Inside the NVIDIA Ampere Architecture

Ronny Krashinsky, Olivier Giroux
GPU Architects

GTC 2020
UNPRECEDENTED ACCELERATION AT EVERY SCALE

54 BILLION XTORS
3rd GEN TENSOR CORES
SPARSITY ACCELERATION
MIG
3rd GEN NVLINK & NVSWITCH
All results are measured

BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase 1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512, V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training

BERT Large Inference uses TRT 7.1 for T4/V100, with INT8/FP16 at batch size 256. Pre-production TRT for A100, uses batch size 94 and INT8 with sparsity
All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE Chroma with szsc121_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 in DGX A100
A100 TENSOR-CORE GPU
54 billion transistors in 7nm

- 108 SMs
- 6912 CUDA Cores

- 40MB L2 Cache
- 6.7x capacity

- 1.56 TB/s HBM2
- 1.7x bandwidth

Multi-Instance GPU
Scale UP
Scale OUT
7x
2x BW
3rd gen. NVLINK
Third-generation Tensor Core
Faster and more efficient
Comprehensive data types
Sparsity acceleration

Asynchronous data movement and synchronization

Increased L1/SMEM capacity
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity
<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
</tr>
</tbody>
</table>

V100 TENSOR CORE

V100

125 8x vs.
TOPS FFMA

FF16/FP32
Mixed-precision
## A100 Tensor Core

<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
</tr>
<tr>
<td><strong>A100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
</tr>
</tbody>
</table>

**V100 → A100**

- 2.5x TOPS
- 2x TOPS/SM
- FF16/FP32 Mixed-precision
## A100 TENSOR CORE

<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
</tr>
<tr>
<td><strong>A100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
</tr>
</tbody>
</table>

TF32 accelerates FP32 in/out data $\rightarrow$ 10x vs. V100 FP32
BFloat16 (BF16) at same rate as FP16
# A100 Tensor Core

## Inference data types

<table>
<thead>
<tr>
<th>Input Operands</th>
<th>Accumulator</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
</tr>
<tr>
<td><strong>A100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP16</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>INT8</td>
<td>INT32</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>INT4</td>
<td>INT32</td>
<td>1248</td>
<td>64x</td>
</tr>
<tr>
<td>BINARY</td>
<td>INT32</td>
<td>4992</td>
<td>256x</td>
</tr>
</tbody>
</table>
# A100 TENSOR CORE

<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
<th>SPARSE TOPS</th>
<th>SPARSE X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP16</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>INT8</td>
<td>INT32</td>
<td>624</td>
<td>32x</td>
<td>1248</td>
<td>64x</td>
</tr>
<tr>
<td>INT4</td>
<td>INT32</td>
<td>1248</td>
<td>64x</td>
<td>2496</td>
<td>128x</td>
</tr>
<tr>
<td>BINARY</td>
<td>INT32</td>
<td>4992</td>
<td>256x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

With Sparsity another 2x, INT8/INT4 reach petaops
# A100 Tensor Core

<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
<th>SPARSE TOPS</th>
<th>SPARSE X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP16</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>INT8</td>
<td>INT32</td>
<td>624</td>
<td>32x</td>
<td>1248</td>
<td>64x</td>
</tr>
<tr>
<td>INT4</td>
<td>INT32</td>
<td>1248</td>
<td>64x</td>
<td>2496</td>
<td>128x</td>
</tr>
<tr>
<td>BINARY</td>
<td>INT32</td>
<td>4992</td>
<td>256x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEEE FP64</td>
<td></td>
<td>19.5</td>
<td>1x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V100 → A100 2.5x FLOPS for HPC
## A100 Tensor Core

<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
<th>SPARSE TOPS</th>
<th>SPARSE X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>A100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP16</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>INT8</td>
<td>INT32</td>
<td>624</td>
<td>32x</td>
<td>1248</td>
<td>64x</td>
</tr>
<tr>
<td>INT4</td>
<td>INT32</td>
<td>1248</td>
<td>64x</td>
<td>2496</td>
<td>128x</td>
</tr>
<tr>
<td>BINARY</td>
<td>INT32</td>
<td>4992</td>
<td>256x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IEEE FP64</td>
<td></td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
INSIDE A100 TensorFloat-32 (TF32)

Range of FP32 with precision of FP16

FP32 input/output
FP32 storage and math for all activations, gradients, ... everything outside tensor cores

Out-of-the-box tensor core acceleration for DL
Easy step towards maximizing tensor core performance with mixed-precision (FP16, BF16)

Up to 4x speedup on linear solvers for HPC

→S22082: Mixed-Precision Training of Neural Networks, 5/20 2:45pm PDT
→S21681: How CUDA Math Libraries can help you unleash the power of the new NVIDIA A100 GPU (recording available)
INSIDE A100 SPARSE TENSOR CORE

2x Tensor Core throughput
Structured-sparsity for efficient HW and SW

~2x reduction in weights footprint and bandwidth

~No loss in inferencing accuracy
Evaluated across dozens of networks: vision, object detection, segmentation, natural language modeling, translation

→ S22085: Accelerating Sparsity in the NVIDIA Ampere Architecture, 5/20 1:30pm PDT
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity
DL STRONG SCALING

DL networks:
Long chains of sequentially-dependent compute-intensive layers

1 layer

Each layer is parallelized across GPU

Input Activations

Output Activations

Weights

Tile: work for 1 SM

Weak scaling

Output Activations

Output Activations

Strong scaling

Fixed network runs ~2.5x faster

~2.5x larger network runs in same time
HOW TO KEEP TENSOR CORES FED?

Math bandwidth
(MACs/clock/SM)

Required data bandwidth
(A+B operands, B/clock/SM)

A100 dense

V100

A100 dense

A100 sparse

3x vs. V100

2x vs. V100
A100 STRONG SCALING INNOVATIONS

Improve speeds & feeds and efficiency across all levels of compute and memory hierarchy

Math
RF
SMEM/L1
L2
DRAM
NVLINK

SM

GPU memory system
Multi-GPU systems
A100 TENSOR CORE
2x throughput vs. V100, >2x efficiency

16x16x16 matrix multiply

<table>
<thead>
<tr>
<th></th>
<th>FFMA</th>
<th>V100 TC</th>
<th>A100 TC</th>
<th>A100 vs. V100 (improvement)</th>
<th>A100 vs. FFMA (improvement)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread sharing</td>
<td>1</td>
<td>8</td>
<td>32</td>
<td>4x</td>
<td>32x</td>
</tr>
<tr>
<td>Hardware instructions</td>
<td>128</td>
<td>16</td>
<td>2</td>
<td>8x</td>
<td>64x</td>
</tr>
<tr>
<td>Register reads+writes (warp)</td>
<td>512</td>
<td>80</td>
<td>28</td>
<td>2.9x</td>
<td>18x</td>
</tr>
<tr>
<td>Cycles</td>
<td>256</td>
<td>32</td>
<td>16</td>
<td>2x</td>
<td>16x</td>
</tr>
</tbody>
</table>

Tensor Cores assume FP16 inputs with FP32 accumulator, V100 Tensor Core instruction uses 4 hardware instructions.
A100 SM DATA MOVEMENT EFFICIENCY
3x SMEM/L1 bandwidth, 2x in-flight capacity
A100 L2 BANDWIDTH

Split L2 with hierarchical crossbar - 2.3x increase in bandwidth over V100, lower latency

<table>
<thead>
<tr>
<th></th>
<th>V100</th>
<th>V100++ (hypothetical)</th>
<th>A100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80 SMs</td>
<td>108 SMs</td>
<td>108 SMs</td>
</tr>
<tr>
<td>V100 TC</td>
<td>64 L2 slices</td>
<td>64 L2 slices</td>
<td>80 L2 slices</td>
</tr>
<tr>
<td></td>
<td>32 B/clk/slice</td>
<td>32 B/clk/slice</td>
<td>64 B/clk/slice</td>
</tr>
<tr>
<td></td>
<td>12 B/clk/SM 47%</td>
<td>24 B/clk/SM 127%</td>
<td>24 B/clk/SM 51%</td>
</tr>
<tr>
<td>Output</td>
<td>Tile: work for 1 SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16 B/clk/SM 63%</td>
<td>32 B/clk/SM 169%</td>
<td>32 B/clk/SM 68%</td>
</tr>
<tr>
<td></td>
<td>24 B/clk/SM 94%</td>
<td>48 B/clk/SM 253%</td>
<td>48 B/clk/SM 101%</td>
</tr>
</tbody>
</table>

- NVLINK
- DRAM
- L2
- SMEM/L1
- RF
- Math
**A100 DRAM BANDWIDTH**

**Faster HBM2**
- 25% more pins, 38% faster clocks
  - $\rightarrow$ 1.6 TB/s, 1.7x vs. V100

**Larger and smarter L2**
- 40MB L2, 6.7x vs. V100
  - L2-Residency controls

Keep data resident in L2 to reduce DRAM bandwidth

---

*S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT*
A100 COMPUTE DATA COMPRESSION

Up to 4x DRAM+L2 bandwidth and 2x L2 capacity for fine-grained unstructured sparsity

Activation sparsity due to ReLU

- ResNet-50
- VGG16_BN
- ResNeXt-101

→ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT
A100 NVLINK BANDWIDTH

Third Generation NVLink

- 50 Gbit/sec per signal pair
- 12 links, 25 GB/s in/out, 600 GB/s total
- 2x vs. V100

→S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)
A100 ACCELERATES CUDA GRAPHS

Grid launches:
• CPU-to-GPU
• GPU grid-to-grid

With strong scaling CPU and grid launch overheads become increasingly important (Amdahl’s law)

One-shot CPU-to-GPU graph submission and graph reuse

Microarchitecture improvements for grid-to-grid latencies
A100 STRONG SCALING INNOVATIONS
Delivering unprecedented levels of performance

A100 improvements over V100

- Math: 2.5x Tensor Core math BW (FP16)
- RF: 2.9x Effective RF BW with A100 Tensor Core
- RF: 2.8x Effective RF capacity with Async-Copy bypassing RF
- SMEM/L1: 3.0x Effective SMEM BW with A100 Tensor Core and Async-Copy
- SMEM/L1: 2.3x SMEM capacity
- L2: 2.3x L2 BW
- L2: 6.7x L2 capacity, +Residency Control
- DRAM: 1.7x DRAM BW
- DRAM: 1.3x DRAM capacity
- NVLINK: 2.0x NVLINK BW
### A100 improvements over V100

<table>
<thead>
<tr>
<th>Category</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>2.5x</td>
</tr>
<tr>
<td>Tensor Core math BW (FP16)</td>
<td>5.0x</td>
</tr>
<tr>
<td>RF</td>
<td>2.9x</td>
</tr>
<tr>
<td>Effective RF BW with A100 Tensor Core</td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>2.8x</td>
</tr>
<tr>
<td>Effective RF capacity with Async-Copy bypassing RF</td>
<td></td>
</tr>
<tr>
<td>SMEM/L1</td>
<td>2.3x</td>
</tr>
<tr>
<td>SMEM capacity</td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>2.3x</td>
</tr>
<tr>
<td>L2 BW</td>
<td>9.2x</td>
</tr>
<tr>
<td>L2 capacity, +Residency Control</td>
<td>13.3x</td>
</tr>
<tr>
<td>L2</td>
<td>6.7x</td>
</tr>
<tr>
<td>L2 capacity, +Residency Control</td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>1.7x</td>
</tr>
<tr>
<td>DRAM BW</td>
<td>6.8x</td>
</tr>
<tr>
<td>DRAM</td>
<td>1.3x</td>
</tr>
<tr>
<td>DRAM capacity</td>
<td></td>
</tr>
<tr>
<td>NVLINK</td>
<td>2.0x</td>
</tr>
<tr>
<td>NVLINK BW</td>
<td></td>
</tr>
<tr>
<td>A100 STRONG SCALING INNOVATIONS</td>
<td>Delivering unprecedented levels of performance</td>
</tr>
</tbody>
</table>

**A100 improvements over V100**

- Tensor Core math BW (FP16): 5.0x
- Effective RF capacity with Async-Copy bypassing RF: 2.8x
- Effective SMEM BW with A100 Tensor Core and Async-Copy: 3.0x
- SMEM capacity: 2.3x
- L2 BW: 9.2x
- L2 capacity, +Residency Control: 13.3x
- Compute Data Compression (max): 6.8x
- DRAM BW: 6.8x
- DRAM capacity: 1.3x
- NVLINK BW: 2.0x
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity
NVLINK: ONE BIG GPU

- **InfiniBand/Ethernet**: travels a long distance, consistency is the responsibility of software.

- **PCI Express**: hardware consistency for I/O, not for programming language memory models.

- **NVLINK**: hardware consistency for programming language memory models, like system bus.

**Diagram:**
- **CPU**: API implements consistency to GPU (NvShmem).
- **NIC**: API strengthens consistency to GPU (managed memory, host memory).
- **GPU**: Hardware consistency.
HGX A100: 3\textsuperscript{RD} GEN NVLINK

- **HGX A100 4-GPU**: fully-connected system with 100GB/s all-to-all BW
HGX A100: 3\textsuperscript{RD} GEN NVLINK & SWITCH

- **HGX A100 4-GPU**: fully-connected system with 100GB/s all-to-all BW

- **New NVSwitch**: 6B transistors in TSMC 7FF, 36 ports, 25GB/s each, per direction

- **HGX A100 8-GPU**: 6x NVSwitch in a fat tree topology, 2.4TB/s full-duplex bandwidth
DGX A100: PCIE4 CONTROL & I/O

Hardware consistency
CLOUD SMALL INSTANCE USAGE

- Small workloads can under-utilize GPU cloud instances, provisioned at whole GPU level
- CSPs can’t use MPS for GPU space-sharing, because it doesn’t provide enough isolation

Volta security boundary.
NEW: MULTI-INSTANCE GPU (MIG)

- Up to 7 instances total, dynamically reconfigurable
- Compute instances: compute/fault isolation, but share/compete for memory
- GPU instances: separate and isolated paths through the entire memory system

MIG security & containment
ELASTIC GPU COMPUTING

- Each A100 is 1 to 7 GPUs
- Each DGX A100 is 1 to 56 GPUs
- Each GPU can serve a different user, with full memory isolation and QoS

→ S21975: Inside NVIDIA’s Multi-Instance GPU Feature (recording available)
→ S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)
→ S21702: Introducing NVIDIA DGX A100: The Universal AI System for Enterprise, 5/20 9:00am PDT
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity
COMPUTE CAPABILITY

Programming Model Development at NVIDIA

Compute 7.0 (V100)

- Concurrent algorithms
- Managed memory
- Bulk parallelism + atomics

Compute 6.0 (P100)
GPU PROGRAMMING IN 2020 AND BEYOND
Math Libraries | Standard Languages | Directives | CUDA

GPU Accelerated C++ and Fortran
Incremental Performance Optimization with Directives
Maximize GPU Performance with CUDA C++/Fortran

GPU Accelerated Math Libraries

---

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
    });
```

```
do concurrent (i = 1:n)
    y(i) = y(i) + a*x(i)
enddo
```

```
#pragma acc data copy(x,y)
{
...
    std::transform(par, x, x+n, y, y,
        [=](float x, float y){
            return y + a*x;
        });
...
}
```

```
__global__
void saxpy(int n, float a, 
    float *x, float *y) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) y[i] += a*x[i];
}
```

```
int main(void) {
    ...
    cudaMemcpy(d_x, x, ...);
    cudaMemcpy(d_y, y, ...);
    saxpy<<<(N+255)/256,256>>>(...);
    cudaMemcpy(y, d_y, ...);
    std::transform(par, x, x+n, y, y,
        [=](float x, float y){
            return y + a*x;
        });
    ...
}
```
PROGRAMMING MODEL WANTED
Software pipelining to hide latency is hard.

```c
__device__ void exhibit_A1()
{
    memcpy(/* ... */); //< blocks here
    /* more work */
    compute(); //< needed here
    /* more work */
}
```

```c
__device__ void exhibit_B1()
{
    compute_head();
    __syncthreads(); //< blocks here
    /* more work */
    compute_tail(); //< needed here
    /* more work */
}
```

Data

Compute
Software pipelining to hide latency is hard.

```c
__device__ void exhibit_A2()
{
    memcpy(/* ... */);  //< blocks here
    /* memcpy( ... ); */
    compute();           //< needed here
    /* compute(); */
}

__device__ void exhibit_B2()
{
    compute_head();
    __syncthreads();   //< blocks here
    /* compute_head();
     __syncthreads(); */
    compute_tail();     //< needed here
    /* compute_tail(); */
}
```
NEW:

- Asynchronous algorithms
- Concurrent algorithms
- Managed memory
- Bulk parallelism

Compute 8.0 (A100)
Compute 7.0 (V100)
Compute 6.0 (P100)
CO-DESIGNED: A100 & C++20 barrier
Key to asynchronous programming in compute_80

#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

class barrier {  // synopsis
    //...
    void arrive_and_wait();
    arrival_token arrive(ptrdiff_t = 1);
    void wait(arrival_token &&) const;
    //...
};
# ASYNCHRONOUS COPY + BARRIER

<table>
<thead>
<tr>
<th>Capability</th>
<th>PTX ISA</th>
<th>CUDA C++ API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous barrier</td>
<td>mbarrier.{&lt;basis functions&gt;}</td>
<td>cuda::barrier&lt;...&gt;</td>
</tr>
<tr>
<td>Asynchronous copy</td>
<td>cp.async.ca + cp.async.mbarrier.arrive</td>
<td>cuda::memcpy_async(...)</td>
</tr>
<tr>
<td>+Cache-bypass</td>
<td>cp.async.cg</td>
<td></td>
</tr>
<tr>
<td>+Zero-fill ragged edge</td>
<td>cp.async.* ... wr-size, rd-size;</td>
<td></td>
</tr>
<tr>
<td>+User-level tracking</td>
<td>cp.async.mbarrier.arrive.noinc</td>
<td>CUDA 11 preview library in experimental:: namespace</td>
</tr>
<tr>
<td>+Single-threaded mode</td>
<td>cp.async.{commit_group, wait_group}</td>
<td></td>
</tr>
</tbody>
</table>
__device__ void exhibit_A3()
{
    __shared__ barrier b1, b2;
    // ^^initialization omitted
    cudaMemcpyAsync(/* ... */, b1);
    cudaMemcpyAsync(/* ... */, b2);
    b1.arrive_and_wait();
    compute();
    b2.arrive_and_wait();
    compute();
}

#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

__device__ void exhibit_B3()
{
    __shared__ barrier b1, b2;
    // ^^initialization omitted
    compute_head();
    auto t1 = b1.arrive();
    compute_head();
    auto t2 = b2.arrive();
    b1.wait(t1);
    compute_tail();
    b2.wait(t2);
    compute_tail();
}
MULTI-BUFFERING PIPELINES IN C++

```cpp
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

__global__ void exhibit_C(/* ... */) {
  __shared__ barrier b[2];
  // ^^initialization omitted
  barrier::arrival_token t[2];
  cuda::memcpy_async(/* ... */ , b[0]);
  t[0] = b[0].arrive();
  for (int step = 0, next = 1; step < steps; ++step, ++next) {
    if (next < steps) {
      b[next & 1].wait(t[next & 1]);
      cuda::memcpy_async(/* ... */ , b[next & 1]);
      t[next & 1] = b[next & 1].arrive();
    }
    b[step & 1].wait(t[step & 1]);
    compute();
    t[step & 1] = b[step & 1].arrive();
  }
}
```
#include <cuda/barrier> // ISO C++20 conforming extension using barrier = cuda::barrier<cuda::thread_scope_block>;

__global__ void exhibit_C(/* ... */) {
    __shared__ barrier b[2];
    // ^^initialization omitted
    barrier::arrival_token t[2];
    cudaMemcpyAsync(/* ... */ , b[0]);
    t[0] = b[0].arrive();
    for (int step = 0, next = 1; step < steps; ++step, ++next) {
        if (next < steps) {
            b[next & 1].wait(t[next & 1]);
            cudaMemcpyAsync(/* ... */ , b[next & 1]);
            t[next & 1] = b[next & 1].arrive();
        }
        b[step & 1].wait(t[step & 1]);
        compute();
        t[step & 1] = b[step & 1].arrive();
    }
}

MULTI-BUFFERING PIPELINES IN C++
OUR PRODUCTIVITY GAINS FROM A100

CUTLASS Relative Performance to cuBLAS (tensor fp16)
- V100
- A100

88% 95%

Optimized tensor kernels
- V100 (launch)
- V100 (6 months)
- A100 (launch)

Thousands

Tens Hundreds
CLOSING
UNPRECEDENTED ACCELERATION AT EVERY SCALE