并行与分布式计算基础：第二十一讲

杨超
chao_yang@pku.edu.cn

2019 秋

内容提纲

1. CUDA 编程-5
   - 补遗

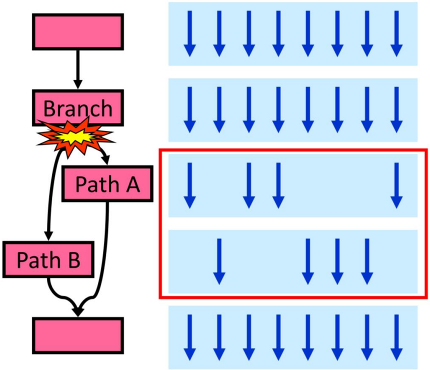
补遗

1. CUDA 编程-5
    - 补遗

线程簇分歧

- 回顾：线程块将按照线程簇 (一般为 32 个线程) 为单元在 SM 上调度，同一线程簇中所有线程采用 SIMD (或称 SIMT) 方式执行.
- 线程簇分歧 (warp divergence)：当同一线程簇中线程执行不同程序路径时，会触发串行执行，导致程序性能下降.

```
if (...) {
    // Path A
} else {
    // Path B
}
```



例如，在向量加法中的边界检查:

```
if (i < n)d_C[i] = d_A[i] + d_B[i];
```

- 向量长度为 100 时，总共 4 个 warp，其中有 1 个 warp 产生分歧，占 25%.
- 向量长度为 1000 时，总共 32 个 warp，其中有 1 个 warp 产生分歧，占 3%.

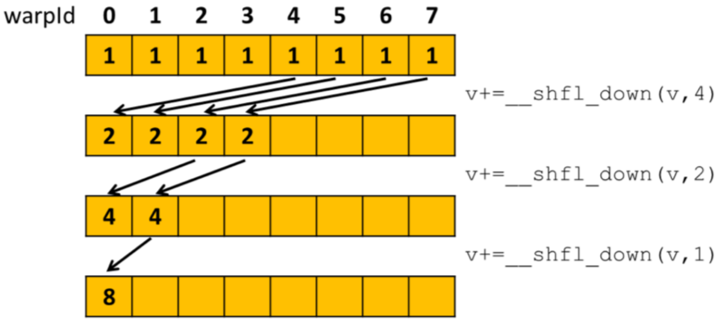
可以看出，对于边界检查，一般来说随着规模增加，分歧的影响会降低.

常见分歧的处理策略

- 线程分支：比如，在代码`if (threadIdx.x > 2)...` 中，线程 0，1，2 和线程 3-31 执行路径不同，程序分支尽量以线程簇大小作为粒度，设法改为`if (threadIdx.x / WARP_SIZE < 2)...`;
- 边界检查：例如向量加法中`if (i < n)d_C[i] = d_A[i] + d_B[i];`，如果开销太大，可以考虑使用两个 kernel，一个处理边界内的计算，一个处理边界情况.
- 线程改变：一些并行算法如 reduction 等，随着时间推移，参与的线程数目发生改变，可以通过设计新的并行算法来减少线程簇分歧.

线程簇混洗

- 线程簇混洗 (warp shuffle)：允许某线程直接读取同一个 warp 中其它线程寄存器中的值，延迟低且不占用额外的存储资源.



原子操作

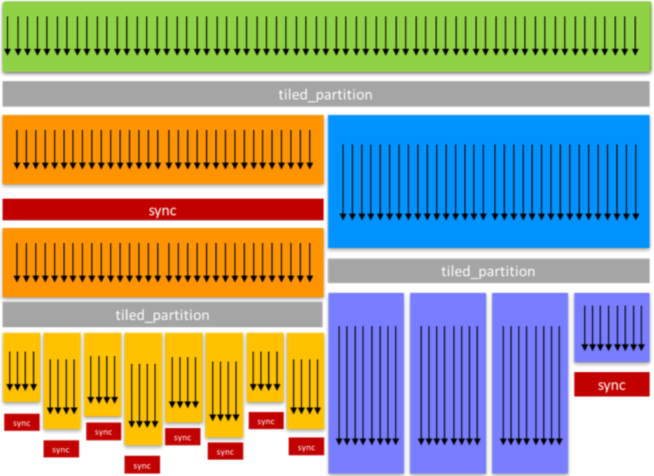
- 原子操作 (atomic operations)：当所有线程同时修改某个全局变量时，需要加锁后访问，保证结果的正确性，常见的原子操作有：

```
atomicAdd, atomicSub, atomicMin, atomicMax,
atomicInc, atomicDec, atomicExch, atomicCAS,
atomicAnd, atomicOr, atomicXor
```

- 线程簇聚合 (warp aggregation)：多个线程原子累加到单个计数器以提高性能，线程簇中的线程首先计算它们之间的总增量，然后选择单个线程将增量原子地添加到全局计数器中；
- CUDA9.0 以上的 NVCC 编译器已官方支持自动的为原子操作执行线程簇聚合.

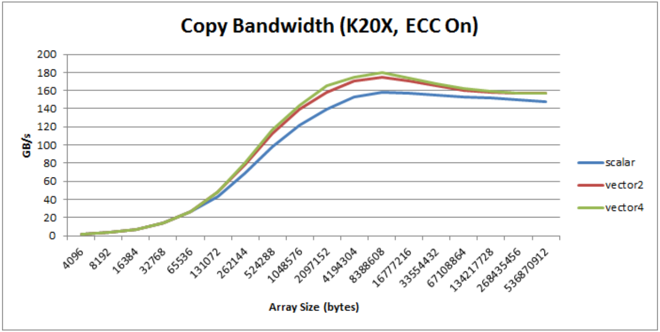
协同分组

- 协同分组 (cooperative groups)：CUDA9.0 以上，支持更为灵活的线程组合方式，从而可以在不同粒度上进行线程间协作.



向量化访存

- 向量化访存 (vectorized memory access)：对于访存受限的问题，在保证数据对其的前提下，通过使用例如 `float2` 等向量化的数据类型，并结合 `reinterpret_cast` 对指针进行强制转换，可以帮助编译器实现访存的向量化，从而提升性能.



NVIDIA GPU 的计算能力 (compute capability)

- 计算能力用于反映 CUDA 设备所支持的不断更迭的功能和特性；
- 计算能力以 `X.Y` 表示，两个数字分别为主版本和从版本号；
- 计算能力的版本之间向后兼容，越新表示设备的功能越强大；

- 不同计算能力、不同产品线的设备构成了庞大的生态体系.

GPU Computing Applications

Libraries and Middleware

- cuDNN TensorRT
- cuFFT, cuBLAS, cuRAND, cuSPARSE
- CULA MAGMA
- Thrust NPP
- VSIPL, SVM, OpenCurrent
- PhysX, OptiX, iRay
- MATLAB Mathematica

Programming Languages

- C
- C++
- Fortran
- Java, Python, Wrappers
- DirectCompute
- Directives (e.g., OpenACC)

CUDA-enabled NVIDIA GPUs

	EMBEDDED	CONSUMER DESKTOP, LAPTOP	PROFESSIONAL WORKSTATION	DATA CENTER
Turing Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier	GeForce 2000 Series	Quadro RTX Series	Tesla T Series
Volta Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier			Tesla V Series
Pascal Architecture (Compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
Maxwell Architecture (Compute capabilities 5.x)	Tegra X1	GeForce 900 Series	Quadro M Series	Tesla M Series
Kepler Architecture (Compute capabilities 3.x)	Tegra K1	GeForce 700 Series GeForce 600 Series	Quadro K Series	Tesla K Series

- CUDA 提供了 `deviceQuery` 样例程序用于检查设备的计算能力.

```
$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro GV100"
  CUDA Driver Version / Runtime Version          10.1 / 10.1
  CUDA Capability Major/Minor version number:    7.0
  Total amount of global memory:                 32508 MBytes (34087305216 bytes)
  (80) Multiprocessors, ( 64) CUDA Cores/MP:     5120 CUDA Cores
  GPU Max Clock rate:                            1627 MHz (1.63 GHz)
  Memory Clock rate:                             850 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 6291456 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 4 copy engine(s)
```

Feature Support	Compute Capability					
(Unlisted features are supported for all compute capabilities)	**3.0**	**3.2**	**3.5, 3.7, 5.0, 5.2**	**5.3**	**6.x**	**7.x**
Atomic functions operating on 32-bit integer values in global memory (Atomic Functions)	Yes					
atomicExch() operating on 32-bit floating point values in global memory (atomicExch())	Yes					
Atomic functions operating on 32-bit integer values in shared memory (Atomic Functions)	Yes					
atomicExch() operating on 32-bit floating point values in shared memory (atomicExch())	Yes					
Atomic functions operating on 64-bit integer values in global memory (Atomic Functions)	Yes					
Atomic functions operating on 64-bit integer values in shared memory (Atomic Functions)	Yes					
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())	Yes					
Atomic addition operating on 64-bit floating point values in global memory and shared memory (atomicAdd())	No				Yes	
Warp vote and ballot functions (Warp Vote Functions)	Yes					
__threadfence_system() (Memory Fence Functions)						
__syncthreads_count(), __syncthreads_and(), __syncthreads_or() (Synchronization Functions)						
Surface functions (Surface Functions)						
3D grid of thread blocks						
Unified Memory Programming						
Funnel shift (see reference manual)	No	Yes				
Dynamic Parallelism	No		Yes			
Half-precision floating-point operations: addition, subtraction, multiplication, comparison, warp shuffle functions, conversion	No			Yes		
Tensor Core	No					Yes

<table>
<tr><th></th><th colspan="12">Compute Capability</th></tr>
<tr><th>Technical Specifications</th><th>3.0</th><th>3.2</th><th>3.5</th><th>3.7</th><th>5.0</th><th>5.2</th><th>5.3</th><th>6.0</th><th>6.1</th><th>6.2</th><th>7.0</th><th>7.5</th></tr>
<tr><td>Maximum number of resident grids per device (Concurrent Kernel Execution)</td><td>16</td><td>4</td><td colspan="4">32</td><td>16</td><td>128</td><td>32</td><td>16</td><td colspan="2">128</td></tr>
<tr><td>Maximum dimensionality of grid of thread blocks</td><td colspan="12">3</td></tr>
<tr><td>Maximum x-dimension of a grid of thread blocks</td><td colspan="12">$2^{31}-1$</td></tr>
<tr><td>Maximum y- or z-dimension of a grid of thread blocks</td><td colspan="12">65535</td></tr>
<tr><td>Maximum dimensionality of thread block</td><td colspan="12">3</td></tr>
<tr><td>Maximum x- or y-dimension of a block</td><td colspan="12">1024</td></tr>
<tr><td>Maximum z-dimension of a block</td><td colspan="12">64</td></tr>
<tr><td>Maximum number of threads per block</td><td colspan="12">1024</td></tr>
<tr><td>Warp size</td><td colspan="12">32</td></tr>
<tr><td>Maximum number of resident blocks per multiprocessor</td><td colspan="4">16</td><td colspan="7">32</td><td>16</td></tr>
<tr><td>Maximum number of resident warps per multiprocessor</td><td colspan="11">64</td><td>32</td></tr>
<tr><td>Maximum number of resident threads per multiprocessor</td><td colspan="11">2048</td><td>1024</td></tr>
<tr><td>Number of 32-bit registers per multiprocessor</td><td colspan="3">64 K</td><td>128 K</td><td colspan="8">64 K</td></tr>
<tr><td>Maximum number of 32-bit registers per thread block</td><td>64 K</td><td>32 K</td><td colspan="4">64 K</td><td>32 K</td><td colspan="2">64 K</td><td>32 K</td><td colspan="2">64 K</td></tr>
<tr><td>Maximum number of 32-bit registers per thread</td><td>63</td><td colspan="11">255</td></tr>
<tr><td>Maximum amount of shared memory per multiprocessor</td><td colspan="3">48 KB</td><td>112 KB</td><td>64 KB</td><td>96 KB</td><td colspan="2">64 KB</td><td>96 KB</td><td>64 KB</td><td>96 KB</td><td>64 KB</td></tr>
<tr><td>Maximum amount of shared memory per thread block [27]</td><td colspan="10">48 KB</td><td>96 KB</td><td>64 KB</td></tr>
<tr><td>Number of shared memory banks</td><td colspan="12">32</td></tr>
<tr><td>Amount of local memory per thread</td><td colspan="12">512 KB</td></tr>
<tr><td>Constant memory size</td><td colspan="12">64 KB</td></tr>
<tr><td>Cache working set per multiprocessor for constant memory</td><td colspan="7">8 KB</td><td>4 KB</td><td colspan="4">8 KB</td></tr>
</table>

不同计算能力的 GPU 所支持的最大资源数

GPU	Kepler GK110	Maxwell GM200	Pascal GP100
Compute Capability	3.5	5.2	6.0
Threads / Warp	32	32	32
Max Warps / Multiprocessor	64	64	64
Max Threads / Multiprocessor	2048	2048	2048
Max Thread Blocks / Multiprocessor	16	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	32768	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB

占有率

- 占有率 (occupancy)：每个 SM 中活跃线程簇数与最大线程簇数的比值，越接近 100% 一般越好.
- 实际占有率往往受限于 kernel 的硬件资源消耗，主要硬件资源有:
  - 每个 SM 寄存器的容量;
  - 每个 SM 共享内存的容量;
  - 每个 SM 允许的最大线程块数;
  - 每个 SM 允许的最大线程数;
  - 每个线程块允许的最大线程数.
- 上述因素与具体硬件的计算能力密切相关.

几个例子

P100 每个 SM 寄存器容量为 256KB、共享内存容量为 64KB、最大活跃线程簇数 64 (即 2048 线程)，最大活跃线程块数 32.

例子 1：寄存器限制

若每个线程使用 32 个单精度寄存器，则活跃线程为 256KB$\div$(4B$\times$32) $=$ 2048，占有率为 1；若每个线程使用 64 个单精度寄存器，占有率则降低为 0.5.

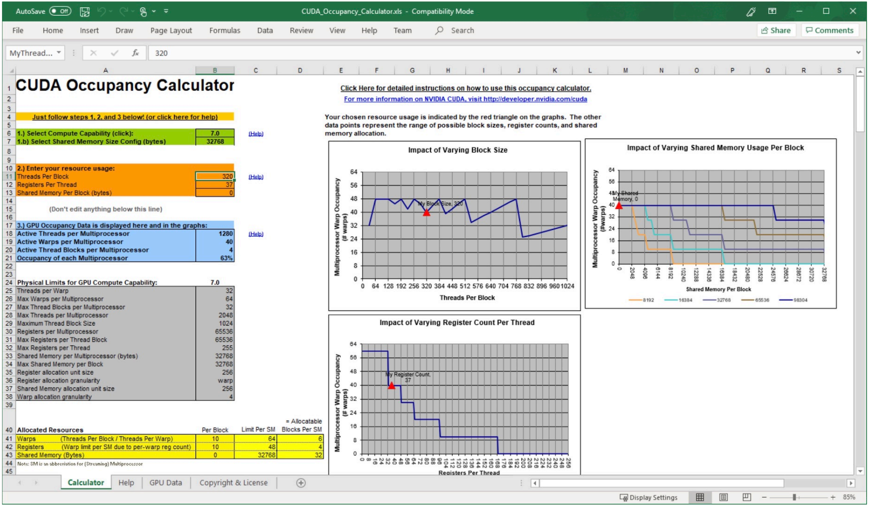
例子 2：共享内存限制

若每个线程使用 32Byte 共享内存，则活跃线程为 64KB/32B=2048，占有率为 1；若每个线程使用 64Byte 共享内存，占有率则降低为 0.5.

例子 3：线程块大小限制

若每个线程块有 32 个线程，占有率则不会高于 $32\times32\div2048 = 0.5$.

CUDA 占有率计算工具



CUDA 占有率计算函数

- 根据 kernel 预估块大小：

```
CUresult cuOccupancyMaxPotentialBlockSize(
    int* minGridSize, int* blockSize, CUfunction func,
    CUoccupancyB2DSize blockSizeToDynamicSMemSize,
    size_t dynamicSMemSize, int  blockSizeLimit )
```

- 返回 kernel 实际运行块数：

```
CUresult cuOccupancyMaxActiveBlocksPerMultiprocessor(
    int* numBlocks, CUfunction func,
    int blockSize, size_t dynamicSMemSize )
```