# Powering TensorFlow's Sparse-Matrix Handling with the Tensor Algebra Compiler (TACO)

Robert Verkuil rverkuil@mit.edu

May 24, 2017

**Abstract**

*Herein is chronicled an preliminary investigation into integrating the Tensor Algebra Compiler into Google's open source machine-learning library, TensorFlow [1]. To evaluate the goodness of fit between TensorFlow and TACO [2] for such an integration, we have focused our performance comparison on the common operation of Sparse-Matrix Vector multiplication. (SpMV). We have concluded that, provided future work related to TACO's compilation feature proceeds as planned, TACO indeed shows promise as an engine for tensor algebra in TensorFlow. Beyond raw performance, we show that TACO's main strength in as such an engine is the flexibility it gives to users to choose index expressions and sparse formats that work the best for them. We have also concluded that any reasonable, permanent integration to bring TACO into TensorFlow must support the movement of TACO tensors between TensorFlow operations, without repeated format conversion.*

## I. Introduction

We have conducted an investigation into a minimum viable integration of the recently developed Tensor Algebra Compiler [2] (called TACO) into TensorFlow[1], Google's open-source machine learning framework. TensorFlow has become one of the most popular such frameworks in recent years. Like many of its peer frameworks, it allows a user to define a static computation graph in Python. The resulting operations defined by the graph are then optimized and intelligently run in a C++ environment, for performance.

TACO is a recent invention of the Commit-Group in MIT's CSAIL. It is a compiler for tensor algebra that advertises the ability to compile most common tensor algebra expressions into a loop nest that can then be optimized by a traditional compiler. Its flexibility and performance make it very attractive for placement inside of TensorFlow, because TensorFlow is based around the concept of computing many composed, potentially computationally intensive, tensor operations. While TensorFlow currently seems to do a fine job with dense tensor computations, its sparse tensor computation lacks support for many common operations and sparse formats, e.g. Compressed-Sparse Row (CSR). Currently, TensorFlow uses a portion Eigen's [3] Tensor module and only supports Coordinate-format (COO). A compiler approach such as TACO's could help address this deficiency by allowing users the freedom to chose between many formats and simply specify their desired index expressions without hand-write complicated kernels.

## II. Quick Overview of Definitions

- **Sparse matrix** - matrix where most elements are zero
- **Dense matrix** - matrix where most elements are non-zero
- **Tensor** - an n-dimensional array ('superclasses' vectors and matrices)

- **Order** - number of dimensions of a Tensor. Order(vector)==1, Order(Matrix)==2, âĂę
- **Dimensions** - the sizes of the axes of the tensor
- **Sparse format** - a compressed format where nonzero elements are ignored. Ex: CSR, CSC
- **COO (Coordinate Format)** - a compressed format that stores each non-zero entry with its own (coordinate, value) pair
- **CSR (Compressed-Sparse Row)** - similar to COO, but additionally compresses row indices
- **Dense format** - every single element is stored, zero or nonzero
- **Index expression** - a means of expressing a tenor algebra operation by giving a tensor expression and specifying the involved iteration variables. E.g. for SpMV, a(i) = B(i,j)*c(j). TACO uses this syntax for concisely stating what tensor expression is desired to be computed.

## III. Goal

Our main actionable goal for evaluating if TACO could be useful within TensorFlow was demonstrate that TACO could be used to power at least one major sparse tensor operation. To accomplish this, we chose to focus on Sparse-Matrix Dense Vector multiplication (SpMV) as an operation, for its simplicity and commonness in sparse computation.

Additionally, we decided to use TensorFlow custom operations as a means of running TACO code within TensorFlow projects. We chose them because TensorFlow custom operations are well documented, and are the de-facto way for users to augment the framework when composing existing python methods isn't sufficient. Doing so proved to be very clean for our one-off examples and also made it simple to reason about transitions between the Python and TensorFlow C++ runtimes that are used in the framework's normal operation.

```
def tf_matmul_func(sparse_m, v):
    with tf.Session(''):
        return tf.matmul(sparse_m, v).eval()

def my_matmul_func(sparse_m, v):
    with tf.Session(''):
        return mymatmul_module.my_matmul(sparse_m, v).eval()
```

**Figure 1:** *On top - a simple call to run TensorFlow's SpMV code On bottom - code that will call a custom operation to also perform SpMV.)*

## IV. Brief TACO Overview

Taco can be thought of as having four steps for setting up and evaluating an index expression.

- 1) **Setup** - Define formats, Tensors, and Tensor dimensions.
- 2) **Add data** - can use TACO loader or manually insert coordinate/value pairs and then call <tensor>.pack().
- 3) **Define** the index expression.
- 4) **Compile**, **Assemble**, and **Compute**.

```
1  Format dcsc({Sparse,Sparse}, {1,0});
2  Format csf({Sparse,Sparse,Sparse}, {1,0,2});
3  Format dv({Dense}, {0});
4  Tensor<double> A({1024,1024}, dcsc);
5  Tensor<double> B({1024,1024,2048}, csf);
6  Tensor<double> c({2048}, dv);
7
8  B.insert({0,0,0}, 1.0);
9  B.insert({1,2,0}, 2.0);
10 B.insert({1,2,1}, 3.0);
11 c.insert({0}, 4.0);
12 c.insert({1}, 5.0);
13 B.pack();
14 c.pack();
15
16 Var i, j, k(Var::Sum);
17 A(i,j) = B(i,j,k) * c(k);
18
19 A.compile();
20 A.assemble();
21 A.compute();
```

Figure 8: C++ tensor-vector multiplication using `taco`.

**Figure 2:** *Example code of a simple TACO operation.*

## V. System Specs

An Amazon AWS EC2 spot instance of type c3.2xlarge was used to record our benchmarks. It had 8 CPU's running @ 2.7 GHz with 15GB of RAM. TensorFlow 1.1 was used and compiled from source with AVX and SSE4.2 instructions and debug symbols. TACO was built from

its April 27th, 2017 source version. TensorFlow was controlled to not use any parallelism. TACO was also built without any parallelism.

## VI. Getting TACO Running in TensorFlow

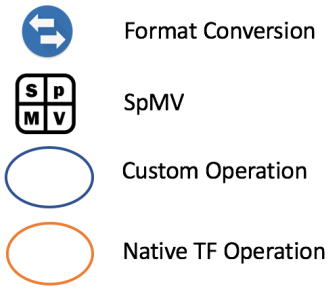
A lengthy amount of time was spent getting TACO code compiling within TensorFlow. After a ramp up period of learning to make libraries and binaries with TensorFlow's recommended build system, Bazel, a suitable method for such a compilation was found. TACO was first separately compiled into a static library and then set as a dependency to future TACO-powered custom operations.

## VII. Road-map of Our Experiments

The next three sections will detail the three major areas of our work. Those areas are:

- 1) Implementing Matrix Vector Multiplication while taking in dense inputs.
- 2) Implementing Matrix Vector Multiplication while taking in sparse inputs.
- 3) Same as above, but separating logic for format conversion and computation.

In each section we take one method from above and compare it to a similar custom operation against a native TensorFlow operation(s). Each section begins with a diagram of the information flow of the two operations. Data flow is pictured as moving from left to right, and the format of the data is specified above each arrows. The diagrams use the following key:




## VIII. Experimenting with Dense Matrix Vector Multiplication

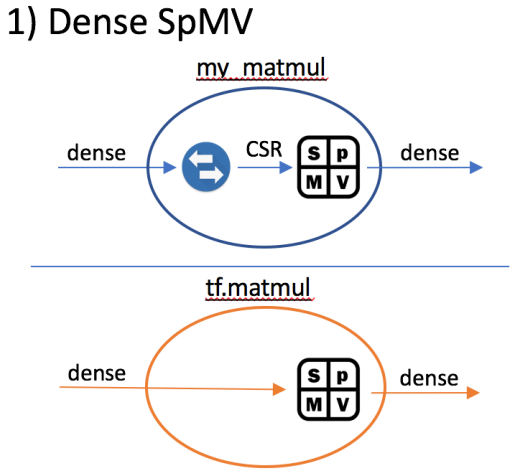



**Figure 3:** *caption*

To keep things simple, the first custom operation made was intended to be a drop in replacement for the TensorFlow dense-matrix dense vector multiplication operation, tf.matmul(). Our results are pictured below.

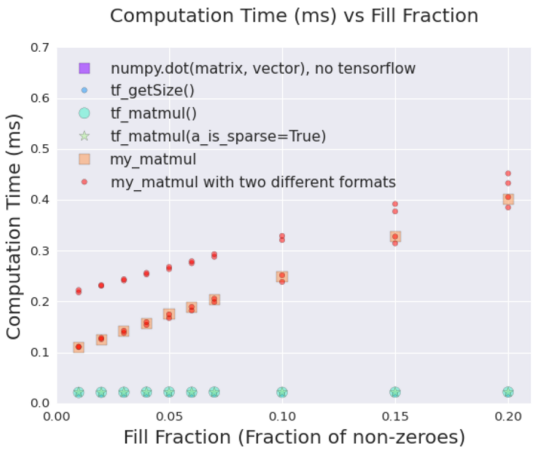



**Figure 4:** *Dense SpMV Comparison*

This graph shows that our custom ops suffered from a tremendous amount of overhead compared with the native TensorFlow dense operation. The reason why can best be understood by looking at a breakdown of the time spent within our custom operation.


**Figure 5:** *Breakdown of steps with my custom operation, my_matmul. Values are milliseconds.*

Each of the labels above corresponds to a major operation in the process of compiling and running an expression in TACO. The top two red arrows, by tacotensor_insert and B_pack show the massive cost of combing through the dense input array and packing the non-zeroes into a complex sparse format (in this case CSR). This makes sense, since tfmatmul() gets its data in exactly the dense format that it requires, but my_matmul must perform a large compression process before it can start any useful work on the computation. TACO creators are planning on making the time spent in the third red arrow, compilation, more amortizable. We will address this more in the final section of this paper, 'Future Work'.
To summarize this experiment:
1) Our custom operation spent the majority of its time in format conversion while the TensorFlow native op didn't need any format conversion. - so it had an unfair advantage for TensorFlow!

2) We are taking in sparse data in a dense representation. This is not so realistic. Why not do a comparison to sparse methods?

## IX. Experimenting with Sparse Matrix Vector Multiplication

In an effort to perform a more apples-to-apples comparison between native and custom TensorFlow matrix vector operations, the next set of operations studied was SpMV.

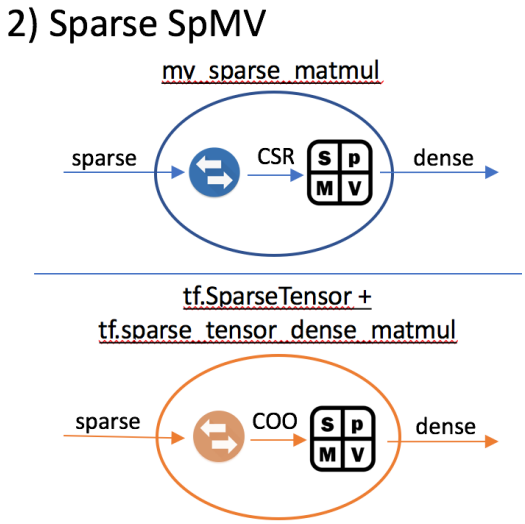



**Figure 6:** *Diagram of custom and TF operations used in sparse comparison.*

TensorFlow tf.sparse_tensor_dense_matmul() is built for this purpose. It takes in a list of indices and values (each pair of which corresponds to exactly one non-zero element), and the shape the tensor would have, were it uncompressed. Since TACO does not currently support coordinate compression (COO), format conversion to a similar format (CSR) was still needed in our custom operation. Our hope was that, due to only passing non-zeros into our operation (roughly 1 to 10% of all values used previously), we would greatly reduce the size of data being moved and also reduce the negative impact of the tacotensor_insert and B_pack stages which plagued our runtime so much, previously.

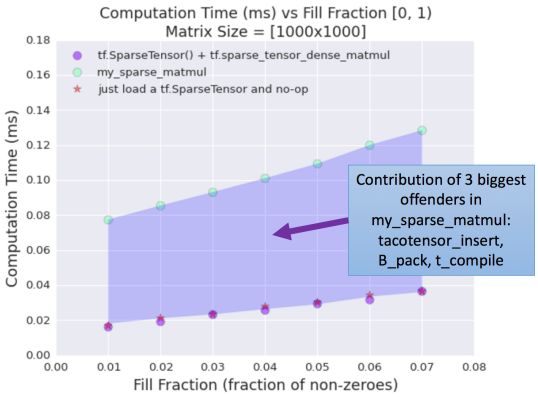
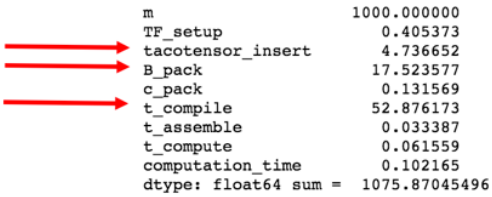



**Figure 7:** *Sparse SpMV Comparison*



```
m                      1000.000000
TF_setup                  0.405373
tacotensor_insert         4.736652
B_pack                   17.523577
c_pack                    0.131569
t_compile                52.876173
t_assemble                0.033387
t_compute                 0.061559
computation_time          0.102165
dtype: float64 sum =   1075.87045496
```

**Figure 8:** *Breakdown of steps with my custom operation, my_sparse_matmul. Values are in milliseconds.*

The above graphic and accompanying breakdown below show us that even when passing sparse format data (no longer passing many many zeros) into our custom operations, the time needed to insert and pack data into the TACO format is too costly. The total time spent in those steps, on average, took over 300 times longer to complete than the actual computation step of the operation! (17.4ms+4.7ms)/0.062ms > 300.

## X. Experimenting with Sparse Matrix Vector Multiplication with Format Conversions Put Into Their Own Operations

Learning from these two earlier experiments, we stayed with SpMV operations, but separated out the code required for format conversion from the code used by TensorFlow and TACO for actual SpMV computation. This resulted in: 1) a TACO pipeline. One operation to produce a taco format. One operation to perform SpMV using that format. 2) a TensorFlow pipeline. One operation to produce a TensorFlow tensor. One operation to perform a SpMV using that format. This is pictured in the diagram below

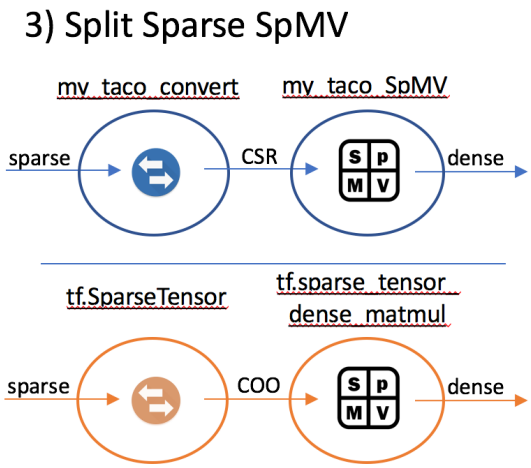



**Figure 9:** *Diagram of custom and TF operations used in sparse comparison with separated out format conversions.*

Doing this allowed us we can compare the format conversions, and SpMV functions separately to see if TACO might hold advantages if given data in it's native format, as TensorFlow operates now.

We find that at high matrix size and low fill rates, TACO can outperform the TensorFlow SpMV method. If, in future work, it is possible to reduce the length of the compilation step via pre-computation or amortization, then the yellow dots at the bottom of the graph become a lower bound of computation time. Achieving a time near this curve would provide an

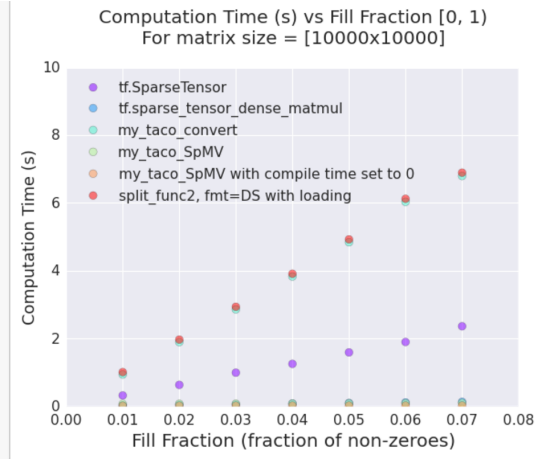



**Figure 10:** *Showing the very high costs of format conversion on 100M doubles with 0-10% fill rate*

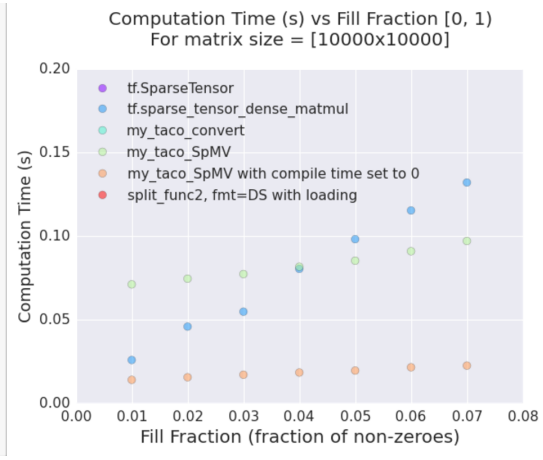



**Figure 11:** *Zooming in on the curves showing just the relative performances of the SpMV portions. At this very high size, TACO really wins out. We believe this is, in large part because TACO stores in CSR, an option that TensorFlow does not have.*

advantage over the TensorFlow method, at all sparsities. they behave roughly according to the model given by their asymptotic size behavior.

The results (Figure 12 and 13) show that TACO is much faster than TensorFlow. We believe that the suitability of CSR vs COO is a large

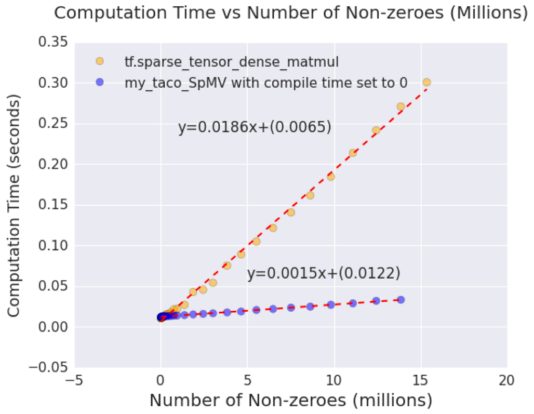



**Figure 12:** *A different look at the relative performance of TACO (without compile) vs TensorFlow. We change the x axis to number of non-zero elements in the matrix. Results are plotted from many sizes in the range [500, 14000] with sparsity in [0.01, 0.07]*

factor (though not the entirety) of the performance difference. Further investigation would be needed to understand the performance gap seen here.

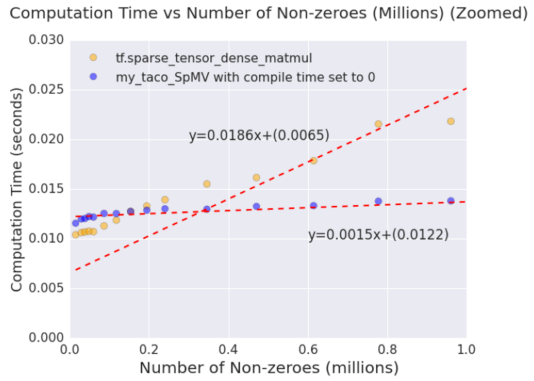



**Figure 13:** *Previous graph, but zoomed in to show the crossover point at <= 0.2 million non-zeroes. TF scales non-linearly through this point, seen in the divergence from the trend line at smaller sizes.*

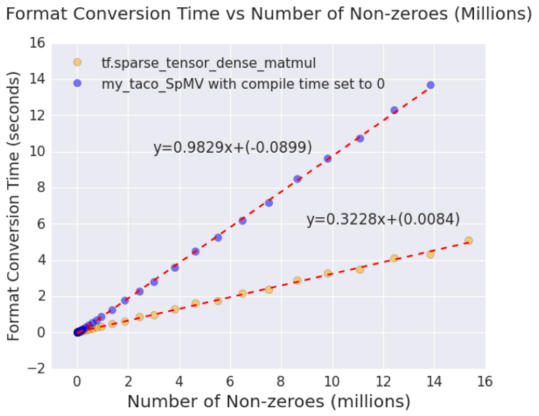



**Figure 14:** *A different look at the relative performance of TACO (without compile) vs TensorFlow. We change the x axis to number of non-zero elements in the matrix. Results are plotted from many sizes in the range [500, 14000] with sparsity in [0.01, 0.07]*

## XI. Conclusion

Our investigation has left us with a good endorsement of TACO. Through our data collection and modeling, we observed TACO giving an advantage over TensorFlow's current sparse handling methods. We believe that this is due to TACO's flexibility in defining formats (and index expressions). In our case, this flexibility allowed us to choose a format that was better suited to the task of SpMV.

Future Work: All the times presented from TACO are lower bounds that make the compilation step free. It is promised that through future features in TACO, pre-compilation and re-use of compilations for the same index expression over different values tensors will allow compile time to be highly amortized over a programs runtime. (provided it performs SpMV, or any other such operation multiple times). Since the compile time dominates so much of the current runtime, it is important that this system be in place, or compile time be sped up by a large factor, in order for the results here to be realized in reality.

It also seems like TensorFlow does not support parallelism within their provided SpMV operation. This is a place where TACO has a large advantage. By moving to the latest taco version and compiling with openMP, the code could reasonably be expected to speed up by a good amount. (due to simple algorithm and data structures).

It would tremendously help users to improve the integration. Right now, the bare minimum for a proof of concept has been implanted. A nicer implementation would give the user a lot of flexibility in the formats and index expressions that they create while building their TensorFlow graphs, without the need for much added code complexity on the part of the user, or many added custom operations for the TACO sparse data types, from a programmer, as was done here.

## References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[2] S. C. D. L. Fredrik Kjolstad, Shoaib Kamil and S. Amarasinghe tech. rep.

[3] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.