Profiling GPU Code

Jeremy Appleyard, February 2016

What is Profiling?

Measuring Performance

- Measuring application performance
  - Usually the aim is to reduce runtime
- Simple profiling:
  - How long does an operation take?
- Advanced profiling:
  - Why does an operation take a long time?

Profiling Workflow

1. Find which parts of the code take time
2. Work out why they take time
3. Optimize
4. GOTO 1.

GPU Performance

Quick overview

- A processor has two key performance limits
  - Floating point throughput (FLOP/s)
    - Peak ~6 TFLOP/s
  - Memory throughput (GB/s)
    - Peak ~300 GB/s (DRAM)
- GPUs also need parallelism
  - This is how they can be so fast

Profiling Tools

General GPU Profiling

From NVIDIA

- nvprof
- NVIDIA Visual profiler
  - Standalone (nvvp)
  - Integrated into Nsight Eclipse Edition (nsight)
- Nsight Visual Studio Edition

Third Party

- Tau Performance System
- VampirTrace
- PAPI CUDA component

In this talk

- We will focus on nvprof and nvvp
- nvprof => NVIDIA Profiler
    - Command line
- nvvp => NVIDIA Visual Profiler
    - GUI based

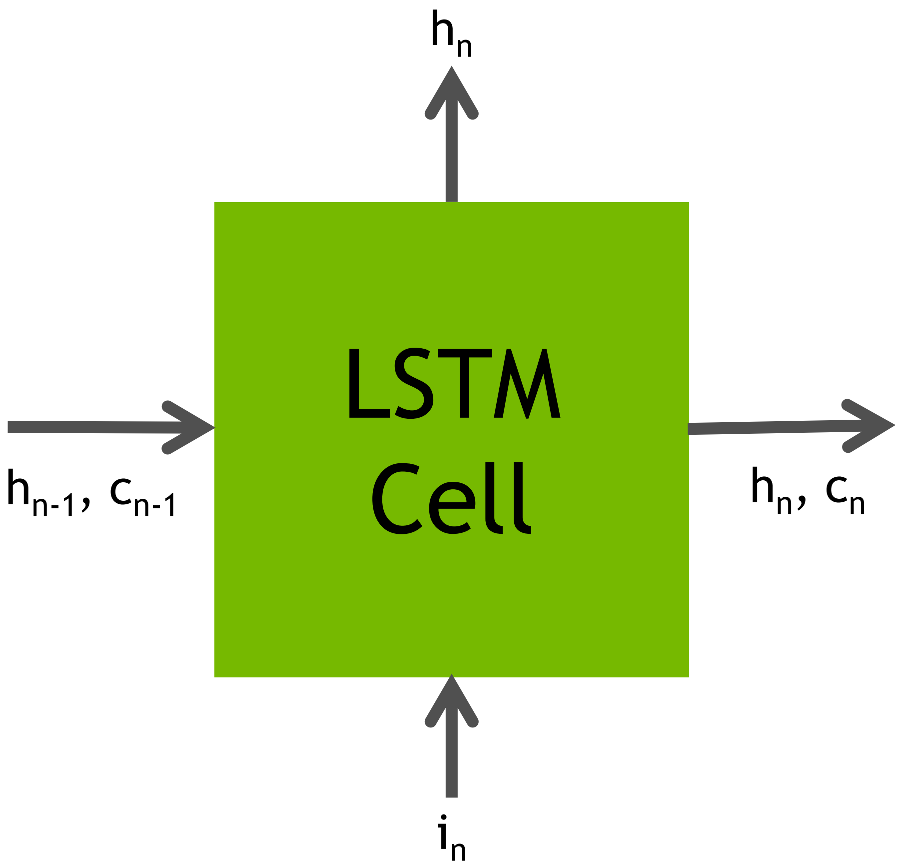
Case Study

Recurrent Neural Network - LSTM

- Uses:
  - Natural language processing
  - Sequences of images (eg. video)
  - Bio/medical
- We will look at optimisation of a single iteration of LSTM

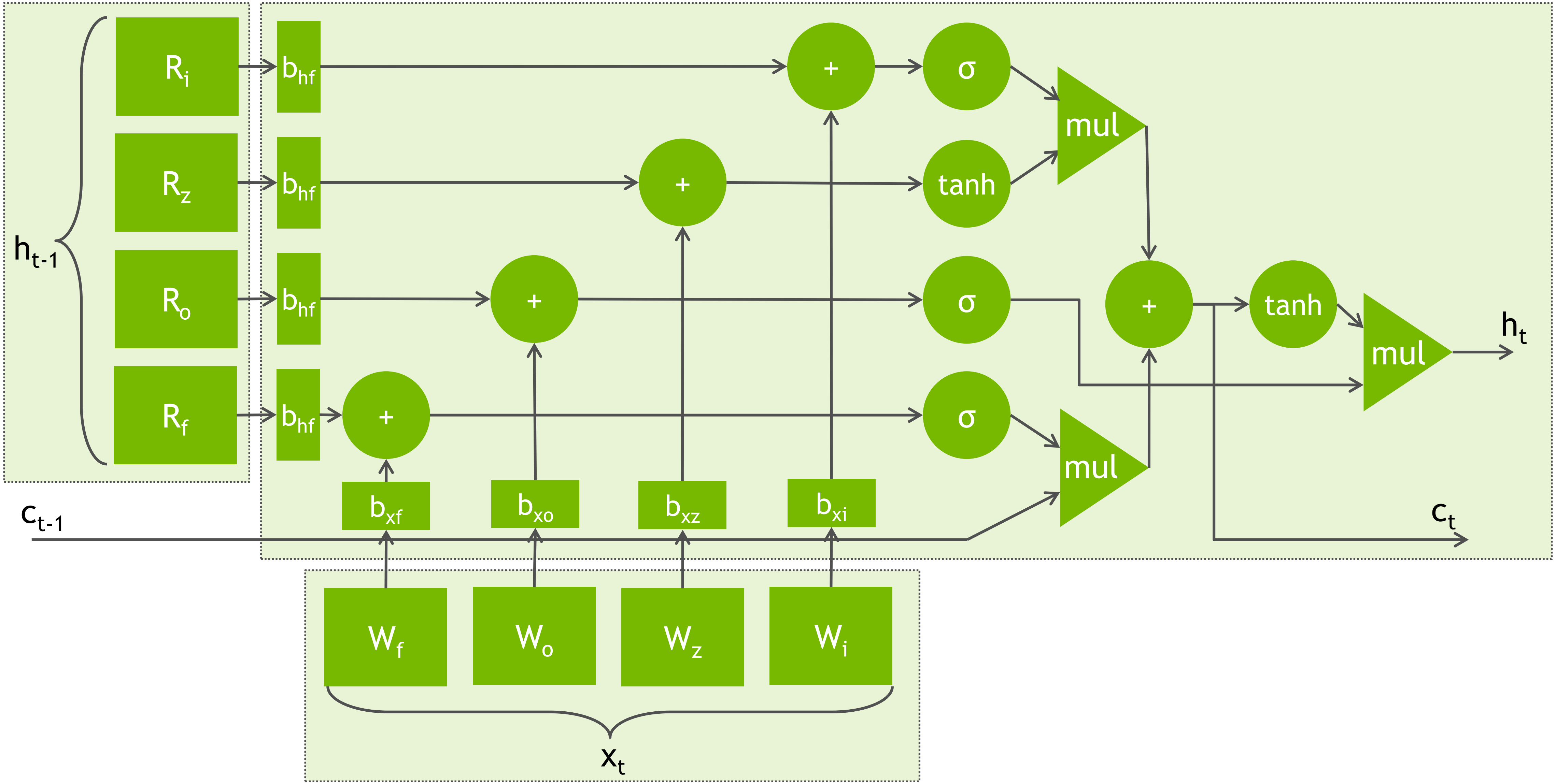
LSTM

Viewed as a black box



- Inputs and outputs are “batched vectors”.
  - ie. A minibatch
- Typical length is 256-2048
- Typical batch size is 32-128

LSTM Details



LSTM Profile

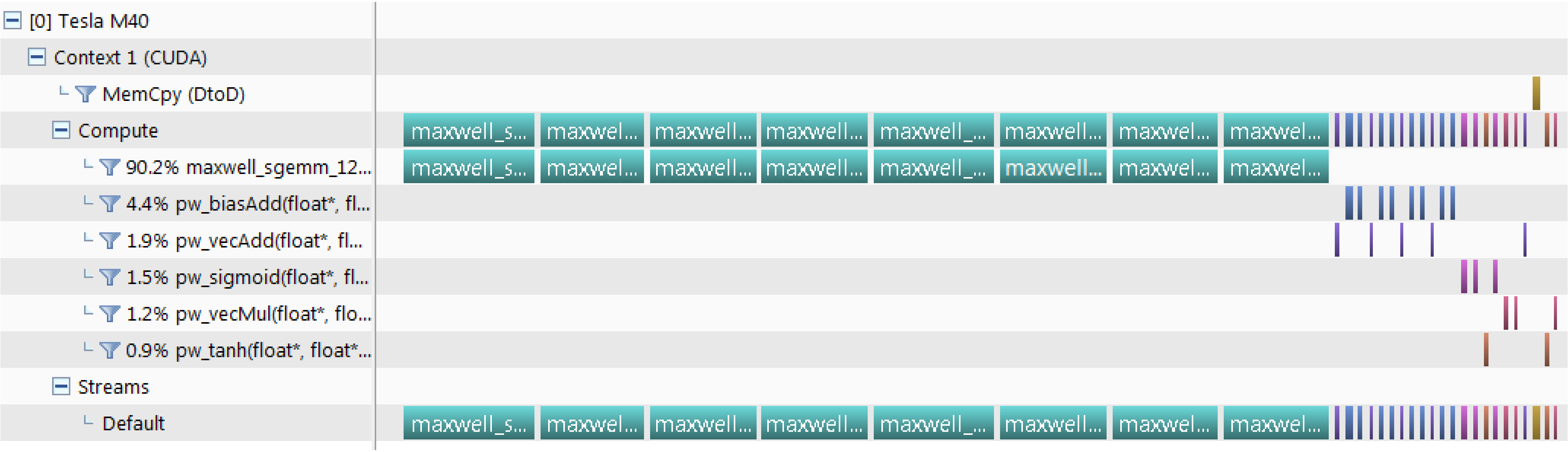
Using nvprof

```
>> nvprof ./RNN 512 64
==6805== NVPROF is profiling process 6805, command: ./RNN 512 64
==6805== Profiling application: ./RNN 512 64
==6805== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 88.46%  512.07us         8  64.009us  60.449us  75.618us  maxwell_sgemm_128x64_tn
  4.26%  24.673us         8  3.0840us  2.9120us  4.1600us  pw_biasAdd(float*, float*, int, int)
  1.93%  11.200us         5  2.2400us  2.0160us  2.9760us  pw_vecAdd(float*, float*, float*, int)
  1.92%  11.136us         3  3.7120us  3.4560us  4.1920us  [CUDA memcpy DtoD]
  1.39%  8.0650us         3  2.6880us  2.3040us  3.4570us  pw_sigmoid(float*, float*, int)
  1.15%  6.6560us         3  2.2180us  1.9840us  2.6560us  pw_vecMul(float*, float*, float*, int)
  0.88%  5.0880us         2  2.5440us  2.3040us  2.7840us  pw_tanh(float*, float*, int)
```

LSTM Profile

Using nvvp

- Can run interactively
- Or use `nvprof -o a.nvp` and import file



SGEMM Performance

Back of the envelope

- SGEMM is a well known operation
- With the inputs chosen each should perform about 33 million floating point operations
- 33 million / 64us = ~516 GFLOPs.
    - GPU can do ~6000 GFLOPs!
- What is wrong?

SGEMM Performance

What is wrong?

- Collect performance metrics:
  - Either via nvprof --analysis-metrics …
  - Or interactively
- A lot of information available
  - Guided analysis helps filter this down
  - Leads me to: “Optimization: Increase the number of blocks executed by the kernel.”
  - Expose more parallelism!

SGEMM Performance

Improvement #1

$$[A_1][h] = [y_1]$$

$$[A_2][h] = [y_2]$$

$$[A_3][h] = [y_3]$$

$$[A_4][h] = [y_4]$$

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} [h] = \begin{bmatrix} \\ y \\ \\ \end{bmatrix}$$

- As our matrix operations share inputs we can combine them

SGEMM Performance

Improvement #1

Before:

Time(%)	Time	Calls	Avg	Min	Max	Name
88.46%	**512.07us**	8	64.009us	60.449us	75.618us	maxwell_sgemm_128x64_tn

After:

Time(%)	Time	Calls	Avg	Min	Max	Name
75.97%	**213.19us**	2	106.59us	104.90us	108.29us	maxwell_sgemm_128x64_tn

SGEMM Performance

Improvement #2

- We are still doing two independent matrix products
  - We can combine them
  - Or compute them simultaneously

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} [h] = \begin{bmatrix} y \end{bmatrix} \qquad \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} [i] = \begin{bmatrix} z \end{bmatrix}$$

SGEMM Performance

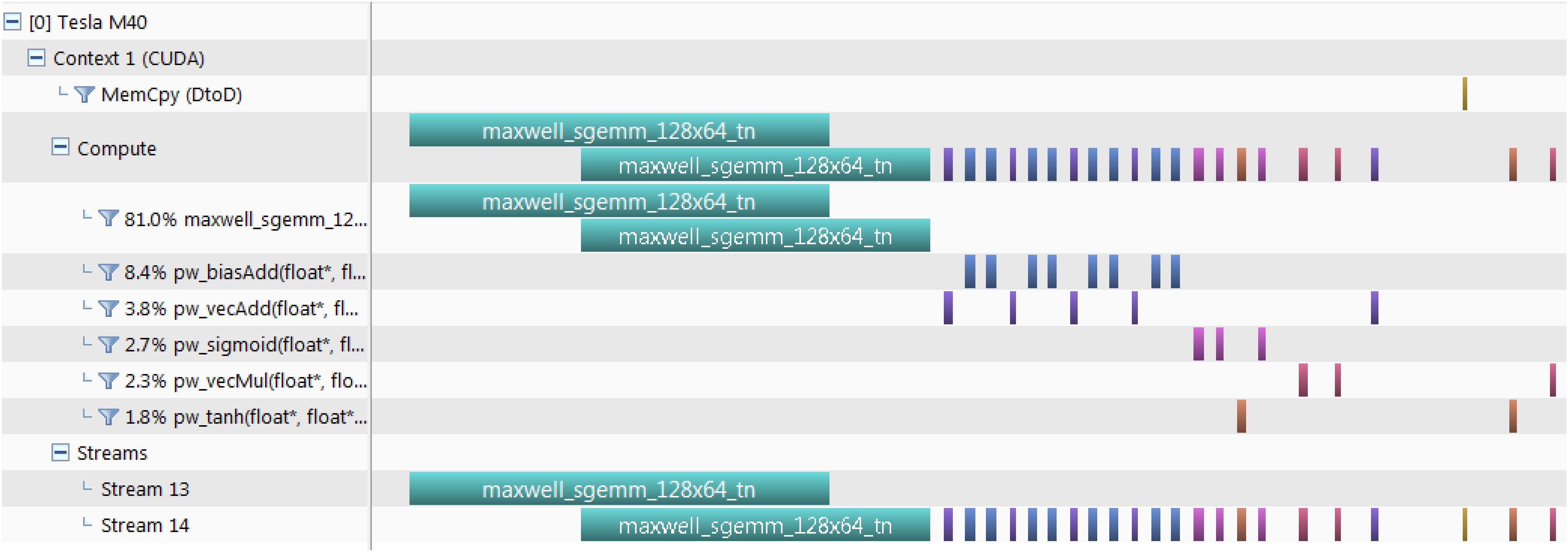
Improvement #2

- We are still doing two independent matrix products
  - We can combine them
  - **Or compute them simultaneously**

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} [h] = \begin{bmatrix} y \end{bmatrix} \qquad \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} [i] = \begin{bmatrix} z \end{bmatrix}$$

SGEMM Performance

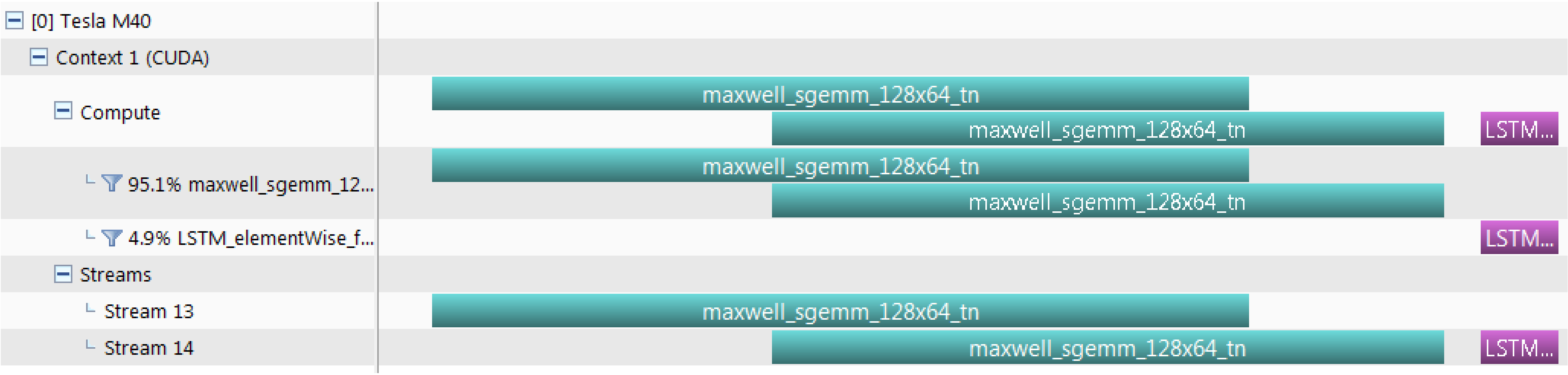
Matrix overlapping



NVIDIA.

Final optimization

Fuse element-wise operations



LSTM

Performance

Optimisation	Runtime	Speedup
Naïve	661us	(1.0x)
Combined matrices	357us	1.9x
Matrix streaming	250us	2.6x
Fused element-wise ops	136us	4.9x

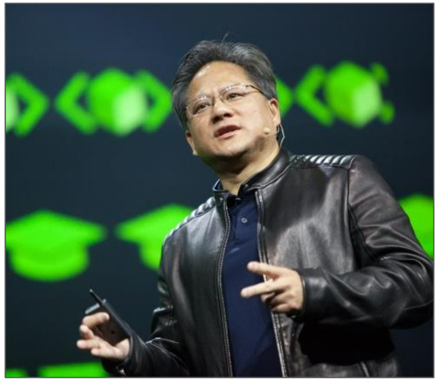
Profiling

5x performance improvement

- Profiling helped to quickly identify the slow parts
- It showed that SGEMM was underusing the GPU
  - This was fixed by exposing more parallelism
- It showed that the pointwise operations were taking a significant proportion of our runtime
  - This was fixed by fusing them

GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley | #GTC16
www.gputechconf.com



CONNECT

Connect with technology experts from NVIDIA and other leading organizations

LEARN

Gain insight and hands-on training through the hundreds of sessions and research posters

DISCOVER

See how GPU technologies are creating amazing breakthroughs in important fields such as deep learning

INNOVATE

Hear about disruptive innovations as early-stage companies and startups present their work

The world's most important event for GPU developers