SCALABILITY OF CONTEXTUAL GENERALIZATION PROCESSING USING PARTITIONING AND PARALLELIZATION

Marc-Olivier Briat, Jean-Luc Monnot, Edith M. Punt

Esri, Redlands, California, USA

mbriat@esri.com, jmonnot@esri.com, epunt@esri.com

Abstract

Contextual generalization tools need to efficiently analyze spatially related data. The generalization tools in ArcGIS 10 use either a TIN structure or an optimization engine which both rely on in-memory data structures. These tools are able to process efficiently approximately 100,000 features, but run out of memory as the number of features increases towards datasets containing millions. Workflows can be created to overcome those limits, but there is a significant cost in terms of data management.

A natural approach is to subdivide the dataset into partitions defined as polygons with a smaller geographic extent. This allows control over the number of features to process within each partition and is easily obtained by deriving rectangles from a quad tree structure. The main challenge with this approach is to ensure a seamless processing that will be independent of the partition boundaries.

A set of network tools is analyzed here to show how the impact of partition boundaries is minimized when processing a large seamless database. A second aspect is to consider what constraints need to be addressed to integrate parallel processing capabilities with partitions for those tools.

1 Introduction

At ArcGIS 10, tools were designed to be able to process approximately one map sheet worth of data. Depending on the dataset and the type of features involved, we generally consider 100,000 as a reasonable average number of features which those tools are able to process. Demand for web mapping requires processing much larger datasets, containing millions of features. Data is available in large seamless datasets contained in relational databases.

A strong constraint is to be able to process the dataset as is. It is always time consuming to prepare, reorganize, split, and reassemble a dataset. Although there are tools to perform those operations, being able to work with the native dataset offers a real advantage for efficient mapping production. A solution that would minimize those operations would make the overall workflow more efficient.

Contextual tools need to analyze surrounding features (Mustiere 2002). Typical operations that such tools need to perform mainly involve querying features in some extent around their location. For practical reasons, most of the tools load all data into memory. This overcomes the overhead of database queries, and significantly reduces the processing time. On the other hand, this limits the amount of features that can be processed.

One intuitive approach is to limit the number of features being processed by subdividing the dataset. For contextual tools, the set of features needs to be defined by a spatial extent, the problem being how to define these extents. They need to:

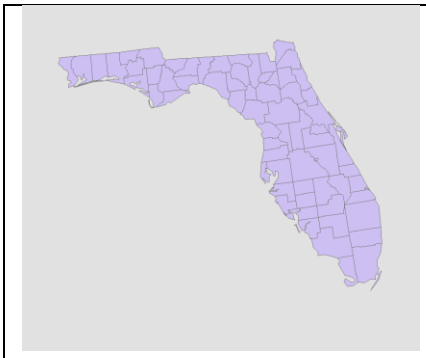
- Cover the entire dataset
- Obey the memory limitations
- Ensure seamless processing of the dataset

One approach is to build appropriate partitions so that algorithms can be applied considering only what is strictly inside the partition (Chaudhry, Mackaness 2010). In our approach, we deliberately chose to control the first two requirements with simple tools, and analyze how we can meet the seamless processing requirement. The first section of this document will look at this aspect.

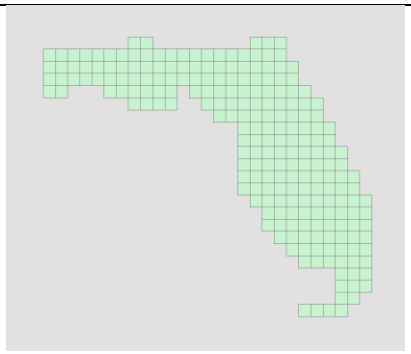
The choice of being flexible about what subsets of features to process is also dictated by the idea that parallel processing should be possible on those subsets. Optimizing such a parallel processing system may require being able to adjust the size of those subsets to best balance the work load. The second section of this document will look at this problem.

2 Partitions

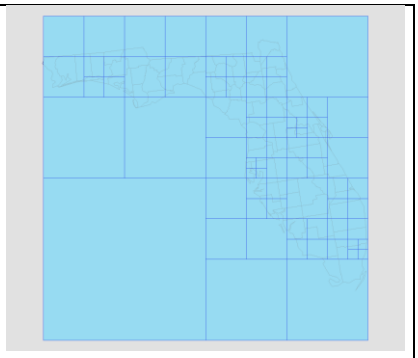
To identify subsets of features to process, we will rely on a mesh of polygons stored in a polygon feature class. These polygons will be referred to as partitions. Each partition defines the spatial extent in which we will consider features to be processed. Below are examples of partitions that we may use to process the road and street network of the state of Florida.



Counties



Grid cells



Quad tree cells

In terms of seamless processing, none of these examples have any sort of advantage. They all present the same artificial limits between partitions, independent of the underlying data. However, the quad tree partitioning makes it easy to control the volume of data in each partition, which has a direct impact on memory consumption and processing time.

In this approach, we consider that all partitions are going to be processed the same way. If partitions are derived such that the underlying data have common properties and can be processed with specific parameters, it would be possible to attach this information to each partition object, and use it to adjust the tool's behavior.

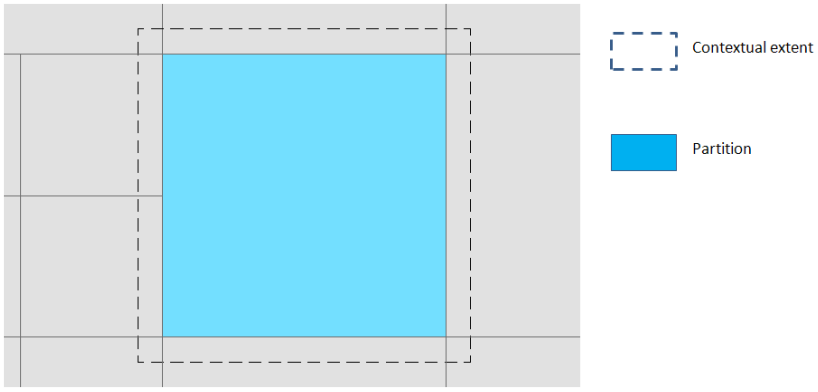
The obvious challenge with this approach is the processing at partition boundaries, and is a direct drawback of the freedom and simplicity allowed to create the partitions. We will definitely have features that overlap multiple partitions. Processing by partitions must ensure that the result is always the same (at least acceptably similar) regardless what partition processing order is used, or regardless what partition mesh is used.

From a workflow standpoint, the simplest approach is to process partitions sequentially, and consider the task done when the last partition is processed. Having a post processing step to fix any inconsistencies at partition boundaries would make the workflow more complex. What we want to do is to consider those constraints upfront and ensure that no post processing is necessary.

3 Contextual Tools

The contextual aspect of the tools consists in considering the environment based on proximity, distance or participation in larger structures (explicit or not). There cannot be an artificial spatial limit when considering only a portion of a seamless database. This requires to make a distinction between the data we want to modify (inside the partition) and the data necessary to perform the analysis (that would extend past the partition boundary), even though that may be the same dataset.

Assuming the tools can predict what extent is sufficient to define the result inside the partition, data should be loaded from that extent, and only results inside the partition should be stored, since this is the only processed area where the context is fully available. Results in the buffer cannot be validated.



The generic pattern is then:

- Calculate a buffered partition
- Load all data overlapping that buffer
- Process
- Save only data overlapping the initial partition

The generalization tools from the ArcGIS toolbox have been reviewed and we found that most of them have an implementation that obeys this logic. A buffer zone can be defined to include relevant contextual data. We'll review the implementation of some ArcGIS generalization tools (Punt, Watkins 2010), and check how they can be modified to work seamlessly at partition

boundaries. They also generally have a different strategy to save the result back to the database. Some examples are given in the next section. Building tools also support partitioning but their approach is similar to one that is described below, and therefore they are not detailed here. For each tool, we will describe the principles dictating what buffer value is to be used, as well as the strategy to maintain coherence between partitions.

It is important to note that not all tools would obey this logic. One example of this happens when contextual information is supported by the features themselves. This is the case for the Propagate Displacement tool, where the displacement value is feature specific. Unless you know the maximum displacement value in the entire dataset, a partition cannot know in advance the area that will influence the results. In this case, it is more appropriate to derive custom partitions. This problem is not studied here. Another example is the case of very long features. This happens when trying to optimize the layout of multiple parallel lines in a transportation network to avoid intersections. Obtaining the best solution requires propagating changes up to a non predictable extent. A full view of the data is necessary.

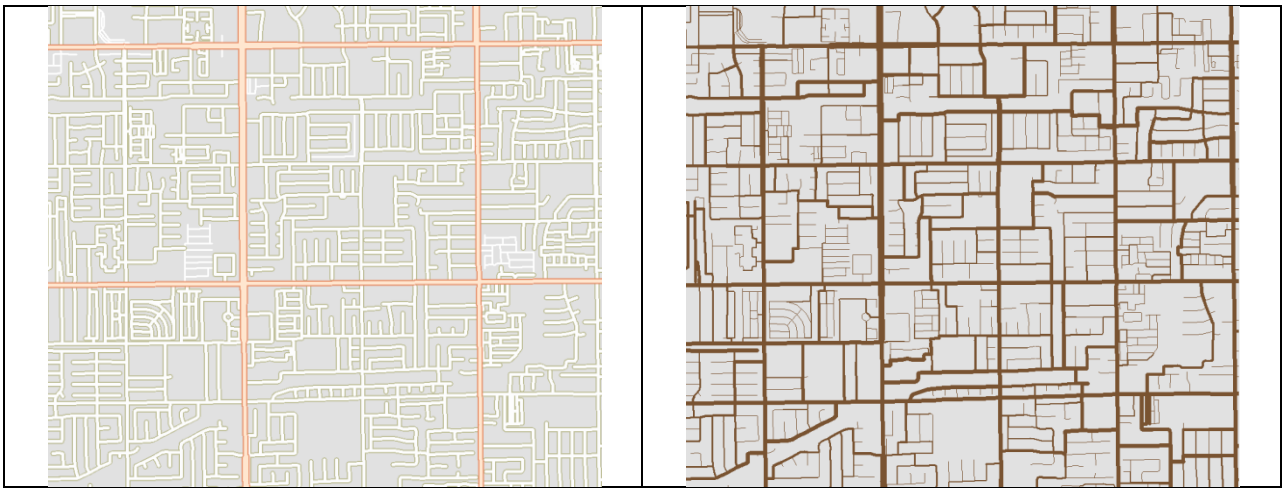
3.1 Thin Road Network

The Thin Road Network tool eliminates road features from display to create a simplified arrangement of roads that maintains connectivity along with the representative pattern and density of the original arrangement. The result is an attribute defining the visibility property of features.

Buffer value

This tool uses one single distance parameter, the Minimum Length, which can be interpreted as how long remaining elements should be. The term elements needs to be understood as a structure composed by individual features. An importance value is calculated and associated to each feature, representing its relevance in the entire network. This value is then compared to the Minimum Length. A notion of density is also considered, taking into account the average symbolization width. Finally, some visual constraints like alignment, continuity, and connection logic are also considered. However, the importance is the main contextual notion for this tool.

Roads being a transportation network, itineraries are used to assess this importance. Positions inside all possible itineraries (distance to extremities) are combined to evaluate the importance. The following figure shows how that differs from a simple attribute value.



Features will most likely be retained if they have an importance above this minimum distance value. Starting itineraries outside of the partition, from a distance at least the Minimum Length ensures we will catch all the values up to the Minimum Length inside the partition. There will be a cap effect (high importance values will be capped by the Minimum Length), but that will not change the decision if the feature should remain visible.

For additional safety, the buffer distance is defined as 1.5 times the Minimum Length.

For a scale jump to 100k, the Minimum Length used is around 2000m. The buffer size is no more than 10% of the size of a partition containing 100,000 features in a dense urban area (area of Los Angeles from the entire California street network). The additional extent is acceptable and corresponds to the worst case scenario. Higher scale jumps would require an increased Minimum Length, but are better managed with incremental steps, i.e. generalizing results from a previous scale jump. This maintains a similar ratio between the partition size and the buffer size.

Consistency

Only features within the partition or overlapping the partition boundary are modified in the dataset. We need to ensure that the next partition will be consistent with results already obtained.

The tool assumes that the visibility field is initially not defined. Any value in the data is then supposed to be the result of a previous partition processing. The Thin Road Network tool has a mechanism to lock the visibility of features. Locked features cannot be masked. The tool takes advantage of this to lock all the features already defined as visible in the buffer extent of the current partition. The optimization process will then provide a compatible solution for that partition.

However, this is not a guaranty that no mismatch will occur. To verify how consistently partitions are handled, only visible features have been locked. Results on a dataset containing 2.8 million features (the entire California road network) show that only 71 features introduce a new

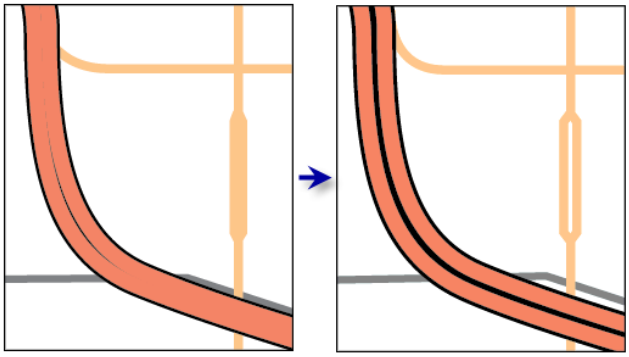
dangle in the network. The entire dataset has been processed with 157 partitions derived from a quad tree, each containing no more than 50,000 features. The number of features overlapping a partition boundary is 15,000 for the entire dataset. This visibility mismatch can be eliminated by also locking the invisible features. Also, the nature of the optimization approach for this tool doesn't guaranty a unique solution, only an acceptable solution. Although not ideal, the side effect at partition boundaries seems acceptable, and in most situations transparent.

Results

Example of iterative processing to assign the visibility.

3.2 Resolve Road Conflicts

The logic of this tool is to fix overlapping features considering the symbol attached to them. An additional gap may be defined to clearly separate the symbols after displacement.



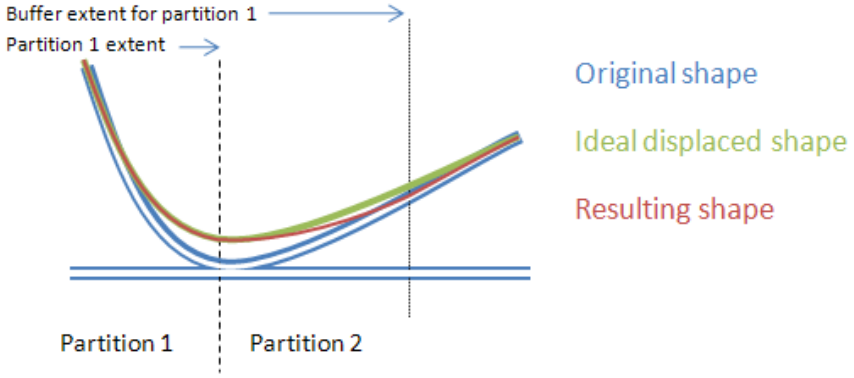
Buffer value

In a simple case involving two features, displacement is no more than gap + symbol width. If multiple parallel features are involved, this displacement may propagate and be multiplied by the number of features overlapping, assuming they have similar orientations. It is reasonable to have a limit for the number of overlaps, otherwise the map is trying to display too much information and some thinning or merging is most likely necessary before resolving the conflicts. This gives us some initial direction of what the partition buffer should be.

Consistency

One characteristic of this tool is to modify the shape of the original features. It is clear that the displacement is applied as calculated for geometries inside the partition. For consistency of the dataset, the displacement produced by the tool is propagated outside the partition, and is progressively decreased as its distance away from the partition border increases, always reaching zero at the buffer distance. This attenuation preserves the topology of the dataset.

Processing the adjacent partition will adapt to the results already produced by previous partitions. The processing happens on a slightly modified shape, so this will not produce strictly identical results. However, this is mitigated by the size of the buffer. The following figure shows the type of incorrect shapes that may be generated. The two blue lines are in conflict. Without partitions, we would obtain the green line as the resolution of the conflict. Processing partition 1 introduces the attenuation in the extent of partition 2. We see that the red shape is no longer in conflict and will be unchanged when processing partition 2. Extending the buffer size decreases this effect.

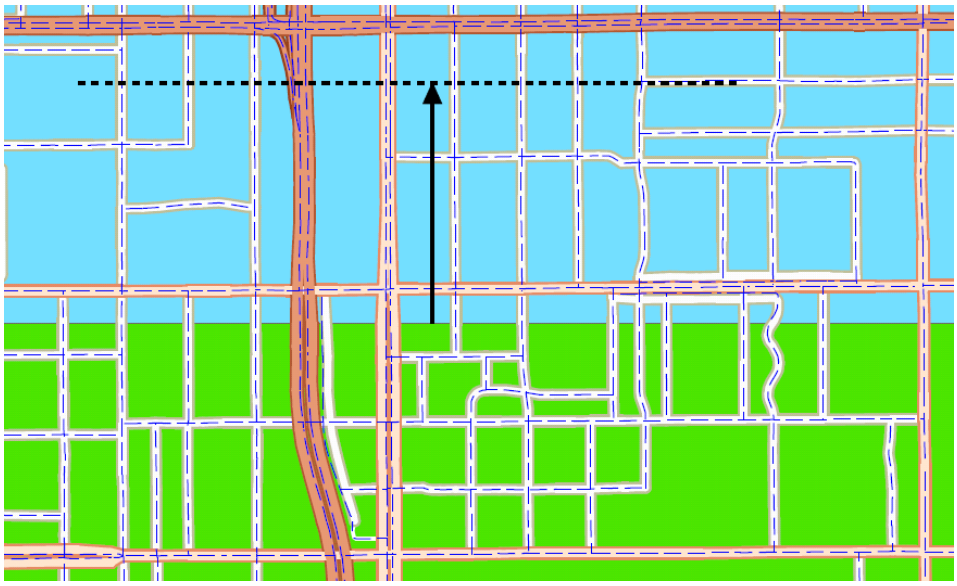


We use a value of 10 times the maximum symbol size + gap, which in practice limits the deformation. Also note that this effect applies to 15,000 out of 2.8 million features (0.5%), the vast majority of them crossing the boundary in a simple configuration.

For a maximum symbol width of 1.5mm at scale 1/50,000, this represents a 750m buffer. On our test dataset (the entire California road network) this never represents more than 5% of additional width or height on each side, which is 20% more data to consider.

Results

By construction, this method preserves connectivity and topology at partition boundaries. The following picture shows the intermediate result when only one partition is processed.



3.3 Merge Divided Roads

The Merge Divided Roads tool creates a single road from a matched pair of parallel-trending, equal classification roads. This tool creates a new output dataset. Its behavior is controlled by a maximum merge distance used to identify potential matching pairs.

Buffer value

The most important part of this tool consists of detecting parallelism in the network. A maximum distance given as a tool parameter limits this detection. The parallelism is validated only if it occurs on a significant length, defined as a multiple of the actual distance between parallels. If a really small part of the parallel roads starts near the edge of a partition, we need to identify it as a divided road to create the proper result. This tells us that the buffer value must be at least this multiple of the maximum distance between parallels.

The value used is 20 times the maximum merge distance, which is at least twice bigger than the multiple used for parallelism.

A common merge distance used for this tool is 100m, which gives a buffer extent of 2,000m. This is the same order of magnitude the Thin Road Network tool is using, so the same considerations apply with respect to feature count and partition extent.

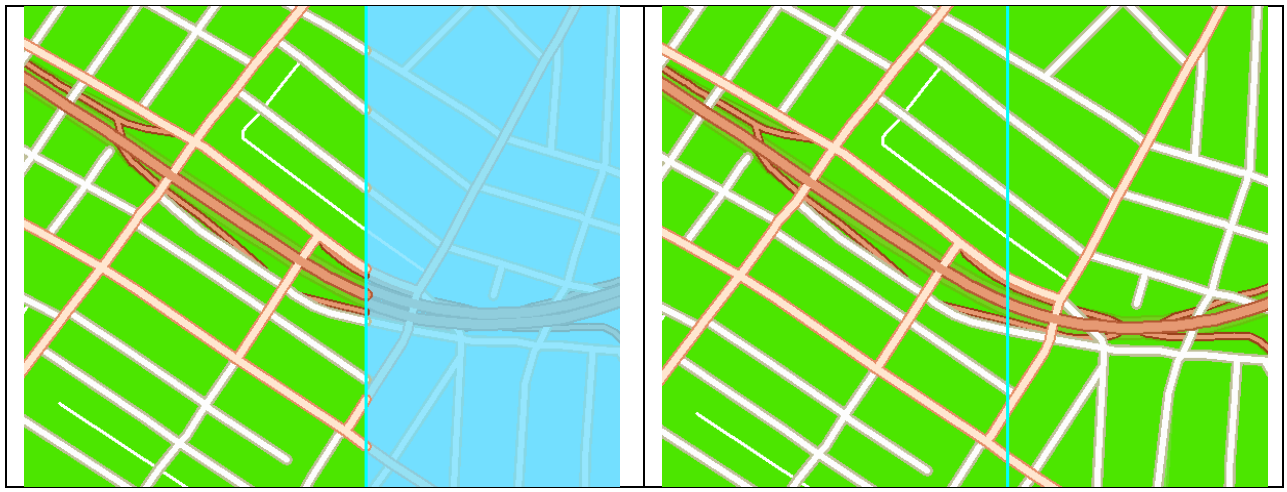
Consistency

The choice made for this tool is to have the output features split at partition boundaries. The tool uses a deterministic approach, which is a first step to ensure results should connect nicely when processed from either side of the boundary. However, the set of input features will come in a different order, which could make the algorithm produce slightly different results. In practice, those results differ by a very small value, usually related to the spatial tolerance of the dataset. It is very important to get a well connected dataset, because the resulting output is the starting point of further processing when creating a multiscale map. An additional step has been introduced to load any existing geometries touching the boundary of the currently processed partition, in order

to snap extremities of new features, assuming there is no ambiguous candidates for connection (in which case the information is recorded in the data for quality control).

Results

The following table illustrates the result of the left partition processing results, then the results from both partitions.



4 Parallel Processing Implementation

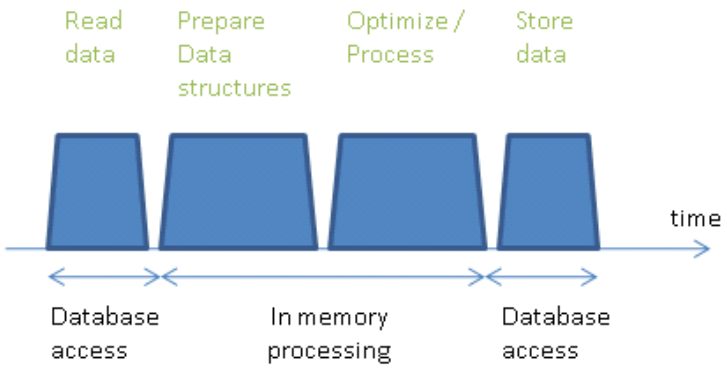
Considering that processing a seamless dataset using this buffered partitions approach is a reasonable way to go, it is possible to parallelize the execution of those tools.

This section describes an implementation of parallelization on conventional multicore PC systems. An Esri file geodatabase is used to manage the data. The file geodatabase has the following properties:

- Multiple simultaneous read operations are allowed,
- Only one single writer is allowed, requiring exclusive access to the dataset being written.

This implementation is designed to fit transparently into the ArcGIS geoprocessing framework. Even if the execution uses parallel processing, the external tool's behavior is unchanged. The tool can still be chained in a geoprocessing model with other tools that will consume the resulting data. Some additional processes will simply be created to work on the same task.

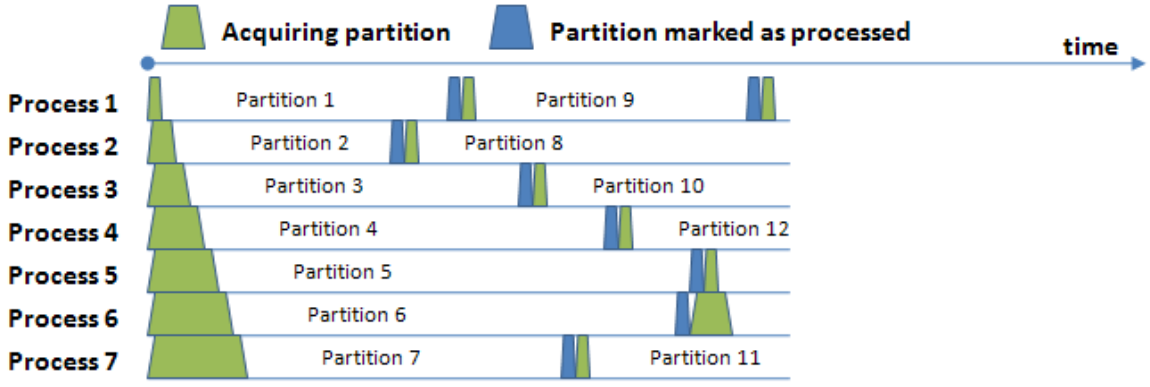
A typical processing looks like the following figure.



There is no question that the in memory processing will take full advantage of the available CPU. However, the database accesses need to be analyzed. The blocking factor will be write operations, since they cannot happen simultaneously on the database. For this problem, we need to distinguish two database components: the partitions and the datasets used by the tools.

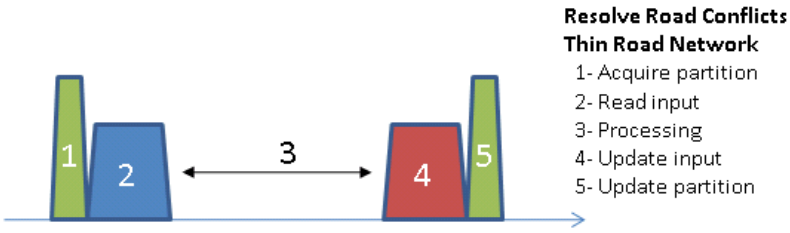
4.1 Partitions

The partition state (non processed, processed, being processed) is stored in a field of the partition feature class. Its access is ruled by the database mechanism. The same partition obviously cannot be assigned to multiple processes. A database lock is set to ensure that only one process picks a partition at any given time. The same is true when modifying the state of the partitions. It is critical that no read access happens simultaneously. Such locks can make processes wait until locks are released. In practice, the time required to acquire or save a partition is negligible compared to the processing time. Conflicts almost never occur. The sequence of partition processing order may look like the following figure. In addition to the processing start, one delay is shown for process 6.

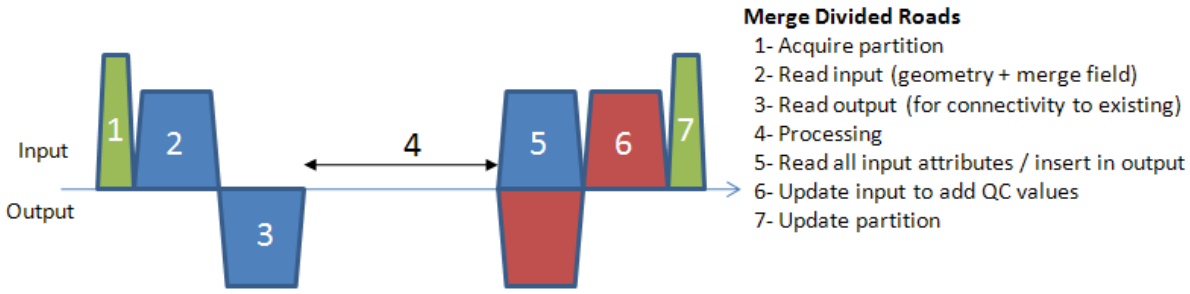


4.2 Input Data

Reading and writing the input data is more time consuming. Depending on the complexity of the tool, there may be multiple database operations involved, as well as multiple database access patterns. Here is the pattern for Resolve Road Conflicts and Thin Road Network.



The next figure shows a more complex pattern for the Merge Divided Roads tool.



Multiplying processes will make those patterns overlap, like those shown for the partitions processing diagram. This increases the potential of conflicts. The ratio of database access time for one dataset to the total partition processing time is a good indicator of the likelihood of being caught in access conflicts. For the Thin Road Network tool, this ratio is around 3%. With 8 processes, the probability of having a conflict is 22% ($1 – (1 – 0.03)^8$). Those numbers are very rough indications since the volume of partitions varies, the complexity of the algorithms is not linear, database lock resolutions add some delays, etc.

Nevertheless, the following CPU chart shows a typical execution on a quad core multithreaded PC. This example uses small partitions, increasing the number of conflicts.

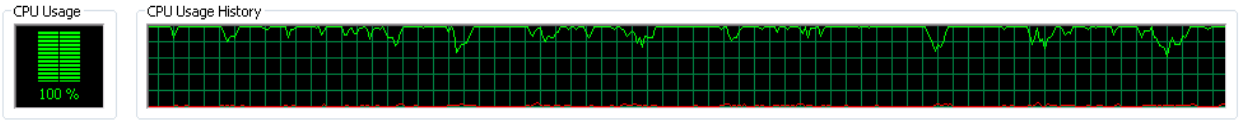
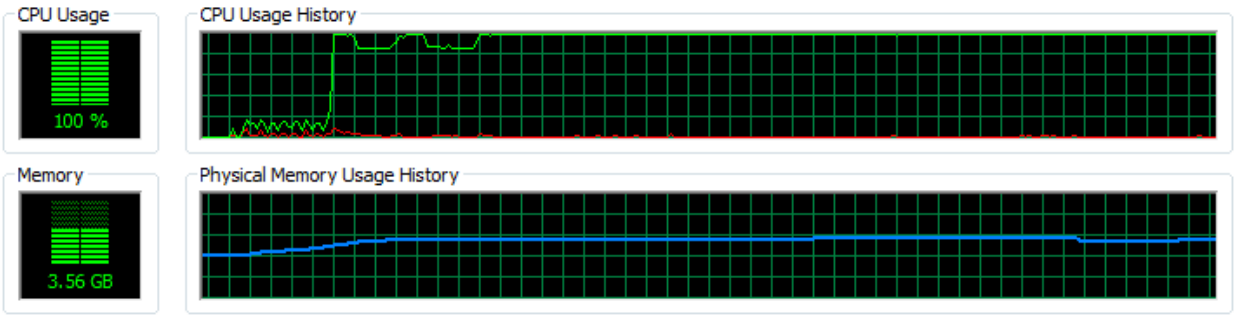


Figure 1: impact of database locks on CPU usage

The next CPU chart shows a more realistic usage with 8 processes and larger partitions. We can identify the initial phase of creating the 7 additional processes (in addition to the main one), then some conflicts where the CPU goes down because some processes are waiting for database access. We can also identify in the memory usage the concurrent data loading for all partitions, as well as some fluctuations due to different partition sizes. However, behavior is overall relatively stable, due to the choice of quad tree based partitions.



5 Adjacent Partitions

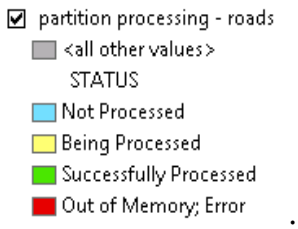
The tools described in the previous section use information from adjacent partitions which have been previously processed. Adjacent in this sense means "overlapping partition + buffer". The information used by each tool is:

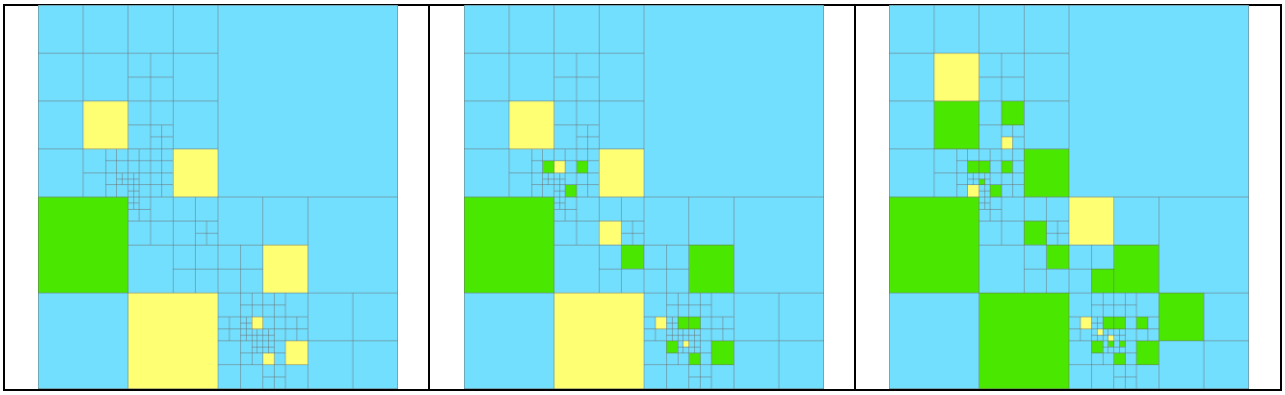
- The visibility attribute by Thin Road Network
- The modified shape by Resolve Road Conflicts
- The extremities of already processed features by Merge Divided Road

For this reason, we cannot process simultaneously partitions that overlap within the buffer distance. Each process needs to chose a partition that has no neighbors in the "being processed" state. This can be obtained by only reading the content of the database. Each process embeds the logic of analyzing partitions adjacency and database state to define what is available for processing. Note that this simple property makes it possible to use multiple machines since the database will provide the synchronization mechanism.

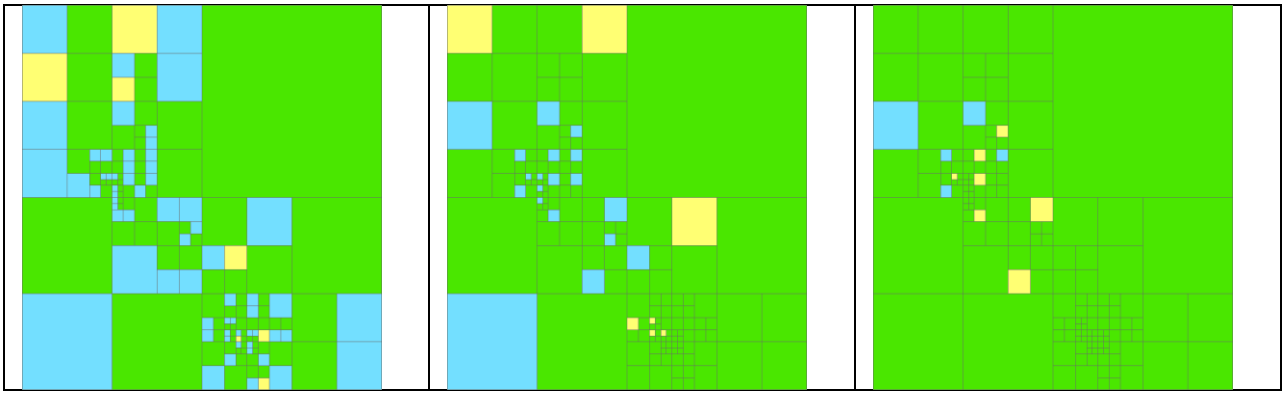
A first implementation considered the features creation order when selecting non adjacent partitions, which usually ended with the last partitions to process being packed together. There was a lot of time wasted waiting for partitions to be available for processing. The tool processed significantly slower at the end of the process, the problem being exacerbated by the number of processes.

A better execution plan is obtained by starting the processing with partitions having the highest number of unprocessed neighbors. By construction, the tool finishes with partitions that have the smallest number of partitions in the "being processed" state, ideally zero. An example of the evolution of processed partitions is given below, showing 7 processes.





Partitions being processed at the beginning



Partitions being processed at the end

The execution pattern may not seem intuitive but ensures the best usage of the available processes.

6 Results

Number of features: 2,860,000 roads (the entire state of California)

Number of partitions: 157 (no more than 50,000 features per partition)

The following table shows some results without and with parallel processing (PP).

Tool	**PC**	**With PP**	**Without PP**	**ratio**
Thin Road Network	4 core 4 processes	3h30min	13h30min	3.85
Check Network	4 core	6min30s	24min	3.7

Connectivity	4 processes			
Thin Road Network	4 core HT 8 processes	2h30min	10h30min	4.2
Merge Divided Roads	4 core HT 4 processes	45min	2h45min	3.7
Resolve Road Conflicts	4 core HT 8 processes	3h35min	12h30min	3.5

7 Conclusion / Perspective

The approach using buffered partitions gives very reasonable results when applied to a large seamless database for the set of tools studied here. Similar results can be obtained for the building tools. Analysis of the tool's execution profile shows a great potential to exploit multicore machines and dramatically reduce processing time, as more powerful machines become available. This approach is flexible since partition size can be adjusted to fit available computer capabilities. Existing tools (Aggregate Polygon) have been upgraded without much effort to take advantage of this approach.

ArcGIS offers a continually evolving variety of technologies to support multi user editing database access, replication, sharing, processing in the cloud. There is obviously some continuing work to do to best take advantage of new capabilities. However, the key point remains the ability to create tools that are not impacted by the partition shapes.

8 References

Mustiere, S.; Moulin, B.; Usery, E. L. (2002) ―What is spatial context in cartographic generalisation? ‖ Symposium on Geospatial Theory, Processing and Applications, Ottawa 2002.

Chaudhry, O. Z.; Mackaness W. A. (2010) – DTM Generalisation: Handling Large Volumes of Data for Multi-Scale Mapping || The Cartographic Journal November 2010

Punt, E.; Watkins, D. (2010) – User-directed generalization of roads and buildings for multi-scale cartography || 13th Workshop of the ICA commission on generalisation and multiple representation