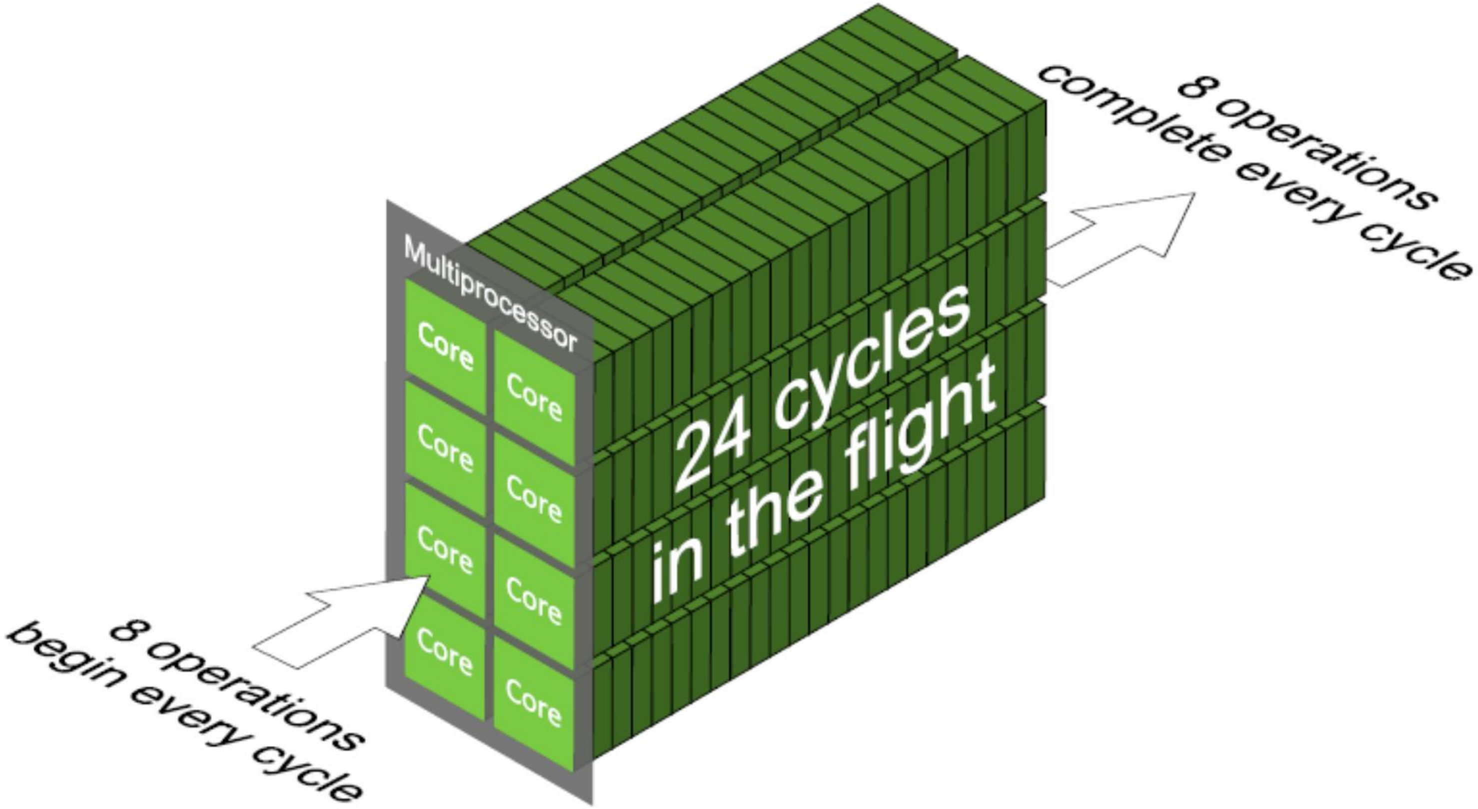


GPU并行计算与CUDA编程 第7课

本周介绍内容

- **优化GPU程序策略**
- 1. GPU优化原则与优化等级
- 2. 优化的步骤与流程
- 3. APOD——分析
- 4. APOD——并行
- 5. 测量程序内存使用率、带宽使用率
- 6. 计算占有率
- 7. 最小化线程发散的策略

1. GPU优化原则与优化步骤

- 优化目标：
- 1. Solve bigger problems
- 2. Solve more problems

- 优化原则：
- 1. 最大化算术强度
- 2. 减少内存操作花费的时间
- 3. 合并全局内存访问
- 4. 避免线程发散
- 5. 把高频使用数据移到共享内存

$$\frac{\text{Math}}{\text{Memory}}$$

- 优化等级：
- 1. 选择好的算法(Picking good algorithms)
- 2. 基本的高效代码的法则(Basic principles for efficiency)
- 3. 体系机构具体优化(Arch-specific detailed optimization)
- 4. 指令级的操作微观优化(micro-optimization for instruction levels)

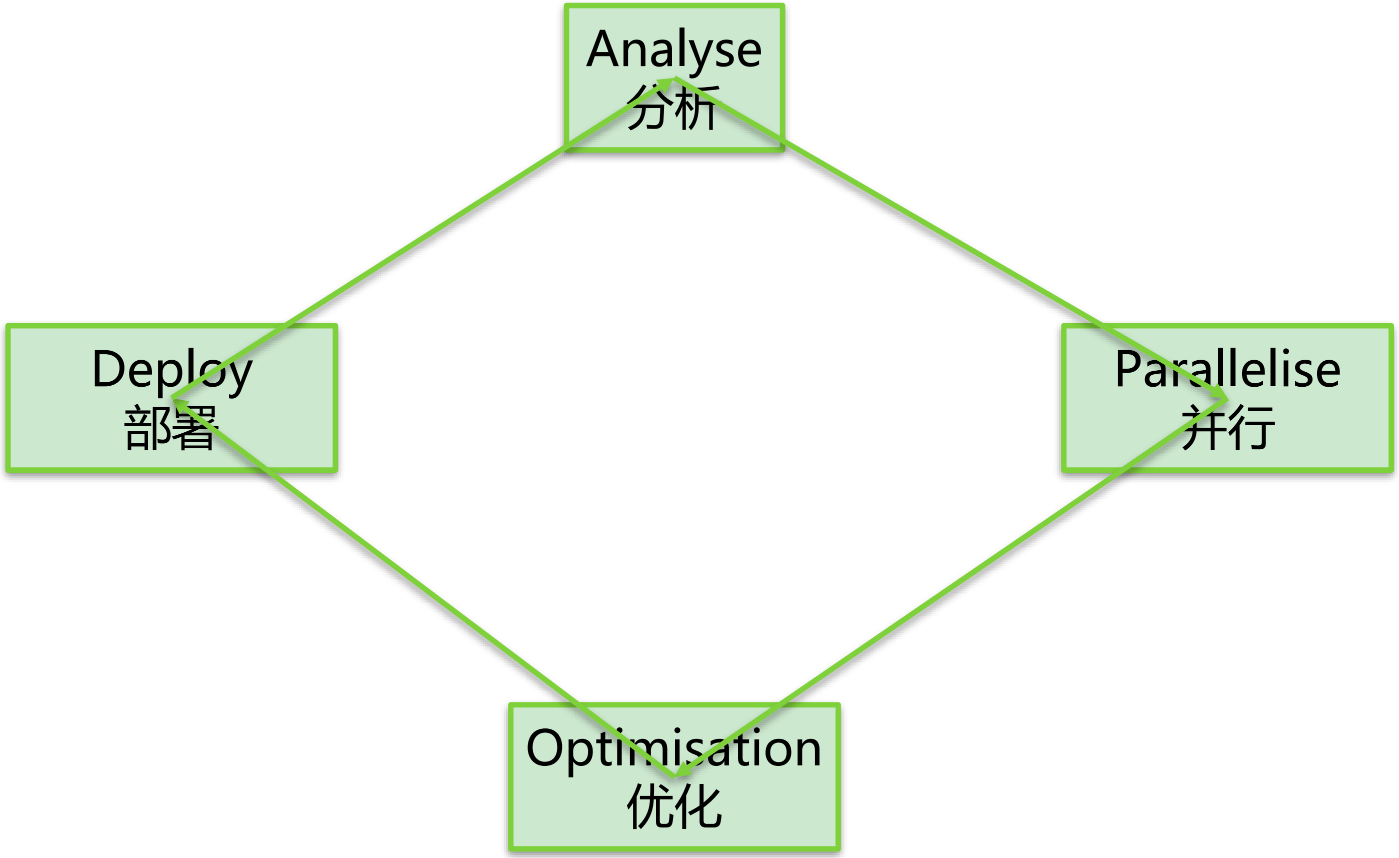
CPU例子：

1. 归并排序，运行时间O( );插入排序，运行时间O( )
2. 有效使用缓存的代码（e.g.通常遍历二维数组的行比列快，数组按行排序布局，可以有更好的缓存性能）
3. L1缓存限制
4. 浮点数移动做运算

GPU例子：

1. 归并排序、插入排序和堆排序
2. 合并全局内存；使用共享内存
3. 优化存储冲突和共享内存，优化寄存器

2. 优化的流程与步骤

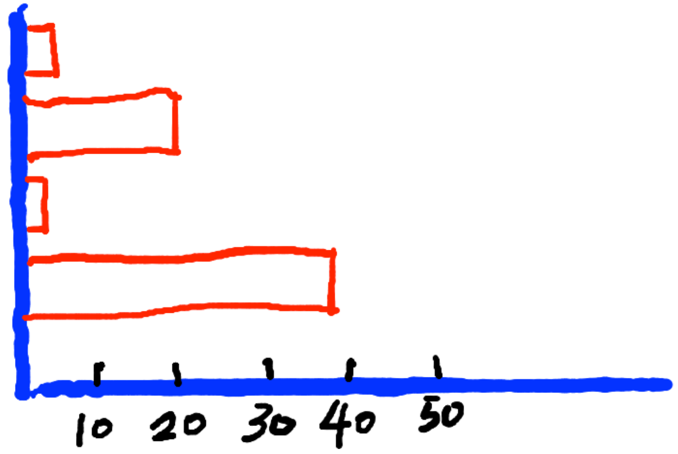


简称：APOD

- 分析：
  分析程序瓶颈、什么地方需要做并行、能够提供的资源
- 并行：
  1. Libraries：
  OpenMP(CPU),OpenACC
  2. Directives
  3. Pick an algorithm
- 优化：
  1. 测量内存、带宽和占用率等指标

3. APOD——分析

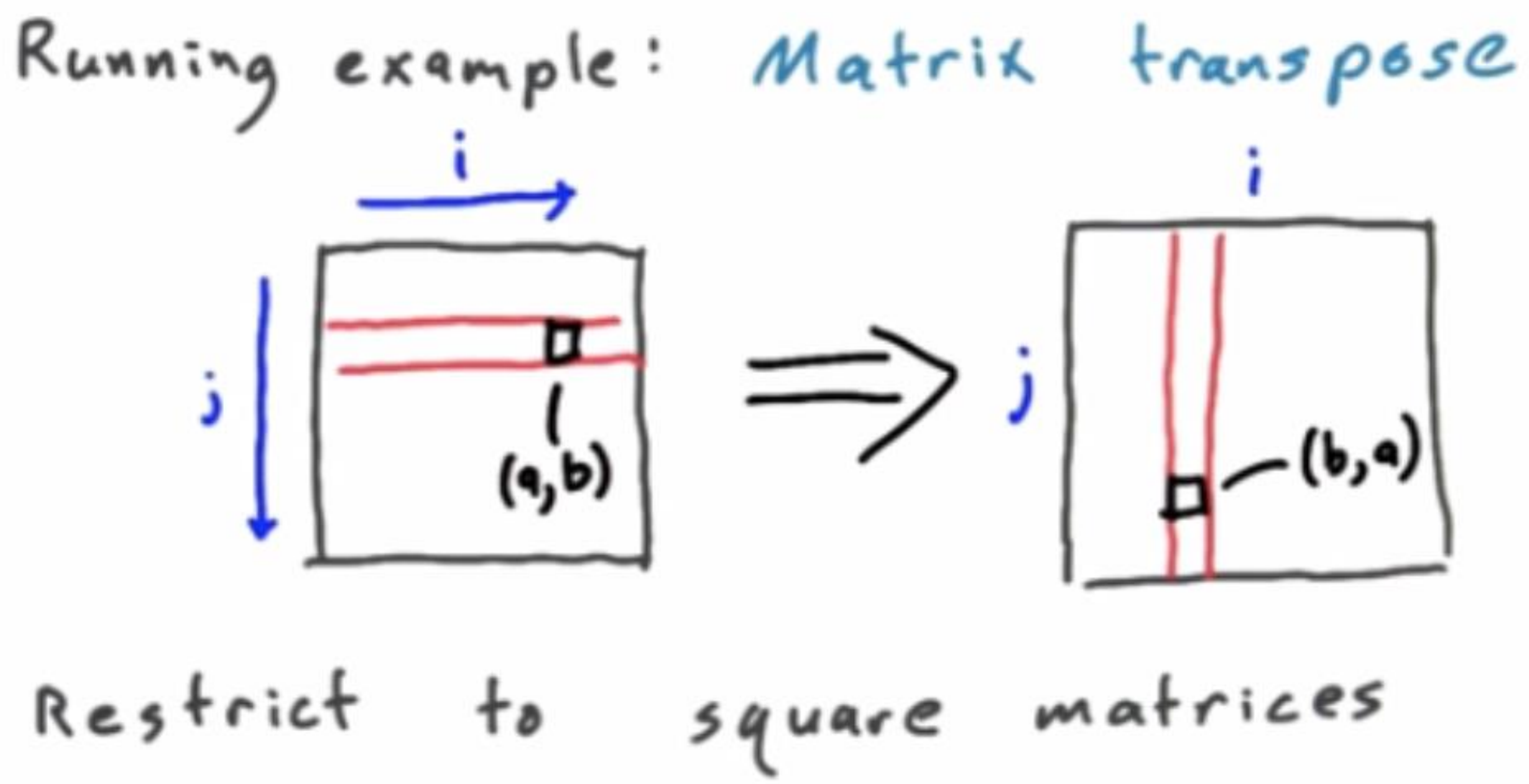
- **不要依赖直觉！**
- **分析工具：**
- 1. gProf
- 2. VTune
- 3. VerySleepy



% of time

4. APOD——并行

- 以矩阵转置为例：



DATAGURU专业数据分析社区

- 1. 单个线程处理

```
// to be launched on a single thread
__global__ void
transpose_serial(float in[], float out[])
{
    for(int j=0; j < N; j++)
        for(int i=0; i < N; i++)
            out[j + i*N] = in[i + j*N]; // out(j,i) = in(i,j)
}
```

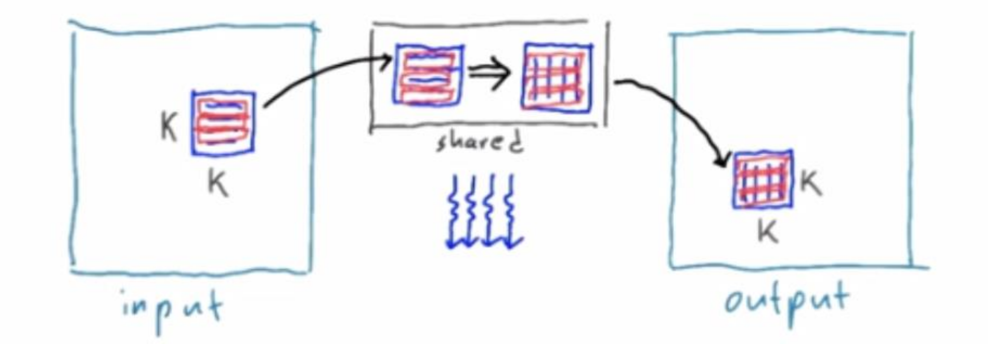
- 2. 矩阵每一行作为一个线程处理

```
// to be launched with one thread per row of output matrix
__global__ void
transpose_parallel_per_row(float in[], float out[])
{
    int i = threadIdx.x;

    for(int j=0; j < N; j++)
        out[j + i*N] = in[i + j*N]; // out(j,i) = in(i,j)
}
```

- 3. 每个线程处理一个元素

```
// to be launched with one thread per element, in KxK threadblocks
// thread (x,y) in grid writes element (i,j) of output matrix
__global__ void
transpose_parallel_per_element(float in[], float out[])
{
    int i = blockIdx.x * K + threadIdx.x;
    int j = blockIdx.y * K + threadIdx.y;

    out[j + i*N] = in[i + j*N]; // out(j,i) = in(i,j)
}
```

- 进一步优化




```
// to be launched with one thread per element, in KxK threadblocks
// thread blocks read & write tiles, in coalesced fashion
// shared memory array padded to avoid bank conflicts
__global__ void
transpose_parallel_per_element_tiled_padded(float in[], float out[])
{
    // (i,j) locations of the tile corners for input & output matrices:
    int in_corner_i  = blockIdx.x * K, in_corner_j  = blockIdx.y * K;
    int out_corner_i = blockIdx.y * K, out_corner_j = blockIdx.x * K;

    int x = threadIdx.x, y = threadIdx.y;

    __shared__ float tile[K][K+1];

    // coalesced read from global mem, TRANSPOSED write into shared mem:
    tile[y][x] = in[(in_corner_i + x) + (in_corner_j + y)*N];
    __syncthreads();
    // read from shared mem, coalesced write to global mem:
    out[(out_corner_i + x) + (out_corner_j + y)*N] = tile[x][y];
}
```

- 进进一步优化，K=16

```
// to be launched with one thread per element, in KxK threadblocks
// thread blocks read & write tiles, in coalesced fashion
// shared memory array padded to avoid bank conflicts
__global__ void
transpose_parallel_per_element_tiled_padded16(float in[], float out[])
{
    // (i,j) locations of the tile corners for input & output matrices:
    int in_corner_i  = blockIdx.x * 16, in_corner_j  = blockIdx.y * 16;
    int out_corner_i = blockIdx.y * 16, out_corner_j = blockIdx.x * 16;

    int x = threadIdx.x, y = threadIdx.y;

    __shared__ float tile[16][16+1];

    // coalesced read from global mem, TRANSPOSED write into shared mem:
    tile[y][x] = in[(in_corner_i + x) + (in_corner_j + y)*N];
    __syncthreads();
    // read from shared mem, coalesced write to global mem:
    out[(out_corner_i + x) + (out_corner_j + y)*N] = tile[x][y];
}
```

- 结果

```
transpose_serial: 213 ms.
Verifying transpose...Success
transpose_parallel_per_row: 2.28794 ms.
Verifying transpose...Success
transpose_parallel_per_element: 0.509376 ms.
Verifying transpose...Success
transpose_parallel_per_element_tiled 32x32: 0.319424 ms.
Verifying ...Success
transpose_parallel_per_element_tiled 16x16: 0.149824 ms.
Verifying ...Success
transpose_parallel_per_element_tiled_padded 16x16: 0.138176 ms.
Verifying...Success
```

5. 测量程序内存使用率、带宽(bandwidth)使用率

- 理论峰值带宽：
- Memory Clock = 2505 x 　　clocks/sec
- Memory Bus = 128 bits = 16 bytes/clock
- 理论的峰值带宽约等于40GB/s

- 程序使用bandwidth低于40%算较低的利用率；
- 程序使用bandwidth高于75%算较高的利用率

```
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 4044 MBytes (4240965632 bytes)
MapSMtoCores for SM 5.0 is undefined.  Default to use 192 Cores/SM
MapSMtoCores for SM 5.0 is undefined.  Default to use 192 Cores/SM
  ( 5) Multiprocessors x (192) CUDA Cores/MP:    960 CUDA Cores
  GPU Clock rate:                                928 MHz (0.93 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 2097152 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65536), 3
D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16
384) x 2048
  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
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           1 / 0
```

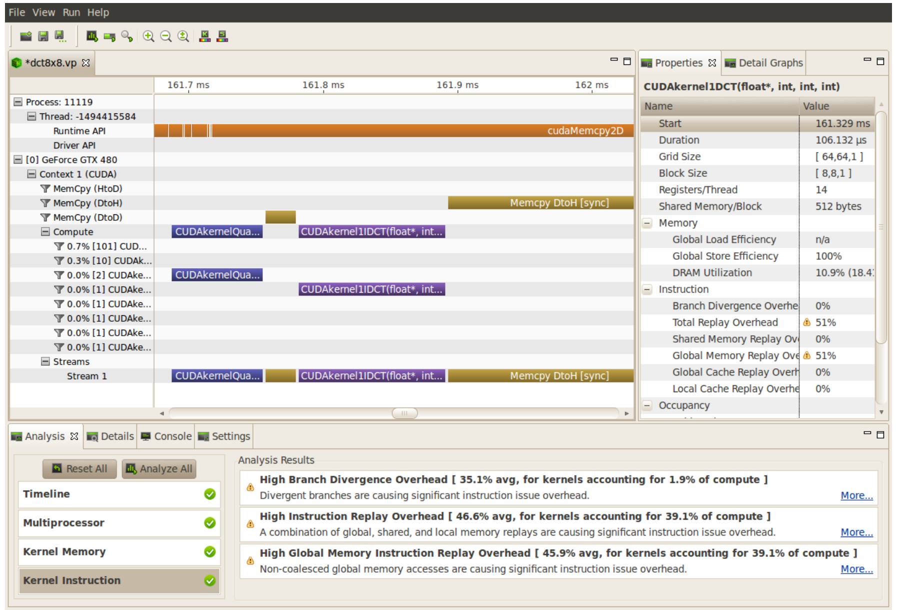
6. 计算占有率

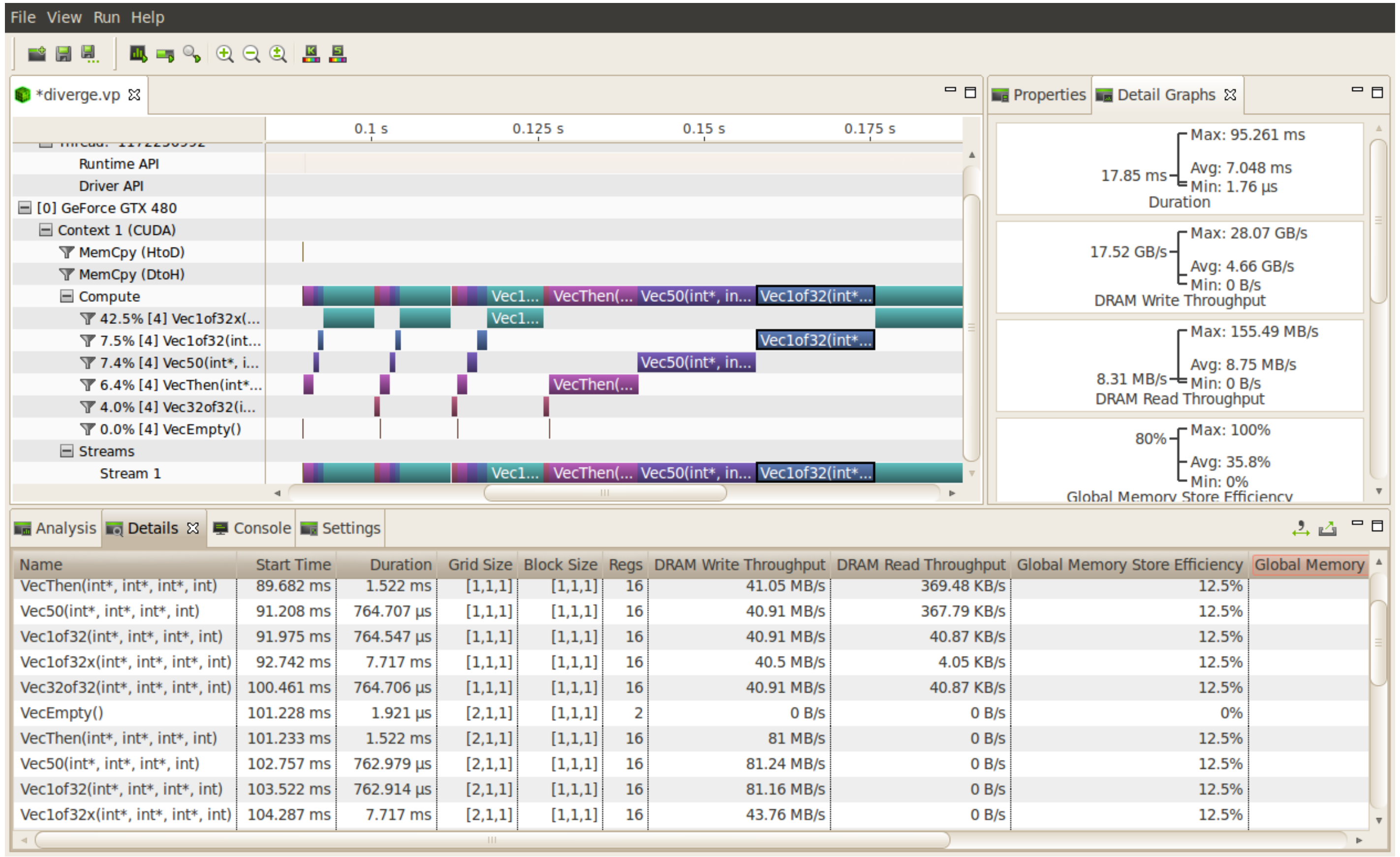
- 每一个SM上的参数：
- 每个SM最大的Block数，每个Block上最大的线程数，regsters for all threads,bytes of shared memory，每个block最大的共享内存。
- 都可以通过deviceQuery查到

```
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 4044 MBytes (4240965632 bytes)
MapSMtoCores for SM 5.0 is undefined.  Default to use 192 Cores/SM
MapSMtoCores for SM 5.0 is undefined.  Default to use 192 Cores/SM
  ( 5) Multiprocessors x (192) CUDA Cores/MP:    960 CUDA Cores
  GPU Clock rate:                                928 MHz (0.93 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 2097152 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65536), 3
D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16
384) x 2048
  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
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           1 / 0
```

NVVP:NVIDIA Visual Profiler

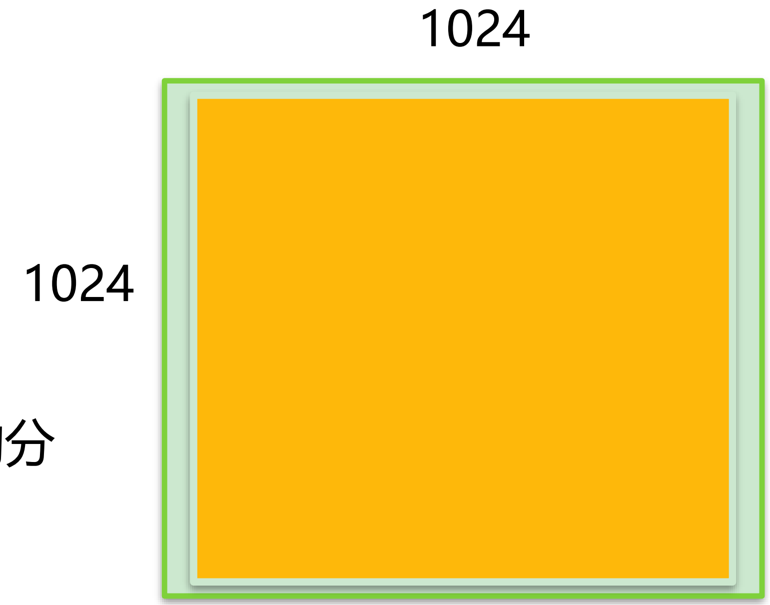
- https://developer.nvidia.com/nvidia-visual-profiler





7.最小化线程发散的策略

- 1. WARP：Sets of threads that execute the same instruction at the same time.
- 2. SIMD：Single instructions ,multiple data
- 3. SIMT: Single instructions ,multiple thread



- 策略：
- 1. 避免分支代码
  如果有if或者switch语句，考虑相邻线程是否可以使用不同的分支，如果可以，则并行化进行重构。
- 2. 避免大量工作量不平衡的线程

本周作业

- 1. 对transpose.cu再设计一个优化的并行计算方法，运行速度起码要比transpose_parallel_per_element要快。
- 2. 计算自己电脑的理论峰值带宽bandwidth，1中新设计的优化方法的占用率。

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

Thanks

FAQ时间