

Day 5

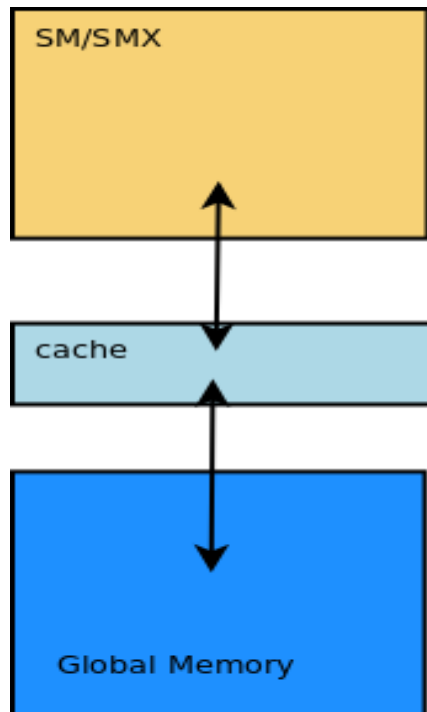
- Revise the idea of coalescing global memory access
- CUDA Visual Profiler

Dr. Tuan-Tu Tran
trant@uni-mainz.de

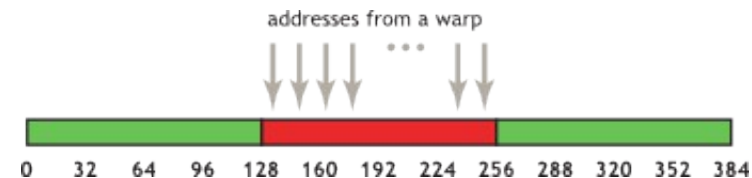
Part 1

Revise the idea of coalescing global memory access

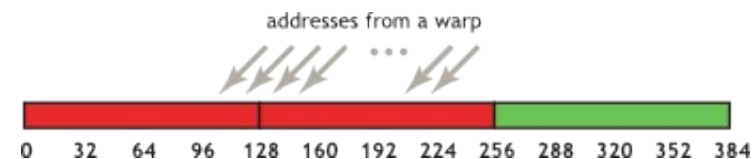
When a SM/SMX accesses to a global memory



Cache line: 128 Bytes



1 cache access transaction



2 cache access transaction

More details can be found in (*in the relation with the cache and the capability of the device*):

NVIDIA, CUDA C Best Practice Guide,

<http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#coalesced-access-global-memory>

Efficient implementation

- To read N integer from global memory to shared memory:

which one is better ?

Method 1

```
__global__ void kernel(int* gData, int N)
{
    extern int sData[];
    int nThread = blockDim.x;
    int tid = threadIdx.x;
    int threadLoad = N/nThread;
    for(int i=0;i<threadLoad;i++)
    {
        int index = tid + i*nThread;
        if(index < N) sData[index] = gData[index];
    }
}
```

Method 2

```
__global__ void kernel(int* gData, int N)
{
    extern int sData[];
    int nThread = blockDim.x;
    int tid = threadIdx.x;
    for(int i=tid;i<N;i+=nThread)
    {
        Sdata[i] = gData[i];
    }
}
```

Efficient implementation

- To read N integer from global memory to shared memory:

Ex: $N = 64$, $nThread = 32$

Method 1: 4 cache transactions

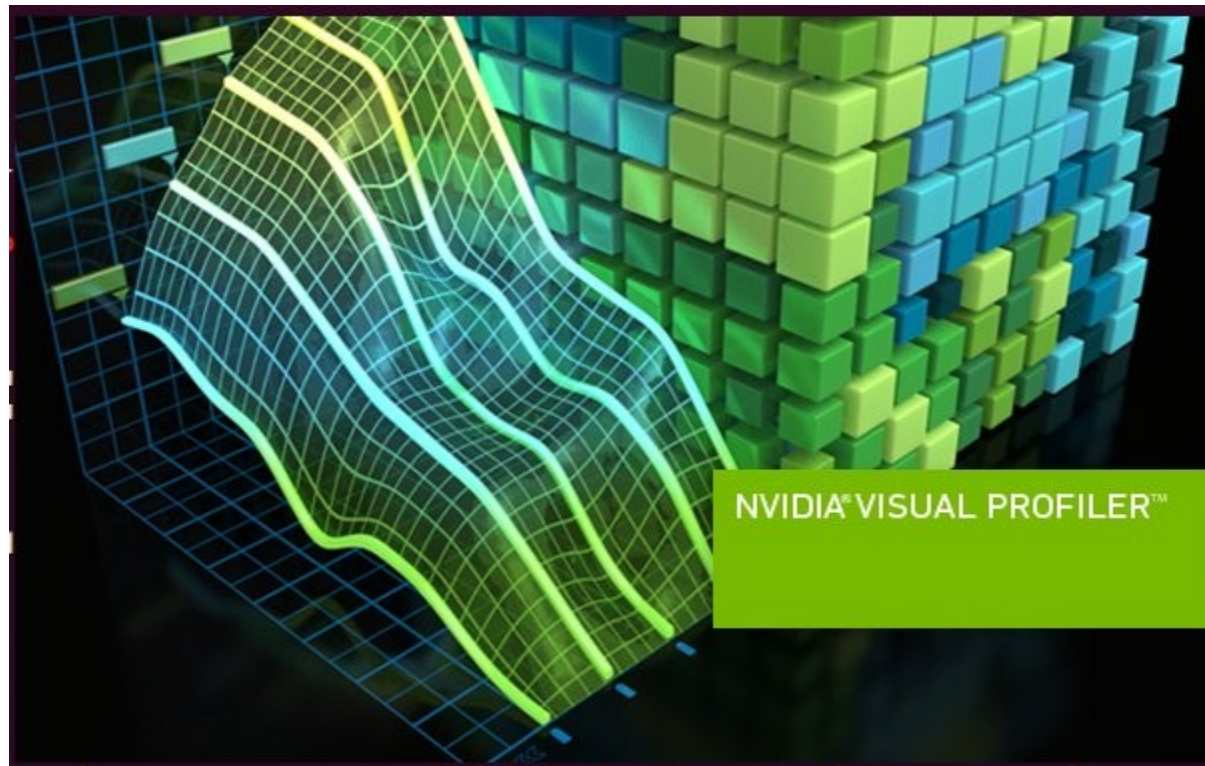
```
__global__ void kernel(int* gData, int N)
{
    extern int sData[];
    int nThread = blockDim.x;
    int tid = threadIdx.x;
    int threadLoad = N/nThread;
    for(int i=0; i<threadLoad; i++)
    {
        int index = tid + i*nThread;
        if(index < N) sData[index] = gData[index];
    }
}
```

Method 2: 2 cache transactions

```
__global__ void kernel(int* gData, int N)
{
    extern int sData[];
    int nThread = blockDim.x;
    int tid = threadIdx.x;
    for(int i=tid; i<N; i+=nThread)
    {
        sData[i] = gData[i];
    }
}
```

Part 2

CUDA Visual Profiler



NVIDIA Visual Profiler

A power tool to examine and analyze CUDA applications.

The profiler is included in the NVIDIA Software Development Kit (SDK) and is installed in the computed node of Mogon.

- By remote access, you can launch the NVIDIA Visual Profiler:
 - With graphical interface: remote access with the X server forwarding (ex: `ssh -X ...`), type: `nvvp`
 - With command line: `nvprof`.
Ex: `nvprof --print-gpu-trace ./cudaReduction -i 100MInt.dat -n 10000000`


- More details can be found in:

NVDA Profiler User's Guide,

<http://docs.nvidia.com/cuda/profiler-users-guide/>

NVIDIA Tools for CUDA

<https://developer.nvidia.com/performance-analysis-tools>




NVIDIA CUDA ZONE Getting Started Downloads Training Ecosystem Register Now Login

PERFORMANCE ANALYSIS TOOLS

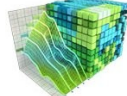
Understanding how your application is using the GPU is crucial to identifying opportunities for performance optimization. Performance analysis tools are also great for identifying performance bottlenecks in your CPU code that can be eliminated by moving computationally intensive algorithms to the GPU.

[Home](#) > [CUDA ZONE](#) > [Tools & Ecosystem](#) > Performance Analysis Tools



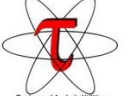
NVIDIA® Nsight™

The ultimate development platform for heterogeneous computing. Work with powerful debugging and profiling tools, optimize the performance of your CPU and GPU code. Find out about the Eclipse Edition and the graphics debugging enabled Visual Studio Edition.




NVIDIA Visual Profiler

This is a cross-platform performance profiling tool that delivers developers vital feedback for optimizing CUDA C/C++ applications. First introduced in 2008, Visual Profiler supports all CUDA capable NVIDIA GPUs shipped since 2006 on Linux, Mac OS X, and Windows.




TAU Performance System®

This is a profiling and tracing toolkit for performance analysis of hybrid parallel programs written in CUDA, and pyCUDA., and HMPP.




VampirTrace

A performance monitor which comes with CUDA, and PyCUDA support to give detailed insight into the runtime behavior of accelerators. Enables extensive performance analysis and optimization of hybrid programs.



The PAPI CUDA Component

A hardware performance counter measurement technology for the NVIDIA CUDA platform which provides access to the hardware counters inside the GPU. Provides detailed performance counter information regarding the execution of



The NVIDIA CUDA Profiling Tools Interface

(CUPTI) provides performance analysis tools with detailed information about GPU usage in a system. CUPTI is used by performance analysis tools such as the NVIDIA Visual Profiler, TAU and



JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Question ?