Sparse Deep Neural Networks (SpDNNs) provide unique scalability difficulties in which optimizations and advancements can be made [1]. Apache TVM [2] is a machine learning compiler framework for CPUs and GPUs which has shown promising improvements in the optimizations of networks [3]. To evaluate its effectiveness, this work presents GPU optimizations using Apache TVM for SpDNNs.

**Introduction**

- Sparse Deep Neural Networks (SpDNNs) provide unique scalability difficulties in which optimizations and advancements can be made [1].
- Apache TVM [2] is a machine learning compiler framework for CPUs and GPUs which has shown promising improvements in the optimizations of networks [3].
- To evaluate its effectiveness, this work presents GPU optimizations using Apache TVM for SpDNNs.

**Problem Definition**

- Input:
  - weight matrices $W_i$
  - MNIST sparse input data $Y_0$
  - bias vector $B_i$
  - truth categories
- Inference: For each layer, we compute the next using: $Y_{l+1} = \text{ReLU}(Y_l \cdot W_{l} + B_l)$

**Benefits of using SpDNN:**

- pruning DNNs increases sparsity and improves generalization results
- High sparsity (more zeros) results in high potential for more efficient storage and computation.
- compatible with devices with low processing power

**Conclusions**

- Sparse deep neural networks provide unique challenges and opportunities. We pursue this with Apache TVM, a machine learning compiler framework for various computer architectures.
- TVM's scheduling and tuning optimizations improve upon the baseline and shows promise compared to other sparse libraries.
- Further research may be done to apply TVM's autoTVM or AutoScheduler to tune the inference, explore TVM's accommodations for other Python machine learning libraries such as Pytorch and Tensorflow, or learn how to optimize multiplication algorithm for multiplication between two sparse matrices.

**Methodology**

- **TVM TE**: a namespace that TVM's optimizations build off of, is used to write the inference function through to compute and related functions
- Code may be written with PyTorch, TensorFlow, etc. before converting to an IR Module
- **Scheduling**: use TE and TVM Relay, a namespace containing the Intermediate Representation (IR) definition and compiler, to partition each layer into equal sizes and use TVM's built in scheduling functions to parallelize the partitions in CPU and GPU.
- **Low-level IR**: convert the model into a low-level IRModule using Relay. The second code block in figure 2 is the generated IRModule script, which is more
  - Further optimizations on the module may be made during this step
- **Runnable Module**: This is the final compiled module. The input parameters for the module are the input tensors and the output tensors.

**Experimental Setup**

**Component Type**  | **Component**
--- | ---
Server | Runs Rocky Linux 8.6 with hyper-threading enabled
CPU | Intel Xeon Silver 4309Y CPU containing 8 cores
GPU | A40 NVIDIA GPU with 48GB running CUDA Toolkit 11.7

**Results**

<table>
<thead>
<tr>
<th>Layers</th>
<th>Edges</th>
<th>CPU (s)</th>
<th>CPU Edges/Second</th>
<th>GPU (s)</th>
<th>GPU Edges/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVM</td>
<td>MATLAB</td>
<td>Sparse library</td>
<td>120</td>
<td>3,932,160</td>
<td>46.19</td>
</tr>
<tr>
<td>TVM</td>
<td>MATLAB</td>
<td>Sparse library</td>
<td>480</td>
<td>15,728,640</td>
<td>184.07</td>
</tr>
<tr>
<td>TVM</td>
<td>MATLAB</td>
<td>Sparse library</td>
<td>1920</td>
<td>62,914,560</td>
<td>733.46</td>
</tr>
</tbody>
</table>

Table 1. Single CPU and GPU performance for the deep neural network computation with 1024 neurons and varying layer size.

**References**

