LightSpMV [3] is a novel CUDA-compatible sparse matrix-vector multiplication (SpMV) algorithm using the standard compressed sparse row (CSR) storage format. It achieves high speed by benefiting from the fine-grained dynamic distribution of matrix rows over warps, where a warp is virtualized as a single instruction multiple data (SIMD) vector and can be further split into a set of equal-sized smaller vectors for finer-grained processing.

In LightSpMV, we have investigated two dynamic row distribution approaches at the vector and warp levels with atomic operations and warp shuffle functions as the fundamental building blocks. We have evaluated LightSpMV using various sparse matrices and further compared it to the CSR-based SpMV subprograms in the state-of-the-art CUSP [2] and cuSPARSE [3] libraries. Performance evaluation reveals that on a single Tesla K40c GPU, LightSpMV is superior to both CUSP and cuSPARSE, with a speedup of up to 2.63 and 2.65 over CUSP and up to 1.93 and 1.79 over cuSPARSE for single and double precision, respectively. The source code of LightSpMV is available at http://lightspmv.sourceforge.net.

Abstract

LightSpMV [3] is a novel CUDA-compatible sparse matrix-vector multiplication (SpMV) algorithm using the standard compressed sparse row (CSR) storage format. It achieves high speed by benefiting from the fine-grained dynamic distribution of matrix rows over warps, where a warp is virtualized as a single instruction multiple data (SIMD) vector and can be further split into a set of equal-sized smaller vectors for finer-grained processing.

In LightSpMV, we have investigated two dynamic row distribution approaches at the vector and warp levels with atomic operations and warp shuffle functions as the fundamental building blocks. We have evaluated LightSpMV using various sparse matrices and further compared it to the CSR-based SpMV subprograms in the state-of-the-art CUSP [2] and cuSPARSE [3] libraries. Performance evaluation reveals that on a single Tesla K40c GPU, LightSpMV is superior to both CUSP and cuSPARSE, with a speedup of up to 2.63 and 2.65 over CUSP and up to 1.93 and 1.79 over cuSPARSE for single and double precision, respectively. The source code of LightSpMV is available at http://lightspmv.sourceforge.net.

Compressed Sparse Row (CSR) Format

- A frequently used format for sparse matrix storage in GPU-centric software
- Efficient compression of structured and unstructured sparse matrices
- Good amenability to efficient algorithms designed for GPUs
- Enables good performance on GPUs, but shows a relatively low performance on CPUs
- Uses three separate vectors: rowOffsets, columnIndices, and values to represent a matrix

Sparsity Matrix-Vector Multiplication

General SpMV equation:

\[
\mathbf{y} = \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}
\]

- \( \mathbf{A} \) is a sparse matrix of size \( R \times C \) with \( NZN \) non-zeros
- \( \alpha \) is the source vector of size \( R \)
- \( \beta \) is a scalar

Product of two vectors:

\[
\begin{align*}
\text{for } i = 0 & \rightarrow R - 1; \text{ do} \\
\text{compute the dot product of two vectors } & \text{ by } \sum_{j} (x_j \times y_j) \\
\text{for } j = 0 & \rightarrow C - 1; \text{ do} \\
\text{store } \sum_{j} (x_j \times y_j) & \\
\text{end for } j \\
\text{end for } i
\end{align*}
\]

Warp-Level Dynamic Row Distribution

- Only one atomic operation is required for a warp
- Distributes auxRip / Y rows to a single warp at a time
- Computes the warp-level CSR kernel by replacing the function getGlobalId() with the function __getGlobalId().

Double Precision Support

- Intra-vector reduction for double precision
- Overloads the __sdp_downd function for double precision
- Uses the reinterpret_cast compiler directive
- Uses the __sdp_downd function to compute the double-type row vector

Warp-Vector Dynamic Row Distribution

A Kepler-based Tesla K40c GPU and CUDA 6.5 toolkit

The vector-level kernel produces an average performance of 14.8 GFLOPS with the maximum performance of 27.4 GFLOPS for single precision, and an average performance of 12.2 GFLOPS with the maximum performance of 20.8 GFLOPS for double precision.

The warp-level kernel yields an average performance of 21.7 GFLOPS with the maximum performance of 32.0 GFLOPS for single precision, and an average performance of 16.6 GFLOPS with the maximum performance of 23.4 GFLOPS for double precision.

Performance Evaluation

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Warp / Vector (GFLOPS)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Double</td>
<td>Single</td>
</tr>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{ CSR-Scalar }</td>
<td>1.04</td>
<td>1.08</td>
</tr>
<tr>
<td>\text{ CSR-Vector }</td>
<td>1.02</td>
<td>1.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance comparison to CUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison to CUSP:</td>
</tr>
<tr>
<td>Acknowledgments:</td>
</tr>
<tr>
<td>CUDA:</td>
</tr>
<tr>
<td>CUSP:</td>
</tr>
<tr>
<td>cuSPARSE:</td>
</tr>
</tbody>
</table>

Compressed Sparse Row (CSR) Format

- A frequently used format for sparse matrix storage in GPU-centric software
- Efficient compression of structured and unstructured sparse matrices
- Good amenability to efficient algorithms designed for GPUs
- Enables good performance on GPUs, but shows a relatively low performance on CPUs
- Uses three separate vectors: rowOffsets, columnIndices, and values to represent a matrix

Sparsity Matrix-Vector Multiplication

General SpMV equation:

\[
\mathbf{y} = \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}
\]

- \( \alpha \) is a scalar
- \( \mathbf{A} \) is a sparse matrix of size \( R \times C \) with \( NZN \) non-zeros
- \( \mathbf{A} \) is the source vector of size \( R \)
- \( \mathbf{x} \) is the destination vector of size \( C \)
- \( \beta \) is a scalar

Resource and Cost

- \( \text{ CSR-Scalar } \) for single precision
- \( \text{ CSR-Vector } \) for double precision

Performance Evaluation

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Warp / Vector (GFLOPS)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Double</td>
<td>Single</td>
</tr>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{ CSR-Scalar }</td>
<td>1.04</td>
<td>1.08</td>
</tr>
<tr>
<td>\text{ CSR-Vector }</td>
<td>1.02</td>
<td>1.37</td>
</tr>
</tbody>
</table>

References