Efficient Quantized Sparse Matrix Operations on Tensor Cores

Shigang Li*, Kazuki Osawa*, Torsten Hoefler+
*School of Computer Science, Beijing University of Posts and Telecommunications
+Department of Computer Science, ETH Zurich
Efficient Quantized Sparse Matrix Operations on Tensor Cores
Shigang Li, Kazuki Osawa, Torsten Hoefler

SC22, Dallas, TX, USA
Nov. 2022
Model size is growing exponentially

Even if the model can be fitted in a single GPU (for example, by swapping parameters between host and device memory), the high number of compute operations required can result in unrealistically long training times without parallelization. For example, training a GPT-3 model with 175 billion parameters would take 36 years on eight V100 GPUs, or seven months with 312 V100 GPUs.
Models are also compressible

- **Sparsification**
  - **SpMM**
    1. Self-attention in sparse Transformers
    2. Forward pass of pruned models
    ...
  - **SDDMM**
    1. Attention score in sparse Transformers
    2. Backward pass of pruned models
    ...

- **Quantization**
  - Uniform symmetric quantization
    - \(-\max(|W_i|) \leq 0.0 \leq \max(|W_i|)\)
    - \([-128, 0, 127]\)
    - Real (fp16)
    - Integer (int8)
  - Uniform asymmetric quantization
    - \(-\max(W_i) \leq 0.0 \leq \max(W_i)\)
    - \([-128, 0, 127]\)
    - Real (fp16)
    - Integer (int8)

- **Combining sparsification with quantization**
  - Mart van Baalen et al., Bayesian bits: Unifying quantization and pruning, NeurIPS 2020
  - H. Yang et al., Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization based approach, CVPR 2020
  - S. Han et al., Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, ICLR 2016
  ...

Sparsity in scientific: > 99%
Sparsity in DL: 50% ~ 90%
Tensor cores for deep learning acceleration

Challenges

(1) How to achieve practical speedup in a large range of sparsity ratio, e.g., 50% ~ 98%?

(2) How to efficiently support sparse workloads with mixed precision (two input matrices with different precision), e.g., 8-bit weights and 4-bit activation?

![Graph showing speedup vs. sparsity ratio for different tensor cores]
Libraries of sparse matrix computation

1 Mixed precision means two input matrices with different precision

<table>
<thead>
<tr>
<th>Library</th>
<th>Precision</th>
<th>Sparsity</th>
<th>Tensor Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fp16</td>
<td>int8</td>
<td>int4</td>
</tr>
<tr>
<td>cuSPARSE [10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sputnik [13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>vectorSparse [14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Magicube (ours)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Data layout of $m8n8k16$ for int8 mma on Tensor Cores

Thread0 provides $b_{00}$, $b_{10}$, $b_{20}$, and $b_{30}$, and each $b_{xx}$ is an 8-bit integer.

Thread0 provides $a_{00}$, $a_{01}$, $a_{02}$, and $a_{03}$, and each $a_{xx}$ is an 8-bit integer.

### A (row-major)

<table>
<thead>
<tr>
<th>Col</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>0</td>
<td>T0: ${a_{00}, a_{01}, a_{02}, a_{03}}$</td>
<td>T1: ${a_{04}, a_{05}, a_{06}, a_{07}}$</td>
<td>T2: ${a_{08}, a_{09}, a_{10}, a_{11}}$</td>
<td>T3: ${a_{12}, a_{13}, a_{14}, a_{15}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>T4: ${a_{16}, a_{17}, a_{18}, a_{19}}$</td>
<td>T5: ${a_{20}, a_{21}, a_{22}, a_{23}}$</td>
<td>T6: ${a_{24}, a_{25}, a_{26}, a_{27}}$</td>
<td>T7: ${a_{28}, a_{29}, a_{30}, a_{31}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T28: ${a_{56}, a_{57}, a_{58}, a_{59}}$</td>
<td>T29: ${a_{60}, a_{61}, a_{62}, a_{63}}$</td>
<td>T30: ${a_{64}, a_{65}, a_{66}, a_{67}}$</td>
<td>T31: ${a_{68}, a_{69}, a_{70}, a_{71}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### B (column-major)

<table>
<thead>
<tr>
<th>Row</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col</td>
<td>T0: ${b_{00}, b_{01}, b_{02}, b_{03}}$</td>
<td>T1: ${b_{04}, b_{05}, b_{06}, b_{07}}$</td>
<td>T2: ${b_{08}, b_{09}, b_{10}, b_{11}}$</td>
<td>T3: ${b_{12}, b_{13}, b_{14}, b_{15}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T4: ${b_{16}, b_{17}, b_{18}, b_{19}}$</td>
<td>T5: ${b_{20}, b_{21}, b_{22}, b_{23}}$</td>
<td>T6: ${b_{24}, b_{25}, b_{26}, b_{27}}$</td>
<td>T7: ${b_{28}, b_{29}, b_{30}, b_{31}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T8: ${b_{32}, b_{33}, b_{34}, b_{35}}$</td>
<td>T9: ${b_{36}, b_{37}, b_{38}, b_{39}}$</td>
<td>T10: ${b_{40}, b_{41}, b_{42}, b_{43}}$</td>
<td>T11: ${b_{44}, b_{45}, b_{46}, b_{47}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T12: ${b_{48}, b_{49}, b_{50}, b_{51}}$</td>
<td>T13: ${b_{52}, b_{53}, b_{54}, b_{55}}$</td>
<td>T14: ${b_{56}, b_{57}, b_{58}, b_{59}}$</td>
<td>T15: ${b_{60}, b_{61}, b_{62}, b_{63}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### n = 8

### k = 16

$m8n8k16$ represents a data layout in which $m = 8$, $n = 8$, and $k = 16$. Each $a_{xx}$ and $b_{xx}$ is an 8-bit integer. The layout is designed to optimize performance on Tensor Cores.
SR-BCRS sparse matrix format

(a) Sparse Matrix

Sparse matrix with 1-D block non-zeros, the length of the 1-D block = 2, 4, or 8

(b) BCRS format

Row pointers = [0, 4, 10, 13]
Column indices = [1, 3, 6, 8, 1, 2, 5, 7, 8, 11, 0, 4, 9]
Values = abcdefghijklmnpqrstuvwxyz

(c) SR-BCRS format (stride=4)

Row pointers = [0, 4, 10, 12, 15]
Column indices = [1, 3, 6, 8, 1, 2, 5, 7, 8, 11, *, *, 0, 4, 9, *]
Values = abcdefghijklmnopqrstuvwxyz

SR-BCRS (ours) is more friendly to Tensor Cores

Matrix A for mma

A (row-major)

<table>
<thead>
<tr>
<th>Col</th>
<th>Row</th>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>T0: (a, a, a, a, a) T1: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T2: (a, a, a, a, a) T3: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>T4: (a, a, a, a, a) T5: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>T6: (a, a, a, a, a) T7: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>T8: (a, a, a, a, a) T9: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>T10: (a, a, a, a, a) T11: (a, a, a, a, a)</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>T12: (a, a, a, a, a) T13: (a, a, a, a, a)</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>T14: (a, a, a, a, a) T15: (a, a, a, a, a)</td>
</tr>
</tbody>
</table>

SR-BCRS (ours) is more friendly to Tensor Cores
SpMM in Magicube

(a) SpMM

(b) SpMM in Magicube at thread-block level
Load rows of matrix B to shared memory for int8

<table>
<thead>
<tr>
<th>warp</th>
<th>bank</th>
<th>8-bit integer</th>
<th>Bank</th>
<th>BSk = 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>T4</td>
<td>0 ~ 7</td>
<td>T24</td>
<td>T28</td>
</tr>
<tr>
<td>T0</td>
<td>T4</td>
<td>16 ~ 23</td>
<td>T24</td>
<td>T28</td>
</tr>
<tr>
<td>T0</td>
<td>T4</td>
<td>0 ~ 7</td>
<td>T24</td>
<td>T28</td>
</tr>
<tr>
<td>T0</td>
<td>T4</td>
<td>16 ~ 23</td>
<td>T24</td>
<td>T28</td>
</tr>
<tr>
<td>T1</td>
<td>T5</td>
<td>0 ~ 7</td>
<td>T25</td>
<td>T29</td>
</tr>
<tr>
<td>T1</td>
<td>T5</td>
<td>16 ~ 23</td>
<td>T25</td>
<td>T29</td>
</tr>
<tr>
<td>T1</td>
<td>T5</td>
<td>0 ~ 7</td>
<td>T25</td>
<td>T29</td>
</tr>
<tr>
<td>T1</td>
<td>T5</td>
<td>16 ~ 23</td>
<td>T25</td>
<td>T29</td>
</tr>
<tr>
<td>T2</td>
<td>T6</td>
<td>0 ~ 7</td>
<td>T26</td>
<td>T30</td>
</tr>
<tr>
<td>T2</td>
<td>T6</td>
<td>16 ~ 23</td>
<td>T26</td>
<td>T30</td>
</tr>
<tr>
<td>T2</td>
<td>T6</td>
<td>0 ~ 7</td>
<td>T26</td>
<td>T30</td>
</tr>
<tr>
<td>T2</td>
<td>T6</td>
<td>16 ~ 23</td>
<td>T26</td>
<td>T30</td>
</tr>
<tr>
<td>T3</td>
<td>T7</td>
<td>0 ~ 7</td>
<td>T27</td>
<td>T31</td>
</tr>
<tr>
<td>T3</td>
<td>T7</td>
<td>16 ~ 23</td>
<td>T27</td>
<td>T31</td>
</tr>
<tr>
<td>T3</td>
<td>T7</td>
<td>0 ~ 7</td>
<td>T27</td>
<td>T31</td>
</tr>
<tr>
<td>T3</td>
<td>T7</td>
<td>16 ~ 23</td>
<td>T27</td>
<td>T31</td>
</tr>
</tbody>
</table>

4-way bank conflict

Coalesced into a 64-byte transaction
Load blocks of matrix B to shared memory for \textit{int8}

- **8-bit integer**: one bank
- **BS\textsubscript{n} = 64**
- **BS\textsubscript{k} = 16**

### Bank conflict-free

- **WS0**: Bank 0 ~ 7
- **WS1**: Bank 8 ~ 15
- **WS2**: Bank 16 ~ 23
- **WS3**: Bank 24 ~ 31

### Padding:

- **WS0**: Bank 0 ~ 7
- **WS1**: Bank 16 ~ 23
- **WS2**: Bank 0 ~ 7
- **WS3**: Bank 16 ~ 23
Local transpose on registers for int8

<table>
<thead>
<tr>
<th>Shared memory</th>
<th>8-bit integer</th>
<th>BSₙ = 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank0</td>
<td>T0</td>
<td></td>
</tr>
<tr>
<td>bank16</td>
<td>T0</td>
<td></td>
</tr>
<tr>
<td>bank0</td>
<td>T0</td>
<td></td>
</tr>
<tr>
<td>bank16</td>
<td>T0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>load</th>
<th>one bank</th>
<th>Padding: Bank 0 ~ 7</th>
<th>BSₖ = 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  4  8 12</td>
<td>T0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1  5  9 13</td>
<td>T4 Bank 0 ~ 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2  6 10 14</td>
<td>T4 Bank 16 ~ 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3  7 11 15</td>
<td>T4 Bank 0 ~ 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registers</th>
<th>Bank 8 ~ 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>Bank 16 ~ 23</td>
</tr>
<tr>
<td>4 5 6 7</td>
<td>Bank 0 ~ 7</td>
</tr>
<tr>
<td>8 9 10 11</td>
<td>Bank 16 ~ 23</td>
</tr>
<tr>
<td>12 13 14 15</td>
<td>Bank 0 ~ 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>transpose in char</th>
<th>Warp0</th>
<th>Warp1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Local transpose on registers for \textit{int8}
MMAs in SpMM with \textit{int8}

The warp-level view of MMAs in SpMM with \textit{int8}

8-bit integer

Shared memory

\begin{align*}
\text{bank0} & \quad \text{bank16} \\
\text{bank0} & \quad \text{bank16}
\end{align*}

\text{load} \quad \text{transpose in char}

\begin{align*}
\begin{array}{cccc}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{array}
\end{align*}

\begin{align*}
\begin{array}{cccc}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{array}
\end{align*}

\begin{align*}
\text{banks} & \quad \text{banks} \\
\text{T0} & \quad \text{T0} \\
\text{T0} & \quad \text{T0} \\
\text{bank0} & \quad \text{bank16} \\
\text{bank0} & \quad \text{bank16}
\end{align*}

\text{Shared memory}

\begin{align*}
\text{mm0 by warp0} & \quad \text{mm1 by warp0} & \quad \text{mm2 by warp0} & \quad \text{mm3 by warp0} \\
\text{mm0 by warp1} & \quad \text{mm1 by warp1} & \quad \text{mm2 by warp1} & \quad \text{mm3 by warp1}
\end{align*}

\begin{align*}
\begin{array}{cccc}
\text{input of mma0} & \quad \text{input of mma1} & \quad \text{input of mma2} & \quad \text{input of mma3}
\end{array}
\end{align*}

\text{input of mma0} \quad \text{input of mma1} \quad \text{input of mma2} \quad \text{input of mma3}

\text{input of mma0} \quad \text{input of mma1} \quad \text{input of mma2} \quad \text{input of mma3}

The warp-level view of MMAs in SpMM with \textit{int8}

8-bit integer

\begin{align*}
\begin{array}{cccc}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{array}
\end{align*}

\begin{align*}
\begin{array}{cccc}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{array}
\end{align*}

\begin{align*}
\begin{array}{cccc}
\text{input of mma0} & \quad \text{input of mma1} & \quad \text{input of mma2} & \quad \text{input of mma3}
\end{array}
\end{align*}

\text{input of mma0} \quad \text{input of mma1} \quad \text{input of mma2} \quad \text{input of mma3}

\text{input of mma0} \quad \text{input of mma1} \quad \text{input of mma2} \quad \text{input of mma3}

The warp-level view of MMAs in SpMM with \textit{int8}
Efficient local transpose for \textit{int4} with indices shuffling

1. Block-wise indices shuffling

2. Load data of the dense matrix \(B\) (via SM) to registers

3. Cast to \textit{char}

4. Transpose in \textit{char}

5. Split

6. Mask and shift

7. Bitwise OR

192 bitwise operations

32 bitwise operations
Prefetch data blocks of matrix B of SpMM

Algorithm 1 Prefetch the data block of dense matrix B

\[
\text{steps} = \frac{\text{nnz}}{BS_k};
\]

- Load data and indices to SM
- Overlap prefetch with MMA
- The tail of pipeline

\[
\begin{align*}
\text{Cold start} & : \text{Load A values and indices to shared(0);} \\
& \quad \_\text{syncnthreads();} \\
& \quad \text{Prefetch B values to registers(0);} \\
\text{for i=1; i < steps; i++ do} & : \text{Load data and indices to SM} \\
& \quad \text{Store B values on regs to shared(i-1);} \\
& \quad \text{Load A values and indices to shared(i);} \\
& \quad \_\text{syncnthreads();} \\
& \quad \text{Prefetch B values to registers(i);} \\
& \quad \text{MMA compute tiles(i-1);} \\
& \quad \_\text{syncnthreads();} \\
\text{end for} & : \text{The tail of pipeline}
\end{align*}
\]
SDDMM in Magicube

(a) SDDMM

(b) SDDMM in Magicube at thread-block level
MMAs in SDDMM

The thread-block level view of SDDMM

The warp-level view of MMAs in SDDMM
Mixed precision

- **a** is an 8-bit **unsigned** integer, **b** is unsigned 4-bit
  
  \[ a = 11101101 \text{ (237 in decimal)} \]
  
  Split
  
  \[ a_{7-4} \quad a_{3-0} \]
  
  \[ \begin{array}{c}
  1110 \\
  1101 \\
  \end{array} \]
  
  unsigned \quad unsigned
  
  Recover
  
  \[ a = 2^4 \times a_{7-4} + a_{3-0} \]
  
  \[ a \times b = 2^4 \times a_{7-4} \times b + a_{3-0} \times b \]

- **a** is an 8-bit **signed** integer, **b** is signed 4-bit
  
  \[ a = 11101101 \text{ (-19 in decimal)} \]
  
  Split
  
  \[ A_{0} \quad A_{1} \]
  
  \[ \begin{array}{c}
  1110 \\
  1101 \\
  \end{array} \]
  
  signed \quad unsigned
  
  (higher 4-bit)
  
  \[ \begin{array}{c}
  1101 \\
  \end{array} \]
  
  signed
  
  \[ C_{0} \quad C_{1} \]
  
  (32-bit)
  
  C0
  
  C1
  
  (b) Stacked into a single mma

(a) Emulation of \( A \) (8-bit) \( \times B \) (4-bit) using 4-bit mma

\[ a \times b = 2^0 \times C_{0} + 2^4 \times C_{1} \]
Evaluation

- NVIDIA A100-SXM4-40GB GPU
  - total 108 SMs
  - each SM has 192KB configurable L1 cache and shared memory, and 256KB registers
  - supported datatypes on Tensor Core: int8, int4, int1, fp16, bf16, tf32, fp64

- Compare the performance of Magicube with sparse libraries (vectorSparse, cuSPARSE) and dense libraries (cuBLAS, cuDNN)

- Micro-benchmarks: 1,536 sparse matrices from Deep Learning Matrix Collection (DLMC) with sparsity 50%~98%, dilating each scalar with 1-D blocks (length V = 2, 4, 8)

- Case study: end-to-end sparse Transformer inference

One streaming multiprocessor (SM) of GA100

Ablation study for SpMM in Magicube

Ablation study for optimizations of SpMM
SpMM with mixed precision in Magicube

SpMM with mixed precision

$L_x$-$R_y$ means $x$-bit A matrix multiplied by $y$-bit B matrix
Benchmarking SpMM and SDDMM

SpMM on A100 GPU using 1,536 sparse matrices

V = 2
N = 128

\[ \text{Speedup} = 2.43 \times \text{median} \]

V = 4
N = 128

\[ \text{Speedup} = 2.38 \times \text{median} \]

V = 8
N = 128

\[ \text{Speedup} = 1.34 \times \text{median} \]

SDDMM on A100 GPU using 1,536 sparse matrices

V = 2
K = 128

\[ \text{Speedup} = 1.18 \times \text{median} \]

V = 4
K = 128

\[ \text{Speedup} = 1.32 \times \text{median} \]

V = 8
K = 128

\[ \text{Speedup} = 1.56 \times \text{median} \]
End-to-end sparse Transformer inference

Attention($Q, K, V$) = softmax \left( \frac{QK^T \odot M}{\sqrt{d_k}} \right) V

Quantized self-attention with sparse attention mask

Latency of end-to-end inference of sparse Transformer

Lower is better
End-to-end sparse Transformer inference

Test accuracy of text classification using sparse Transformer with num_heads=4 and seq_len=4,096
Conclusion

1. Challenges

2. SR-BCRS format

3. SpMM in Magicube

4. SDDMM in Magicube

5. Mixed precision

6. Evaluation

https://zenodo.org/record/6924338
https://github.com/Shigangli/Magicube