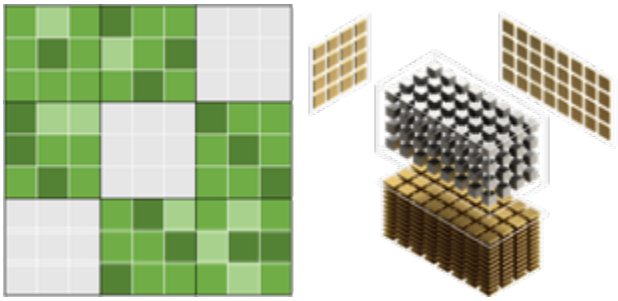
Accelerating Matrix Multiplication with Block Sparse Format and NVIDIA Tensor Cores | NVIDIA Developer Blog

https://developer.nvidia.com/blog/accelerating-matrix-multiplication-with-block-sparse-format-and-nvidia-tensor-cores/

Takuma Yamaguchi Wed Apr, 14 13:02

Sparse-matrix dense-matrix multiplication (SpMM) is a fundamental linear algebra operation and a building block for more complex algorithms such as finding the solutions of linear systems, computing eigenvalues through the preconditioned conjugate gradient, and multiple right-hand sides Krylov subspace iterative solvers. SpMM is also an important kernel used in many domains such as fluid dynamics, deep learning, graph analytics, and economic modeling. In the specific context of deep learning, sparsity has emerged as one of the leading approaches for increasing training and inference performance as well as reducing the model sizes while keeping the accuracy.



Even though sparse linear algebra allows representing huge matrices very efficiently, it typically does not provide competitive performance compared to dense counterparts in cases when sparsity is below 95%. This is due to irregular computation and scattered memory accesses. In fact, many of the linear algebra applications that benefit from sparsity have over 99% sparsity in their matrices.

To overcome this limitation, the NVIDIA Ampere architecture introduces the concept of *fine-grained structured sparsity*, which doubles throughput of dense-matrix multiplies by skipping the computation of zero values in a 2:4 pattern. Recently, NVIDIA introduced the cuSPARSELt library to fully exploit third-generation Sparse Tensor Core capabilities.

The primary alternative to *fine-grained sparsity* is through the organization of matrix entries/ network weights in groups, such as vectors or blocks. This *coarse-grained sparsity* allows regular access pattern and locality, making the computation amenable for GPUs. In deep learning, block sparse matrix multiplication is successfully adopted to reduce the complexity of the standard self-attention mechanism, such as in Sparse Transformer models or in its extensions like Longformer.

Starting with cuSPARSE 11.4.0, the CUDA Toolkit provides a new high-performance *block sparse matrix multiplication* routine that allows exploiting NVIDIA GPU dense Tensor Cores for nonzero sub-matrices and significantly outperforms dense computations on Volta and newer architecture GPUs.

cuSPARSE Block-SpMM: Efficient, block-wise SpMM

Figure 1 shows the general matrix multiplication (GEMM) operation by using the block sparse format. On the left are the full matrix organized in blocks and its internal memory representation: compressed values and block indices. As the usual dense GEMM, the computation partitions the output matrix into tiles. The kernel computes the output tile by stepping through the active tiles (left to right) and accumulates the results into the C matrix. Differently from classical GEMM, not all values of the dense-matrix B are accessed for computing the output. This approach allows skipping unnecessary computation represented by nonzero values and dramatically improves the performance.

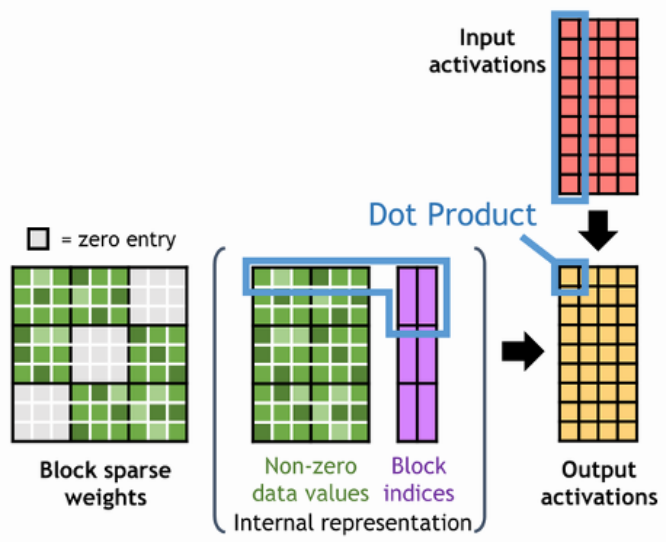


Figure 1. GEMM using block sparse weights and dense activations.

Blocked-Ellpack format

Figure 2 shows that the *Blocked-Ellpack* (Blocked-ELL) storage format contains two 2-D arrays. The right array stores nonzero values in consecutive blocks, while the second array contains the column indices of the corresponding nonzero blocks. All rows in the arrays must have the same number of blocks. Non-structural zero blocks are also accepted as padding. These arrays store components in row-major order, like the compressed sparse row (CSR) format.

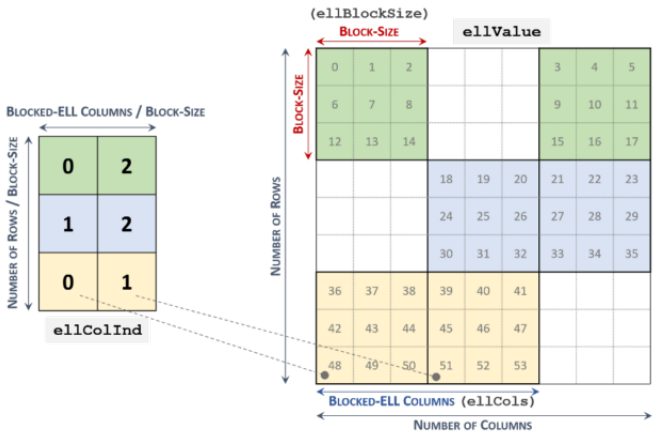


Figure 2. Definition of Blocked-ELL format.

cuSPARSE SpMM

The cuSPARSE library provides `cusparseSpMM` routine for SpMM operations. Compute the following multiplication:

$$\mathbf{C} = \alpha \cdot op(\mathbf{A}) \cdot op(\mathbf{B}) + \beta \cdot \mathbf{C}$$

In this operation, `A` is a sparse matrix of size *MxK*, while `B` and `C` are dense matrices of size *KxN* *MxN*, respectively. Denote the layouts of the matrix `B` with *N* for row-major order, where `op` is non-transposed, and *T* for column-major order, where `op` is transposed.

`cusparseSpMM` selects suitable kernels depending on the storage format, the number of nonzero components, and matrix layouts. This routine supports CSR, Coordinate (COO), as well as the new Blocked-ELL storage formats. Table 1 shows the supported data types, layouts, and compute types.

A/B type	C type	Compute type	op(A)	op(B)	Compute capability	Architecture
half	half	half	N	N, T	$\geqslant$ 7.0	$\geqslant$ Volta
half	half	float	N	N, T	$\geqslant$ 7.0	$\geqslant$ Volta
half	float	float	N	N, T	$\geqslant$ 7.0	$\geqslant$ Volta
int8	int8	int	N	N	$\geqslant$ 7.5	$\geqslant$ Turing
bfloat16	bfloat16	float	N	N, T	$\geqslant$ 8.0	$\geqslant$ NVIDIA AMPERE
bfloat16	float	float	N	N, T	$\geqslant$ 8.0	$\geqslant$ NVIDIA AMPERE
float	float	float	N	N, T	$\geqslant$ 8.0	$\geqslant$ NVIDIA AMPERE
double	double	double	N	N, T	$\geqslant$ 8.0	$\geqslant$ NVIDIA AMPERE

Table 1. Supported data types, layouts, and architectures in cusparseSpMM with Blocked-ELL storage format.

Block-SpMM performance

Here's a snapshot of the relative performance of dense and sparse-matrix multiplications exploiting NVIDIA GPU Tensor Cores. Figures 3 and 4 show the performance of Block-SpMM on NVIDIA V100 and A100 GPUs with the following settings:

- Matrix sizes: M=N=K=4096.
- Block sizes: 32 and 16.
- Input/output data type: half (fp16).
- Computation data type: float (fp32).

The speedup ratio compared to cuBLAS is nearly linear to the sparsity on both NVIDIA V100 and A100 GPUs. When the block size is 32, the kernel is faster than cuBLAS if the density is less than 40% on NVIDIA Volta and 50% on NVIDIA Ampere architecture.

For better performance, it is important to satisfy the following conditions:

- Use large block sizes, preferably a power-of-2.
- Use 128-byte aligned pointers for matrices for vectorized memory access.

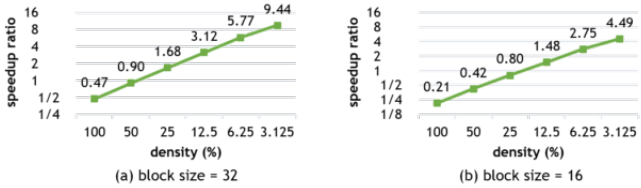


Figure 3. Speedup of cuSPARSE Block-SpMM over Dense GEMM in cuBLAS on NVIDIA V100, fp16 in/out, fp32 compute, NN layout, CUDA Toolkit 11.2.1.

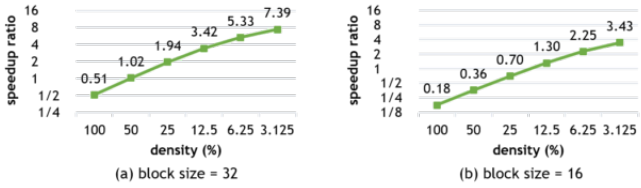


Figure 4. Speedup of cuSPARSE Block-SpMM over Dense GEMM in cuBLAS on NVIDIA A100, fp16 in/out, fp32 compute, NN layout, CUDA Toolkit 11.2.1.

Block-SpMM code example

For this new storage format, perform similar steps as with CSR and COO `cusparseSpMM`. For more information, see the [cuSPARSE/spmm_blockedell](#) repo.

First, include the cuSPARSE header, set up some device pointers, and initialize the cuSPARSE handle:

```
#include <cusparse.h>
cusparseHandle_t handle = nullptr;
cusparseCreate(&handle);
float   alpha         = 1.0f;
float   beta          = 0.0f;
int*    d_ell_colidx  = ...
__half* d_ell_values  = ...
__half* dB            = ...
__half* dC            = …
int     ell_blocksize = 32;
```

Next, create the block sparse input matrix `A`, dense input matrix `B`, and dense output matrix `C` descriptors:

```
cusparseSpMatDescr_t matA;
cusparseDnMatDescr_t matB, matC;
cusparseCreateBlockedEll(&matA, A_num_rows, A_num_cols,
                         ell_blocksize, ell_cols,
                         d_ell_colidx, d_ell_values,
                         CUSPARSE_INDEX_32I, CUSPARSE_INDEX_BASE_ZERO,
                         AB_type);
cusparseCreateDnMat(&matB, B_num_rows, B_num_cols, B_ld,
                    d_B, AB_type, CUSPARSE_ORDER_ROW);
cusparseCreateDnMat(&matC, C_num_rows, C_num_cols, C_ld,
                    d_C, C_type, CUSPARSE_ORDER_ROW);
```

Then, allocate an external buffer for the multiplication:

```
void*  d_buffer;
cusparseSpMM_bufferSize(handle,
                        CUSPARSE_OPERATION_NON_TRANSPOSE,
                        CUSPARSE_OPERATION_NON_TRANSPOSE,
                        &alpha, matA, matB, &beta, matC, CUDA_R_32F,
                        CUSPARSE_SPMM_ALG_DEFAULT, &bufferSize);
cudaMalloc(&dBuffer, bufferSize);
```

Now you can execute SpMM:

```
cusparseSpMM(handle, opA, opB, alpha, matA, matB,
             beta, matC, compute_type,
             CUSPARSE_SPMM_ALG_DEFAULT, d_buffer);
```

Finally, destroy the cuSPARSE descriptors and handle and clean up the used memory:

```
cusparseDestroySpMat(matA);
cusparseDestroyDnMat(matB);
cusparseDestroyDnMat(matC);
cusparseDestroy(handle);
cudaFree(dBuffer);
```

Get started with cuSPARSE Block-SpMM

The cuSPARSE library now provides fast kernels for block SpMM exploiting NVIDIA Tensor Cores. With the Blocked-ELL format, you can compute faster than dense-matrix multiplication depending on the sparsity of the matrix. The latest version of cuSPARSE can be found in the CUDA Toolkit.

For more information, see the following resources:

- OpenAI Block-Sparse GPU Kernels
- Efficient Transformers: A Survey
- Generating Long Sequences with Sparse Transformers
- Fast Block Sparse Matrices for Pytorch
- cuSPARSE documentation