USING AI TO ACCELERATE YOUR GAME (PART 2)

Stuart Schaefer, Microsoft
Yury Uralsky, NVIDIA
Daniel Kennett, Microsoft
Applications of ML to make games more real are rapidly becoming more practical.

Windows Machine Learning allows developers to utilize trained models for inference on AI capable silicon running with Windows.

Create ML models with training frameworks, convert to ONNX, and integrate Windows ML in your game workflow.

ML Superscaling vs Bilinear Upscaling
**WINDOWS ML ARCHITECTURE**

WinML API

- Win32 & WinRT Layers
- Converts images to Tensor Resources
- Available on all Windows editions in 2018

Inference Engine

- Model & Device resource management
- Loads and compiles operator kernels
- Execute dataflow graph

Device Layer

- CPU instruction optimizations up to AVX-512
- DirectML generates DX12 Compute shaders
DIRECTML

ML model defines dataflow graph of mathematical operators

Evaluation is computationally expensive and highly parallelizable

DirectML operators defined using DX12 Compute and HLSL

Baseline for breadth platform support, end-to-end use of GPU

Enable the Inferencing Engine to leverage non-CPU silicon

Provide strong hardware platform coherency to unlock performance

DirectML manages interaction with D3D resource model - PSO, CommandLists, ...

MetaCommands - IHV accelerated replacement for graph operators
WINDOWS ML ACCELERATION
MACHINE LEARNING MODELS
Can be viewed as programs

\[ \text{res} = \text{sqr}(a \times 10) + b - w; \]
MACHINE LEARNING MODELS
Can be viewed as programs

\[
\text{res} = \text{FC}(\text{ReLU}(\text{Conv2d}(a, f)) + b, w);
\]
TENSORS
Multi-dimensional arrays of scalars

feature (C)  width (W)

Height (H)

R  G  B

R  G  B
PROGRAM-CENTRIC VIEW

Of machine learning

Machine learning operators → low-level machine instructions
Tensors → basic datatypes
WinML graph optimizer → code compiler

Want to define a tensor machine ISA
D3D12 METACOMMANDS
METACOMMANDS

Tensor machine ISA
METACOMMANDS

Tensor machine ISA

Fully connected metacommand
METACOMMANDS
Tensor machine ISA

Element-wise metacommand

- conv 2d
- ReLU
- tmp1
- tmp2
- tmp3
- FC
- w
- b
- res
METACOMMANDS
Tensor machine ISA

Convolution + activation metacommand
METACOMMANDS

General definition

Meta Command

Abstracts an optimized machine function
**METACOMMANDS**

**General definition**

Input 0 → Meta Command → Scratch space

- **Persistent meta-data**: Initialized at creation time
- **Scratch space**: Read and written during meta command execution

Output 0 → ... → Output M
LIFE OF A METACOMMAND

Metacommmand overview

Enumerate metacommands and their signatures

Query implementation for supported metacommands

Create metacommmand

Create metacommmand object instance

Get required resource sizes

Query memory footprint requirements for parameters and scratch space
LIFE OF A METACOMMAND

Metacommand overview

Initialize metacommand

Initialize persistent scratch resources of the metacommand

Insert into the command list

Schedule metacommand instance for execution on the command list
### METACOMMAND PARAMETERS

Specified in memory structures

<table>
<thead>
<tr>
<th>Per input, output tensor</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>GPUVIRTUALADDRESS</td>
</tr>
<tr>
<td>Element type</td>
<td>FP32 or FP16</td>
</tr>
<tr>
<td>Dimensions</td>
<td>{batch, width, height, depth}</td>
</tr>
<tr>
<td>Memory layout</td>
<td>NCHW or hardware native</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Per metacommand instance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scratch</td>
<td>GPUVIRTUALADDRESS</td>
</tr>
<tr>
<td>Persistent meta-data</td>
<td>GPUVIRTUALADDRESS</td>
</tr>
<tr>
<td>Metacommand options</td>
<td>&lt;varies&gt;</td>
</tr>
</tbody>
</table>
METACOMMAND EXAMPLES
FULLY CONNECTED OPERATOR

Each element in the output depends on all elements in the input

\[ y_i = f(\sum_j w_{ij} \cdot x_j) \]
FULLY CONNECTED OPERATOR

Each element in the output depends on all elements in the input

\[ y_i = f\left( \sum_j w_{ij} \cdot x_j \right) \]
FULLY CONNECTED OPERATOR
Each element in the output depends on all elements in the input

\[ y_i = f\left( \sum_j w_{ij} \cdot x_j \right) \]
FULLY CONNECTED OPERATOR

Each element in the output depends on all elements in the input

\[ y_i = f(\sum_j w_{ij} \cdot x_j) \]
FULLY CONNECTED OPERATION AS A MATRIX MULTIPLICATION
FULLY CONNECTED OPERATOR

As matrix multiplication

\[
\begin{pmatrix}
W
\end{pmatrix}
\begin{pmatrix}
X
\end{pmatrix}
= \begin{pmatrix}
W \times X
\end{pmatrix}
\begin{pmatrix}
Y
\end{pmatrix}
\]

\[m = 16\quad n = 12\quad m = 16\]
CONVOLUTIONS ON TENSORS

Convolving input tensor with 3 filter kernels

Input tensor $X [4, 8, 8]$  
3 filter kernels $W [4, 2, 2]$
CONVOLUTIONS ON TENSORS

Applying red filter kernel across entire tensor with a stride of 3

\[ y_j = \sum_i w_i \cdot x_k \]
CONVOLUTIONS ON TENSORS

... Repeat for green

\[ y_j = \sum_i w_i \cdot x_k \]
CONVOLUTIONS ON TENSORS

... Repeat for blue

\[ y_j = \sum_i w_i \cdot x_k \]
CONVOLUTIONS ON TENSORS

Stack them together to form the resulting tensor
CONVOLUTIONS ON TENSORS

Convolving input tensor with 3 filter kernels

Input tensor [4, 8, 8] 3 filter kernels [4, 2, 2] = Output tensor [3, 3, 3]
LOWERING CONVOLUTIONS TO MATRIX MULTIPLICATION
CONVOLUTION AS MATRIX MULTIPLICATION

Forming the filter matrix

$$F = \begin{pmatrix} \text{red} & \text{green} & \text{blue} \\ \text{red} & \text{green} & \text{blue} \\ \text{red} & \text{green} & \text{blue} \end{pmatrix} \quad k = 16 \quad m = 3$$
CONVOLUTION AS MATRIX MULTIPLICATION

Forming the input tensor matrix

\[ X = \begin{pmatrix} \text{Input tensor } X [4, 8, 8] \end{pmatrix} \]

\[ n = 9 \]

\[ k = 16 \]
CONVOLUTION AS MATRIX MULTIPLICATION

Evaluating the matrix product

Output tensor [3, 3, 3]
EFFICIENT MATRIX MULTIPLICATION
MATRIX MULTIPLICATION
MATRIX MULTIPLICATION

A naïve method
MATRIX MULTIPLICATION

A naïve method
MATRIX MULTIPLICATION

A naïve method
MATRIX MULTIPLICATION

A naïve method
MATRIX MULTIPLICATION

A naïve method
MATRICES MULTIPLICATION

A naïve method
TILED MATRIX MULTIPLICATION

Memory efficient

\[ D = A \cdot B \]
TILED MATRIX MULTIPLICATION

Memory efficient

\[ \mathbf{D} += \mathbf{A} \cdot \mathbf{B} \]
TILED MATRIX MULTIPLICATION

Memory efficient

\[
D += A \cdot B
\]
TILED MATRIX MULTIPLICATION

Memory efficient

\[ D += A \cdot B \]
TILED MATRIX MULTIPLICATION

Memory efficient, hierarchical

\[ D += A \cdot B \]
VOLTA TENSOR CORES
VOLTA SM TENSOR CORES
Dedicated hardware datapaths for machine learning acceleration

8 Tensor cores per SM
512 FMA operations per clock
Mixed precision operation
110 TFLOPs peak on TitanV
TENSOR CORE OPERATION
A fixed-size matrix multiply-add

\[
\begin{pmatrix}
\text{FP16 or FP32} \\
\end{pmatrix}
= \begin{pmatrix}
\text{FP16} \\
\end{pmatrix}
\begin{pmatrix}
\text{FP16} \\
\end{pmatrix}
+ \begin{pmatrix}
\text{FP16 or FP32} \\
\end{pmatrix}
\]
TENSOR CORE OPERATION

Warp-wide Matrix Math

Warp-synchronizing operation
Composed Matrix Multiply and Accumulate
Result distributed across warp
TENSOR CORE PERFORMANCE
Almost an order of magnitude performance increase

cuBLAS Mixed Precision
(FP16 input, FP32 compute)
Matrix Multiply (M=N=K=2048)

9.3x faster
STYLE TRANSFER DEMO
STYLE TRANSFER
Some facts about the demo

16 convolution layers with residual connections

Almost 640 billion floating-point operations per evaluation

80% of peak tensor core performance delivered in most convolution layers

Up to 88 TFLOPs compute density
STYLE TRANSFER PERFORMANCE

NVIDIA TitanV @ 1080p, absolute perf

Frames per second

- Baseline DirectML: 3.4
- FP32 Metacommmands: 8.6
- Tensor-core accelerated metacommmands: 27

Performance enhancements:
- 2.5x increase in frames per second compared to Baseline DirectML
- 3.1x increase in frames per second compared to FP32 Metacommmands
STYLE TRANSFER PERFORMANCE
NVIDIA TitanV @ 1080p, relative speedup

Baseline DirectML: 1x
FP32 Metacommands: 2.5x
Tensor-core accelerated metacommands: 8x
SUMMARY

DirectML implements core machine learning operations in DirectCompute...

Metacommands allow implementations export optimized versions of those operations...

NVIDIA’s HW and SW provides accelerated support for machine learning...

Machine learning is here and can be used in games today...

Windows ML is available now in Windows 10...
GDC ATTENDEES
SAVE 20%

REGISTER AT: www.gputechconf.com

USE CODE: GMGDC

Featuring new gaming track focused on deep learning and AI on Monday, 26 March 2018
CONVOLUTIONS ON TENSORS
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[ y_i = f(\sum_{j=0}^{k-1} w_j \cdot x_{i+j-\frac{k}{2}}) \]
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[ y_i = f \left( \sum_{j=0}^{k-1} w_j \cdot x_{i+j-k/2} \right) \]
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[ y_i = f \left( \sum_{j=0}^{k-1} w_j \cdot x_{i+j-k/2} \right) \]
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[
y_i = f\left(\sum_{j=0}^{k-1} w_j \cdot x_{i+j-k/2}\right)
\]
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[ y_i = f \left( \sum_{j=0}^{k-1} w_j \cdot x_{i+j-k/2} \right) \]
CONVOLUTION OPERATOR

Each element in the output depends on the local neighborhood in the input

\[ y_i = f\left( \sum_{j=0}^{k-1} w_j \cdot x_{i+j-\frac{k}{2}} \right) \]
CONVOLUTION AS MATRIX MULTIPLICATION
METACOMMANDS
Tensor machine ISA

Activation metacommand