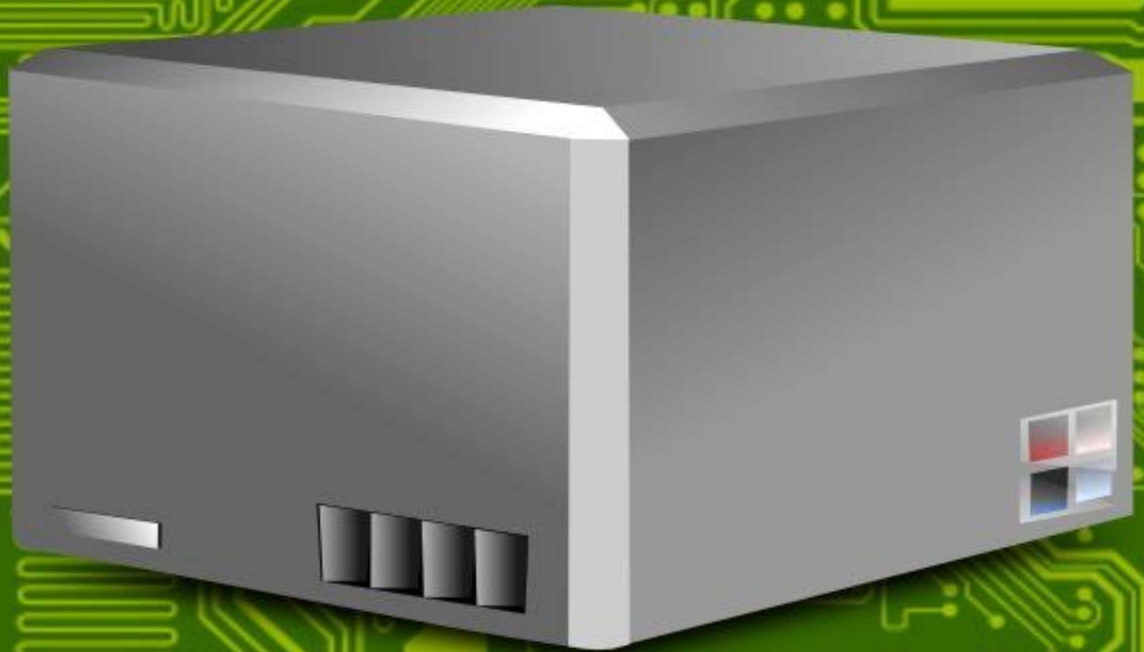
GPU Computing



CUDA 程序优化

Hui Liu
Email: hui.sc.liu@gmail.com

优化 GPU 指令

基本策略

- Minimize use of low-throughput instructions
- Use high precision only where necessary
- Minimize divergent warps

算术指令

- `float` add/mul/mad, `int` add, shift, min, max: 4 cycles per warp
  - `int` multiply (*) is by default 32-bit
    - Requires multiple cycles per warp
    - Use `__[u]mul24()` intrinsics for 4-cycle 24-bit `int` multiply

- Integer divide and modulo are more expensive
  - Compiler will convert literal power-of-2 divides to shifts
    - But we have seen it miss some cases
  - Be explicit in cases where compiler can't tell that divisor is a power of 2!
  - Useful trick: `foo % n == foo & (n-1)` if `n` is a power of 2

算术指令

- Reciprocal, reciprocal square root, sin/cos, log, exp: 16 cycles per warp
  - These are the versions prefixed with “_______”
  - Examples:_____`rcp()`, `sin()`, `exp()`

- Other functions are combinations of the above:
  - `y/x==rcp(x)*y` takes 20 cycles per warp
  - `sqrt(x)==rcp(rsqrt(x))` takes 32 cycles per warp

数学运行时库

- There are two types of runtime math operations:
  - `__func()`: direct mapping to hardware ISA
    - Fast but lower accuracy (see prog. guide for details)
    - Examples: `__sin(x)`, `__exp(x)`, `__pow(x,y)`
  - `func()`: compile to multiple instructions
    - Slower but higher accuracy (5 ulp or less)
    - Examples: `sin(x)`, `exp(x)`, `pow(x,y)`
- The -use_fast_math compiler option forces every **`func()`** to compile to **`__func()`**

控制流

- Main performance concern with branching is *divergence*
  - Threads within a single warp take different paths
  - Different execution paths must be serialized
- Avoid divergence when branch condition is a function of thread ID
  - Example with divergence:
    - `If (threadIdx.x > 2) { }`
    - Branch granularity < warp size
  - Example without divergence:
    - `If (threadIdx.x / WARP_SIZE > 2) { }`
    - Branch granularity is a whole multiple of warp size

结论

- CUDA with latest GPUs can achieve great results on data-parallel computations if you use a few simple performance optimization strategies:
  - Structure your application and select execution configurations to maximize exploitation of the GPU's parallel capabilities,
  - Minimize CPU $\leftrightarrow$ GPU data transfers
  - Coalesce global memory accesses
  - Take advantage of shared memory
  - Avoid shared memory accesses with high degree of bank conflicts
  - Minimize use of low-throughput instructions
  - Minimize divergent warps

THANK YOU