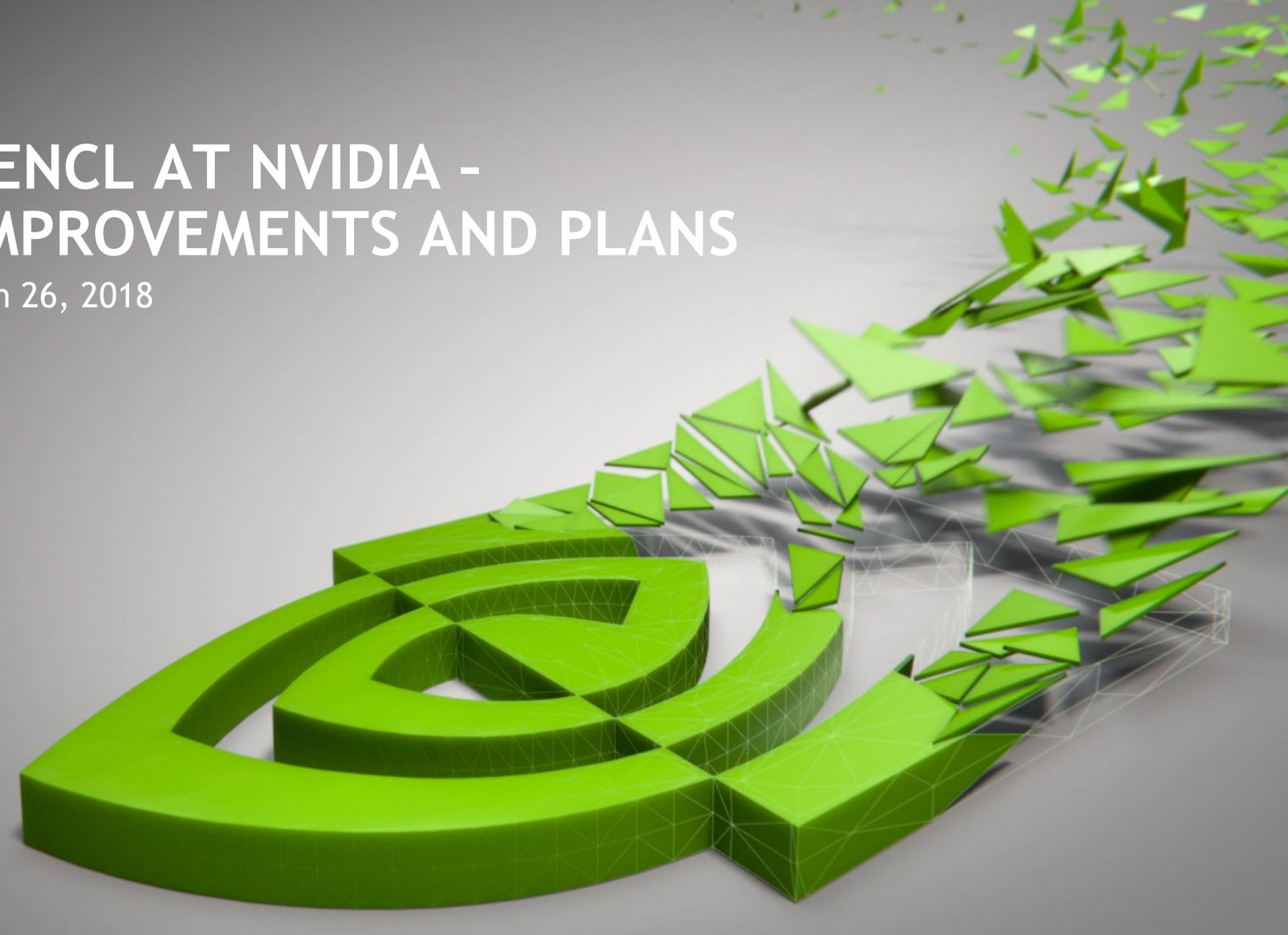
S8837 OPENCL AT NVIDIA – RECENT IMPROVEMENTS AND PLANS

Nikhil Joshi, March 26, 2018

NVIDIA



AGENDA

Power optimizations

Performance tuning

- Data transfer

What's New?

- cl_nv_create_buffer
- MultiGPU improvements

Upcoming

POWER OPTIMIZATIONS

POWER OPTIMIZATION

Existing behaviour

- Work-load patterns vary
  - Bursty vs continuous work-loads
- Driver heuristics
  - Designed for performance, not for power
  - Leads to higher power consumption
  - Run-To-Completion always?

POWER OPTIMIZATIONS

New behaviour

Revamping heuristic to optimize for power

Key goals

- Default behaviour that suits wider use-cases
- Lower CPU and GPU utilization when there is no work
- Potentially finer grained control for addressing specific, unusual cases

*Work in progress, expect production in Q2’18.

DATA TRANSFER

DATA TRANSFER

Perf tuning

Different perf characteristics based on

- Type of host memory (Pinned vs pageable)
- Size of the buffer
- Choice of API (Read/WriteBuffer vs Map/Unmap)

DATA TRANSFER

Type of memory

Pinned/ Page-locked memory

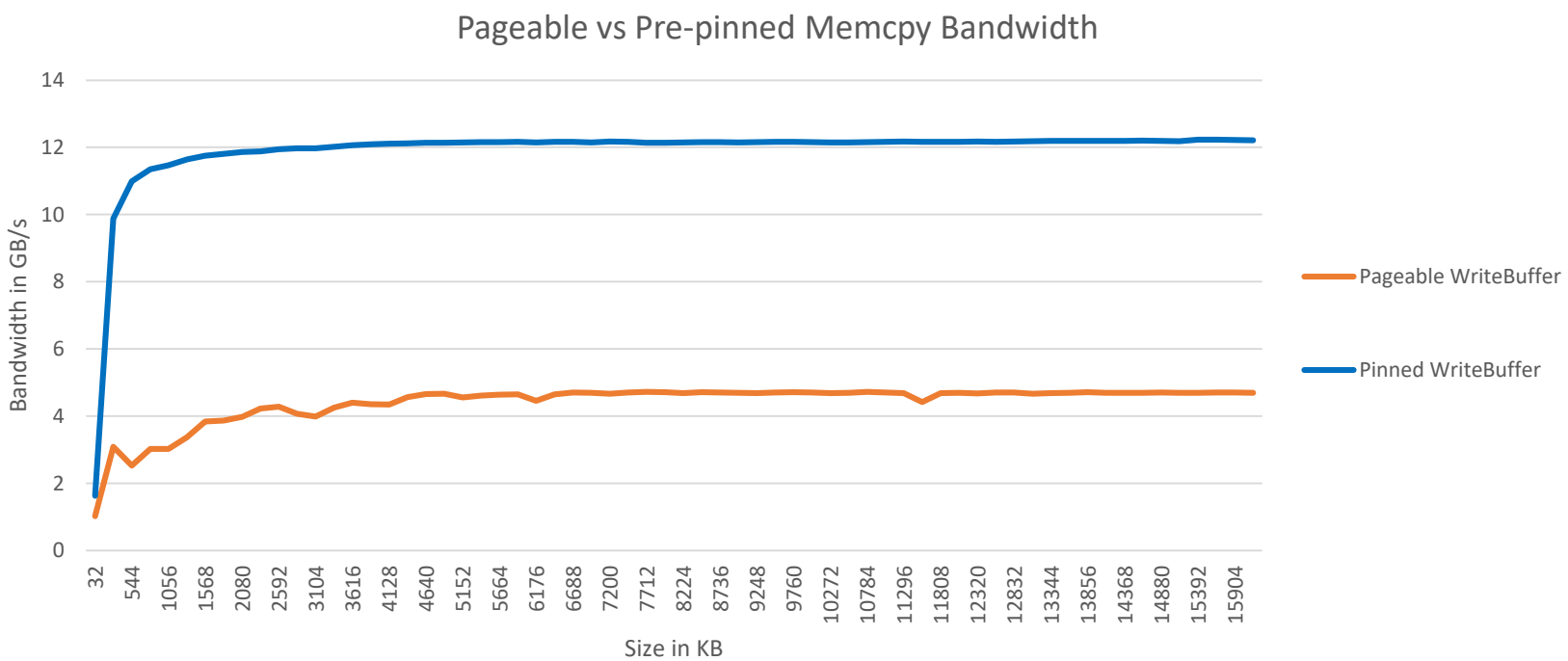
- Guaranteed to be in memory and not swapped out
- Limited by RAM size

Pageable memory

- Typically malloced memory
- Can be swapped out
- Not limited by RAM size, but limited VA space

DATA TRANSFER

Pageable Vs Pre-pinned



DATA TRANSFER

Best Practices

Use pinned memory for fast async copies

- Pinned memcpy 2-3x faster than pageable memcpy.
- Can be truly async, pageable memcpy may not.
- Power efficient.

Use pinned memory judiciously

- Scarce resource.
- Overuse may affect system stability.

DATA TRANSFER

Best Practices (..contd)

Prefer Map/Unmap over Read/WriteBuffer

- Read/WriteBuffer requires memory to be allocated and pre-pinned.
- Map/Unmap internally allocates pinned memory.
- Pinned memcpy bandwidth close to the peak performance.

DATA TRANSFER

Best Practices (..contd)

Avoid small-sized (<200 KB) copies

- Small copies have poor bandwidth.
- DMA setup overhead larger than actual copy cost.
- Can not saturate PCI-E bandwidth.

Prefer larger sizes

- Better bandwidth
- Fewer copies

MEMORY OWNERSHIP AND PLACEMENT

ALLOCATING MEMORY

What the OpenCL spec says (and does not)

CL_MEM_ALLOC_HOST_PTR

"This flag specifies that the application wants the OpenCL implementation to allocate memory from host accessible memory"

Spec DOES NOT specify

Type of host memory (pinned vs pageable)

Memory placement (host vs device)

ALLOCATING PINNED MEMORY

Existing way on NVIDIA

Use CL_MEM_ALLOC_HOST_PTR

```
cl_mem mem = clCreateBuffer(context,
                            CL_MEM_ALLOC_HOST_PTR,
                            size, NULL, NULL);

void* host_ptr = clEnqueueMapBuffer(command_queue, mem, …);
```

ALLOCATING MEMORY

Existing way - limitations

Implementation defined behavior

- not guaranteed to be consistent across platforms

Does not guarantee pinned host memory

Memory placement close to GPU

Designed for performance

Allocations limited by GPU RAM, while CPU RAM can be still available

CL_NV_CREATE_BUFFER

New extension for memory allocation

New extension with new set of flags to control

```
cl_mem clCreateBufferNV (cl_context context,
                         cl_mem_flags flags,
                         cl_mem_flags_NV flags_NV,
                         size_t size,
                         void *host_ptr, cl_int *errcode_ret);
```

cl_mem_flags_NV

- **CL_MEM_PINNED_NV**
- **CL_MEM_LOCATION_HOST_NV**

*Available for production in Q2’18.

CL_NV_CREATE_BUFFER

Allocating pinned memory

CL_MEM_PINNED_NV

+ Guaranteed pinned host memory on mapping
+ Fast async data copies
+ Kernel access through GPU memory, hence faster
- Scarce resource, subject to availability

CL_NV_CREATE_BUFFER

Allocating GPU accessible host memory

CL_MEM_LOCATION_HOST_NV

+ Places memory close to CPU
+ Saves GPU memory
+ Suitable for sparse kernel accesses
- GPU access through host memory, hence slower

CL_NV_CREATE_BUFFER

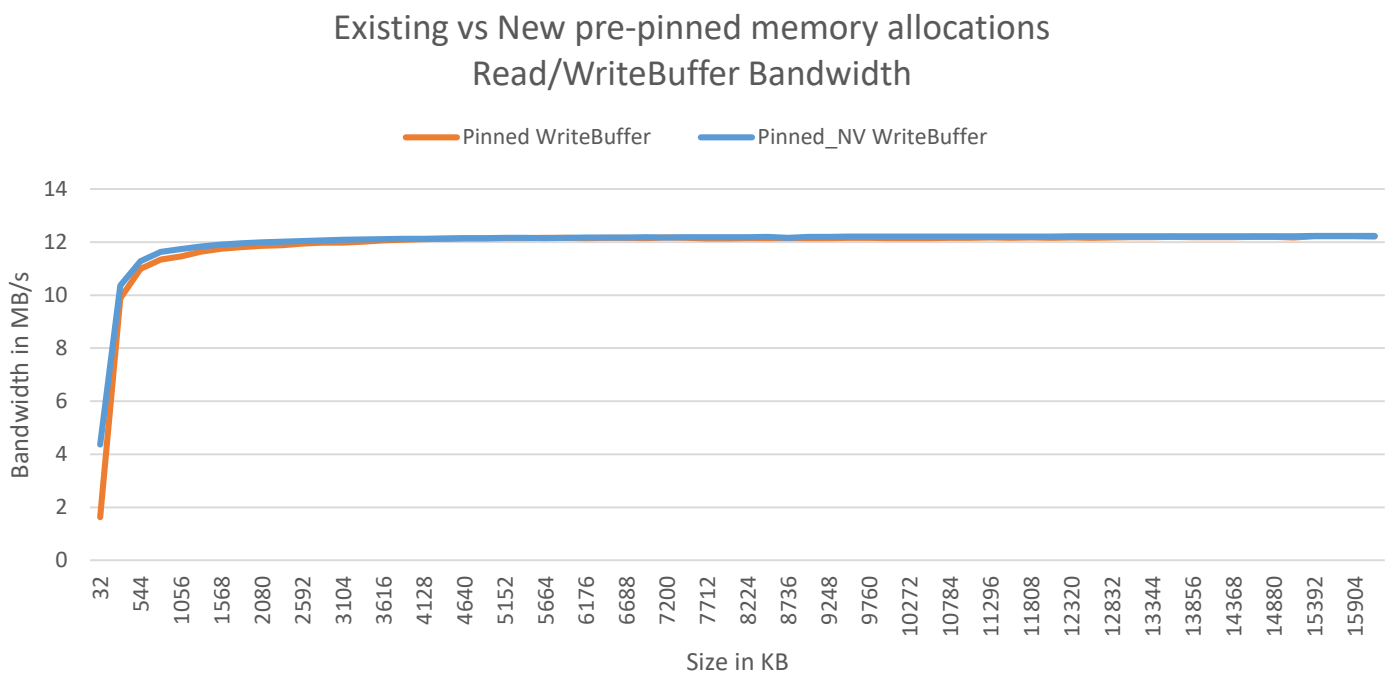
Performance

Pinned Memory performance

- Same as existing pinned memory performance.
- Read/WriteBuffer and Map/Unmap perf at peak.

CL_NV_CREATE_BUFFER

Easier to use, same performance



MULTI-GPU IMPROVEMENTS

PINNED MEMORY ACCESS

MultiGPU use-cases - existing way

Mapping buffer on a command queue

- Gives optimal performance on that device.

Mapping on one and using on different device

- Works, but incurs performance penalty.

To get the best performance,

- Need to map buffers on each device separately.
- Not suitable for multiGPU use-cases.

PINNED MEMORY ACCESS

MultiGPU use-cases - New way

No need to map on each device separately

Mapping on one and using it on another device

- As optimal as using it on the same device

Note - Need to ensure event dependencies as before

*Available for production in Q2’18

PINNED MEMORY ACCESS

Existing vs New way

Using pinned mappings – Existing way

```
ptr = clEnqueueMapBuffer(cq1, buff, ….)
// Use ptr on host
clEnqueueWriteBuffer(cq1, buff2, …, ptr, …)
clEnqueueUnmapMemObject(cq1, ..., buff, …, &ev1)
ptr2 = clEnqueueMapBuffer(cq2, buff, …., &ev1, …)
// Use ptr2
clEnqueueWriteBuffer(cq2, buff3, …, ptr2 …)
```

Using pinned mappings - New way

```
ptr = clEnqueueMapBuffer(cq1, buff, ….)
// Use ptr on host
clEnqueueWriteBuffer(cq1, buff2, …, ptr, …)


// No need to map on cq2. Use ptr on cq2
clEnqueueWriteBuffer(cq2, buff3, …, ptr, …, &ev1, ..)
```

SUMMARY

OPENCL PERFORMANCE

Multi-year effort

Improvements over the years

- Robust and more efficient CL-GL Interop
- Copy-Compute overlap using Map/Unmap
- Preview subset of 2.0 features

* See references for previous talks on driver/runtime improvements.

OPENCL PERFORMANCE

Continued effort

Improvements this year

- cl_create_buffer_nv extension for explicit control of allocation attributes
- Improvements in multiGPU use-cases wrt pinned memory access

UPCOMING

UPCOMING

Power Optimizations

Improve pageable memcpy performance

Improve multiGPU, multi-command queue use-cases.

Your use case? – Let’s talk offline

- nikhilj@nvidia.com

QUESTIONS ??

PREVIOUS TALKS

Focused on Kernel performance

“Better Than All the Rest: Finding Max-Performance GPU Kernels Using Auto-Tuning”

by Cedric Nugteren (SURFsara HPC centre)

“Auto-Tuning OpenCL Matrix-Multiplication: K40 versus K80”

by Cedric Nugteren (SURFsara)

PREVIOUS TALKS

Focused on Applications

“Using OpenCL for Performance-Portable, Hardware-Agnostic, Cross-Platform Video Processing”

by Dennis Adams (Sony Creative Software Inc.)

“Boosting Image Processing Performance in Adobe Photoshop with GPGPU Technology”
by Joseph Hsieh (Adobe)

PREVIOUS TALKS

Driver/Runtime performance

“Performance Considerations for OpenCL on NVIDIA GPUs”

by Karthik Raghavan Ravi, GTC2016

“OpenCL at NVIDIA – Best Practices, Learnings and Plans ”

by Karthik Raghavan Ravi, GTC2017

THANK YOU !!