10/22/21
Papers:

Sparse GPU Kernels for Deep Learning
https://arxiv.org/abs/2006.10901

Adaptive Tiling
https://www.researchgate.net/publication/330891126_Adaptive_sparse_tiling_for_sparse_matrix_multiplication

Software:

https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-generic-function-sddmm

https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-generic-function-spmm

https://github.com/google-research/sputnik

Fused MM Equivalent?

https://github.com/NVIDIA/cutlass/blob/master/examples/13_two_tensor_op_fusion/README.md

10/27/21

Paper:
Sparse GPU Kernels for Deep Learning
https://arxiv.org/abs/2006.10901

Sparse GPU Kernels Notes:

- Method is designed with no constraint on sparsity structure (good for high accuracy in dnn's)

Primary Methods Used:

- **row swizzle load balancing**: an approach for load balancing computation between processing elements that is decoupled from the parallelization scheme
- **subwarp tiling and reverse-offset memory alignment**: use of vector memory instructions on misaligned memory addresses in sparse data structures
- **1-dimensional tiling:** decomposing the computation across processing elements that facilitates reuse of operands and lends itself to an extensible implementation

Analytical Motivations:

row length = number of nonzeros

Measures:
coefficient of variation (CoV): standard deviation of the row lengths divided by their mean (↑CoV = increased load imbalance across matrix rows)
average row length: average amount of work required on per row basis (↑rowlen = better, as startup overhead and one-time costs can be amortized over more useful work)
sparsity: fraction of values that are zero valued in a matrix

Scientific vs DL sparse computations:

DL dataset: ResNet-50 and Transformer models trained on ImageNet and WMT14 English-to-German
Scientific dataset: SuiteSparse Matrix Collection

On average, deep learning matrices are…

- 13.4× less sparse
- have 2.3× longer rows,
- have 25× CoV

Paper:
Efficient Tensor Core-Based GPU Kernels for Structured Sparsity
under Reduced Precision
https://seal.ece.ucsb.edu/sites/default/files/publications/vector_sparse_transformer_camera_ready_.pdf

Primary Methods Used

- **Column Vector Sparse Encoding**: each index corresponds to a nonzero column vector
- **TCU-based 1-D Octet Tiling**: a novel mapping between the warp tile and the TCU (Tensor Core Unit)

Results:

SpMM:

- 1.71-7.19x geometric mean speedup over the Blocked-ELL SpMM kernel from cuSPARSE and
- 1.34-4.51x geometric mean speedup over a FPU-based kernel that we directly extended from Sputnik [6]

SDDMM:

- 1.27-3.03x speedup over the FPU-based kernel extended from Sputnik
- 0.93-1.44x speedup over the TCU-based kernel that uses the classic mapping between GEMM-like warp tile and TCU.

11/3/21

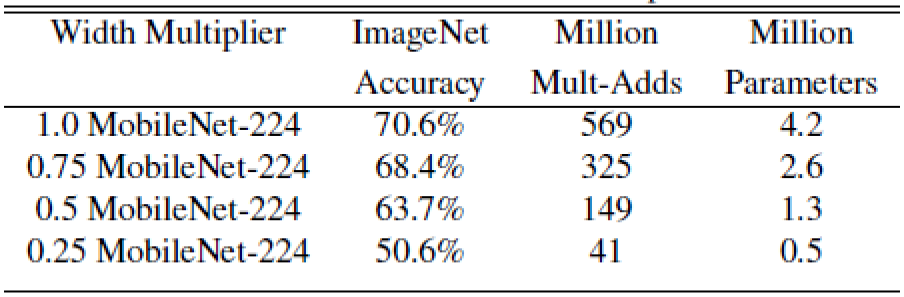
- tensorflow fused operations (https://www.tensorflow.org/lite/convert/operation_fusion)

Sparse GPU kernel Testing (Models)

(Had to modify Dockerfile to get working)

Testing 1: MobileNetV1 -> Dataset intended for training on embedded systems like phone soc's

Table 6. MobileNet Width Multiplier



Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Different Values of Width Multiplier α

```
bash benchmark.sh ../../sgk_models/mbv1/fused-sparse-18-90 1.8 0.9
```

```
2021-11-03 04:05:49.520469: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN
version 8204
E1103 04:05:53.152168 140072011511616 main.py:218] Iterations 0 processed 217.717460 FPS.
E1103 04:05:54.650494 140072011511616 main.py:218] Iterations 1 processed 667.517581 FPS.
E1103 04:05:56.150667 140072011511616 main.py:218] Iterations 2 processed 666.711704 FPS.
E1103 04:05:57.650732 140072011511616 main.py:218] Iterations 3 processed 666.757702 FPS.
E1103 04:05:59.153332 140072011511616 main.py:218] Iterations 4 processed 665.632372 FPS.
E1103 04:06:00.657781 140072011511616 main.py:218] Iterations 5 processed 664.783303 FPS.
E1103 04:06:02.163858 140072011511616 main.py:218] Iterations 6 processed 664.074736 FPS.
E1103 04:06:03.675386 140072011511616 main.py:218] Iterations 7 processed 661.698138 FPS.
E1103 04:06:05.183012 140072011511616 main.py:218] Iterations 8 processed 663.416677 FPS.
E1103 04:06:06.692718 140072011511616 main.py:218] Iterations 9 processed 662.495072 FPS.
E1103 04:06:08.203012 140072011511616 main.py:218] Iterations 10 processed 662.242145 FPS.
E1103 04:06:09.716861 140072011511616 main.py:218] Iterations 11 processed 660.685538 FPS.
E1103 04:06:11.230345 140072011511616 main.py:218] Iterations 12 processed 660.838350 FPS.
E1103 04:06:12.744751 140072011511616 main.py:218] Iterations 13 processed 660.437006 FPS.
E1103 04:06:14.260128 140072011511616 main.py:218] Iterations 14 processed 660.013819 FPS.
E1103 04:06:15.776419 140072011511616 main.py:218] Iterations 15 processed 659.617937 FPS.
E1103 04:06:17.295921 140072011511616 main.py:218] Iterations 16 processed 658.222755 FPS.
E1103 04:06:18.816529 140072011511616 main.py:218] Iterations 17 processed 657.743601 FPS.
E1103 04:06:20.338800 140072011511616 main.py:218] Iterations 18 processed 657.030603 FPS.
E1103 04:06:21.863760 140072011511616 main.py:218] Iterations 19 processed 655.880708 FPS.
```

```
E1103 04:06:23.385728 140072011511616 main.py:218] Iterations 20 processed 657.129115 FPS.
E1103 04:06:23.386128 140072011511616 main.py:222] Mean, Std, Max, Min throughput = 661.646443,
3.419187, 667.517581, 655.880708
```