A Pipeline Processing Approach To GIS

Michael Koutroumpas

EDINA National Data Centre, University of Edinburgh, 160 Causewayside, Edinburgh EH9 1PR, UK, m.koutroumpas@ed.ac.uk

Chris Higgins

EDINA National Data Centre, University of Edinburgh, 160 Causewayside, Edinburgh EH9 1PR, UK, chris.higgins@ed.ac.uk

Abstract. Much effort has been expended in the past researching parallel processing in GIS. However, uptake has been slow as the information architectures used were not based on interoperability standards and generally addressed only a small subset of GIS operations. Most research has concentrated on parallelizing specific algorithms for large-scale deployment rather than providing generic solutions. In this paper, based on open interoperability standards from both the Grid and Geospatial communities, we propose a pipeline processing approach to access, process and deliver geospatial data. The architecture is designed to be generic enough to be applicable in a wide range of spatial data infrastructure scenarios. We avoid the control and data dependencies that can hinder parallelization by taking into account the characteristics inherent in geographical data processing. GIS operations can then be modeled as a collection of processes arranged so that the output of each one is the input to the next. Using a feature based approach, a parallel processing pipeline can be constructed that overcomes problems imposed by dependencies between its processes. This paper describes an architecture based on widely available Grid middleware uniquely integrated with Open Geospatial Consortium (OGC) interface specifications. We further examine the requirements that should apply in the data stream between the processes to enable their parallelization. The viability of the model is demonstrated by applying it to selected GIS operations.

1 INTRODUCTION

Multiple core CPUs or multiprocessor machines are now routinely used for geospatial processing. Multiple GIS operations can be run simultaneously as different operating system processes on different processing units. This approach increases the throughput of the system under heavy load, but does not speed up single operations. On the other hand, speeding up single operations requires fine-grained parallelization to distribute the execution workload into many processing units. This approach requires extensive dependency analysis and is usually tightly coupled with specific operations. Most parallelization approaches therefore serve a specific niche of the geospatial community.

The work reported in this paper takes advantage of the fact that many GIS operations share common characteristics. In particular, we take advantage of the feature centric basis of the ISO TC/211 and Open Geospatial Consortium (OGC) suite of standards. Input data to an operation can usually be represented as a collection of independent geographical features that can be processed independently.

Furthermore, most GIS operations can be divided into subprocesses for data retrieval, processing and delivery. The processing step could include smaller subprocesses, such as generalization of the feature geometries (e.g. Deveau, 1985) or conversion to a raster image. Our goal is to create a group of threads that execute these subprocesses simultaneously while avoiding any dependency issues.

We call these processes generically as “tasks” in order to represent the overall model in a human readable format that identifies the tasks, inputs and outputs that constitute the whole operation. This representation, typically called a workflow, can be leveraged by a middleware to create a group of threads that work together to form a processing pipeline.

The use of open standards is crucial for the adoption of any new model. Thus, while the model we propose is generic and middleware independent, we created a prototype implementation that adheres to popular standards. Specifically, we tested our model on the Open Grid Services Architecture - Data Access Integration (OGSA-DAI), a widely available grid middleware that is part of the Globus toolkit. Also all operations we performed were accessing data available from services adhering to the OGC standards.

Our work is the first we are aware of that uses the atomic properties inherent in GIS features to apply the pipeline processing model. Even though pipeline frameworks have been extensively studied in e.g. RISC processors (Hennessey, 1994), little research has taken place about the application of this approach to GIS operations, along with a complete model for dependency analysis specific to this area.

The contribution of our work is twofold. Firstly, the presentation of a middleware architecture for GIS using open standards and therefore highly interoperable. Secondly, the model of a pipeline framework that can improve the performance of individual operations in common deployments scenarios, such as servers with 4 to 8 processors.

The organization of the paper is as follows. Section 2 describes the pipeline framework we propose. Section 3 describes how to implement typical GIS operations within this framework. Section 4 describes a prototype implementation and provides performance measures for a sample scenario in typical server environments. Finally, we present our conclusions in section 5.

2 MODEL DESCRIPTION

In computing, a pipeline is a chain of data processing tasks arranged so that the output of each task is the input of the next. If the data stream that flows from one task to the other is broken into independent blocks, then each task can be reused after processing a block of data.

The “tasks” can be any process that acts on an input to produce an output. This definition is abstract on purpose because it is implementation dependent. Usually the middleware will associate the name of the task with a piece of code that runs as a thread to act on the data.

The natural choice for the “blocks of data” for GIS operations are discrete geographical features represented as a collection of attributes, one of which is usually geometry. This choice is not mandatory for the application of our model. However, we chose a feature based approach because information modeling based on geographic features is the foundation for semantic and syntactic interoperability (ISO 19125-1, 2004) as expressed in the ISO 19000 series of standards and the OGC interface specifications and encodings. Consequently, there is an extensive and growing body of work; algorithms, tooling, applications, open source software, etc. based on these open interoperability standards.

To easily construct a fully parallel pipeline that processes geographical features we use diagrams that represent the dependencies among the tasks within the pipeline. These diagrams can also be represented in many alternative ways such as a series of calls to an application programming interface (API) or as an XML document. In this paper we use the term “workflow” to generically describe any representation of the data flow between tasks that is expressive enough to be used for dependency analysis.

2.1 Workflows

A key concept in our approach is the use of a middleware that uses a workflow representation to construct and connect the processing threads from the tasks. A middleware that implements this model correctly should be able to prepare an execution plan that honors the data dependencies of this whole operation. This enables any software using that middleware to specify what its actual operation is in a clear and concise way.

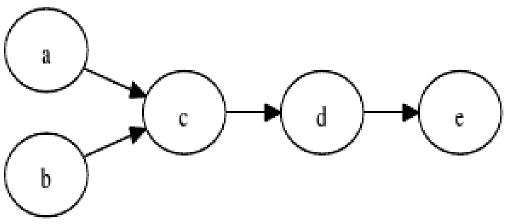

Figure 1: A simple workflow.

This representation is the first step for our dependence analysis. Figure 1 shows a workflow diagram that could represent a GIS operation with two inputs A,B, two processing steps C,D and an output delivery method E. Here tasks A and B can be executed in parallel. The aggregate output of A,B and processes C,D,E can also be executed in a pipeline. Between the processes there is usually a buffer to compensate for differences in the execution speeds of each process. In other words this workflow representation is used as a data dependence graph. It should be noted that although dependencies are usually resolved by the use of a pipeline, tasks that are completely independent like A and B can be executed purely in parallel.

There are several examples of middleware software that use workflow representations to execute an operation. Some well known ones are BPEL (Akram, et.al.,2006), Taverna and OGSA-DAI. Our implementation was based on OGSA-DAI, an open source middleware that is part of the Globus toolkit. The implementation is discussed in more detail in section 3. The rest of this section describes the whole process of dependence resolution and the metrics typically used to evaluate the efficiency of the resulting parallel operation. The analysis is middleware agnostic and is concentrated more to solving the dependencies in geospatial applications.

2.2 Dependence Resolution

Data dependencies arise when two tasks access or modify the same feature. There is extensive literature in the area of dependence resolution for compilers, as part of automatic parallelization (Bacon, et.al., 1994). We use a similar analysis to trace the dependencies that would apply to a workflow representation for geospatial operations. Assuming a task T_1 should act on a feature before a task T_2 in the sequential code there are four possibilities to consider:

Flow dependence if task T_1 modifies geographical features that T_2 reads then they cannot be executed purely in parallel. In the workflow diagram the task at the end of an arrow is dependent on the task at the beginning of that arrow. Fortunately, the pipeline method guarantees that each task will operate on that feature strictly in order. It is therefore a suitable approach if we need to parallelize an operation that has a flow dependence.

Anti-Dependence if task T_2 modifies features that T_1 reads then, just as with flow-dependence, T_1 has to operate on those specific features before T_2 . Again each feature is processed sequentially in the pipeline, even though the tasks are running in parallel and all these dependencies are honored.

Output dependence When two or more tasks modify the same feature they cannot be executed out of order. It can break a software pipeline if two or more tasks write to the same random access shared resource such as a database. This, however, would violate our proposed implementation model explained in section 2.3 where there is only one task that reads or writes to a resource and resources are not shared. This situation can still occur if someone extends a workflow-aware middleware to add more tasks and uses third party libraries that do not work on streams but act directly on databases. In general, it is a much cleaner, robust and less error-prone approach to avoid sharing resources globally among tasks.

No dependence When two tasks have no dependencies then they can run fully in parallel and no pipelining is necessary. This is the case of nodes A and B in figure 1. It should be noted that the slowest process of these tasks defines the overall throughput. More specifically, there is an obvious need for synchronization where C will block the faster task until it receives enough data.

2.3 Assumptions

We can therefore make the following assumptions for our model to ensure dependencies are resolved:

1. The whole GIS operation is constructed by joining together tasks in a workflow represented as a directed acyclic graph. It is directed to denote the flow of data in the pipeline. It should also be acyclic i.e. there is no nonempty directed path that starts and ends on the same task. There is rarely justification for having a task that post-processes features that it produces itself – a new task could be used to achieve the same result.

2. Input and output data for each task should be a buffered stream. No other resources are shared between the tasks. This is required to avoid output dependencies.

These restrictions can be beneficial even for sequential code because they encourage a clean design. If the geographical tasks are already provided by the middleware then the frontend software just needs to join them together in any way it needs. In this case, the pipeline model is used as a design pattern not only for parallelization but as a coding practice for reusable code. This is most commonly known as the "Pipes and Filters" design pattern (Bushmann, et. al., 1996).

If the tasks are not already implemented by the middleware then it should be easy to extend it to support those tasks. When creating a new task the two restrictions described above should apply to it. Extra care should be exercised with the use of third-party libraries that could prove problematic.

2.4 Reusing existing code

Any existing thread-safe libraries that can operate on streams can be used in our model. All tasks must accept and produce streams. A stream can be any reusable memory buffer that is used to access a large amount of data, block by block. In Java this can be an *InputStream* or an *OutputStream* and in C++ an *istream* or an *ostream* respectively. However, many libraries are designed to work on files or databases only. If we used a third party library that would operate on files instead of streams, then we could still wrap its functionality in a workflow task that outputs a file pointer instead of the usual stream of bytes. However, the whole pipeline model would break because the task in question should flush all the data on the disk and pass a file descriptor to the next. The benefits of parallelization would therefore be lost. There are also possible output dependencies, especially when the library is not thread-safe. Fortunately, many existing libraries don't have these limitations or provide the source code and can be trivially modified to work on memory instead of files.

2.5 Metrics

To evaluate the performance of a pipeline, metrics like *throughput* and *threading speedup* are commonly used (Liao, et.al., 2004).

The throughput of our pipeline model is the number of features that are processed in the unit of time. In a parallel pipeline this is defined as is the inverse of the maximum execution time among all tasks. So assuming t_i is the execution time for task i out of n total tasks:

$$throughput = \frac{1}{\max\limits_{1 \le i \le n} t_i}$$

and assuming that the execution time for the single and n-threaded implementation of a task is t_1 and t_n respectively, the threading speedup is defined as:

$$S = \frac{t_1}{t_n}$$

2.6 Optimizations: Node balancing

The slowest task in the workflow will define the throughput. It is therefore important to try and balance the processing time among the tasks. To make a perfectly balanced equivalent of any work flow orientation we need to calculate the mean time t spent in each task and assign it to all tasks. This will maximize the throughput if the number of tasks and the total time are constant

Proof: We know that the slowest task t_{max} multiplied to the number of tasks n will always be greater or equal to the total time for the process:

$$t_{\max} \cdot n \ge \sum_{i=1}^{n} t_i$$

Therefore:

$$t_{\max} \ge \frac{\sum_{i=1}^{n} t_i}{n} \Leftrightarrow t_{\max} \ge \bar{t}$$

The optimum t = t_{max} is achieved when all tasks have equal times since:

$$\bar{t} = \frac{\sum_{i=1}^{n} t_i}{n} = \frac{n \cdot t_{\max}}{n} = t_{\max}$$

In real applications, this ideal situation is uncommon. In practice, we should change the number of tasks in the workflow by breaking the slowest processes into two or more tasks. These would decrease $t_{\max}$ and therefore increase the throughput.

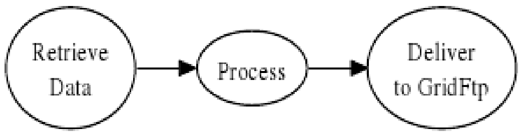

Figure 2: An inefficient pipeline.

Furthermore, we have assumed the number of processors is enough to run the whole workflow in parallel. In practice, this is not always the case and therefore the maximum speedup is limited to the total single-threaded time over the number of CPUs. That speedup is only achieved when the pipeline is perfectly balanced.

Figure 2 shows a workflow with a very simple pipeline that could be used for retrieval, processing and delivery of vector data. In this case no real benefit is achieved from parallelization. Usually the time for the data retrieval and delivery is minimal in comparison to the processing time even for remote data. Furthermore, only three CPUs can be utilized for the execution of this pipeline. The more tasks we have, the more CPUs we can take advantage of.

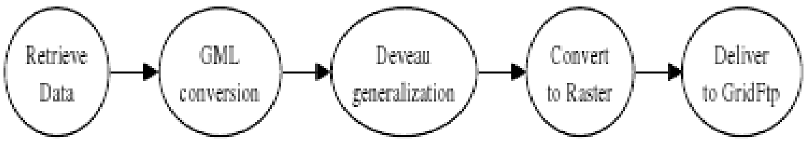

Figure 3: A more balanced pipeline.

As an example, lets assume that the processing step in figure 2 includes i) the conversion of features received to Geography Markup Language (GML) (ISO 19136, 2005), ii) geometry generalization using the Deveau algorithm (Deveau, 1985) and iii) conversion to a raster format. A workflow diagram like the one in figure 3 would significantly improve the speedup of the operation.

Workflow approaches can also enable other forms of optimizations. Visualizing the whole process can enable us to find ways of decreasing the processing time by changing the position of the tasks. For example, the task of generalization reduces the size of the data by applying the Deveau generalization algorithm to the geometries of the features. If we generalize the geometries before converting to GML, we reduce the processing time for the rest of the tasks in the pipeline because of the smaller data size.

So far, we have described our model generically and independently of the middleware used. The following section applies the model as presented to perform real geospatial operations with a host of potential applications.

3 IMPLEMENTING GIS OPERATIONS

We have applied the model as described above to implement two draft standards that are currently under evaluation within the OGC Interoperability Programme. The OGC is an international standards defining organization that develops standards to serve the needs for interoperable geospatial technology. The draft standards we used are the Geolinked Data Access Service (GDAS) and GeoLinking Service (GLS) as described below:

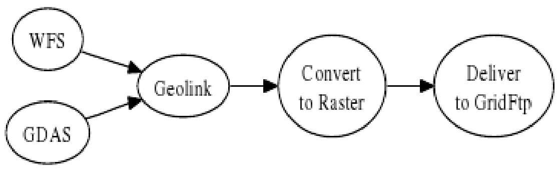

Figure 4: Geolinking.

The **GDAS** provides a way to publish and access data related to spatial features (e.g. Census data). One of the strengths of GDAS is that it can expose data from any data source including non-GIS databases in a standardized way so that it can be manipulated with the aid of a GLS.

The **GLS** takes attribute data received from a GDAS server and joins them to a geospatial dataset. The standard does not currently specify what the exact output is. Usually, it is a thematic map which emphasizes spatial variation according to an attribute. It could also be in a vector format, such as GML, with all the attributes received from the GDAS server added to the existing attributes of the geographic features.

One possible way to implement the functionality of the GLS service is shown in figure 4. The representation of this operation is identical to the workflow in figure 3 that was analyzed previously in the paper.

To execute the process described in this workflow we used OGSA-DAI - well known Grid middleware included in the Globus toolkit. OGSA-DAI provides a data service framework for accessing and integrating data resources on to the Grid. The data resources are typically files, relational and XML databases. An XML representation called the **perform document** is used to describe the workflow. It can execute many predefined tasks, called **activities**, that are joined together according to the perform document to execute an operation.

We modified OGSA-DAI version 3.0 and added new geospatial activities, including the ability to use any OGC compliant Web Feature Service (WFS) as a data resource. Generally this is time consuming, but the intention is that, in careful design of the activities, they will be reusable in multiple applications.

Figure 5 is an example perform document that corresponds to our implementation. The task *wfsGetFeature* receives vector data as GML from a WFS and *gdasGetData* receives attributes from a GDAS compliant server. Both of these tasks are executed in parallel. The *geoLink* activity joins the features with the attribute data. Next comes the *convertRaster* activity that converts the input data to a raster image and finally the *deliverToGFTP* activity delivers it to a remote server.

```
<perform >
 <flow name="inputFlow">
  <wfsGetFeature name="getFeature">
    <classname value="Leeds_2001"/>
    <result name="featureOutput"/>
  </wfsGetFeature>
  <gdasGetData name="gdasGetData">
    <framework value="Leeds_2001" />
    <dataset value="2001CAS" />
    <attributes value="4,5,7-15" />
    <result name="dataOutput"/>
  </gdasGetData>
 </flow>
 <geoLink name="geoLink">
  <wfsinput from="featureOutput" />
  <gdasinput from="dataOutput" />
  <geolinkage value="ons_label" />
  <result name="geolinkOutput" />
 </geoLink>
 <convertRaster name="convert">
  <gmlInput from="geolinkOutput" />
  <format value="png" />
  <result name="convertOutput" />
 </convertRaster>
 <deliverToGFTP name="deliverFeature">
  <fromLocal from="convertOutput"/>
  <toGFTP host="somehost.org" port="2811"
    file="path" append="false"/
 </deliverToGFTP>
</perform>
```

Figure 5: A perform document.

4 ANALYSIS OF RESULTS

This section describes the expected and actual measurement of the pipeline performance for the operations analyzed in section 3. To assess the speedup offered by the pipeline we executed the GLS operation described in figure 4 for an increasing number of hardware threads. We ran a single GLS request for 2440 variable sized features on the threaded application and measured the time and speedup.

Our primary goal is to assess the applicability of parallelization methods for common setups. We want to be able to accelerate single operations on machines with around 4 to 16 processing units as is most frequently found. Indeed, to make the execution of a workflow representation fully parallelizable we only need a number of processing units equal to the number of tasks in the OGSA-DAI perform document.

The machine we performed the tests on was a Dell Poweredge 6800 with 4GB DDR-RAM and two dual core Intel Xeon processors running at 2800 Ghz. Each core has two hardware threads totaling 8 hardware threads. The operating system is a GNU/Linux system with kernel version 2.6.9. Our middleware was written in Java and executes as described in the workflow diagram in figure 1.

Since each hardware thread is sharing the resources of a single processor core, each thread has some fraction of the core's overall performance. Such configurations are common for servers delivering geospatial services and it is helpful to see whether we can apply our model to take advantage a total of 5 nodes, which is the number of tasks in our workflow.

To evaluate the results we measured the speedup for 1 to 8 nodes, which are really hardware threads. We had to use the CPU affinity system calls of the Linux kernel that can force a process to be bound to specific nodes. Our measures include absolute execution times for comparison. A more useful metric is in figure 7 that shows the speedup of the parallel process. It indicates how the operation scales with the addition of more processing units.

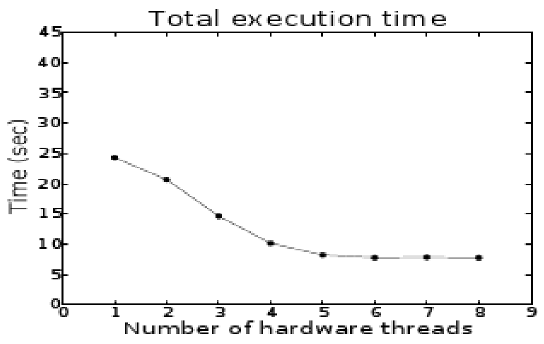

Figure 6: Execution time per number of nodes.

Obviously, since there are 5 tasks in total, there was no need to utilize all 8 nodes. As it turns out the addition of more nodes does not seem to affect performance. The speedup we got was around 3 which is less than the optimum 5. The reason is that the pipeline was not perfectly balanced. Also, the geographical features that enter and leave the tasks of the pipeline have a variable size so it is almost impossible to achieve perfect balancing.

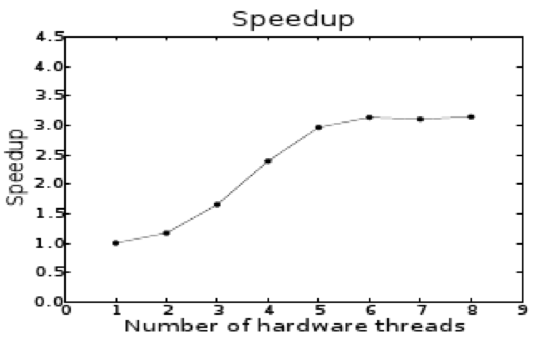

Figure 7: The speedup per number of nodes.

The main purpose of our approach was to increase the uptake of parallelization for existing sequential approaches with minor changes. It is impressive that one can achieve significant benefits for even small requests on common setups just by using a different software engineering pattern.

5 CONCLUSIONS

This paper proposed an approach to increase the availability of parallel GIS. We described a pipeline based approach specifically tailored to geospatial operations to solve the unnecessary dependencies in most common GIS operations.

This approach can achieve significant improvements, especially in small to medium scale servers that are in common use today for geographical services. The current processor trend towards multi core designs adds up to the value of such approaches.

An attractive property of this model is that it while it accelerates the execution of an operation it also helps in speeding up the development time. The workflow based development adheres to well-known approaches in software engineering like the separation of concerns (Anderesen, 1992) and the "pipes and filters" design pattern (Wiley, 1996). Different logical operations are separated into tasks. Any number of tasks can be joined together in different combinations to perform different operations without having to write new code.

Our test implementation promoted an approach that is based on standards commonly used by both the geospatial and the Grid communities. We extended an existing middleware to support data input and deliver from OGC specified services like WFS and WMS servers. As GLS is currently a profile of the OGC Web Processing Specification (WPS), what we have effectively described in this paper is a significant step towards the creation of a powerful and flexible toolkit for the creation of OGC compliant WPS in a Grid environment.

There is certainly scope for future improvements. Since this paper touched multiple subjects relating to pipelining, GIS and software engineering we decided the emphasis should be on the theoretical model. The reason was that we could not find any existing research that analyzed the dependencies of geospatial operations with the goal of parallelizing them and we thus concentrated on that specific subject. To increase flexibility and uptake we would encourage the creation of more pre-built tasks for geospatial operations. Additional benchmarking needs to be carried out for large scale operations. It should prove especially interesting to test the model for very large pipelines on large-scale servers.

BIBLIOGRAPHY

Frank Bushman, Regine Meunier, Hans Rohnert, Peter Sommerland and Michael Stal, Pattern Oriented Software Architecture, Volume 1, A System of Patterns, John Wiley & Sons,August 1996

E. P. Andersen and T. Reenskaug. System design by composing structures of interacting objects. In ECOOP '92: Proceedings of the European Conference on Object-Oriented Programming, pages 133–152, London, UK, 1992. Springer-Verlag.

D. Bacon, S. Graham, and O. Sharp. Compiler transformations for high performance computing. In ACM Computing Surveys, pages 345–420, 1994.

T. J. Deveau. Reducing the number of points in a plane curve representation. In Proc. of Auto-Carto 7 (Seventh Intl. Symp. on Computer-Assisted Cartography), pages 152–160, Baltimore, MD, 1985. American Congress of Surveying and Mapping.

A. Akram, R. Allan, and D. Meredith. Evaluation of BPEL for Scientific Workflows. In 6th IEEE International Symposium on Cluster Computing and the Grid, pages 269–274, 2006.

International Symposium on Cluster Computing and the Grid, CCGrid 2006 (CCGrid 2006), Singapore, May 2006.

Geographic information – Simple feature access – Part 1: Common architecture, ISO 19125-1, 2004.

W.-K. Liao, A. Choudhary, D. Weiner, and P. Varshney. Performance evaluation of a parallel pipeline computational model for space-time adaptive processing. J. Supercomput., 31(2):137–160, 2004.

Geographic information – Geography Markup Language, ISO 19136, 2005.

Hennessey, John L. and Patterson, David A. Computer Organization and Design: The Hardware/Software Interface; 2nd edition, Morgan Kaufmann, San Mateo, pp. 125-214, 1994.