University of Central Florida

ST: CDA 6938: Multi-Core/Many-Core Architecture and Programming

Performance Optimization of GPGPU

Huiyang Zhou

School of Electrical Engineering and Computer Science
University of Central Florida

Performance Optimization

- Goal:
  - Balanced utilization of critical resources, including memory BW, thread-level parallelism, instruction-level parallelism, register usage, shared memory usage, etc.
  - If one resource is under utilized, consider trade off for other resources.
    - E.g., if ALU is under utilized and the TEX ops are the bottleneck, consider load reuse to increase the ratio of #ALU ops per TEX op

A more detailed guideline for memory-bound applications

- 1. Check whether there is **enough** thread-level parallelism, i.e., # of threads that can run in a thread block/cluster, to hide the memory access latency.
  - Memory access latency: 200 cycles
  - Each warp (warp size = 32 threads in G80/ 64 threads in R670) takes 4 cycles.
  - If there are 2 independent instructions before the use of the loaded value, how many threads are necessary to hide the latency?
    - 200 / (2 * 4) = 25 warps => requiring too much thread-level parallelism that is supported in either G80 or R670
    - Solution: Increasing independent operations, including other memory accesses (i.e., Memory-level parallelism).
    - Check the register usage of each thread (N) since it limits how many threads can run in a block (G80: #warps = 8k/N/32) or cluster (R670: #wavefronts = 64k/N/64)
  - Key is "enough" thread-level parallelism (TLP
    - Once there is enough TLP, further increase in number of threads will have limited benefit.

A more detailed guideline for memory-bound applications

- 2. Optimizing the code of the thread, e.g., reducing the instruction count by trading of register usage (data reuse, common-expression elimination, etc) and TLP
- 3. Make best use of fast memory (register file and shared memory)
  - May treat shared memory as an extended register file
    - Limitations on operands: only one operand can be from shared memory; additional instruction may be needed to generate the address offset.
  - Checking whether shared memory usage limits thread-level parallelism.
    - # thread blocks = 16kB / (shared memory per thread * block size)
  - Organize the shared memory accesses to avoid bank conflicts
    - 16 banks
- 4. Reorganize/partition the workload in each thread to satisfy the first three objectives
  - E.g. tile size: 16x16 => 2kB shared memory, 256 threads. Shared memory seems to be under-utilized
  - Tile size 32 x 32 => 1024 threads (too much for a block), use the tile size of 16 x16 and unroll & jam to add more work to each thread.

A more detailed guideline for memory-bound applications

- 5. Checking instruction mix, start the loads as early as possible if it does not result in register spill & refill
- 6. Consider prefetching for the next iteration, check register usage to avoid register spill.
- 7. Taking advantage of hardware features such as the texture cache and constant cache.

General Purpose Computing on Different GPUs: A First Look

	G80 (Geforce 8800)	R670 (AMD Radeon HD 3870)
FLOPS	367 GFLOPS	521 GFLOPS (102 GFLOPS double-precision)
Memory BW	86.4 GB/s Mem 4 GB/s to CPU	64 GB/s
Mem size	768 MB	512MB
# of transistors	686 M	
Freq	Core clock 575 MHz, Shader 1.35 GHz	Core clock 825MHz
Power	80W	

General Purpose Computing on Different GPUs: Thread Execution

	G80 (Geforce 8800)	R670 (AMD Radeon HD 3870)
Thread Hierarchy	16 SMs, 8 SP per SM 4 SMs share 1 TEX subsys	4 clusters, 16 x 5 cores per cluster, each cluster time-multiplex 1 TEX subsys
# threads	768 per SM * 16 SM	64 per wavefront * 256 wave fronts
# active threads in execution	32 (warp size) * 16 Each warp takes 4 cycles to issue	64 (wavefront size) * 4 Each wavefront takes 4 cycles to issue
Instruction-level parallelism	Scalar operation for each thread	5-way VLIW for each thread

General Purpose Computing on Different GPUs: Memory Model

	G80 (Geforce 8800)	R670 (AMD Radeon HD 3870)
Register File	512 kB = 32kB per SM * 16 SM 8K registers per SM 1K register per SP	1MB = 256kB per cluster * 4 cluster 64K registers per cluster 1K register per core (SFU does not have its own register file)
Shared Memory	256 kB = 16kB per SM * 16 SM	N/A
R/W cache	N/A	A small cache
Local/Global/Texture memory	Device Mem size	Device Mem Size
Constant Cache	8KB per SM, 128KB in total	?L1 (no L2)
Constant Memory	64KB in total	64KB(?)
Texture cache	8kB per SM	32kB L1 128kBL2 shared by all clusters

General Purpose Computing on Different GPUs: Programming Support

	G80 (Geforce 8800) CUDA	R670 (AMD Radeon HD 3870) Brook+/CAL
Programming model	SPMD	SPMD
High-level Programming Language	C/C++	C
IL	PTX	AMD/ATI IL
Assembly-level analysis	Decuda	ShaderAnalyzer/CTM
Thread management	Thread hierarchy	Streaming model