
NVIDIA
SW ARCHITECTURAL TRENDS

OVERVIEW

- GPUs are becoming larger and faster as apps become more complex.
  - **CUDA Graphs** allow workflows to be submitted to GPU rather than single operations, to reduce overheads and allow more holistic optimizations.
- Hierarchical parallelism is becoming increasingly important (within and across GPUs)
  - **Cooperative Groups** allow the programmer to map application-level parallelism to the hardware in a flexible and efficient manner.
  - **Multi-GPU programming** techniques are becoming more sophisticated and performant.
- Programming difficulty associated with complex hardware can be alleviated with use of **Unified Memory**. This makes it easier for users to get started with GPUs.
- There is an increasing awareness of the fact that use of **Reduced Precision** is feasible in many cases, allowing improved performance. Hardware and software support continues to evolve.

CUDA GRAPHS

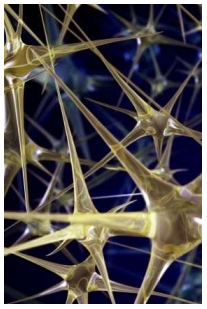
ASYNCHRONOUS GRAPHS

Execution optimization when workflow is known up front

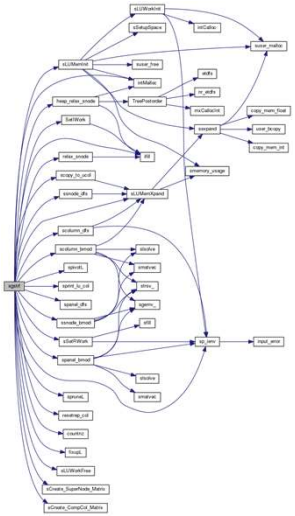
```
// Basic function to test primality.
bool IsPrime( size_t n)
{
      if (n == 2) return true;
      if ((n == 1) || ((n % 2) == 0)) return false;
      size_t iters = (unsigned int)sqrt( (double)n);
      for (size_t i = 3; i <= iters; i+=2) if (n % i == 0) return false;
      return true;
}

// Compute primes from 1 to 100,000,000.
size_t ComputePrimes()
{
      size_t Primes = 0;
      for ( size_t Start = 1; Start <= 100000000; ++Start)
      {
            if ( IsPrime( Start) )
            {
                  ++Primes;
            }
      }
      return Primes;
}
```

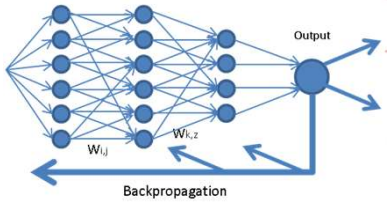
Loop & Function offload



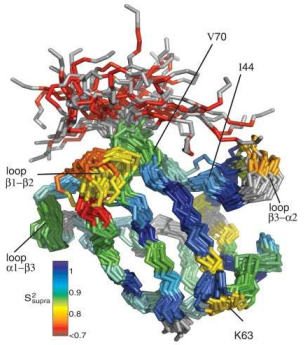
DL Inference



Linear Algebra



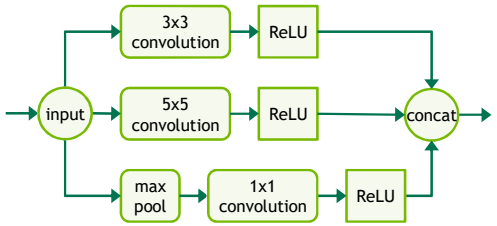
Deep Neural Network Training



HPC Simulation

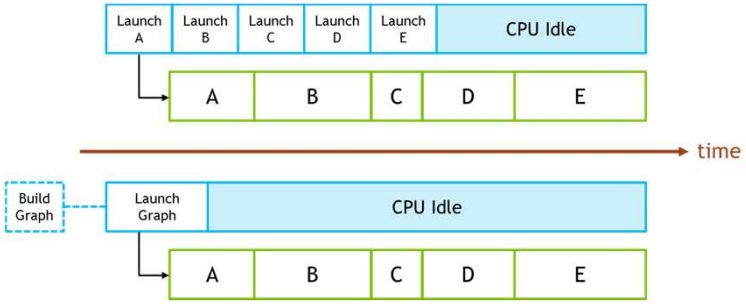
THE GRAPH ADVANTAGE

WHOLE WORKFLOW OPTIMIZATIONS



Seeing all work at once enables
new optimizations in hardware and software

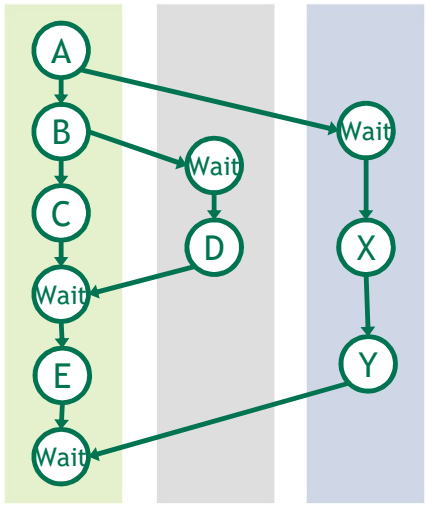
EFFICIENT LAUNCH OF COMPLEX WORK



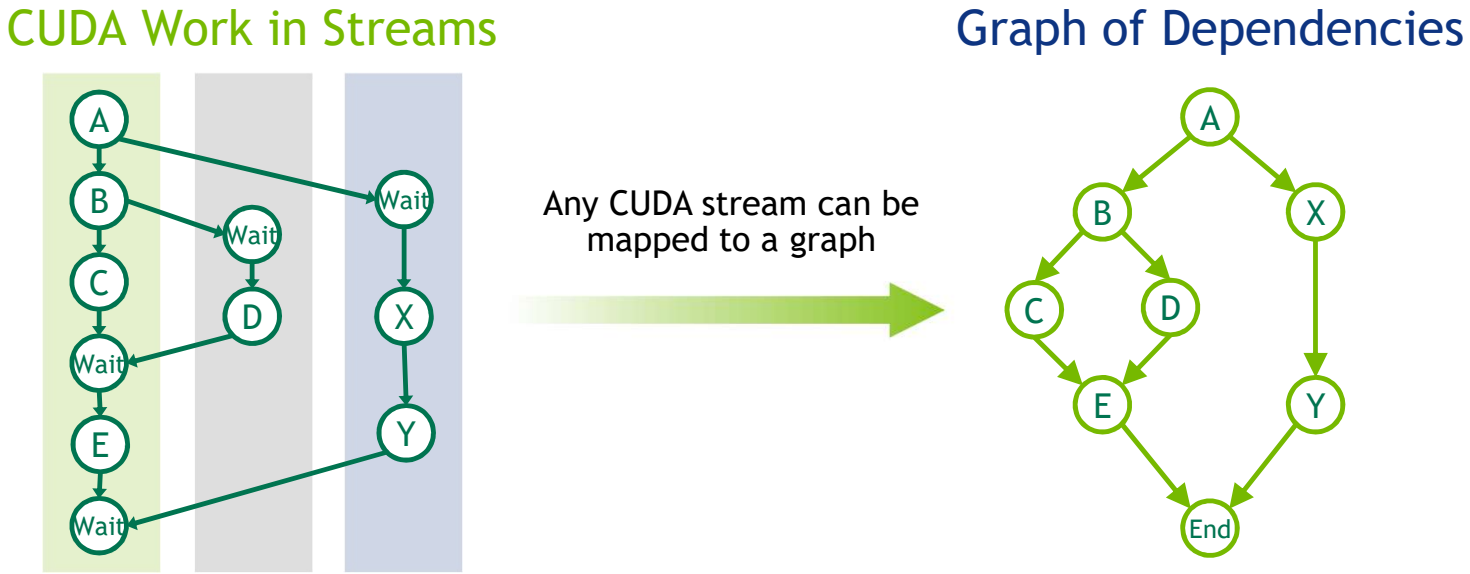
Launch potentially thousands of work items
with a single call

ALL CUDA WORK FORMS A GRAPH

CUDA Work in Streams

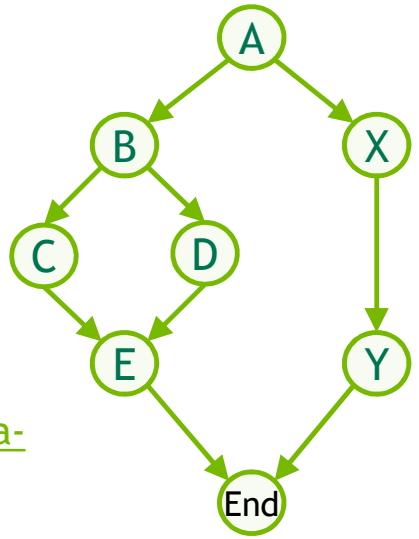


ALL CUDA WORK FORMS A GRAPH



CUDA GRAPHS

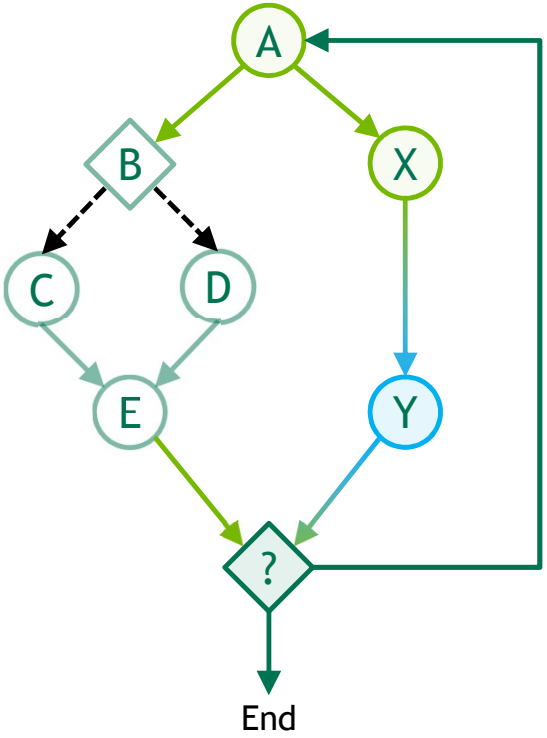
- More complex cases provide more opportunities for savings.
  - Multiple interacting streams with different types of GPU operations.
  - Graphs may span multiple GPUs
- Can define using stream capture or directly using API.
- CUDA – New Features and Beyond, Stephen Jones (NVIDIA)
  - https://on-demand-gtc.gputechconf.com/gtcnew/sessionview.php?sessionName=s9240-cuda%3a+new+features+and+beyond
- Programming Guide:
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs
- CUDA sample: `samples/0_Simple/simpleCudaGraphs`



FUTURE GRAPH DIRECTIONS

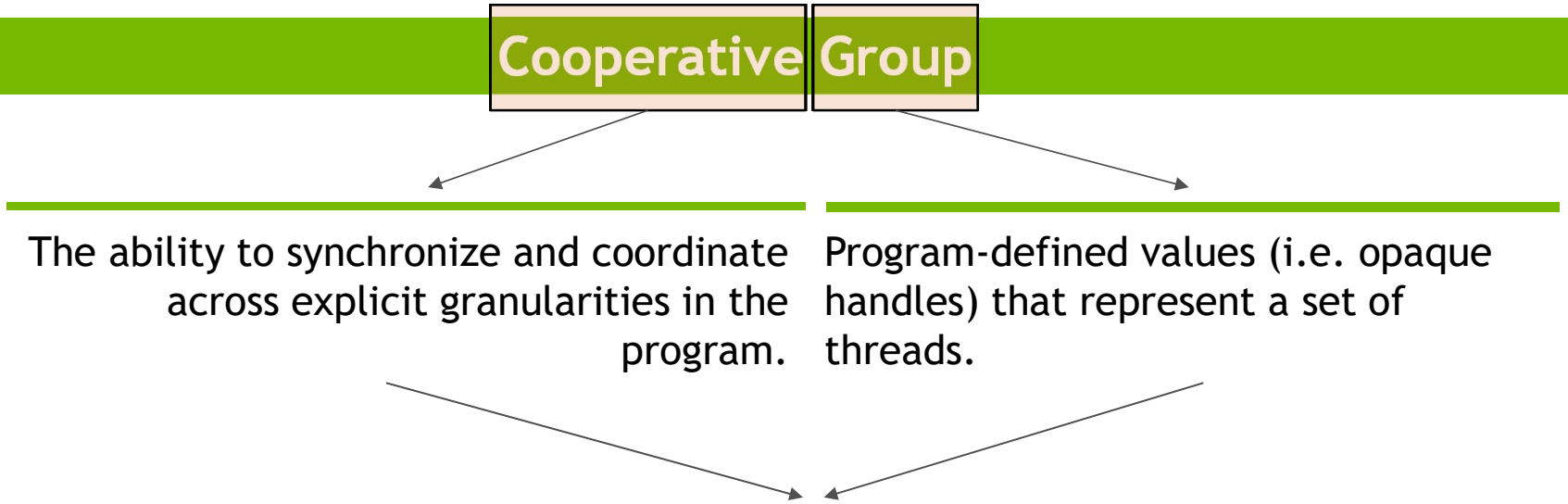
Dynamic Control Flow

- Cyclic Graphs
  - "Iterate until converged" is a common pattern (e.g. Stochastic Gradient Descent)
- Dynamic Branching
  - Run-time, data-dependent execution (e.g. B may be followed by either C or D)



COOPERATIVE GROUPS

CUDA 9+: COOPERATIVE GROUPS



SCOPE OF COOPERATION

Warp

SM

GPU

Node

For currently converged threads:
```
auto group = cg::coalesced_threads();
```

For CUDA thread blocks:
```
auto group = cg::this_thread_block();
```

For cooperative grids:
```
auto group = cg::this_grid();
```

For cooperative grids that span devices:
```
auto group = cg::this_multi_grid();
```

Cooperating on an SM

Cooperating across a GPU

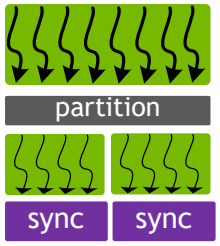
Cooperating across many GPUs with NVLINK

SYNCHRONIZE AT ANY SCALE

Three Key Capabilities

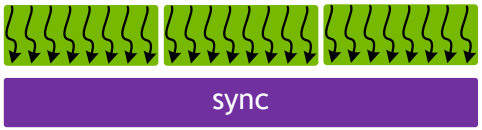
FLEXIBLE GROUPS

Define and synchronize arbitrary groups of threads

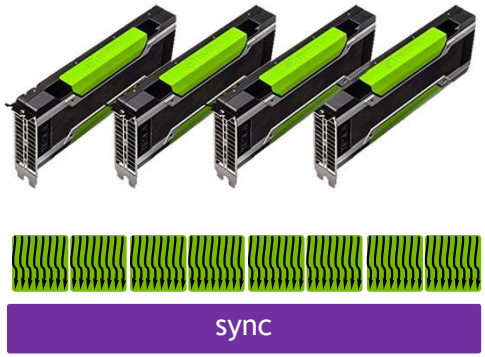


WHOLE-GRID SYNCHRONIZATION

Synchronize multiple thread blocks



MULTI-GPU SYNCHRONIZATION

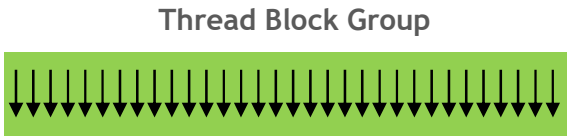


COOPERATIVE GROUPS BASICS

Flexible, Explicit Synchronization

Thread groups are explicit objects in your program

```
thread_group block = this_thread_block();
```

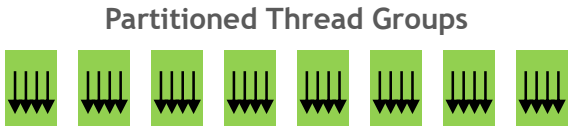


You can synchronize threads in a group

```
block.sync();
```

Create new groups by partitioning existing groups

```
thread_group tile32 = tiled_partition(block, 32);
thread_group tile4 = tiled_partition(tile32, 4);
```

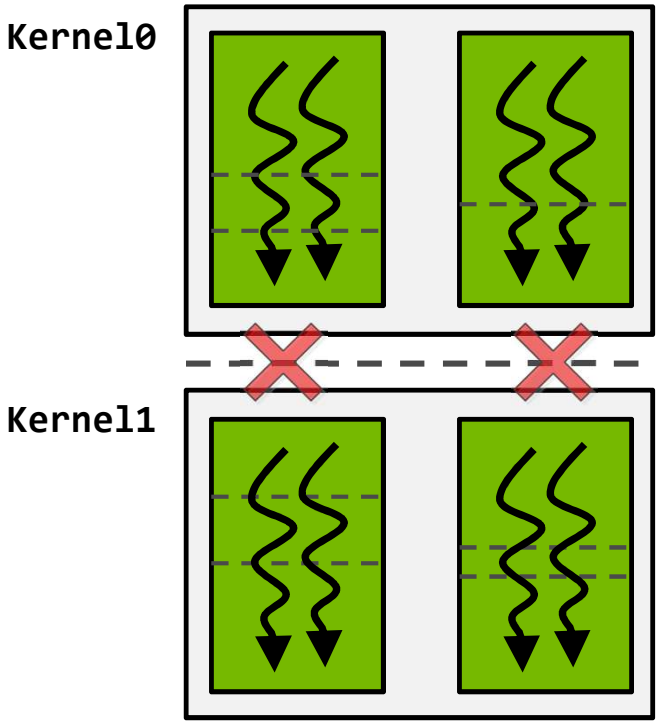


Partitioned groups can also synchronize

```
tile4.sync();
```

Note: calls in green are part of the cooperative_groups:: namespace

CUDA SYNCHRONIZATION MODEL



`__syncthreads()`

"Global" sync between kernels

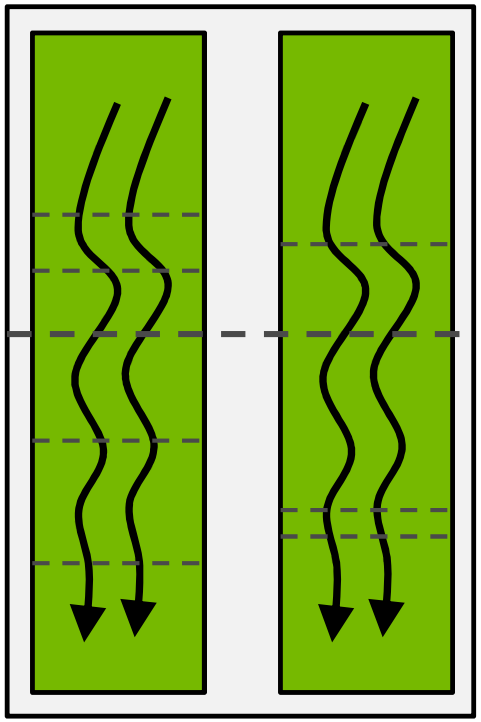
State Invalidation

- CUDA Blocks define the set of threads that communicate and synchronize
- All other synchronization occurs at CUDA Grid boundaries
- Lose register and shared memory state between kernels

  ~20MB on V100

GLOBAL SYNCHRONIZATION

Multi-Block Cooperative Groups



No state invalidation

- Maintain register and shared memory state between phases of execution
- “Persistent Kernel” execution

GRID GROUP

A set of threads within the same grid, guaranteed to be resident on the device

New CUDA Launch API to opt-in:
`cudaLaunchCooperativeKernel(…)`

```
__global__ kernel() {
    grid_group grid = this_grid();
    // load data
    // compute, share data
    grid.sync();
    // grid is now synced
}
```

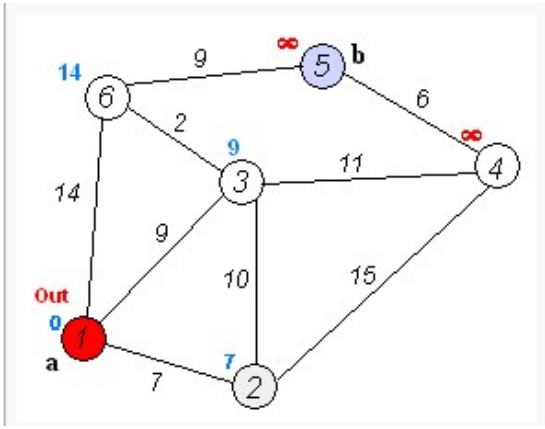
Device needs to support the `cooperativeLaunch` property.

```
cudaOccupancyMaxActiveBlocksPerMultiprocessor(&numBlocksPerSm, kernel, numThreads, 0));
```

GRID GROUP

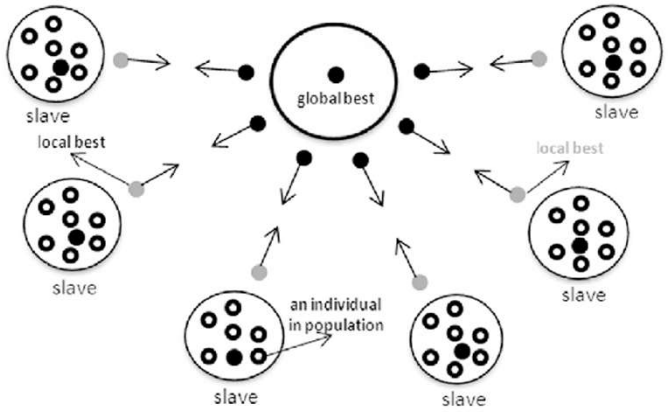
Goal: keep as much state as possible resident

Shortest Path / Search



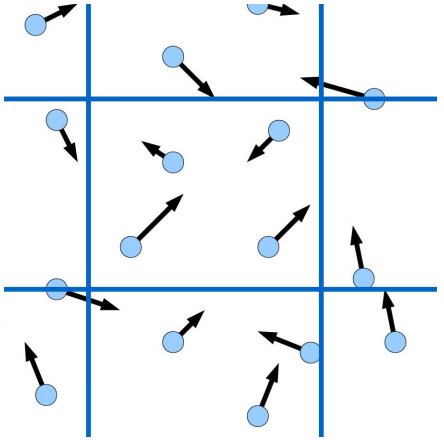
Weight array perfect for persistence
Iteration over vertices?
Fuse!

Genetic Algorithms / Master driven algorithms



Synchronization between a master block and slaves

Particle Simulations

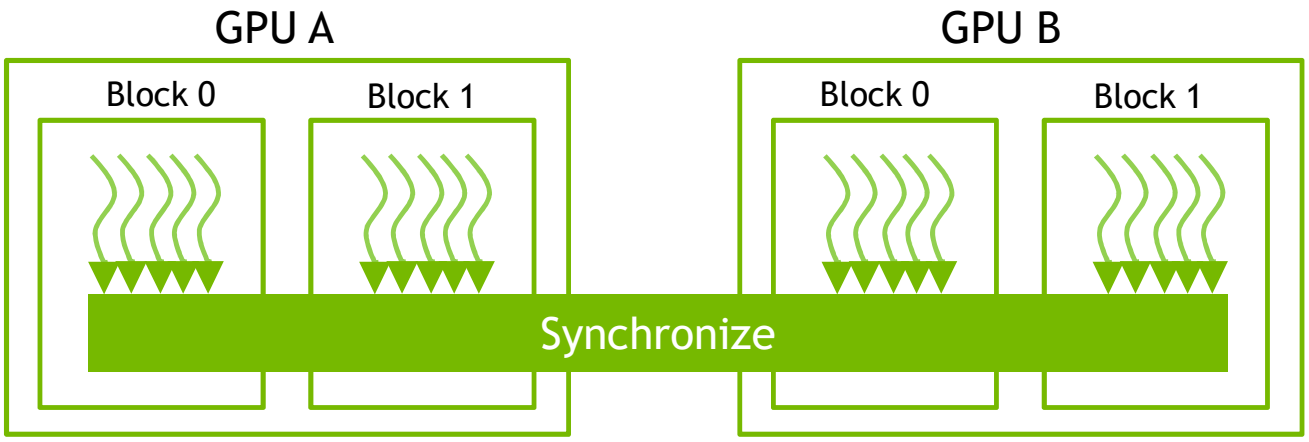


Synchronization between update and collision simulation

MULTI-GRID GROUP

A set of threads guaranteed to be resident on the same system, on multiple devices

```
__global__ void kernel() {
    multi_grid_group multi_grid = this_multi_grid();
    // load data
    // compute, share data
    multi_grid.sync();
    // devices are now synced, keep on computing
}
```



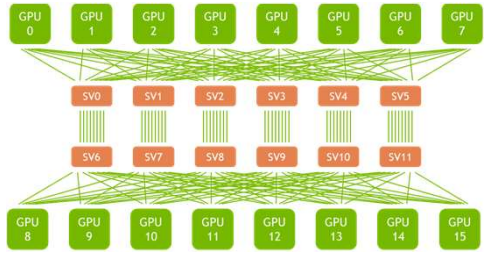
MULTI-GRID GROUP

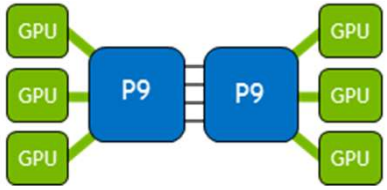
Launch on multiple devices at once

New CUDA Launch API to opt-in:
`cudaLaunchCooperativeKernelMultiDevice(…)`

Devices need to support the `cooperativeMultiDeviceLaunch` property.

```
struct cudaLaunchParams params[numDevices];
for (int i = 0; i < numDevices; i++) {
    params[i].func = (void *)kernel;
    params[i].gridDim = dim3(…); // Use occupancy calculator
    params[i].blockDim = dim3(…);
    params[i].sharedMem = …;
    params[i].stream = …; // Cannot use the NULL stream
    params[i].args = …;
}
cudaLaunchCooperativeKernelMultiDevice(params, numDevices);
```





MULTI-GPU PROGRAMMING

CUDA-AWARE NETWORKING

Getting more GPU centric, easier, more efficient, more standard

- Enabling and extending through standards
  - MPI
    - Stream synchronous send/receive
  - NCCL
    - collectives are similar to MPI, but use a CUDA Stream, are tuned to topology
  - OpenSHMEM / NVSHMEM (EA2) with IB
    - Strength in memory model, relaxed ordering
    - Stream synchronous, Cooperative Groups-based communications with a group of threads
  - UCX enabling for RDMA, IPC underneath OpenMPI, MPICH

NCCL

NVIDIA Collectives Communication Library

- Building block that abstracts highly-optimized communication for each topology
  - Rings within node over NVLink
  - Trees among nodes over network
- Like MPI collectives, but with a CUDA stream
- Essential to deep learning, can be relevant to traditional HPC
- Open sourced as of v2.3

UNIFIED MEMORY

WHAT YOU CAN DO WITH UNIFIED MEMORY

Works everywhere today

```
int *data;
cudaMallocManaged(&data, sizeof(int) * n);
kernel<<< grid, block >>>(data);
```

Works on POWER9 + CUDA 9.2 and forthcoming x86 + HMM

```
int *data = (int*)malloc(sizeof(int) * n);
kernel<<< grid, block >>>(data);
```

```
int data[1024];
kernel<<< grid, block >>>(data);
```

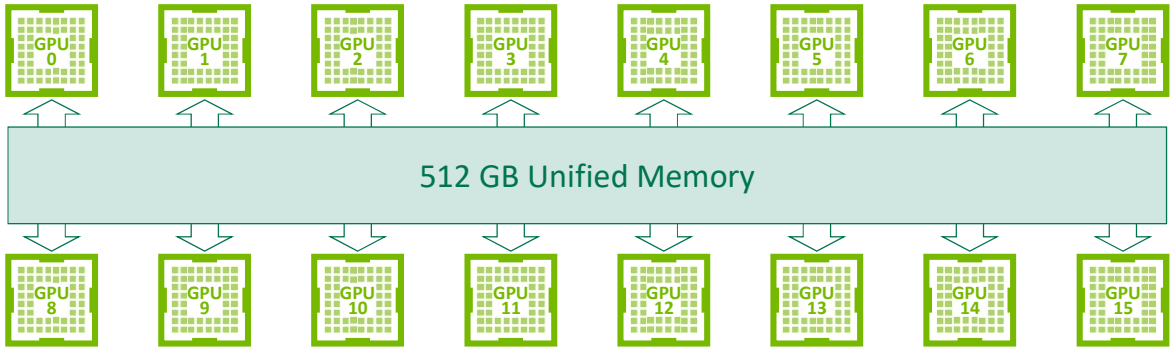
```
int *data = (int*)alloca(sizeof(int) * n);
kernel<<< grid, block >>>(data);
```

```
extern int *data;
kernel<<< grid, block >>>(data);
```

16 GPUs WITH 32GB MEMORY EACH

- **NVSWITCH PROVIDES**
- All-to-all high-bandwidth peer mapping between GPUs
- Full inter-GPU memory interconnect (incl. Atomics)

UNIFIED MEMORY + DGX-2



- **UNIFIED MEMORY PROVIDES**
- Single memory view shared by all GPUs
- Automatic migration of data between GPUs
- User control of data locality

REDUCED PRECISION

REDUCED PRECISION

Why?

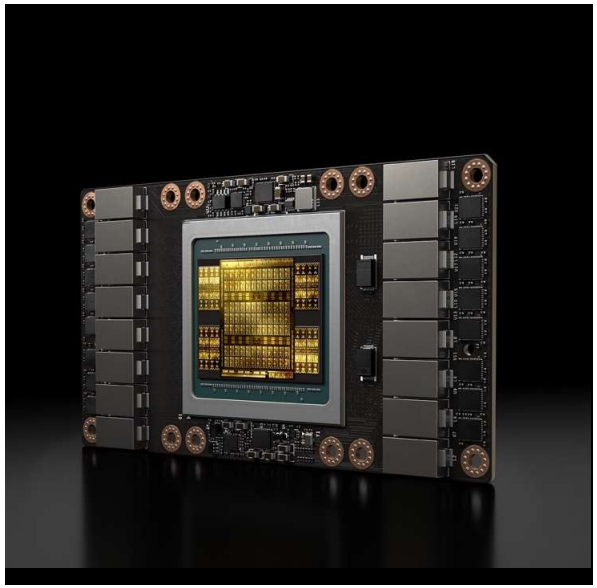
1. Reduce memory traffic

2. Reduce memory footprint

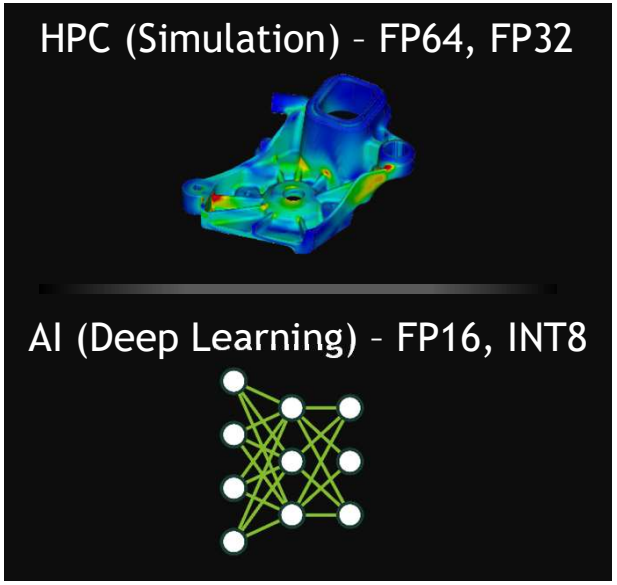
3. Utilize accelerated hardware

….without compromising the end result

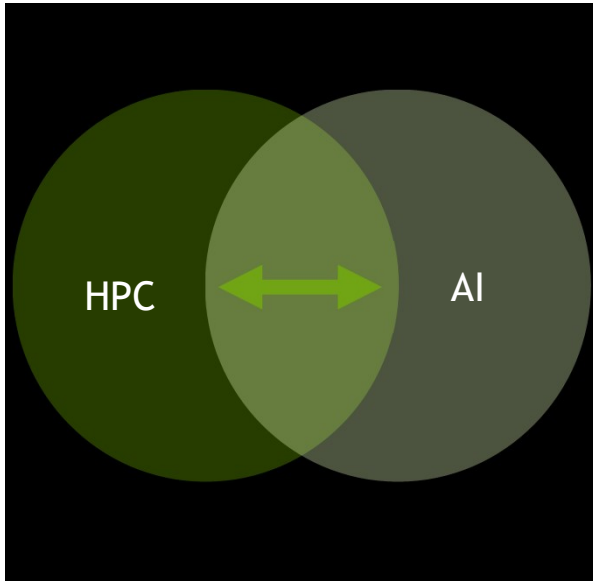
PRECISION MATTERS FOR HPC AND AI



VOLTA TENSOR CORE GPU



MULTI-PRECISION COMPUTING

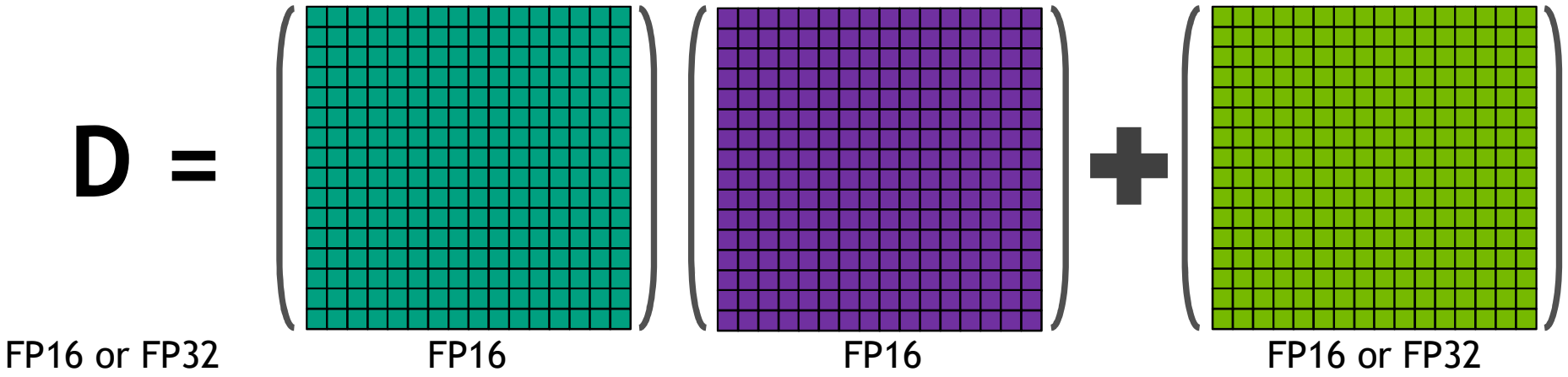


FUSION OF HPC & AI

CUDA TENSOR CORE PROGRAMMING

16x16x16 Warp Matrix Multiply and Accumulate (WMMA)

```
wmma::mma_sync(Dmat, Amat, Bmat, Cmat);
```

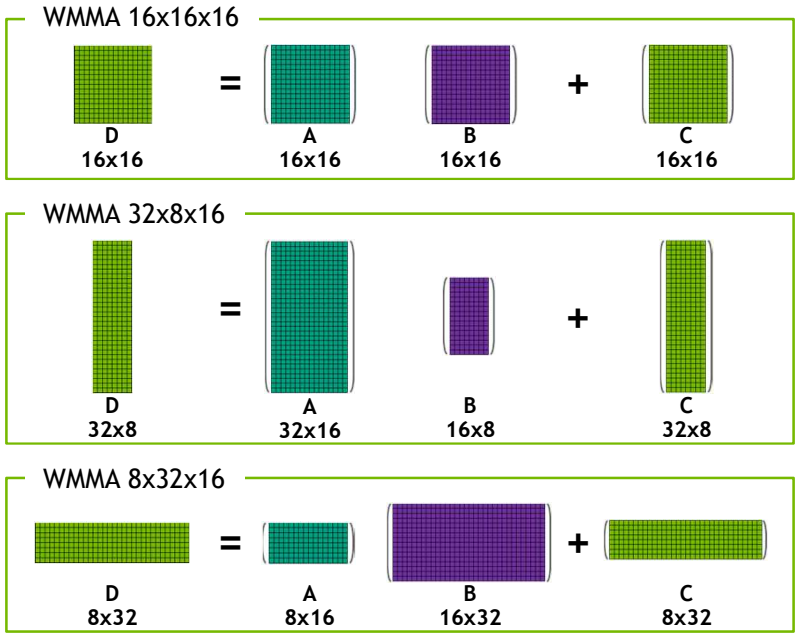


$$D = AB + C$$

TURING TENSOR CORE

New Warp Matrix Functions

- **WMMA operations now include 8-bit integer**
- Turing (sm_75) only
- Signed & unsigned 8-bit input
- 32-bit integer accumulator
- Match input/output dimensions with *half*
- 2048 ops per cycle, per SM



ACCESSING TENSOR CORES

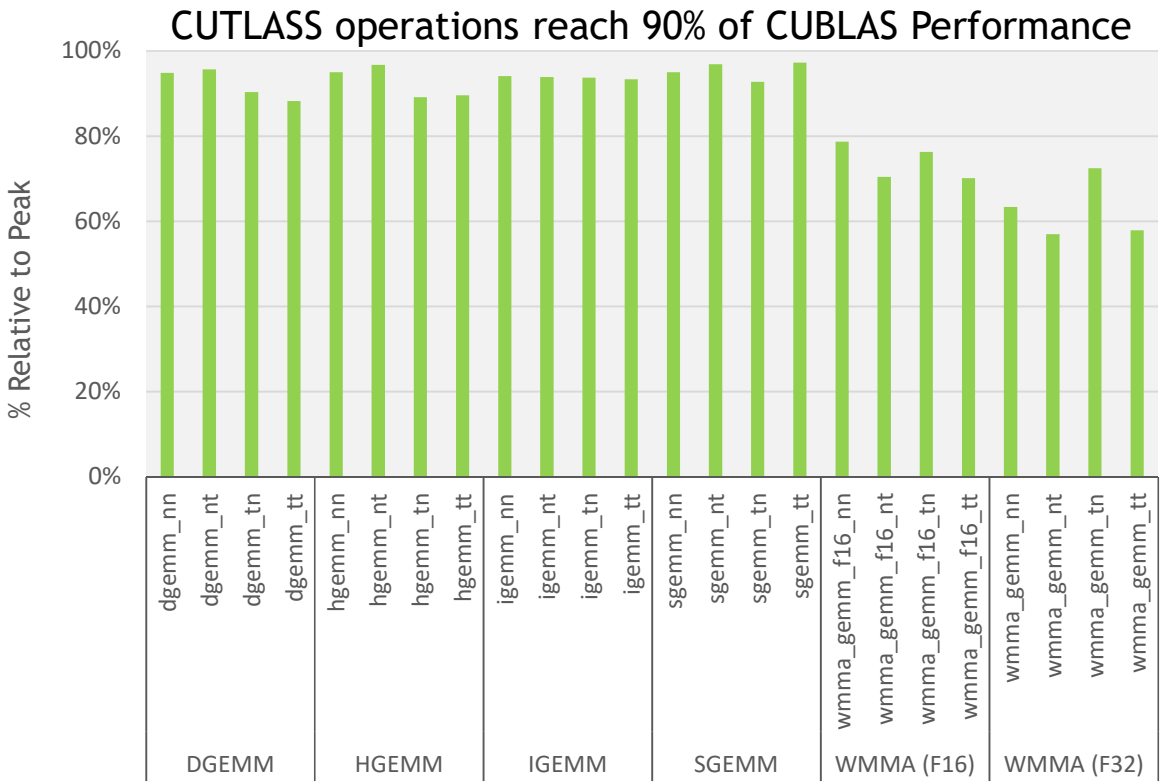
- Directly in CUDA
- Automatically through cuBLAS library calls
- Using CUTLASS C++ template library

CUTLASS 1.1

High-performance Matrix Multiplication in Open Source CUDA C++

- Turing optimized GEMMs
  - Integer (8-bit, 4-bit and 1-bit) using WMMA
- Batched strided GEMM
- Support for CUDA 10.0
- Updates to documentation and more examples

https://github.com/NVIDIA/cutlass

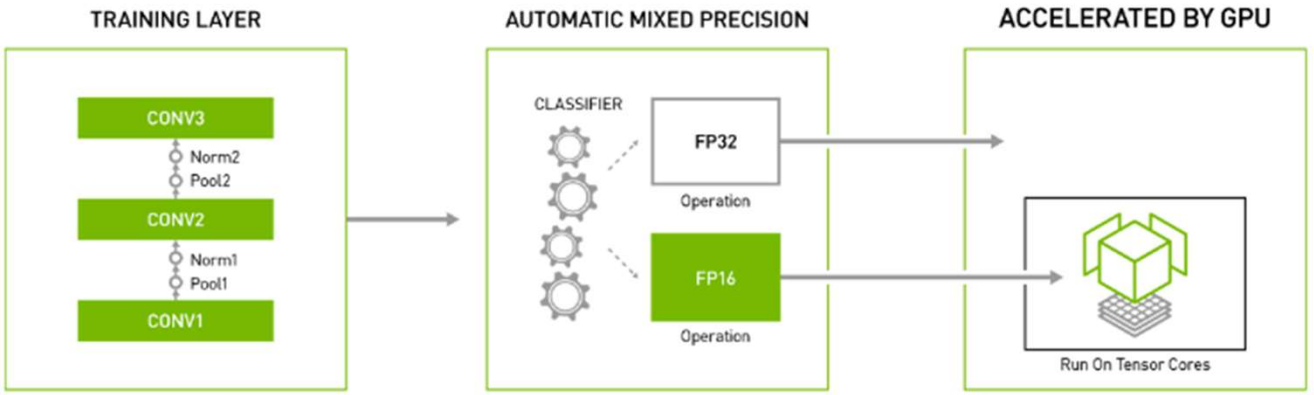


CUTLASS 1.1 on Volta (GV100)

AUTOMATIC MIXED PRECISION

NEW

Easy to Use, Greater Performance and Boost in Productivity



Insert ~ two lines of code to introduce Automatic Mixed-Precision and get upto 3X speedup

AMP uses a graph optimization technique to determine FP16 and FP32 operations

Support for TensorFlow, PyTorch and MXNet

Unleash the next generation AI performance and get faster to the market!

ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code, Get Upto 3X Speedup

TensorFlow	`os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'`
	OR
	`export TF_ENABLE_AUTO_MIXED_PRECISION=1`
	
	Explicit optimizer wrapper available in NVIDIA Container 19.07+, TF 1.14+, TF 2.0:
	
	`opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)`
		GA

PyTorch	`model, optimizer = amp.initialize(model, optimizer, opt_level="O1")`
	`with amp.scale_loss(loss, optimizer) as scaled_loss:`
	`  scaled_loss.backward()`
		GA

MXNet	`amp.init()`
	`amp.init_trainer(trainer)`
	`with amp.scale_loss(loss, trainer) as scaled_loss:`
	`  autograd.backward(scaled_loss)`
		GA
		Coming Soon

More details: https://developer.nvidia.com/automatic-mixed-precision

ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code

- TensorFlow
  - container 19.03+:
    - `export TF_ENABLE_AUTO_MIXED_PRECISION=1` [automatic casting *and* automatic loss scaling]
  - container 19.0 +, TF 1.14+, TF 2.0:
    - We provide an explicit optimizer wrapper to perform loss scaling – which can also enable auto-casting for you:

```
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)
```

ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code

```
from apex import amp

                    amp.initialize(model, optimizer, opt_level="O1")
# …
for train_loop():
  loss = loss_fn(model(x), y)
  with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
  # Can manipulate the .grads if you'd like
  optimizer.step()
```

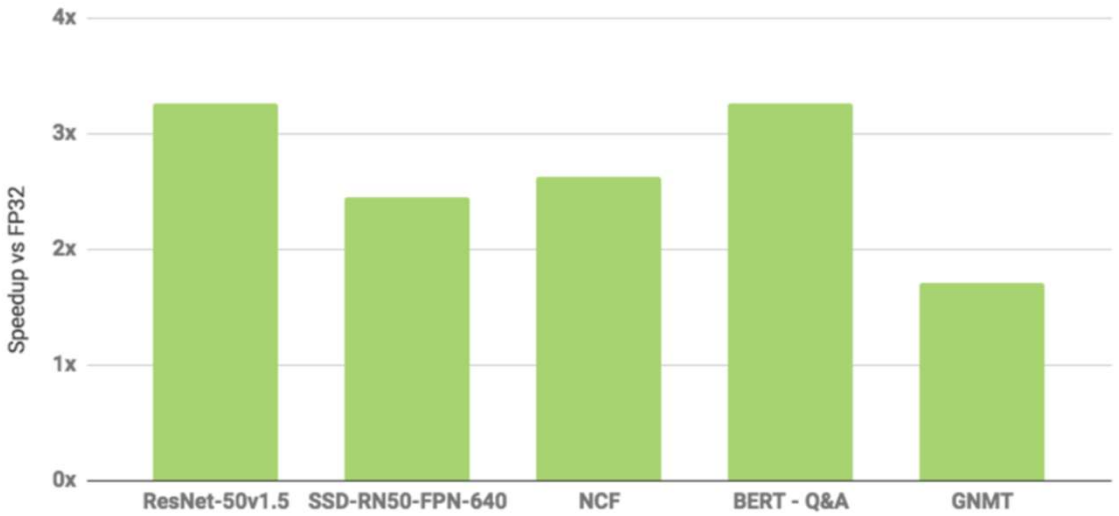
ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code

```
from mxnet.contrib import amp
amp.init()
net = get_network()
trainer = mx.gluon.Trainer(...)
amp.init_trainer(trainer)
for data in dataloader:
  with autograd.record(True):
      out = net(data)
      l = loss(out)
      with amp.scale_loss(l, trainer) as scaled_loss:
          autograd.backward(scaled_loss)
  trainer.step()
```

AUTOMATIC MIXED PRECISION IN TENSORFLOW

Upto 3X Speedup



TensorFlow Medium Post: Automatic Mixed Precision in TensorFlow for Faster AI Training on NVIDIA GPUs

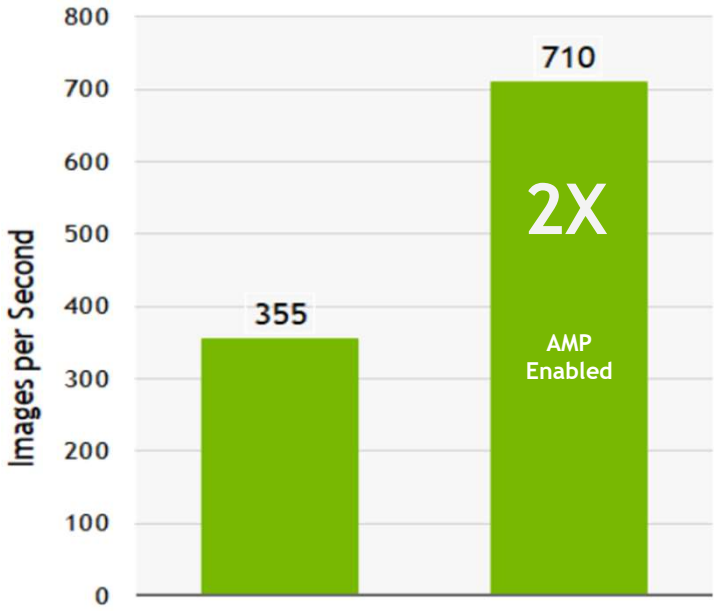
All models can be found at:
https://github.com/NVIDIA/DeepLearningExamples/tree/master/TensorFlow, except for ssd-rn50-fpn-640, which is here: https://github.com/tensorflow/models/tree/master/research/object_detection
All performance collected on 1xV100-16GB, except bert-squadqa on 1xV100-32GB.
Speedup is the ratio of time to train for a fixed number of epochs in single-precision and Automatic Mixed Precision. Number of epochs for each model was matching the literature or common practice (it was also confirmed that both training sessions achieved the same model accuracy).
Batch sizes:. rn50 (v1.5): 128 for FP32, 256 for AMP+XLA; ssd-rn50-fpn-640: 8 for FP32, 16 for AMP+XLA; NCF: 1M for FP32 and AMP+XLA; bert-squadqa: 4 for FP32, 10 for AMP+XLA; GNMT: 128 for FP32, 192 for AMP.

AUTOMATIC MIXED PRECISION IN PYTORCH

https://developer.nvidia.com/automatic-mixed-precision

- Plot shows ResNet-50 result with/without automatic mixed precision(AMP)
- More AMP enabled model scripts coming soon:

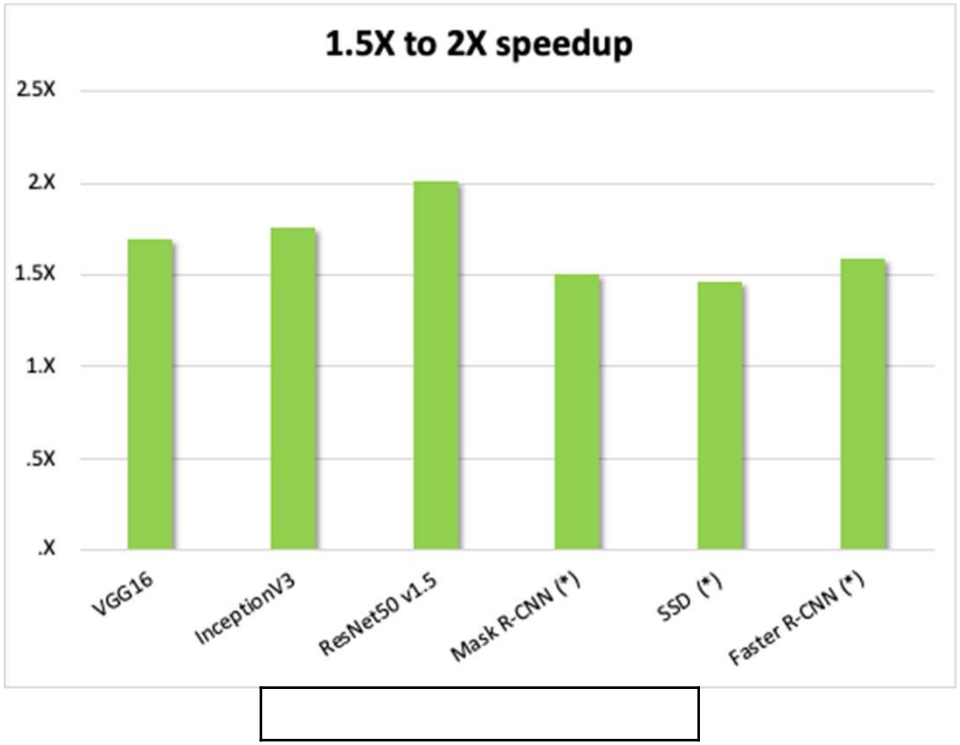
  Mask-R CNN, GNMT, NCF, etc.



https://github.com/NVIDIA/apex/tree/master/examples/imagenet

AUTOMATIC MIXED PRECISION IN MXNET

AMP speedup ~1.5X to 2X in comparison with FP32



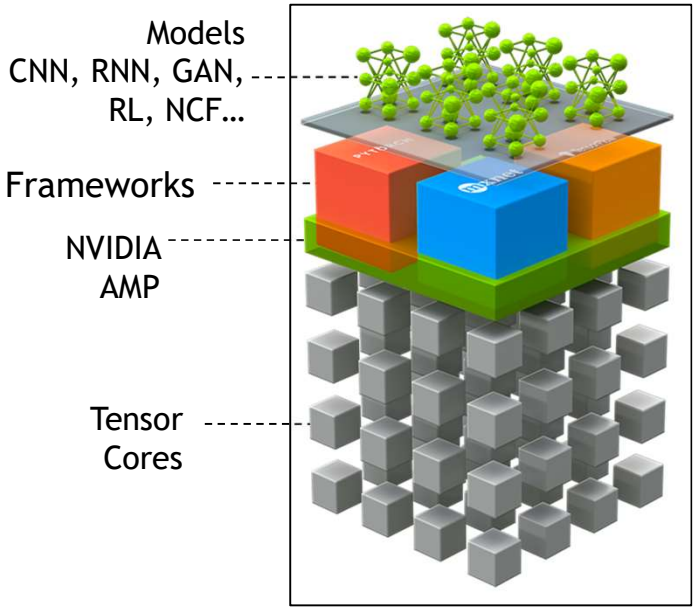
https://github.com/apache/incubator-mxnet/pull/14173

AUTOMATIC MIXED PRECISION

Speedup Your Network Across Frameworks With Just Two Lines of Code

"This easy integration enables TensorFlow developers to literally flip a switch in their AI model and get up to 3X speedup with mixed precision training while maintaining model accuracy."

Rajat Monga, Engineering Director, TensorFlow

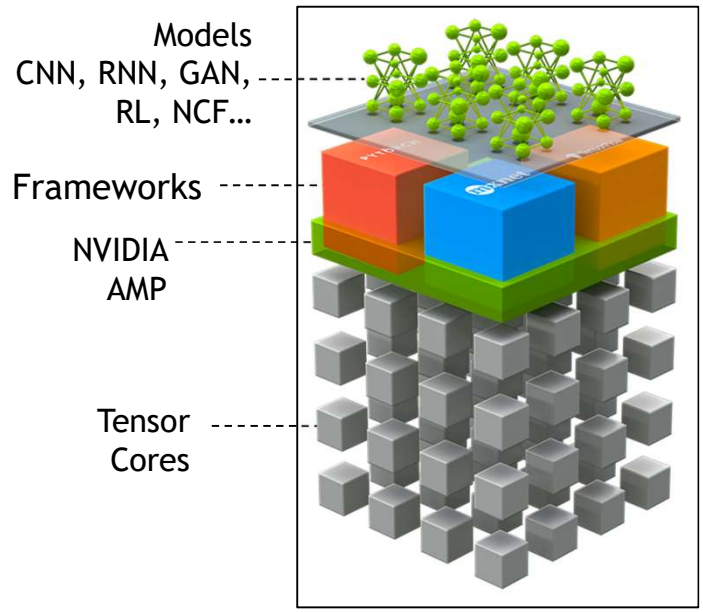


AUTOMATIC MIXED PRECISION

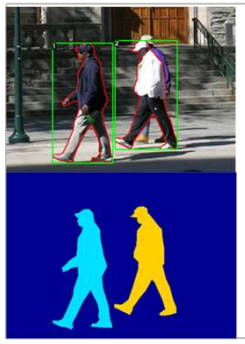
Speedup Your Network Across Frameworks With Just Two Lines of Code

“

Wei Lin, Sr Director, Alibaba Computing Platform



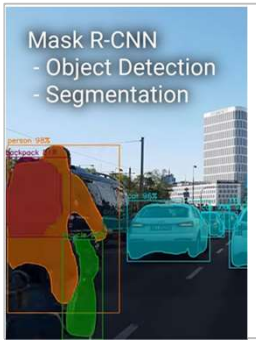
AUTOMATIC MIXED PRECISION IS BEING ADOPTED IN THE COMMUNITY



TORCHVISION

Popular datasets, model architectures, common image transformations for computer vision

Facebook
PyTorch



MASK R-CNN

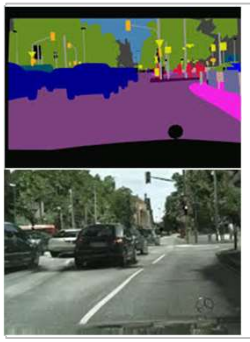
Fast, modular reference implementation of Instance Segmentation and Object Detection

Facebook Research
PyTorch

PYTEXT

A natural language modeling framework

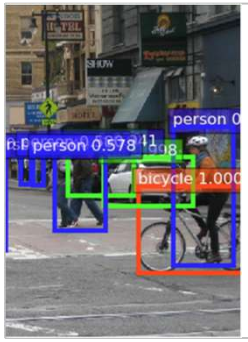
Facebook Research
PyTorch



PIX2PIXHD

Synthesizing and manipulating 2048x1024 images with conditional GANs

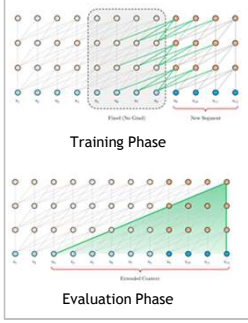
NVIDIA
PyTorch



GLUON-CV

Deep learning toolkit for computer vision

DMLC
MXNet



TRANSFORMER-XL

Attentive language models beyond a fixed-length context

CMU & Google
PyTorch

Link to NVIDIA Developer Page
Link to Github NVIDIA DL Examples


NVIDIA