Feature-based Analysis of Large-scale Spatio-Temporal Sensor Data on Hybrid Architectures

Joel Saltz¹, George Teodoro¹, Tony Pan¹, Lee Cooper¹, Jun Kong¹, Scott Klasky², and Tahsin Kurc¹,²

¹Center for Comprehensive Informatics and Biomedical Informatics Department Emory University
²Scientific Data Group, Oak Ridge National Laboratory

Abstract

Analysis of large sensor datasets for structural and functional features has applications in many domains, including weather and climate modeling, characterization of subsurface reservoirs, and biomedicine. The vast amount of data obtained from state-of-the-art sensors and the computational cost of analysis operations create a barrier to such analyses. In this paper, we describe middleware system support to take advantage of large clusters of hybrid CPU-GPU nodes to address the data and compute-intensive requirements of feature-based analyses in large spatio-temporal datasets.

1. Introduction

Advances in sensor technologies make it possible to rapidly collect vast quantities of low-dimensional, spatio-temporal datasets¹. Analysis and characterization of features (e.g., spatial structures, their properties, function of the properties over space and time) in these datasets are important in many scientific domains, including weather prediction and climate modeling, earth systems science, biomedicine, and materials science. Satellite observations of the earth, for instance, provide valuable data regarding spatially distributed phenomena (wild fires, air and water pollution, vegetation index, and weather patterns) at regional to global scales. An earth systems researcher may want to investigate the growth of residential zoning in a given area and the decline of certain vegetation in surrounding areas using satellite imagery. The researcher would use sensor readings from different satellites to detect residential areas and the density and types of the flora. She could then compute correlations between vegetation density and the size and proximity to vegetation of residential areas using a series of readings over time. In biomedical research, advanced microscopy scanners can capture incredibly detailed pictures of tissue in a few minutes and enable healthcare organizations to collect thousands of images daily. Histo-pathological analysis of images from tissue samples enables the examination of disease morphology at the sub-cellular level. In a recent study of glioblastoma, Cooper et al. have characterized morphological extracts from serial sections of high resolution image data and correlated them with clinical data and

¹Spatio-temporal datasets contain data elements, which are associated with coordinates in a multi-dimensional space with low-dimensionality and potentially obtained at multiple time steps.
extracted various subtypes\[1\]. That work demonstrated that morphological features self-aggregate into distinct clusters with significantly different clinical outcomes.

In order to fully exploit the potential of spatio-temporal sensor datasets in scientific research, high performance computing capabilities are needed to rapidly extract and classify various features from large volumes of data, ranging from multiple terabytes to petabytes, using data and computation intensive analysis pipelines. While datasets in an application domain are used to answer domain specific questions, analysis processes have common patterns. Analysis of a spatio-temporal dataset involves detecting spatial objects of interest, characterizing their structural and functional features\[2\], and monitoring and quantifying changes over space and time. These processing patterns are realized by composition of a suite of core operations (see Table 1) into analysis pipelines. A middleware framework that can support the core operation categories and processing patterns can benefit a wide range of applications.

In this paper we examine middleware system support that targets clusters containing large numbers of hybrid CPU-GPU nodes. These systems are becoming widely deployed at national laboratories and supercomputing centers. While they provide significant processing power and memory capacity, they introduce increased complexities of programming and scalability due to more complex architectural characteristics, including higher concurrency, deep memory hierarchies, and new forms of heterogeneity.

2. Data Processing Patterns and Challenges in Analysis of Spatio-temporal Sensor Datasets

Table 1 lists core operation categories along with data access and computation patterns in each category. Table 2 shows a list of application specific operations in pathology image analysis\[2\], weather prediction studies\[3\], and change analysis using satellite imagery\[4\]. As is seen from the table, applications from these domains have similar operations, although they use spatio-temporal datasets for different purposes. The core operations can produce different levels of data products that can be consumed by client applications. For example, a client application may request only satellite imagery data covering the east coast of the US; this request can be satisfied by a data subsetting operation applied on global coverage satellite data to retrieve sensor readings corresponding to the US East Coast. The operations can also be chained to form analysis workflows to create data products. An example workflow could be a pipeline of: [data cleaning → mapping → object segmentation → object classification → change detection] operations.

The data access patterns and computational complexity of the core operations presented in Table 1 are important factors in I/O, communication, memory, and processing overheads in analysis applications. The local data access patterns involve retrieval and processing of a single data element or data elements within a small neighborhood in a spatial and temporal region (e.g., data accesses by data cleaning and low-level transformation operations). The

---

\[2\] We refer to structural features as the spatial characteristics of objects such as shape and area, whereas functional features represent signals measured within the space occupied by the object such as texture and intensity.
regular access patterns perform sweeps over data elements, while irregular accesses may involve accesses to random collections of data elements (e.g., data accesses by certain types of object classification algorithms, morphological reconstruction operations in object segmentation). Generalized reductions (e.g., aggregation) and indexed accesses (e.g., queries for data subsetting and change quantification) also are common patterns. Analytics applications composed of the core operations encapsulate several application-level data processing patterns as well. First, datasets are often partitioned into tiles or chunks, and some operations in Table 1 can be executed on each chunk independently. This leads to a bag-of-tasks processing pattern. Second, processing of a single chunk or a group of chunks can be expressed as a hierarchical coarse-grain dataflow pattern\[5, 6\]. For example, transformation, filtering, mapping, and segmentation operations can be composed into a workflow. The segmentation operation itself may consist of a pipeline of lower-level operations as well. Third, several types of operations such as aggregation and classification can be represented as MapReduce style\[7\] computations.

We describe pathology image analysis in biomedicine as an example to illustrate the core operation categories and the challenges of synthesizing and classifying features in more detail. Biopsied tissues contain morphology information that manifests from various molecular interactions that arise from disease or therapy. Microscopic examination of a tissue reveals this information enabling the pathologist to render accurate diagnoses and guide therapy. Some crucial disease sub-types can be identified through image analysis of whole slide tissue images. A typical analysis starts with selecting a subset of images and performing color normalization to reduce image acquisition artifacts. If the dataset consists of stacks of images, the images may be registered to each other to form a 3D volume. If there is an application-specific atlas (e.g., a brain atlas), each image may be mapped to the atlas. A segmentation algorithm is executed on each image to determine the boundaries of micro-anatomic objects, such as nuclei, cells, and small blood vessels. A set of shape and texture features is computed for each segmented object. The objects are classified into groups using a machine-learning algorithm such as clustering. Feature vectors for objects may also be aggregated to compute image level features. The images are grouped into categories using similar machine-learning algorithms. Feature sets from different image datasets can be compared to find common occurrences and differences in patient populations, or to assess the sensitivity of analysis algorithms to input data.

Analysis of whole slide images is a data and computation intensive process. There are about $10^5$ to $10^7$ cells in a $10^5\times10^5$-pixel image – the size of such an image is about 40GB uncompressed. The process of classifying a segmented cell is done using roughly 10–100 shape and texture features. A state-of-the-art scanner can capture hundreds of images per day, and a project can collect and reference thousands of slides. An analysis limited to classifying cells in a dataset with a thousand images could require processing of $10^{12}$ features. Computational requirements for segmentation or feature extraction are approximately 10–15 hours on a single processor system for a single image at a resolution of 0.5 microns/pixel – this corresponds to processing approximately $10^8$ features. The computational requirements increase significantly when a detailed characterization of morphology in a large image dataset is carried out. This process requires the coordinated use
of many interrelated analysis pipelines in order to improve analysis accuracy and
classification power and to assess the sensitivity of analysis to input data.

3. Middleware Software Support

The middleware framework to support the core operations and analysis pipelines should
provide solutions in the form of efficient data representations, programming abstractions,
and runtime that leverage the heterogeneity and memory hierarchies of available computing
platforms. In the following sections, we present our research in these areas.

Data Representation

We are developing a data representation and library to support data exchange among
application-specific instances of the core operations and facilitate runtime functionality in a
unified framework with high performance. For this purpose, we define a region template
abstraction to describe 2D/3D static and temporal regions. The objective with the region
template is to provide a generic container template for common data types in spatio-
temporal datasets: points, arrays, meshes, and spatial objects within a spatial and temporal
bounding box – an array represents a set of connected points on a regular grid, whereas a
mesh is a set of points connected in a graph. A data region object is a storage materialization
of a region template and contains elements from a spatio-temporal region. Data elements
contained in a data region can be stored in dense or sparse structures; an image tile, for
example, would be represented with a dense 2D array, while a group of objects would be
stored as a set. Application operations would interact with data region objects to read and
write/store data elements; the runtime system would manage the data regions and their
content across memory hierarchies for high performance. A data region is identified by a
(namespace::key, timestamp, version number) triple. This triple identifier intends to provide
temporal relationships among data regions related to the same spatial area. The metadata for
a data region contains one or more bounding boxes and the type of the data region: array;
mesh, points, objects, and compound – a compound data region contains other data regions.
Data regions corresponding to the same spatial area may contain different data types and
data products. For instance, a region template may define a data region of a 1km by 1km
regular grid over the USA; another one may define a data region for states; and a third may
specify a data region storing the precise spatial coordinates of satellite-borne sensor
readings. Data regions may be related to one another in a defined manner. For instance,
region templates may specify a precise correspondence or mapping between 30km by 30km
grid elements in data region A and 1km by 1km grid elements in data region B. The spatial
and temporal relationship information can be used for supporting decisions regarding the
placement and computation of data regions in analyses applications. For instance, in
operations applied to objects, objects’ location in the data domain can be used to perform
better distribution of the computation in a distributed environment, as compared to statically
partitioning the data domain in equal size data chunks without considering object locations.

We are in the process of implementing a region template library. This library provides
constructors for defining region templates and instantiating data regions and mechanisms for
accessing data elements. Access mechanisms will support associative queries and direct
access to data elements. Data regions can be associated with Data Source components, which encapsulate mechanisms for storage. This design allows for the decisions regarding data movement and placement to be delegated to the runtime environment, which may use different layers of a system memory to store the data according to the analysis pipeline used. The current implementation supports Data Sources for file based, CPU memory, and GPU memory storage of data regions. The ability for the runtime system to transfer data regions back and forth between different memory layers is especially important to take advantage of CPU-GPU equipped nodes.

Programming Abstractions and Runtime Services for Distributed Execution

Applications that make use of spatio-temporal sensor datasets share common processing structures, which include bag-of-tasks execution, generalized reduction and MapReduce[7], and coarse-grain dataflow[5, 6], as described in Section 2. Our earlier work has shown that many processing patterns in multi-dimensional scientific data analyses can be implemented as MapReduce style applications[8]. The coarse-grain dataflow pattern can be supported by a filter-stream framework [5, 6], in which an analysis application is expressed as a network of components linked through logical streams to exchange data and control information. These frameworks provide a flexible application development platform and hide the complexities of runtime optimizations from the developer. At the same time, the data access, communication, and processing structure of the application is exposed which can be leveraged for performance optimization at the middleware level. Nevertheless, optimizing the transport and analysis of large volumes of data on high performance systems requires characterizing operational states and making appropriate and timely scheduling decisions along multiple application and hardware levels. Managing complex processing and data interactions among operations in an analysis application is a significant challenge on large-scale homogeneous clusters. It becomes more difficult on systems with nodes equipped with multiple GPUs and CPUs due to significant differences in their computing and memory characteristics[9–13].

We have developed runtime support that couples bag-of-tasks and filter-stream processing with function variants and implements performance-aware scheduling heuristics to exploit the aggregate computing capacity in hybrid systems via coordinated use of CPUs and GPUs[14]. A function variant is defined as a group of functions with the same name, arguments, and result type of a high-level, abstract operation, and enables the runtime system to choose the appropriate function or functions during execution, allowing multiple computing devices (e.g., CPUs and GPUs) to be used in a coordinated manner. In our image processing applications, we have observed performance variations in GPU vs CPU efficiency across different input data as well as across different types of operations[11]. A scheduling strategy for CPU-GPU machines should, therefore, take into account performance variability due to both different operations and different input datasets in order to achieve more effective assignment of operations to computing devices. We have devised a scheduling strategy, called PRIORITY, which uses a sorted queue of (task, data element) tuples based on the relative GPU/CPU speedup expected for each tuple. As more (task, data element) tuples are created by analysis operations, these tuples are inserted into the queue such that the queue remains sorted. When a CPU core or GPU remains idle, one of the tuples
from the queue is assigned to the idle device. If the idle device is a CPU core, the tuple with minimum estimated speedup is assigned to the CPU core. If the idle device is a GPU, the tuple with the maximum estimated speedup is assigned to the GPU. Although PRIORITY relies on speedup estimates, the speedup values are used only to order tasks. Thus, the most accurate speedup values are not needed as long as the order of tuples is not affected.

Analysis of large spatio-temporal datasets may also incur high I/O overheads. In addition to large volumes of input and potentially output data, intermediate results or checkpoints may need to be staged to disk. For example, studying the sensitivity to input parameters and algorithm variations of the segmentation stage output would require us to execute multiple runs, the results of which may not fit in memory due to time and resource constraints. The output from a stage in a single run may need to be stored on disk for inspection or visualization at a later time. The runtime system, hence, should provide capability to stage data to and from disk efficiently by buffering and organizing in-memory data and disk-bound data chunks, distributing data chunks across multiple storage nodes (e.g., on a high performance file system like Lustre), and overlapping I/O and computations. Since data can be processed in bag-of-tasks, dataflow, and MapReduce patterns, there can be concurrent access requests to the shared storage resources from tens of thousands of processes. The runtime system needs to coordinate all of these requests to minimize contentions. We are currently investigating the integration of Data Sources with efficient I/O mechanisms, such as ADIOS[15], which would provide I/O scalability on very large machines. These I/O mechanisms are based on a stream-I/O approach, drawing from filter-stream networks[5, 16] and data staging[17, 18]. In the stream I/O approach, a set of CPU cores, designated as I/O unit, are connected to the computation units via logical streams, over which data chunks are transmitted. Streams are managed by a scheduler, which assigns data chunks to I/O units to increase I/O performance and reduce resource usage. A group of I/O units can be collocated with a group of computation units or can be mapped to a separate set of machines in the system, such as its dedicated special-purpose I/O nodes.

4. Implementation and Experimental Evaluation

We have implemented and evaluated the PRIORITY scheduler[11] on the Keeneland system[19] for segmentation of nuclei and computation of object features in pathology image datasets. Keeneland has 120 nodes equipped with a dual socket Intel X5660 2.8 Ghz processor, 3 GPUs NVIDIA Tesla M2070 (Fermi), and 24GB of DDR3 RAM. The input image dataset used in this evaluation was obtained from in silico study of brain tumors[2]. Each image was partitioned into tiles of 4Kx4K pixels. The region template prototype for this implementation specifies a 2D array to store an input image tile and another 2D array for the mask array, which is created at the end of the segmentation stage to store segmented objects in the respective image tile. The feature computation stage computes features for segmented nuclei (“Per Nucleus”) and for the entire input image (“Per Image”). We implemented CPU and GPU versions of four types of features: Haralick, Sobel, Intensity, and Gradient. We refer the reader to [11] for implementation details. The GPU-vs-CPU speedup values for each operation using different tile types (Figure 1) are presented in Figure 3. The speedup values are relative to a single CPU core implementation.
The results highlight the fact that there are significant variations in GPU-vs-CPU speedups for these operations. The performance of “Per Nucleus” feature computations is both operation and data dependent – operations on Type 4 tile have the highest speedup, since Type 4 tile has more nuclei and present more parallelization opportunities. There is also variation in performance across different types of operations in “Per Image” computations, but the variation across tile types is small since the operations are applied on the entire tile irrespective of the tile’s content. We observe small speedup values for the intensity and gradient “Per Image” computations compared to “Per Nucleus” computations. This is because a histogram is computed for each nucleus in a tile in “Per Nuclei” computations, while only one histogram is computed for a tile in parallelism. These results justify the use of a dynamic, performance aware scheduling strategy.

We performed a scaling experiment in which the size of the input data and number of nodes used were increased proportionally for the feature computation stage. The input dataset for the baseline single node execution contains an image with 100 (4K × 4K) mixed type tiles, and the subsequent executions in multiple nodes scale this baseline dataset. Figure 2 presents the throughput (tiles processed per second) as the configuration of the application and the number of nodes is varied. The “Linear” curve refers to the linear throughput improvement based on the best single node execution. The results show that PRIORITY results in additional speedups of 1.88 and 1.80, respectively, with respect to the 3-GPU only cases and a First Come First Serve (FCFS) strategy, in which task tuples are assigned to idle devices in the order they are inserted into the queue.

Our initial investigation of middleware optimization for concurrent I/O demonstrated the need and benefits of special IO handling on large scale systems. Figure 4 presents the weak scaling performance of the baseline implementation where each compute unit performed its own I/O (baseline) vs an implementation where the disk writes were isolated in 300 dedicated I/O processes (dedicated I/O). This experiment, conducted on the Jaguar system at ORNL, showed that runtime scaled non-linearly with process count due to I/O concurrency. Using dedicated I/O processes improved the overall running time significantly, but network delays and read operations remained the majority of the running time. Complex interactions discovered during the initial investigation, such as reduction in read times when optimizing write operations, are subjects of our ongoing studies.

5. Related Work

Several libraries for scientific analysis have been proposed with the intention of increasing scientists’ ability to process and share data[21, 22]. These libraries provide high-level abstractions on spatial discretization and vector spaces, but do not have implementations on systems with CPU-GPU nodes. A recent work, EA VL[23], is designed to take advantage of CPUs and GPUs for visualization of mesh based simulations. Region Templates leverage some of the data description concepts proposed by Kohn and Baden[24], as we allow for a hierarchical representation of a low dimensional data domain, but region templates differ from the previous work in several ways. Region templates enable association of data from multiple sources to the same spatial region. This is a common scenario in sensor data analysis where multiple data measurements may be taken for the same region; for instance,
measurements of the humidity of certain region over time in monitoring and change detection analysis[4]. In pathology image analysis, it is common to have images stained with different markers associated with the same region. Region templates also support objects within regions. This adds flexibility and expressivity to the analysis framework that may employ work partitioning policies, which can more accurately estimate the computation costs of regions based on the density of objects, as is seen in the GPU-GPU scheduling results. There are several active projects targeting runtime support for efficient execution on CPU-GPU equipped machines, such as DAGuE[9] and StarPU[10]. These two projects focus on execution of regular linear algebra applications on CPU-GPU machines. They offer different scheduling policies, including those that prioritize computation of critical paths in the dependency graph to maximize parallelism. DAGuE assumes that the application DAG is static and known before execution. In our applications, however, the dependency graph representing the application must be dynamically built during the execution, as the computation of the next stage of the analysis pipeline may depend on the results of the current stage. Classification of features is an important step in image analysis. A commonly used methodology is to run k-means[25] or consensus clustering[26] on features (or labels) computed for each object or image. Recent work by Le et al. investigate unsupervised learning techniques to create high level features for categorizing images[27]. They present an implementation of their technique on a computation cluster with 16K cores.

6. Conclusions

Spatio-temporal datasets have applications in numerous scientific and engineering domains. While specific analyses are application and domain dependent, they share common processing patterns. Middleware systems designed for efficient realization of these patterns on emerging hybrid CPU-GPU systems can benefit a wide range of applications. Hybrid systems offer significant processing power for researchers to make use of very large sensor datasets. Efficient use of these systems requires new smart data structures, programming models, and runtime support because of the new forms of heterogeneity and deep memory hierarchies. In this work we have presented a framework that combines the concept of region templates, function variants, and dynamic performance-aware scheduling to address some of these challenges. A prototype of this framework on a hybrid system shows significant performance improvements in a pathology image analysis application, indicating its potential for enabling analysis pipelines for large-scale spatio-temporal datasets. Using the framework, we were able to execute the segmentation and feature computation steps on a dataset with 36,848 4Kx4K-pixel image tiles (about 1.8TB uncompressed) in less than 4 minutes on 100 nodes with the PRIORITY strategy[14, 20].

Acknowledgments

This work was funded, in part, by contract HHSN261200800001E by the NCI; and grants 5R01LM009239-04 and 1R01LM011119-01 from the NLM, R24HL085343 from the NHLBI, NIH P20EB000591, RC4MD005964 from NIH, and PHS Grant UL1TR000454 from the CTSA Program, NIH, NCATS. This research used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the NSF under Contract OCI-0910735.
Bibliography


Figure 1.
Tiles with different tissue coverage
Figure 2.
Feature Computation Scalability
Figure 3.
Performance evaluation of the GPU-based feature computation operations
**Figure 4.**

IO weak scalability with naive and dedicated IO implementations

<table>
<thead>
<tr>
<th>Process Count</th>
<th>2560</th>
<th>5120</th>
<th>10240</th>
<th>10240</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>baseline</td>
<td>dedicated IO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>network</td>
<td>21.27</td>
<td>140.36</td>
<td>1910.75</td>
<td>787.28</td>
</tr>
<tr>
<td>read</td>
<td>17.27</td>
<td>98.65</td>
<td>676.44</td>
<td>355.14</td>
</tr>
<tr>
<td>write</td>
<td>1.00</td>
<td>3.54</td>
<td>12.57</td>
<td>9.86</td>
</tr>
<tr>
<td>compute</td>
<td>81.17</td>
<td>73.63</td>
<td>64.86</td>
<td>68.11</td>
</tr>
</tbody>
</table>
## Table 1

Core operation categories in analytics applications for spatio-temporal sensor data.

<table>
<thead>
<tr>
<th>Core Operation Category</th>
<th>Operations</th>
<th>Data Access Patterns and Computational Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Cleaning and Low Level Transformations</td>
<td>Transformations to reduce effects of sensor/measurement artifacts. Transform sensor-acquired measurements to domain specific variables.</td>
<td>Mainly local and regular data access patterns. Moderate computational complexity.</td>
</tr>
<tr>
<td>Data Subsetting, Filtering, Subsampling</td>
<td>Select portions of a dataset corresponding to regions in atlas and/or time intervals. Select portions of a dataset based on value ranges (e.g., regions with temperature larger than X degrees). Resample data to reduce resolution and data size or to match resolutions for comparative or integrative analysis downstream.</td>
<td>Local data access patterns as well as indexed access. Low to moderate, mainly data intensive computations.</td>
</tr>
<tr>
<td>Spatio-temporal Mapping and Registration</td>
<td>Map datasets to an atlas. Resolve data redundancy at tile boundaries to form mosaics. Create composite dataset from multiple spatially co-incident datasets. Create derived dataset from spatially co-incident datasets obtained at different times.</td>
<td>Irregular local and global data access patterns. Moderate to high computational complexity.</td>
</tr>
<tr>
<td>Object Segmentation</td>
<td>Segment “base level” objects such as nuclei, buildings, lakes. Extract features from “base level” objects.</td>
<td>Irregular, but primarily local, data access patterns. High computational complexity.</td>
</tr>
<tr>
<td>Object Classification</td>
<td>Classify “base level” objects through possibly iterative combination of clustering, machine learning and human input (active learning).</td>
<td>Irregular and global data access patterns. High computational complexity.</td>
</tr>
<tr>
<td>Spatio-temporal Aggregation</td>
<td>Construct “high level” objects composed of classified “base level” object aggregates, e.g., residential areas vs industrial complexes. Compute time-series aggregates over a given imaged area.</td>
<td>Primarily local with a crucial global component for aggregation. Moderate/high computation complexity.</td>
</tr>
<tr>
<td>Change Detection, Comparison, and Quantification</td>
<td>Quantify changes over time in domain specific low level variables, base level objects and high level objects. Construct “change objects” to describe changes in low level domain specific variables, base level and high level objects. Spatial queries for selecting and comparing segmented objects.</td>
<td>Compute and data-intensive computations. Mixture of local and global data access patterns as well as indexed access.</td>
</tr>
</tbody>
</table>
Table 2
Operations from example application scenarios mapped to the core operation categories.

<table>
<thead>
<tr>
<th>Operation Category</th>
<th>Pathology Image Analysis</th>
<th>Weather Prediction</th>
<th>Monitoring and Change Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Cleaning and Low Level Transformations</td>
<td>Color normalization. Thresholding of pixel and regional gray scale values.</td>
<td>Remove anomalous measurements and convert spectral intensities to value of interest.</td>
<td>Remove unusual readings. Convert signal intensities to color and other values of interest.</td>
</tr>
<tr>
<td>Data Subsetting, Filtering, Subsampling</td>
<td>Selection of regions within an image. Thresholding of pixel values.</td>
<td>Spatial selection/cross match to find portion of a dataset corresponding to a given geographic region.</td>
<td>Spatial selection/cross match to find portion of a dataset corresponding to a given geographic region.</td>
</tr>
<tr>
<td>Spatio-temporal Mapping and Registration</td>
<td>Deformable registration of images to anatomical atlas.</td>
<td>Generation of mosaic of tiles to get complete coverage.</td>
<td>Registering low and high resolution images corresponding to same regions.</td>
</tr>
<tr>
<td>Object Segmentation</td>
<td>Segmentation of nuclei and cells. Compute texture and shape features.</td>
<td>Segmentation of regions with similar land surface temperature.</td>
<td>Segmentation of buildings, trees, plants, etc.</td>
</tr>
<tr>
<td>Object Classification</td>
<td>K-means clustering of nuclei into categories.</td>
<td>Classification of segmented regions.</td>
<td>Classification of buildings, trees, plants.</td>
</tr>
<tr>
<td>Spatio-temporal Aggregation</td>
<td>Aggregation of object features for per image features.</td>
<td>Time-series calculations on changing land and air conditions.</td>
<td>Aggregation of labeled buildings, trees, plants into residential, industrial, vegetation areas.</td>
</tr>
<tr>
<td>Change Detection, Comparison, and Quantification</td>
<td>Spatial queries to compare segmented nuclei and features.</td>
<td>Spatial and temporal queries on classified regions and aggregation to look for changing weather patterns.</td>
<td>Characterize vegetation changes over time and area.</td>
</tr>
</tbody>
</table>