Rapid Geospatial Processing for Hazard and Risk Management using the Geostack Framework

James Hilton
CSIRO Data61, Melbourne, Australia
james.hilton@csiro.au

Nikhil Garg
CSIRO Data61, Melbourne, Australia
nikhil.garg@csiro.au

ABSTRACT

Operational predictive and risk modelling of landscape-scale hazards such as floods and fires requires rapid processing of geospatial data, fast model execution and efficient data delivery. However, geospatial data sets required for hazard prediction are usually large, in a variety of different formats and usually require a complex pre-processing toolchain. In this paper we present an overview of the Geostack framework, which has been specifically designed for this task using a newly developed software library. The platform aims to provide a unified interface for spatial and temporal data sets, deliver rapid processing through OpenCL and integrate with web APIs or external graphical user interface systems to display and deliver results. We provide examples of hazard and risk use cases, particularly Spark, a Geostack based system for predicting the spread of wildfires. The framework is open-source and freely available to end users and practitioners in the hazard and geospatial space.

Keywords

Hazards, modelling, simulation, data processing

INTRODUCTION

Hazard and risk models are an essential tool in planning, mitigation and operational management of environmental threats and disasters (Newman *et al.*, 2017). These generally take a broad range of forms from use of long-term climate projections for spatial predictions of local impact, to predictive natural hazard models for cyclones, floods, fires or earthquakes (Ward *et al*. 2020). Despite the breadth of domains these models cover there are commonalities between all of these applications. Specifically, these are the requirement for observed or modelled geospatial data as an input into the model, along with various parameters required, the ability to rapidly execute the model based on this input data, and the ability to provide model outputs to a downstream service or as standardized data. The requirement for rapid execution of a model can serve different purposes dependent on the model. For hazard predictions during live events, such as a wildfire or flood, the ability to rapidly understand the behavior of the hazard can guide short timescale operational management strategies such as evacuation (Simpson *et al.*, 2016). For risk predictions (including operational risk predictions) the faster a model can be executed allows both higher spatial fidelity for the model, providing greater detail, as well as allowing more scenarios to be evaluated for a more statistically complete understanding of the risk from a set of uncertain possibilities.

A number of large, well-maintained and extensively utilized open-source libraries and frameworks exist for reading and processing geospatial data. Some prominent examples are GDAL, GeoPandas, Shapely and Pangeo. Typical workflows for hazard and risk data processing usually involve these, or similar, libraries in some form (Zhu *et al.*, 2021). Geostack augments these tools by adding mid-level functionality to simplify geospatial data processing by internally managing local data resources, removing the explicit need for geospatial reprojection and spatial sampling of data. Furthermore, Geostack also maintains a small set of dependency-free data readers and writers for the most common geospatial data types, allowing minimal overhead if only these data types are used. Such libraries also usually do not provide general-purpose processing of data, for example the ability to express a given formula over the data sets, and are usually separate custom or proprietary modules.

One of the main features of Geostack is the ability to express general-purpose functional processing as *scripts* within the system. These scripts are compiled at run-time to either computer processing unit (CPU) code or graphics processing unit (GPU) code which are then run on targeted hardware. This allows data processing to take full advantage of the latest multi-core CPUs or GPU graphics cards, which typically have hundreds to thousands of times the potential computational ability of CPUs. This functionality is provided by OpenCL, which is supported by all major hardware manufactures. All scripts, expressions and solvers are run using OpenCL in Geostack, making the computation scalable over a wide range of hardware. The typical performance improvement for processing in Geostack is dependent on the exact algorithm or functions used, but our trials have found some processing can be improved from tens to hundreds of times faster than native Python implementations. Details and examples of these scripts are given in a later section.

The requirement for rapid execution of models is similarly covered by the use of OpenCL in Geostack. A number of models are provided with the framework, including hydraulic networks, level sets and particle modelling. Here we provide an example of a complete natural hazard workflow for Spark using Geostack data processing and the internal level set solver.

ARCHITECTURE

Geostack operates on three data types:

- Raster data layers, these are gridded two- or three-dimensional data sets with uniform cell spacing and optional information such as the geospatial projection system for the data.
- Vector data layers, these are sets of point, linestrings and polygon geometry elements with associated properties assigned to each element and optional information such as the geospatial projection system for the data.
- Series data, these are one-dimensional data sets composed of ordered pairs of abscissa and ordinate values.

Raster layers

The internal processing workflow for raster layers is shown in Figure 1. The input data here are two raster layers called *A* and *B*. Once named, the raster layers can be referenced in scripts using this given name. In the example show, the script simply specifies the output raster cell value as equal to the value of *A* plus the value of *B*. The language used in scripts is C90 with a wide range of additional mathematical and vector functions provided by OpenCL. Scripts can be as complex as needed and depend on any number of input layers and variables. These are compiled using OpenCL into native vectorized CPU or GPU instructions then run in parallel over all cells of the output raster. Internally each raster is broken into tiles that are loaded in memory as required and cached to local disk once user-specified memory limits are reached.

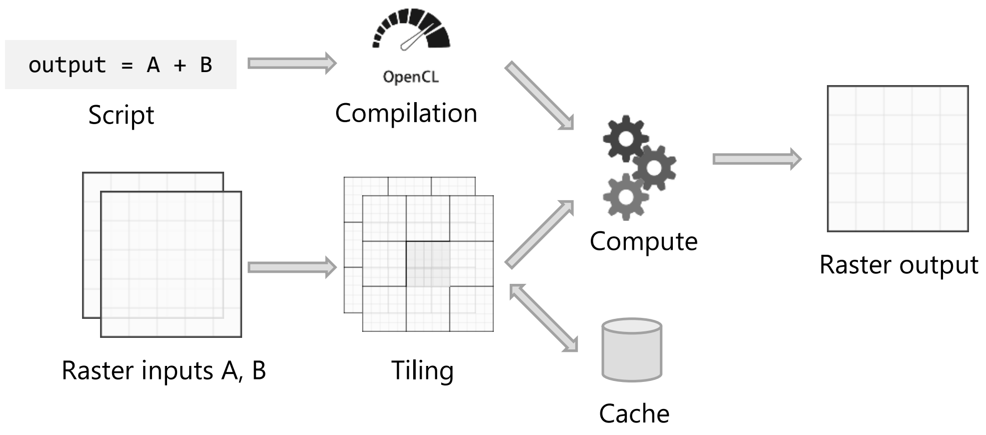
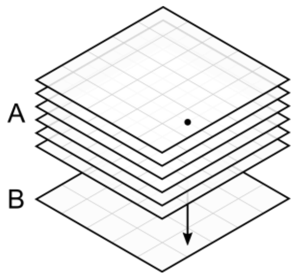

```
output = A + B
```

Figure 1. Internal processing workflow for raster script operations. Scripts are compiled on-the-fly by OpenCL and applied on a per-tile basis to compute a raster output.

The input rasters *A* and *B* can be any resolution or geospatial projection, with re-projection and sampling between the layers transparently handled. There are multiple options for how the output layer is created, although by default it is created to be identical in size, resolution and geospatial projection to the first raster named in the scripting function. Similarly, there are a range of options for the sampling method such as nearest-neighbor or bilinear interpolation. Outputs layers, as in the above example, do not have to be created and rasters can read and write to other rasters during processing, for example, *A* could be set equal to *B* using a script of the form *A* = *B*. One further feature of raster scripting is the ability to define a reduction to be carried out over each of the rasters. This performs operations including *maximum*, *minimum*, *mean*, *standard deviation* and *summation* of all values in the raster into a single value. These operations are carried out using parallel algorithms implemented in OpenCL.

Raster layers in Geostack can be three-dimensional. In this case special handling and functions can be applied over the third dimension to simplify many common operations. Figure 2 shows a 3D raster *A* and a 2D raster *B*. The minimum value can be found by looping over all layers of *A* and updating the minimum value in *B*, as in the inset script, where cell values in the third dimension can be referenced using a square bracket operator. Alternatively, Geostack contains a *sortColumns* function as well as an option to sort every column of *A* (by value in ascending order) before or after a script is run, so limits and quartiles over the column can easily be found. Testing has found that this OpenCL sort operation can be around 10-100 times faster than native sort operations. The third dimension is dependent on the application. For example, in gridded climate data the third dimension can be time and a script could be used to find projected climate extremes within the dataset. For hyperspectral analysis the third dimension may be the frequency band and script operations for frequency analysis could be used find a dominant wavenumber.



```
for (uint k = 0; k < A_layers; k++)
    B = min(B, A[k]);
```

Figure 2. Three-dimensional raster A being reduced over columns into second raster B. This can either be carried out using a script (inset) or a dedicated sort operation.

The capabilities detailed above can be used in many risk and hazard applications. Some examples include:

- Determining quartiles for climate projections from three-dimensional NetCDF data. This involves simply reading the NetCDF file using the inbuilt Geostack raster reading function, sorting the data and writing the various sorted layers.
- Rapid processing of remote sensing optical data, for example, masking, noise reduction or terrain correction. Masking of different land types can use input data layers of different resolution and projection to the remote sensing data, but implemented without needing any reprojection step. Algorithms such as noise reduction or terrain correction can be implemented using a scripting function.
- Calculation of impact cost from a natural hazard. A script implementing loss functions can be implemented based on given spatial maps of impact from the hazard and a spatial cost layer. If needed the reduction over the entire domain can be performed resulting in a single cost value using a *summation* reduction.

The Geostack library natively supports GeoTIFF, NetCDF3, ascii and binary data formats. For other formats the Geostack Python library uses the GDAL, RasterIO or NetCDF libraries as backend readers. The ability to read from a remote server through thredds is also supported. All data reads are localized, where only the tiles requiring processing are read from the underlying data, ensuring IO is minimized.

Vector layers

Vector data layers contain point, linestring or polygon geometry elements, along with associated properties attached to each element. These properties can be numbers, strings or numeric arrays. A range of common geospatial operations are provided for operations on vectors, such as sub- regioning and joining vector data sets. Functionality is also provided to convert and self-intersect vector elements, such as the example shown in Figure 3, where two polygons (left) are processed to provide a set of non-intersecting polygons.

Geostack supports script operations on vectors, where any numeric properties can be read or written to in the script. For example, a set of polygons with properties *p* and *q* could have a new property *r* added, and a script $r = p + q$ applied to populate *r*. As with rasters, these scripts are compiled to CPU or GPU code and run over multiple geometry elements in parallel.

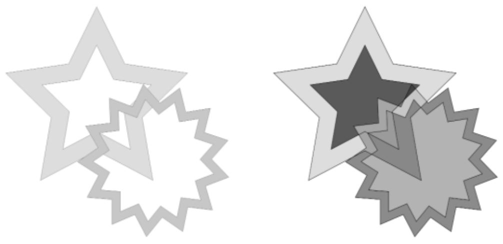


Figure 3. Example of polygon intersection using internal conversion function.

A key feature in Geostack is the ability to mix raster and vector datasets in vector scripts, allowing raster values to be sampled and written to properties in the vector. For example, if a raster *A* is used along with a vector containing a property *r*, the script *r* = *A* writes either a value or reduction of *A* into *r*. The rules for how the raster values of *A* are mapped to *r* are dependent on the data type of the geometry element:

- If the geometry is a point, *r* is the value of the enclosing cell of *A*.
- If the geometry is a linestring, *r* is the reduction over the cells of *A* touching the linestring.
- If the geometry is a polygon, *r* is the reduction over the cells of *A* within the polygon (that is, any cells where the central point of the cell is within the bounds of the polygon).

The reductions are the same as those for the raster scripting, and are passed as an argument to the scripting function. If the raster is three dimensional the reduction is carried out over all layers. If the reduction type is not specified, all data values are returned from each layer as a vector property.

An example use case for this type of scripting is shown in Figure 4, where the raster layer is a multi-dimensional data set containing weekly global temperature predictions for a year, and the vector data set is a set of country polygons, shown as the black outlines.

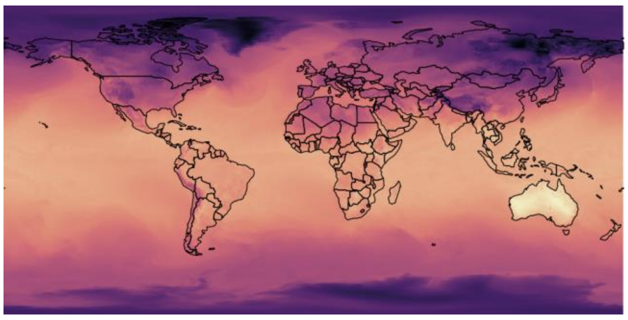


Figure 4. Slice of weekly global temperature prediction (colored layer) overlaid with country polygons. Reductions over each polygon can be carried out using a vector scripting command.

The maximum temperature over the year in each country can be found by loading the raster data for the temperature into a raster called *temp*, the vector data for the countries, adding a new property to hold the maximum temperature, *max_t*, and running a script of the form *max_t* = *temp*. The vector property for maximum temperature will then be populated for each country polygon. Altogether, the script for this in Geostack takes only a few lines of code due to the native interoperability of vector and raster data sets within the framework.

Type conversion

A range of conversion functions from raster data to vectors and vectors to rasters are also included in the framework, although a large number of tools already exist for these processes. However, Geostack provides some features of note for these operations.

To convert a vector to a raster a *rasterise* function is supplied. This takes a user-defined script as a parameter allowing the value written to the raster to be programmatically set from any numeric property. For example, a set of polygons with numeric properties *p* and *q* could be converted to a raster with the sum of *p* and *q* written at each cell within each polygon using the script *output* = *p* + *q*. Overlapping polygons can also be handled using the script and basic cumulative metrics, such as mean values or the count of polygons overlapping the cell, can be written. Another form of vector to raster conversion is a *distance mapping*, where each cell holds the Euclidian distance to the nearest geometry element. Again, an optional script allows users to determine the distance calculation and final output value in the distance map to allow more complex mappings such as spherical basis functions or inverse distance weightings to be implemented.

SPARK

Spark is a wildfire simulator, currently under development as the new national Australian prediction system. It is designed to be used for both operational wildfire simulation and management, with the requirement of much faster-than-real-time response, as well as offline risk mapping of potential fires under various climate and fire mitigation scenarios. Spark is detailed here as it is an example of a full geospatial hazard system developed using Geostack. Furthermore, the development of Spark has guided the development of Geostack at many levels, especially around the requirement for rapid data processing and the need for openness in model development and configuration. The system is built from Geostack components and comprises a set of Python scripts encapsulating the central model and a web API for interfacing with the system.

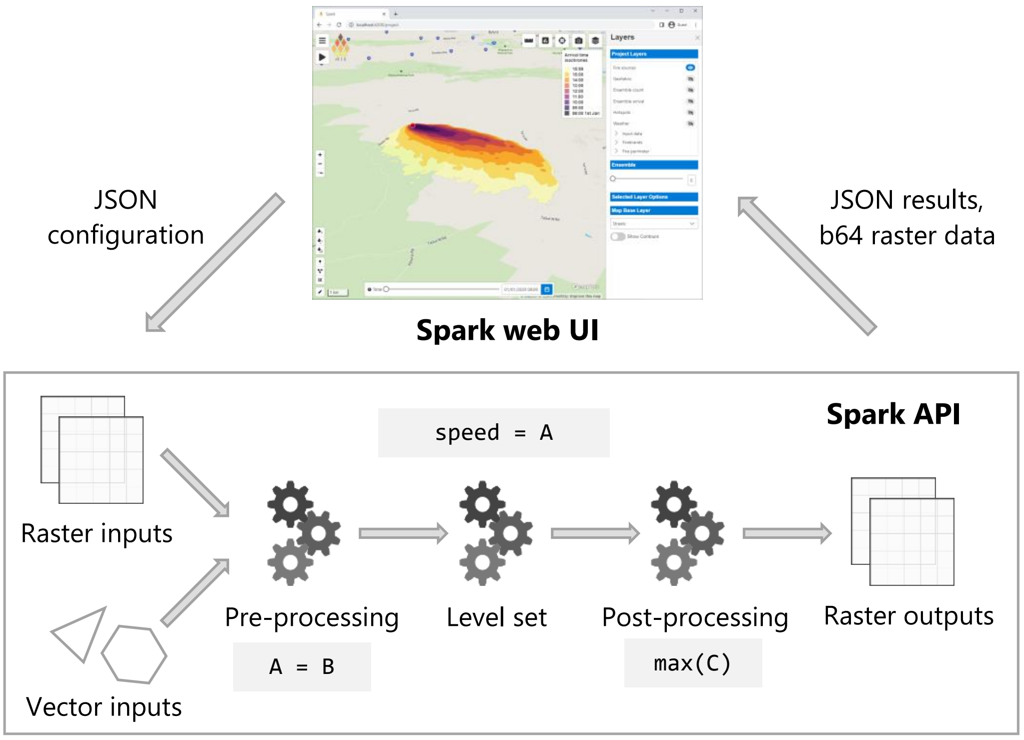


Figure 5.- Architecture of Spark configured for operational wildfire simulation and management. JSON configuration commands are sent to the Spark API, which processes data and models wildfire spread using Geostack. Output data is encoded and send back to the web UI for display.

A highly simplified architecture diagram of Spark is shown in Figure 5. configured for operational use through a web user interface (UI). The lower portion of the figure contains a schematic of the Spark API and Geostack workflow. One of the key differentiators of Spark is the ability to completely configure the simulator through plaintext configuration. All processing and wildfire spread models used for different vegetation types are defined in a JSON configuration file. This makes Spark high flexible in terms of only requiring the configuration JSON to be modified for entirely different regions (with different vegetation types and input data layers) as well as risk analysis (with different output requirements). This configuration is defined for the operational example through forms and inputs in the web UI, which are then encapsulated as a series of calls to the Spark API with JSON payloads. These define both the processing and models, though plaintext scripts, as well as data sources (raster inputs), wildfire sources and firebreaks (vector inputs), variables and time series. The input layers are used in a pre-processing step before being passed to a generic perimeter spread model based on a level set solver (Osher and Sethian, 1988). When the simulation is complete, an optional post-processing step is carried out such as finding maximum wildfire intensities or summing losses over affected areas. Finally, for the operation system, simulation information including the raster outputs are encoded as base64 text and send via the API for display in the web UI.

Basing Spark on Geostack provides several benefits, including:

- The accelerated geospatial data processing described in earlier sections. The transparent reprojection and sampling of spatial and temporal data places no data pre-processing requirements on the end user. In addition, the flexibility of the system allows any raster file formats to be used, and data sources can be swapped easily in the configuration from local files to remote data services.

- The ability to accelerate the computational modelling using OpenCL. The level set model is highly parallelizable and well-suited to OpenCL vectorization. The scripting paradigm described in the earlier sections are also applied to all solvers within the Geostack framework. For the level set solver, the calculation of the outward speed of the fire is defined using a script, allowing a record of the exact model run for each fire to be stored with the configuration for post-wildfire auditing and provenance. This also allows new wildfire spread models to be easily tested and deployed in the framework without the requirement of a full platform update, as the models are defined and encoded outside the Spark system. The Spark and Geostack framework also contain a Lagrangian particle model with a similar scripting system, used for firebrand and spot fires within Spark.
- The tile-based system. This is heavily utilized by the level set model, ensuring the model is run only within tiles spatially bounding the wildfire perimeter. This reduces computational costs and means a maximum domain size does not have to be determined in advance of the wildfire simulation. As previously discussed, the use of tiles also significantly saves data IO, which is a major bottleneck in such systems as IO time usually greatly exceeds the computational calculation times.
- The modular Python-based system, allowing common OS components to be used in the API.

The workflow and framework described above could be applied to many other hazards, and implementing such frameworks is a future focus for our group.

CONCLUSION

Geostack has been developed as an end-to-end geospatial data processing and modelling system. The emphasis in Geostack is on flexibility and ease of use for hazard modelers and data scientists in the geospatial domain, and to provide straightforward access to modern accelerated computer resources (such as graphics processing units and multi-core CPUs) though a transparent framework. All scripts expressed in the system are compiled into highly optimized and vectorized code, without user intervention or configuration, and run directly on these hardware units providing significant acceleration in some cases. The system also includes a growing range of geospatial models such as a perimeter spread level set model and Lagrangian particle model, both used in the Spark framework for wildfire prediction.

The system is currently under development as new use cases and models are explored. The library is available as an open-source project (Geostack, 2022), to which we freely welcome member of geospatial hazards community to use, trial and contribute. The library is also available as a Conda package for Python and has been developed to be interoperable with many common Python libraries including NumPy, GDAL, RasterIO, NetCDF and SpatialLite. We hope our contribution of this library is useful to the hazards community and helps the development of future systems to model, understand and mitigate natural hazards.

REFERENCES

Newman, J. P., Maier, H. R., Riddell, G. A., Zecchin, A. C., Daniell, J. E., Schaefer, A. M., van Delden, H., Khazai, B., O'Flaherty, M. J., and Newland, C. P. (2017) Review of literature on decision support systems for natural hazard risk reduction: Current status and future research directions, *Environmental Modelling & Software*, 96, 378-409.

Osher, S., and Sethian, J.A. (1988) Fronts Propagating with Curvature Dependent speed: Algorithms Based on Hamilton-Jacobi Formulation (1988), *Journal of Computational Physics*, 79, 12-49

Simpson, M., James, R., Hall, Jim W., Borgomeo, E., Ives, M. C., Almeida, S., Kingsborough, A., Economou, T., Stephenson, D., Wagener, T. (2016), Decision Analysis for Management of Natural Hazards, *Annual Review of Environment and Resources*, 41, 489-516

Ward, P. J., Blauhut, V., Bloemendaal, N., Daniell, J. E., de Ruiter, M. C., Duncan, M. J., Emberson, R., Jenkins, S. F., Kirschbaum, D., Kunz, M., Mohr, S., Muis, S., Riddell, G. A. ,Schafer, A., Stanley, T., Veldkamp, T. I. E., and Winsemius, H. C., (2020) Review article: Natural hazard risk assessments at the global scale, *Natural Hazards and Earth System Sciences*, 20, 1069-1096

Zhu, A., Zhao, F., Liang, P., Qin, C. (2021) Next generation of GIS: must be easy, *Annals of GIS*, 27, 71-86

Hilton, J.E. and Garg, N. (2022) Geostack library repository: https://gitlab.com/geostack/library