Scalable Data-Intensive Geocomputation: A Design for Real-Time Continental Flood Inundation Mapping

Yan Y. Liu[1] and Jibonananda Sanyal[1]

Computational Urban Sciences Group
Computational Sciences and Engineering Division
Oak Ridge National Laboratory [†]
Oak Ridge, Tennessee, USA
yanliu@ornl.gov, sanyalj@onrl.gov

Abstract. The convergence of data-intensive and extreme-scale computing behooves an integrated software and data ecosystem for scientific discovery. Developments in this realm will fuel transformative research in data-driven interdisciplinary domains. Geocomputation provides computing paradigms in Geographic Information Systems (GIS) for interactive computing of geographic data, processes, models, and maps. Because GIS is data-driven, the computational scalability of a geocomputation workflow is directly related to the scale of the GIS data layers, their resolution and extent, as well as the velocity of the geo-located data streams to be processed. Geocomputation applications, which have high user interactivity and low end-to-end latency requirements, will dramatically benefit from the convergence of high-end data analytics (HDA) and high-performance computing (HPC). In an application, we must identify and eliminate computational bottlenecks that arise in a geocomputation workflow. Indeed, poor scalability at any of the workflow components is detrimental to the entire end-to-end pipeline. Here, we study a large geocomputation use case in flood inundation mapping that handles multiple national-scale geospatial datasets and targets low end-to-end latency. We discuss the benefits and challenges for harnessing both HDA and HPC for data-intensive geospatial data processing and intensive numerical modeling of geographic processes. We propose an HDA+HPC geocomputation architecture design that couples HDA (e.g., Spark)-based spatial data handling and HPC-based parallel data modeling. Key techniques for coupling HDA and HPC to bridge the two different software stacks are reviewed and discussed.

[†]This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Keywords: geocomputation, data science, high-performance computing

1 Introduction

High-end Data Analytics (HDA) [2] have introduced new infrastructure and tools for data analytics that are now widely adopted in the science community as enabling technologies for rapidly emerging data-intensive science [11]. The convergence of HDA and simulation-oriented high-performance computing (HPC) presents tremendous opportunities for scientific advancement in computing applications and workflows by orchestrating simulations, experiments, data, and learning-based knowledge. However, since HDA and HPC present separate software ecosystems [2], fusing HDA and HPC, at both the application and infrastructure levels, requires the dismantling of the boundaries of computing- and data-intensive paradigms so that an integrated software and data ecosystem can be built.

In Geographic Information Systems (GIS) environments [10], geocomputation [5] provides computing paradigms for interactive computing of geographic data, processes, models, and maps. Geocomputation is data-centric. The computational scalability of a geocomputation workflow is directly related to the scale of the GIS data layers, their resolution and extent, and the velocity of the geolocated data streams to be processed. Scalable geocomputation solutions have evolved from desktop computing to distributed computing paradigms that harness service computing, HDA, or HPC. Because geocomputation is unique in high user interactivity and low end-to-end latency requirements, performance will dramatically improve with the convergence of HDA and HPC. The application level challenge, however, is to identify and eliminate computational bottlenecks that arise along the entire geocomputation workflow. Indeed, poor scalability at any of the workflow components is detrimental to the entire end-to-end pipeline. Similar challenges have arisen in scalable database research [6].

Here, we study the convergence of HDA and HPC in geocomputation by examining a typical large-scale geospatial application—continental flood inundation mapping. We analyze the bottlenecks that arise when scaling the geocomputation workflow from the regional level to the continental level.

2 A Geocomputation Use Case

The continental flood inundation mapping (CFIM) framework is an HPC framework [18] that provides continental-level hydrologic analysis. At the national level, the input datasets include the Digital Elevation Model (DEM) produced by U.S. Geological Survey (USGS) 3DEP (the 3-D Elevation Program), the NHDPlus hydrography dataset produced by USGS and the U.S. Environmental Protection Agency (EPA), and real-time water forecasts from the National Water Model (NWM) at the National Oceanic and Atmospheric Administration (NOAA). With these data, a hydrologic terrain raster, Height Above Nearest

Drainage (HAND) (Fig. 1), and HAND-derived flood inundation measures are computed for 331 river basins in the conterminous U.S. (CONUS) [37]).

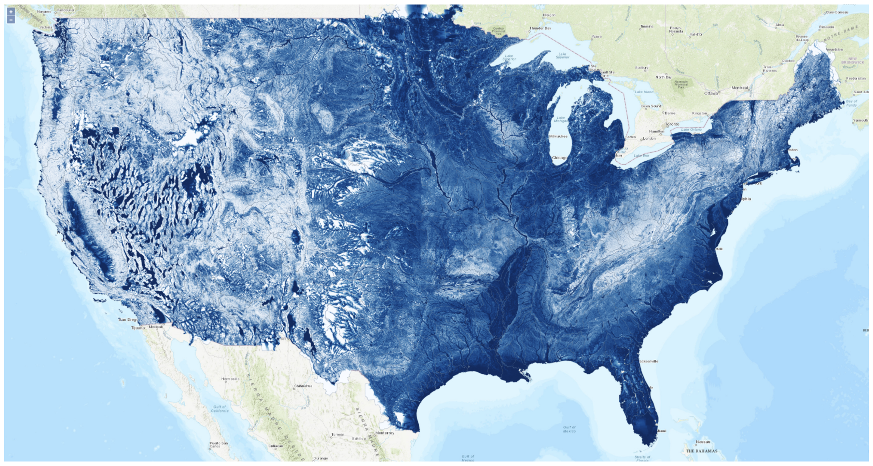


Fig. 1. The HAND map at 10m resolution for CONUS, version 0.2.0, produced on March 01, 2020. Deep blue areas are prone to flood inundation.

The CFIM computation features data-intensive vector and raster operations for water feature querying, clipping, and reprojection, as well as data- and computing-intensive hydrologic analysis of terrain pits, flow direction and accumulation, and stream networks. The entire workflow consists of 20 steps. The input DEM for CONUS at 10m resolution is a 718GB raster grid of 180 billion cells. When 1m DEM becomes available, the size of the raster grid will increase 100 fold. The vector input has 2.7 million polygons (watershed boundaries) and lines (flow lines). A higher resolution version would have 30 million vectors. In addition, the NWM water forecast data consists of an hourly data stream for the subsequent 18 hours. The hydrologic analyses are parallelized using MPI [18]. In addition, in-situ analytics of HAND and flood inundation, such as flood depth maps, need to be delivered to web browsers and mobile apps. For instance, generating a HAND-sized map layer involves computing 230k map tiles for 8 zoom levels or millions of contour vectors.

Currently, the CFIM HPC workflow is deployed on the Condo cluster at the Compute and Data Environment for Science (CADES) at the Oak Ridge National Laboratory (ORNL). The entire output at 10m resolution, including HAND and derived raster and vector products, is about 3.7TB. Each version of the dataset [19, 20] is registered and published on the Scalable Data Infrastructure for Science (SDIS) [33], housed at the Oak Ridge Leadership Computing Facility (OLCF). Community access is provided via HTTP download as well as the Globus bulk transfer service.

3 Data and Computing Challenges

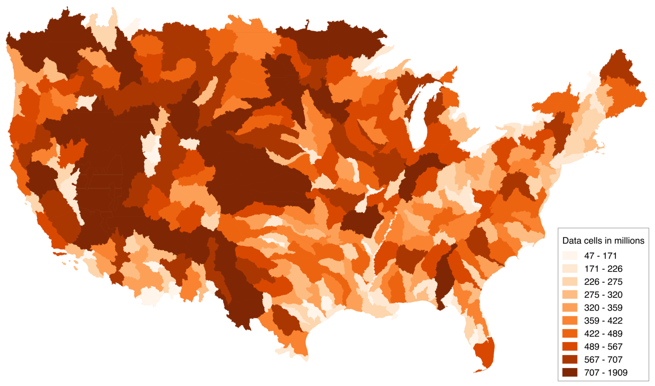


Fig. 2. The computing intensity map for the 331 HUC6 units on CONUS, derived from the number of data cells that are involved in actual computing. This map guides the allocation of HPC resources for the parallel computing of all the HUC6 units.

In the CFIM HPC workflow, a two-level parallelization strategy is applied to systematically scale HAND computation to finer DEM resolutions and flowline scale. The first level of the parallelization strategy spatially decomposes CONUS into contiguous hydrologic units that follow the hierarchical Hydrologic Unit Code (HUC) system. The delineation of the HUC boundary by hydrologists minimizes interference between neighboring HUC units and creates a batch of high-throughput computing jobs at each HUC level, as shown in Fig. 2. For each job, a second-level parallelization via MPI is applied to a series of hydrologic analysis functions that operates on the entire raster grid of each HUC unit.

The most recent HAND data was computed on the CADES Condo cluster using RAM disk and burst buffer. On average, computing an HUC6 unit took half an hour (with a standard deviation of 1,210 seconds). The MPI parallelization effectively accelerated the performance of several key hydrologic analysis functions so that they were no longer bottlenecks. However, new bottlenecks arose in the serial GIS operations, particularly those that clip DEM and HAND rasters. These two raster clipping functions required 454 seconds, on average, with a standard deviation of 407 seconds, which amounted to 25% of the entire computing time.

This geocomputation scenario presents an interesting but challenging case for further acceleration. The workflow does not read and write large geospatial datasets only once, but applies frequent GIS and hydrologic analysis operations on them, generating copious intermediate data at runtime. Furthermore, in order to enable first responders in extreme weather events (e.g., hurricanes), the computing time of HAND and inundation forecast information must be further reduced. For example, to match the pace of real-time water forecasting, the computing time for inundation forecasts must be reduced from hours to minutes. In this quest, HPC alone may not be sufficiently effective for two reasons.

1. First, GIS libraries are often built as a geospatial extension to the common data manipulation, query, and processing capabilities of general database and data management systems. In the literature, HPC-based GIS development are individual efforts. A more systematic approach that manages the complicated interconnectedness of the individual components is needed. Given this base, it is then a daunting task to develop a full-scale reconceptualization of the entire stack of data handling libraries and GIS extensions using HPC.
2. Second, GIS functions often operate on multiple layers of vector and raster data with different resolutions and spatial extent. Accordingly, computing a GIS function requires frequent and dynamic data operations at multiple levels of data granularity. This requirement is overly taxing on the distributed data management model of HPC.

Data-intensive computing software infrastructure, such as Spark [36], provide a desirable solution to the challenges that we have identified, provided that it can be systematically integrated into an HPC workflow. As a scalable data analytic software infrastructure, Spark provides a rich set of data handling features with distributed processing capabilities. Since most of the GIS operations in the CFIM workflow are commutative and associative, it is possible to rewrite them using the *mapreduce* paradigm. With the additional functional programming support through Spark, the dependencies between the steps in the workflow can be represented implicitly in the code. For example, at the infrastructure level, Spark provides distributed data management and associated data parallelism (through the Resilient Distributed Dataset (RDD) abstraction). Spark jobs are executed as Directed Acyclic Graphs (DAG) that optimize the execution plan on managed resources. Accordingly, it is not necessary to consider the task scheduling problem at the infrastructure level. Furthermore, the lazy execution feature in Spark allows a more performant workflow execution by reducing the intermediate data footprints. Spark has been used for vector processing in geocomputation [14, 13, 32, 35, 34, 4]. Spark connectors to high-dimensional arrays have been developed [16, 30]. Spark has also been utilized for large raster-based deep learning inference [21].

4 Data-Driven Geocomputation on HDA+HPC

To harness HDA in a geocomputation workflow, we propose a general HDA+HPC fusion model for geocomputation, shown in Fig. 3, that defines a fusion software architecture to meet the end-to-end performance requirements in data-intensive geospatial computing, such as those in CFIM. To effectively integrate HDA and

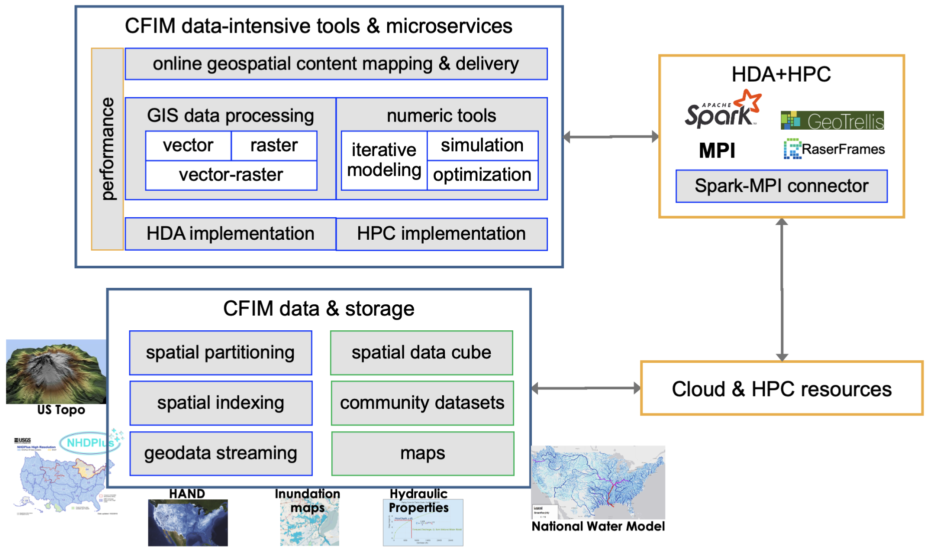


Fig. 3. A geocomputation design for CFIM on integrated HDA and HPC.

HPC, an application needs to manage software and data that enables flexible construction of computing elements in both HPC in a "move-data-to-code" fashion and HDA in which code is packaged and moved to data nodes for computing (i.e., "move-code-to-data").

In this data-driven model, hydrology and GIS data are imported into a distributed data storage system, which are then spatially partitioned using regular or adaptive 2D domain decomposition mechanism (e.g., adaptive quadtree) into data blocks (e.g., partitioned RDDs in Spark) that are distributed on multiple data nodes or a parallel file system. On a parallel file system, the data parallelism is provided by the parallel IO capability of the system. A spatial index, using the space filling curve, is built to link these partitions. This spatial index then accelerates the spatial selection of the data blocks that participate in the actual computing. The geodata streaming module handles data streams such as the hourly water forecast as well as any data version update by using the spatial data cube as runtime storage. The cube runs in a smaller sized distributed file system (e.g., HDFS) or an in-memory database (e.g., Redis) on RAM disk or burst buffer. The cube also serves the purpose of caching frequently accessed datasets and maps from community users and applications. The use of the cube in the online geospatial content mapping and delivery module provides important and necessary performance for real-time GIS scenarios such as the CFIM application.

Storing and indexing large geospatial datasets into data blocks provides basic computing elements for both HDA and HPC algorithms. Because the data blocks are loaded and processed on multiple computing nodes, we are able to execute GIS operations that do not fit into the memory of a single machine.

Spatial knowledge on distributed data blocks can also be effectively leveraged to make runtime scheduling decisions that affect resource management and walltime budget. In the context of Spark, we can use spatial characteristics to efficiently configure CPU, memory, and storage resources for a given data processing job.

Transforming sequential or HPC-based GIS functions into data-parallel algorithms is key to enabling geocomputation on data-intensive computing platforms. This transformation requires algorithmic innovations for existing GIS implementations. To scale GIS data processing in CFIM, a set of Spark-based HDA functions for vector, raster, and vector-raster operations needs to be integrated. For example, individual functions can be directly incorporated from open source Spark-based geospatial data processing tools, such as GeoTrellis [14] and RasterFrames [3]. However, GIS operations that can efficiently implement the two-level parallelization in a single pass must be developed. In CFIM, it is possible to develop an efficient clipping operation on the entire CONUS DEM using the boundary polygons of all the HUC units. These boundary polygons can be checked against each data block's spatial extent to determine which ones intersect with this HUC unit. Multiple clipped raster segments at each data block can then be returned as a key-value list, where the key is the HUC id. They can then be grouped at the reshuffling stage into each HUC's boundary. In this way, the clipping operation for all HUC units can be computed as a Spark job. In CFIM, 70% of the operations can be converted to HDA operations. The chaining of them in the workflow logic is captured in Spark as task DAG. This composition and lazy execution of the DAG in Spark significantly improves a geocomputational workflow in two ways. First, the functional programming pattern in Spark provides a way to dynamically package and send the workflow code to data nodes. Second, the delayed execution of all *transformation* operations on the chain eliminates the need for storing intermediate results at each workflow step. In HPC-based workflow solutions, these intermediate results are usually written to and read from disk for large datasets.

We must also consider that HDA and its *mapreduce* programming paradigm may not be well-suited for iterative processes that involve intensive numerical operations, such as those in iterative modeling, simulation, and optimization. For instance, the pit filling algorithm in CFIM operates on a large elevation raster by flooding the entire terrain first and then iteratively letting water recede until all the pits are filled. Such MPI parallelization does not need to be converted to an HDA function unless a data-driven parallel algorithm is more efficient.

Note that the two-level parallelization strategy employed in the CFIM HPC workflow is still effective in HDA+HPC. For an operation to be applied to all of the HUC units, it can simply be invoked as an independent Spark job. Each job's DAG can then be executed in parallel, managed by Spark. With sufficient resource allocation, the asynchronous *actions* in Spark with the FAIR scheduler setting can be leveraged to process multiple HUC units simultaneously.

At the second level of parallelization where we run the workflow on an HUC unit, we must determine how to invoke an HPC step in the Spark context. In the literature, we evaluate three Spark–MPI connector solutions: Alchemist [8],

Table 1. Comparison of three Spark–MPI connectors.

	Alchemist	Spark+MPI	Spark-MPI
MPI coupling	Spark workers contact the Alchemist server, which launches MPI	Spark code runs *mpiexec* as a separate process	Dynamic PMI or MCA OpenRTE launch in Spark
Spark to MPI connection	TCP/IP socket on the Alchemist server	Command line invocation in Python	*mpi4py*
Data exchange	Matrix RDD to/from the *Elemental* format	Node-based HDFS partitions on RAM disk or burst buffer	Python *Pickle* for objects; single-segment buffer for contiguous arrays
Data transfer protocols	TCP/IP between Spark nodes and Alchemist nodes	Distributed file IO	Direct memory access/copy with *mpi4py*

Spark+MPI [1] (also known as Spark MPI Adapter), and Spark–MPI [23, 24]. There are two basic requirements to interoperate the two separate software stacks of Spark and MPI. First, we need to launch an MPI executable in the Spark JVM. Second, we must define a message and data exchange protocol between them. Table 1 compares the working mechanisms of these solutions. Alchemist is a broker solution that spawns a set of Alchemist server and worker processes to bridge the communication and data exchange between Spark and MPI. Since it uses a matrix format as a data exchange format between Spark's RDD and MPI's data structure, serializing geospatial raster data in Alchemist is desirable. The data exchange process in Spark–MPI can be efficient here since there is no memory copy when contiguous arrays (such as a raster) are passed from Spark to MPI through *mpi4py*. However, Spark–MPI leverages dynamic process management features in specific MPI implementations. Portability is a potential issue. Spark+MPI uses a file system as a data exchange media, which poses limitations on the IO cost for frequent data exchange.

In general, the proposed geocomputation design captures three aspects of HDA and HPC integration and interoperability. First, the aforementioned Spark–MPI connection is an example of how to launch HPC code in HDA, which is important for compute-intensive functions. Horovod in Spark [12] is another example of machine learning computation using MPI within an HDA context. Second, HDA empowers data gateway functionalities that face end users. In geocomputation, HDA may accelerate geovisualization, spatial analytics, and spatial data and map query, but the computing power on an online gateway may be limited. When large-scale analytics, optimization, and simulation are involved, middleware solutions are needed to launch HDA on HPC. For Spark, Spark connectors are needed to conduct in-situ processing. This can be done in two ways. First, a middleware (e.g., DASK) with application programming interface (API) sends a Spark application to HPC batch schedulers, which then instantiate a dynamic Spark cluster that application drivers connect to. Results are sent back to the gateway using the middleware. Alternatively, a Spark connector can be built in an in-situ service such as ADIOS2 [15, 17]). This connector is responsible for connecting a Spark application on gateway to a Spark cluster in HPC and handling the data transfer between them.

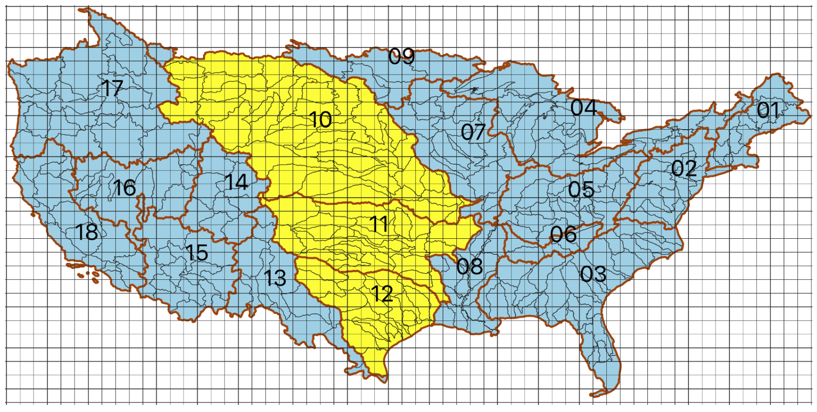
At the infrastructure level, the design is based on the assumption that an HPC environment allows the sharing of the data repository between the gateway cloud instance and the backend HPC resources. Otherwise, data transfer cost must be considered. A hybrid infrastructure that supports data center and HPC operations would be desirable for our geocomputation use case.

5 Preliminary Results

GIS operations create heterogeneous computing and data load as a result of graphic and geometric calculations between shapes and geospatial data contained in shapes. As a GIS operation is transformed into data-parallel implementation, it is essential to understand the associated computational performance variants in order to systematically develop algorithmic strategies for data-parallel geocomputation. We conducted computational experiments on a representative vector-raster operation to measure the computational scalability and load balance of a Spark cluster for handling large raster data.

Clipping or subsetting is a common GIS operation for extracting a subset of raster within the boundary of a polygon vector. A serial implementation often creates a rectangle bounding box of the polygon as a clipping window. The polygon is then rasterized to mask the subset of raster cells within the window. The clipped raster is then output with the same spatial extent as the window. The data parallelization of this operation consists of three distributed functions. The *tiling()* function decomposes an input raster into tiles of the same dimensions and registers all the tiles as a binary RDD indexed by their bounding box rectangle. The *clipping()* function applies the clipping on an input tile and supports multi-polygon clipping. The output of the clipping function on a tile is a set of subsets on the tile, indexed by shape id. The clipped tiles of the same shape id are then aggregated into a single raster, which is output by the *save()* function. In this implementation, the tiling and clipping are *map* functions that can be chained for lazy execution. A *groupByKey()* call in Spark shuffles the clipped tiles using unique shape identifiers. Data shuffling is memory- and IO-intensive. The *save()* function is a parallel IO operation for saving multiple clipped rasters simultaneously, each of which is named after their shape id. The clipping algorithm is written using PySpark, the Python library of Spark.

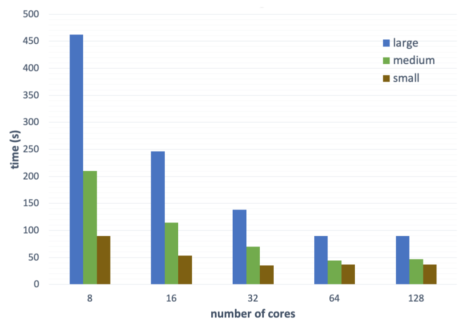
Three test rasters of different sizes, *large*, *medium*, and *small*, are generated from the national elevation dataset produced by the U.S. Geological Survey, as shown in Fig. 4. The default tile size is 10812×10812. A Spark job takes an input of all HUC6 unit shapes, whose boundary is colored in black in Fig. 4, in a test raster and outputs a clipped raster for each of them. A Spark cluster



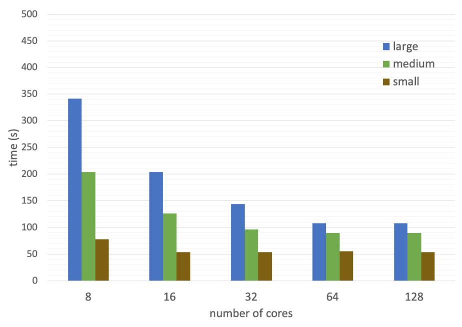
Dataset	*HUC2 Regions*	*1˚x1˚ Tiles*	*Size*	*HUC6 Units*
Large	10, 11, 12	303	141.7GB	90
Medium	11, 12	142	66.4GB	47
Small	12	62	29GB	22

Fig. 4. Study areas and raster data characteristics.

of 8 virtual working nodes is configured on a cloud instance with 128 physical cores (AMD EPYC 7702 2GHz), 1TB memory, and 512GB disk in the private cloud at the Compute and Data Environment for Science (CADES) at ORNL. The 1TB memory is split into 512GB RAMDISK as Spark worker disk cache and 512GB for the 8 Spark worker nodes. On each node, one Spark executor is launched with 32GB memory. The number of cores per executor is specified as a runtime parameter. A Jupyter Spark driver connects to the Spark cluster to run each test job.



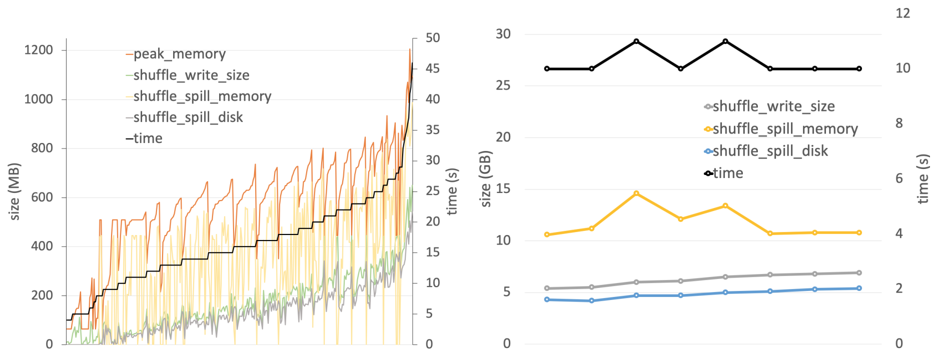
(a) clipping stage



(b) output stage

Fig. 5. Computational performance on the three test datasets.

Fig. 5 shows the computational scalability of the *map* stage (Fig. 5a), including the tiling and clipping, and the *reduce* stage of outputting (Fig. 5b). For all three datasets, doubling the number of cores used by each executor, from 1 to 4, reduced the stage time. As more cores were used, the data shuffling overhead outweighs the benefit of additional cores. Weak scaling can also be seen by looking at the dataset-cores combinations. The (*small–8 cores, medium–16 cores, large–32 cores*) combination shows a sublinear increase of overhead from increased data shuffling cost in Spark, which is normal. Since the computing complexity of the clipping algorithm is linear, this scaling performance is not surprising. At the same time, Spark did not introduce significant overhead in data and task management.



(a) task view (b) executor memory use

Fig. 6. Memory profile and load balance on the large dataset (303 tasks) using 8 executors, each using 32GB memory and 8 cores.

Fig. 6 shows the computational profile of the *map* stage of a run using 8 executors and 8 cores per executor to clip the 141GB large dataset, which contains 303 data blocks. Fig. 6a depicts the time and memory usage for each of the 303 tasks, ordered by the task time, i.e., the black solid line in the diagram. The clipping time is heterogeneous among tasks, depending on how computing intensive a shape intersection operation can be (tiles out of a shape's bounding box is calculated faster) and how many tile cells are intersected. The memory usage is also heterogeneous. A time-consuming shape intersection operation may result in a small number of data to be extracted, which explains why some tasks took longer but consumed less memory. The four memory profiling measures show the maximum memory usage, the number of clipped tile cells, the average runtime memory and disk cache consumption. Fig. 6b, however, shows that the computing and data load of such heterogeneous geocomputation are evenly distributed among the eight executors. The explanation has two components. First, RDD data blocks are evenly sized into tiles and distributed to executors. On aggregation, the memory and cache usage among executors are thus balanced. Second, small variations in time and runtime memory usage, i.e., *shuffle_spill_memory*, are caused by task heterogeneity but smoothed across tasks running on an executor.

Table 2. Turnaround clipping time (in seconds) by different clipping methods using 32 processor cores. The buffered method used a 40MB memory buffer.

clipping method	large dataset	medium dataset	small dataset
buffered	2106	726	247
in-memory	729	389	113
spark	282	166	89

The turnaround time of the Spark clipping implementation was also compared with embarrassingly parallel processing of sequential clipping functions using the open source GDAL [29] library. The existing clipping in CFIM uses a 40MB memory buffer. Another configuration that uses only in-memory processing was also tested. The Spark run used 8 executors with 32GB memory and 4 cores per executor. Each scenario used 32 cores in total. Table 2 shows that the Spark implementation clearly outperformed both batch processing methods on each test dataset, mainly due to the RDD data parallelism and the resulting load balance.

A CONUS clipping test was also conducted to obtain DEMs for each of the 331 HUC6 units on the entire elevation dataset using 32 cores in total. The *map* stage took 7.3 minutes to tile the input DEM and clip HUC6 shapes, and generated 2829 data blocks, 170GB in total. The total memory and cache usage was approximately 475GB. The output stage took 12 minutes to dump output rasters to a Network File System (NFS) mount due to the limitation of local disk size. The total turnaround time was 19.3 minutes for all 331 HUC6 units. Compared to the average 4 minutes of single HUC6 clipping in the existing CFIM workflow, this is a dramatic performance improvement, which can be further optimized by using parallel file system storage.

In summary, these computational experiments demonstrate a desirable computing and data management performance for the Spark environment. Task management, DAG execution, RDD management, and memory/disk spilling at runtime did not introduce obvious overhead and interference with actual computing and data handling.

6 Concluding Discussion

Science communities have been actively employing both data science and scientific computing for science discovery and innovation. The fusion of HDA and HPC becomes a prominent need. Here, we have explored the convergence of

HDA and HPC in a geocomputation scenario and studied the software components that require technical innovations to accelerate the end-to-end performance. Spark is a scalable data-intensive computing solution that has comprehensive virtualization, scheduling, and resource allocation strategies. It provides an enabling software infrastructure for HDA in geocomputation. Integrating MPI applications in Spark context is feasible.

Our proposed design is applicable to general geocomputation applications—transforming an HPC workflow into data-driven HDA+HPC hybrid solutions is a promising path for resolving the computational bottlenecks introduced by GIS software limitations and its associated data and computing challenges. Specific spatial characteristics and geospatial workflow patterns may also be leveraged for improving data logistics and resource management on cloud and HPC infrastructure. Raster operations such as local, focal, and zonal map algebra can be effectively transformed into mapreduce functions. Vector operations can also be transformed using distributed graph libraries, such as GraphX in Spark [9], and vector decomposition techniques, such as vector tiles [25]. Sequential implementation can be directly incorporated into the map functions. Development and computation of the reduce functions, however, are non-trivial and require further computational studies. When multiple distributed datasets (RDDs) interoperate, frequent data shuffling may significantly increase computational cost. Specific spatial indexing, caching, and partitioning schemes are needed to address the challenges in runtime data management and task scheduling.

HPC has been a major accelerator for machine learning algorithms. As deep learning turns to self-supervised learning to identify patterns and create knowledge within a dataset itself, large-scale data transformation and augmentation solutions [28] become critical for enabling scalable learning from massive datasets. GeoAI [22, 21] is no exception. In general, as data and learning become increasingly important in a scientific computing application, the fusion of HDA and HPC will pave the way to a converged platform and programming interface for domain application development. For instance, to make data interoperable, there have been efforts to make columnized table and distributed datasets [7] standard in data analytics and machine learning libraries for GPU [26, 27], data-intensive computing [36, 31], and cluster computing to seamlessly share data in different memory hierarchies.

Acknowledgements

Liu's work is partly supported by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725. This research used resources of the Compute and Data Environment for Science (CADES) at ORNL, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The data registration and publishing used the Constellation Data Portal, a feature in

the Scalable Data Infrastructure for Science (SDIS) at the Oak Ridge Leadership Computing Facility (OLCF) in ORNL.

References

1. Anderson, M., Smith, S., Sundaram, N., Capota, M., Zhao, Z., Dulloor, S., Satish, N., Willke, T.L.: Bridging the gap between hpc and big data frameworks. Proceedings of the VLDB Endowment **10**(8) (2017) 901–912
2. Asch, M., Moore, T., Badia, R., Beck, M., Beckman, P., Bidot, T., Bodin, F., Cappello, F., Choudhary, A., de Supinski, B., Deelman, E., Dongarra, J., Dubey, A., Fox, G., Fu, H., Girona, S., Gropp, W., Heroux, M., Ishikawa, Y., Keahey, K., Keyes, D., Kramer, W., Lavignon, J.F., Lu, Y., Matsuoka, S., Mohr, B., Reed, D., Requena, S., Saltz, J., Schulthess, T., Stevens, R., Swany, M., Szalay, A., Tang, W., Varoquaux, G., Vilotte, J.P., Wisniewski, R., Xu, Z., Zacharov, I.: Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. The International Journal of High Performance Computing Applications **32**(4) (2018) 435–479
3. Astraea: Rasterframes. https://rasterframes.io/ (2020)
4. Baig, F., Mehrotra, M., Vo, H., Wang, F., Saltz, J., Kurc, T.: Sparkgis: Efficient comparison and evaluation of algorithm results in tissue image analysis studies. In: Biomedical Data Management and Graph Online Querying. Springer (2015) 134–146
5. Bhaduri, B., Lunga, D.D., Yang, L., Mckee, J., Laverdiere, M., Kurte, K.R., Sanyal, J.: High performance geocomputation for assessing human dynamics at planet scale. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States) (2019)
6. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/x100: Hyper-pipelining query execution. In: Cidr. Volume 5. (2005) 225–237
7. DLPack-RFC: Dlpack: Open in memory tensor structure. https://github.com/dmlc/dlpack (2020)
8. Gittens, A., Rothauge, K., Wang, S., Mahoney, M.W., Kottalam, J., Gerhardt, L., Prabhat, Ringenburg, M., Maschhoff, K.: Alchemist: An apache spark-mpi interface. Concurrency and Computation: Practice and Experience **31**(16) (2019) e5026
9. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: Graph processing in a distributed dataflow framework. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). (2014) 599–613
10. Goodchild, M.F., Longley, P.A.: The practice of geographic information science (2014)
11. Hey, T., Tansley, S., Tolle, K., et al.: The fourth paradigm: data-intensive scientific discovery. Volume 1. Microsoft research Redmond, WA (2009)
12. Horovod: Horovod in spark. https://horovod.readthedocs.io/en/stable/spark_include.html (2020)
13. Hughes, J.N., Annex, A., Eichelberger, C.N., Fox, A., Hulbert, A., Ronquest, M.: Geomesa: a distributed architecture for spatio-temporal fusion. In: Geospatial Informatics, Fusion, and Motion Video Analytics V. Volume 9473., International Society for Optics and Photonics (2015) 94730F

14. Kini, A., Emanuele, R.: Geotrellis: Adding geospatial capabilities to spark. Spark Summit (2014)
15. Klasky, S., Wolf, M., Ainsworth, M., Atkins, C., Choi, J., Eisenhauer, G., Geveci, B., Godoy, W., Kim, M., Kress, J., et al.: A view from ornl: Scientific data research opportunities in the big data age. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE (2018) 1357–1368
16. Liu, J., Racah, E., Koziol, Q., Canon, R.S., Gittens, A.: H5spark: bridging the i/o gap between spark and scientific data formats on hpc systems. Cray user group (2016)
17. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., et al.: Hello adios: the challenges and lessons of developing leadership class i/o frameworks. Concurrency and Computation: Practice and Experience **26**(7) (2014) 1453–1473
18. Liu, Y.Y., Maidment, D.R., Tarboton, D.G., Zheng, X., Wang, S.: A cybergis integration and computation framework for high-resolution continental-scale flood inundation mapping. JAWRA Journal of the American Water Resources Association **54**(4) (2018) 770–784
19. Liu, Y.Y., Tarboton, D.G., Maidment, D.R.: Height above nearest drainage (hand) and hydraulic property table for conus version 0.2.0. (20200301). https://cfim.ornl.gov/data/ (2020)
20. Liu, Y.Y., Tarboton, D.G., Maidment, D.R.: Height above nearest drainage (hand) and hydraulic property table for conus version 0.2.1. (20200601). https://cfim.ornl.gov/data/ (2020)
21. Lunga, D., Gerrand, J., Yang, L., Layton, C., Stewart, R.: Apache spark accelerated deep learning inference for large scale satellite image analytics. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **13** (2020) 271–283
22. Lunga, D.D., Bhaduri, B., Stewart, R.: The trillion pixel geoai challenge workshop. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States) (2019)
23. Malitsky, N., Chaudhary, A., Jourdain, S., Cowan, M., OLeary, P., Hanwell, M., Van Dam, K.K.: Building near-real-time processing pipelines with the spark-mpi platform. In: 2017 New York Scientific Data Summit (NYSDS). (2017) 1–8
24. Malitsky, N., Castain, R., Cowan, M.: Spark-mpi: Approaching the fifth paradigm of cognitive applications. arXiv preprint arXiv:1806.01110 (2018)
25. Mapbox: Vector tiles reference. https://docs.mapbox.com/vector-tiles/reference/ (2020)
26. Nvidia: cudf - gpu dataframes. https://github.com/rapidsai/cudf (2020)
27. Nvidia: Gpu data science. https://rapids.ai (2020)
28. Nvidia: The nvidia data loading library (dali). https://developer.nvidia.com/DALI (2020)
29. OSGeo: The geospatial data abstraction library (gdal). https://gdal.org (2020)
30. Palamuttam, R., Mogrovejo, R.M., Mattmann, C., Wilson, B., Whitehall, K., Verma, R., McGibbney, L., Ramirez, P.: Scispark: Applying in-memory distributed computing to weather event detection and tracking. In: 2015 IEEE International Conference on Big Data (Big Data), IEEE (2015) 2020–2026
31. Rocklin, M.: Dask: Parallel computation with blocked algorithms and task scheduling. In: Proceedings of the 14th python in science conference. Number 130-136 (2015)

32. Tang, M., Yu, Y., Malluhi, Q.M., Ouzzani, M., Aref, W.G.: Locationspark: A distributed in-memory data management system for big spatial data. Proceedings of the VLDB Endowment **9**(13) (2016) 1565–1568
33. Vazhkudai, S.S., Harney, J., Gunasekaran, R., Stansberry, D., Lim, S.H., Barron, T., Nash, A., Ramanathan, A.: Constellation: A science graph network for scalable data and knowledge discovery in extreme-scale scientific collaborations. In: 2016 IEEE International Conference on Big Data (Big Data), IEEE (2016) 3052–3061
34. You, S., Zhang, J., Gruenwald, L.: Large-scale spatial join query processing in cloud. In: 2015 31st IEEE International Conference on Data Engineering Workshops, IEEE (2015) 34–41
35. Yu, J., Wu, J., Sarwat, M.: Geospark: A cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. (2015) 1–4
36. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I., et al.: Spark: Cluster computing with working sets. HotCloud **10**(10-10) (2010) 95
37. Zheng, X., Tarboton, D.G., Maidment, D.R., Liu, Y.Y., Passalacqua, P.: River channel geometry and rating curve estimation using height above the nearest drainage. JAWRA Journal of the American Water Resources Association **54**(4) (2018) 785–806