S7445 - WHAT THE PROFILER IS TELLING YOU

OPTIMIZING WHOLE APPLICATION PERFORMANCE

Mathias Wagner, Jakob Progsch
GTC 2017
BEFORE YOU START
The five steps to enlightenment

1. Know your application
   • What does it compute? How is it parallelized? What final performance is expected?

2. Know your hardware
   • What are the target machines and how many? Machine-specific optimizations okay?

3. Know your tools
   • Strengths and weaknesses of each tool? Learn how to use them.

4. Know your process
   • Performance optimization is a constant learning process

5. Make it so!
PERFORMANCE OPTIMIZATION
What exactly is the performance bottleneck?

You might have a feeling where your application spends most of it’s time ...

... but a more analytic approach might be better
WHOLE APPLICATION PERFORMANCE
Need to consider all components and their interaction

You might spend hours on optimizing GPU kernels ...
... but at the end your application is still not really faster

Introduce imbalance in your system
Amdahl’s law applies

Kernel performance in companion talk: S7444 (next)

Image: https://www.flickr.com/photos/jurvetson/480227362
ASSESS PERFORMANCE

Tools required

Various tools available depending on your requirements

Different levels of sophistication (amount of information scales with effort ...)

Simple wallclock time

Timers build in your code

Simple CPU profilers (gprof)

GPU Timelines and profiles and details: CUDA profiling tools
nvprof, NVIDIA Visual Profiler (NVVP), NVIDIA Nsight Vistual Studio Edition

MPI, OpenMP, CPU Details (3rd party tools)
### 3rd Party Profiling Tools

(Without even trying to be complete)

<table>
<thead>
<tr>
<th>TAU PERFORMANCE SYSTEM ®</th>
<th>VAMPIRTRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="TAU Performance System" /></td>
<td><img src="image2.png" alt="VAMPIRTRACE" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAPI CUDA COMPONENT</th>
<th>HPC TOOLKIT</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="PAPI" /></td>
<td><img src="image4.png" alt="HPC Toolkit" /></td>
</tr>
</tbody>
</table>
You should get an idea about how to assess whole application performance and identify some bottlenecks.

There are way to many potential bottlenecks to provide a cook book.
Introduction
HPGMG as sample application
Timeline
Data transfer
Multi-GPU (MPI)
Multi-grid solves elliptic PDEs (Ax=b) using a hierarchical approach. Solution to hard problem is expressed as solution to an easier problem. Accelerates iterative method and provides $O(N)$ complexity.
High-Performance Geometric Multi-Grid

Lawrence Berkeley National Laboratory

FVM and FEM variants, we focus on FVM

Proxy AMR and Low Mach Combustion codes

Used in Top500 benchmarking

http://crd.lbl.gov/departments/computer-science/PAR/research/hpgmg/
https://bitbucket.org/nsakharnykh/hpgmg-cuda
HPGMG implements F-cycle which has better convergence rate than V-cycle

Poisson or Helmholtz operator using 2\textsuperscript{nd} or 4\textsuperscript{th} order discretization
FIRST LOOK AT THE PERFORMANCE
RUN THE APPLICATION

You might have a feeling where your application spends most of it’s time ...

... but a more analytic approach might be better
PROFILE IT
On the CPU side

Tracing using SCORE-P
GUI: CUBE
THE PROFILER WINDOW

NVVP as one possible tool to display timelines
IF REQ’D: REMOTE PROFILING

Application you want to profile for might not run locally

various approaches to remote profiling

A. Run profiler on remote machine and use remote desktop (X11 forwarding, NX, VNC, ...)

B. Collect data using command line profiler `nvprof` and view your local workstation

```
# generate time line
nvprof -o myprofile.timeline ./a.out
# collect data needed for guided analysis
nvprof -o myprofile.analysis --analysis-metrics ./a.out
# custom selection of metrics for detailed investigations
nvprof -o myprofile.metrics --metrics <...> ./a.out
```

C. Use remote connection feature to create a new session
CREATE A REMOTE CONNECTION
TIMELINE
Only shows GPU activity...
NVTX MARKUP
NVIDIA Tools Extension

NVVP by default only shows GPU activity on timelines

Markup can be used to mark regions with CPU activity

Also useful to group phases of your application for easier navigation

Annotate your code

Link against libnvToolsExt
#include "nvToolsExt.h"

... void init_host_data( int n, double * x ) {
    nvtxRangePushA("init_host_data");
    //initialize x on host
    ...
    nvtxRangePop();
}
NVTX MARKUP
Simplify use of NVTX

Use macros: PUSH_RANGE(name, cid), POP_RANGE

Use C++ tracer class
Exploit compiler instrumentation

Details:
CPU TIMELINE

One single solve
... PORTING TO GPUS ...
GPU TIMELINE
GPU TIMELINE
Shows when GPU Kernels run
CAN WE DO BETTER?
HYBRID IMPLEMENTATION
Take advantage of both architectures

Fine levels are executed on throughput-optimized processors (GPU)
Coarse levels are executed on latency-optimized processors (CPU)
HYBRID TIMELINE
HYBRID TIMELINE

For each level: Decide whether to run on the CPU or GPU

Naïve performance estimate:

GPU Time Level 0 - GPU Time Level X + CPU Time Level X
HYBRID TIMELINE

For each level: Decide whether to run on the CPU or GPU

Naïve estimate:

GPU Time Level 0 - GPU Time Level X + CPU Time Level X

81.467 ms - 3.211 ms + 0.419 ms = 78.765 ms
TAKEAWAYS

**CPU PROFILING**
- Get an estimate of your hotspots
- Profile might not be detailed enough (sum, avg, max)
- Useful first estimate

**TIMELINE**
- See what is going on
- Information on each call
- NVTX markup for CPU, grouping
- May also show voids, dependencies, ...
- Estimate speedups

**HYBRID**
- Strong GPU and Strong CPU
- Use both
- Throughput -> GPU
- Latency -> CPU
DATA MIGRATION
MEMORY MANAGEMENT
Using Unified Memory

No changes to data structures
No explicit data movements
Single pointer for CPU and GPU data

Use `cudaMallocManaged` for allocations
UNIFIED MEMORY
Simplified GPU programming

Minimal modifications to the original code:

(1) `malloc` replaced with `cudaMallocManaged` for levels accessed by GPU

(2) Invoke CUDA kernel if level size is greater than threshold

```c
void smooth(level_type *level,...){
...
if(level->use_cuda) {
    // run on GPU
    cuda_cheby_smooth(level,...);
}
else {
    // run on CPU
    #pragma omp parallel for
    for(block = 0; block < num_blocks; block++)
        ...
}
}```
PAGE FAULTS
Segmented view
## PAGE FAULTS

### Details

### GPU Page Fault groups

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>13.042 s (13,042.16)</td>
</tr>
<tr>
<td>End</td>
<td>13.042 s (13,042.16)</td>
</tr>
<tr>
<td>Duration</td>
<td>20.963 μs</td>
</tr>
<tr>
<td>Virtual Address</td>
<td>0x3b7d625000</td>
</tr>
<tr>
<td>Memory Access Type</td>
<td>Read</td>
</tr>
<tr>
<td>GPU Page Faults</td>
<td>8</td>
</tr>
<tr>
<td>Process</td>
<td>39075</td>
</tr>
</tbody>
</table>

- **Profile Overview**
  - **Default Domain**
  - **Profiling Overhead**
  - **Unified Memory**
    - **CPU Page Faults**
    - **[0] Tesla P100-PCIE-16GB**
      - **Unified Memory**
        - **Data Migration (Dto...**
        - **GPU Page Faults**
        - **Data Migration (Hto...**
      - **Context 1 (CUDA)**
        - **Memcpy (DtoD)**
**Solution:** allocate the first CPU level with `cudaMallocHost` (zero-copy memory)
PAGE FAULTS
Almost gone
NEW HINTS API IN CUDA 8

Not used here

cudaMemPrefetchAsync(ptr, length, destDevice, stream)

Migrate data to destDevice: overlap with compute
Update page table: much lower overhead than page fault in kernel
Async operation that follows CUDA stream semantics

cudaMemAdvise(ptr, length, advice, device)

Specifies allocation and usage policy for memory region
User can set and unset at any time
CONCURRENCY THROUGH PIPELINING

Use CUDA streams to hide data transfers

Serial

Concurrent - overlap kernel and D2H copy

CUDA streams to hide data transfers
...OPTIMIZE CPU AND GPU (S7444 - UP NEXT)...
HIGHER PERFORMANCE
Optimized GPU Kernels + OpenMP on CPU
REVISITING HYBRID STRATEGY

Time [seconds]

Max grid size on CPU threshold

Unoptimized

Optimized
## TAKEAWAYS

### UNIFIED MEMORY
- No manual data transfers necessary
- Avoid page faults
- Use prefetch and cudaMemAdvise

### CUDAMALLOC
- Familiarize with variants
- CudaMalloc
- CudaMallocHost
- CudaMallocManaged

### RE-ITERATE
- After optimizing kernels revisit your timeline
- Previous assumption might no longer apply
- Hybrid approaches strongly depend on used CPU and GPU
- Bottlenecks shift
DEPENDENCY ANALYSIS
DEPENDENCY ANALYSIS
Easily find the critical parts to optimize

The longest running kernel is not always the most critical optimization target
DEPENDENCY ANALYSIS

Visual Profiler

Unguided Analysis

Generating critical path

Dependency Analysis

Functions on critical path
DEPENDENCY ANALYSIS

Visual profiler

Inbound dependencies:
- Launch copy_kernel
- MemCpy HtoD [sync]

Outbound dependencies:
- MemCpyDtoH [sync]
MULTI-GPU USING MPI
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
PROFILING MPI APPLICATIONS

Using nvprof

Embed MPI rank in output filename, process name, and context name

```
mpirun -np $np nvprof --output-profile profile.%q{OMPI_COMM_WORLD_RANK}.nvvp
```

OpenMPI: OMPI_COMM_WORLD_RANK
MVAPICH2: MV2_COMM_WORLD_RANK

Use the import Wizard
MULTI GPU TIMELINE
MPI ACTIVITY IN NVVP

Use NVTX

MPI provides a MPI Profiling interface (PMPI)

Intercept MPI calls and perform actions before and after the MPI call

Python script to generate necessary wrapper is available

```
python wrap/wrap.py -g -o nvtx_pmpi.c nvtx.w
mpicc -c nvtx_pmpi.c
mpicc ... nvtx_pmpi.o -o myapplication -L$CUDA_HOME/lib64 -lnvToolsExt
```

Details:
MULTI GPU TIMELINE
MPI TRANSFERS
Without CUDA aware MPI
CUDA AWARE MPI

MPI knows about the GPU

Use MPI directly on GPU pointers (no manual copy to host required)

Unified Memory needs explicit support from the CUDA-aware MPI implementation

  Check your MPI implementation for support (OpenMPI >1.8.5, MVAPICH2-GDR > 2.2b)

Unified Memory and regular (non CUDA-aware) MPI

  Requires unmanaged staging buffer

  Regular MPI has no knowledge of Unified Memory
NVIDIA GPUDIRECT™
Peer to Peer Transfers

GPU1
Memory

GPU2
Memory

System
Memory

CPU

Chip
set

IB

PCI-e
NVIDIA GPUDIRECT™

Peer to Peer Transfers

GPU1
Memory

NVLink

GPU2
Memory

System
Memory

CPU

Chip
set

IB

GPU 1

GPU 2

PCI-e
PEER TO PEER

Using a pinned buffer for MPI

Not supported on unified memory buffers

Use pinned memory buffer for MPI

Still staging through host ???
TOPOLOGY

DGX-1

Fat GPU nodes

Multiple CPUs

System memory attached to a single CPU

Multiple GPUs

P2P via NVLink or shared PCIe

Multiple Network (IB) adapter

Without direct connection: staging through host
## TOPOLOGY

Query information using `nvidia-smi topo -m`

<table>
<thead>
<tr>
<th></th>
<th>GPU0</th>
<th>GPU1</th>
<th>GPU2</th>
<th>GPU3</th>
<th>GPU4</th>
<th>GPU5</th>
<th>GPU6</th>
<th>GPU7</th>
<th>mlx5_0</th>
<th>mlx5_2</th>
<th>mlx5_1</th>
<th>mlx5_3</th>
<th>CPU Affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19</td>
</tr>
<tr>
<td>GPU1</td>
<td>NV1</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19</td>
</tr>
<tr>
<td>GPU2</td>
<td>NV1</td>
<td>NV1</td>
<td>X</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>0-19</td>
</tr>
<tr>
<td>GPU3</td>
<td>NV1</td>
<td>NV1</td>
<td>X</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>0-19</td>
<td></td>
</tr>
<tr>
<td>GPU4</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB 20-39</td>
</tr>
<tr>
<td>GPU5</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>20-39</td>
</tr>
<tr>
<td>GPU6</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>NV1</td>
<td>X</td>
<td>NV1</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>20-39</td>
</tr>
<tr>
<td>GPU7</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td></td>
<td>20-39</td>
</tr>
<tr>
<td>mlx5_0</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td></td>
</tr>
<tr>
<td>mlx5_2</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td></td>
<td>0-19</td>
</tr>
<tr>
<td>mlx5_1</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td></td>
<td></td>
<td>0-19</td>
</tr>
<tr>
<td>mlx5_3</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**

- **X** = Self
- **SOC** = Connection traversing PCIe as well as the SMP link between CPU sockets (e.g. QPI)
- **PHB** = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
- **PXB** = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
- **PIX** = Connection traversing a single PCIe switch
- **NV#** = Connection traversing a bonded set of # NVLinks
PEER TO PEER
With directly connected GPUs

No staging through host
Here via NVLink
## TAKEAWAYS

### MARKUP MPI
- Use NVTX wrapper
- Show MPI calls and duration
- More details:
- User 3rd party tools

### CUDA AWARE MPI
- Call MPI on GPU buffer
- Unified memory aware
- Can use D2D copies
- Avoids host staging Direct RDMA to network

### TOPOLOGY
- Multiple CPUs, GPUs, Net?
- Consider options to place your jobs (mpirun, numactl)
- Query topology of your system (hwininfo, nvidia-smi)
SUMMARY
TAKEAWAYS

PROFILE
- Use available tools
- Use as many as needed
- Consider your whole application
- Focus on representative parts / smaller problems
- Check for overhead

OPTIMIZE
- Look for dependencies
- Use streams to overlap work
- Be aware of unified memory page faults
- Use CUDA aware MPI
- Consider topology

LEARN
- Build knowledge
- No optimization cookbook
LEARNING RESOURCES
S7444 - WHAT THE PROFILER IS TELLING YOU: OPTIMIZING GPU KERNELS

3RD PARTY PROFILING
L7115 - PERFORMANCE ANALYSIS [...] SCORE-P and VAMPIR
S7684 - PERFORMANCE ANALYSIS OF CUDA DEEP LEARNING NETWORKS USING TAU
S7573 - DEVELOPING, DEBUGGING, AND OPTIMIZING [...] WITH ALLINEA FORGE

CUDA PROFILING
S7495 - OPTIMIZING APPLICATION PERFORMANCE WITH CUDA PROFILING TOOLS
S7824 - DEVELOPER TOOLS UPDATE IN CUDA 9

MPI AND UNIFIED MEMORY
S7133 - MULTI-GPU PROGRAMMING WITH MPI
L7114 - MULTI GPU PROGRAMMING WITH MPI AND OPENACC
S7142 - MULTI-GPU PROGRAMMING MODELS
S7285 - UNIFIED MEMORY ON THE LATEST GPU ARCHITECTURES

CUDA DOCUMENTATION
Programming Guide, Best Practices, Tuning Guides
Parallel for-all Blog, GTC on-demand
Stackoverflow