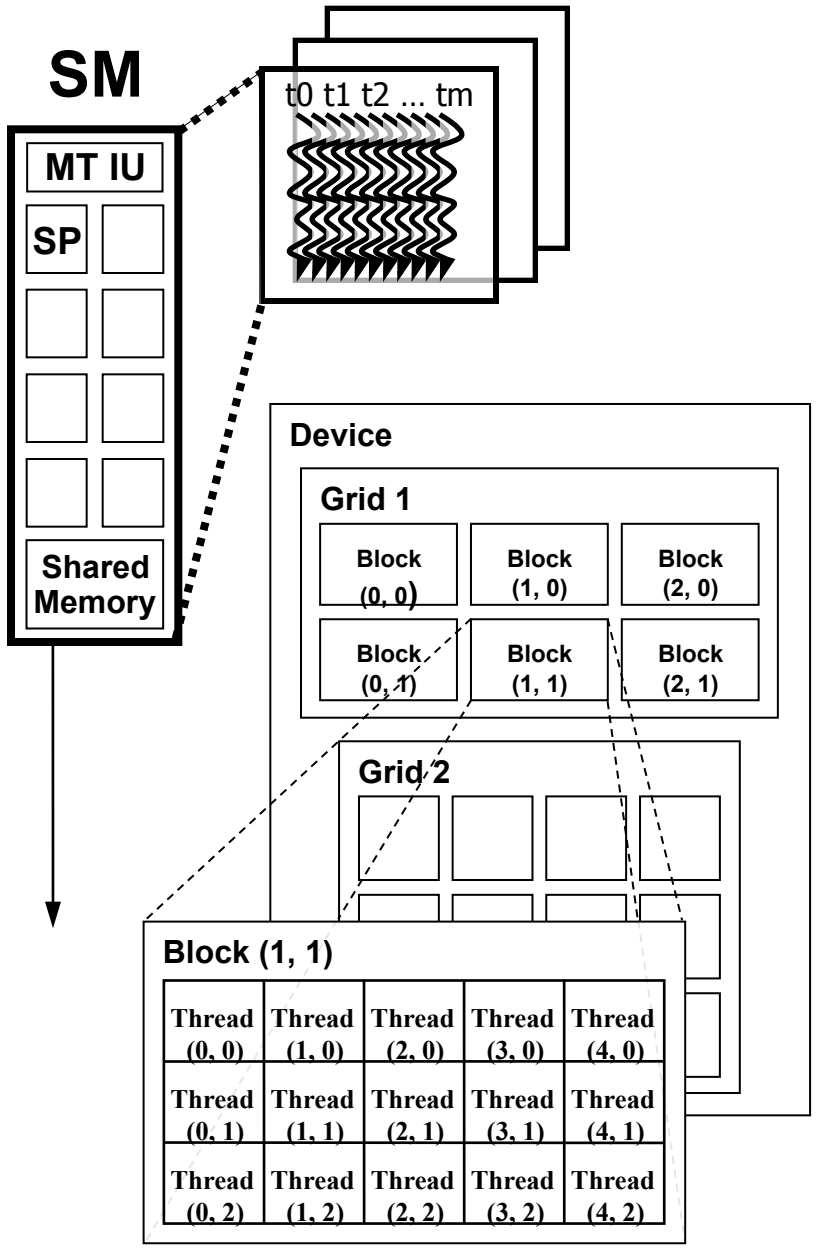


Occupancy

- A minimum number of warps needed to hide memory latency
- Varying the block size
  - 8x8: 64 threads/block, max 8 blocks, only 512 threads/SM
  - 16x16: 3 blocks;
    can we hide the latency?
- Consider an application using 32 registers (4 bytes floats) per thread
- Each SM gets 512 registers, 64 per core, how many threads/SM?



fotosearch.com

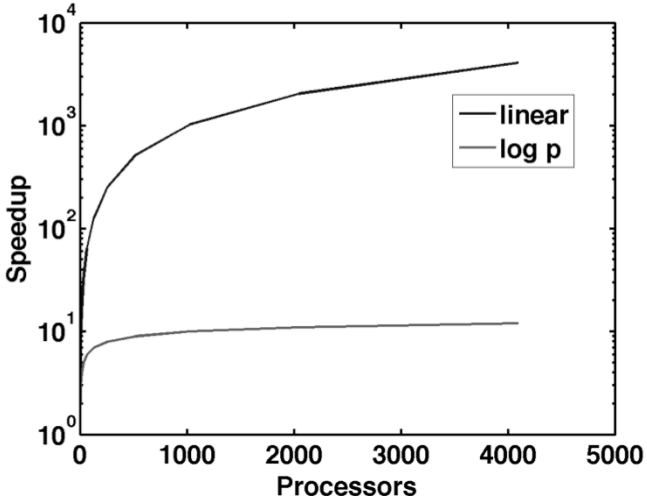


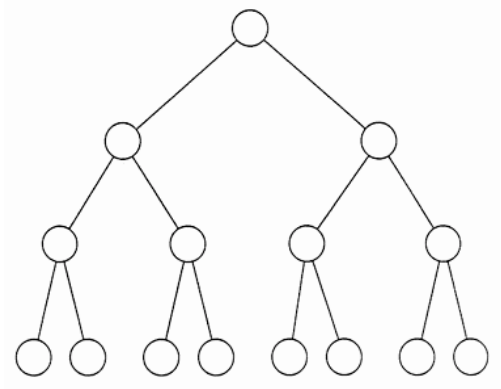
Programming issues

- Branches serialize execution within a warp
- Registers dynamically partitioned across each block in a Streaming Multiprocessor
- Bound to and only accessible from their thread until the block finishes execution
- Shared memory and registers do not persist across kernel invocations
- Tradeoff: more blocks with fewer threads or more threads with fewer blocks
  - Locality: want small blocks of data (and hence more plentiful warps) that fit into fast memory
  - Register consumption
  - Scheduling: hide latency

How to obtain good speedups

- Avoid algorithms that present intrinsic barriers to utilizing the hardware
- Hide latency of host ↔ device memory transfers
- Reduce global memory accesses
  - Global memory accesses → fast on-chip accesses
  - Coalesced memory transfers
- Avoid costly branches, or render them harmless





Is it that simple?

- Simplified processor design, but more user control over the hardware resources
  - Rethink problem solving technique & implementation
  - Redesign the software

- If we don't use the parallelism, we lose it
  - Amdahl's law - serial sections
  - Von Neumann bottleneck – data transfer costs
  - Workload Imbalances