A TENSOR BASED FRAMEWORK FOR LARGE SCALE SPATIO-TEMPORAL RASTER DATA PROCESSING

Sukriti Bhattacharya[1]*, Christian Braun[1], Ulrich Leopold[1]

[1] Environmental Informatics Unit
Department for Environmental Research and Innovation (ERIN)
Luxembourg Institute of Science and Technology (LIST)
(sukriti.bhattacharya, christian.braun, ulrich.leopold)@list.lu

Commission IV, WG IV/4

KEY WORDS: Tensor, Tensorflow, Raster, Spatio-temporal simulation, Scalability

ABSTRACT:

In this paper, we address the curse of dimensionality and scalability issues while managing vast volumes of multidimensional raster data in the renewable energy modeling process in an appropriate spatial and temporal context. Tensor representation provides a convenient way to capture inter-dependencies along multiple dimensions. In this direction, we propose a sophisticated way of handling large-scale multi-layered spatio-temporal data, adopted for raster-based geographic information systems (GIS). We chose Tensorflow, an open source software library developed by Google using data flow graphs, and the tensor data structure. We provide a comprehensive performance evaluation of the proposed model against *r.sun* in GRASS GIS. Benchmarking shows that the tensor-based approach outperforms by up to 60%, concerning overall execution time for high-resolution datasets and fine-grained time intervals for daily sums of solar irradiation [Wh.m-2.day-1].

1. INTRODUCTION

Over the decades, the earth science community has been working hard towards a unifying model that addresses and deals with the "curse of dimensionality" [Bellman, 1961]. Multidimensional raster data [Fiume, 1989] can be captured by satellite observations commonly used in earth sciences, precisely in atmospheric, oceanographic (salinity, sea temperature, etc.), meteorological (humidity, wind speed, etc.) and terrestrial (NDVI, land cover, etc.). Multidimensional data can be aggregated, interpolated or simulated from other data sources. So often exhibit different characteristics depending on the scale of the observations. These multidimensional data also provide unique information regarding "where" and "when", which is essential to answer many important questions in geographic information systems (GIS) [Shekhar, Xiong, 2007] studies. Therefore, the community has been looking for a framework that underpins a scalable and extensible approach to drive distributed processing for solving the problem of efficiently managing and disseminating huge volumes of multidimensional spatio-temporal raster data.

We proposed a sophisticated way of handling large-scale multi-dimensional spatio-temporal data, adopted for raster-based GIS. Tensor [Papalexakis et al., 2016] representation provides a convenient way to capture inter-dependencies along multiple dimensions. Tensor's multiway array methods, compatibility with sparse matrices can also help to reduce the cognitive load and cue the attention by highlighting the appropriately reduced dimensionality. Therefore, tensor can be a way to represent the multivariate spatio-temporal data in tensor. We used the multidimensional tensor framework to model such problems and adopted data-flow based computational graphs for efficient execution of the calculation processes. In this approach, spatio-temporal data can be represented as non-overlapping, regular tiles of 2-D raster data, stacked according to the time of data captured. We used Tensorflow, Google's open source library to model problem. In Tensorflow, numerical computations are done with data flow graphs where, nodes represent mathematical operations, while the edges represent the data as tensors. As a case study, we quantified the spatio-temporal dynamics of solar irradiation calculations and 2.5-D shadow calculations for cities at very high space-time resolution using the proposed framework. We described a prototype implementation that brings a high-level space-time scalability integrated over fine temporal granularity, for an entire city.

*Corresponding author

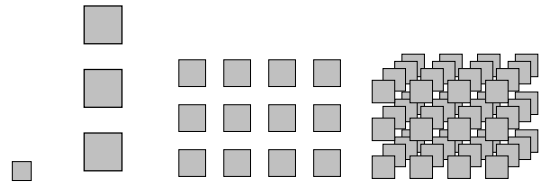
2. BACKGROUND

This section provides a few necessary definitions and describe our notation before we start with the main topic. There are two parts of this discussion. The first part [Papalexakis et al., 2016] deals with the theoretical aspect of tensor and in the second part [Abadi et al., 2015] we will discuss tensor in the context of a programming environment.

Definition 2.1 (Tensor)**.** *Tensors are mathematical objects which represent generalizations of vectors and matrices to potentially higher dimensions, described by the order, a unit of dimensionality; shape, the size of each dimension and a static type assigned to the tensor's elements.*

As shown in Figure 1 notation-wise, scalars are denoted by lower case letters $x \in \mathbb{R}$, vectors by lower case bold letters $\mathbf{x} \in \mathbb{R}^{I_1}$, matrices by upper case bold letters $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$, and a tensor of order N is denoted by upper case bold Euler script letters $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$.

Definition 2.2 (Tensor Indexing)**.** *A tensor slice (subfields), can be extracted by fixing all but two tensor's indices. Where as, Fibers are created when fixing all but one index.*



(a) $x \in \mathbb{R}$ (b) $\mathbf{x} \in \mathbb{R}^3$ (c) $\mathsf{X} \in \mathbb{R}^{3\times4}$ (d) $\mathcal{X} \in \mathbb{R}^{4\times3\times4}$

Figure 1. Scalar(x), Vector($\mathbf{x}$), Matrix(X) and Tensor($\mathcal{X}$)

For the third order tensor shown in Figure1(d) (Figure 2(f)) the slices for each frontal, lateral and horizontal are shown in Figure 2(a), (b) and (c), respectively. And the fibers for each column, row and tube are shown in Figure 2(e), (f) and (g), respectively.

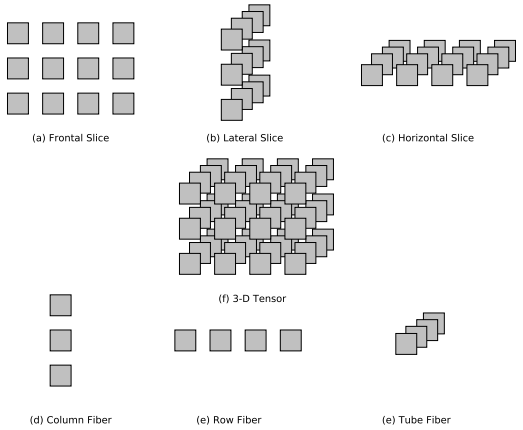


Figure 2. a single slice and a fiber of a $4 \times 3 \times 4$ Tensor

Definition 2.3 (Vectorization)**.** *Vectorization of a given tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ can be denoted as $vec(X)$, where $vec : \mathbb{R}^{I_1\times I_2\times\ldots\times I_N} \rightarrow \mathbb{R}^{I_1.I_2\ldots I_N}$. The outcome is a tall column vector stacking the column fibers as shown in Figure 3 for a tensor $\mathcal{X}$ with two frontal slices $\mathbf{X_1}$ and $\mathbf{X_2}$*

$$\mathbf{X_1} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad \mathbf{X_2} = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$vec(\mathcal{X}) = \begin{bmatrix} a_{00} \\ a_{10} \\ \vdots \\ a_{22} \\ b_{00} \\ \vdots \\ b_{22} \end{bmatrix}$$

Figure 3. Vectorization of a tensor $\mathcal{X} \in \mathbb{R}^{2\times3\times3}$ with 2 frontal slice X_1 and X_2

Definition 2.4 (Reshaping)**.** *The reshape operator for a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ to a size specified by a vector $\boldsymbol{x} = [x_1, x_2, ..., x_M]$ with $\prod_{m=1}^{M} \boldsymbol{x}_m = \prod_{n=1}^{N} I_n$ returns an order-M tensor $\mathcal{Y}$, such that $vec(\mathcal{X}) = vec(\mathcal{Y})$, and is expressed as $\mathcal{Y} = reshape(\mathcal{X}, \mathbf{x}) \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_M}$*

Definition 2.5 (Sparsity)**.** *The sparsity (low-rankness) of a vector/matrix can be rationally measured by the number of nonzero entries and can be quantified with a score, which is the number of zero values in the vector/matrix divided by the total number of elements in the vector/matrix.*

Tensor sparsity is interpreted beyond the low-rank property of all its unfolded subspaces and should more importantly consider how such subspace sparsities are affiliated over the entire tensor structure. Figure 4 shows a sparse tensor the black cells represent 'no-value' fields.

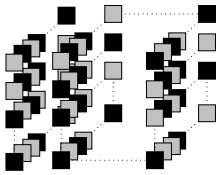


Figure 4. A Sparse Tensor with no value denoted as black cells

2.1 Tensorflow

TensorFlow [Abadi et al., 2015] is an open source software library, developed by Google Brain Team within Google's Machine Learning Intelligence research organization, mainly to conduct machine learning and deep neural network research. However, the system is general enough to be applicable in a wide variety of other domains as well. Moreover, tensorflow's flexible numerical computation core can be used across many other scientific domains. TensorFlow combines the computational algebra of compilation optimization techniques, making accessible the calculation of many mathematical expressions where the problem is substantial and the time required to perform the computation is long.

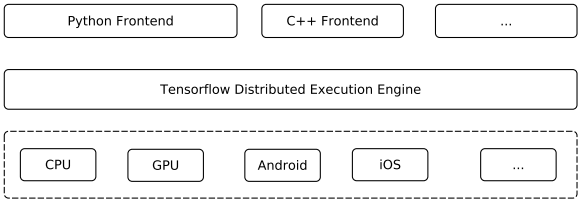


Figure 5. Tensorflow Architecture

Tensorflow defines a general purpose computational graph to achieve ease of expressions and creates tools for running it in different environments. It can execute the operation on various hardware platforms like CPU, GPU, Android, etc. Tensorflow achieves that using Tensorflow Distributed Execution Engine as shown in Figure 6. One can write code in either Python, C++, Java and Go kind of frontend and after that Tensorflow Distributed Execution Engine takes the code and converts it into underlying hardware instruction set.

Tensors are the primary and central data structure that TensorFlow uses to operate on the computational graph. Tensors are declared as variables and feed in as placeholders into the computational graph to perform certain mathematical operations. These computational graphs are directed graphs with no recursion, which allows for computational parallelism. A tensorflow core programs comprised of following steps shown in Figure 6.

3. TRANSFORMING RASTERS TO SPATIO-TEMPORAL TENSOR

Spatial information can be represented in raster data models as a raster map consists of a grid (or matrix) of cells, each one

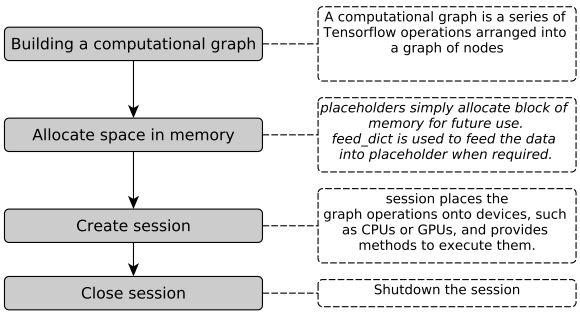


Figure 6. Block diagram a Tensorflow core program

storing a value, represents an area whose size changes depending from the resolution of the map. Cells are arranged in rows and columns where rows represent the x-axis and columns y-axis of a Cartesian plane. Stored values in each cell represents continuous data such as altitude or temperature, or categorical data. consider a set of raster maps of fixed size. To formally describe, a raster R with $I = \{i_1, i_2, ..., i_m\}$ rows and $J = \{j_1, j_2, ..., j_n\}$ columns is a set of fixed locations distributed regularly in space with constant distance between adjacent locations. For every location, we record observations on a fixed set of time stamps, $\mathcal{T} = \{c_1, c_2, ..., c_t\}$, which can again be regularly spaced with equal delays between consecutive measurements as shown in Figure 7.

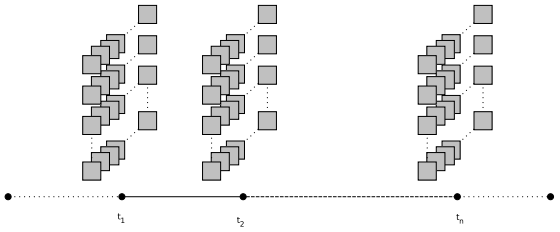


Figure 7. Observation rater maps for equal time intervals

It is the Cartesian product of $R \times T$ that results in the complete spatio-temporal tensor grid $\mathcal{X} \in \mathbb{R}^{(t \times m \times n)}$ as shown in Figure 8, where every cell on the tensor $\mathcal{X}$, (c, i, j), has a distinct measurement.

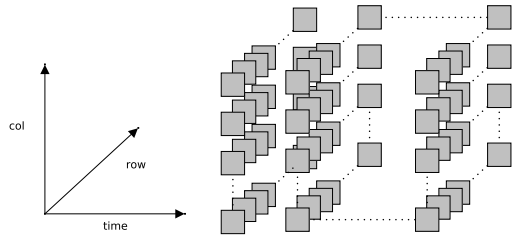


Figure 8. Observation rasters stored in a single 3-D tensor

Cell values can be either positive or negative, integer, or floating point. At the same time, Many of the cell combinations might not make sense or the data for them might be missing. Cells have a `NoData` value to represent the absence of data (Figure 4). For example, building footprint raster dataset or land cover dataset are sparse in nature. Figure 9 shows the sparse representation of the building foot print raster data of Esch-Sur-Alzette, the second largest town in Luxembourg. The white cells represents the `NoData` values which occupies more than 70% of the total cell counts (1828×1874).

We used a dataflow programming model used for TensorFlow [Abadi et al., 2015]. A model is represented by a directed cyclic graph, while the nodes represent operations or primitive functions and the edges represent the data flow between operations. The inputs of a note is represented by the edges incident into the node, and edges incident out of the node represent its outputs. The data on edges can be either immutable tensors (`tf.constant`) or mutable variables (`tf.Variable`, `tf.placeholder`). Both tensors and variables are multi-dimensional arrays of primitive types. A session (`tf.Session`) allows to execute graphs or part of graphs. It allocates resources (on one or more machines) for that and holds the actual values of intermediate results and variables. When all their inputs are ready, the node gets ready to run. The same way, multiple nodes can be executed in parallel when ready. Tensor vectorization (Definition 2.3) can be achieved through `tf.reshape()` and `tf.squeeze()`. Both are cheap in that they operate only on the metadata (i.e. the shape) of the given tensor, and don't modify the data itself. provides `tf.SparseTensor` and `tf.sparse_tensor_to_dense` APIs to perform dense to sparse conversion and vice-versa, respectively and allows us to consider additional dimensions such as time, in order to identify dense regions of interest in the raster more accurately and specifically.

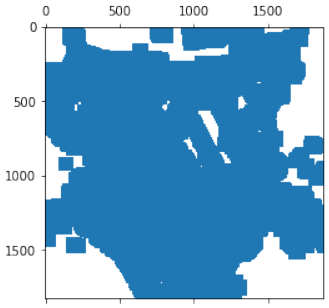


Figure 9. Sparse representation of the raster map

3.1 Motivating Example

We will explain how did we calculate solar angle of incidence and stored the spatio-temporal resultant data in tensor using Tensorflow. The angle of incidence (θ) is the angle between the solar rays and the surface normal shown in Equation. In order to evaluate the angle of incidence we need to specify the orientation of the surface normal. This is done in terms of the surface tilt angle also known as slope (θ_β), and the surface azimuth angle also known as aspect (θ_γ). The position of the sun relative to surface is expressed by the angle of incidence using two additional solar angles solar zenith angle (θ_z) and solar azimuth angle (θ_α) given in Equation 1 and Equation 2, respectively. Where, ϕ denotes the latitude, δ denotes the declination angle and ω denotes the hour angle. The azimuth angle convention used in our model, is defined as degrees east of north (e.g. North = 0, East = 90, West = 270). The solar angle of incidence is a vital parameter for solar irradiation calculation. Both beam and diffuse irradiation depend on the angle of incidence value [**?**].

$$\theta_z = \cos^{-1}(\cos\phi\cos\delta\cos\omega + \sin\phi\sin\delta) \quad (1)$$

$$\theta_\alpha = sign(\omega)\left|\cos^{-1}\left(\frac{\cos\theta_z\sin\phi - \sin\delta}{\sin\theta_z\cos\phi}\right)\right| \quad (2)$$

$$\theta = \cos^{-1}(\cos\theta_\beta \cos\theta_z + \sin\theta_\beta \sin\theta_z \cos(\theta_\alpha - \theta_\gamma)) \quad (3)$$

Next, our objective is to calculate the angle of incidence from sunrise to sunset for a specific date in a year for Esch-sur-Alzette with spatial resolution of 1 meter, 1828×1874 raster maps. Therefore, the resultant data is spatio-temporal in nature and can be stored in a tensor $\mathcal{X} \in \mathbb{R}^{(no\ of\ time\ frames \times 1828 \times 1874)}$. The corresponding code snapshot written in python using Tensorflow APIs is shown in Figure 10. In the above code snapshot,'_tf' suffixed variables are used to represent placeholders where as '_' (underscore) prefixed variables are represented as Python NumPy arrays. As described in Section 2.1 feed_dict feeds the placeholders with values inside a session to create the computation graph. The hour angles, _hrA is the angular displacement of the sun east or west of the local meridian due to rotation of the earth on its axis at 15° per hour from sunrise to sunset for a given time interval _interval; morning negative, afternoon positive. We vectorized the _hrA in the next line to make it a column matrix. Next, we created a tensor lat2D that is a 3-D tensor, where the first dimension represents the dimension of _hrA and the lat two dimension is equal to the sparsed latitude matrix _lat. Next, the solar zenith _zn and amimuth (_azm) angles are calculated for each time frames for the whole city at once. Finally, we calculate the solar angle of incidence for the whole city for each time interval from sunrise to sunset. Finally we restore the raster map converting the sparse matrix to dense using tf.SparseTensor and tf.sparse_tensor_to_dense.

4. TENSOR BASED FRAMEWORK FOR MODELING SOLAR IRRADIATION

The the underlying tensor-based framework is depicted in Figure 11. We compute the beam, diffuse and ground reflected solar irradiation raster maps for a given date, latitude, surface e.g., slope, aspect and atmospheric conditions e.g., albedo, linke turbidity factor and solar parameters e.g., solar constant in python using TensorFlow, Google's open-source library used to simplify mathematical computations (Table 1).

The workflow of the running program that implements the solar irradiation model in TensorFlow can be described in four steps:

1. Build a computational graph: Nodes in the graph represent mathematical operations present in the equations (i.e, 1, 2, 3), while graph edges present the operands i.e, input parameters as tensors. The underlying data-flow graph of the implementation of shadow calculation is shown in Figure 12 using Tensorboard. Tensorboard is the interface included with any standard TensorFlow installation used to visualize the data-flow graph and other tools to understand, debug, and optimize the model.

2. Allocate space in memory: Tensorflow placeholders simply allocate block of memory for future use. As shown in Figure 11, the inputs from Table 1 form a tensor data frame. The feed_dict API is used to feed the data into placeholder when required. By default, placeholders supports unconstrained shape, which allows to feed tensors of different shapes in a session.

```
# variable names are self-explanatory
# hour angle
hrA = tf.multiply (0.261799, tf.subtract
    (time_tf, 12.00))
# vectorising hour angle
hrA = tf.expand_dims (hrA, 1)
# adjusting latitude raster according to the
    time frame
row = tf.size (hrA)
col = tf.size (latitude_tf)
lat2D = tf.reshape (tf.tile (latitude_tf,
    [row]),[row, col])

# solar zenith angle
zn = tf.acos (tf.add (tf.multiply (tf.sin
    (latitude_tf), tf.sin (declinationA_tf)),
    tf.multiply (tf.multiply(tf.cos
    (declinationA_tf), tf.cos (hourA_tf)),
    tf.cos (latitude_tf))))

# solar azimuth angle
azm = tf.multiply (tf.sign(hourA_tf), tf.abs
    (tf.acos ((tf.divide (tf.subtract
    (tf.multiply (tf.cos (zn_tf), tf.sin
    (latitude_tf)), tf.sin (declinationA_tf)),
    tf.multiply (tf.sin (zn_tf), tf.cos
    (latitude_tf)))))))

# solar angle of incidence
aoi = tf.acos (tf.add (tf.multiply (tf.cos
    (zn_tf),tf.cos (slope_tf)),
tf.multiply (tf.subtract (azm_tf, aspect_tf)),
tf.multiply (tf.sin (zn_tf), tf.sin
    (slope_tf)))))

delta = tf.SparseTensor(indices = msk2d, values
    = aoi_tf, dense_shape = [1828, 1874])
aoi3D = tf.sparse_tensor_to_dense(delta)

# computing the dataflow graph
with tf.Session() assess:
sess.run( tf.global_variables_initializer())
   _lat2D = sess.run(lat2D, feed_dict =
       {time_tf: _interval, latitude_tf: _lat})
   _hrA = sess.run(hrA, feed_dict = {time_tf:
       _interval})
   _zn = sess.run(zn_tf, feed_dict = {hourA_tf:
       _hrA, latitude_tf: _lat2D,
       declinationA_tf: _decA})
   _azm = sess.run(azm_tf, feed_dict =
       {hourA_tf: _hrA, latitude_tf: _lat2D,
       zn_tf: _zn, declinationA_tf: _decA})
   _aoi = sess.run(aoi, feed_dict = {slope_tf:
       _slope, aspect_tf: _aspect, azm_tf: _azm,
       zn_tf: _zn})
   _aoi3D = sess.run(aoi3D, feed_dict = {aoi_tf:
       np.degrees(_aoi)})
```

Figure 10. Implementing Solar Angle of Incidence Using TensorFlow

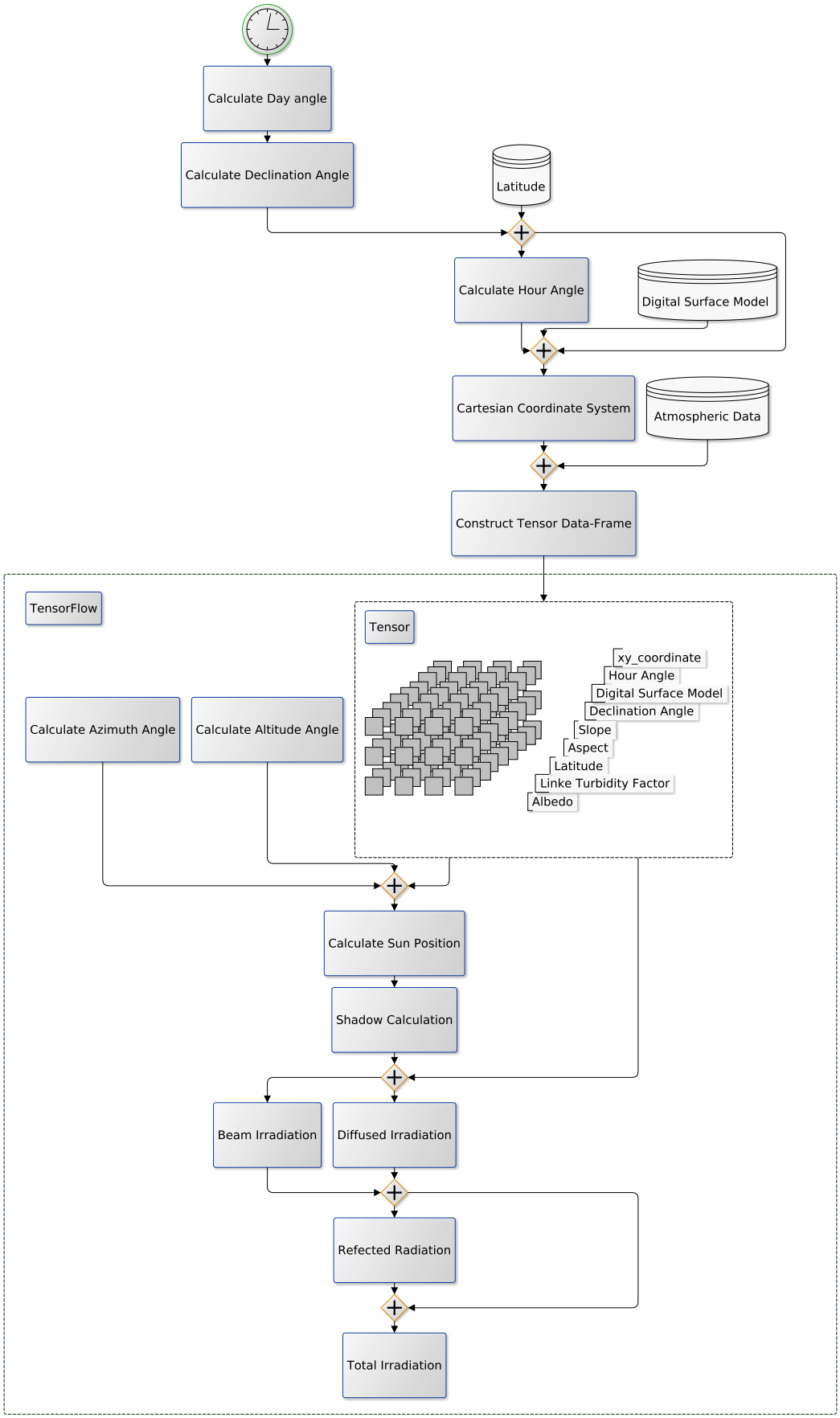


Figure 11. Tensor Based Solar Irradiation Modeling Framework

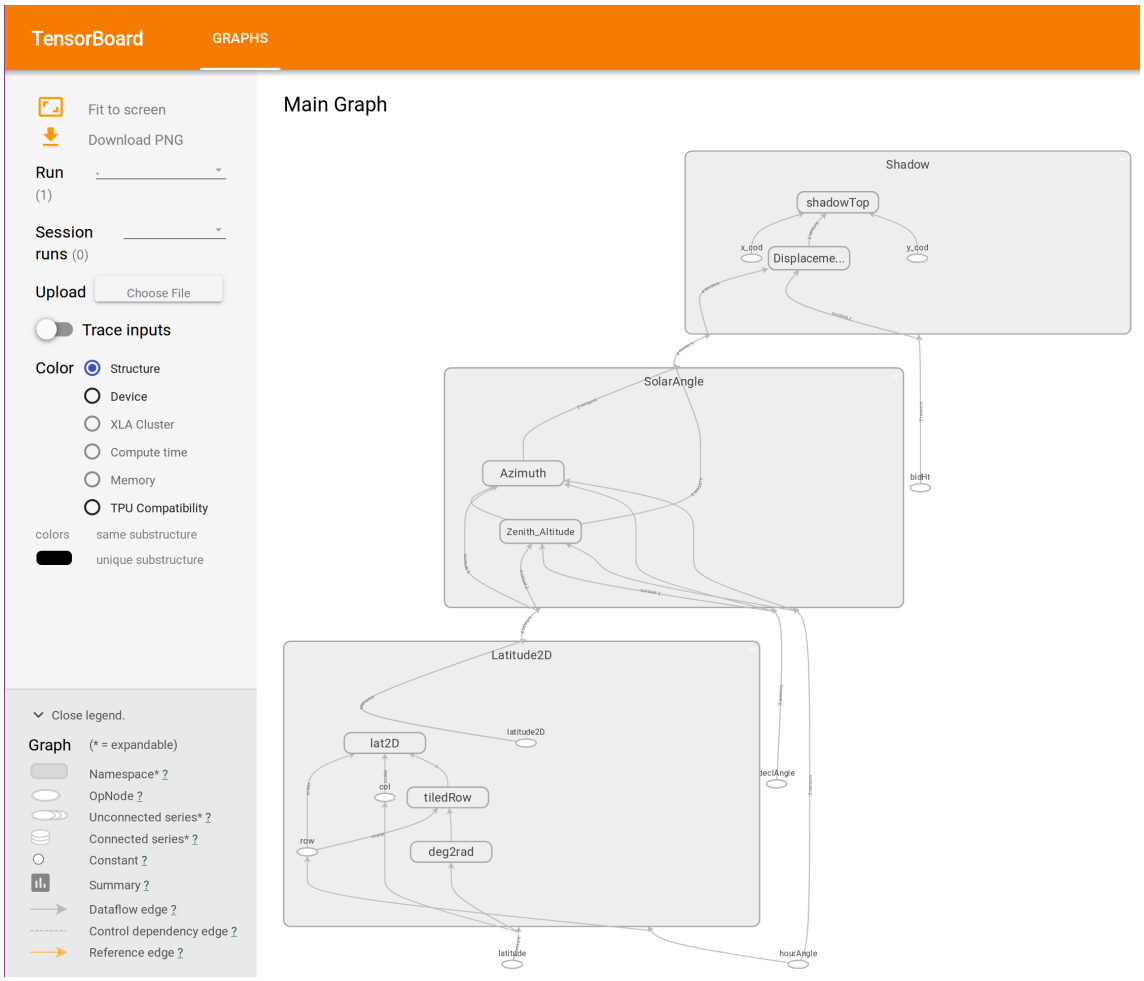


Figure 12. Dataflow Graph Generated by TensorBoard For Shadow Calculation

Table 1. Input/ output of Solar Irradiation model

Input			Output		
Input Name	Type	Unit	Output Name	Type	Unit
elevation	raster	meters	beam irradiance (single day)	raster	$W.m^{-2}$
slope	raster	decimal degrees	beam irradiance (whole day)	raster	$Wh.m^{-2}.day^{-1}$
aspect	raster	decimal degrees	diffuse irradiance (single day)	raster	$W.m^{-2}$
latitude	raster	decimal degrees	diffuse irradiance (whole day)	raster	$Wh.m^{-2}.day^{-1}$
linke turbidity	raster	dimensionless	reflected irradiance (whole day)	raster	$W.m^{-2}.day^{-1}$
albedo	raster	dimensionless	total irradiance (whole day)	raster	$Wh.m^{-2}.day^{-1}$
date	single value	date			
time	single value	decimal hours			
time step	single value	decimal hours			

3. Create session: To compute anything, a graph must be launched in a session and the session places the graph operations onto devices, such as CPUs or GPUs, and provides methods to execute them. These methods return resultant tensors as numpy ndarray objects in Python.

4. Close session: Shutdown the session.

The most fundamental and obvious task for the designer of a parallel programming system ([Cole, 1991]) is the problem decomposition, i.e the identification of parallelism by deciding which part of the problem to be handled implicitly and which to leave to the programmer. Tensorflow offers implicit parallelism and distributed execution. We solve the existing space-time trade-off in solar irradiation calculation by the above 4 steps. Step 1 is designed in such a way that it helps the system to identify operations that can execute in parallel, using explicit edges to represent dependencies between operations. Along with step 2, we used tensor aggregation to reduce the raster maps to concept level in one dimension, discarding the `NoData` values through dimensionality reduction. A sparsity regularization is usually added in order to achieve good compression and aggregation properties on the raster data. In step 3 and 4, we partition program across multiple devices (different CPU cores) to speed-up the overall computation through distributed execution.

We run our model on Esch-sur-Alzette ($49°29'44.988''$N, $5°58'50.016''$E), located 5,505.41 km north of the equator, in

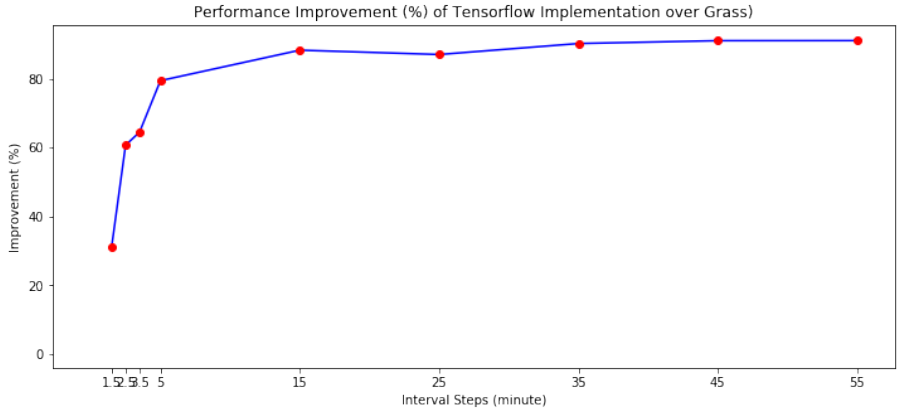


Figure 13. Performance Comparison

the northern hemisphere [1]. It is situated in south-western Luxembourg on the border with France, the second largest town after the country's capital Luxembourg city. The total area of 14.35 km^2 with elevation ranging from 279m to 426m. In this paper, we used a digital surface model of Esch-sur-Alzette. The grid is 1874 columns by1828 rows, with a spatial resolution of 1 meter. The proof of concept run on a 64 bit machine with Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz processor. The performance is quite significant. We compare our model against r.sun ([Jaroslav Hofierka and GRASS Development Team, 2017]), solar irradiance and irradiation model inside GRASS GIS ([GRASS Development Team, 2017]). We provide a comprehensive performance evaluation of our model in Figure 13 against `r.sun`. Based on our evaluation, we can say that our model achieves at least 30%-90% performance improvement with respect to overall execution time for much larger datasets and fine grained time interval for daily sums of solar irradiation [Wh.m-2.day-1] calculation.

5. CONCLUSION

We formulate the "curse of dimensionality" problem in handling multidimentional spatio-tmporal data in raster-based geocomputation into a tensor learning framework which explores different approaches to defining the geospatial grid used to construct the data tensor. Using it, we developed a fast and accurate model with theoretical guarantees for its convergence and validated the correctness and the efficiency of our implementation on real application datasets. Precisely, the main characteristics of the proposed framework include defining, optimizing and efficiently calculating mathematical expressions involving multi-dimensional arrays (tensors); Transparent use of GPU computing such that the same code can be run either on CPUs or GPUs; Implicit parallelism and distributed execution with high scalability offered by data-flow based implementation. Moreover, the Python implementation of the proposed model makes it GRASS GIS 'Add-on' compatible. The tensor-based conceptual framework also enables agile analytics on large scale spatio-temporal datacubes, including simulation, sensor, time-series analysis, and statistical data. Future work concerns deeper analysis on formalizing the multidimensional versions of the primitive operations defined in map algebra; namely, local, focal and zonal operations using the proposed tensor-based framework.

[1] `https://fr.distance.to/Esch-sur-Alzette`

ACKNOWLEDGEMENTS

This work has been funded and supported by the ENOVOS Foundation Luxembourg and the Luxembourg Institute of Science and Technology (LIST) through the SECURE project.

REFERENCES

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

Bellman, R. E., 1961. *Adaptive Control Processes: A Guided Tour*. MIT Press.

Cole, M., 1991. *Algorithmic skeletons: structured management of parallel computation*. MIT Press Cambridge, MA, USA, 3–41.

Fiume, E. L., 1989. *The Mathematical Structure of Raster Graphics*. 1st edn, Academic Press Professional, Inc.

GRASS Development Team, 2017. Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2. Open Source Geospatial Foundation.

Jaroslav Hofierka and GRASS Development Team, 2017. r.sun - Solar irradiance and irradiation model. Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2. Open Source Geospatial Foundation.

Papalexakis, E. E., Faloutsos, C., Sidiropoulos, N. D., 2016. Tensors for Data Mining and Data Fusion: Models, Applications, and Scalable Algorithms. *ACM Trans. Intell. Syst. Technol.*, 8(2), 16:1–16:44. http://doi.acm.org/10.1145/2915921.

Shekhar, S., Xiong, H., 2007. *Encyclopedia of GIS*. 1st edn, Springer Publishing Company, Incorporated.