Flexible Batched Sparse Matrix-Vector Product on GPUs

Hartwig Anzt, Gary Collins, Jack Dongarra, Goran Flegar, Enrique, S. Quintana-Orti
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

Input $A, x, y$    Output $y = A \cdot x$

• Matrix $A$ contains only few nonzero elements.
• Storing all entries results in large overhead (memory & computation).
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- Matrix $A$ contains only few nonzero elements.
- Storing all entries results in large overhead (memory & computation).
- **Idea:** Store only nonzero elements [nz] explicitly.

\[
A = \begin{pmatrix}
5.4 & 1.1 & 0 & 0 & 0 & 0 \\
2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\
0 & 0 & 4.2 & 0 & 0 & 0 \\
5.4 & 0 & 0 & 9.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 8.1
\end{pmatrix}
\]

\[
\text{value} = [5.4 \ 1.1 \ 2.2 \ 8.3 \ 3.7 \ 1.3 \ 3.8 \ 4.2 \ 5.4 \ 9.2 \ 1.1 \ 8.1]
\]

Input $A, x, y$  Output $y = A \cdot x$
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- Matrix $A$ contains only few nonzero elements.
- Storing all entries results in large overhead (memory & computation).
- Idea: Store only nonzero elements [nz] explicitly.

Need to also store location of nonzero elements!

```
Memory footprint of COO format:
 nz(val) + 2*nz(int)
```

Input $A, x, y$  
Output $y = A \cdot x$

$$A = \begin{pmatrix}
5.4 & 1.1 & 0 & 0 & 0 & 0 \\
2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\
0 & 0 & 4.2 & 0 & 0 & 0 \\
5.4 & 0 & 0 & 9.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 8.1
\end{pmatrix}$$

**COO format:**

- Value
  $\text{value} = [ \begin{array}{cccccccccccc}
5.4 & 1.1 & 2.2 & 8.3 & 3.7 & 1.3 & 3.8 & 4.2 & 5.4 & 9.2 & 1.1 & 8.1 \\
\end{array} ]$

- Column-index
  $\text{colidx} = [ \begin{array}{cccccccccccc}
0 & 1 & 0 & 1 & 3 & 4 & 5 & 2 & 0 & 3 & 4 & 5 \\
\end{array} ]$

- Row-index
  $\text{rowidx} = [ \begin{array}{cccccccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 3 & 3 & 4 & 5 \\
\end{array} ]$
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- Matrix $A$ contains only few nonzero elements.
- Storing all entries results in large overhead (memory & computation).
- **Idea:** Store only nonzero elements [nz] explicitly.
  
  **Need to also store location of nonzero elements!**

$$A = \begin{pmatrix}
5.4 & 1.1 & 0 & 0 & 0 & 0 \\
2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\
0 & 0 & 4.2 & 0 & 0 & 0 \\
5.4 & 0 & 0 & 9.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 8.1 \\
\end{pmatrix}$$

Memory footprint of COO format:
$$\text{nz(val)} + 2*\text{nz(int)}$$

Memory footprint of CSR format:
$$\text{nz(val)} + \text{nz(int)} + (n+1) \text{ (int)}$$

**CSR format:**

- **value** = [ 5.4  1.1  2.2  8.3  3.7  1.3  3.8  4.2  5.4  9.2  1.1  8.1 ]
- **colidx** = [ 0  1  0  1  3  4  5  2  0  3  4  5 ]
- **rowptr** = [ 0  2  7  8  10  11  12 ]

Value

Column-index

Points to the first element in each row

Number of nonzero elements
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

How to parallelize this?

\[
A = \begin{pmatrix}
5.4 & 1.1 & 0 & 0 & 0 & 0 \\
2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\
0 & 0 & 4.2 & 0 & 0 & 0 \\
5.4 & 0 & 0 & 9.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 8.1
\end{pmatrix}
\]

value = [ 5.4  1.1  2.2  8.3  3.7  1.3  3.8  4.2  5.4  9.2  1.1  8.1 ]  Value

colidx = [ 0  1  0  1  3  4  5  2  0  3  4  5 ]  Column-index

rowidx = [ 0  0  1  1  1  1  1  2  3  3  4  5 ]  Row-index
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

How to parallelize this?

• Parallelize by rows:
  • Every “thread” handles the computation of one sum in local memory.
  • Significant workload imbalance!
  • Need branching logic, branch divergence on vector machines.
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

How to parallelize this?

- Parallelize by rows:
  - Every “thread” handles the computation of one sum in local memory.
  - Balanced workload.
  - Can result in significant overhead for unbalanced problems.

<table>
<thead>
<tr>
<th>T1</th>
<th>5.4 1.1 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>2.2 8.3 0 3.7 1.3 3.8</td>
</tr>
<tr>
<td>T3</td>
<td>0 0 4.2 0 0 0</td>
</tr>
<tr>
<td>T4</td>
<td>5.4 0 0 9.2 0 0</td>
</tr>
<tr>
<td>T5</td>
<td>0 0 0 0 1.1 0</td>
</tr>
<tr>
<td>T6</td>
<td>0 0 0 0 0 8.1</td>
</tr>
</tbody>
</table>

ELL format:

Values and column-index padded for uniform “row-length”

value = [ 5.4 1.1 0.0 0.0 0.0 2.2 8.3 3.7 1.3 3.8 4.2 0.0 0.0 0.0 0.0 0.0 0.0 5.4 9.2 0.0 0.0 0.0 0.0 1.1 0.0 0.0 0.0 0.0 0.0 8.1 0.0 0.0 0.0 ]

colidx = [ 0 1 – – – 0 1 3 4 5 0 – – – – 0 3 – – – 4 – – – 5 – – – ]

access to input vector x

* =

access to output vector y

Number of nonzero elements
How to parallelize this?

- Parallelize by rows:
  - Every “thread” handles the computation of one sum in local memory.
  - Significant workload imbalance!
  - “Ordered” access to input vector \( x \).

\[
\begin{align*}
T1 &= \begin{pmatrix} 5.4 & 1.1 & 0 & 0 & 0 & 0 \\ \\
T2 &= \begin{pmatrix} 2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\ \\
T3 &= \begin{pmatrix} 0 & 0 & 4.2 & 0 & 0 & 0 \\ \\
T4 &= \begin{pmatrix} 5.4 & 0 & 0 & 9.2 & 0 & 0 \\ \\
T5 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 1.1 & 0 \\ \\
T6 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 8.1 \\
\end{align*}
\]

Value = [ 5.4, 1.1, 2.2, 8.3, 3.7, 1.3, 3.8, 4.2, 5.4, 9.2, 1.1, 8.1 ]

Colidx = [ 0, 1, 0, 1, 3, 4, 5, 2, 0, 3, 4, 5 ]

Rowidx = [ 0, 0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5 ]
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

How to parallelize this?

- Parallelize by elements:
  - Balanced workload.
  - Partial sums need synchronization: Write conflicts!

<table>
<thead>
<tr>
<th>value</th>
<th>5.4</th>
<th>1.1</th>
<th>2.2</th>
<th>8.3</th>
<th>3.7</th>
<th>1.3</th>
<th>3.8</th>
<th>4.2</th>
<th>5.4</th>
<th>9.2</th>
<th>1.1</th>
<th>8.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>colidx</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>rowidx</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

access to input vector \( x \) = access to output vector \( y \)

- \( A \) never ending story: The sparse matrix vector Product (SpMV) on Manycore

Column-index

Row-index
How to parallelize this?

• Parallelize by elements:
  • Balanced workload.
  • Partial sums need synchronization: Write conflicts!

A never ending story: The sparse matrix vector Product (SpMV) on Manycore

\[
\begin{bmatrix}
5.4 & 1.1 & 0 & 0 & 0 & 0 \\
2.2 & 8.3 & 0 & 3.7 & 1.3 & 3.8 \\
0 & 0 & 4.2 & 0 & 0 & 0 \\
5.4 & 0 & 9.2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 8.1 \\
\end{bmatrix}
\]

value = [5.4 1.1 2.2 8.3 3.7 1.3 3.8 4.2 5.4 9.2 1.1 8.1]

colidx = [0 1 0 1 3 4 5 2 0 3 4 5]

rowidx = [0 0 1 1 1 1 1 2 3 3 4 5]

A never ending story: The sparse matrix vector Product (SpMV) on Manycore

“Different kernels optimal for different problem classes”

**CSR**
- small memory footprint
- Needs some logic for row-parallel processing

**ELL**
- zero-padding allows for efficient SIMD execution
- Efficient for balanced matrices

**COO**
- can compensate workload imbalance for irregular patterns

...
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

“Different kernels optimal for different problem classes”

**CSR**
- small memory footprint
- Needs some logic for row-parallel processing

**ELL**
- zero-padding allows for efficient SIMD execution
- Efficient for balanced matrices

**COO**
- can compensate workload imbalance for irregular patterns

...
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- What if we process many different matrices at a time? (Assume they are all small...)

...
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

• What if we process many different matrices at a time? (Assume they are all small...)

• Design a **batched SpMV kernel**.
  • Process a large number of data-independent problems in parallel.
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- *What if we process many different matrices at a time? (Assume they are all small...)*

• Design a **batched SpMV kernel**.
  • *Process a large number of data-independent problems in parallel.*

**Batched, Reproducible, and Reduced Precision BLAS**

**SESSION LEADER:** Piotr Luszczek

**ADDITIONAL SESSION LEADERS:** Jack Dongarra, Cris Cecka, Timothy Costa, Sivasankaran Rajamanickam, Azzam Haidar, Mawussi Zounon

**EVENT TYPE:** Birds of a Feather

**EVENT TAGS:** Ex TP

**TIME:** Tuesday, November 14th, 12:15pm - 1:15pm

**LOCATION:** 402-403-404
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- What if we process many different matrices at a time? (Assume they are all small...)

- Design a **batched SpMV kernel**.
  - Process a large number of data-independent problems in parallel.
  - Are the problems
    - Same Size?
    - Same number of nonzeros overall?
    - Same number of nonzeros in every row?
    - Same sparsity pattern?
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- What if we process many different matrices at a time? (Assume they are all small...)

- Design a **batched SpMV kernel**.
  - **Process a large number of data-independent problems in parallel.**
  - **Are the problems**
    - Same Size?
    - Same number of nonzeros overall?
    - Same number of nonzeros in every row?
    - Same sparsity pattern?

**Let’s be as flexible as possible!**
- Flexible means “everything can be different”
- Every Thread block handles one system
- Memory pointers to distinct systems
- Load input vector $x$ into shared memory
- Kernel for all matrices in CSR, COO, ELL
A never ending story: The sparse matrix vector Product (SpMV) on Manycore

- **What if we process many different matrices at a time? (Assume they are all small...)**

Let's be as flexible as possible!

- Flexible means “everything can be different”
- Every Thread block handles one system
- Memory pointers to distinct systems
- Load input vector $x$ into shared memory
- Kernel for all matrices in CSR, COO, ELL

All matrices stored the same format.
Flexible batched SpMV

First experiment:
• Use different batched SpMV kernels (COO, CSR, ELL ...)
• A batch consisting of the same matrices (*homogeneous batch*)
Flexible batched SpMV

First experiment:
- Use different batched SpMV kernels (COO, CSR, ELL ...)
- A batch consisting of the same matrices (*homogeneous batch*)

Disclaimer: This is an artificial problem setting!
In a real-world scenario, a homogeneous batched SpMV would be handled as SpMM.
Flexible batched SpMV

- **CAGE8**
- **CAN838**
- **DWT_922**
- **ex2**
- **ex27**
- **GR_30_30**
Flexible batched SpMV
Flexible batched SpMV

Second experiment:
- Use different batched SpMV kernels (COO, CSR, ELL …)
- A batch consisting of different matrices \((\text{in-homogeneous batch})\)

1. “somewhat similar” (similar size, nonzero count)
2. completely different
Flexible batched SpMV

Batch of random “similar-sized” problems
\[ n \in [900, 1000] \]
\[ nz \in [3000, 40000] \]
Flexible batched SpMV

Batch of random 
“similar-sized” problems
$n \in [900, 1000]$
$nz \in [3000, 40000]$

Batch of random 
“any-sized” problems.
$n \in [10, 1000]$
$nz \in [100, 40000]$
Flexible batched SpMV on GPUs

- Large number of small SpMV simultaneously
- Matrices can be different in size, nnz, pattern
- COO format most suitable for inhomogeneous batches