如何使用MPS提升GPU计算收益

NVIDIA GPU计算专家团队 吴磊 2019.10.31


多进程并发的简单案例

测试环境

- 操作系统: Ubuntu 16.04
- CUDA版本: 9.1.85
- GPU: Titan V
- 启动版本: 390.48

测试用例

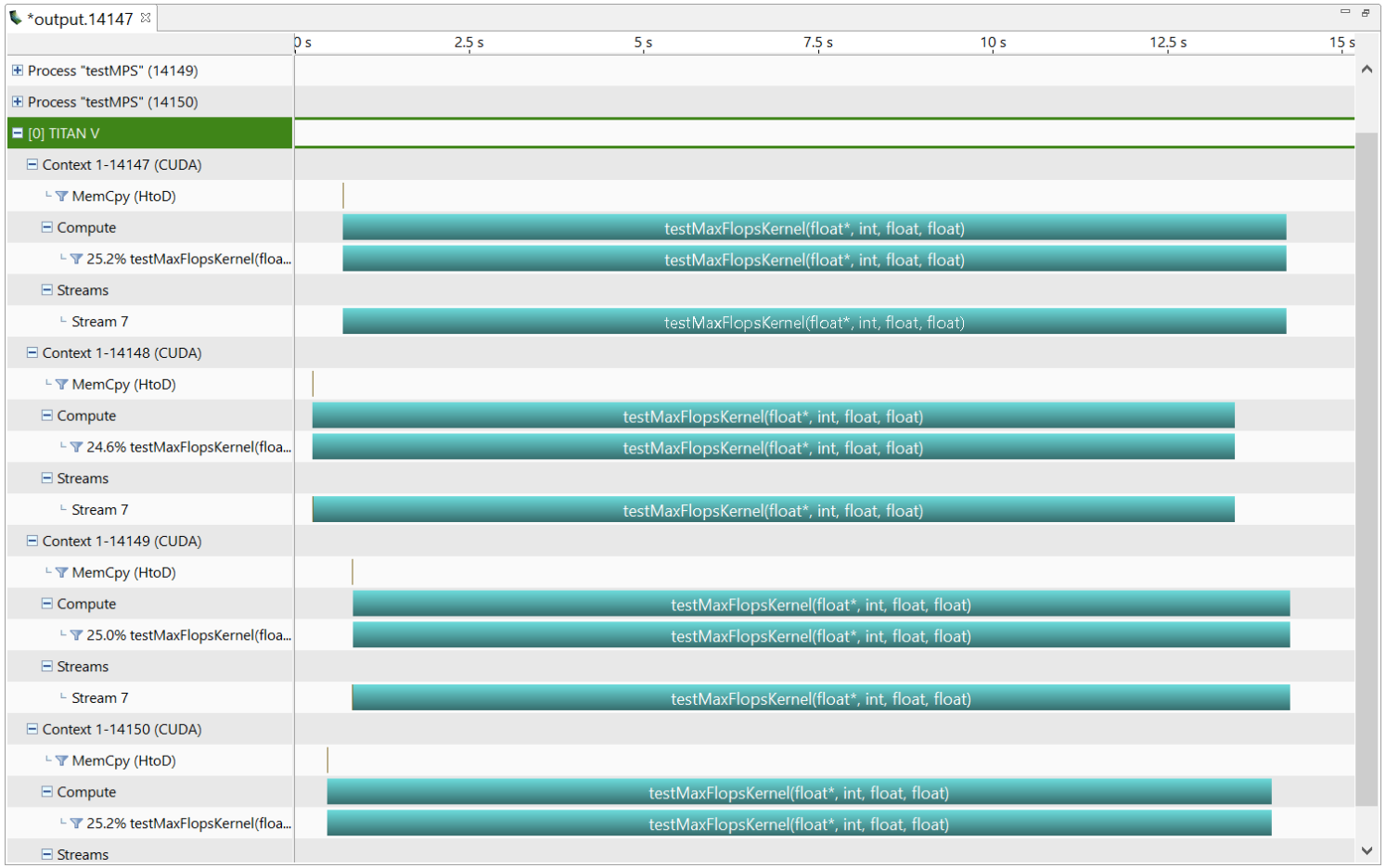
- nRepeats = 1000000000
- Block Size: (1, 1, 1)
- Grid Size: (1, 1, 1)

```
__global__ void testMaxFlopsKernel(float *
pData, int nRepeats, float v1, float v2)
{
        int tid = blockIdx.x* blockDim.x+
threadIdx.x;

        float s = pData[tid], s2 = 10.0f - s, s3
= 9.0f - s, s4 = 9.0f - s2;

        for(int i = 0; i < nRepeats; i++)
        {
            s=v1-s*v2;
            s2=v1-s1*v2;
            s3=v1-s2*v2;
            s4=v1-s3*v2;
        }

        pData[tid] = ((s+s2)+(s3+s4));
}
```

测试结果

- 1 x process: 3518ms
- 2 x processes: 6854ms, 6838ms
  - 几乎是单个进程执行时间的2倍
- 4 x processes: 13952ms, 13947ms, 13934ms, 13940ms
  - 几乎是单个进程执行时间的4倍

NVVP 性能分析



4 x processes


背景知识：
CUDA CONTEXT和HYPER-Q技术

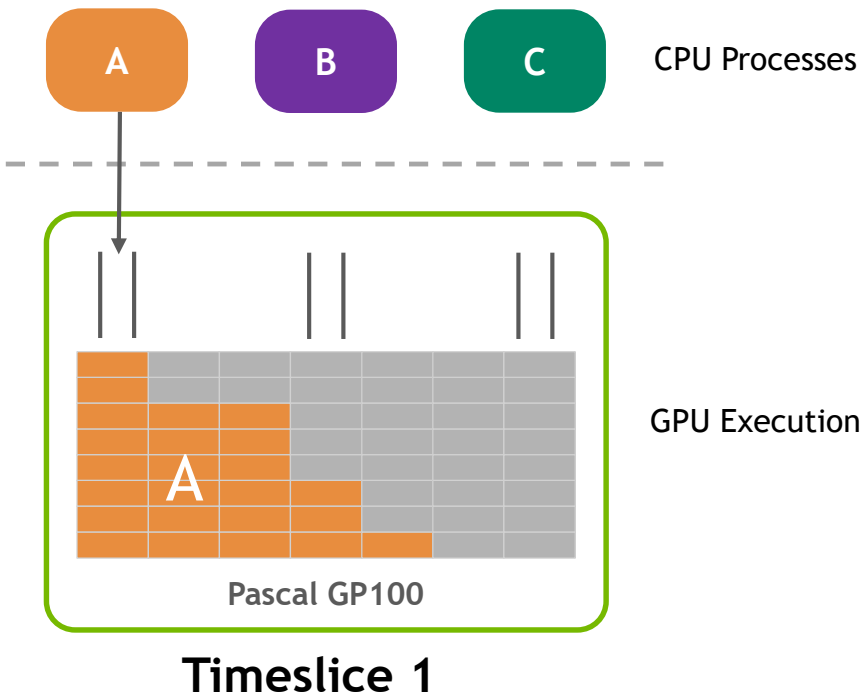
背景知识 (1)

CUDA context

- 什么是 CUDA context?
  - 类似于CPU进程上下文，表示与特定进程关联的所有状态
    - 从CPU端分配的GPU上的Global memory (cudaMalloc/cudaMallocManaged)
    - Kernel函数中定义和分配的堆栈空间，例如local memory
    - CUDA streams / events 对象
    - 代码模块(*.cubin, *.ptx)
  - 不同的进程有自己的CUDA context
  - 每个context有自己的地址空间，并且不能访问其他CUDA context的地址空间

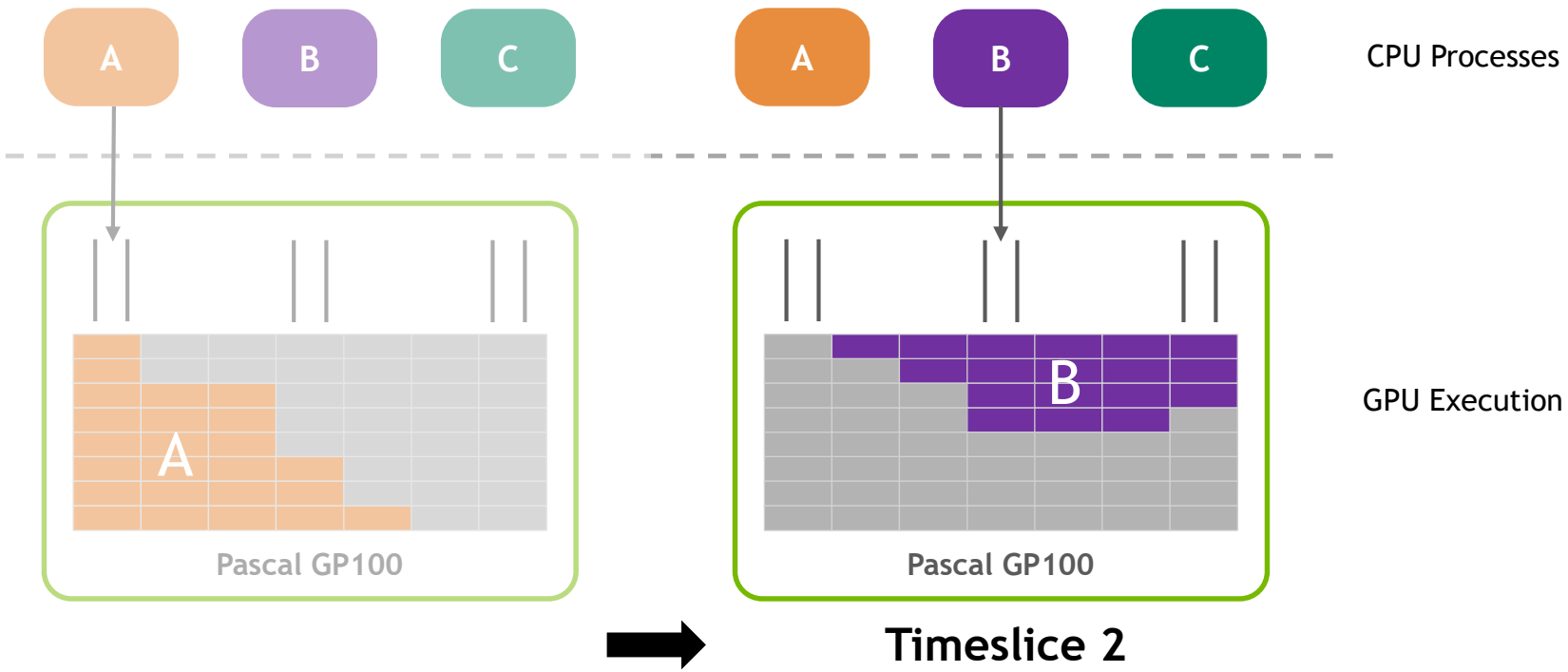
背景知识 (1)

CUDA context



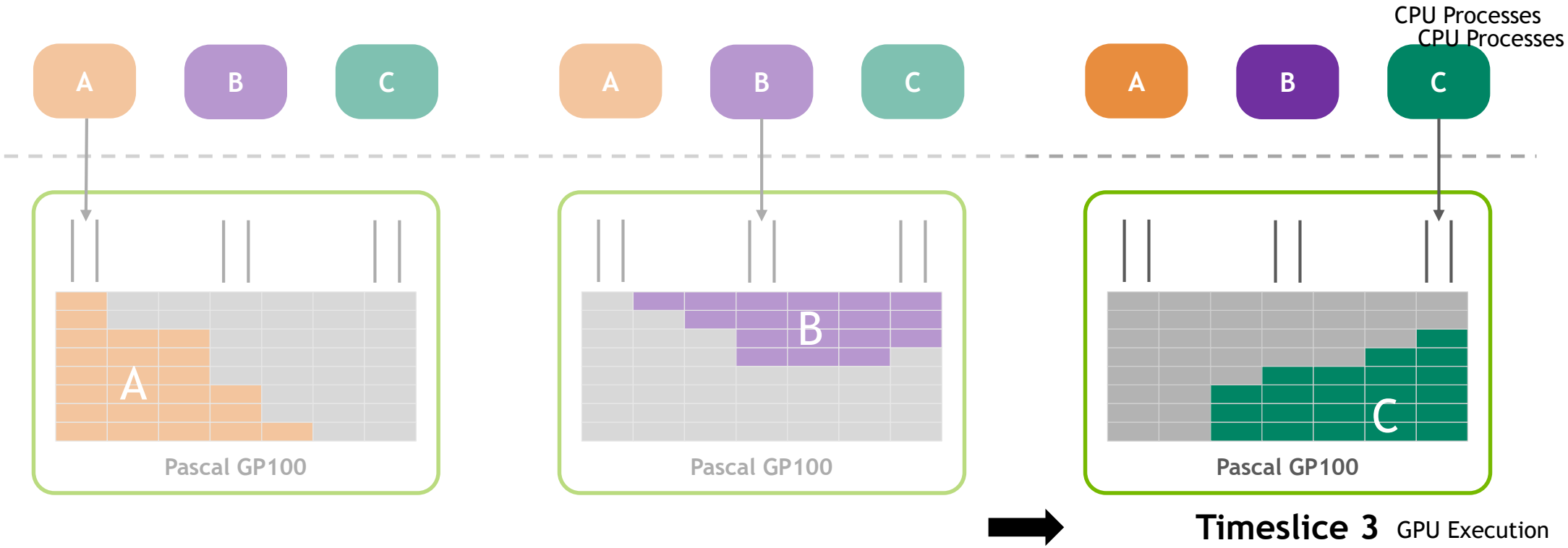
背景知识 (1)

CUDA context



背景知识 (1)

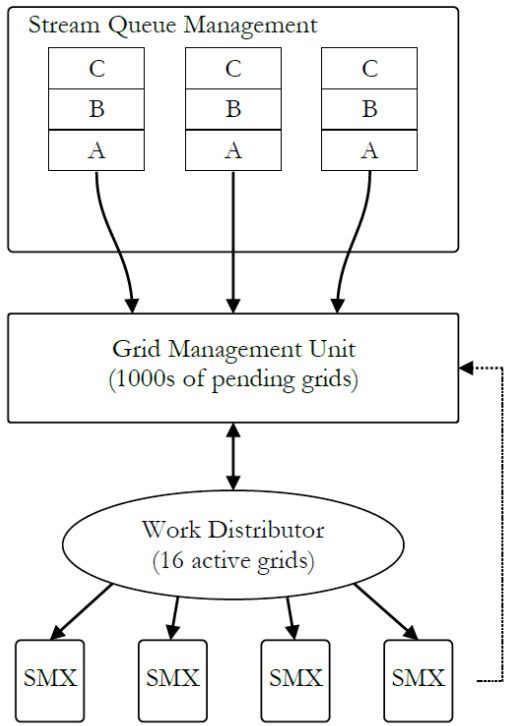
CUDA context



背景知识 (2)

Hyper-Q

- 什么是 Hyper-Q? -- Hyper Queue
  - 允许多个CPU 线程或进程同时加载任务到一个GPU上，实现CUDA kernels的并发执行 -- 硬件特性
- 支持的连接类型
  - Multi cuda streams
  - Multi cpu thrreads
  - Multi cpu processes——MPS
- 管理可并发的最大连接数
  - CUDA_DEVICE_MAX_CONNECTIONS = 32 (默认是8)



背景知识 (2)

Hyper-Q

- 带来的好处
  - 增加GPU利用率（utilization）和占用率（occupancy）
  - 减少CPU空闲时间
  - 增加吞吐率并减少延迟
- 使用限制
  - 当kernel A正在执行时，只有当GPU上任意SM上有足够的资源来执行kernel B中的1个线程块时，kernel B才会被发射
    - 寄存器，共享内存，线程块槽位，等等.
  - 要求计算能力大于等于3.5
  - 最大连接数限制：32个

背景知识 (2)

Hyper-Q

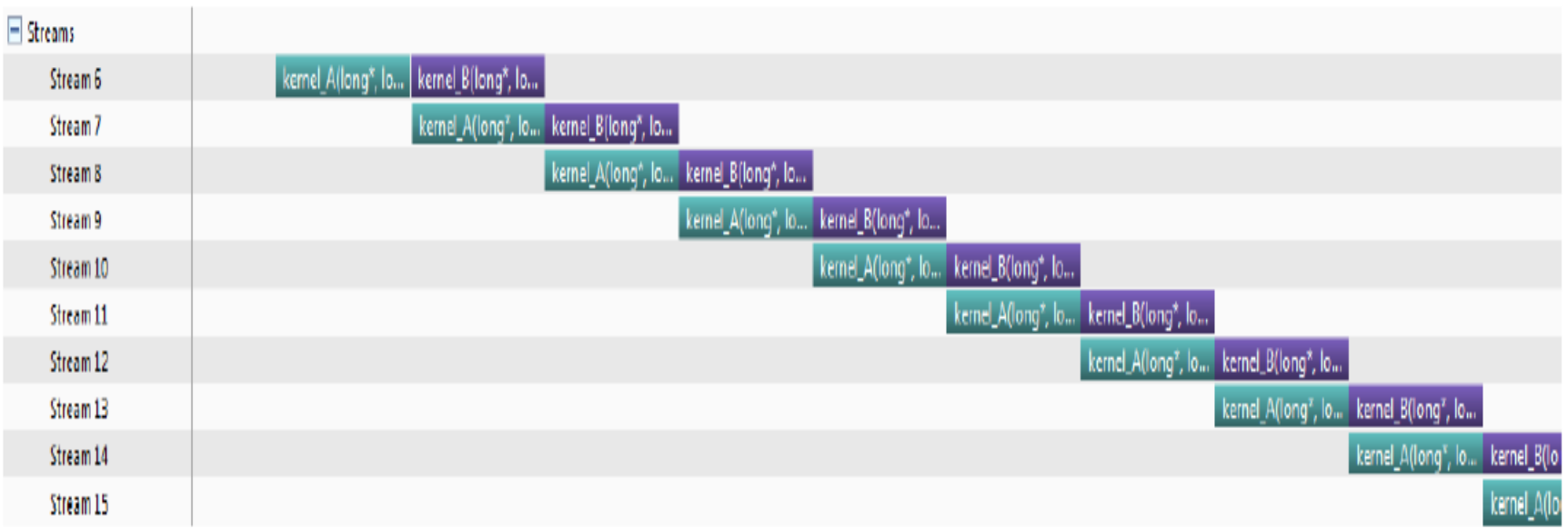
- 示例代码: $CUDA_PATH/samples/6_Advanced/simpleHyperQ

```
for (int i = 0 ; i < nstreams ; ++i)
{
        kernel_A<<<1,1,0,streams[i]>>>(&d_a[2*i], time_clocks);
        total_clocks += time_clocks;
        kernel_B<<<1,1,0,streams[i]>>>(&d_a[2*i+1], time_clocks);
        total_clocks += time_clocks;
}
```

背景知识 (2)

Hyper-Q

- 示例代码：$CUDA_PATH/samples/6_Advanced/simpleHyperQ

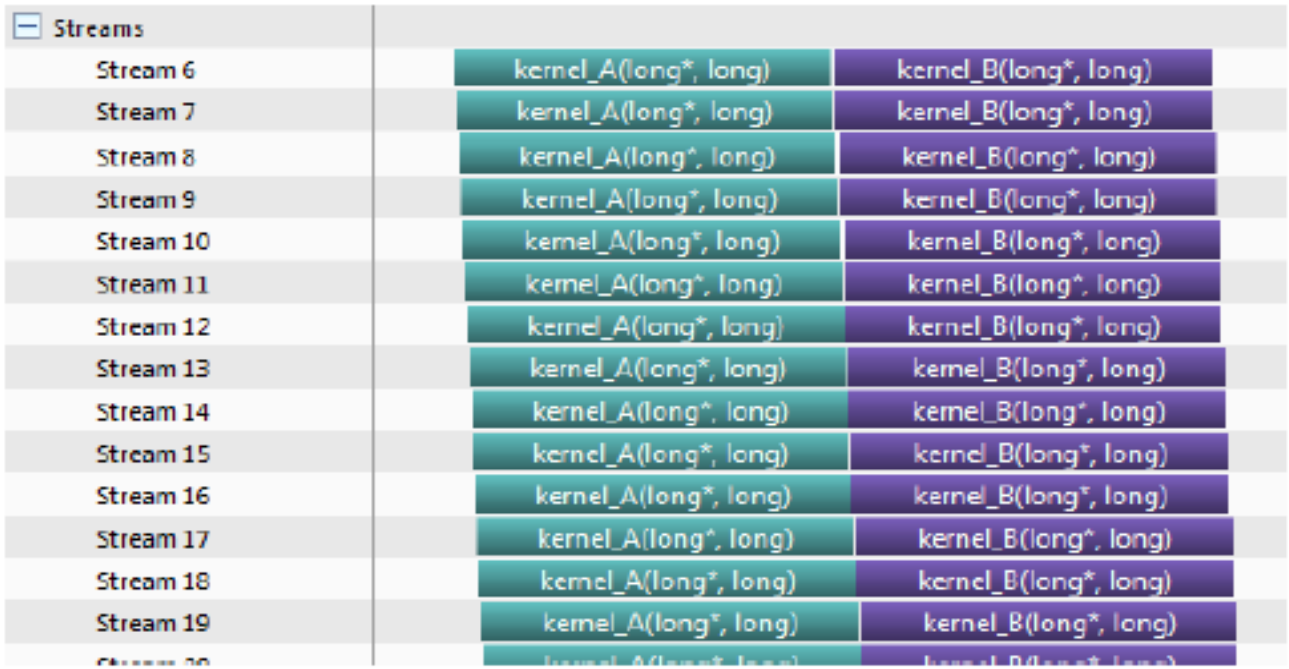


没有Hyper-Q时的执行时间线

背景知识 (2)

Hyper-Q

- 示例代码：$CUDA_PATH/samples/6_Advanced/simpleHyperQ



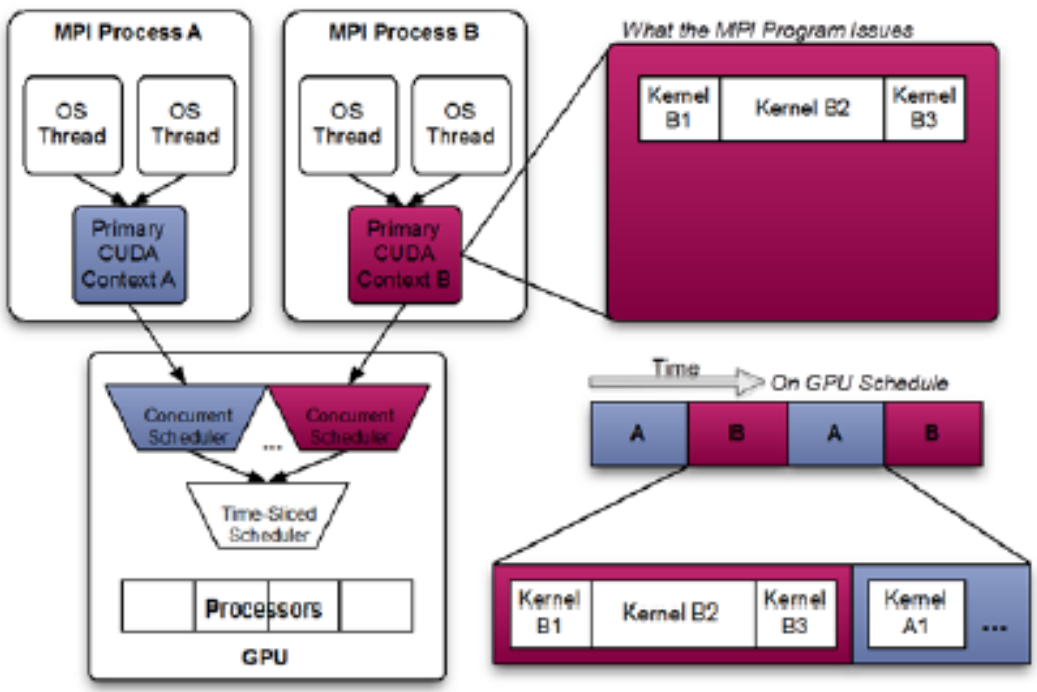
有Hyper-Q时的执行时间线

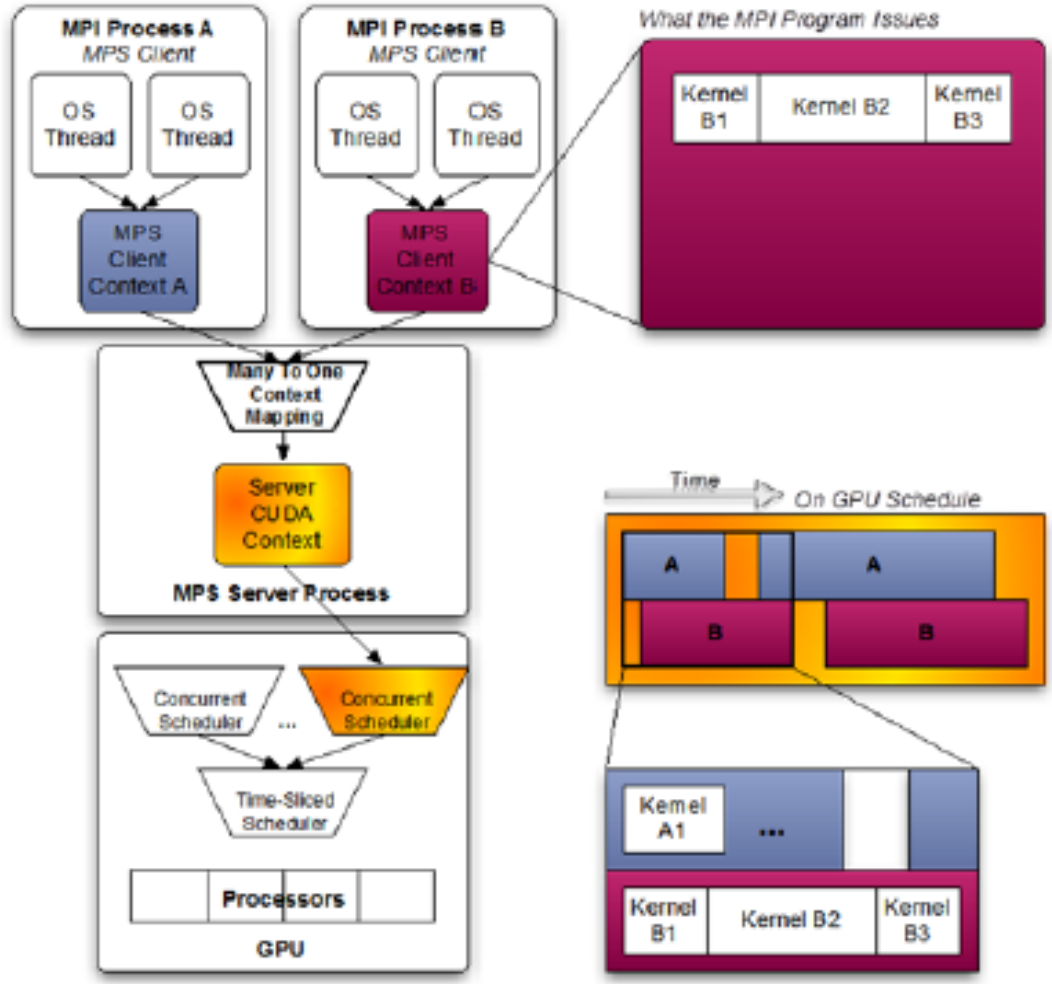

什么是MPS

什么是MPS

- 什么是 MPS – Multi-Process Service， 多进程服务
  - 一组可替换的，二进制兼容的CUDA API实现，包括：
    - 守护进程
    - 服务进程
    - 用户运行时
  - 利用GPU上的Hyper-Q 能力
    - 允许多个CPU进程共享同一GPU context
    - 允许不同进程的kernel和memcpy操作在同一GPU上并发执行，以实现最大化GPU利用率.

什么是MPS





什么是MPS

- 带来的好处
  - 提升GPU利用率（时间上）和占用率（空间上）
  - 减少GPU上下文切换时间
  - 减少GPU上下文存储空间
- 使用限制
  - 操作系统: 仅支持linux
  - GPU版本：大于等于CC 3.5
  - CUDA版本：大于等于5.5
  - 最大用户连接数量：
    - Pascal及之前GPU: 16
    - Volta及之后GPU: 48


如何使用MPS

如何使用MPS

- *启动*
  - 设置GPU compute mode 为 exclusive mode (非必须，但推荐设置)
    - sudo nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
  - 启动MPS 守护进程
    - export CUDA_VISIBLE_DEVICES=0
    - export CUDA_MPS_PIPE_DIRECTORY=/tmp/nvidia-mps (cuda 7.0 以后非必须)
    - export CUDA_MPS_LOG_DIRECTORY=/tmp/nvidia-log (cuda 7.0 以后非必须)
    - nvidia-cuda-mps-control -d

如何使用MPS

- 启动
  - 查看MPS 守护进程是否正在运行
    - ps -ef | grep mps
  - 运行应用程序
    - mpirun -np 4 ./test

```
david@GOS:~$ ps -ef | grep mps
david     4164     1  0 15:30 ?        00:00:00 nvidia-cuda-mps-control -d
david     4167  3660  0 15:30 pts/11   00:00:00 grep --color=auto mps
```

应用程序运行前

```
david@GOS:~$ ps -ef | grep mps
david     4164     1  0 15:30 ?        00:00:00 nvidia-cuda-mps-control -d
david     4213  4164  0 15:31 ?        00:00:02 nvidia-cuda-mps-server
david     5199  3660  0 15:39 pts/11   00:00:00 grep --color=auto mps
```

应用程序运行后

如何使用MPS

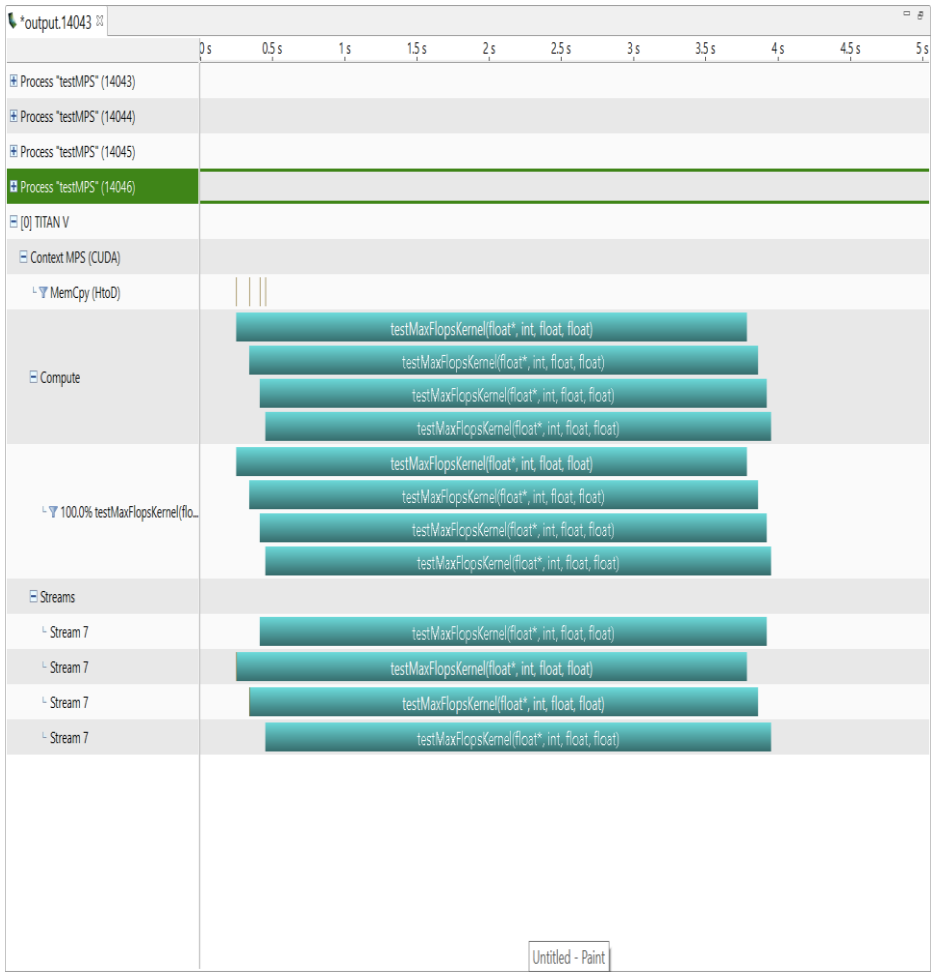
- 停止
  - echo quit | nvidia-cuda-mps-control
  - sudo nvidia-smi -i 0 -c 0
- 监视
  - nvidia-smi

```
Tue May 15 14:20:54 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.48                 Driver Version: 390.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  TITAN V            Off  | 00000000:02:00.0  On |                  N/A |
| 31%   44C    P2    41W / 250W |   3669MiB / 12065MiB |    100%   E. Process |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1376      G   /usr/lib/xorg/Xorg                           152MiB |
|    0     30613      C   nvidia-cuda-mps-server                       416MiB |
|    0     30916    M+C   ./testMemUsage                              1536MiB |
|    0     30917    M+C   ./testMemUsage                              1536MiB |
+-----------------------------------------------------------------------------+
```

2个进程同时在GPU上执行

如何使用MPS

- 分析
  - 运行 NVPROF 收集执行信息
    - nvprof --profile-all-processes -o output.%p
  - 在另一个窗口中运行应用程序
    - mpirun -np 4 ./test
  - 回到窗口一，按Ctrl+c结束nvprof收集信息
  - 将 nvprof 结果 (output.%p) 导入到 nvvp
    - 选择File->Import
    - 选择nvprof, Multiple Processes
    - 每个process会对应一个nvprof的profile文件，选中多个文件




MPS应用案例

MPS 应用案例 (1)

简单kernel的并发

- 测试环境
  - 操作系统: Ubuntu 16.04
  - CUDA版本: 9.1.85
  - GPU: Titan V
  - 启动版本: 390.48
- 测试用例
  - nRepeats = 1000000000
  - Block Size: (1, 1, 1)
  - Grid Size: (1, 1, 1)

```
__global__ void testMaxFlopsKernel(float *
pData, int nRepeats, float v1, float v2)
{
        int tid = blockIdx.x* blockDim.x+
threadIdx.x;

        float s = pData[tid], s2 = 10.0f - s, s3
= 9.0f - s, s4 = 9.0f - s2;

        for(int i = 0; i < nRepeats; i++)
        {
            s=v1-s*v2;
            s2=v1-s1*v2;
            s3=v1-s2*v2;
            s4=v1-s3*v2;
        }

        pData[tid] = ((s+s2)+(s3+s4));
}
```

MPS 应用案例 (1)

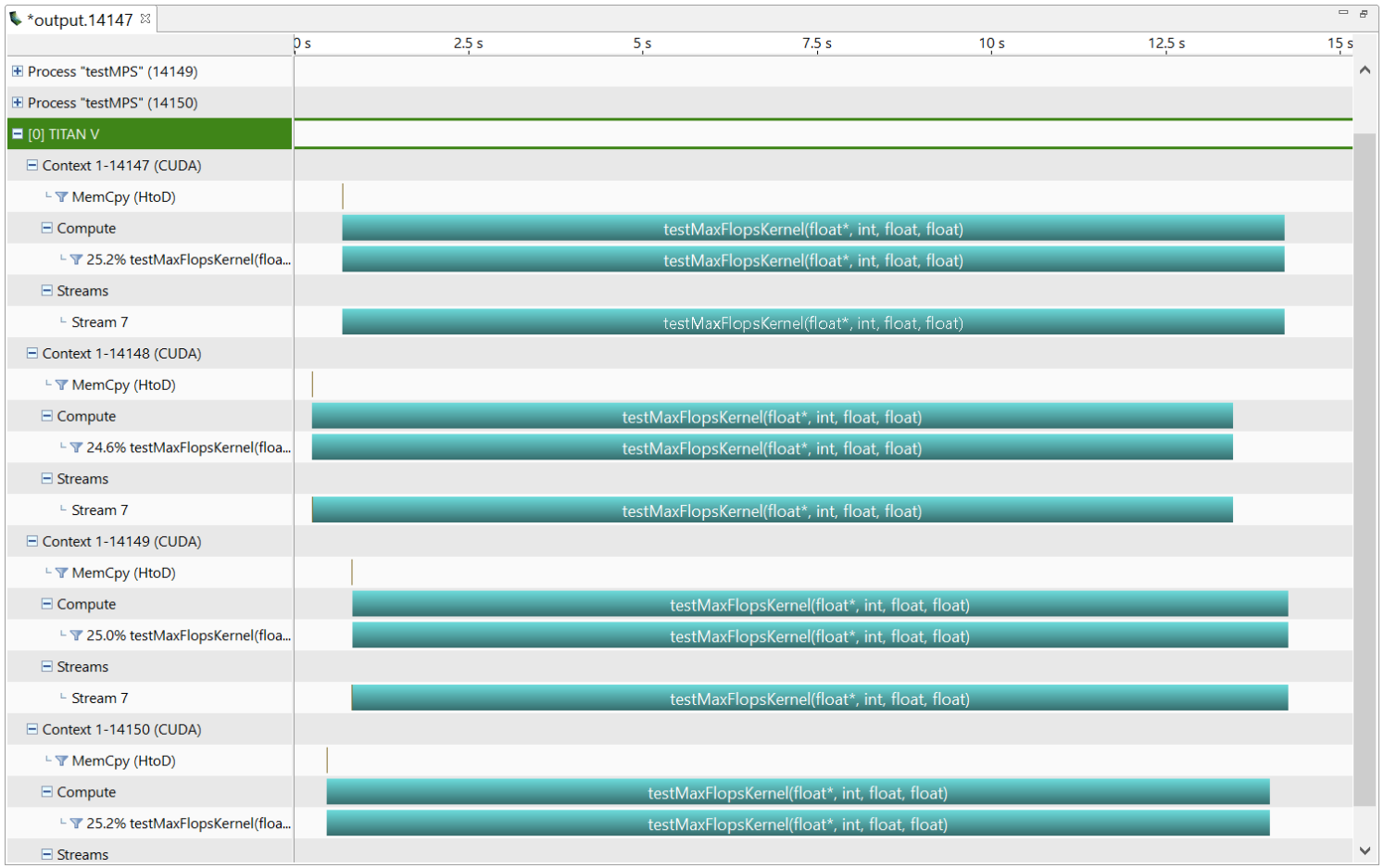
简单kernel的并发

- 测试结果
  - 1 x process: 3518ms
  - 2 x processes **关闭 MPS**: 6854ms, 6838ms
    - 几乎是单个进程执行时间的2倍
  - 2 x processes **开启 MPS**: 3467ms, 3467ms
    - 几乎等于单个进程执行时间
  - 4 x processes **关闭 MPS**: 13952ms, 13947ms, 13934ms, 13940ms
    - 几乎是单个进程执行时间的4倍
  - 4 x processes **开启 MPS**: 3505ms, 3492ms, 3492ms, 3492ms
    - 几乎等于单个进程执行时间

MPS 应用案例 (1)

简单kernel的并发

- NVPROF & NVVP 性能分析



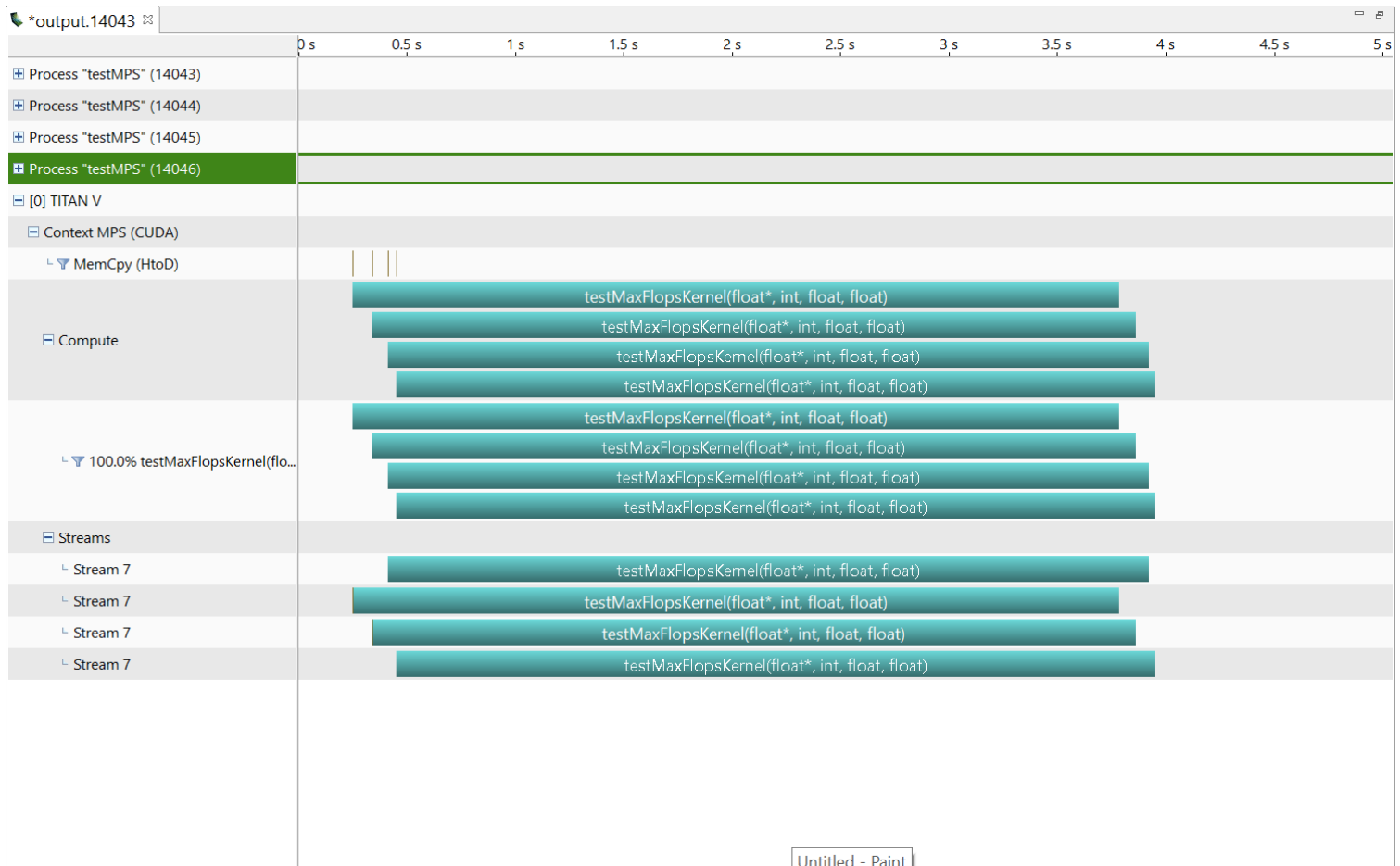
4 x processes

关闭 MPS

MPS 应用案例 (1)

简单kernel的并发

- NVPROF & NVVP 性能分析



4 x processes
开启 MPS

MPS 应用案例 (1)

简单kernel的并发

- NVPROF & NVVP 性能分析结论
  - 对比 MPS 关闭和开启状态下，应用程序执行情况：
    - 4 x cuda contexts　VS　1 x cuda context
    - 4 x kernel 执行时间 VS　1 x kernel 执行时间

MPS 应用案例 (2)

JPEG图像的Resizing 处理

- JPEG resizing 是图像分类、目标检测等图像处理相关神经网络中常用的预处理方法
- 测试环境：
  - Fastvideo SDK
  - Tesla V100 16GB GPU
  - Resizing from 1920x1080 (2K) to 480x270

MPS 应用案例 (2)

JPEG Resizing 处理

- 测试结果：

Processes number	FPS without MPS	FPS with MPS	Speedup
2	1152	1633	1.42
4	1025	2319	2.26
8	1014	3024	2.98
16	1012	3458	3.42
18	1009	3558	3.53

MPS 应用案例 (3)

ResNet-50 模型推理

- 使用TensorRT推理引擎运行ResNet-50模型
- 测试环境：
  - Tesla V100 16GB GPU, CUDA 10.1, Driver 418.67
  - 使用nvidia docker环境：nvcr.io/nvidia/tensorrt:19.07-py3
  - 在container中，执行命令进行trt engine building：$ trtexec --batch=1 --iterations=100 --workspace=1024 --deploy=/xxx/ResNet-50-deploy.prototxt --model=/xxx/ResNet-50-model.caffemodel --output=prob --fp16 --saveEngine=/workspace/rn50-bs1.engine
  - 使用build好的trt engine，在多个CPU进程中执行trt推理：$ mpirun -np 8 --allow-run-as-root trtexec --loadEngine=/workspace/rn50-bs1.engine --iterations=1000 --workspace=1024 --fp16 > trt-mps-mpi-8.log

MPS 应用案例 (3)

ResNet-50 模型推理

```
root@dgx2:~# nvidia-smi -i 0
Mon Aug 12 08:37:25 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   59C    P0   272W / 300W |   7026MiB / 16130MiB |    100%   E. Process |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1752      C   nvidia-cuda-mps-server                        29MiB |
|    0     46530    M+C   trtexec                                      873MiB |
|    0     46539    M+C   trtexec                                      873MiB |
|    0     46546    M+C   trtexec                                      873MiB |
|    0     46554    M+C   trtexec                                      873MiB |
|    0     46562    M+C   trtexec                                      873MiB |
|    0     46570    M+C   trtexec                                      873MiB |
|    0     46578    M+C   trtexec                                      873MiB |
|    0     46586    M+C   trtexec                                      873MiB |
+-----------------------------------------------------------------------------+
```

MPS 应用案例 (3)

ResNet-50 模型推理

	Without MPS		With MPS			
Process number	Latency (ms)	Throughput (images/s)	Latency (ms)	speedup	Throughput	speedup
1	1.86	537	1.86	1	537	1
2	3.38	591	2.2	1.54	909	1.54
4	6.9	579	3.42	2.02	1169	2.02
8	13.6	588	5.2	2.61	1538	2.61

参考资料

- 《 Hyper-Q Example 》
- 《MULTI-PROCESS SERVICE》
- 《IMPROVING GPU UTILIZATION WITH MULTI-PROCESS SERVICE (MPS)》

NVIDIA GTC

2019 年 12 月 16 日 – 19 日
苏州，中国

沟通

与来自 NVIDIA 和其他业界领先组织的技术专家互动。

学习

通过百余场讲座、动手实验和研究海报获取宝贵见解和实践培训。

发现

了解 GPU 技术如何为深度学习等重要领域带来重大突破，描绘最新 AI 世界观。

创新

共同探索改变世界的颠覆性创新，定义未来。

立即注册，扫码立享 75 折邀请优惠购票
或使用我的优惠邀请码：NVDAVIDWU
前往 www.nvidia.cn/gtc/ 完成报名


Q & A


NVIDIA