Efficient Block Algorithms for Parallel Sparse Triangular Solve

Zhengyang Lu, Yuyao Niu, Weifeng Liu

China University of Petroleum, Beijing
Outline

1. Background
2. Motivation
3. Block Algorithms
4. Recursive Block Algorithm
5. Experiment
6. Our Source Code
7. Conclusion
Sparse Triangular Solve (SpTRSV)

The sparse triangular solve (SpTRSV) is the solution operation of linear equation $Lx = B$ (or $Ux = B$), where $L$ (or $U$) is an upper triangle matrix (or lower triangle matrix), $b$ is a dense right-hand vector, and $x$ is a result vector to be solved.
The sparse triangular solve (SpTRSV) is the solution operation of linear equation \( Lx = B \) (or \( Ux = B \)), where \( L \) (or \( U \)) is an upper triangle matrix (or lower triangle matrix), \( b \) is a dense right-hand vector, and \( x \) is a result vector to be solved.
Background

Sparse Triangular Solve (SpTRSV)

The sparse triangular solve (SpTRSV) is the solution operation of linear equation \( Lx = B \) (or \( Ux = B \)), where \( L \) (or \( U \)) is an upper triangle matrix (or lower triangle matrix), \( b \) is a dense right-hand vector, and \( x \) is a result vector to be solved.

\[
\begin{align*}
1 & \times x_0 = a \\
2 & \times x_0 + 1 & \times x_1 = b
\end{align*}
\]
The sparse triangular solve (SpTRSV) is the solution operation of linear equation $Lx = B$ (or $Ux = B$), where $L$ (or $U$) is an upper triangle matrix (or lower triangle matrix), $b$ is a dense right-hand vector, and $x$ is a result vector to be solved.

The system is given by:

\[
\begin{align*}
1 \times x_0 &= a \\
2 \times x_0 + 1 \times x_1 &= b \\
2 \times x_1 + 1 \times x_2 &= c
\end{align*}
\]

The solution is:

\[
\begin{align*}
x_0 &= a \\
x_1 &= b - 2a \\
x_2 &= c - 2b + 4a
\end{align*}
\]
Sparse Triangular Solve (SpTRSV)

The sparse triangular solve (SpTRSV) is the solution operation of linear equation $Lx = B$ (or $Ux = B$), where $L$ (or $U$) is an upper triangle matrix (or lower triangle matrix), $b$ is a dense right-hand vector, and $x$ is a result vector to be solved.
Background

Sparse Triangular Solve (SpTRSV)

The sparse triangular solve (SpTRSV) is the solution operation of linear equation \( Lx = B \) (or \( Ux = B \)), where \( L \) (or \( U \)) is an upper triangle matrix (or lower triangle matrix), \( b \) is a dense right-hand vector, and \( x \) is a result vector to be solved.

\[
\begin{align*}
1 \times x_0 &= a \\
2 \times x_0 + 1 \times x_1 &= b \\
2 \times x_1 + 1 \times x_2 &= c \\
3 \times x_0 + 1 \times x_3 &= d
\end{align*}
\]

\[
\begin{align*}
x_0 &= a \\
x_1 &= b - 2a \\
x_2 &= c - 2b + 4a \\
x_3 &= d - 3a
\end{align*}
\]
Level-set Parallel SpTRSV

This method sees a matrix as a graph and divides its components into multiple sets. The components in each set do not depend on each other, so they can be solved in parallel.
This method sees a matrix as a graph and divides its components into multiple sets. The components in each set do not depend on each other, so they can be solved in parallel.
Motivation

1. We aim to make the data locality better, through dividing the triangular matrix.
Motivation

1. We aim to make the data locality better, through dividing the triangular matrix.

2. We in this work exploit 2D block layout to naturally cut those long rows and columns into shorter segments.
Motivation

1. We aim to make the data locality better, through dividing the triangular matrix.

2. We in this work exploit 2D block layouts to naturally cut those long rows and columns into shorter segments.

3. We in this paper will find the best formulation from various blocking methods and their parameters, and accelerate SpTRSV on modern GPUs.
A matrix can be divided into a number of column blocks, and each one consists of a triangular submatrix on top and a rectangular sub-matrix on bottom.

The arrows indicate the processing order of the blocks.
The row block algorithm cuts a matrix horizontally. Now each row block includes a rectangular sub-matrix on the left and a triangular sub-matrix on the right.

The arrows indicate the processing order of the blocks.
recursive block approach divides a triangular matrix into two smaller triangular and one square or near square blocks, and the two triangular sub-matrices could be further recursively divided into the three parts.
Block Algorithms
Comparison of the Three Block Algorithms

- The arrows indicate the processing order of the blocks.
- The bold blue and red lines mean reading x and updating b, respectively.
Block Algorithms

Comparison of the Three Block Algorithms

- The arrows indicate the processing order of the blocks.
- The bold blue and red lines means reading $x$ and updating $b$, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula $(n \text{ is } #\text{rows})$</th>
<th>#triangular parts divided</th>
</tr>
</thead>
<tbody>
<tr>
<td>col. block</td>
<td>$2^{x-1}n + 0.5n$</td>
<td>2.5$n$</td>
</tr>
<tr>
<td>row block</td>
<td>$2n - 2^{x-1}n$</td>
<td>1.75$n$</td>
</tr>
<tr>
<td>rec. block</td>
<td>$0.5nx + n$</td>
<td>2$n$</td>
</tr>
</tbody>
</table>

Table 1: The number of items updated to right-hand side $b$.  

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula $(n \text{ is } #\text{rows})$</th>
<th>#triangular parts divided</th>
</tr>
</thead>
<tbody>
<tr>
<td>col. block</td>
<td>$n - 2^{x-1}n$</td>
<td>0.75$n$</td>
</tr>
<tr>
<td>row block</td>
<td>$2^{x-1}n - 0.5n$</td>
<td>1.5$n$</td>
</tr>
<tr>
<td>rec. block</td>
<td>$0.5nx$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

Table 2: The number of items loaded from solution vector $x$.  

Block Algorithms

Comparison of the Three Block Algorithms

It is clear to see that our recursive block algorithm achieves a good tradeoff between the column and row block methods.

- The bold blue and red lines means reading $x$ and updating $b$, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula ($n$ is #rows)</th>
<th>#triangular parts divided</th>
</tr>
</thead>
<tbody>
<tr>
<td>col. block</td>
<td>$2^{x-1}n + 0.5n$</td>
<td>4  16  256  65536</td>
</tr>
<tr>
<td>row block</td>
<td>$2n - 2^{-x}n$</td>
<td>1.75n 1.94n 1.99n 1.99n</td>
</tr>
<tr>
<td>rec. block</td>
<td>$0.5nx + n$</td>
<td>2n   3n   5n   9n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula ($n$ is #rows)</th>
<th>#triangular parts divided</th>
</tr>
</thead>
<tbody>
<tr>
<td>col. block</td>
<td>$n - 2^{-x}n$</td>
<td>0.75n 0.94n 0.99n 0.99n</td>
</tr>
<tr>
<td>row block</td>
<td>$2^{x-1}n - 0.5n$</td>
<td>1.5n  7.5n  127.5n 32767.5n</td>
</tr>
<tr>
<td>rec. block</td>
<td>$0.5nx$</td>
<td>n    2n   4n   8n</td>
</tr>
</tbody>
</table>
Block Algorithms

Comparison of the Three Block Algorithms

We evaluate the three block algorithms by running two representative sparse matrices on an NVIDIA Titan RTX GPU and plot their execution time on SpMV in the following figure.
Block Algorithms

Comparison of the Three Block Algorithms

We evaluate the three block algorithms by running two representative sparse matrices on an NVIDIA Titan RTX GPU and plot their execution time on SpMV in the following figure.

As a result, we will focus on further accelerating the recursive block data structure and algorithm.
Recursive Block Algorithm

Improved Recursive Block Data Structure

- The original input matrix $L$ in Figure (a) is reordered according to its level-set order and turned into Figure (b).
- Two triangular parts in Figure (b) are further reordered according to their level-set orders, and the resulting matrix is the one shown in Figure (c).
The original input matrix $L$ in Figure (a) is reordered according to its level-set order and turned into Figure (b).

Two triangular parts in Figure (b) are further reordered according to their level-set orders, and the resulting matrix is the one shown in Figure (c)

nnz (square) = 8

nnz (square) = 11
Recursive Block Algorithm

Store Matrix into Arrays

We assume the input matrix is in the **CSC format** including col_ptr, row_idx and val arrays as shown in the following three arrays.
Recursive Block Algorithm

Store Matrix into Arrays

Once it is reordered and saved as three sub-matrices, the triangular parts can be still saved in CSC, but the square parts should be transposed into CSR for using faster SpMV kernel. The blue items in the three arrays in the following figure store the CSR data of the square part between the two CSC sub-matrices.
Recursive Block Algorithm

Store Matrix into Arrays

When the matrix is further blocked into seven sub-matrices, the two CSC sub-matrices are further stored with the same layout. In the following figure, two new squares are in light green and purple, and the triangular parts are in orange and light yellow.
Recursive Block Algorithm

Store Matrix into Arrays

In addition, the square blocks may be very sparse, meaning that a large portion of rows are probably empty. In such case, we store the CSR data with a simplified row pointer with an extra array saving the actual indices. We call this format DCSR in the paper.
Recursive Block Algorithm

Adaptive Kernel Selection for SpTRSV in Recursive Block Algorithm

<table>
<thead>
<tr>
<th>Four kinds of sparsity structures and four kernels for SpTRSV:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) diagonal structure</td>
</tr>
<tr>
<td>(2) a small number of level-sets</td>
</tr>
<tr>
<td>(3) tens of or more than hundreds of levels</td>
</tr>
<tr>
<td>(4) near serial structure</td>
</tr>
</tbody>
</table>
Recursive Block Algorithm

Adaptive Kernel Selection for SpTRSV in Recursive Block Algorithm

Four kinds of sparsity structures and four kernels for SpTRSV:
(1) diagonal structure  perfect parallelism
(2) a small number of level-sets  level-set algorithm
(3) tens of or more than hundreds of levels  Sync-free algorithm
(4) near serial structure  cuSPARSE

Two critical parameters:
(1) nnz/row indicating the average row length of a sub-matrix
(2) nlevels giving the number of level-sets in the triangular part.
Adaptive Kernel Selection for SpTRSV in Recursive Block Algorithm

data set: we divide the 159 sparse matrices into submatrices of various sizes, and run all kinds of kernels on the Titan RTX GPU to collect a large amount of performance data. 203,251 sets of SpTRSV performance data and 170,563 sets of SpMV data are collected.
Recursive Block Algorithm

Adaptive Kernel Selection for SpTRSV in Recursive Block Algorithm

**Data Set:** We divide the 159 sparse matrices into submatrices of various sizes, and run all kinds of kernels on the Titan RTX GPU to collect a large amount of performance data. **203,251 Sets** of SpTRSV performance data and **170,563 Sets** of SpMV data are collected.

**Selection Principle of Kernel:**
1. nnz/row ≤ 15 and nlevels ≤ 20 _levelset_
2. nnz/row = 1 and nlevels ≤ 100. _levelset_
3. nlevels > 20000 _cuSPARSE_
4. In the other area, Sync-free is almost always the best choice.
5. Note that the completely parallel case is easily selected and not plotted in the figure.

(a) Best SpTRSV kernels matching parameters nnz/row and nlevels
Recursive Block Algorithm

Adaptive Kernel Selection for SpMV in Recursive Block Algorithm

Two kinds of sparsity structures and Two kernels for SpMV:
(1) structure with short rows scalar-CSR
(2) structure with long rows vector-CSR
For SpMV kernel selection, we use two parameters:
(1) the \texttt{nnz/row} mentioned above.
(2) \texttt{emptyratio} reflecting the ratio of the number of empty rows and the total number of rows.

Two kinds of sparsity structures and Two kernels for SpMV:
(1) structure with short rows \texttt{scalar-CR}
(2) structure with long rows \texttt{vector-CR}
Recursive Block Algorithm
Adaptive Kernel Selection for SpMV in Recursive Block Algorithm

data set: we divide the 159 sparse matrices into submatrices of various sizes, and run all kinds of kernels on the Titan RTX GPU to collect a large amount of performance data. 203,251 sets of SpTRSV performance data and 170,563 sets of SpMV data are collected.
Recursive Block Algorithm

Adaptive Kernel Selection for SpMV in Recursive Block Algorithm

Data set: We divide the 159 sparse matrices into submatrices of various sizes, and run all kinds of kernels on the Titan RTX GPU to collect a large amount of performance data. 203,251 sets of SpTRSV performance data and 170,563 sets of SpMV data are collected.

Selection principle of kernel:
(1) When nnz/row ≤ 12 and emptyratio ≤ 50% the kernels scalar-CSR work best.
(2) When nnz/row ≤ 12 and emptyratio > 50% the kernels scalar-DCSR work best.
(3) When nnz/row > 12 and emptyratio ≤ 15% vector-CSR should be used.
(4) When nnz/row > 12 and emptyratio > 15% vector DCSR should be used.
Recursive Block Algorithm

Adaptive Kernel Selection for SpTRSV and SpMV in Recursive Block Algorithm

The algorithm gives a decision tree for selecting the best kernels in the improved recursive block algorithm.
We select 159 sparse matrices from the SuiteSparse Matrix Collection. They are selected through three filter conditions:

1. square matrices.
2. the number of rows should be no less than 500,000.
3. the number of nonezeros should be no less than 5,000,000 and no greater than 500,000,000.
Experiment

SpTRSV Performance Comparison

- Performance (in GFlops) of the three SpTRSV methods
- Speedups of our block algorithm over the cuSPARSE v2 (top) and Sync-free (bottom) methods
On Titan X, the best speedups over cuSPARSE and Syncfree are 113.84x and 57.97x, and the average speedups are 5.00x and 10.34x, respectively.

On Titan RTX, the best speedups over cuSPARSE and Syncfree are 72.03x and 61.08x, and the average speedups are 4.72x and 9.95x, respectively.
Figure: Performance ratio of double precision to single precision in box plots of running the three SpTRSV algorithms on the 159 matrices on the Titan X and Titan RTX GPUs.
Figure: Performance ratio of double precision to single precision in box plots of running the three SpTRSV algorithms on the 159 matrices on the Titan X and Titan RTX GPUs.

Sync-free is around 0.9
block algorithm is between 0.8 and 0.9
cuSPARSE is between 0.7 and 0.8
## Experiment

Preprocessing Cost Comparison

<table>
<thead>
<tr>
<th>Methods</th>
<th>Preprocessing time</th>
<th>Single SpTRSV time</th>
<th>Overall time of 100 iters</th>
<th>Overall time of 500 iters</th>
<th>Overall time of 1000 iters</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuSPARSE v2</td>
<td>91.32</td>
<td>103.09</td>
<td>10400.71</td>
<td>51638.30</td>
<td>103185.29</td>
</tr>
<tr>
<td>Sync-free</td>
<td>2.34</td>
<td>94.79</td>
<td>9461.10</td>
<td>47396.15</td>
<td>94789.96</td>
</tr>
<tr>
<td>block algorithm</td>
<td>104.44</td>
<td>11.40</td>
<td>1244.05</td>
<td>5802.48</td>
<td>11500.52</td>
</tr>
</tbody>
</table>

Table: The average time (in milliseconds) for preprocessing an input matrix, completing a single SpTRSV and solving problems of a preprocessing and 100, 500 and 1000 SpTRSV
Experiment

Preprocessing Cost Comparison

<table>
<thead>
<tr>
<th>Methods</th>
<th>Preprocessing time</th>
<th>Single SpTRSV time</th>
<th>Overall time of 100 iters</th>
<th>Overall time of 500 iters</th>
<th>Overall time of 1000 iters</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuSPARSE v2</td>
<td>91.32</td>
<td>103.09</td>
<td>10400.71</td>
<td>51638.30</td>
<td>103185.29</td>
</tr>
<tr>
<td>Sync-free</td>
<td>2.34</td>
<td>94.79</td>
<td>9481.10</td>
<td>47396.15</td>
<td>94789.96</td>
</tr>
<tr>
<td>block algorithm</td>
<td>104.44</td>
<td>11.40</td>
<td>1244.05</td>
<td>5802.48</td>
<td>11500.52</td>
</tr>
</tbody>
</table>

As can be seen, our method uses a bit longer preprocessing time, but the overall cost for a complete computation with a number of iterations is still obviously lower than cuSPARSE and Sync-free, because of the much faster SpTRSV computation.

Table: The average time (in milliseconds) for preprocessing an input matrix, completing a single SpTRSV and solving problems of a preprocessing and 100, 500 and 1000 SpTRSV
Source code

https://github.com/LUZY0726/-
1. We have implemented three block algorithms for parallel SpTRSV on modern GPUs.

2. We proposed an adaptive approach that automatically selects the best kernels and parameters for computing sub-matrices divided.

3. Our experiments conducted on 159 matrices and on two high-end NVIDIA GPUs have shown on average 4.72x (up to 72.03x) and 9.95x (up to 61.08x) speedups over cuSPARSE and Sync-free methods, respectively.
Thank you!

Efficient Block Algorithms for Parallel Sparse Triangular Solve

Zhengyang Lu, Yuyao Niu, Weifeng Liu

China University of Petroleum, Beijing

Contact us: 2017010055@student.cup.edu.cn
               2019211256@student.cup.edu.cn
               weifeng.liu@cup.edu.cn