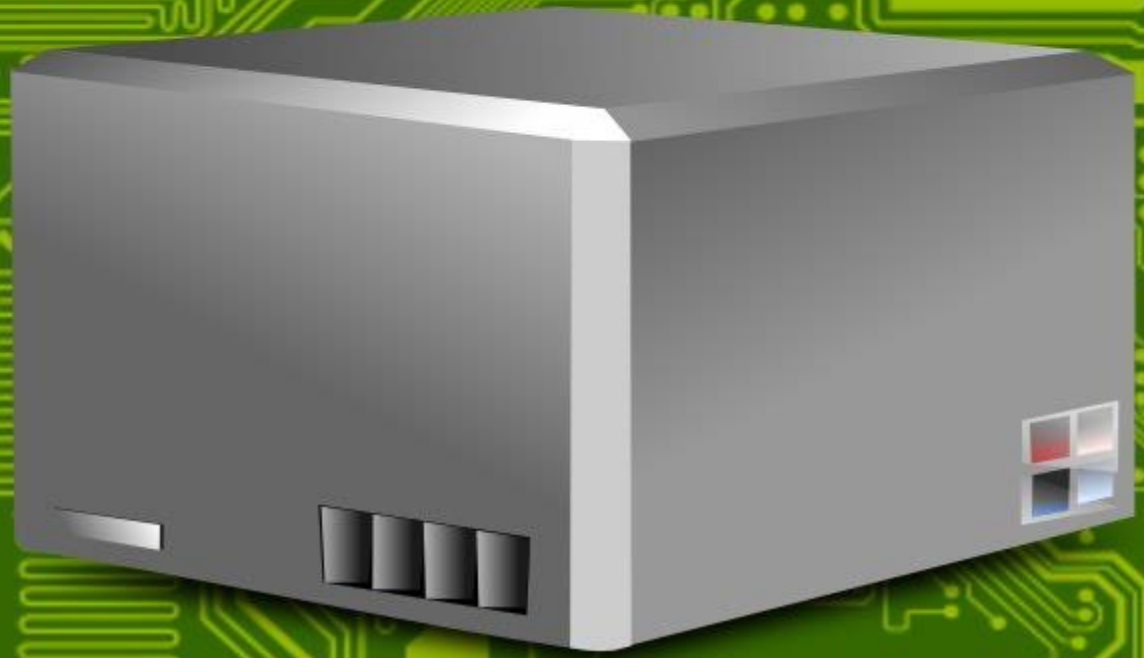
GPU Computing



CUDA 程序优化

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

最大化并行执行

探索并行化

- GPU thread parallelism:
  - 消除依赖: Structure algorithm to maximize independent parallelism
  - 块内通信: If threads of same block need to communicate, use shared memory and syncthreads()
  - 块间通信: If threads of different blocks need to communicate, use global memory and split computation into multiple kernels
    - No synchronization mechanism between blocks
  - 必要性: High parallelism is especially important to hide memory latency by overlapping memory accesses with computation

探索并行化

- CPU/GPU parallelism:
  - 异步启动: Take advantage of asynchronous kernel launches by overlapping CPU computations with kernel execution
  - 异步传输: Coming soon: Asynchronous CPU ↔ GPU memory transfer that overlaps with kernel execution

优化线程块的规模

- (# of blocks) /(# of multiprocessors) > 1
  - So all multiprocessors have at least a block to execute
- Per-block resources (shared memory and registers) at most half of total available
- And: (# of blocks) /(# of multiprocessors) > 2
  - So multiple blocks run concurrently on a multiprocessor
  - If multiple blocks coexist that aren't all waiting at a __syncthreads(), the multiprocessor can stay busy
- (# of blocks) > 100 to scale to future devices
  - Blocks stream through machine in pipeline fashion
  - 1000 blocks per grid will scale across multiple generations

优化线程块的大小

- Choose # of threads per block as a multiple of warp size
  - Avoid wasting computation on under-populated warps
- More threads per block == better memory latency hiding
- But, more threads per block == fewer registers per thread
  - Kernel invocations can fail if too many registers are used

- Heuristics
  - Minimum: 64 threads per block
    - Only if multiple concurrent blocks
  - 192 or 256 threads a better choice
    - Usually still enough registers to compile and invoke successfully
  - This all depends on your computation!
    - Experiment!

最大化 Occupancy

- Given total # of threads in a grid
  - Choose # of blocks and # of threads per block to maximize *occupancy*:

    $$\frac{\text{\# of warps running concurrently on a multiprocessor}}{\text{maximum \# of warps that can run concurrently on a multiprocessor}}$$

- Increasing occupancy does not necessarily increase performance
- But, low-occupancy multiprocessors cannot adequately hide latency on memory-bound kernels

程序参数化

- Parameterization helps adaptation to different GPUs
- GPUs vary in many ways
  - # of multiprocessors
  - Shared memory size
  - Register file size
  - Threads per block
  - Memory bandwidth
- You can even make apps self-tuning (like FFTW)
  - “Experiment” mode discovers and saves optimal

THANK YOU