A Taxonomy of Parallel Vector Spatial Analysis Algorithms

by

Jason Laura

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved September 2015 by the
Graduate Supervisory Committee:

Sergio J. Rey, Chair
Luc Anselin
Shaowen Wang
WenWen Li

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

Nearly 25 years ago, parallel computing techniques were first applied to vector spatial analysis methods. This initial research was driven by the desire to reduce computing times in order to support scaling to larger problem sets. Since this initial work, rapid technological advancement has driven the availability of High Performance Computing (HPC) resources, in the form of multi-core desktop computers, distributed geographic information processing systems, e.g. computational grids, and single site HPC clusters. In step with increases in computational resources, significant advancement in the capabilities to capture and store large quantities of spatially enabled data have been realized. A key component to utilizing vast data quantities in HPC environments, scalable algorithms, have failed to keep pace. The National Science Foundation has identified the lack of scalable algorithms in codified frameworks as an essential research product. Fulfillment of this goal is challenging given the lack of a codified theoretical framework mapping atomic numeric operations from the spatial analysis stack to parallel programming paradigms, the diversity in vernacular utilized by research groups, the propensity for implementations to tightly couple to underlying hardware, and the general difficulty in realizing scalable parallel algorithms. This dissertation develops a taxonomy of parallel vector spatial analysis algorithms with classification being defined by root mathematical operation and communication pattern, a computational dwarf. Six computational dwarfs are identified, three being drawn directly from an existing parallel computing taxonomy and three being created to capture characteristics unique to spatial analysis algorithms. The taxonomy provides a high-level classification decoupled from low-level implementation details such as hardware, communication protocols, implementation language, decomposition method, or file input and output. By taking a high-level approach implementation specifics are broadly proposed, breadth of coverage is achieved, and extensibility is ensured. The taxonomy is both informed and informed by five case studies implemented across multiple, divergent hardware environments. A major contribution of this dissertation is a theoretical framework to support the future development of concrete parallel vector spatial analysis frameworks through the identification of computational dwarfs and, by extension, successful implementation strategies.

I dedicate this dissertation to my wife Jody and my children Abigail and Thomas, who keep me grounded and focused on the important things in life.

ACKNOWLEDGEMENTS

It is challenging to articulate the debt of gratitude I owe Serge Rey. Without Prof. Rey's guidance and trust, this journey would have been far more challenging and far less rewarding. Along the path of graduate study, while I focused intently on the trees, Prof. Rey always reminder that I was standing in a forest. When I struggled to calm my nerves for a proposal defense, Prof. Rey reminder me to enjoy the experience because 'you only get to do it once'. Prof. Rey has an uncanny ability to aim me in the right direction and let me explore. I fear that only two guys from Jersey could really appreciate the amount of trust Prof. Rey showed me, from bringing me on to the CyberGIS project, to letting my interests wander between different algorithms and implementations. Through the entire process, I have appreciated Prof. Rey's trust in me; it has provided strength and confidence. It is with immense gratitude that I write these acknowledgements.

I appreciate all of the time that WenWen Li spent, both mentoring me and helping spur my interest in spatial regionalization algorithms. Prof. Li's feedback has been invaluable in strengthening this dissertation and integrating more firmly within the GIScience domain. This work would not have been possible with the support of Luc Anselin. I am thankful not only for Prof. Anselin's support of my research, but also for his uncanny ability to strike immediately to the core of a topic. Prof. Anselin's questions pushed me from my comfort zone and continue to drive deeper, critical thinking. I would be remiss not to thank Shaowen Wang for his steadfast dedication to promoting CyberGIS and providing the opportunity for this work, both theoretical and applied, to be further integrated into the CyberGIS community.

I would also like to thank the other members of the GeoDa Center. In particular, Rob Pahle, not only for supporting my work there, but also for the remarkable collaborative efforts we continue to share working on cREST, and Julia Koschinsky for being an amazing spokeswoman and so remarkably welcoming into this research group. Finally, I would like to thank the PySAL development team at large. This group of developers has helped push my coding to the next level by always offering valuable feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Nearly 25 years ago, Griffith (1990) illustrated the application of parallel computing methods to vector spatial analysis tasks and called upon quantitative geographers to leverage all available computing resources in their work. That work heralded the beginning of broad efforts to improve the performance of spatial analysis techniques through algorithmic advances. Thirteen years later, Clematis *et al.* (2003), suggested that the promise of universally accessible parallel GISystems were not yet achieved and cited three major roadblocks requiring reassessment: (1) accessibility to high-performance Computing (HPC) resources, (2) a lack of parallel spatial analysis algorithms, and (3) the significant learning curve for the utilization of high-performance systems. By in large, road blocks two and three are still significant hurdles in the development and utilization of parallel GIS. This work targets item two and sets out to develop a taxonomic classification of methods of parallel spatial algorithm implementation within the larger framework of methods of algorithm parallelization (Asanovic *et al.*, 2006).

The development and application of parallel, vector, spatial analysis techniques can be divided into three temporal clusters. These clusters can be defined topically, beginning with dissertation research focusing on parallel computational geometry, shifting to parallel spatial analysis methods, and presently exploring spatial analysis methods in HPC environments. The genesis of parallel spatial analysis, in the late 1970s and early 1980s, exists at the intersection of what was HPC and computational geometry. Initial foundational efforts focused on parallel graph operations (Arjomandi, 1975; Hirschberg, 1976; Eckstein, 1977; Savage, 1978), polygon intersection, convex hull, line sweep, and nearest neighbor search (Chow, 1980; Aggarwal *et al.*, 1988; Stojmenovic and Evans, 1987; Atallah and Goodrich, 1984, 1985; Blelloch and Little, 1988; Akman *et al.*, 1989). These are core processing functions required by many more complex spatial analysis operations. These early efforts provide insight into spatial domain decomposition techniques to support concurrent processing and the utilization of tree data structures to facilitate load balancing. Each of these implementations is marked by the small data sizes at which problems were considered intractable and the tight coupling between hardware and software.

The work of Griffith (1990) heralds the application of parallel computing methods to the spatial analysis domain and the start of the second temporal cluster. Efforts pioneered by earlier computational geometry works are leveraged within the spatial analysis domain (Waugh and Hopkins, 1992; Armstrong and Densham, 1992) and the first implementations of parallel spatial analysis algorithms developed (Armstrong *et al.*, 1993; Armstrong and Marciano, 1995; Clematis *et al.*, 1996; Ding and Densham, 1996). A critical mass of interest in parallel spatial algorithms continued to grow through the 1990s as computational intensity was identified as one limiting factor to the broad application of spatial analysis to larger problem sets (Armstrong and Densham, 1992). Presumably, a critical mass of research effort was the impetus for the publication of Healey's Parallel Processing Algorithms for GIS (Healey *et al.*, 1998), a collection of articles focusing on parallel spatial analysis.Armstrong (2000) identifies a key hardware paradigm shift, the transition from shared memory to distributed memory systems as computer hardware vendors sought to keep pace with the requirements of Moore's Law. This transition, he suggests, will require the reimplementation of many algorithms dependent upon shared memory. This shift in hardware and resultant shift in research interest was further driven by the sale of multi-core consumer grade hardware. The end of the second temporal cluster can be identified with the editorial by Clematis *et al.* (2003) where the development and more importantly utilization of parallel GIS is lamented as an unfulfilled opportunity; the computational hurdle which parallelization sought to overcome is a mountain which is still being climbed.

The so-called Atkins Report (Atkins, 2003) identified a framework within which HPC is married to cross domain and institution cooperation in order to broadly drive research efforts. Within the spatial domain, Geospatial CyberInfrastructure (GCI), CyberGIS, and Spatial CyberInfrastructure (Yang *et al.*, 2010; Wang, 2010; Wright and Wang, 2011), respectively, identify different facets of principals articulated in the Atkins Report principals. A key component of all three is the development and deployment of parallel spatial analysis algorithms in HPC environments. Much of the effort in the development and codification of spatial analysis algorithms, within the CI domain, from 2003 onwards supports this goal with a focus on deployment into high-performance, distributed memory systems and not consumer grade desktop computers. Armstrong *et al.* (2005) provides the first look at a unified, spatial implementation of CI within a computational grid. Considerable effort has also been applied leveraging the spatial nature of data and joining that with a computational domain representation in order to drive efficient decomposition (Wang and Armstrong, 2005; Wang *et al.*, 2005, 2008). Parallel spatial algorithm implementations have focused largely on explicitly decomposable components with limited aggregation requirements, e.g. Inverse Distance Weighted interpolation or embarrassingly parallel implementations. This focus has left a significant portion of the vector spatial analysis algorithm library ripe for future research. It is within the context of the so-called 'middleware' layer, the glue which supports collaborative decision making using high-performance computing resources, that this work is cited.

1.1 Research Objectives

Increasing data sizes, a general move towards collaborative multi-disciplinary research teams, and complex process modeling are some of the major drivers behind the adoption of both Cyberinfrastructure and the sub-discipline, Geospatial Cyberinfrastructure. This work is motivated by the need for algorithmic development to support ever increasing data sizes and complex process models. I note that increases in data sizes are observable in both the physical and social science domains, with the latter being enabled by the end-to-end creation of purely digital artifacts. A key component of CI is high performance algorithms to support collaborative research efforts and this work targets that domain. The process of algorithm optimization and parallelizations is non-trivial (McKenney, 2013). Unfortunately, this difficulty has resulted in many one off implementations that frequently suffer from cookbook style implementation specifications with tight hardware coupling and divergent vernacular. A core set of atomic computational operations have not been mapped to the spatial analysis domain. I suggest that this significantly reduces the portability of the root method to new, potentially similar implementations, or new hardware environments; a theoretical framework is lacking. Difficulties in portability and commonality in vernacular are not constrained to the spatial analysis domain. National Science Foundation (NSF) report 12-113 calls for 'High-level abstractions and frameworks that promote code reuse and sharing, model extensibility and interoperability, and simplify domain specific programming while achieving high performance;' (National Science Foundation, 2012). These software frameworks are designed to significantly reduce the burden of implementations, providing a library of proven methods.

I argue that the NSF call can be answered in two ways. First, ad-hoc implementation across disparate research groups, can continue with the goal of developing an integrated approach at some future point. Alternatively, this work seeks to provide a theoretical framework, in the form of a taxonomy, that supports the identification of core computational and communicational primitives that can be mapped across the spatial analysis stack, inform implementation, and provide the necessary theoretical framework to support the preceding NSF call. The development of this taxonomy requires identification of computational similarities across the spatial analysis stack. These efforts have been previously undertaken and realized as taxonomies. Next, this work seeks to map these similarities to parallel programming paradigms without high classification dimensionality, tight hardware coupling, or low-level implementation directives. That is, avoiding a devolution from a means of classification into a one-to-one algorithm to implementation specification mapping. Finally, this work seeks to leverage existing taxonomies, not just from the spatial analysis domain, but also from the operations research and computer science domains.

1.2 Parallel Computing

To define parallel computing, it is essential to first understand the process by which a single Central Processing Unit (CPU) performs some unit of computation. Serial computation, performed by a single CPU, decomposes a set of instructions into a streaming, sequentially processed workflow, composed of atomic compute operations and data. The limit of performance is the speed with which the individual data objects can be retrieved, ordered with the compute instructions, and processed by the CPU. The performance of this process is partially, and indirectly, governed by Moore's Law, which states that the number of transistors which can be placed on a CPU is expected to double every 18 to 24 months (Moore, 1965). By extension, the performance of a single CPU is expected to roughly double at the same pace (Bell and Gray, 1997). Two performance gains are attainable while maintaining a constant CPU speed: vectorization and algorithmic optimization. Vectorization is made possible by leveraging vector processors, now largely referred to as SIMD (Same Instruction Multiple Data) processors, that improve sequential performance by processing contiguous memory vectors using the same instruction. See Chapter 2 for a full explanation of vectorized computation. While the benefits of vectorization can be marked, serial performance is limited by the speed with which a single CPU can access data. [1] In addition to improvements in hardware performance, algorithmic optimization can provide significant performance improvements within a serial environment. (Bell and Gray, 1997) suggests that algorithm performance improvements also appear to conform to Moore's Law, suggesting that a fixed data size problem can anticipate exponential performance improvement every two years.

Unfortunately, performance gains from improved CPU speeds, leveraging of vectorized processing, and algorithmic optimization are not sufficient to support the universal application of serial spatial analysis algorithms for three reasons. First, data sizes continue to grow at a higher rate than performance gains due to improved collection efforts, increases in sensor resolutions, and increases in derived analytical products. Second, the computational complexity of spatial analysis algorithms continues to grow, offsetting benefits from improved serial implementations. Finally, the sustainability of Moore's Law within the manufacturing domain is questionable with critics suggesting that the end of the law is predicted at decadal intervals; this prediction has never come to fruition.

In order to continue providing performance improvement while mitigating energy and cooling requirements, CPU manufacturers have largely focused on multi-core architectures. These architectures provide the capability to perform many concurrent

[1] While this holds for algorithms with sufficient computational complexity, the actual performance of many algorithms is no longer constrained by the speed of the CPU, but by the speed at which data can be moved to the CPU, the starving CPU problem. See Alted:2010cn.

computations across multiple CPUs. In step with the deployment of multi-core architectures, the rise of the Internet has driven interest in distributed systems which require distributed, parallel code execution. Within the context of a parallel algorithm, each processing core applies some atomic compute operation to some data stream, exactly as described above. In contrast to a serial algorithm, the data and/or operations must have been decomposed in such a way that the CPUs can concurrently work. Significant performance improvements can be realized by leveraging concurrent computation. The limitation of this processing model is not then, the amount of CPU power that can be requisitioned for an analytical task, but the ability for an algorithm to efficiently utilize the available computational resources.

1.2.1 Amdahl's Law, Gustafson's Law, and Efficiency

The goal of any parallelization effort is performance improvement. This improvement exists through the maximization of the quantity of concurrent processing or, inversely, the minimization of serial processing. Two oft-cited laws model the theoretical performance gains through parallelization and can help drive the decision to parallelize an algorithm. Amdahl's law states that the total expected speedup is a function of the ratio of serial to parallel processing time and the total number of processing cores (Amdahl, 1967; Hill and Marty, 2008). This assumes that all processing cores are symmetric and is presented, as formulated by Gustafson (1988) as

$$speedup = (s + p)/(s + p/N), \tag{1.1}$$

where s is the amount of time required for serial processing, p is the amount of time required for parallel processing, and N is the number of processing cores. Arbitrarily setting $s + p = 1$ or assuming that $s + p$ is the cumulative percentage of total computation, one can see that the maximization of speedup requires the minimization of s. This requires minimization of serial Input / Output (I/O) operations, synchronous inter-core communication which requires blocking, e.g. bulk synchronization parallelization paradigms, and inter-core data dependencies which could require wait periods. Gustafson (1988) shows that even a small percentage of serial computation, on the order of one to four percent, can significantly degrade performance. Assuming 512 processing cores, Figure 1.1Comparison of Estimated Performance Using Amdahl's and Gustafson's Lawsfigure.1.1 illustrates, as a solid line, the exponential drop in speedup with an increase in s. This initially, suggests that only those algorithms most amenable to parallelization should be targeted.

Gustafson (1988) asserts that Amdahl's Law fails to account for scaling of p as a function of the total data size. In practice, N is dependent on the total problem size, and as N increases, so does p. Serial spin-up time, s, is largely invariant to total problem size, suggesting that algebraic manipulation of Amdahl's Law is warranted. This yields:

$$scaledSpeedup = s + p \cdot N, \tag{1.2}$$

Scaled speedup, shown as a dashed line in Figure 1.1Comparison of Estimated Performance Using Amdahl's and Gustafson's Lawsfigure.1.1, provides a significantly improved view of the performance gains attainable through parallelization. At 4% serial processing and 512 processing cores, speedup is still over 491 times.

Whether assessing performance using Amdahl's or Gustafson's Law Hill and Marty (2008) suggest that minimization of s is not always the paramount priority as multicore asynchronously computing processing cores can provide better global speedups even when s is locally high, e.g. an excess computation programming model. That is, an increase in the serial compute time locally is acceptable, and possibly desirable, if

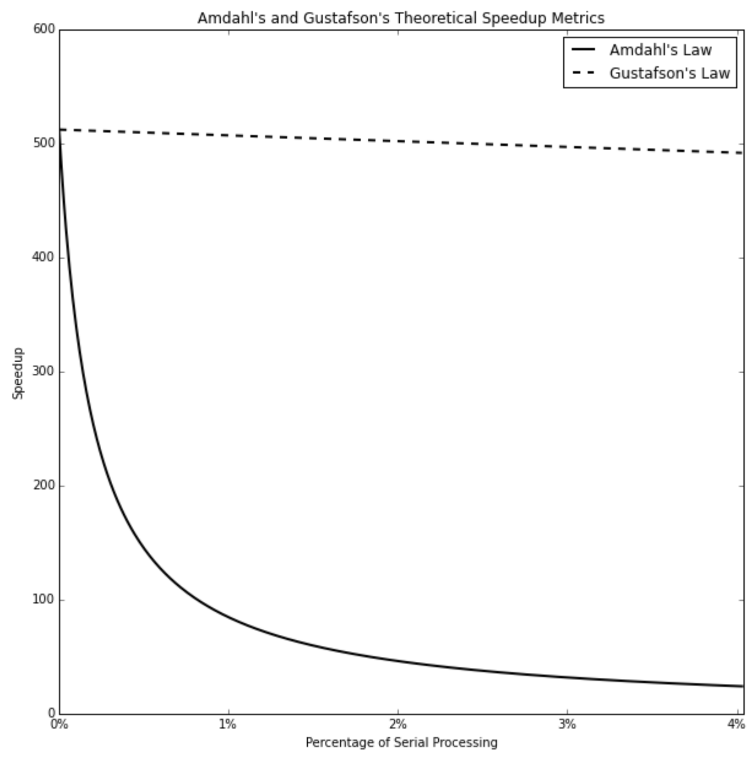


Figure 1.1: Comparison of Estimated Performance Using Amdahl's and Gustafson's Laws.

the global result is an increase in p. This in turn suggests that a singular focus on the decomposition of existing serial implementations, without algorithmic modification, is ill-advised and that excess sequential processing may offer global improvements.

Finally, efficiency provides an additional view of how well utilized all processing cores are during parallel computation. Grama *et al.* (2003) formulate efficiency as:

$$E = S/N \tag{1.3}$$

where S is the speedup as computed by Amdahl's Law and N is the number of processing cores. In an idealized (or serial) system, $E = 1$. In a highly parallel environment E trending towards 1 is highly desirable, but difficult to achieve.

These concepts, drawn directly from the Computer Science domain, are immediately applicable to spatial analysis algorithms as one can estimate the ratio of serial to parallel computation such that anticipated performance can be derived.

1.2.2 Parallel Programming Models

The implementation of a parallel algorithm requires identification of the processing bottlenecks. These bottlenecks provide a set of constraints upon the amount of communication required between processing cores and the methods of decomposition appropriate for concurrent processing. The selection of implementation methods suitable for performant, scalable implementations are then also a function of the hardware to be used. This section describes the interaction of these constraints upon a generalized parallel algorithm. This idealized algorithm can be considered a proxy for a spatial algorithm requiring parallelization. These concepts are more fully explored in Chapter 2.

Communication

Inter-core communication describes data or message transfers which occur during parallel processing and granularity describes the frequency and size of messages communicated between processing cores. I identify three communication models which exist along a spectrum: coarse-grained, fine-grained, and embarrassingly parallel. Coarse-grained communication indicates that large quantities of data are being infrequently sent and received. In general, coarse-grained granularity is more often employed due to the cost imposed by synchronous communication (and the complexity introduced by asynchronous communication). An implementation using fine-grained communication models communicate small quantities of data with high frequency. This style of implementation is utilized most frequently when more complex load-balancing is required or highly decomposed representations are possible. Finally, embarrassingly parallel models require no inter-core communication with a single bulk synchronization phase occuring at the conclusion of processing.

Decomposition

Decomposition can be either data parallel, where a data set is subset using some strategy and an identical algorithm is applied to each subset of the data, or task parallel, where a series of tasks can be decomposed for concurrent execution. Within the spatial analysis domain, the vast majority of parallel implementations leverage a data parallel model. Data parallel model decomposition can be spatial or aspatial. Spatial decomposition commonly leverages regular gridded decomposition methods, more complex tessellations, or adaptive decomposition, e.g. quad-trees (Akman *et al.*, 1989; Armstrong and Densham, 1992; Cramer and Armstrong, 1999; Wang and Armstrong, 2003). Both Akman *et al.* (1989) and Waugh (1986) highlight the potential costs associated with the application of complex decomposition methods, with the former suggesting that an order of magnitude difference in the spatial density of data points is required before gridded decomposition fails to provide adequate load balancing and the latter generally suggesting that elegant quad tree decompositions should not be universally utilized. These ideas are explored in Chapter 4. Wang and Armstrong (2005) extend the concept of spatial decomposition to account for a computational domain which describes the aggregated memory, I/O, and compute costs. Using this more complex representation of the surface, more efficient decomposition, and by extension better performance, results are reported. In contrast to spatial decomposition, aspatial decomposition methods utilize some other means of data decomposition. For example, Monte Carlo simulation, required by many algorithms across the spatial analysis stack, runs p simulations. Aspatial, data decomposition can take the form of running $\frac{p}{n}$, where n is the number of processing cores, simulations and aggregating the necessary results. In these instances, the spatial information intrinsic in the data does not offer a means for decomposition. Finally, while not largely utilized for spatial analysis algorithms, one potential task level parallelization can be identified utilizing methods of spatial regression. Given a set of analytical techniques and some metric to assess the suitability of a model, all models could be concurrently processed and the ideal model selected.

1.2.3 Parallel Hardware Architectures

As eluded to above, the parallel programming model selected is not only a function of granularity and decomposition, but also of the hardware on which the model will run. Three prominent architectures exist to support parallel computing: shared memory, distributed memory, and hybrid compute environments. Shared memory systems are characterized by a single, globally accessible memory space from and into which all compute units can access data, a Symmetric Multi Processing (SMP) system. Management of the shared memory space, through the use of locks, semaphores, or an embarrassingly parallel decomposition, are essential to avoid concurrent write operations and race conditions. The potential additional complexity to manage a shared memory space can be mitigated by the performance gains achievable through asynchronous I/O operations. This access is significantly faster than network communication. Two commonly leveraged SMP environments are the desktop computer and Graphics Processing Units (GPUs). The former are characterized by the commonality in development to a traditional processes, while the latter required custom libraries designed to support general computing applications on a GPU. The OpenMP (2013) library provides a framework for non-GPU based development in an SMP environment and the CUDA library supports GPU development. Distributed memory systems are composed of two or more discrete memory blocks accessible by some subset of the computing cores. Communication and data management is performed through message passing, with the Message Passing Interface communications protocol being one of the most widely used systems (Forum, 1994). The potential performance degradation introduced by higher communication overhead is potential offset by an increased ability to scale the number of processors with the dataset. HPC clusters and cloud computing resources are two examples of distributed memory systems. Finally, hybrid systems seek to leverage the benefits achievable through shared and distributed memory systems. In general, distributed memory systems can be leveraged as a hybrid environment, assuming that each compute node is composed of one or more CPUs. In this case, the developer manages the interface between shared memory and distributed memory portions of the parallel program.

This work focuses on the development and classification of algorithms in SMP and HPC environments, composed of many individual SMP systems, as these deployment architectures are most often utilized for spatial analysis.

1.3 Organization

In Chapter 2 I propose a taxonomy of spatial analysis methods developed at the intersection of the GIScience, Computer Science, and operations research domains. That chapter introduces the idea of computational dwarfs and leverages them as the primary classification mechanism. It is within the context of this taxonomy that the remaining chapters are framed. Chapter 3 describes an implementation of the Fisher-Jenks optimal choropleth map classification algorithm in a Symmetric Multi-Processing (SMP) environment and describes implementation challenges working with algorithms characterized by Dense Linear Algebra operations. Chapter 4 describes Geometrically and Topologically classified problems through the implementation of

Nearest Neighbor Search (NNS) and polygon adjacency algorithms. Chapter 5 and Chapter 6 focus on spatial regionalization algorithms leveraging two distinct implementation methods, MapReduce and Exploratory. Finally, Chapter 7 summarizes this dissertation and offers avenues for future research.

Chapter 2

TAXONOMY

2.1 Introduction

Wang *et al.* (2013) suggest that monolithic Geographic Information Systems (GIS) are increasingly unable to support necessary analytical, modeling, and visualization operations. This is a product of ever increasing data size, depth, and analytical requirements. Increases in data size are attributable to increased data capture initiatives, increases in sensor resolution (e.g. spatial, temporal, and spectral), and increasingly complex model outputs with high granularity (Yang *et al.*, 2008; Yang and Raskin, 2009; Yang *et al.*, 2010). The digital creation and capture of social science data (e.g. individual level social media or transport data), coupled with ever increasing temporal collection resolutions contributes directly to increased data depth (Manovich, 2012; Sui and Goodchild, 2011; Burgess and Bruns, 2012; Goodchild, 2007). Finally, increasingly complex process models, coupled with larger data sizes render non-distributed Spatial Analysis and Modeling (SAM) computationally intractable. For example, Anselin and Rey (2012) identify five computationally expensive algorithmic tasks to support spatial econometrics.[1] Tang *et al.* (2011), within the physical sciences domain, identifies similar intractability due to serial computational methods in the context of land use change analysis. Participation in the big data / big process domain requires an integrated, spatially enabled, system to support large-scale data storage, analysis, and synthesis.

[1] These are: the computation of a spatial weights object, computation of the log Jacobian (requiring computation of the determinant of a potentially large matrix), large matrix inversion, constrained numerical optimization, and Monte Carlo simulation.

Highly scalable algorithms are a small, but essential component of an integrated approach to tackling these big data / big process challenges. This is not unique to the GIScience domain and I look to the seminal, cross domain, Atkins (2003) report that describes Cyber Infrastructure (CI) as an integrated system capable of addressing the above limitations within a broader, general scientific research context. Stewart *et al.* (2010), citing the Pervasive Technology Institute (2007) provides the following definition of Cyber Infrastructure.

> Cyberinfrastructure consists of computing systems, data storage systems, advanced instruments and data repositories, visualization environments, and people, all linked together by software and high-performance networks to improve research productivity and enable breakthroughs not otherwise possible.

Wang *et al.* (2013) describe Cyberinfrastructure based GIS (CyberGIS) as a more domain specific solution which is a 'fundamentally new GIS modality comprising a seamless integration of CI, GIS, and SAM capabilities'. Envisioned as a trinity of components, GCI is composed of high-performance Computing (HPC) infrastructure that provides the hardware and storage required to support a middleware layer composed of component based, highly scalable algorithms. These lower, infrastructure layers provide the functionality required to facilitate collaborative, cross-domain research. The development, funding, and utilization of CI (and by extension GCI) is in and of itself a large research project. It is within the context of CI that I seek to motivate this work.

In motivating this work I must address two questions. First, lacking a large body of existing, intractable science questions leveraging spatial analysis techniques, is the development of high-performance spatial analysis algorithms a technical exercise in search of a purpose? I answer this by framing the argument similarly to how investment in infrastructure is being managed. Second, assuming algorithmic development is warranted, what theoretical scaffolding is necessary to reduce the complexity of parallel spatial algorithm development and support breadth of implementation coverage across the spatial analysis stack? In answering this question I propose the development of a taxonomy of spatial analysis algorithms. Each question is addressed in turn below.

2.1.1 Purpose of Parallel Spatial Algorithm Development

Kelbert (2014) describes a chicken and egg style argument focused on the development and implementation of funding models for CI projects. The crux of the issue is that infrastructure, e.g. hardware and software, are expensive to develop. Given a large quantity of funding to support infrastructure development, is the underlying research agenda not unduly biased towards science which makes use of said infrastructure? By extension, would it not be prudent to provide a minimum of funding for infrastructure, such that the science requiring it was supported, but not so much funding that the science research agenda became dominated by CI. Within the context of this argument Kelbert (2014), suggests that a balance must be struck by identifying both the problems to be solved and the underlying infrastructure required to support these. This balance must be agile, and broad enough in scope that a generalized set of functionality is provided to support community engagement and uptake. This final requirement is in line with the community engagement challenges identified by Anselin and Rey (2012) in a spatial econometrics context. The success or failure of a broad, agile approach can be measured by the amount of buy-in from the community.

Using Kelbert's argument as an analog, it is possible to shift from CI driven research and CI infrastructure funding, an integrated view, to the development of middleware algorithms and the their application within a CI environment. Like the investment in infrastructure, a key question to ask is whether research targeted at the development of scalable algorithms without immediate, broad application is a prudent course of action. Are these algorithms in search of a purpose? I argue that targeted algorithm development with broad application to drive taxonomic classification provides the same benefits as broadly scoped functionality within an integrated CI.

It is essential that any effort to break this causality dilemma focus on high-level classification of functionality usable in the iterative development, deployment, and utilization loop that embodies CI research. Asanovic *et al.* (2006), in describing the impetus behind the collaborative development of a classification scheme for highly scalable, parallel algorithms states that, '[t]he hypothesis is not that traditional scientific computing is the future of parallel computing; it is that the body of knowledge created in building programs that run well on massively parallel computers may prove useful in parallelizing future applications.' Therefore, the individual implementations described in subsequent chapters may or may not directly address a presently identified, computationally intractable science question. The algorithms have been selected because they are collectively representative of broader algorithmic characteristics which find repeated use across the spatial analytical stack.

2.1.2 *Motivating a Taxonomy of Implementation Techniques*

The development of a taxonomy of implementation techniques focusing on the spatial analysis domain is essential for a number of reasons. First, the integration of modern parallelization techniques, CI, and spatial analytical methods is emergent (Wright and Wang, 2011) and middleware implementations are lacking fundamental comparability. Current implementations, distributed across domains and the literature, offer algorithm implementations with varying focus, level of description, and vocabulary. This hinders breadth of parallel algorithm coverage across the spatial analysis stack due to difficulty in identifying commonality between atomic algorithmic operations from divergent analytical methods. A taxonomy is well suited to support standardization of vocabulary and classification criteria.

Second, McKenney (2013) suggests that parallel programming is difficult due to the limited availability of highly scalable hardware and hard to use tools. The application of parallel programming methods to spatial analysis algorithms is no different. Within the context of parallel spatial analysis algorithms, works focusing on multiple implementation methodologies, (e.g. (Rey *et al.*, 2013)), with the goal of quantitatively and qualitatively comparing performance and ease of implementation are not commonly found. While highly focused works are essential, increasing the breadth of implementations can significantly benefit from a set of best practices or lessons learned. A taxonomy provides a framework through which analytical tasks can be decomposed into composite parts, as well as a set of proven implementation techniques. Coupled with comparability, the taxonomy proposed below seeks to provide the theoretical scaffolding to support improved (re)implementation efforts.

Finally, working within the GCI ecosystem Wang (2013), Anselin *et al.* (2014) identify two hurdles in the deployment and utilization of a spatial analysis library, [2] interoperability and the deployment of serial analytic code to distributed systems. Previous works within this domain have largely focused on the implementation of single algorithms within the context of a single hardware deployment. In aggregate, these efforts form a bottom-up approach to the definition of a corpus of implementation specifics. These efforts are, unfortunately, tightly coupled to specific hardware and serve primarily to offer a literature from which larger meta methods can be

[2] Python Spatial Analysis Library (PySAL) Rey and Anselin (2010).

identified. In contrast to this bottom-up approach, I seek to provide a top-down classification of the broad characteristics of spatial analysis algorithms within a parallel domain with the goal of providing guidance for the (re)development of serial analytical code for HPC environments. It is through the top-down development of classification criteria and the bottom-up assignment of membership within that classification scheme, that a usable tool for robust implementation within CI environments can be developed. Within the context of CyberGIS, the proposed taxonomy is an essential theoretical component of the middleware framework that broadly supports algorithmic implementation. This in turn is a key infrastructure component, addressing the second hurdle identified by (Anselin *et al.*, 2014) at a scale well beyond the individual algorithm, to support the iterative utilization and refinement of CGI.

The remainder of this chapter is organized as follows. Sections 2.2 describes the implementation environment, the Python Spatial Analysis Library. Section 2.3 reviews existing taxonomies from the Computer Science, GIScience, and operations research domain. In Section 2.4 I describe those criteria included and excluded from the taxonomy. Section 2.5 forms the bulk of this work and outlines the proposed taxonomy of parallel spatial algorithms. I explore the combination of taxonomic classes to form analytical methods in Section 2.6, offer an introduction to the subsequent chapters framed as case studies in Section 2.7, and close with concluding remarks in Section 2.8.

2.2 Python and the Python Spatial Analysis Library

All implementations referenced through this work have been developed in the Python programming language for a number of reasons. Langtangen and Cai (2008) provide an excellent comparative analysis of the development of high-performance scientific computing code with an eye towards the qualitative development process and quantitative performance considerations. To summarize, Python offers a robust infrastructure of numerical, graphic user interface development, and data I/O libraries. The existence of these libraries, coupled with the higher level interface Python offers serves to significantly reduce development time. The cost of this qualitative improvement is realized at runtime. Python overcomes this shortfall by providing direct access to C and Fortran code, which can be utilized in performance bottlenecks, e.g. nested `for` loops. Using a pure Python approach, Langtangen and Cai (2008) show that runtimes are on the order of 2 times slower than C (and Fortran) code. Using a hybrid approach Python offers comparable speeds. In a HPC environment hybrid C / Python code generally performs as well as pure C, with significantly less code being developed (Langtangen and Cai, 2008). [3] The choice of Python as the primary development language was a function of the aforementioned strengths and the existence of a robust spatial analysis library written in pure Python.

The Python Spatial Analysis Library (PySAL) is an open-source spatial analytics library (Rey and Anselin, 2010). PySAL utilizes a modular design centered around a core of low-level functionality, e.g. file Input/Output, geometric data structures and operations. Additional modules provide analytical methods for spatial regionalization, Exploratory Spatial Data Analysis (ESDA), network constrained spatial analysis, measures of spatial dynamics, spatial weights operations and measures of spatial inequality (Anselin *et al.*, 2014). Code generated for this work is integrated within the PySAL library and the associated Parallel PySAL (pPySAL) branch. [4]

A key consideration in algorithm optimization is the existence of some benchmark implementation. A key reason the PySAL library is being used is because of the

[3] I note that the referenced implementation focuses on a problem domain inherently amenable to vector representation.

[4] See http://github.com/pysal/pPysal.

breadth of existing coverage across the spatial analysis stack; PySAL provides many serial benchmark implementations. Targeting PySAL largely removed the need to develop both reference implementations and parallel test implementations. I note that no library offers complete coverage across the spatial analysis stack and the included review of existing parallel vector spatial analysis algorithms is invaluable in supporting comprehensiveness of the proposed taxonomy.

2.3 Existing Taxonomies

This dissertations is sited at the intersection of fields: parallel computing within the computer science domain and spatial analysis algorithms within the GIScience domain (Goodchild, 1992, 2010). Therefore, existing taxonomies from both fields provide a foundation from which to synthesize a new representation of the state of parallel spatial algorithms.

From the Computer Science domain, Asanovic *et al.* (2006) propose a taxonomy of atomic computational methods and the associated communication patterns ('dwarfs') which are the core algorithmic building blocks common across a range of computational problems. These abstracted mathematical procedures are defined by the high-level computational problem they solve and the communication pattern realized in a distributed environment. The development of these dwarfs serves to answer the question 'What are the common kernels of the applications?' (Asanovic *et al.*, 2006).

The field of GIScience also offers insight into broad methods of classification for families of algorithmic operations. An early classification of spatial statistical methods, provided by Cressie (1990, p. 9, 10), subsets spatial statistics methods based upon the underlying representation of the data. These classes are geostatistical, for continuous data, lattice for areal unit data, and point pattern. Egenhofer *et al.* (2010)

suggest that classic vector-based GIS operations are definable as being *topological*, indicating that the underlying information is invariant to transformation, *directed*, indicating that the direction described relationship between geometries is essential, or *metric*, indicating that the distance between entities is of the utmost interest. Likewise, Dowers *et al.* (2000) broadly classifies spatial analysis methods suitable for parallelization in the context of supporting more robust development and roll-out via strict adherence to principals of interoperability. A subset of these methods, which focus on the vector analysis domain, include feature generalization, feature manipulation, e.g. topology creation and feature analysis, e.g. overlay. Andrienko *et al.* (2011) provides a taxonomy of movement analysis targeting spatio-temporal data representation, processing, and visualization, while Jain *et al.* (1999) provide a classification of data clustering, an inherently spatial problem, which focuses on the hierarchical structure of the analytic technique. Specifically within the context of parallel spatial algorithms, Ding and Densham (1996) focuses on decomposability and load balancing in order to identify problems based upon homogeneity in the attribution of the data and regularity of the spatial distribution.

One can also look to the operations research domain for some insight into the classification of pseudo-spatial algorithms. Both Trienekens and Bruin (1992) and Crainic *et al.* (1997) provide domain specific taxonomies for the classification of enumerated (former) and heuristic (latter) search techniques with both classifying algorithms based on the methods of communication and synchronization as a product of the decomposition. Trienekens and Bruin (1992) classifies algorithms based on classic decomposition and synchronization techniques as well as the methods by which knowledge is shared and leveraged. In the case of fully enumerated branch and bound algorithms the process by which other works become aware of changes to the global knowledge base is essential. Crainic *et al.* (1997), in classifying parallel Tabu

Search methods, identifies control (synchronization and management), communication paradigms, and heuristic search differentiation as the three primary dimensions. Both works present a logical partitioning of algorithm attributes existing as a hybrid of fundamental parallel computing methods and domain specific operations. The former seeks to identify those implementation realities which all algorithms must conform to, while the latter provides the necessary linkages back to the domain of interest such that the taxonomy remains relevant.

Clearly, as one moves from the Computer Science domain, to the spatial analysis domain, the focus shifts from communication and synchronization, lower level implementation requirements, to methods, applications, and data structures. Therefore, a synthesis of these domains must maintain some common vocabulary to be accessible and ultimately, relevant.

This work leverages and extends the work of Asanovic *et al.* (2006), through the classification of spatial algorithms into a framework of computational dwarfs and their communication patterns. The proposed classification scheme provides a common platform from which computer scientists can assess the placement of spatial algorithms within the parallel computing domain and GIScientists can identify computational commonalities within classes of spatial analysis algorithms. That is, classification of methods by intrinsic commonalities in data structures and analytical approach naturally lends itself to classification by atomic mathematical operations and communication patterns.

2.4 Defining Dwarfs

Dwarfs are defined using two criteria, computational method and communication pattern. The computational method is the basic mathematical operation which defines some expensive process. The communication pattern is the observed frequency and topology of inter-node communication. Communication patterns are identified from existing parallel implementations. In instances where two similar mathematical operations exhibit distinct communication patterns, one can (and should) identify a new processing dwarf. The inverse is also true, where commonality in communication, but not computation, is indicative of a new dwarf.

2.4.1 Atomic Compute Operations

Atomic compute operations are high-level classification constructs designed to abstract low-level implementation details, i.e. these are the 'computational kernels' which can be aggregated to create most if not all of the currently utilized mathematical, analytical methods (Colella, 2004; Asanovic *et al.*, 2006; Kaltofen, 2014). The concept of identifying high-level classes of low-level operations within the context of computational simulation was first proposed by Colella (2004). He identifies the following seven dwarfs: (1) structured grids, (2) unstructured grids, (3) Fast Fourier Transforms, (4) dense linear algebra, (5) sparse linear algebra, (6) particles (e.g. N-Body problems), and (7) Monte Carlo simulation. Leveraging this initial work within the context of a generalized taxonomy of parallel algorithm characteristics, Asanovic *et al.* (2006) identify thirteen dwarfs which are distinct from underlying hardware implementations. This latter point is essential in decoupling the classification of an algorithm's behavior from a low-level classification of the algorithm coupled with the implementation hardware. The defined dwarfs are: (1) Dense Linear Algebra, (2) Sparse Linear Algebra, (3) Spectral Methods, (4) N-body methods, (5) Structured and (6) Unstructured Grids, (7) MapReduce (Monte Carlo), (8) Combinatorial Logic, (9) Graph Traversal, (10) Dynamic Programming, (11) Backtrack and Branch and Bound, (12) Graphical Models, and (13) Finite State Machines. While this listing appears exceptionally inclusive, Asanovic *et al.* (2006) are proponents of the identification, definition, and inclusion of additional computational dwarfs. To that end, Kaltofen (2014) extend the Berkley dwarfs with seven additional dwarves focusing on symbolic computation. While the specifics of the implementation are beyond the scope of this work, the methodology of extension is one which I emulate.

When identifying the atomic operations which characterize some analytical task, it is clear that the vast majority of algorithms from the spatial analysis stack can be composed of one or more dwarfs. Therefore, the utilization of an analytical method to highlight or explore classification within a dwarf is not an explicit assertion that the method exists only within that dwarf. The concept of chaining dwarfs is explored below. It is also clear that the definition of atomic compute operations, as a means of characterization, lacks linkage defining strict implementation rules back to the lower level implementation of a parallel algorithm. This linkage is made during the heuristic process used to identify algorithms ripe for parallelization. That is, paramount in the development of a successful algorithm parallelization are the identification of the processing bottleneck(s), suitable methods of decomposition, communication, and processes synchronization, the parallel programming model to be used.

2.4.2 Communication Patterns

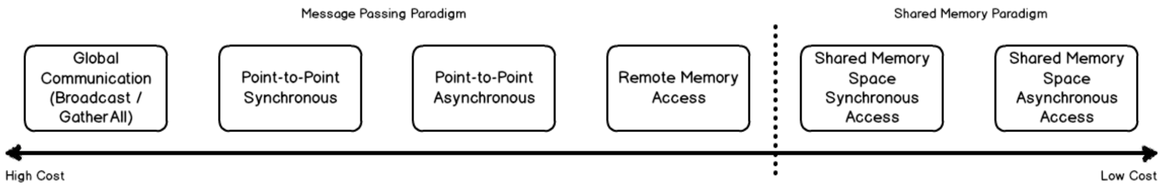
Amdahl's Law provides unlimited scalability assuming an idealized load balancing, no serial processing component, and zero communication costs. Unfortunately, parallel programs only theoretically conform to this law. The key cost in the process of parallelization is that of inter-process communication (Grama *et al.*, 2003), or the act of transmitting data between two or more CPUs. Due to the cost of communication, the synthesis of computational dwarfs and communication patterns is a natural pair-ing; identification of the most expensive aspect of the parallelization processes has led to the development of optimized solutions within the parallel computing literature. [5]

Communication patterns are an aggregate descriptor used to describe the frequency, size, and scope of interprocess communication. The two previously defined system architectures, shared and distributed memory, give rise to two broad categories of communication (1) message passing and (2) shared memory. The Message Passing Interface (MPI) (Forum, 1994) is becoming the de facto message passing protocol in scientific computing and provides a standardization of common message passing methodologies that are largely portable across hardware environments.

Within the MPI framework, three communication strategies are defined: Collective, e.g., `Scatter-Gather` or `Broadcast`, Point-to-Point, and Remote Memory Access (RMA). Global communication sees one or more processing cores sharing data to all other processes. For example, `Broadcast` distributes data from one process to all other processes. The `Scatter-Gather` paradigm roughly evenly distributes data across some number of processes and later gathers that data back to a single manager. Point-to-point communications are composed of messages directly passed between processes, e.g., a word passed from a mother process to a child process. These can be synchronous, where the sender waits for confirmation that the word has been transmitted, or asynchronous, where the sender transmits information and immediately continues processing. Finally, RMA is a communication model where a remote process directly accesses, without synchronization, the memory space of a different processing core. Each of these methods incurs communication costs during three phases: (1) startup time which includes message package and routing, (2) transmission time which is the time required to move headers between nodes, and (3) data

[5] It should be noted that these method hold until peta-scale computing, after which point optimized communication of any kind is an open problem, e.g. quadratic cost in routing algorithms alone add significant overhead (Gropp, 2009).

transfer time (Grama *et al.*, 2003). Within the shared memory paradigm, a single memory space is globally accessible, either synchronously, or asynchronously, to all processing (OpenMP, 2013). Assuming an identical data transfer requirement, the sequence of methods presented represents a descending cost of communication model, Figure 2.1The Continuum of Communication Costs Incurred by Different Data Sharing Modelsfigure.2.1.


Message Passing Paradigm
Shared Memory Paradigm
Global Communication (Broadcast / GatherAll)
Point-to-Point Synchronous
Point-to-Point Asynchronous
Remote Memory Access
Shared Memory Space Synchronous Access
Shared Memory Space Asynchronous Access
High Cost
Low Cost
Figure 2.1: The Continuum of Communication Costs Incurred by Different Data Sharing Models.

The process of disentangling communication patterns from concepts of granularity, decomposition methods, and load balancing is challenging due to the multi-faceted relationship between these concepts. For this reason, algorithms are intentionally classified at a high-level describing only the patterns by which information is transferred. At a lower, implementation level, all dwarfs benefit from ideal load balancing via decomposition, parallelized input/output and a careful balance in communication granularity. The following provides a brief overview of these idealized implementation details. More specific implementation information is provided within the definition of each dwarf.

Decomposition Methods

The decomposition of an algorithm requires the identification of those tasks which are composed of repeated computation and the identification of data dependencies between these tasks such that a directed task dependency graph can be generated. The remainder of this subsection describes task and data parallelism.

By way of example, imagine two different analytical flows focusing on spatial regression. In the first example, a spatial regression model is to be estimated (Task A). This model requires some measure of spatial interaction, a W object, be computed (Task B). Task A can not be computed without first computing Task B, yielding a two node directed graph with no branches. It is not possible to exploit a task level parallel approach in this example. Using an example from LeSage (2014), assume that two methods of describing spatial interaction wish to be defined for use in an extension to a standard spatial regression model in the form:

$$y = \rho_S W_S y + \rho_T W_T y + X\beta + \epsilon, \tag{2.1}$$

where the key variables of note are W_S (spatial proximity) and W_T (technological proximity), two differently specified spatial adjacency objects. [6] Both W objects can be computed independently without the need for inter-process data communication. Therefore, task parallelism can be leveraged to perform two different computations concurrently (Tasks B and C). Task A is still data dependent upon the completion of Tasks B and C.

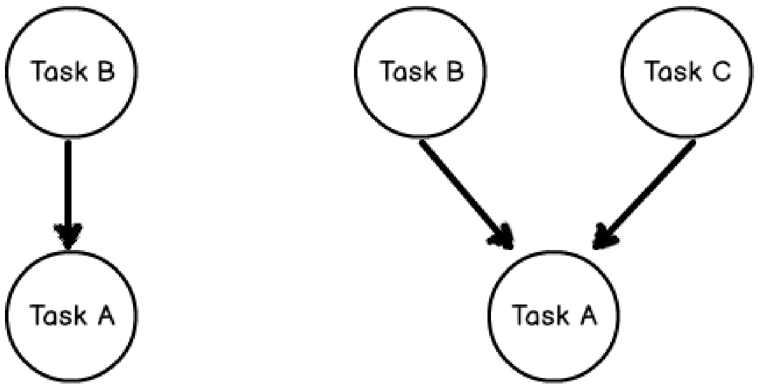


Figure 2.2: Two Sample Directed Dependency Graphs Illustrating (a) No Potential Task Parallelism and (b) a Two Node Level with Potential Task Parallelism.

[6] I note that LeSage (2014) specifically suggest **not** specifying a model in this way and use this example only as a straightforward task parallel example.

Using the generation of a W object as an example, it is possible to identify a candidate for data level decomposition, and parallelism. Here the same computational problem, the generation of an adjacency structure using a distance metric or an adjacency matrix, must be applied to some set of data. Decomposition of that data can take many forms, (e.g. spatial decomposition using a regular grid), but the underlying algorithmic function is invariant to the decomposition method. Likewise, the row standardization of said matrix, which ensures that each row sums to unity, could be row-wise decomposed and computed exploiting data level parallelism.

Clearly, an analytical workflow will not be composed of distinctly task or data parallel requirements, but a potentially complex dependency graph with task and data parallel computation levels. At the proposed taxonomic level, I identify each node as a potential dwarf and therefore, do not attempt to make the task or data parallel distinction. This distinction is key at a higher, composite level where distinct dwarfs are chained (described below). By extension, I explicitly exclude task scheduling or load balancing as these concerns are a function of the hardware, specific implementations, and input data.

Synchronization

Many of the methods above make a distinction between synchronous and asynchronous access. Synchronicity is an essential component of the performance of any communication paradigm. In instances where synchronization is required, whether, through the use of locking, semaphores, or message confirmations, one or more processing cores are potentially idle. The aggregate of minuscule costs of process idle time can rapidly exceed the benefits of parallelization. For this reason, asynchronous communication methods generally provide better overall performance. The cost of asynchronous communication in non-embarrassingly parallel implementations manifests as significantly higher complexity. From an implementation perspective, asynchronous implementations can provide the most performance at the cost of testability, reliability, and maintainability. Synchronous implementations are frequently required, but at a tangible cost to performance. Therefore, experience has shown that performant solutions are possible through the utilization of periodic bulk synchronization Valiant (1990) where all processing core temporarily halt and communicate. The benefit to this model is a more predictable pattern to core idle time, as a function of communication requirements.

Again, synchronization is explicitly excluded as a classification criteria as individual implementations will seek to balance ease of implementation with raw performance improvement. Including synchronization doubles the effective number of classes without offering a concrete classification tool; I assert that most implementations could leverage synchronous or asynchronous methods dependent upon algorithm structure and underlying hardware.

Input/Output

Finally, a key cost to any implementation is data Input and Output (I/O). I use I/O to mean not just the process of moving data from disk to memory, but also the process of CPU access to data in some memory location (RAM or cache). This data movement is a primary bottleneck inefficient computation (serial or parallel). As early as 1993, the starving CPU is being identified as a primary performance bottleneck with Alted (2013) quoting Kevin Dowd as saying

> We continue to benefit from tremendous increases in the raw speed of microprocessors without proportional increases in the speed of memory. This means that 'good' performance is becoming more closely tied to good memory access patterns, and careful re-use of operands.

Clearly, this problem extends through the I/O stack, from initial data read to processing. The introduction of a hierarchy of caches, which provide higher speed CPU access to progressively smaller memory areas helps to improve performance, assuming the code has been developed and tuned for a given architecture. Herein lies a reason why I/O can not be a high-level classification criteria; two identical algorithm implementations with identically identified processing dwarfs can perform divergently assuming the low-level implementation of one is tightly tuned for some hardware environment. These low-level details can be the exploitation of cache layouts or the exploitation of parallelized data reads and writes. The inclusion of this criteria exponentially increases the size of the classification mechanism and effectively renders a one-to-one implementation to class relationship.

Performance Metrics

I look to existing taxonomies within the computer science domain for guidance on the inclusion of performance metrics as a classification criteria or as an additional informational item. Stratton *et al.* (2012) offers a taxonomy of parallelization methods for GPU based programs targeting Parboil benchmarks. While performance improvement is the over-riding concern and performance metrics are occasionally provided as a comparison between implementations using divergent techniques, performance is not a universally leveraged metric. Asanovic *et al.* (2006) suggests performance is a function of many concerns beyond algorithmic formation such as memory access latency, memory availability, CPU size, CPU capacity, or CPU distribution, to name a few. Clearly, a successful implementation is one which provides performance improvement, but the explicit inclusion of performance adds a dimension to the classification schema that tightly couples of hardware; I seek to avoid this coupling.

2.5 Proposed Taxonomy of Parallel Spatial Analysis Algorithms

Leveraging the concept of computational dwarfs, I propose the following taxonomy of spatial analysis methods. The proposed classification of methods is constrained to vector spatial analysis algorithms. With an eye towards this boundary and the wealth of spatial analysis occurring in the micro-scale, desktop environment, I seek to bridge implementation hardwares and offer a classification suitable for shared and distributed memory architectures. To that end, I utilize the communication vocabulary defined within the broadly used MPI communication protocol, while suggesting that lower level implementations realizing similar communication patterns in an SMP environment are possible.

The taxonomy is composed of six classes which describe six computational dwarfs commonly encountered within the spatial analysis stack. Three of these dwarfs are drawn directly from Asanovic *et al.* (2006) as the mathematical operations are ubiquitous. These are the (1) MapReduce (Monte Carlo), (2) Sparse Linear Algebra, and (3) Dense Linear Algebra dwarfs. In asserting that spatial is special, I identify three additional computational dwarfs specific to the spatial analysis domain. These are (4) Geometric, (5) Topological, and (6) Exploratory.

2.5.1 MapReduce

MapReduce, Monte Carlo, or embarrassingly parallel implementations[7] are the most basic data-parallel implementations with no inter-core communication required. MapReduce style dwarfs find wide application across the spatial analysis stack. Commonly applied analytics amenable to MapReduce style parallelization include simulation for local Moran and local G statistics (Wang *et al.*, 2013). To support inference,

[7] All three names describe an identical process and this work uses MapReduce in line with Asanovic *et al.* (2006).

these methods utilization simulation, for example, the random permutation of some attribute vector across a set of spatial observations and subsequent computation of a local statistic. Decomposition is generally trivial for MapReduce style algorithms as tasks are identical and independent. Therefore, a simple n/p decomposition, where n is the required number of operations and p the number of cores, is generally sufficient. Using the above example computing a local Moran's I statistic using 100 permutations, a four core parallel implementation could perform 25 permutations local to each core and then aggregate the necessary scalar results back to a master process.

I identify this type of computational dwarf being widely leveraged within the spatial analysis domain. For example, Laura *et al.* (2015) implement a MapReduce based spatial regionalization algorithm in both SMP and HPC environments. Li *et al.* (2014c) deploys a spatial adjacency algorithm into a distributed Hadoop environment and shows massive performance improvements. [8] Local Moran's I is explicitly identified as being computable using an embarrassingly parallel implementation by Wang *et al.* (2013) with conditional permutation being ideally suited to basic domain decomposition and concurrent processing. Finally, Liu *et al.* (2010) describe a MapReduce style implementation using a distributed file system to compute the $G_i^*(d)$ statistic.

It is with careful consideration that I maintain the term MapReduce to describe this paradigm of programming, understanding that the reader may immediately draw linkage between the prevalent Hadoop MapReduce framework and this class. While Hadoop MapReduce has become one main implementation of the MapReduce paradigm, it remains just that - an implementation. The key defining characteristics of decomposition, mapping of some functional operand, and reduction can be traced

[8] I note that this method does not appear to leverage any explicit global sorting, an essential component of a geometric dwarf.

to the development of functional programming languages, e.g. LISP and observed in common data analysis workflows, e.g. the split-apply-combine paradigm employed by the R scripting language. The common linkage for all members of this dwarf is the ability to avoid communication during processing and remain embarrassingly parallel.

2.5.2 Dense Linear Algebra

Dense linear algebra problems are defined by the application of some mathematical function to sub-sections (column(s), row(s), or blocks(s)) of a dense matrix. These implementations generally leverage aspatial decomposition methods which seek to balance a raw count of the number of computations across available compute cores. Communication is generally fine grained and topologically close, seeking to transmit scalars, vectors, and blocks to nearby neighbors for aggregation or population of some composite data structure.

As a trivial example, assume the computation of a subset of the Moran's I statistic, namely:

$$\frac{n}{\sum_{i=i}^{i=n} \sum_{j=1}^{j=n} W_{ij}}, \tag{2.2}$$

where W is a spatial weights matrix, n is the number of elements in W, and i, j are element indices. Computation of the denominator requires the summation of all elements within the W matrix. It is possible to decompose said matrix into rows, columns, or blocks and communicate said data to available compute resources. Once distributed, each core can apply come operand, summation in this case, in order to compute a local interim solution. These solutions can then be aggregated, with the possible application of another operand, to iteratively compute the final solution. More broadly, this process is composed of a data `scatter`, application of the same operand to different data, and hierarchal aggregation of the result.

Common applications in the spatial analysis domain include basic matrix operations on dense (continuous) **W** objects and some data classification algorithms, e.g. Fisher-Jenks optimal choropleth map classifier (Rey *et al.*, 2013; Laura and Rey, 2014). Another composite example of two dense linear algebra dwarfs, in the context of spatial regression, is the computation of

$$(I - \rho W)^{-1}, \tag{2.3}$$

where, I is an identity matrix, ρ is the spatial autoregressive parameter, and W is a spatial weights object. Here both the matrix inversion and computation of ρW can be classified as dense linear algebra operations. [9]

Additional applications of the dense linear algebra dwarfs within the spatial analysis stack include the decomposition and load balancing approach developed by Shook *et al.* (2013). This work leverages row wise decomposition of a, potentially, dense matrix in order to approximate the total computational cost prior to the iteration of an agent-based model.

2.5.3 Sparse Linear Algebra

Like Dense Linear Algebra, Sparse Linear Algebra applies some mathematical function to a row, column, or block. Implementations use metadata structures to manage a compressed form by which most if not all of the zero entries can be removed. This storage requirement alters the implementation requirements, in that an efficient means to communicate data vectors (as is the case with Dense Linear Algebra) plus potentially irregular metadata is required. Assuming these criteria are met, the

[9] The W can initially have a sparse representation, resulting in the combination of sparse and dense linear algebra dwarfs.

distinction between linear algebra dwarfs is largely definable by an increase in the distribution of communication from the close topological neighbor to a slightly broader neighborhood as a result of increased communication to query potential points of interest in the sparse representation.

2.5.4 Geometric

Geometric algorithms compare sets of geometries, without requiring an explicit topological structure defining the connectivity between components. This is not to suggest that the spatial distribution of the geometries is not invaluable, but that global distribution (`broadcast`) of the connectivity is not a required process. Communication patterns are generally point to point, transmitting pivot information and geometries during the process of applying global sort methods.

Two prime examples of this dwarf are single nearest neighbor computation and the generation of a binary spatial adjacency matrix using either rook (shared edge) or queen (shared vertex) contiguity measures. In both cases, the global distribution or topology of the data is not required to be known by all processes. This facilitates the use of spatial decomposition methods, without requiring the global transmission of the entire dataset, and by extension topology, to all processors. This in stark contrast to the *topological* dwarf, described below.

Using single nearest neighbor as an example, performant implementations are characterized by distributed global sort (Atallah *et al.*, 1989; Dehne *et al.*, 1996) using one or more phases of fine-grained point-to-point communications interspersed by the application of a traditional, serial, nearest neighbor algorithm. This is a spatial decomposition that leverages the high-performance of sorting algorithms. To briefly describe this process, imagine a matrix, composed of two rows containing x and y coordinates, respectively. The computation of nearest neighbor is performable by globally sorting on the y-axis, computing nearest neighbors from the set of locally stored points,e.g. those sorted points local to a given processor. Sorting and nearest neighbor search is then repeated for the x-axis. Finally, those points which are nearer to the x and y pivot values than an already identified nearest neighbor are found and another potential nearest neighbor computed. Given these three nearest neighbors (x dimension, y dimension, and edge case), global sort can be used a final time to identify the nearest neighbor. For a full formulation see Atallah *et al.* (1989) and Chapter 4 for an in-depth description of this method.

I identify this computational dwarf within the computational geometry domain and identify cases where the algorithms are used to support GISystems operations. For example, Puri and Prasad (2014) implements an optimal, output sensitive polygon clipping algorithm which requires a global sort to segment and initialize a line sweep algorithm. Similarly, Hoel and Samet (2003) identifies a sort-based method to support line segment polygonization. Dehne *et al.* (1996) offers a sort-based method to support nearest neighbor search and Atallah and Goodrich (1984, 1985); Atallah *et al.* (1989) describe sort based nearest neighbor search, convex hull, polygon triangulation, and line segment intersection detection.

2.5.5 Topological

Topological dwarfs perform algorithmic operations based upon the connectivity of bodies. This is similar to the N-body problem (Asanovic *et al.*, 2006) in that the interaction is between discrete observations and the Graph dwarf focusing on sequential lookup of information. A distinction can be drawn because the topology is a necessary additional data component, in the case of the former, and the computations can require significantly more complex operations than simple lookup table access, in the case of the latter. In computation, the underlying topology is leveraged to apply some analytical method utilizing approaches similar to those leveraged by geometric dwarfs. A necessary pre-processing step for the application of a topologically constrained algorithm, is the distribution of the topology using some global communication method, e.g. `Broadcast`. Efforts to decompose topologies into component parts, perform non-complete computations, and aggregate these partial computations back into a global solution, in the style of a sub-optimal branch and bound style implementation, have proved exceptionally difficult to manage.

Two examples of topological dwarfs are the computation of a network nearest neighbor and the parallel query of a KD-Tree. In the case of the former, knowledge of the entirety of the network allows for an efficient decomposition of the point observations and the application of logic similar to the geometric, planar case. Without knowledge of the entire network, significant boundary information must be either duplicated or communicated. In practice this has been found to be prohibitive. Likewise, a parallel tree search begins are some node and then leverages a depth or breadth first search procedure. Management of the metadata defining sub-graph placement within a distributed KD-Tree and the message passing required to travel between sub-graphs has proved less efficient than a global communication and parallel search approach. Without a doubt, it may be possible to decompose topologies and utilize a geometric dwarf, but the cost of this will likely be a significantly less maintainable algorithm.

Communication is characterized by a single, global `Broadcast` where the topology of the network is distributed across all process. Once completed, either fine gained communication can be used in the same manner as geometric dwarfs to traverse topologies or a MapReduce style implementation can manage non-communicating computation. The necessity to utilize global communication suggests that only the most efficiently communicated data representations can be leveraged. That is, a method for parallel I/O to bulk load a pre-generated topology, a highly efficient structure to communicate the topology, or an efficient serial construction algorithm that allows for an excess communication model are required to facilitate successful parallelization of topological dwarfs.

By in large, the distinction between topological dwarfs and geometric dwarfs is small. The distinction between the use of point-to-point global sort methods as the predominant form of communication (*geometric* dwarfs) and the requirement for precursor global communication (*topologica*l dwarfs) must be made as the implications in implementation are meaningful. While the root numerical methods both dwarfs utilize can be defined as solving topological problems (in a GIScience sense), geometric dwarfs do not require the connectivity between entities be globally know. The topological dwarf is therefore named not for the numerical method, but for the communication requirement; a globally transmitted data structure describing the connectivity of all entities must be populated and transmitted resulting in significantly divergent communication requirements. This distinction is further realized at the intersection of the computational geometry and GIScience domains. The vast majority of works identified as geometric dwarfs exist within the former body of literature. In the latter, Armstrong and Marciano (1996) utilizes the global topology of a number of grid cells to significantly improve the performance of IDW interpolation and Wang and Armstrong (2003) utilizes a quad tree based approach, with parallel search for the k nearest neighbors and IDW computation. Finally, Tang (2013) identifies the need to generate, store, and update a spatial index in the generation for circular cartograms. This operation requires the distribution of a topological, tree representation of the spatial index, allowing classification as a topological problem, albeit in a shared memory environment. While not largely leveraged outside of the database domain, e.g. in cached driving directions, it is also possible to identify network constrained spatial analysis as requiring, in most instances, the full topology prior to parallel computation.

2.5.6 Exploratory

Asanovic *et al.* (2006) identify branch and bound search as an algorithm family suitable for solving optimization problems. Assume that the optimization function and constraints define an n-dimensional solution space which can be continuous, in the case where integer requirements are not enforced or discrete, in the case where integer requirements are enforced. Branch and bound style algorithms seek to solve sub-areas within the solution domain and prune sub-optimal solutions to find one or more optimal solutions. Many spatial optimization problems are not solvable using this approach due to difficulty in formulating end enforcing a spatial contiguity constraint. In fact, the computational complexity makes search of a high percentage of the solutions domain infeasible. In those instances, heuristic solution methods are often employed. It is these heuristic search methods that this dwarf seeks to define. The root numerical methods seek to permute realizations of some solution with the goal of identifying an optimal solution. This process is made significantly more efficient through the use of knowledge sharing approaches, as evidenced in Chapter 6. That is concurrently solved solutions can share algorithm parameterization or solution characteristics. The atomic compute component could be any heuristic search method, e.g. GRASP, Simulated Annealing, Tabu Search, Neural Networks, Genetic Algorithms, with the additional constraint that an inter-process, cooperative search strategy is employed.

Communication can be characterized by iterative fine-grained, topologically close message passing terminated by periodic bulk synchronization (Valiant, 1990) and global, *broadcast* / *gatherall* style communication. The exploratory component of spatial regionalization algorithms can be decomposed as either a series of concurrently running searches or a single cooperative search. In the former an embarrassingly parallel decomposition leverages fine grained, topologically close message passing (or shared memory access) to share individual solution information. These searches are interrupted by periodic global bulk synchronization, realized as coarse-grained communication where all solutions are collected, ranked, and broadcast. Chapter 6 provides a sample implementation using this method. In contrast, the latter method employs a spatial decomposition over a single local search. Decomposition occurs by distributing the enumeration of all possible solution permutations across processing cores and the sequential aggregation and ranking via fine grained topologically close communications.

2.5.7 Extensions

Implementations will have to adjust with hardware, but general trends should be largely constant. That is, the core mathematical operations and communication methods used to facilitate concurrent computation will remain largely unchanged. For this reason, I assert that the extension of the Asanovic *et al.* (2006) computational dwarfs provides a robust theoretical framework into which this work can integrate. The classifications are largely invariant to (1) improved communication, e.g. reduced latency and (2) distributed data storage and asynchronous I/O, e.g. by leveraging a spatial database. The identification of new dwarfs or modification of class membership can, and should, be assessed in those cases where significant algorithm refactoring, as is the case when a low-level parallelization yields an entirely new algorithm.

Dwarf	Description	Communication Pattern	Examples
Map Reduce†	Embarrassingly parallel, repeated trials.	None	Permutation based analytical methods, Moran's I, LISAs
Dense Linear Algebra†	Data represented as dense matrices and vectors.	Point-to-Point, Local	Fisher-Jenks, $(I - \rho W)^{-1}$
Sparse Linear Algebra†	Data represented as sparse matrices, generally yielding dense representations and using offset counting mechanisms.	Point-to-Point, Neighborhood	Cholesky Decomposition, W Object Manipulation(ρW or transformation)
Geometric∗	Data represent discrete geometric entities without the need to represent the topology of two or more elements.	Point-to-Point	Adjacency Metrics, Convex Hull, Line Segment Polygonization
Topological∗	Data represents the interaction of N-observations by storing the underlying topological relationship. (Similar to N-Body problems.)	Global (`Broadcast`)	Network Analysis, Tree Traversal, Nearest Neighbor Search
Exploratory∗	The exploratory search of solution space with data defining the space extent and constraints on traversal.	Global, Point-to-Point, and RMA	Spatial Regionalization

Table 2.1: Sample Classifications Using the Proposed Taxonomy. *Dwarf*† Are Drawn Directly from Asanovic *et al.* (2006), While *Dwarf*∗ Are Proposed to Leverage the Assertion That Spatial Is Special.

2.6 Combinations of Dwarfs

The vast majority of analytic algorithms within the spatial analysis stack are composed of multiple dwarfs. For example, the act of spatial regionalization, Figure 2.3Sequential Chaining of Computational Dwarfs in a Cooperative Regionalization Algorithmfigure.2.3, requires the generation of a spatial weights, W object, to capture the adjacency structure of the areal unit data. The generation of this data structure is classified as a *geometric* problem. Next, one or more initial feasible solutions must be generated. This processes can be parallelized using a *MapReduce* approach, where each core generates some number of Initial Feasible Solutions (IFS) with no inter-process communication. Once the generation of an IFS has been completed, the regionalization algorithm, classified as *exploratory*, can be applied. All three dwarfs are essential components of the analytical task and the combination of these has direct implications on the global performance of the implementations.

Two key considerations, when combining dwarfs, are the methods of intra-dwarf decomposition, and the data structure requirements imposed through the parallelization efforts (Asanovic *et al.*, 2006). I assert that most, if not all multi-dwarf implementations exhibit time-sharing, rather than spatial decomposition and distribution approaches (Asanovic *et al.*, 2006). That is, existing spatial analysis algorithms tend to be composed of a series of discrete steps with periodic bulk synchronization requirements, i.e. strongly data dependent. Therefore, intra-dwarf decomposition can leverage these properties and schedule sequentially occurring dwarfs on the same compute resources. Again leveraging the spatial regionalization example, above, the implications are that the *geometric* dwarf, *MapReduce*, and *exploratory* dwarves could be scheduled to utilize the same compute resources, knowing that the former would complete before the latter could process.

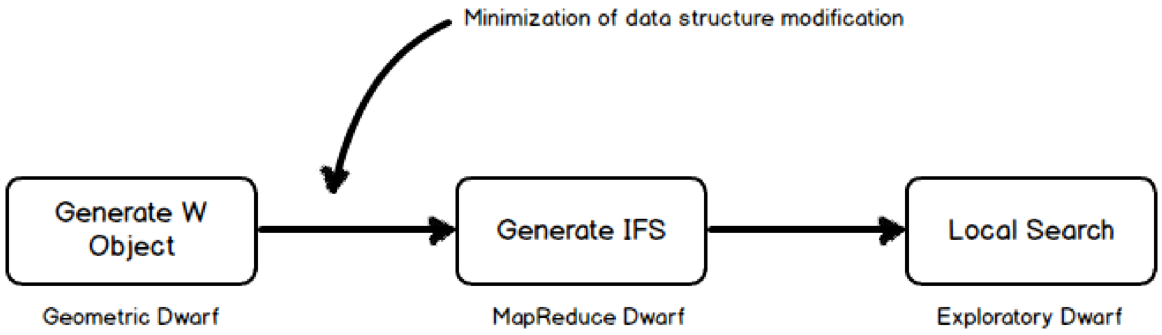


Figure 2.3: Sequential Chaining of Computational Dwarfs in a Cooperative Regionalization Algorithm.

The next hurdle in the chaining of dwarfs is the identification of data structures amenable to highly scalable computation across dwarfs. For example, the data structure used for rapid computation of the W object, should also be either (1) easily convertible or (2) directly usable by the next MapReduce dwarf. Careful consideration must be applied when this is not the case, as the cost of conversion to a necessary data structure can render parallelization efforts unsuccessful.

2.7 Case Studies

The remainder of this dissertation serves as a series of case studies focusing on the implementation of Dense Linear Algebra, Geometric, Exploratory, and MapReduce dwarfs. These case studies both inform and are informed by the taxonomy. In the case of the former, implementation and simulation, as opposed to analytical examination identified the stark distinction between the Topological and Geometric dwarfs. Analytical examination alone suggested that communication would be a function of the underlying data and hardware, where simulation showed that efficiency in representation is the primary determining factor. Likewise, the MapReduce and Linear Algebra dwarfs were directly informed by the taxonomy and implementation could mirror implementations from the Computer Science domain.

Representative algorithms were selected to explore implementation in both SMP and distributed memory environments to illustrate low-level implementation specifics within the context of this high-level classification. These case studies also provide insight into the methods found to be successful in mitigating data transformation when chaining dwarfs.

2.8 Conclusion

This work addresses the hurdle identified by Anselin *et al.* (2014) by providing a high-level taxonomy of parallel computational methods, i.e. dwarfs. With a focus on spatial analysis methods, a taxonomic classification provides an essential framework to support the robust (re)implementation of serial spatial analysis algorithms and reduce the implementation overhead within the context of CyberGIS.

The presented taxonomy utilizes the concept of dwarfs which are defined by their core computational method and the dominant communication paradigm. Synchronization and decomposition are identified as being essential to a successful implementation, but the multifaceted relationship precludes the explicit inclusion of these as means for classification; the complexity devolves the classification scheme to be implementation specific. Using this classification scheme, I adopt three dwarfs, Dense Linear Algebra, Sparse Linear Algebra, and MapReduce, directly from the work of Asanovic *et al.* (2006) and then define Geometric, Topological and Exploratory dwarfs.

Chapter 3

DENSE LINEAR ALGEBRA

3.1 Introduction

Dense and sparse matrices play a key role in many scientific and mathematical applications including supporting graph theoretic operations, e.g. connectivity matrices, the storage of stochastic matrixes for use in probability based statistics, e.g. Markov chains, measures of similarity, e.g. covariance matrices, and for the storage of systems of nonlinear equations, e.g. for use in Maximum Likelihood estimation. The wide utilization of matrices has led to the development of a number of computational libraries specifically designed to offer highly optimization matrix operations. For example, the Linear Algebra Pack (LAPACK), leveraging low level Basic Linear Algebra Subprograms (BLAS) algorithms offers highly optimized routines, tuned to specific hardware environments, for the manipulation of sparse and dense matrices. Highly optimized code can be developed with limited low-level development assuming that a matrix (or vector) representation of the data is achievable and that inter-data processing dependencies can be limited.

The storage and representation of sparse and dense matrices within computer memory is largely similar in that one or more units of contiguous memory are allocated, along with metadata describing the stride position of these data elements within the global array. In the case of the sparse matrix, composed of some number of zero elements, an additional metadata section describes not only the memory layout but also the positional layout of the nonzero elements within the matrix. These representations are highly amenable to high-performance computing because the memory size and representation shape are known a priori; this makes communication of these data elements highly efficient.

The remainder of this chapter is organized as follows. First, Section 3.2 describes the process of vectorization, a major contributing factor to the overall performance improvements. The next Section, 3.3 describes the Fisher-Jenks algorithm, an optimal data classification algorithm which requires dense matrix data representation. In Section 3.4 reports previous parallel algorithm implementations and Section 3.5 introduces the improved algorithms. In Section 3.6 I describe the experimental setup and provide testing results. This chapter concludes with Section 3.7.

3.2 Vectorization

Implicit, hardware level parallelization, is realizable through the use of vectorized computation. Extending the program flow description, above, one can conceptualize a modern, serial, program as flowing through a von Neumann machine, Figure 3.1Simplified Von Neumann Architecture Illustrating the Von Neumann Bottleneckfigure.3.1. That is, information sequentially flows from memory, through a communication pipe, e.g. the Bus, and onto the processor. Armstrong and Densham (1992) suggests that most geospatial analysis occurs using this sequential model. Inherent in this design is a processing bottleneck, the von Neumann bottleneck (von Neumann, 1945, p. 28) which exists at the bus. In order to improve performance within this system caches are used to provide small, yet increasingly large, memory spaces close to the CPU.

Assuming then that computation is memory bound, e.g. the processor waits for data more frequently than it processes data and that compute time is limited by the speed at which data can be accessed and moved from memory to the CPU, one performance strategy would be to maximize utilization of the pipe. This in turn

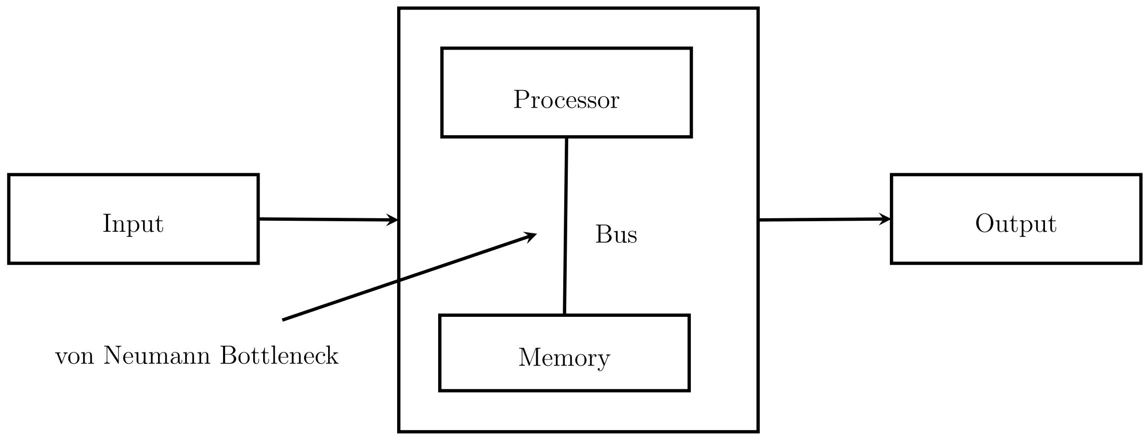


Figure 3.1: Simplified Von Neumann Architecture Illustrating the Von Neumann Bottleneck.

would seek to keep the CPU local memory (caches) as full as possible at all times. Vectorization seeks to realize this goal.

Vectorization is the process by which a Single Instruction is applied to Multiple Data elements. That is the conversion of a scalar direction set that sequentially applies some operations to a flow of data into a concurrent operation where the same operand is applied to multiple subsets of the data flow. Nearly all modern processor have Single Instruction, Multiple Data (SIMD) capabilities allowing for vectorized computations Flynn (1972). Figure 3.2Vectorizationfigure.3.1 illustrates the difference in computation, with the upper three lines illustrating the sequential addition of scalar values. Realized as a loop, the process would be repeated for scalars in element positions two through four, e.g. $A_1 + B_1 = C_1$, followed by $A_2 + B_2 = C_2$. Non-trivial performance gains can be realized by concurrently computing element-wise vector addition in a single operation, e.g. $vector_A + vector_B = vector_C$.

A key component of leveraging this type of computation is data representation. When possible data is represented as a vector or matrix and a high-level processing library, NumPy[1] Oliphant (2006), is used to handle machine level vectorization. Suc-

[1] Which in turn leverage low level libraries such as BLAS.

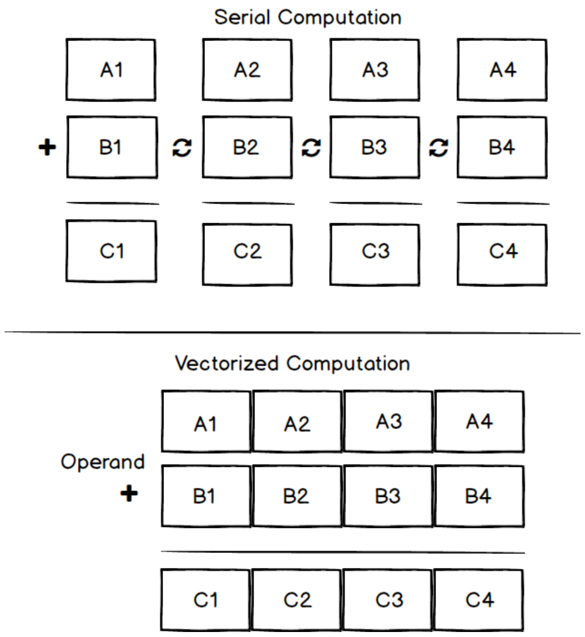


Figure 3.2: Scalar Implementation (Top) and Vectorized Implementation (Bottom) Leveraging Hardware Level Parallelism Within a Single Processing Core.

cessful vectorization imposed two requirements. First inter-operation dependencies can not be present. Using the example above, the vectorization would not be possible in use cases where the result was dependent upon a previous result, e.g. C_2 is data dependent on C_1. Second, the input data must be representable as a vector, matrix, or matrix subset, e.g. column, row, or block, as this representation matches the input that the processor is expecting.

Buzbee (1986) suggests that meaningful performance gains are achievable assuming that at least 50% of a given algorithm can be reformulated such that vectorization can be leveraged. In conjunction with the heuristic rule to identify and target processing bottlenecks for initial parallelization, it is possible to formulate, an a priori assessment of the suitability of vectorization for performance improvement. This work describes the application of vectorization to two phases of an algorithm in both serial and SMP environments.

3.3 Optimal Map Classification for Choropleth Mapping

Considerable research effort has been applied to the application, suitability, and generation of choropleth maps (Burrough and McDonnell, 1998; Brewer and Pickle, 2002; Slocum *et al.*, 2008). Choropleth maps utilize color or patterning to differentiate between areal units based upon some underlying attribute. The partitioning of attribute data for visualization is the focus of this work. Classification, or data partitioning, can take many forms including equal interval, frequency, or standard deviations from a mean (Brewer and Pickle, 2002). One popular method for data segmentation employs the Fisher-Jenks optimal classification algorithm to break data into statistically derived classes such that the variation between classes is maximized and the variation within classes is minimized. This is a non-spatial data partitioning algorithm applied to spatial data.

From a cartographic perspective, a strong case can be made against developing a highly scalable choropleth mapping algorithm as the ability for one to visualize a high number of individual observations is questionable. I suggest that this critique is potentially shortsighted for two reasons. First, with increased utilization of interactive, multi-scalar visualizations a clear need exists to support classification of a large number of observations to visualize large scale, local distributions within the context of small scale regional patterns. For example, optimal data classification allows visualization of the spatial distribution of some attribute, using US Census Tracts as the unit of observation, at both the city and national levels without the need to arbitrarily reclassify data based on sound bounding area. Second, data clustering algorithms find applications beyond map visualization, for example for the identification of optimal bins prior to the application of a machine learning algorithm. In these contexts, the ability to rapidly and optimally classify a dataset is essential.

3.3.1 Fisher-Jenks Algorithm

The Fisher-Jenks algorithm optimally classifies n observations into k classes such that all observations are members of a single class. Structured as an optimization problem, the algorithm is constrained to minimize some measure of variance within each class and maximize variance between classes. This can be the absolute sum of squares deviation from the class median or the sum of the squared deviations around the class mean (Rey *et al.*, 2013). The algorithm consists of three steps: (1) the computation of a diameter matrix which stores the sum of squares variance from the mean for all clusters, (2) the computation of an error matrix which stores the minimum variance of a set of n observations for k classes, and (3) the query of the error matrix to find those pivots which fulfill the optimization constraints (Hartigan, 1975).

1. Compute the diameter $D_{i,j}$ for all pairs of n such that $1 \leq i \leq j \leq n$. Diameter in this work is defined as the sum of squared deviations about the mean.

2. Populate each element, L, of the error matrix for rows $[2, k]$ by $E[P_{i,L}] = min(D_{1,j-1} + E[P_{j-1,L-1}])]$. This is dynamically generated as the error of the optimal partition for the current row index, $2 \leq j \leq k$, is derived from the preceding row index, $j - 1$.

3. Locate the optimal partition from the error matrix as $E[P_{n,k}] = E[P_{j-1,k-1}] + D_{j,n}$

It is possible to reduce the total number of steps to three by populating the first row of the error matrix from the first row of the diameter matrix. This is in contrast to the original publication by Hartigan (1975) and subsequent work by Rey *et al.* (2013) which describe a fourth step, occurring between steps one and two to populate the first row of the error matrix. Additionally, previous works implemented this algorithm either in serial, or through the parallelization of step one. For this chapter, both steps one and two are parallelized.

This algorithm is constrained not only by runtime, but also by memory requirements. The creation of a dense diameter matrix, step 1, requires either n^2, if storing a symmetric matrix, or $\frac{n^2}{2}$, if only storing the upper triangle, units of memory. The diameter matrix becomes a lookup table during step 2, and the algorithms requires an additional $k*n$ units of memory for error matrix storage. Therefore, total memory used is $(k*n)+\frac{n^2}{2}$.

3.4 Previous Work

Rey *et al.* (2013) implemented a parallel Fisher-Jenks algorithm using three freely available parallel Python libraries: the built-in Python module multiprocessing, Parallel Python, and PyOpenCL. Language syntax requirements differ between each library and therefore require divergent implementations. The built-in multiprocessing module ships with Python versions greater than 2.6, offers shorter development times due to more straightforward implementation requirements, and reported the best results. Parallel Python, an external library, requires an additional user installation and reported the worst parallel performance. Finally, PyOpenCL, a library designed to leverage either the Central Processing Unit (CPU) or Graphics Processing Unit (GPU), requires an additional installation step, complex implementation requirements, and returned repeated memory allocation errors (Rey *et al.*, 2013). In light of these results, this implementation utilizes multiprocessing.

In addition to implementing and testing three libraries Rey *et al.* (2013) offer multiple insights into the parallelization and implementation of the Fisher-Jenks algorithm. These insights have been utilized to drive this research to further improve algorithm portability and performance. First, in-memory duplication limited total sample size to half of the available RAM space. Second, the parallel computation of the diameter matrix improved total compute time sufficiently that the computation of the error matrix is revealed as a new bottleneck. Third, parallelization is only beneficial with medium to large sample sizes as costs associated with parallel overhead must be accounted for.

The Fisher-Jenks algorithm has an $O(n^k)$ runtime for unordered data and an $O(k^n)$ runtime for ordered data sets (Hartigan, 1975; Rey *et al.*, 2013). Additionally, the Fisher-Jenks algorithm requires either a full, nxn distance matrix or an upper triangle $((nxn)/2)$ to be stored in memory. Given the runtime, computation of medium to large dataset is infeasible in a serial ESDA environment and given the memory requirements, scalability is questionable as available RAM is a major limitation.

3.5 Implementation

The improved SMP implementation focuses on three extensions of the work by Rey *et al.* (2013). First, this work explores the ability to avoid in-memory data duplication through the use of shared memory space. Second, the computation of the diameter matrix is refactored to leverage vectorized computation. Finally, both parallelization and vectorization are applied to the computation of the error matrix. After implementing these changes a range of sample sizes (n) and classes (k) are tested and the performance results compared to both the serial Fisher-Jenks implementation and the initial parallel implementation.

CTypes Shared Memory Prior to initiating the three algorithm phases, described above, all necessary data structures are initialized. This alleviates the need for in memory data duplication (Rey *et al.*, 2013). Accomplishing this requires that two contiguous memory blocks be pre-allocated using the built-in ctypes library (van Rossum and Drake, 2013). This library provides a Pythonic interface to non-local functions and, in this usage, facilitates the access of a single globally available memory space by all child processes. In this context, in RAM storage must either be allocated at the largest possible data type, 64-bit floating point, or the input data must be sampled and the data type intelligently determined. This work utilizes the former approach. Finally, inter-dwarf data dependencies are not identified. Therefore, the ctypes allocated memory does not have accompanying locking mechanisms (locks or semaphores).

Two constraints and two benefits are introduced through the use of shared memory. First, access to shared memory using Python requires the use of pointers to a memory address; this is not direct in-language access to the stored elements. It is therefore necessary to read directly from the memory buffer. This is accomplished through the use of the frombuffer() function within NumPy (Oliphant, 2006). The second constraint requires that the buffer is stored as a flat array, (i.e. one dimensional). The Fisher-Jenks algorithm requires that the diameter matrix be nxn and the error matrix be nxk. Therefore, it is necessary to reshape the buffer view before in-language processing. While it is not possible to make a Pythonic view of the shared memory space globally available to all children on a non-POSIX system, it is possible to pass the pointer to a shared memory space and then recapture a Pythonic view without issue. In this manner, the use of ctypes facilitate OS portability in a manner that the use of global shared memory, internal to Python, would not.

The pre-allocation phase concludes with packing each row of the diameter matrix with the sorted input values. This is to facilitate the vectorization of the computation in the next phase.

Diameter Matrix Computation Generation of the diameter matrix is the most computationally expensive portion of the Fisher-Jenks algorithm and Rey *et al.* (2013) show that the parallelization of this phase provides non-trivial speed increases for medium and large problem sets. This implementation follows theirs, but differs in two ways. First, memory duplication is avoided by passing a pointer and row indices from the mother to the child process. Second, all `for` loops are removed from the code to leverage vectorization in the computation of the diameter matrix. These two general improvements provide substantial speed improvements over the initial parallel implementation and are more explicitly described below.

Diameter matrix computation is initiated with the mother process computing the load for each core using the equation

$$interval = \frac{n}{c}\ , \tag{3.1}$$

where n is the total number of values and therefore rows, and c is the number of cores. Data decomposition using this method often leaves excess rows that are processed by the first core to complete its initial load. Once the segmentation of the load is computed a memory pointer and the indices of the rows to be processed are passed to each child process. Once distributed, each core is assigned $\frac{n}{numbercores}$ rows.

Each child process then iterates over its assigned rows and computes each row using vectorized computation that fully leverages the SIMD capabilities of the processor. Due to the fact that the lower triangle of the matrix is zero, before processing a row the first i elements are replaced with zeros where i is the row number. Once preprocessing of the row is completed, computation the entire diameter matrix row proceeds with a single operation, i.e. Same Instruction Multiple Data, using the scalar equation

$$D_{I,J} = \sum_{i=I}^{J} (y_i - \bar{y}_{I,J})^2 \tag{3.2}$$

where $D_{I,J}$ is the diameter of the cluster consisting of values I through J, $\bar{y}_{I,J} = \frac{1}{J-I+1} \sum_{i=I}^{J} y_i$, and y_i is the attribute value for observation i. Each row of the matrix D is obtained through vectorization.

Once each child has completed the assigned load the jobs synchronize and the mother process initiates computation of the error matrix. In all phases the mother process manages synchronization, but also acts as a child process, performing a segment of the total load. This functionality exists within the Python language without programmer implementation.

Error Matrix Computation Unlike Rey *et al.* (2013) I also parallelize the computation of the error matrix. This is a direct extension to earlier work as computation that was sufficiently fast previously now becomes the primary processing bottleneck. The first step of this phase is to copy the top row of the diameter matrix to the top row of the error matrix. This reduces the total number of computations from k^n to $(k-1)^n + 1$.

The computation of the error matrix is decomposed differently than the diameter matrix. Instead of sending complete rows to each child process, the decomposition technique used for the diameter matrix, it is necessary to send segments of a single row to each process. This is because each row of the error matrix depends upon values from both the diameter matrix and the preceding row of the error matrix. Therefore,computation of each row is distributed over each available core using the process summarized in: in Table 3.1Segmentation of the Error Matrix over Available

Table 3.1: Segmentation of the Error Matrix over Available Cores

Core Number	Error Row Segment (Vector)					
c_1	$e_{i,1}$	$e_{i,2}$	$e_{i,3}$	$e_{i,4}$	$\cdots$	$e_{i,\frac{n}{c}}$
c_2	$e_{i,\frac{n}{c}+1}$	$e_{i,\frac{n}{c}+2}$	$e_{i,\frac{n}{c}+3}$	$e_{i,\frac{n}{c}+4}$	$\cdots$	$e_{i,2*(\frac{n}{c})}$
$\vdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
c_m	$e_{i,m*(\frac{n}{c}+1)}$	$e_{i,m*(\frac{n}{c}+2)}$	$e_{i,m*(\frac{n}{c}+3)}$	$e_{i,m*(\frac{n}{c}+4)}$	$\cdots$	$e_{i,n}$

Corestable.3.1. Here e is an element in the error row, c is a child process (core), n is the total number of values, and m is a total count of the available cores. As the row index increases the total computational load increases, but the computational load required to intelligently distribute the load exceeds the total computational cost of this phase.

Once row segmentation is computed, the mother process distributes memory pointers and row indices to each child process, as above. Next, each error element is computed as the minimum of the sum of elements from the preceding error matrix row and elements from a column of the diameter matrix. Both sequences can be represented as vectors and therefore provide a means to performed vectorized computation using the following equation

$$E_{i,j} = [e_{i-1,j-1}, \quad e_{i-1,j-2}, \quad e_{i-1,j-3}, \quad \cdots \quad e_{i-1,(j-n)+2}, \quad e_{i-1,(j-n)+1}]$$

$$D_{i,j} = [d_{i,j}, \quad d_{i+1,j}, \quad d_{i+2,j}, \quad \cdots, d_{(i+k)-1,j}, \quad d_{i+k,j}]$$

$$e_{i,j} = min(E_{i,j} + D_{i,j}) \; , \tag{3.3}$$

where $E_{i,j}$ is a vector extracted from the previous row of the error matrix, $D_{i,j}$ is a vector extracted from the diameter matrix, and $e_{i,j}$ is the minimum scalar element of $E_{i,j} + D_{i,j}$. While iteration over each index in the error matrix is still required, it is possible to leverage the SIMD capabilities of the processor to populate each error index.

Pivot Matrix Computation Finally, pivot indices or values are identified in the error matrix such that the variance is maximized between classes and minimized within classes. This is an extremely fast lookup that is performed in serial. The implementation is identical to that of Rey *et al.* (2013), except that the underlying data structure is an array instead of a list. This incurs a negligible performance hit to this implementation less than 0.5% of total compute time.

3.6 Experiment, Hardware, & Results

Below I report the results of testing the improved parallel implementation against both the implementation created by Rey *et al.* (2013) and the original serial implementation in PySAL, which mirrors Hartigan (1975). To control for hardware variation comparative results from a single machine are reported after performing a clean reboot. To test the impact of parallelization of both the diameter matrix and error matrix computation the parameters k = 5,7,9, which are reported to be the most commonly selected numbers of classes Rey *et al.* (2013) and a range of sample sizes in the set from $n = \{1000, 2000, 4000, 6000, 8000, 12000\}$. For tests including the original serial implementation the maximum value tested is $n = 8,000$ due to excessive runtimes. The test data is randomly generated floating point numbers with a range of $(0, 1]$. Finally, experiments are performed where $n = \{1000, 2000, 4000, 8000, 16000, 24000, 32000, 40000, 48000, 56000, 64000, 72000, 80000\}$ on a server level machine to extend the performance curve and explore the current upper bounds of this implementation.

Hardware The SMP test hardware consisted of an Intel 3.1GHz i3-2100 Sandy Bridge dual core processor, that reports as 4 cores due to hyper-threading Intel (2003), with 4GB of RAM, running KUbuntu linux. This is a pseudo POSIX compliant system that offers processor level process forking and mirrors a low end machine users are likely to find readily accessible. For $n > 12,000$ experiments utilize a 12-core 2.26GHz Mac Pro with 64GB of RAM.

Results A single-core, serial Fisher-Jenks implementation provides a benchmark against which it is possible to compute the total speedup attained through parallelization. Figure 3.3Piecewise Linear Speed Curve Showing Total Compute Time of the Serial Fisher-jenks Algorithmfigure.3.3 shows the piece-wise linear compute time curves generated by the serial algorithm which clearly grow with the sample size. Additionally, the number of classes increases total compute time, but this impact is small when compared to the correlation between n and t, the total compute time.

To compare the results speedup results the standard speedup curve, defined in Chapter 1, is utilized.

Figure 3.4Speedup Curve Benchmarking the Serial Implementation to This Parallel Implementationfigure.3.4 illustrates the speedup attained compared to the original serial implementation. I find that for $n > 1,000$ the overhead associated with parallelization is significantly less than the total speedup attained. This is in-line with

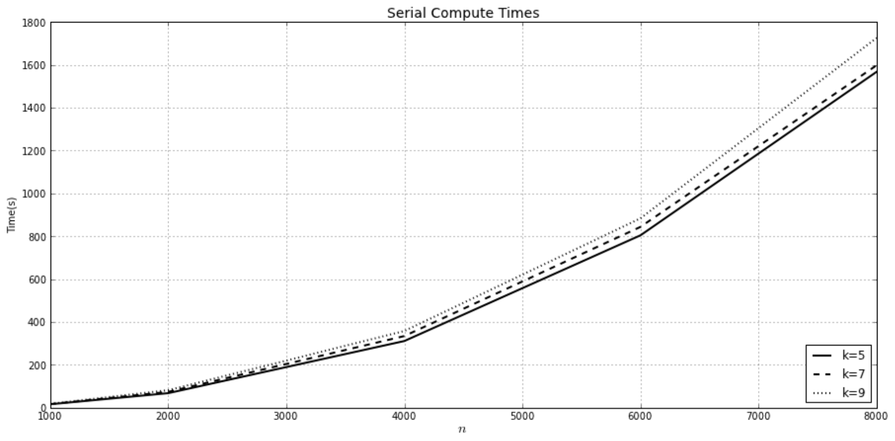


Figure 3.3: Piecewise Linear Speed Curve Showing Total Compute Time of the Serial Fisher-jenks Algorithm.

previous results (Rey *et al.*, 2013). The speedup curves are largely linear and clearly vary with k. This is expected as increases in k introduce both an additional k^n computations and the associated parallelization overhead. Unlike, Rey *et al.* (2013) a plateau is not identified at $n = 2,000$. Finally, total speedups ranging from 50 times faster to nearly 1,000 times faster are reported.

Moving to a comparison of this implementation to the previous parallel implementation (Rey *et al.*, 2013), Figure 3.5Speedup Curve Comparing the Original and Improved Parallel Implementationsfigure.3.5 shows, a general speed increase between 25 times and 200 times faster. This is largely attributable to the reduction in in-memory duplication, the use of vectorization, and the parallelization of computation for the error matrix. Interestingly, a plateau exists and overall decrease in speedup from $n = 8,000$ to $n = 12,000$. This is potentially a product of naive data decomposition for error matrix computation and additional tests comparing larger numbers of samples are required. Finally, I identify a marked improvement comparing the single core, vectorized, algorithm to the previously published multi-core algorithm (Rey *et al.*, 2013). Vectorization alone provides speedups of between twenty-five and fifty over non-vectorized multi-core implementations.

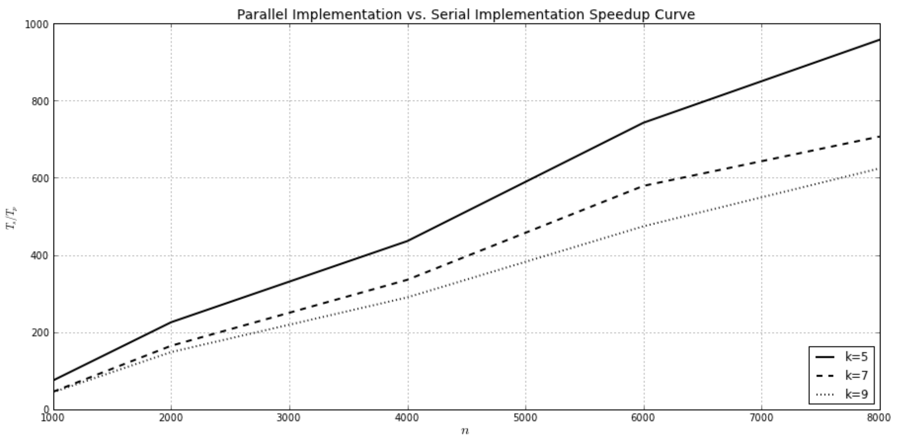


Figure 3.4: Speedup Curve Benchmarking the Serial Implementation to This Parallel Implementation.

Figure 3.6Speedup Curve Benchmarking Comparing Both Vectorized and Parallel Implementations to the Serial Implementationfigure.3.6 compares the speed gains attained by a solely vectorized implementation and the final implementation leveraging both vectorization and parallelization. Clearly the later provides greater speed increases, but the total difference between implementations is negligible until $n > 1,000$. Given the hardware specific requirements inherent to leveraging all available processing cores, and the human time required to implement a parallel implementation, I suggest that single core vectorization may provide implementations which are sufficiently fast. This must be assessed on a problem specific basis. For $n < 1,250$ vectorization out performs parallelization; this is due to the overhead associated with spawning child processes.

Figure 3.7Total Computation of the Vectorized and Parallel Implementationsfigure.3.7 depicts benchmarking performed on the server level machine to compare total computational time for the parallel and vectorized implementations. Tests were performed from $n = 1,000$ to $n = 42,000$ and show total compute time leveraging both parallelization and vectorization for large problem sets remains well under two minutes. Performance is describable using a piece-wise linear function for solely vectorized

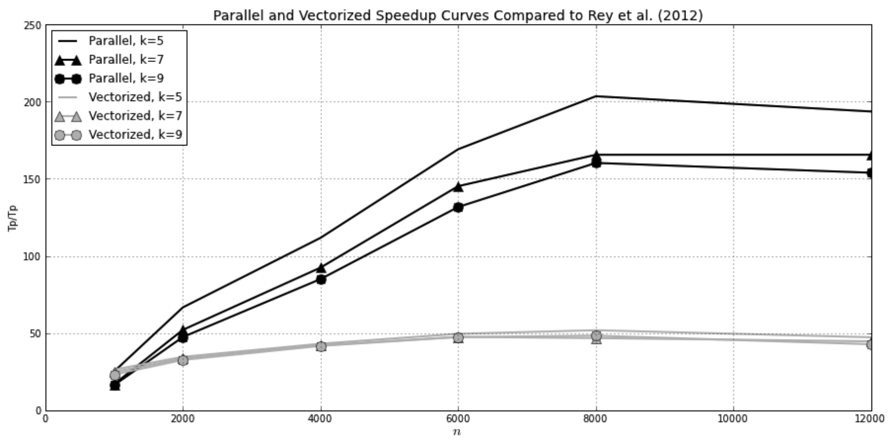


Figure 3.5: Speedup Curve Comparing the Original and Improved Parallel Implementations.

computation. Additional testing with larger values of n is required to classify the expected behavior of the parallel implementation speed curve.

Implementation Challenges

The development of an improved parallel Fisher-Jenks algorithm was an iterative process encountering multiple implementation challenges and identifying opportunities for future work. First, refactoring the original Fisher-Jenks algorithm to allow for vector representation in the computation of the diameter and error matrices was human time intensive. This required that the problem be recast and represented in a completely different structure. Second, porting this code from a POSIX to a non-POSIX system required an additional refactoring of the shared memory space and exploration of efficient means to pass access to shared memory between processes which do not exist in the same variable space (namespace).

3.7 Extensions and Future Work

This extension to Rey *et al.* (2013) highlights future research objectives and provides additional insight into deploying parallel algorithms throughout the spatial anal-

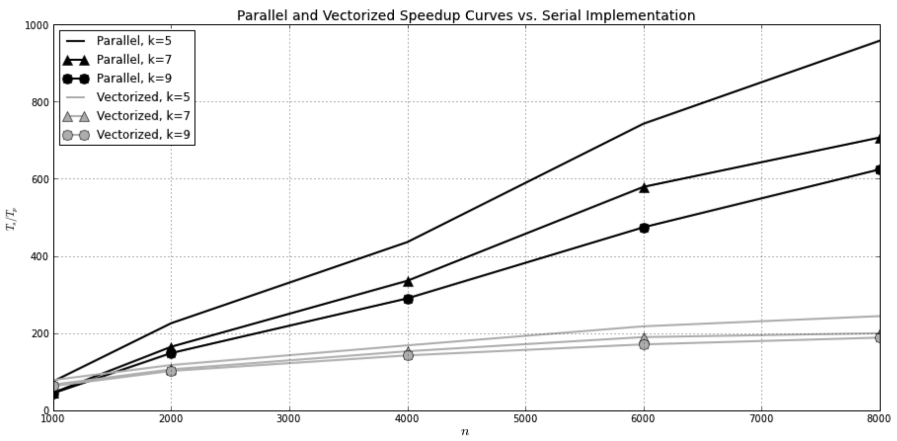


Figure 3.6: Speedup Curve Benchmarking Comparing Both Vectorized and Parallel Implementations to the Serial Implementation.

ysis stack. First, using open source, built-in libraries, it is possible to develop and deploy system agnostic, parallel, code. This requires that code be developed understanding the limitations placed by each of the three modern desktop operating systems. Second, I concur with Rey *et al.* (2013) in that speed improvements attained through parallelization are valid only for medium to large values of n. This is in-line with expectations as processing forking and inter-core communication incur an overhead that is non-trivial above a threshold. Vectorization provides a method by which increased performance can be attained for small values of n.

This work highlights the following three insights into the parallelization of this algorithm. First, the representation of data as regular arrays, when possible, is essential to providing the means by which vectorization can occur. Major performance gains are attainable using CPU level parallelization, i.e. vectorization. Second, when refactoring for a high-level parallel implementation, it is necessary to iteratively deploy code and highlight processing bottlenecks at each iteration. This is evidenced by the performance gains attained by leveraging and improving computation of the distance matrix Rey *et al.* (2013) as well as parallelizing the computation of the error matrix. Parallelization of the former led to a shit in processing bottleneck to the

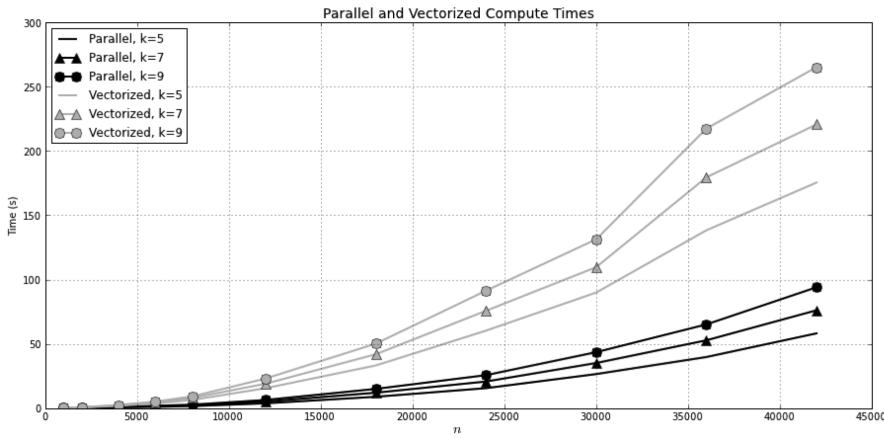


Figure 3.7: Total Computation of the Vectorized and Parallel Implementations.

latter. Third, this algorithm is still memory constrained in the SMP environment and the parallelization of spatial algorithms to improve performance must focus on both overall computational speed and efficient data representation. Additional work is forthcoming focusing on the ability to apply this algorithm to a sampled subset of the data and the impacts on accuracy. Finally, work is underway to deploy this algorithm into an HPC environment using a distributed array paradigm.

Chapter 4

GEOMETRIC AND TOPOLOGICAL SPATIAL ANALYSIS PROBLEMS

4.1 Introduction

A key research theme in the computational geometry domain is the theoretical discovery and implementation of optimal or near optimal algorithms which leverage the geometric properties of some input data (Nagy and Wagle, 1980; McAllister, 1999). Interest in parallel computational geometry has a long history with Arjomandi (1975); Hirschberg (1976); Savage (1978) and Eckstein (1977) all describing opportunities for parallelism in graph-based representations, an intrinsically geometric problem. Later, Chow (1980), in a Ph.D. thesis articulates parallel methods for rectangle intersection, planar nearest neighbor search, two-dimensional convex hull, and planar Voronoi diagram generation. Here the first obvious linkages between GIScience and computational geometry can be made. Coupled with earlier successes, increased availability of hardware capable of parallelism heralded a flurry of additional work with Aggarwal *et al.* (1988); Stojmenovic and Evans (1987); Atallah and Goodrich (1984, 1985); Blelloch and Little (1988); Akman *et al.* (1989)describing parallel geometric and topological operations across a range of theoretical parallel computing models and implementing said operations in parallel computing environments. With the growth of GISystems and identification of computational geometry problems with high compute costs, cross domain interest naturally increased.

Within the context of vector spatial analysis I identify wide utilization of neighborhood search methods such as the G or G variant statistics (Armstrong *et al.*, 1993; Armstrong and Marciano, 1995; Wang *et al.*, 2008), Inverse Distance Weighted

(IDW) interpolation (Armstrong and Marciano, 1996), Kriging (Hawick *et al.*, 2003) and nearest neighbor analysis (Barbini *et al.*, 1996). These implementations leverage both *geometric* processing dwarfs, with the utilization of global sort methods as a key processing component (Armstrong and Marciano, 1996), or *topological* dwarfs with the generation and utilization of a topological tree data structure, e.g. (Wang *et al.*, 2008). Overlay and intersection analysis are also widely utilized within GISystems and I identify both geometrically and topologically classifiable implementations within both the GIScience and parallel computation geometry domains (Atallah and Goodrich, 1985; Akman *et al.*, 1989; Harding *et al.*, 1998; Hoel and Samet, 2003; Puri and Prasad, 2014).

In Chapter 2 I asserted that the key distinction between *geometric* and *topological* processing dwarfs is that of the quantity and complexity of information which must be communicated between processing cores with *geometric* dwarfs leveraging global sorting methods and *topological* dwarfs requiring the transmission of a potentially complex topological data structure, e.g. a tree, graph, or network. Previous works illustrate the application of both approaches in answering similar questions through the alteration of data structures, i.e. representations of the observed spatial objects. Two key components in the selection of the implementation methods (the dwarf) are the efficiency of the decomposition such that data dependency is limited and the efficiency with which that data structure can be communicated.

The remainder of this chapter is organized as follows. In Section 4.2 I describe the processes of domain decomposition. Section 4.3 introduces the targeted problem domains, nearest neighbor search and polygon adjacency. In Section 4.4 I describe a number of different decomposition algorithms. Section 4.5 describes the experiments performed and Section 4.6 reports these results. In Sections 4.7 and 4.8 I offer a discussion comparing the geometric and topological implementations and summarize this work, respectively.

4.2 Domain Decomposition

Domain decomposition is the act of identifying boundaries at which logical partitioning can be used to allow for more efficient processing. By way of example, imagine a randomly distributed point pattern, P in a planar space with the analytical goal of identifying all nearest neighbors (formulated below). In the naive case, the distance between p_i and all other points $p_j \in P \forall j \neq i$ is computed for each i. Asymptotically, this algorithm scales quadratically ($O(n^2)$). Some decomposition method can be applied such that a subset of the points can be compared and logic applied to handle potential decomposition boundary conditions. Assuming an optimal computational handling of boundary conditions, the constant time cost, n can be significantly reduced, resulting in a faster, but potentially non-optimal algorithm. [1] In a parallel environment, significant performance improvements can be achieved by leveraging the realization that non-data dependent decomposition can yield performance on the order of $O(\frac{n}{p})$ plus some communication constant (e.g. $O(\frac{n\ log\ n}{p})$) in the case of global sort (Dehne *et al.*, 1996).

The act of domain decomposition then becomes a preprocessing step, during which the data is converted from its storage form into a form amenable to more robust computation (Nagy and Wagle, 1980). The method of decomposition is invariant to the type of processing (e.g. serial or parallel). That is the approaches described below can, in some form, be applied in both the serial and parallel cases. What is of consequence is the cost of decomposition, whether added to the total algorithm computation time

[1] I intentionally consider this case due to the simplicity of description and to illustrate the performance improvement attained through the reduction in n. Asymptotically, these implementations perform identically.

as in the case of a one-off processing effort or amortized over many applications of analytics as is generally realized in spatial databases using indexing structures (Nagy and Wagle, 1980). Waugh (1986) vigorously questions the application of a more complex decomposition method, the quad-tree, within GISystems. He suggests that the advice of Nagy and Wagle (1980) must be heeded and that the wholesale use of an elegant decomposition method without a careful assessment of the computational cost is foolhardy. Akman *et al.* (1989) address this issue, within the computational geometry domain, and asserts that a regular, gridded decomposition, as describes in the above example, significantly outperforms the single use of a quad-tree decomposition as long as the spatial density of the geometries in question are within an order of magnitude of each other. Later, Armstrong and Densham (1992) identifies the minimization of overhead as a second criteria in parallel decomposition, [2] suggesting that methods have been tested and discarded with high decomposition costs.

The level of complexity, in generating and representing different domain decompositions maps well to *geometric* and *topological* dwarfs. For example, more complex decompositions are naturally representable using a graph structure. The generation, communication, and query of that structure maps well the *topological* dwarf. In contrast, regular decomposition methods do not require a globally accessible data structure and map well to the *geometric* dwarf. This chapter explores planar nearest neighbor search and planar polygon adjacency as two common use cases where classification as either *topological* or *geometric* is possible. This work explores the total, aggregate compute cost of each algorithm using different domain decomposition methods in parallel environments.

[2] Load balancing is the primary criteria.

4.3 Problem Domains

4.3.1 Planar Nearest Neighbor Search

Given a set of P points embedded in a plane, $\Re^2$, the all-nearest neighbor problem or all Nearest Neighbor Search (NNS) is to identify, $\forall p_i \in P$ the p_j with the minimum distances. The closets pairwise points are defined as those points where $Distance(p_i, p_j) \leq Distance(p_i, p_k) \forall p_k \in P \backslash i$ (Dehne *et al.*, 1996). In the case of this work Euclidean distance is used though another distance metric could be substituted. [3]

Within the spatial analysis stack, all nearest neighbor computations find wide utilization in point pattern analysis. For example, Clark and Evans (1954), introduce a mean nearest neighbor distance metric in the form:

$$\bar{\Gamma}_N = \frac{\sum_{i=1}^{n} \Gamma_i}{N}, \tag{4.1}$$

where, N is a set of all point observations and Γ is the minimum distance between N_i and N_j. Extending this approach, the F and G functions attempt to provide a more descriptive metric than a single scalar value (O'Sullivan and Unwin, 2010a, pg. 132). These statistics utilize some measure of nearest neighbor distance. For example, $G(d)$ can be formulated as:

$$G(d) = \frac{Count(Distance(N_i < d)}{N}, \tag{4.2}$$

where $Count(Distance(N_{i,j}))$ is the scalar count of all observations with nearest neighbor distances less than some distance threshold, d. By selecting a range of d it is

[3] The utilization of network constrained distance shifts this dwarf from being *geometric* to being *topological*. I focus on the former case.

possible to compute a function describing the ever increasing number of observations less than the current threshold. Likewise, $F(d)$ can be formulated as

$$F(d) = \frac{Count(Distance(P_i, N) < d}{m}, \tag{4.3}$$

where P is a set of a m randomly selected points within the study area, N is the set of observations, d is a distance threshold, and $Count(Distance(P_i, S)$ is a scalar count of all points fulfilling the distance threshold condition (O'Sullivan and Unwin, 2010a).

While the general shape of $G(d)$ and $F(d)$ are of interest (O'Sullivan and Unwin, 2010a), tests of significance are essential in identifying patterns which deviate from the null hypothesis, Complete Spatial Randomness (CSR). [4] In this case, Monte Carlo simulation can be utilized to generate a high number of point patterns and compute significance envelopes. Within the context of this work, this simulation is important for two reasons. First, Monte Carlo simulation highlights a point of *MapReduce* style parallelization where n simulations can be distributed over p processing cores. Secondly, the computation of $Min(D_{i,j})$ must be as inexpensive as possible due to the potential number of repeated applications.

4.3.2 Polygon Adjacency

Given a set of potentially conterminous geometries, G, embedded in a plane $\Re^2$, the polygon adjacency problem seeks to identify $\forall g_{i,j} \in G, i \neq j$ those $g_{i,j}$ with a shared vertex or a shared edge.

[4] I note that for the mean nearest neighbor metric, the R test Clark and Evans (1954) can be utilized without Monte Carlo simulation.

Within the context of spatial analysis and spatial econometrics the topology of irregularly shaped and distributed observational units plays an essential role in modeling underlying processes (Anselin, 1988). The concept of first and higher order spatial adjacency finds application in tests for global spatial autocorrelation, e.g., Moran's I (Anselin and Smirnov, 1996), spatial regression models (Anselin, 1988; Ward and Gleditsch, 2007) and spatially constrained regionalization models (Duque *et al.*, 2012). The execution of these aforementioned spatial analytical techniques requires the generation of some representation of the underlying connectivity of the observational (polygon) units.

A spatial weights object or weights matrix, W, is an adjacency matrix[5] that represents potential interaction between each i, j within a given study area of n spatial units. Within the context of spatial analysis, the interaction between observational units is generally defined as either binary, $w_{i,j} = 0, 1$, depending on whether or not i and j are considered neighbors, or a continuous value reflecting some general distance relationship, e.g. inverse distance weighted, between observations i and j.

This work focuses on binary, and not distance or kernel, weights where the adjacency criteria requires either a shared vertex (Queen case) or a shared edge (Rook case). Using regular lattice data, Figure 4.1Rook and Queen Contiguity Criteria, Where the Light Gray Geometry Is the Current i^{th} Element and the Dark Gray Geometries Are All j Considered Neighborsfigure.4.1 illustrates these two adjacency criteria. In the Queen case implementation is in line with expectations, i.e. a single shared vertex is sufficient to assign adjacency. The Rook case, adjacency is more complex and two shared vertices are a necessary, but not sufficient threshold to assert adjacency, i.e. a queen case implementation with a counter for the number of shared vertices. Full geometry edges must be compared as it is feasible that two shared

[5] or list of lists or adjacency hash table.

vertices do not indicate a shared edge. For example, a crescent geometry can share two vertices with another geometry but fail to share an edge as another, interceding geometry is present.

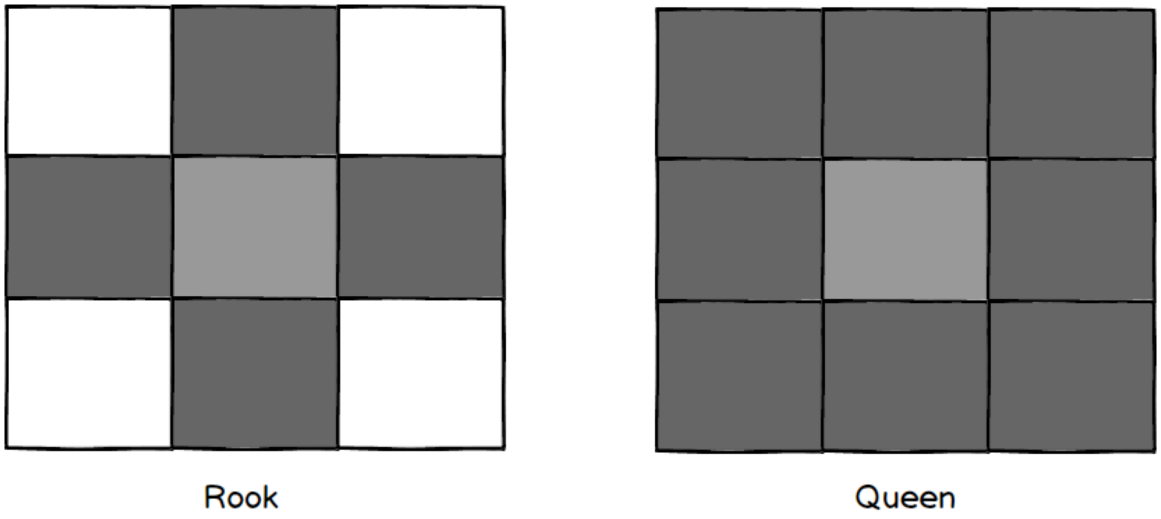


Figure 4.1: Rook and Queen Contiguity Criteria, Where the Light Gray Geometry Is the Current i^{th} Element and the Dark Gray Geometries Are All j Considered Neighbors.

The population of an adjacency list, A, or adjacency matrix must identify all polygon geometries which are conterminous. The definition of adjacent is dependent upon the type of adjacency matrix to be generated. Each adjacency algorithm requires a list of polygon geometries, L, composed of sublists of vertices, $L = [p_1, p_2, \ldots, p_n]$. Traditionally, the vertices composing each polygon, p_i, are stored in a fixed winding order (clockwise or counter-clockwise) and share a common origin-termination vertex, $p_i = [v_1, v_2, v_3, \ldots, v_1]$. This latter constrain facilitates differentiation between a polygon and polyline.

The computation of a spatial adjacency structure is most frequently a precursor to more complex process models, i.e. a pre-processing step. This processing step occurs dynamically, i.e. the data is not loaded into a spatial database where efficient indexing structures can be pre-generated. Therefore, the computational cost of generating these data structures is often overlooked in the assessment of global algorithmic performance.

Spatial adjacency is one example of an algorithm that does not scale to large observation counts due to the complexity of the underlying algorithm. For example, a key requirement of Exploratory Spatial Data Analysis (ESDA) (Anselin, 1996) is the rapid computation and visualization of some spatially defined measures, e.g. Local Moran's I. Within the Python Spatial Analysis Library (PySAL), the computation of a local indicator of spatial autocorrelation utilizes binary adjacency in computing Local Moran's I as a means to identify the cardinality of each observation. In a small data environment ($n < 3000$) a naive implementation is sufficiently performant, but as the resolution of the observational unit increases (a move from U.S. counties or county equivalents to census tracts) compute time increases non-linearly. When combined with the compute cost to perform the primary analytical technique, and potential network transmission costs in a Web based environment, ESDA at medium to large data sizes is infeasible.

Scaling to even larger observation counts where longer runtimes are expected, heursitically solved regionalization models, e.g., Max-P-Regions (Duque *et al.*, 2012), require that a spatial contiguity constraint be enforced. In large data setting, where a high number of concurrent heuristic searches are to be performed, the computation of adjacency can be a serial processing bottleneck. Improved adjacency metrics are required within this domain for two reasons. First, in a distributed environment with shared resources, reduction of pre-processing directly correlates with time available for analysis. Using heuristic search methods this translates to additional time available to search a solution space and potential identify a maxima. Second, the scale at which regionalization is initiated is an essential decision in research design as underlying attribute data or processes may only manifest at some limited scale range. Therefore, a significant bottleneck in adjacency computation can render the primary analytical task infeasible.

The previous discussion is important in highlighting the composite nature of computational dwarfs, where a *geometric* or *topological* dwarf can be leveraged as a component of a larger workflow composed of multiple difference computational dwarfs. It is through this aggregate approach that workflows will achieve higher scalability.

4.4 Decomposition Algorithms

Both all nearest neighbor distances and polygon adjacency algorithms can leverage a set of common decomposition algorithms. Assume a naive linear implementation, described below, requiring $O(n^2)$ computations, the goal of decomposition is to perform some number of local $O(n^2)$ computations such that a significant reduction in n, the number of geometric comparisons required, is achieved. That is n_{local} seeks to be a small subset of N_{global}. The scaling is still quadratic, but the significant reduction in the scalar number of computations, n, results in a more performant algorithm. This decomposition can be realized through space partitioning, location aware hashing for bin assignment (Leskovec *et al.*, 2014), and graph based representations (Malkov *et al.*, 2014). I focus on exact solutions to NNS[6] and therefore constrain the exploration of decomposition methods to those which leverage a space partitioning approach. Below four decomposition methods are described: (1) a naive linear implementation, (2) serial and parallel grid based spatial binning approaches, (3) tree based approaches and (4) sort based methods using column and row major decompositions.

[6] I note that approximate solutions, e.g. approximate KDTree decompositions and search, offer high solution quality at low compute costs and in those instances where a non-exact solution is viable, can significantly outperform the methods described here.

4.4.1 Naive Approach

A naive linear approach requires the comparison of each vertex to each other vertex, in the case of NNS and polygon adjacency using queen contiguity, or the comparison of each edge, defined by a pair of vertices, in the case of polygon adjacency using the rook contiguity criteria. The requisite comparison is accomplished by iterating over a data structure of input geometries, removing the first geometry from the data structure, and then applying some distance operand or likeness comparator to all remaining geometries. Conceptually, this is the population of a symmetric matrix, where the entries could be distance or a boolean adjacency indicator. This algorithm is $O(n^2)$ as each input vertex or edge is compared against each remaining, unchecked vertex or edge. As described above, in instances where n is sufficiently small, this approach can sufficiently performant.

4.4.2 Spatial Binning

Binning seeks to leverage the spatial distribution of either a point pattern or lattice data structure to reduce the total number of pairwise geometric comparisons. I identify two broad classes of spatial binning approaches: (1) static and (2) adaptive. This section focuses on the former.

Static binning defines both the size and shape of each bin without assessment of the underlying data distribution and seeks to assign membership of a given observation to one or more grid cells(bins). Geometric decomposition using row major, column major or grid overlay geometries, Figure 4.2Column Major, Row Major, and Gridded Decompositions of a Point Pattern. These All Employ a Simple Spatial Binning Approachfigure.4.2 is applied to a given point, line, or lattice data set and membership within each grid cell assigned. This is a single layer approach as a scale relationship between grid cells does not exist and each grid cell partitions the space exclusively. Once decomposed both the operand to be applied within the bin, e.g. the geometric comparison and the operand used to catch edge cases must be applied. Imagine a binned NNS problem. It is essential that not only a local NNS within a grid cell be performed, but also a sub-global[7] NNS which includes candidate points external to the current grid cell. This latter case requires an additional layer of metadata.

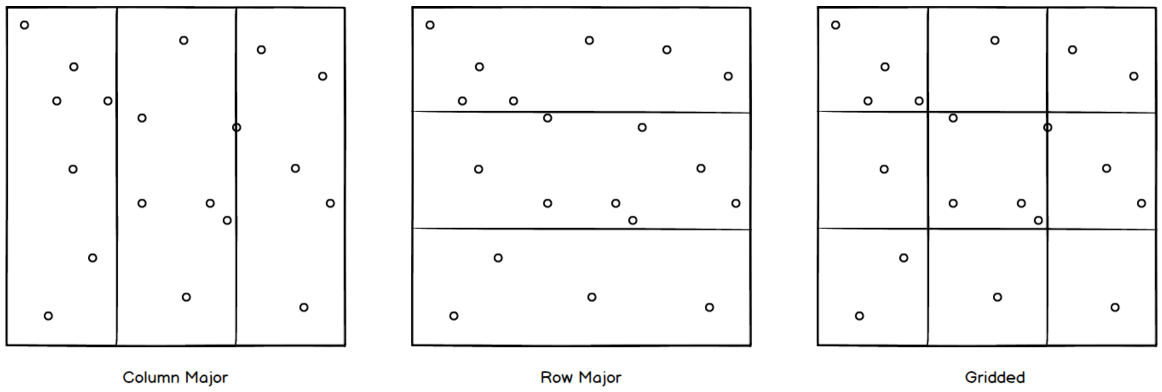


Figure 4.2: Column Major, Row Major, and Gridded Decompositions of a Point Pattern. These All Employ a Simple Spatial Binning Approach.

The primary advantage of spatial binning over the naive linear approach is the reduction in the total number of geometric checks to be performed. Full enumeration of the local NNS occurs only within a grid cell and some logic controlling the management of boundary crossers can be applied to perform a minimal number of intra-cell comparisons.

Parallel Spatial Binning

The primary computational cost to both the NNS and polygon adjacency algorithms is that of independent pairwise comparisons. Therefore, one would anticipate that a MapReduce style implementation, with sufficient logic to handle edge cases, should be

[7] I use sub-global to indicate a search with larger extent than a single grid cell which has not degenerated to full enumeration, a naive linear search.

both efficient to develop and afford robust performance. In practice, the complexity of the geometric representation and simplicity of the comparator yields implementations with *worse* performance than an efficient serial implementation. Significant disparity exists between the costs of communication, both in the distribution of the geometries and aggregation of suboptimal solutions, and the costs of pairwise geometric comparison.

Trees

In contrast to static binning, adaptive binning is a hierarchal approach that utilizes recursive space decomposition (Samet, 1984) to create a traversable, tree based representation. The rules governing balancing and decomposition criteria vary with the input data representation, but the overall logic remains consistent.

Using a classic example, illustrated in Figure 4.3Spatial Decomposition and Resultant Topological Data Structure Using a Quad-tree Decompositionfigure.4.3, assume a point data set containing some marked point data set, i.e. North American cities. Given the goal of performing a fast NNS, it is possible to recursively decompose cities based on their 2d distribution. First, select a city as the root node. The 2d point coordinates are used to bisect the space into four quadrants, e.g. NE, NW, SE, SW. For each newly defined quadrant one of three potential scenarios are present: (1) if no points are present, the preceding node is a leaf node and no further decomposition is applied, (2) a single point if found within the quadrant, that point is identified as a final leaf node, and the preceding point is classified as an intermediary node, or (3) multiple points are present in the quadrant and the process of decomposition is recursively continued. Once generated fast graph traversal can be applied to query the tree. This approach suffers from the curse of dimensionality (Marimont and Shapiro, 1979) and therefore methods such as the KD-Tree, which partitions space using a k-1 dimension axis orthogonal hyperplane can be applied (Maneewongvatana and Mount, 1999).

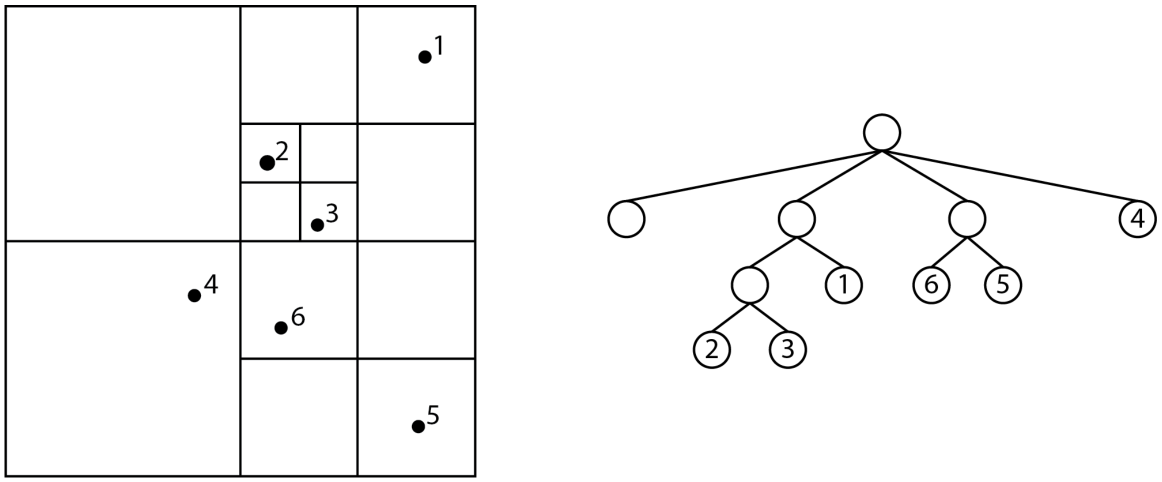


Figure 4.3: Spatial Decomposition and Resultant Topological Data Structure Using a Quad-tree Decomposition.

For the polygon case, a similar decomposition method, the R-Tree can be utilized (Gutman, 1984). An R-Tree stores aggregated groups of Minimum Bounding Rectangles (MBRs) at sequentially finer spatial resolutions. At the root node, a single MBR encompasses the MBRs of all geometries. As the tree is traversed depth-wise the global space is decomposed and each node represents a progressively smaller number subset of geometries. When generating an R-Tree two key considerations are the maximum size of each node and the method used to split a node into sub-nodes. An R-Tree query uses a depth-first search to traverse the tree and identify those MBRs which intersect the provided MBR. For example, assume that geometry A has an MBR of A_{MBR}. An R-Tree query begins at level 0 and steps down only those branches which could contain or intersect A_{MBR}.

I identify two potential disadvantages to tree based representations in a parallel environment: (1) the cost of generation for one time use and (2) the cost of communication of the data structure in a distributed memory environment. For example, in the case of the R-Tree the computation of MBRs for each geometry, the recursive space decomposition with the potential addition of a balanced tree criteria, and the distribution of said topological tree structure to all nodes can incur high computational costs.

4.4.3 Hybrid Approaches

Each of the preceding algorithms, save the naive approach, leverages a decomposition strategy to improve performance. Even with decomposition, the inter-cell or inter-MBR computation is still $O(n^2)$. Combined with the cost to generate intermediary data structures required to capture the decomposition, it is possible to leverage a higher number of lower cost operations and robust error checking to significantly improve performance. This section describes three different hybrid approaches: (1) a serial implementation using high performance containers and set operations, (2) a scalable NNS search for cluster architectures that utilizes parallel sorting and piecewise spatial decomposition, and (3) a hybrid method to support scalable, parallel spatial adjacency using queen contiguity.

High-Performance Containers and Set Operations

High-performance containers can be defined as those data structures which have been highly optimized to perform a single or small set of operations with the highest possible efficiency. Generally, utilization of the data structure for other operations is possible, but the major performance improvements, implemented at a lower level, are not realized. The following describes the use of high-performance containers in the context of Queen case polygon adjacency though the underlying logic also holds for Rook case contiguity. This approach is not suitable for NNS without some modification, which is not discussed here. [8]

At the heart of the serial polygon adjacency approach is the hash table (dictionary). A dictionary is composed of key, value pair entries, where the key is accessible via an average $O(1)$ lookup cost. The value can take a myriad of forms, for example a tuple of vertex coordinates, or a list of adjacent polygons. Additionally, the implementation utilizes sets, a data structure and set of accompanying methods that mirror what one would anticipate when requiring standard mathematical set operations, e.g. union or difference. The implementation leverages $O(length(set_a) + length(set_b))$ set unions.

In implementation, the algorithm utilizes a hashtable where the key is the vertex coordinate and the value is a set of those polygon identifiers which contain that vertex (Queen case). Stepping over an input data source, this data structure is iteratively populated. he implementation assumes serial I/O, whether from a binary shapefile or a streaming data source. For each polygon geometry, the vertex list is truncated such that the final vertex is ignored, knowing that standard polygon encoding is a counter clockwise wind order with duplicate first and final vertices. In the case of multi-polygons, each component part is treated independently, but tagged with the same polygon identifier. This method does not currently support holes, though a wind order check could be employed. For each geometry, the algorithm steps over each vertex and populates said hashtable with the key being the vertex, if not already present, and the value being the set addition of the current polygon identifier and any existing polygon identifiers. Once this data structure is generated, the algorithm creates another dictionary of sets where the key is a polygon identifier and the value is

[8] The utilization of hash based approaches whereby the coordinates are hashed and stored in high-performance containers are one possible realization of this approach using high-performance containers.

a set of those polygons which are adjacent. Stepping over the previous dictionary, the algorithm iterates over the value, a set of neighbors, and populates a new dictionary of sets which are keyed to the polygon identifiers. This yields a dictionary with keys that are polygon ids and values which are sets of neighbors. I define this as a two step algorithm due to the two outer for loops. This algorithm requires two steps:

1. While looping over the input stream off geometries, loop over each vertex defining a given geometry and pack a hash table with the key being the vertex coordinates and the value being an identifier of the parent geometry. A set union is performed such that duplicate entries are not found within the value of the hash table.

2. While looping over the values in the previously computed hash table, loop over each set of neighbors and pack a W Object where the key is the current polygon ID and the value is the set union of all other members of the neighbor set.

The Rook case is largely identical with the initial vertex dictionary being keyed by shared edges (pairs of vertices) instead of single vertices.

Parallel Global Sort

The preceding approach depends upon the streaming ingestion of input geometries to support polygon adjacency. This method does not support NNS as no guarantee that the read order correlates to a given neighborhood exists, i.e. the order in which points are stored and read from the dataset do not also contain implicit information about other near points, i.e. topology. Dehne *et al.* (1996) introduce a parallel NNS designed to use high-performance containers in conjunction with global sorting with an $O(n\ log\ n)$ runtime. I first describe the logic behind a parallel global sort and then, using parallel global sort describe both the Dehne *et al.* (1996) implementation and the modification to support polygon adjacency.

A parallel quicksort implementation is used for all sorting. Quicksort is a recursive divide and conquer style algorithm with average time $O(n\ log\ n)$ performance (Cormen *et al.*, 2001, pg. 170). This performance is inline with other efficient sorting algorithms such as mergesort or heapsort. Quicksort sorts in-place, requiring just $O(n)$ memory. The quicksort algorithm works by partitioning the input data array into two subarrays at a given pivot point. That is, each entry in the input array is classified as being larger than, or smaller than the given pivot point and the position of the data within the array is updated to be contiguous with other similarly classified values. Once all values are partitioned, the process is recursively applied to the two partitioned arrays. This process continues until the entirety of the data is sorted (Cormen *et al.*, 2001)

The algorithm for parallel distributed memory quicksort can be expressed via pseudo code as:

1. Perform either a serial read to a root node or a distributed read to all nodes such that the entirety of the unsorted data is stored in memory.

2. Apply a local quicksort to the data local to the node (in the distributed case) or scatter the data and then perform a local quicksort.

3. Stochastically sample or regularly select a sample of pivot values. These are the boundary values used to break the data into p distinct groups, where p is the number of processing cores. Regular sampling affords better load balancing.

4. Gather all $p - 1$ pivot values to a managing process yielding $p * (p - 1)$ pivots values. Sort said pivots and select a regular or random sampling of pivot values with a total size of $p - 1$.

5. Broadcast the pivots to all p and partition the local data into k classes, where $k = p$

6. Using point to point communication gather all data elements in k_i to process p_i.

7. Apply quicksort to the local data, yielding k sorted data vectors distributed over p processes.

The selection of pivot values is an essential component in the performance of the algorithm. Ideally, pivot values are distributed across the data vector such that an idealized decomposition is realized, i.e. $\frac{n}{p}$, where n is the number of observations. To achieve this decomposition the pivots can be statically or adaptively selected, much like spatial decomposition methods. In the static case, the data can be classified, for example into quantiles, and the identified break points set as pivots. In the latter case, the data can be regularly sampled and from this subset pivots can be drawn. Endogenous, adaptive selection of pivot values is advantageous for two reasons. First, unlike adaptive spatial decomposition, regular sampling from a sorted vector requires constant runtime. Second, adaptive pivot select renders this method invariant to data density issues, with the caveat that data sets with a significant number of coincident entries can cause balancing issues.

Utilizing parallel quicksort, it is possible to perform a parallel NNS as outlined by Dehne *et al.* (1996):

1. Globally sort the point coordinates by the y-axis value, yielding a sorted subset H_i bounded by two horizontal split lines. Perform a standard NNS $\forall v \in H_i$ and cache the results.

2. Globally broadcast all horizontal split lines to all p processors.

3. Globally sort the point coordinates by the x-axis, yielding a sorted subset V_j bounded by two vertical split lines. Perform a standard NNS $\forall v \in V_j$ and cache the results.

4. Using the horizontal split lines broadcast in step 2 and the vertical split lines defined in step 3, compute all horizontal and vertical intersection points, I_{ij}.

5. Using I_{ij} and the locally stored V_i points, compute C_{ij} as all points in V_j, not in H_i which are closer to a member of the set I_{ij}. This check identifies those points which are closer to a member of I_{ij} than to any nearest neighbor in the vertical direction. These are the potential edge case points which are not checked in either the horizontal or vertical decomposition directions.

6. Using point-to-point communication, collect and union all C_{ij}, where $j = p_i$, yielding C_i, to the appropriate processing core and apply the standard NNS to all points within the set $H_i \cup C_i$. Cache the results.

7. Each process now stores three potential nearest neighbors for each point in the initial distribution: (1) nearest neighbor in the horizontal decomposition, (2) nearest neighbors in the vertical decomposition, and (3) edge case nearest neighbor not defined in the previous two decomposition. A linear search for the minimum distance value for each point can be performed to identify the nearest neighbor.

First, the algorithm first identifies candidate nearest neighbors in the horizontal direction. Next candidate nearest neighbor points are identified in the vertical direction. The computation of the intersection coordinates, I_{ij}, and subsequent enumeration of those observation points nearer to a member of I_{ij} than another neighbors ensures that edge cases, where a nearest neighbor is neither a member of the vertical nor a member of the horizontal decomposition, are found.

Dehne *et al.* (1996) first introduced the above algorithm and within the context of this work, the algorithm has been ported from the original hardware to a MPI based environment. The algorithm has been implemented and tested in the original form and the contribution of this work is an update for deployment on modern hardware and testing of the algorithm within the context of the proposed taxonomy. I note that the NNS algorithm inspired development of the polygon adjacency algorithm.

Using the NNS implementation as a springboard, I develop a polygon adjacency algorithm using similar decomposition and sort based logic. This approach leverages high-performance data structures, initially developed for the serial case, in conjunction with the global sorting method utilized by Dehne *et al.* (1996).

1. Load the polygon geometries into a $nx3$ matrix where the first element is a vertex x-coordinate, the second element is a vertex y-coordinate and the final element is an identifier matching the coordinates to a given geometry. This can happen on a single processing core or over a distributed memory space.

2. Perform a local lexicographic sort using the x and y dimension.

3. Identify all coincident points within the locally sorted data and populate a local adjacency object.

4. Gather all local W objects to a single processing core and concatenate into a global adjacency object.

Figure 4.4Three Phases of a Geometric, Parallel NNSfigure.4.4 illustrates the horizontal decomposition prior to sorting, Phase I, the vertical decomposition prior to sorting, Phase II, and the identification of both I_{ij}, cyan triangles, and one member of

C_{ij}, red circle. Three NNS are performed using information from all three processing phases.

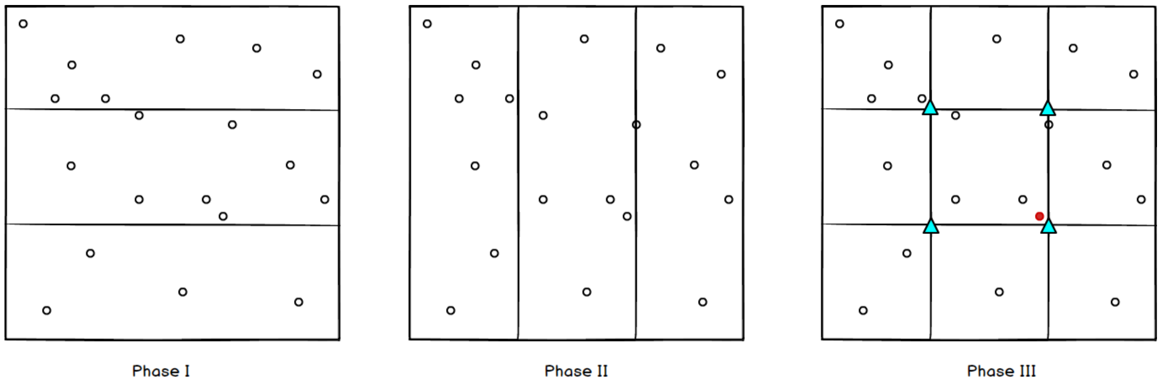


Figure 4.4: Three Phases of a Geometric, Parallel NNS.

4.5 Experiments

I perform classic scaling experiments using synthetic data sets and a range of processing core / node combinations. NNS experiments utilize 1, 8, 16, 32, and 64 processing cores in an HPC environment. The single core tests utilize the naive linear approach and I assert that the 8 core, single node tests approximate the performance of the algorithm in an SMP environment assuming that a single shared memory shape is not used. Both a parallel list based approach and a parallel KD-Tree approach are tested. The parallel KD-Tree approach computes the tree data structure in serial and communicates said data structure for distributed local query. For both cases, the nearest neighbor data to a single process is not reaggregated; the data remains distributed across all processing cores. For polygon adjacency in an SMP environment I compare a serial R-Tree decomposition, a regular gridded decomposition, and a hybrid list based approach. In the HPC environment, a parallel list based algorithm is compared to a parallel R-Tree algorithm. As with the above KD-Tree, the tree structure is generated in serial, distributed to all processing cores, and concurrently queried. In contrast to the tree based NNS, the distributed local solutions are aggregated to generate a single adjacency object.

These experiments are designed to compare implementation of these processing workflows as a *geometric* dwarf, e.g. for sort and hybrid list based methods, and topological dwarfs, e.g. tree based approaches where the entirety of the tree is communicated.

All SMP tests were performed on a 3.1 Ghz, dual core Intel i3-2100 machine with 4GB of RAM. The HPC environment is composed of up to 8 homogeneous nodes with dual quad core Intel Xeon processors with 16GB of RAM, and communication via high speed infiniband network.

All data used is synthetically generated to control for clustering as well as vertex count, edge count, and average neighbor cardinality, in the case of polygon adjacency. Randomly distributed and clustered datasets are used for all tests ranging in the size from 640 points to 1,146,880 points. [9] For polygon adjacency tests, regularly tessellating, randomly distributed, and clustered synthetic data ranging in size from 1024 geometries to 262,144 geometries[10] are used. In the SMP testing, the 4096 hexagon lattice is densified to test the impact of increased vertex count as the number of edges remains static.

4.6 Results

4.6.1 All Nearest Neighbors

NNS is tested in an HPC environment, using a single core implementation as a baseline. Figure 4.5Total Compute times Using Parallel Sort Based Methods Show-

[9] 640,1280, 2560, 5120, 10240, 20480, 81920, 163840, 327680, 491520, 655360, 819200, 983040, 1146880 points.

[10] $32, 64, 128, 160, 192, 256, 288, 320, 384, 448, 512$ geometries squared.

ing Quadratic Time Growthfigure.4.5 illustrates significant reduction in total compute time as additional processing cores are utilized. As anticipated, the algorithm still performs quadratically due to the local $O(n^2)$ geometric comparisons. Varying line lengths are a function of the five hour limit placed on all processing. Serial computation completes for up to 10,240 point observations (blue), while the 64 core, parallel implementation completes for the total range of test data, up to 1,146,880 points in approximately 2.5 hours.

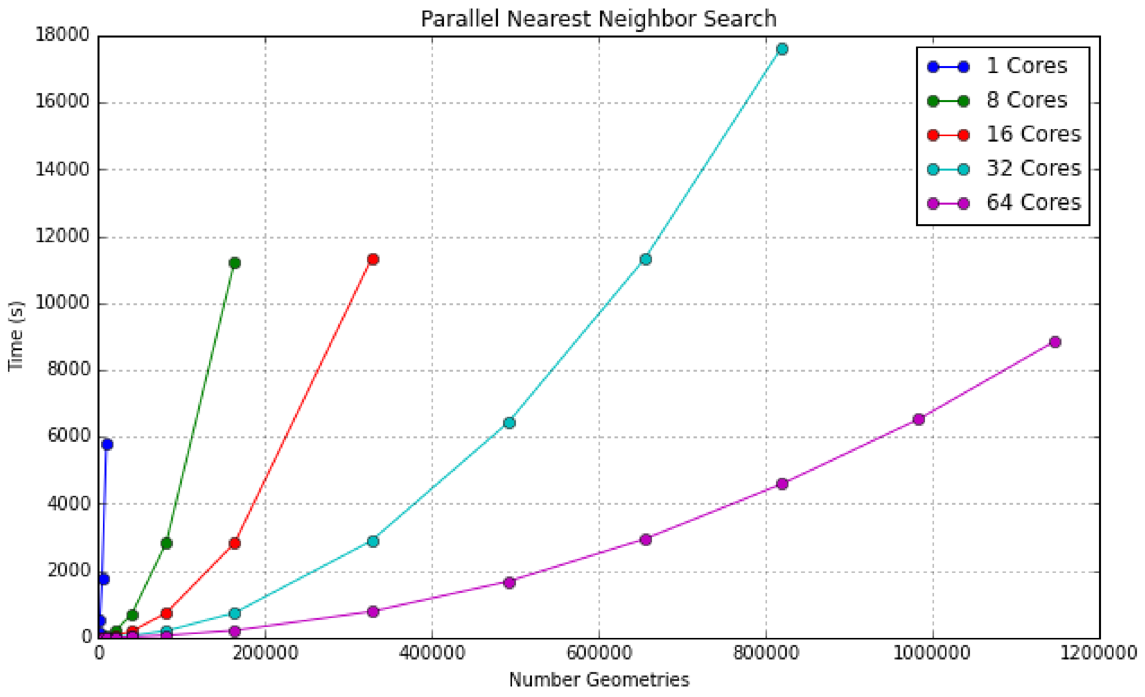


Figure 4.5: Total Compute times Using Parallel Sort Based Methods Showing Quadratic Time Growth.

The previous figure fails to describe where the algorithm spends the bulk of the computation time. Figure 4.6Aggregate Compute Time for NNS Using Parallel Sortingfigure.4.6 illustrates the decomposed compute times including, the time cost for data generation, global scatter and sorting, identification and communication of I_{ij} and C_{ij}, aggregation of C_i for edge case computation, and final reduction. The figure is composed of a left and right component, with the former showing geometry count up to 40,960 and the latter, with a rescaled y-axis two orders of magnitude larger, showing geometry counts up to 1,146,880. The bulk of computation is performed during the x-decomposed and y-decomposed NNS phases. This is as anticipated as the local compute time for the decomposed subset is $O(n^2)$ using a naive linear implementation. The compute times for I_{ij} and C_{ij} are non-trivial. Data communication and global sort phases are extremely efficient, suggesting that improvement to the local NNS using an optimal algorithm, e.g. an $O(n \ log \ n)$ implementation would yield significantly improved aggregate performance.

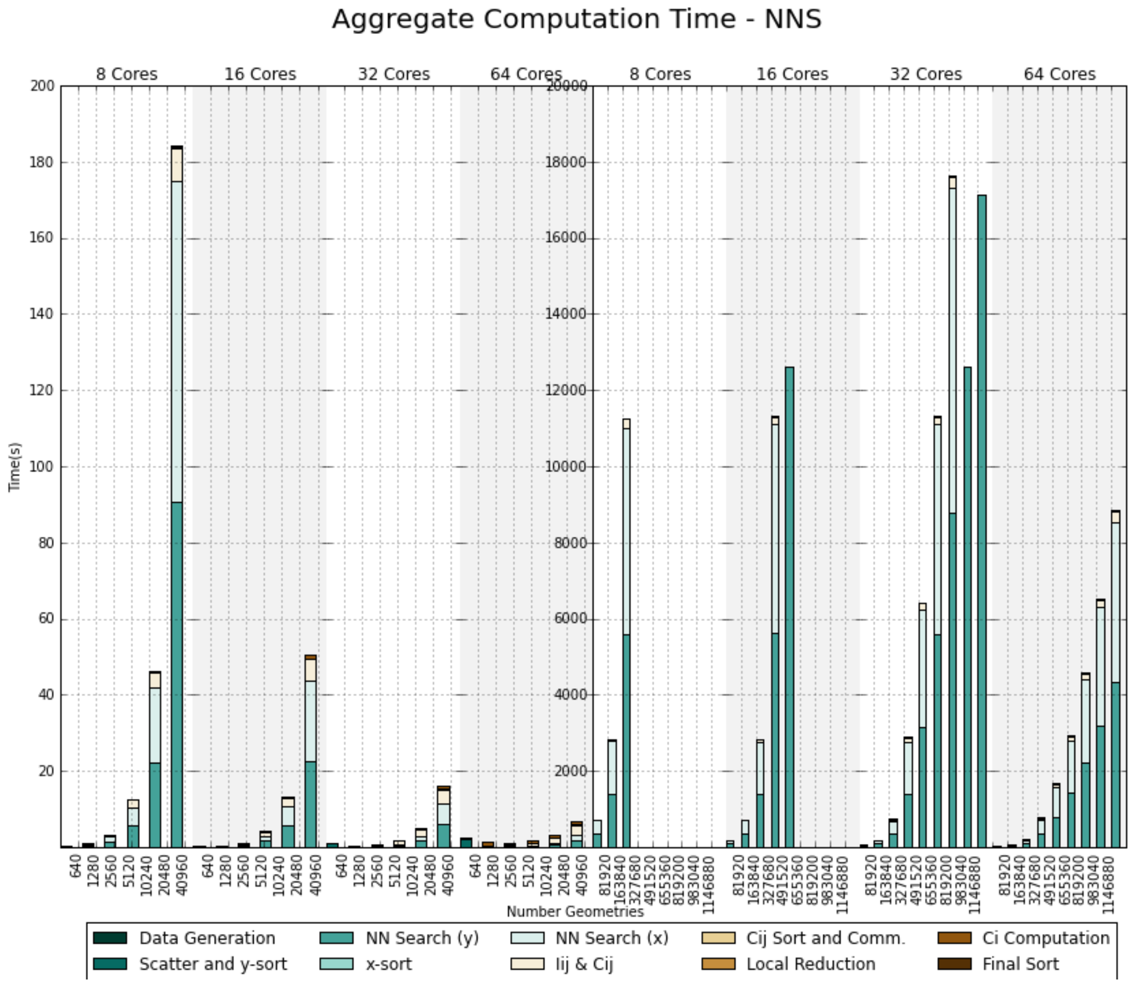


Figure 4.6: Aggregate Compute Time for NNS Using Parallel Sorting.

Figure 4.7Aggregate Compute Time for NNS Using a Tree Based Approachfigure.4.7 shows the total compute time and aggregate costs for a tree based (KD-Tree) approach. I originally hypothesized that the serial computation and communication of a complex topological data structure would be extremely inefficient. In the case of NNS, this is hypothesis is false. While serial KD-Tree generation compute costs account between a third and a half of total compute times across all core counts, local query, e.g. walking the topological structure and performing a lookup, is extremely efficient. Figure 4.5Total Compute times Using Parallel Sort Based Methods Showing Quadratic Time Growthfigure.4.5 illustrated runtimes between over 180 seconds for an 8 core, 40,960 points NNS search and 90 seconds for a 64 core, 1.14 million points NNS search. In contrast, the more complex domain decomposition KD-Tree approach required just 2.8 seconds in the former and 50.7 seconds in the latter case. The cost of communication is non-trivial, but the efficiency in query offsets this cost.

4.6.2 Polygon Adjacency

In the SMP environment, across all synthetic data tests the R-Tree implementation was 7 to 84 times slower than the binning implementation and 22 to 1400 times slower than the list based contiguity measure. Additionally, the R-Tree implementation required significant quantities of RAM to store the tree structure. [11] Due to these factors, only the binning and list based approaches appear in subsequent figures.

Figure 4.8The Continuum of Communication Costs Incurred by Different Data Sharing Modelsfigure.4.8(a - d) illustrate the results of four experiments designed to compare the performance of the list based and binning approaches as a function of total geometry count, total vertex count (and by extension edge count), average neighbor cardinality, and data distribution. Figure 4.8The Continuum of Communication

[11] In fact, some of the performance degradation may be a function of memory swapping.

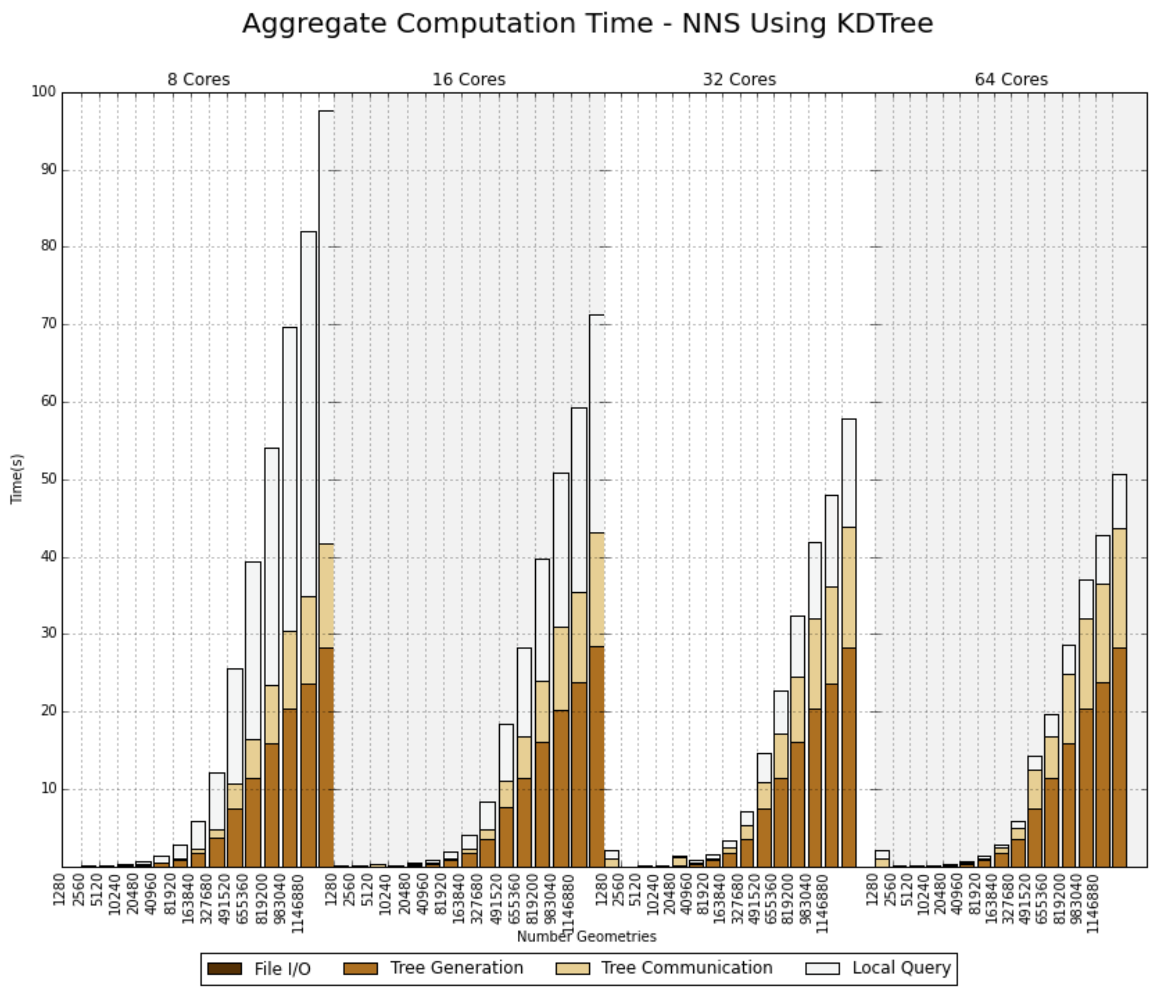


Figure 4.7: Aggregate Compute Time for NNS Using a Tree Based Approach.

Costs Incurred by Different Data Sharing Modelsfigure.4.8(a) illustrates the scaling performance of the list and binning algorithms. The former scales linearly as the total number of polygons is increased and the latter scales quadratically. As anticipated, the Rook contiguity measures require slightly more processing time than the associated Queen contiguity measures. In Figure 4.8The Continuum of Communication Costs Incurred by Different Data Sharing Modelsfigure.4.8(b), the algorithm exhibits increased computational cost as a function of geometric complexity, e.g. the number of vertices, number of edges, and mean number of neighbors. This is illustrated by the general trend of compute times with the triangular tessellation requiring the least time and the hexagon tessellation requiring the most. Densification of the 4096 hexagon polygon with between 6 and 300 additional vertices per edge highlights an inversion point, where binning regains dominance over the list based approach, Figure 4.8The Continuum of Communication Costs Incurred by Different Data Sharing Modelsfigure.4.8(c). Finally, in Figure 4.8The Continuum of Communication Costs Incurred by Different Data Sharing Modelsfigure.4.8(d)the total compute time using randomly distributed polygon datasets are shown. Again, this figure demonstrates quadratic scaling for the existing binning approach and linear scaling for the list based approach.

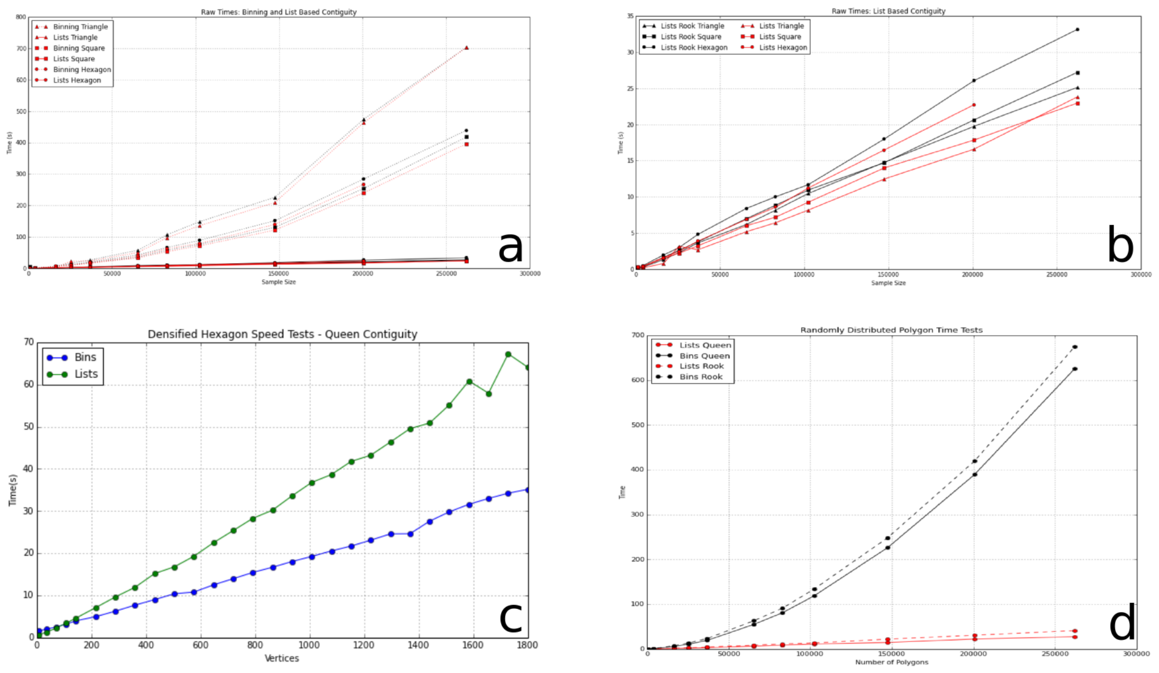


Figure 4.8: The Continuum of Communication Costs Incurred by Different Data Sharing Models.

In the HPC environment, I focus on parallel list based and parallel R-Tree implementations. In Figure 4.9Speedup Using Parallel Sorting for Polygon Adjacency. Note That the 64 Core Tests Omit the 1024 Polygon Tests in Most Casesfigure.4.9 I report significant speedup as computed by Amdahl's Law across almost all tested geometries and core counts. Varying line lengths are a product of the 5 hour limit placed on serial processing. Globally, I report increased speedup as a function of total geometry count. Across all core counts randomly distributed polygon data displays the best performance with over a 20 time speedup for 200,704 randomly distributed polygons. At geometry counts less than 65,536 , I report low speedup, with 32 and 64 cores; parallel computations of the 1024 geometry problem sets perform worse than the serial implementation. As with the SMP tests, I see performance of this method being tied to the geometric complexity of the underlying geometry with square and hexagon lattices consistently performing worse than triangle lattices. Additionally, for randomly distirbuted data, I see better load balancing, a function of better pivot selection in the global sort phase.

As above, the speedup curves fail to report either total runtime or the decomposition of compute time such that a new processing bottleneck could be identified. In Figure 4.10Aggregate Compute Costs for Polygon Adjacency over All Tested Geometries Using a Sort Based Methodfigure.4.10 shows aggregate compute times for all tested geometries at all tested core counts. Globally, maximum compute times for the highest geometry counts remain under 40 seconds, irrespective of core count. The vast majority of compute time is spent in serial File I/O (dark brown) with the parallel 'Scatter and Sort' and 'Local Coincident' point computations requiring significantly less total compute time. Data aggregation is also non-trivial and this can be identify as another serial processing bottleneck. 'Scatter and Sort' operations require the bulk of compute time for 1024 and 2096 triangle lattice cases when using 32 or 64 cores. This is attributable to poor pivot selection such that the load balancing results in one or more idle processing cores.

Finally, Figure 4.11Aggregate Compute Costs for Polygon Adjacency over All Tested Geometries Using a R-tree Based Methodfigure.4.11 shows aggregate compute

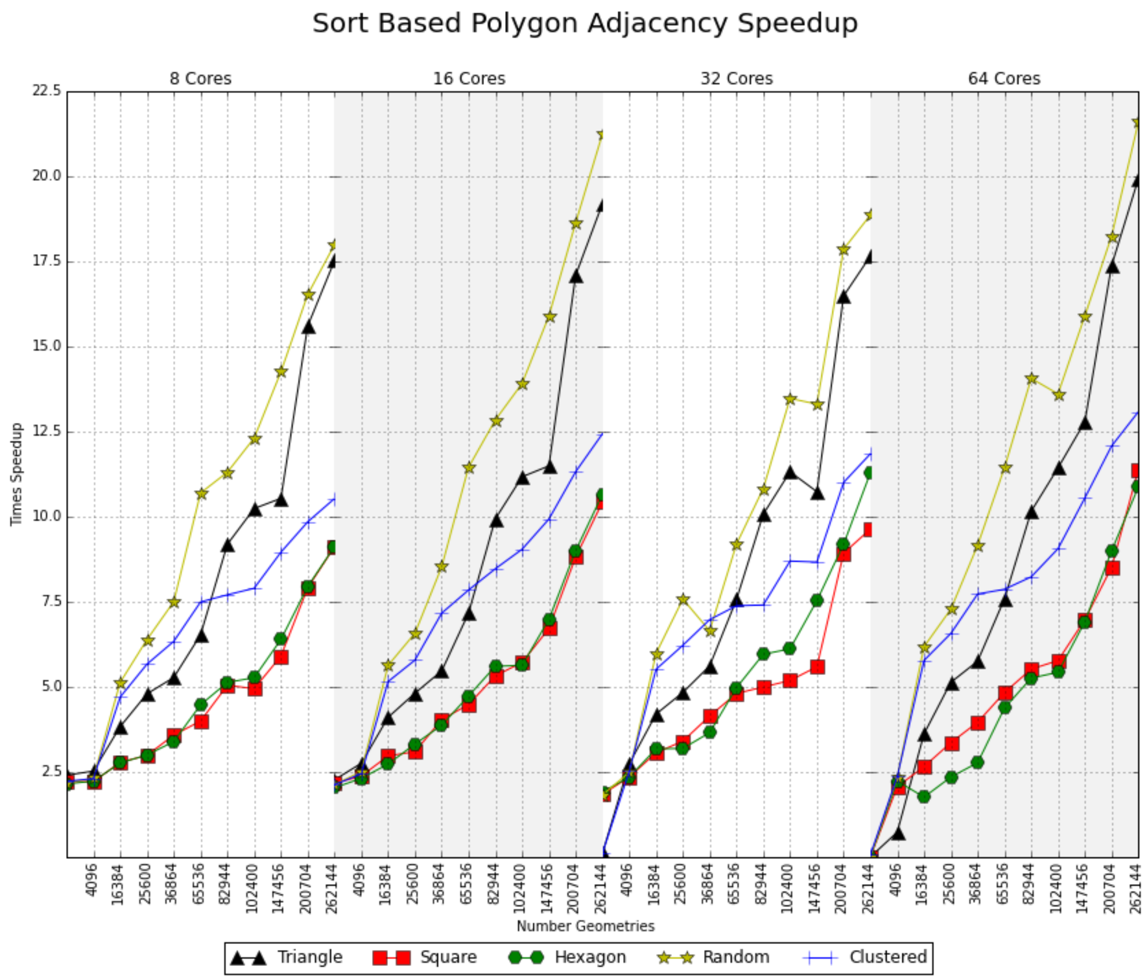


Figure 4.9: Speedup Using Parallel Sorting for Polygon Adjacency. Note That the 64 Core Tests Omit the 1024 Polygon Tests in Most Cases.

times for an identical set of tests using a parallel R-Tree implementation. Compute times are under 600 seconds for irregular lattice datasets and under 7200 seconds for regular triangle, square, and hexagon lattices. Collectively comparing the regular lattice data results to the irregular lattice data results, the former spent the vast majority of compute time aggregating the distributed W objects to the managing process. The latter spent significantly more time performing the distributed R-Tree query. Having controlled for geometry count, this can be attributed to the higher aggregation costs due to high quantities of duplicate information. That is, the regular

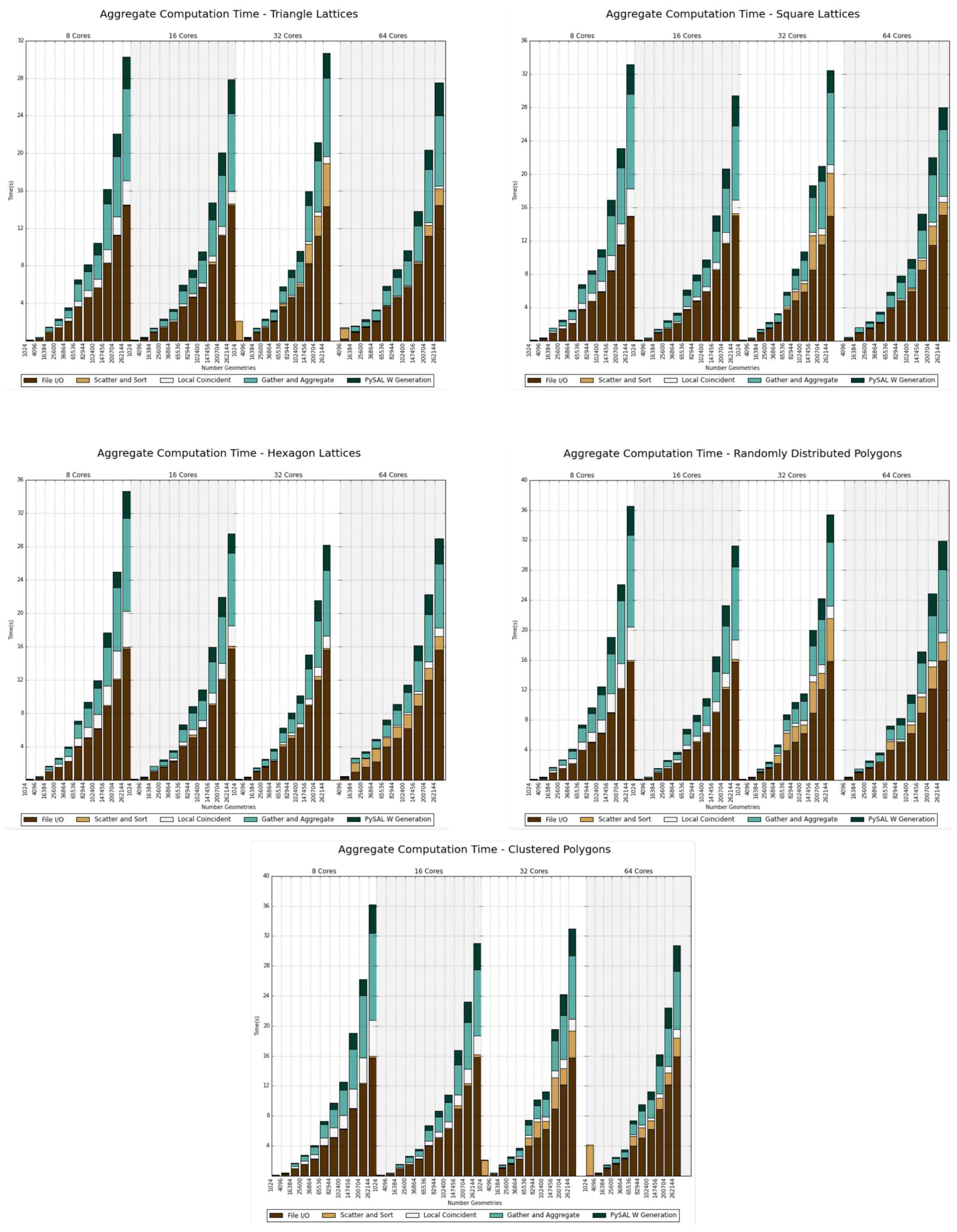


Figure 4.10: Aggregate Compute Costs for Polygon Adjacency over All Tested Geometries Using a Sort Based Method.

lattice data decomposition into a balanced R-Tree causes efficient yet redundant computations to be performed. Conversely, irregular lattice data is better decomposed to avoid duplicative computation at the cost of increased query complexity. Irregular lattice data computations incur non-trivial communication costs as the R-Tree becomes increasingly more complex. The cost to communicate the more complex R-Tree data structure is on the order of the total compute times required for the list based approach.

4.7 Discussion

The above results highlight the dual nature of topological data representations. In the case of NNS, significant performance improvement is realized utilizing a KD-Tree over a more simplistic decomposition. The best case $O(logn)$ insert and query[12] performance of the KD-Tree offsets the potential higher costs of decomposition, tree generation, and communication. Even if an optimal $O(n\ log\ n)$ local NNS algorithm is utilized, the more complex decomposition will remain more efficient. Therefore, I suggest that topological data structures with good creation performance and small data footprints can be communicated over high-speed networks to drive efficient, parallel, topological algorithm implementations.

Moving to the polygon adjacency algorithms, both the SMP and HPC environments suffer from significant performance degradation when using the more complex tree based representation. These can be attributed to inefficiencies in the generation of the tree data structure when more complex geometries, e.g. not regular tessellations, are used and attribute this to the decomposition process which requires the computation of the Minimum Bounding Rectangle, the addition of the MBR to the tree, and potential rebalancing of leaf nodes. Additional inefficiencies are also seen

[12] Worst case $O(n)$.

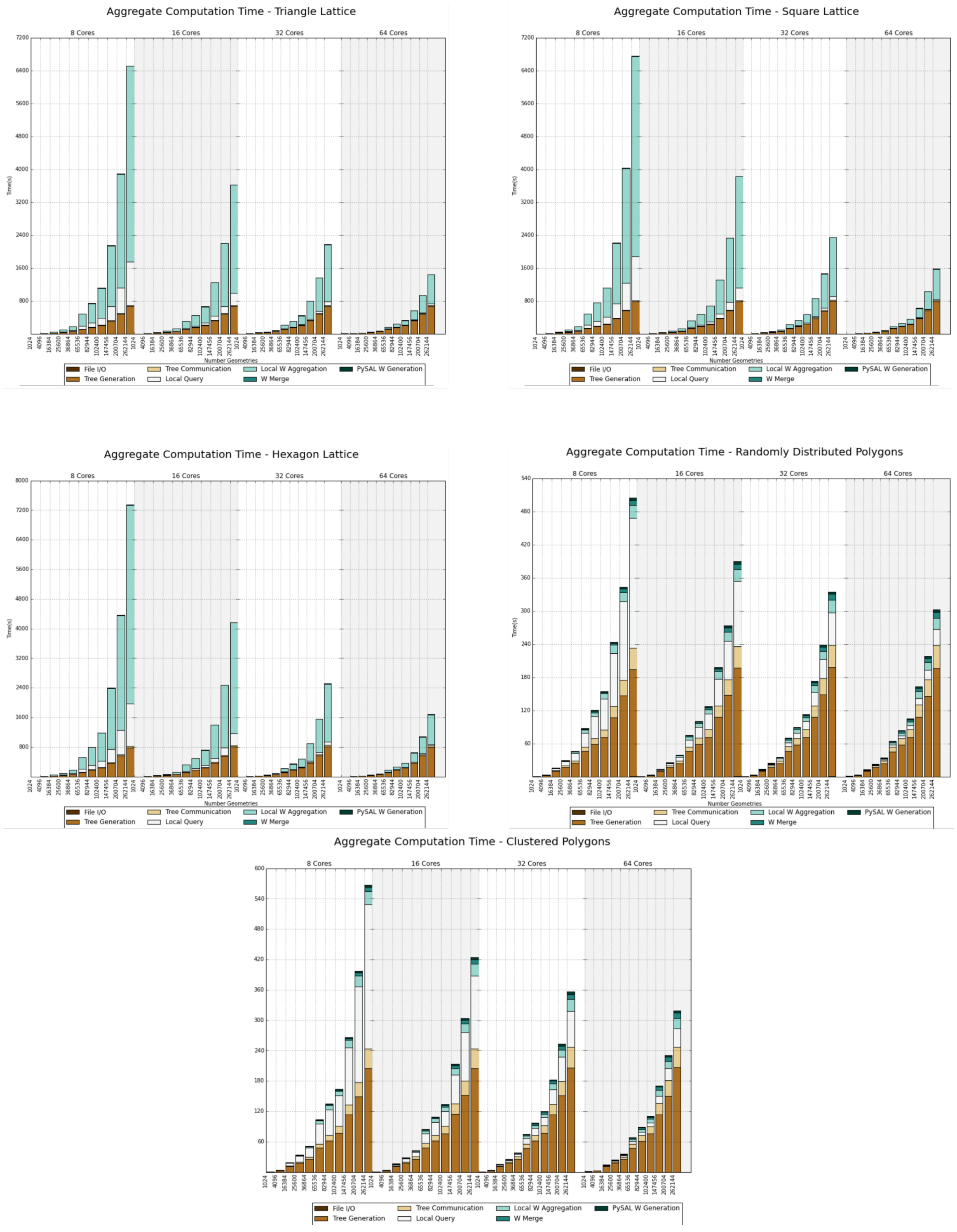


Figure 4.11: Aggregate Compute Costs for Polygon Adjacency over All Tested Geometries Using a R-tree Based Method.

in the communication of the more complex topological data structure. By comparing the low R-Tree query costs, seen in the irregular lattice data, with the high query costs seen in the regular lattice data, I suggest this is a function of the artificial ordering of the data. Therefore, the efficiency of simple spatial binning and geometric implementations are globally more efficient. This statement is not without the caveat that, should a more memory efficient and creation performant R-Tree algorithm be leveraged this relationship could invert and a significant number of geospatial algorithms operating on geometric entities could be classified as *topological* dwarfs.

The two major serial processing costs within the polygon adjacency algorithms are file I/O and data aggregation. These costs are invariant to the methods tested and account for a significant quantity of the total processing time. A shift, away from the ubiquitous shapefile, to data formats that support parallel I/O such as the Hadoop HDFS filesystem or parallel HDF5 would allow for significant reduction in total compute times. Likewise, analytical software expects a single memory address containing a single spatial adjacency object. This requires that the distributed W objects be aggregated once computation is completed. Clearly, this does not scale well as the total number of observations is increased. Additional work is required to explore the potential for distributed adjacency objects and associated metadata to remove the aggregation phase.

4.8 Conclusion

In this chapter I explore the implementation of NNS and polygon adjacency as two representative algorithms requiring geometric comparisons. The algorithms were implemented as both *topological* and *geometric* dwarfs to assess the costs of decomposition, communication, local computation, and potential data aggregation. Clear linkage exists between the type of decomposition utilized and the classification of the implementation. Therefore, I describe naive algorithm implementations as well as regular and adaptive spatial binning approaches. This work offers a regular, parallel sort based method to perform polygon adjacency in the HPC environment and a hybrid high performance container approach in the SMP environment. To my knowledge, parallel sorting methods have not been applied to the polygon adjacency method previously in an HPC environment.

More complex decomposition strategies are efficient assuming that the resultant data structures are memory efficient and rapidly queryable; this is the case of NNS. Increases in the geometric complexity of the input data, along with additional decomposition, communication, and query costs are shown to make *topological* implementations of polygon adjacency significantly more expense than a *geometric* implementation.

Future work will focus on five major areas. First, the implementation of an asymptotically optimal local NNS algorithm will be deployed to test whether significant local search improvement alters the current assertions regarding the classification of NNS as a topological dwarf. Next, more efficient tree creation algorithms and storage structures will be explored in an effort to test whether casting polygon adjacency as a topological problem is feasible. Third, the potential to realize a distributed W object in an HPC environment will be tested with the goal of abstracting the representation such that seamless integration with existing functionality is possible. Fourth, the implemented algorithms provide high-speed algorithms to support research into the impact of approximate nearest neighbor estimators at large data sizes with low dimensionality, and the addition of a 'fuzzy' operator to account for spatial error in the case of polygon adjacency. Finally, parallel I/O technologies will be explored in an effort to significantly reduce total polygon adjacency compute times using the proposed algorithms.

Chapter 5

SPATIAL REGIONALIZATION AS A MAPREDUCE CLASS OF PARALLEL ALGORITHM

5.1 Introduction

Regionalization, a fundamental GIScience research area, is an NP-Hard, complex combinatorial problem that seeks to aggregate n polygon spatial units into p regions or zones ($p \leq n$). Due to the computational complexity, commercial solvers, such as C-PLEX (ILOG 2013), formulations the model using a Mixed Integer Programming (MIP) model are constrained to optimally solving small to medium problem sets. For example, Duque et al. (2011) propose the p-regions problem, which aims to generate p contiguous regions from n polygonal units with the objective of minimizing the intra-zonal heterogeneity of some attribute. Duque reports that a problem set where $n = 50$ required up to three hours[1] before a best known optimal solution was computed. Therefore, researchers have developed heuristic regionalization algorithms to solve medium to large problems in a reasonable amount of time.

A regionalization heuristic often includes two phases. The first phase, region growth, seeks to generate an Initial Feasible Solution (IFS), and the second phase, local search aims to permute a given IFS toward better solutions. Although a well-developed and parameterized heuristic algorithm is capable of locating a good solution quickly, it is still possible to become trapped in a local optima. [2] Therefore, a key

[1] Utilizing Dell Precision T3400 running 64-bit Windows XP operating system with a 2.99 GHz Intel Core 2 Extreme processor and 8 GB of RAM.

[2] A minima or maxima within a local neighborhood from which the algorithm may or may not be able to escape, that is, a sub-optimal solution that is erroneously computed to be optimal.

design principle is to allow the algorithm to search as much of the solution space as possible without becoming trapped in a local optimum. Finding an optimal solution remains a compute intensive task even when applying heuristic algorithms. This is a huge challenge for the pervasive, serial processing environment.

Applications of regionalization in spatial analysis include the aggregation of spatial units to analytical zones such as Traffic Analysis Zones (TAZ) for transportation research (Miller and Shaw 2001), congressional districting for political science research (Gonzlez-Ramrez et al. 2011), Primary Care Service Area (PCSA) identification for public health research (Pathman et al. 2006) and census data for demography studies (Openshaw 1977). Additionally, a substantial portion of GIScience research involves the analysis of geographic phenomena and patterns at different scales, such as comprehensive modeling of urban economic, land use and transportation (Li et al. 2014a), hydrological cycle simulation and prediction at a wide range of spatiotemporal scales (Gupta and Waymire 1998), electoral district partitioning to facilitate governmental administration (Duque et al. 2007), and the generation of optimal coverage regions for public/commercial service delivery (Armstrong et al. 1991). Advancement in zonation research is largely attributable to the development of methods and techniques to solve regionalization problems, which allow the dynamic generation of aggregated spatial data that achieves some predefined objective and often also satisfies constraints for different applications (Li et al. 2014).

This chapter reports efforts in exploiting shared memory processing (SMP) and High Performance Computing (HPC) platform to improve the effectiveness (solution quality) of a heuristic-based regionalization problem without significant increases in compute time. A non-linear set of the regionalization problem – the p-Compact Regions problem (Li et al. 2014), which aims at generating p compact and contiguous regions, is used as the case study in implementing a low overhead parallelization strategy. The SMP platform was chosen to implement the parallelization strategy because: (1) multi-core SMP desktops are readily available to all researchers in recent years, and most spatial analysis tasks are conducted in this environment, as evidenced by the over 300,000 ESRI ArcGIS desktop users (Howell 2009); (2) development of an SMP implementation requires algorithm decomposition for multiple processing cores (workers) communicating using message passing, a paradigm portable to the HPC deployments.

The remainder of this chapter is organized as follows: Section 2 describes the maximum theoretical speed improvement attained through parallelization, strategies of parallelization, and the platforms parallel code can be deployed to. Section 3 describes and formulate the p-Compact Regions optimization problem. Section 4 describes the developed parallel implementation in detail, Section 5 describes the experiments performed and Section 6 reports results. Finally, Section 7 concludes with a discussion of this work and avenues for future research.

5.2 p-Compact Regions Problem

PCR enforces a contiguity constraint, a hallmark of a spatial regionalization problem. In contrast, PCR requires that p be defined a priori, does not define a floor constraint to maintain a minimum unit size, and does not account for a similarity or dissimilarity metric. Regions are formed and permuted with the goal of maximizing the likeness of a region to a circle as measured by the normalized moment of inertia (Li *et al.*, 2014b; Laura *et al.*, 2015). The objective function formulation, as presented by Li *et al.* (2014b); Laura *et al.* (2015) of PCR is:

$$\textbf{Maximize:} \tag{5.1}$$

$$\sum_{k=1}^{p} C(Z_k) \tag{5.2}$$

$$\textbf{Subject to:} \tag{5.3}$$

$$\sum_{i=1}^{n} x_i^{k0} = 1 \forall k = 1, \ldots, P \tag{5.4}$$

$$\sum_{k=1}^{p} \sum_{c=0}^{q} x_i^{k,c} = 1 \forall i = 1, \ldots, n \tag{5.5}$$

$$x_i^{ko} \leq \sum_{j \in N_i} x_j^{k(o-1)} \tag{5.6}$$

$$t_{ij} \geq \sum_{o}^{q} x_i^{ko} + \sum_{o}^{q} x_j^{ko} - 1 \tag{5.7}$$

where k is a region index, $C(Z_k)$ is the compactness index for region k measured by Equation 5.8p-Compact Regions Problemequation.5.2.8, and p is the number of regions. $C(K_z)$ is the compactness index as measured by the normalized moment of inertia, which can be represented as the ratio between the second moment of inertia I_0 of a circle, with area Z_k, and the second moment of inertia of Z_k around an axis perpendicular to it and passing through the centroid G. Li *et al.* (2014b); Laura *et al.* (2015) formulate this as:

$$C(Z_k) = NMI(Z_k) = \frac{I_0}{I_{Z_k}^G} = \frac{A_{Z_k}^2}{2\pi I_{Z_k}^G} \tag{5.8}$$

where $A_{Z_k}^2$ is the area of region Z_k.

The contiguity constraint is enforced by Equations 5.4p-Compact Regions Problemequation.5.2.4 through 5.7p-Compact Regions Problemequation.5.2.7 where $x_i^{k,c}$ is a binary indicator of whether atomic unit i is assigned to region k in order c. 'Order' here refers to the closeness to the seed of a region, which has an order 0. x_i^{ko} is a binary indicator of whether unit i is assigned to region k in order o. N_i is the set of atomic units that are adjacent to atomic unit i, c is an index of contiguity order with $q = (n-p)$, and t_{ij} is a binary indicator of whether unit i and unit j are members of the same region k. Equation 5.4p-Compact Regions Problemequation.5.2.4 requires that each region contains a single root or seed unit. Equation 5.5p-Compact Regions Problemequation.5.2.5 constraints unit assignment to one region and one contiguity order. Equation 5.6p-Compact Regions Problemequation.5.2.6 requires that a unit be assigned to a region only if at least one of its adjacent units had been assigned to the region at a lower order. Equation 5.7p-Compact Regions Problemequation.5.2.7 prevents cycling by ensuring that each unit can occur only once in a given order.

The NMI value has range of $(0, 1]$, in which 1 refers to the most compact shape, a circle, and when a shape is the least compact, it will have a value close to 0. The contiguity constraint can be defined using ORDER model proposed by Cova and Church (2000) and described by Duque *et al.* (2011); Li *et al.* (2014b). In a heuristic algorithm, the contiguity is preserved by an identifiable path from any unit in a region to any other unit in the same region.

Broadly, the PCR algorithm follows the same multistep process as MPR, utilizing a MERGE heuristic to search the solution space and return a solution. MERGE is a two-phase algorithm consisting of (1) the generation of an initial feasible solution (IFS) and (2) a local search phase where atomic units are swapped in an attempt to improve the solution. Throughout this section I define each polygon unit within the dataset as an atomic unit, a group of conterminous atomic units as a region, and the atomic unit initially selected to start a region as the seed unit.

5.2.1 Phase I: Generation of an Initial Feasible Solution

1. PCR requires that the number of regions to be generated, p, be known a priori and that a seed unit be selected for each region. Seed units, selected from the global set of all atomic units, n, can be defined in three ways: (1) manually identify p atomic units to be classified as seed units, (2) randomly select p atomic units to be defined as seed units, or (3) systematically, i.e., at regular spacing or in the atomic units with greatest area, select p atomic units to be defined as seeds.

2. The study area now consists of p seed units assigned to p regions and $n - p$ unassigned atomic units. Starting from a randomly select region and using a round-robin (dealing) approach, each region is grown by l atomic units. This process is performed by first selecting all unassigned atomic units that are neighbors to a region. Neighborhood is determined by shared borders in the rook contiguity model. Next, the candidate atomic unit that provides the maximum improvement to the objective function value is selected and assigned to the region. This process is also named region growth process and at each step, only one region grows by adding one atomic unit to it. Once one region finishes growing, the process iterates to the next region. Then this process continues until $l * p$ atomic units have been assigned to p root regions.

3. At the conclusion of step 2, p regions have been defined, each of size $l+1$ atomic units. Figure 5.1Phase I, Step 3, Assignment of All Atomic Unitsfigure.5.1 illustrates three regions at the start of this step. Therefore, $n-p(l+1)$ atomic units remain unassigned. To assign the remaining atomic units a randomized greedy algorithm is utilized. First, for each region, all possible atomic units to region assignments are enumerated, atomic elements 1, 2, and 3 are identified adjacent

to region A, 5.1Phase I, Step 3, Assignment of All Atomic Unitsfigure.5.1. This yields, j potential region growth plans for each region, where j is a function of the number of unassigned atomic units adjacent to a given region. Next, all region growth plans are aggregated and sorted in descending order, i.e., those atomic unit assignments that improve the objective function (Eq. 2) are ranked highest. Finally, a randomized greedy algorithm selects an atomic unit assignment from the first x, sorted, potential assignments. This results in a single atomic unit assignment to a single region. This process, exploration of all potential assignments, ranking as a function of the objective value obtained by potential assignment, and random assignment selection is then repeated for all remaining, unassigned, atomic units. Assuming atomic units selected as seed units remained constant, i.e., the spatial location of the seed unit is unchanged, this step ensures that sequentially generated IFS can have divergent solutions.

In pseudo-code, this process can be realized as:

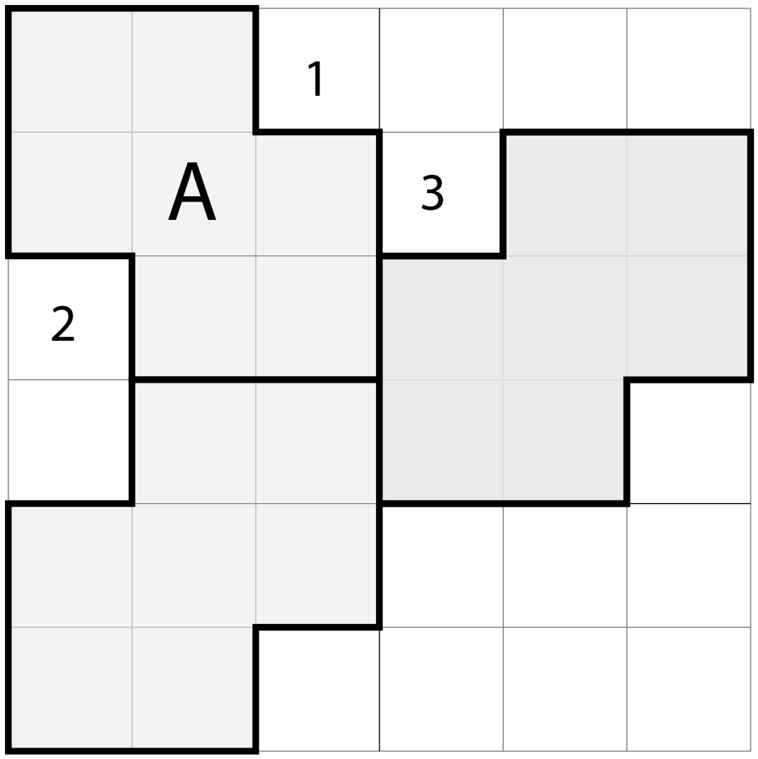


Figure 5.1: Phase I, Step 3, Assignment of All Atomic Units

```
Data: Spatial Weights Object, MI, n, p , dealSize, Optionally: Seeds
Result: Vector of length n + 1
regionmembership = dictionary
regionproperties = list
ndealt = 0
r = 0
for s in Seeds do
    regionmembership[r] = s
    r += 1
end
while $ndealt \leq dealSize * p$ do
    for Each region in regionmembership do
        for Each neighbor, unassigned areal unit adjacent to region do
            Compute the objective function for each possible assignment
            Make the best assignment
        end
    end
end
while length(ndealt ¡ n do
```

where, MI is a list containing the precomputed Moment of Inertia (MI) for each input areal unit, n is the number of areal units, p is the number of regions to be formed, dealSize is the total number of areal units to be dealt to each region in a round robin manner, and $Seeds$ is an optional list of the seed or root areal units from which regions are grown.

5.2.2 Phase II: Local Search

Local search is controlled by SA and seeks to alter atomic unit membership within adjacent regions such that the global objective function value increases. The process of reassigning atomic units is completed using an edge reassignment technique, which selects a single atomic unit bordering an adjacent region and reassigns the atomic unit to the neighboring region towards finding better solutions. For example, if atomic unit A is a member of region 1 and adjacent atomic unit B is a member of region 2, an edge reassignment by moving A from region 1 to 2 will be allowed if this move increases the overall objective function value. This process will continue until no improvement can be found by moving any of the units on the edge of any given region.

I note two important considerations. First, the contiguity needs to be preserved at all phases of the regionalization process, e.g., an edge reassignment that breaks regional contiguity is prohibited. Second, the compactness of regions needs to be computed at both the region growth and local search phases. The additive nature of this measure ensures that re-computation of all regions is not required with each permutation of the local search, e.g., an edge reassignment alters compactness for two regions, whose region compactness is subtracted from the global compactness measure, recomputed, and then added back to the global compactness measure. Therefore, this measure is efficient to compute during local changes in region membership.

One possible realization of a serial implementation is as follows:

```
Data: Spatial Weights Object, $region_{membership}$, $region_{properties}$, $initial_{temperature}$,
      $cooling_{rate}$, $final_{temperature}$
Result: region-membership, final-ojective-value
while $current_{temperature} > final_{temperature}$ do
    Randomly select an areal unit
    for each adjacent areal unit, i do
        Get region membership for areal unit i
    end
    unique regions = list of all unique adjacent regions to the randomly selected
    areal unit
    for each region in unique regions do
        Check contiguity constraint
        Compute the objective function assuming reassignment of the selected areal
        unit into the region
    end
    if Objective improves then
        Make the areal unit reassignment
    end
    Cool the $current_{temperature}$ by the $cooling_{rate}$
end
```

Algorithm 2: Local Search

where, the spatial weights object describes the adjacency structure of all areal units, $region_{membership}$ is a dictionary containing assignment information for all areal units, $region_{properties}$ are descriptors of each region, e.g. NMI, and $initial_{temperature}$, $cooling_{rate}$, and $final_{temperature}$ are parameters of the SA process.

5.3 Parallel p-Compact Regions

Like the MPR implementation, a parallel PCR implementation focuses on maximizing the probability of finding a high quality solution by maximizing the global compactness of all regions. Given the difficulty in creating a cooperative parallelization of the local search phase and the desire to highlight differences between implementation requirements in a parallel environment, overall performance speed is of concern for this PCR implementation. The goal of this implementation is to minimize the overhead introduced by parallelization.

This parallel PCR implementation utilizes a low level (Trienekens and Bruin, 1992), bulk synchronization strategy in both SMP and cluster environments. The former utilizes a shared memory space, while the latter embodies distributed memory systems and uses a shared nothing, message passing approach. For both architectures, this implementations consists to two parallel processing phases that align with the original serial implementation. As described below, this implementation style has been selected to support load balancing across all available nodes. That is, the discrete computation of IFS, followed by bulk synchronization, and then local search, is a function of the need to load balance this algorithm as a single phase parallel implementation yields the potential of a higher number of idle cores.

First, file I/O occurs and a spatial weights object, used to describe contiguity, as well as all necessary attribute vectors are generated. Next, i IFS are generated using an embarrassingly parallel implementation. Finally, local search is applied using Simulated Annealing. Unlike MPR, the majority of the compute cost is spent in the generation of IFS. Specifically, the assignment of unassigned areal units, requires the enumeration of all possible growth plans, and then a Greedy based assignment. The cost of this process is nontrivial. The cost of the two preceding steps is quite small, and an excess computation paradigm is leveraged. While one core could compute steps one and two of the IFS generation process, described above, and communicate the results of this process to all cores such for subsequent Greedy assignment it is equally efficient to perform steps one and two local to all cores, having non-idle processes perform excess computation.

5.3.1 Initial Feasible Solution Generation

The generation of high numbers of IFSs helps to improve the likelihood that a heuristic algorithm attains optimality as the better the IFS, the more likely that the final solution will be a high-quality global optima (Ram *et al.*, 1996). Therefore, this implementation seeks to generate a suitably higher number of IFSs distributed over all available compute cores (processors). This is a two-phase process, as illustrated in Figures 5.2Parallel MERGE Implementationfigure.5.2 and 5.3Parallel Local Search Implementation. Mother Process Tasks Shown with Dashed Border and Child Process Tasks Shown with Solid Border. Note That the Mother Process Becomes a Child When Finished with Initializationfigure.5.3.

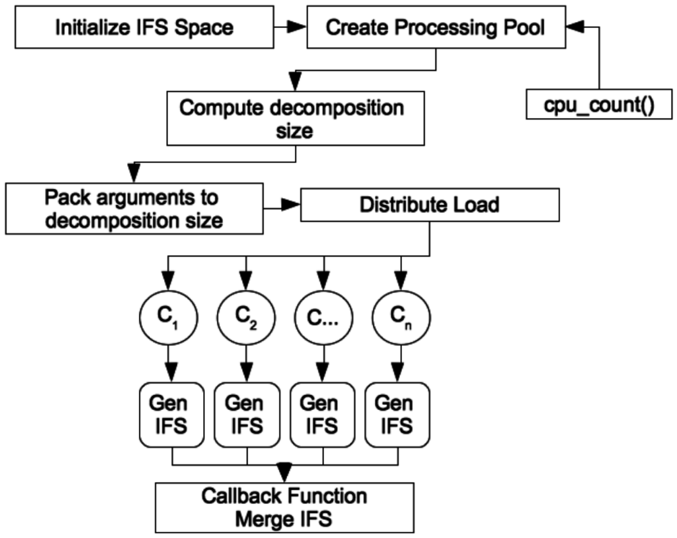


Figure 5.2: Parallel MERGE Implementation

First, the total computation time to generate a single IFS is known, given the same set of parameters, requires roughly the same compute time. Therefore, the distribution of load to each processor is simply a function of the number of IFS that each processor must generate. In the SMP environment, the near optimal distribution of jobs to each core can be computed as:

$$C_l = \frac{s}{c} \tag{5.9}$$

where C_l is the total load for each core, s is the number of solutions to be generated to populate the global solution space, and c is the total number of available cores. This split does not account for the likely remainder, and therefore, any remained is assigned to a single core.

Once load distribution is computed, c child processes are launched and the generation of multiple IFS initiated. This application is asynchronous and as a core completes the assigned load a callback function is called to merge the individual IFS in a single memory space via a pipe. The generation of IFS by each child process is identical to the description by Li et al. (2014b). It is within the initialization phase that the algorithm deals x atomic units to each seed unit to set the initial region membership for atomic units.

In the HPC environment, a single shared memory space is not available. Therefore, a processing queue of IFS to be generated is used, with each available core drawing from the queue when processing is completed. As a single core completes the generation of an IFS, a message, containing the IFS solution, is passed to the managing process which either (1) assigns the worker to generate another IFS or (2) signals that IFS generation is completed.

These implementations were selected for two reasons. First, the derivation of a single IFS is not dependent upon the derivation of other IFSs. It is not necessary to manage the quality of an IFS, seek to maintain individuality among a pool of IFSs, or attempt to synchronize the generation of IFS over all cores. Second, in cases where an intensification strategy will be used prior to local search, it is essential to gather and rank all solutions based on some criteria. Therefore, in generating a low communication cost benchmark, I seek to mimic this processing requirement.

The initialization phase ends once all child processes have merged their IFSs in to a single IFS space. Each IFS includes the current global objective function value, the membership of all atomic units in regions, the local compactness of each region, and the requisite unit attribution data, for example contiguity and moment of inertia.

5.3.2 Local Search

In Figure 5.3Parallel Local Search Implementation. Mother Process Tasks Shown with Dashed Border and Child Process Tasks Shown with Solid Border. Note That the Mother Process Becomes a Child When Finished with Initializationfigure.5.3, illustrates the implement of a p-control, Multi-Point Same Strategies Simulated Annealing (SA) driven local search with a single bulk synchronization phase to terminate processing. Given the potential for each SA driven search to require a different amount of processing time, all IFS are loaded into a single processing queue from which jobs are drawn. This is the primary justification for splitting IFS generation and local search. The latter does not load balance well and combination with the former yields worse global performance.

Local search proceeds as per Li et al. (2014) using SA. Reassignment of an atomic unit is performed local to the worker and it is possible that one or more workers could perform identical computations that is the parameters that govern SA (e.g., cooling rate) could consistently find local optima from which the algorithm cannot escape. For this reason, both the size of the IFS pool and the SA parameter selection are paramount.

In the SMP environment, once local search has completed, the child process opens a pipe and a callback function communicates the local search results to a joinable processing queue. In the HPC environment, a completed worker sends a message containing the final solution back to the managing process. Once the global IFS queue has been processed, each child exits and the mother process regains control. In the SMP implementation, the mother process has been waiting,that is, not using any computational resources, and has acted as a child process. In the HPC deployment, the managing process acts only as the manager and does not process local solutions. Once all child processes have completed computation, the mother process queries the global solution space for the indices of all current best solutions. One or more global best solutions can exist, and each is checked for uniqueness. In the case where all solutions are identical, the global best is written. In instances where one or more solutions are unique, all global best solutions are reported.

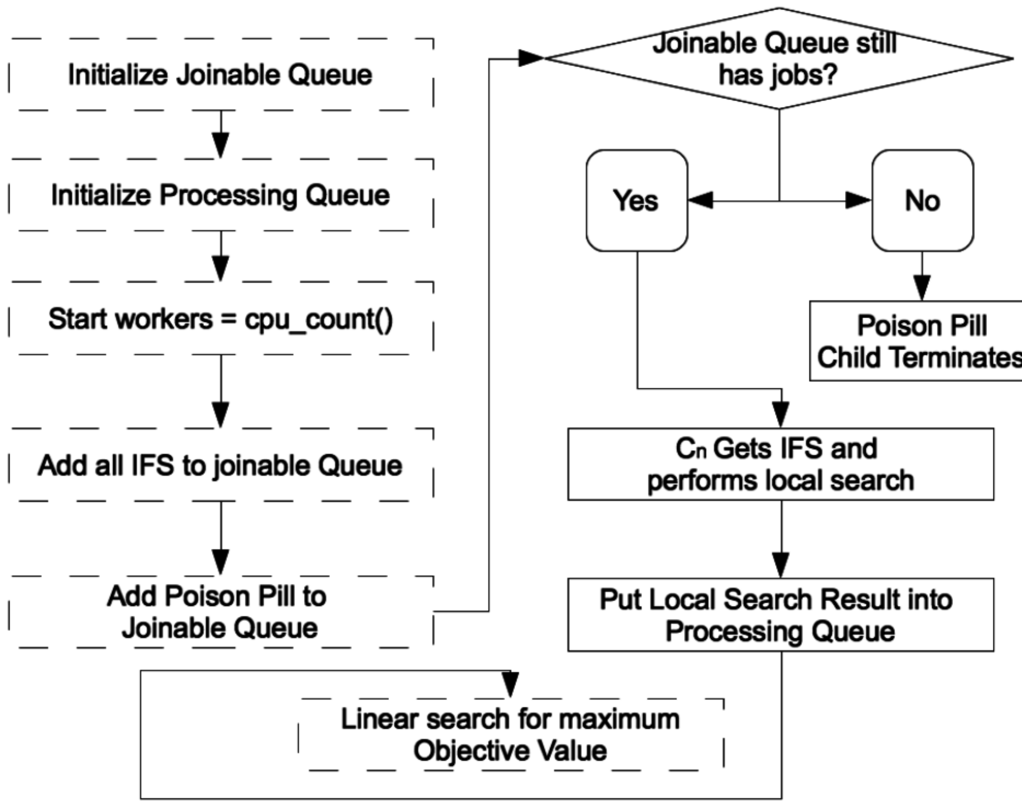

Initialize Joinable Queue
Initialize Processing Queue
Start workers = cpu_count()
Add all IFS to joinable Queue
Add Poison Pill to Joinable Queue
Joinable Queue still has jobs?
Yes
No
Poison Pill Child Terminates
Cn Gets IFS and performs local search
Put Local Search Result into Processing Queue
Linear search for maximum Objective Value

Figure 5.3: Parallel Local Search Implementation. Mother Process Tasks Shown with Dashed Border and Child Process Tasks Shown with Solid Border. Note That the Mother Process Becomes a Child When Finished with Initialization.

5.4 Experiments

Three experiments are performed in an SMP environment and one experiment is performed in a HPC environment in order to explore the speedup attained through parallelization and the impact of parallelization on solution quality utilizing synthetic data in the form of Voronoi diagrams. Data for test architectures was pregenerated with 2,500, 10,000, 22,500, 40,000, and 62,500 atomic units. Figure 5.4Sample Test Data with 10,000 Atomic Units Generated from Randomly Distributed Point Datafigure.5.4, below, depicts the 10,000 unit sample data. Each dataset was created by first generating the required number of units as randomly distributed points. Then Thiessen polygons were created from the random point dataset. Therefore, the density and number of neighbors per atomic unit varies with each dataset. Adjacency and moment of inertia attributes were computed a priori. Finally, seeds were automatically selected by first creating a regular, equidistant grid of points over the test data and then spatially intersecting each point with a a single atomic unit. These units were used as seeds for phase I of PCR.

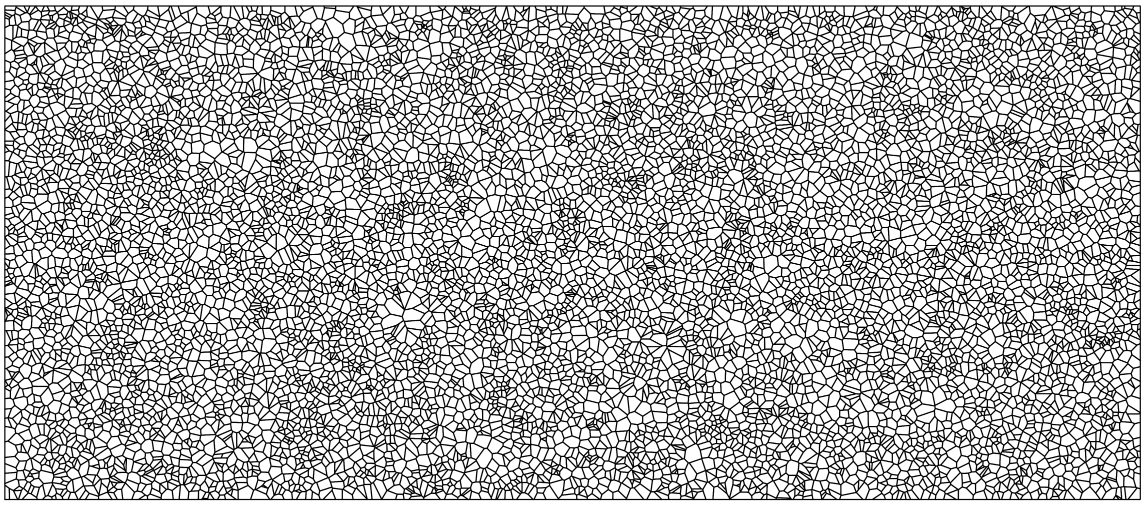


Figure 5.4: Sample Test Data with 10,000 Atomic Units Generated from Randomly Distributed Point Data.

To standardize the testing environment, all SMP tests were run on a Mac Pro with dual six-core 2.93 GHz CPUs, 64 GB of RAM and the 64-bit Python by Enthought (version 7.0-2). The dual six core processors report as 24 cores due to hyper threading. Only mandatory OS controlled system processes and a single Secure Shell (SSH) process were running. This resulted in an average pre-test load of under 1% of total compute power. All distributed memory tests were run on a homogeneous compute cluster with each node consisting of two quad core 2.93GHz Intel Xeon processors with 16 gigabytes of shared memory.

5.5 Results

Experimental results are reported to: (1) quantify the speedup attained through parallelization, (2) explore the potential solution quality improvements afforded by increasing the area of the solution space explored, (i.e., the size of the solution set) without increasing the wall time, a.k.a, human waiting time, and (3) test the performance of the parallel algorithm for large ($n \geq 10,000$) PCR problems.

5.5.1 Speedup

To quantify the speedup parallelization affords in an SMP environment, IFS solution spaces containing 12, 24, 48, and 96 solutions using serial, six-core, and twelve-core implementations are generated. All speedup tests utilized the same 10,000 atomic unit polygon data, generated as described above. The total size of the solution set dictates the number of iterations required for the serial implementation. For example, the 48 element solution set required 48 sequential runs of the serial code. For parallel execution, IFS solution spaces of the total required size (e.g., 12, 24, 48 or 96) are generated. Synchronization occurs twice, once when all required IFS are generated and once at the conclusion of local search. Therefore, timings are computed as the sum of all required serial runs, in the serial case, and total execution time, in the parallel case. This chapter shows that parallelization improves overall solution quality while reducing runtime to explore a larger area of the solution space than is possible in serial. Figure 5.5Total Processing Time for a Varied Number of Solutions Using Serial and Parallel Implementationsfigure.5.5, below shows total computation time, using the 10,000 atomic unit test data, for serial, 6 core, and 12 core implementations. All variations in number of processing cores show a linear increase in total processing time. At the smallest solution set, serial processing of 12 realizations required over 55 minutes of wall time, while the six-core implementation required slightly more than 9 minutes and the twelve-core implementation required slightly less than 5 minutes. To generate a 96-element solution set, serial processing required more than 7 hours, parallel processing using six cores required slightly over one hour, and the twelve-core implementation required just over 38 minutes. Finally, average speedups of 11.5 times using twelve processing cores and 6 times using six processing cores are reported. The actual speedup in both cases almost equal to the theoretical speedups: 12 for 12-core processing and 6 for 6-core processing. This is inline with expectations using a low communication overhead model.

In the distributed memory environment same 10,000 atomic units polygon data is utilized, generating IFS solution spaces containing 128, 256, 512, 1024, and 2048 solutions using 13, 17, 25, and 49 processing cores. One core acted as a manager, resulting in 12, 16, 24, and 48 worker processes. Utilizing the 12 core implementation as a baseline, Figure 5.6Distributed Memory Implementation Speedup Using a 12-core Implementation as a Baselinefigure.5.6 illustrates the scaling performance of this implementation. Moving from 12 to 24 cores a roughly two time speedup for the 256, 512, and 1024 solution space tests is seen. Moving to the 2048 solutions space size, speedup decreases to one and a half times. Finally, using 48 workers, speedup is near

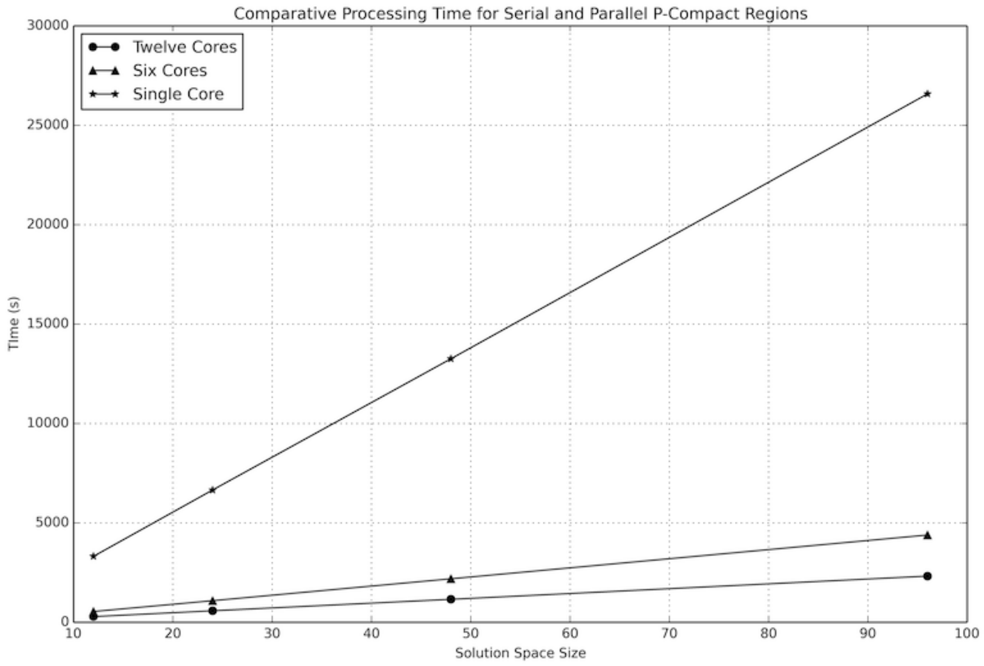


Figure 5.5: Total Processing Time for a Varied Number of Solutions Using Serial and Parallel Implementations.

the theoretical maximum, four times, only for the 256 and 512 solutions space tests. For 128, 1024, and 2048 solutions tests, speedups are between two and a half and three and a half times.

Figures 5.7Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87figure.5.7 and 5.8Final Result after Local Search Using 12 Cores and Generating 12 Solutions. Final Global Compactness, i.e., The Object Function Value as Defined by Equation 2, is 0.934figure.5.8 show the results of the PCR, using a 10,000 atomic unit, randomly distributed polygon test data set. p is set to be 152, which means initially 152 seeds were randomly selected to grow the regions. The seed selection follows a dispersion strategy. In Figure 5.7Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87figure.5.7, results of MERGE algorithm, generated from a serial program are shown. Figure 5.8Final Result after Local Search

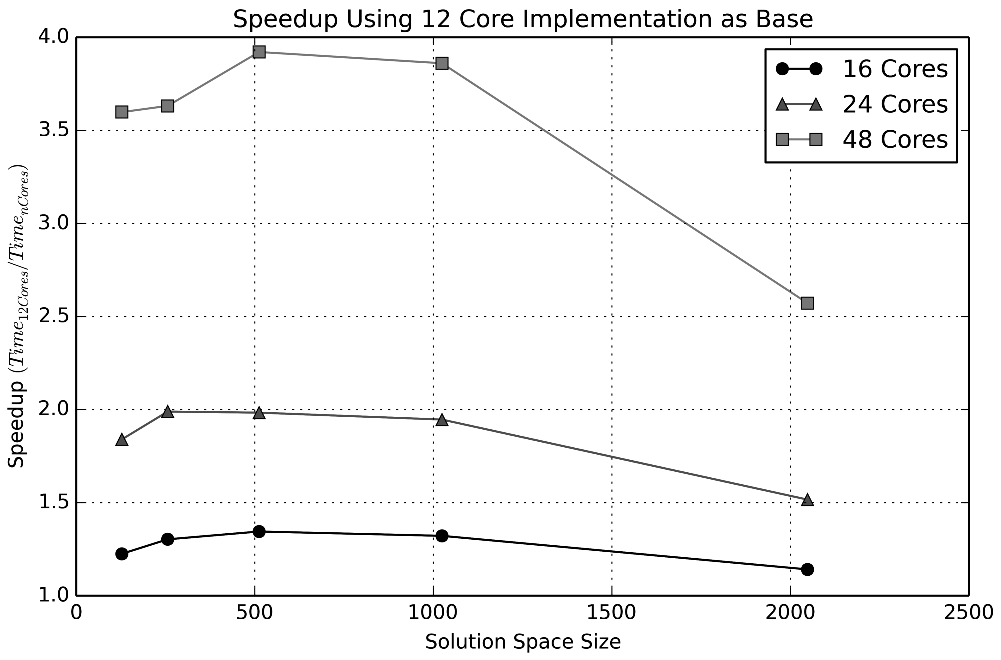


Figure 5.6: Distributed Memory Implementation Speedup Using a 12-core Implementation as a Baseline.

Using 12 Cores and Generating 12 Solutions. Final Global Compactness, i.e., The Object Function Value as Defined by Equation 2, is 0.934figure.5.8 depicts the best solution from a parallel with a solution set of 12. Visually, many regions become rounder in Figure 5.8Final Result after Local Search Using 12 Cores and Generating 12 Solutions. Final Global Compactness, i.e., The Object Function Value as Defined by Equation 2, is 0.934figure.5.8 than Figure 5.7Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87figure.5.7 after a larger area of the solution space has been explored. Quantitatively, the averaged compactness (NMI) increased to 0.934 (1 is the upper bound) from 0.87 for region plan in Figure 5.7Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87figure.5.7.

Additionally, a synthetically generated, spatially clustered data set was generated to test the performance of the distributed algorithm. Results are inline with previous tests in terms of overall performance with a slight degradation of approximately 7.5%

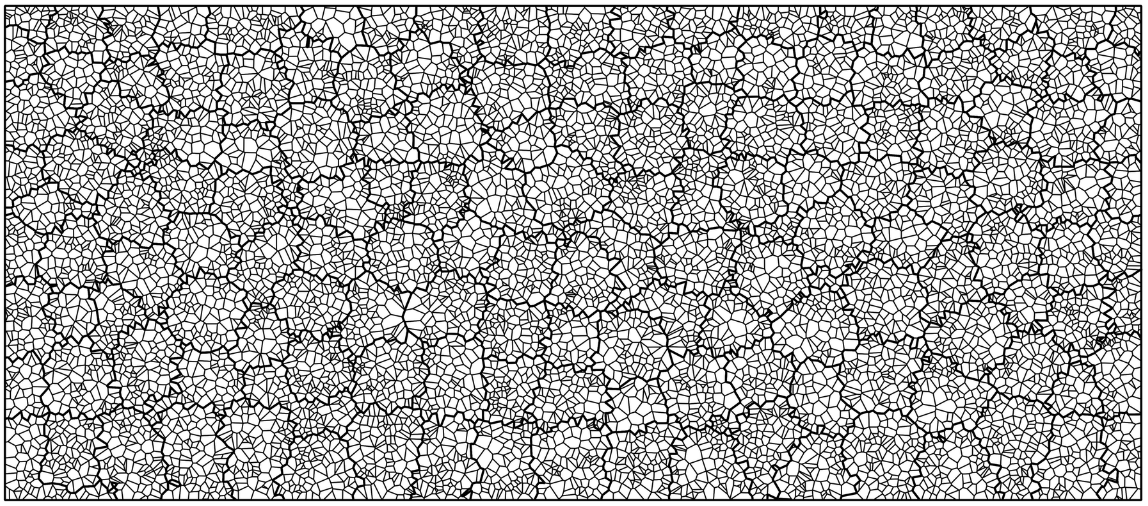


Figure 5.7: Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87.

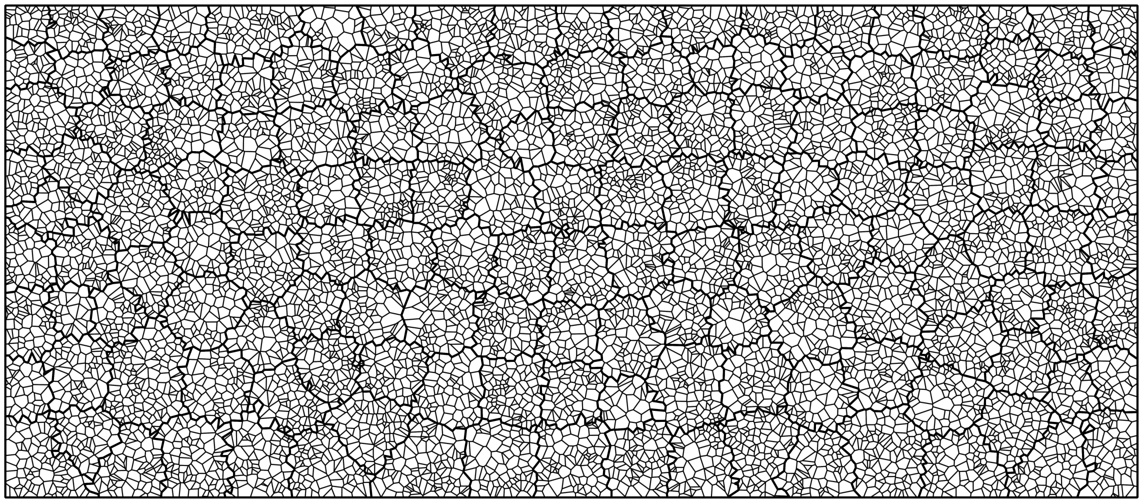


Figure 5.8: Final Result after Local Search Using 12 Cores and Generating 12 Solutions. Final Global Compactness, i.e., The Object Function Value as Defined by Equation 2, is 0.934.

in total speedup. Given that serial runtimes for clustered data are slightly less than those for randomly distributed data and constant communication costs, the slight reduction in parallel performance is due to communication costs requiring a slightly higher percentage of total runtime.This experiment validates the portability of the low communication parallelization strategy to a diverse array of spatial distribution patterns.

In summary, the results in this experiment verify that the proposed parallelization strategy is capable of achieving high speedup due to the minimization of inter-core communication and minimal intervention of the mother process during child processing. An increase in the solution quality is also attributable to the parallelization strategy. The next section analyzes the improvement of solution quality in detail.

5.5.2 Solution Quality

To test the impact of parallelization on solution quality in the SMP environment, the total runtime is fixed to 10 hours per iteration and varied the number of processing cores, starting in serial and then testing 2, 4, 8 and 12 cores. For example, the 8-core iteration was allowed to generate and solve for as many potential solutions as possible within a 10-hour time limit.

Figure 5.9The Distribution of Objective Function Values for 10 Hour Processing over a Range of Coresfigure.5.9 shows the range of solutions computed for a varied number of processing cores during 10-hour tests. All tests utilize a 2500 atomic unit Voronoi diagram. Table 1, below, describes the results of these tests. There is a general increase in the overall maximum objective function value as the number of processing cores increases. Additionally, it can be observed that the average objective function value stabilizes as the number of solutions increases. This is consistent with expectations as the total sample increases as a function of the number of processing cores. Finally, the range of solutions found also increases as the total number of solutions explored increases.

Interestingly, Figure 5.9The Distribution of Objective Function Values for 10 Hour Processing over a Range of Coresfigure.5.9, shows that a single-core processing generates high mean solution quality. This is not surprising, because (1) as a randomized greedy heuristic is used in the MERGE algorithm, it is possible to identify a good

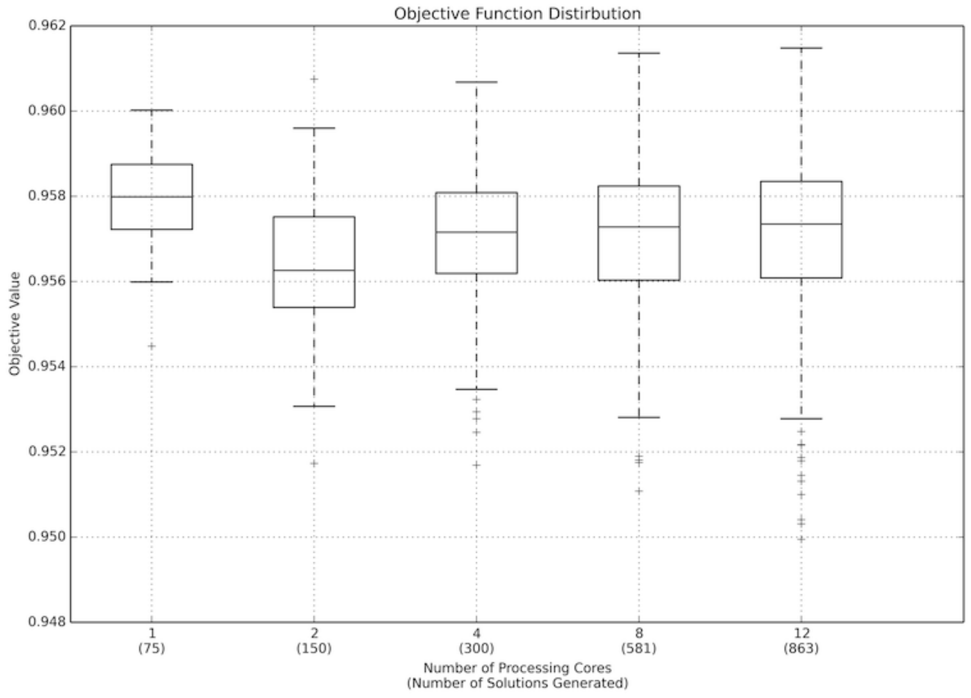


Figure 5.9: The Distribution of Objective Function Values for 10 Hour Processing over a Range of Cores.

solution by a single run; (2) when a larger solution set is searched by utilizing the power of multiple processing cores, the likelihood of finding outlier solutions, both good and bad, also increases. Therefore, an increase in the number of processing cores, and by extension an increase in area of the solution space explored, serves to generate a sufficiently large sample such that the impact of outliers is minimized.

Another interesting finding in Table 1 is the decrease in objective function value moving from two to four processing cores. Although the parallelization strategy generally realizes an improved objective function value without increased wall time, the heuristic search process itself does not guarantee that a best known solution will be found just because the number of solutions is large. Therefore, the reduction in maximum objective function between the two and four core tests as an example of the pseudo random traversal of the search space across independent process runs and suggest that parallelization offers one means by which a higher number of permutations can be explored in a reasonable amount of time.

5.5.3 Performance improvement in a SMP environment solving large PCR problems

Specific to large PCR problems, this experiment seeks to test improvement in computation time and solution quality attained through parallelization. Polygonal datasets with basic units of 10,000, 22,500, 40,000, 62,500 were used in this test. These tests aim to maintain approximately the same number (≈ 65) of basic units in each region for datasets with different sizes. This region size is chosen because earlier experiments with a real world dataset has shown that dealing 10-15 units to each region before moving to randomized greedy phase will help generate the best solutions (Li et al. 2014b). Therefore p increases proportionally to increase in number of polygonal units in a dataset: $p = 152$ for 10,000 datasets, $p = 364$ for 22,500 dataset, $p = 592$ for 40,000 dataset, $p = 922$ for 62,500 dataset. The exact value of p is a function of the seed selection process as the equidistant placement constraint requires seeds to be placed in a regular n by m grid without omitting points. This test was run in serial and using 12 processing cores. Total runtime and objective function value were retained for each iteration.

Figure 5.10Best Solution Quality for Large Problem Sets Using Serial and Parallel PCR Implementationsfigure.5.10 compares best solution qualities using serial and 12-core parallel PCR implementations. It can be observed that for large PCR problems, parallelization provides a means to explore a larger portion of the solution space and improve the likelihood of finding a high quality objective function value. This result is consistent with that found in Figure 5.7Regionalization Results after Phase I, Merge with an Initial Compactness of 0.87figure.5.7. On average the parallel implementation

computed an objective function that was 0.005 better than the serial solution. This value is expected to continue increasing when a much larger space is explored by increasing the total number of iterations.

From Figure 5.10Best Solution Quality for Large Problem Sets Using Serial and Parallel PCR Implementationsfigure.5.10, it is clear that the overall objective function value is higher, for both parallel and serial implementations, when the problem size is smaller. This is primarily due to the same stopping condition for SA used at local search phase used for all datasets. That means, when the same, or very close, number of local tuning steps can be utilized, the edge units receive more chances to be reassigned for smaller dataset (the number of regions is less). Therefore, a better regionalization plan can be generated. This result suggests that better algorithm performance does not rely solely on the parallelization strategy but also the proper customization of the algorithm according to its characteristics in a regionalization heuristic context.

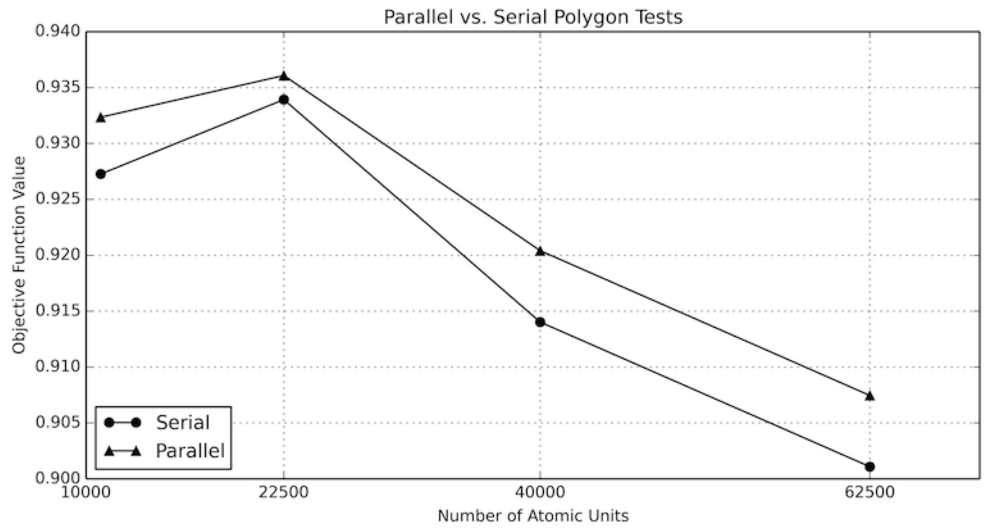


Figure 5.10: Best Solution Quality for Large Problem Sets Using Serial and Parallel PCR Implementations.

5.6 Discussion

This chapter reports efforts in designing a parallelized strategy for the MERGE heuristic to solve, to near optimality, a popular spatial analysis problem, a compactness-driven regionalization problem. This chapter explored implementations in SMP and HPC environments in order to generate minimal overhead benchmarks against which more complex, communication intensive parallelization efforts can be compared. Benchmarking must account for both solution quality and total compute time. In the SMP environment, the parallelization strategy succeeds in limiting parallel overhead with a consistently high speedup. Additionally, this chapter reports a linear improvement in solution quality as the total size of the solution space is increased. This is as anticipated, though it must be noted that the random nature of the heuristic could still generate outlier solutions at even the smallest solution space sizes. In the future, the distribution of the SMP solution quality can be compared with the distribution of more communication intensive implementations in order to assess the compute time and solution quality relationship.

In the HPC environment, the algorithm and parallelization strategy behave less consistently with total speedup decreasing as the total number of solutions to be computed increases. At the lowest solution space size, 128, underperformance is attributed to the overhead associated with initializing jobs, i.e. the overhead associated with initiating processing is a large component of the overall runtime. While, the decrease in speedup at larger global solution space sizes may probably due to additional communication overhead that is associated with receiving an IFS during the initialization, sending an IFS for local search, and finally receiving a final solution. The cost of these blocking communications is potentially causing idle processes and reducing the total anticipated speedup.

Computational efficiency within initialization and local search phases during the parallelization process is supported by the use of an efficient data structure to store a, potentially, high number of region growth and region change plans. As described in section 3, this data structure has a fixed length, growing linearly as a function of the total number of regions, for any given PCR problem. This data structure effectively controls the memory consumption of candidate region growth plans, making the algorithm memory efficient at large problem sizes.

This work will to contribute significantly to the CyberGIS high performance computing community. This chapter illustrates the successful deployment of a low-overhead parallel computation model to solve a regionalization problem in a SMP environment, which remains the most popular compute architecture for the majority of GIS researchers. The SMP algorithm was also ported into a HPC environment to demonstrate the cross-platform transplantability of the proposed parallelization methods. This new parallel implementation will also contribute to the spatial analysis and regional science communities by providing implementations which fully utilize all available compute resource.

In the future, I will work on the implementation of diversification and intensification strategies. These require inter-core communication and this implementation serves as a benchmark by which the solution quality versus processing time trade-off can be explored. Additionally, I seek explore a hybrid SMP / HPC deployment that leverages an OpenMP style shared memory space and a higher level MPI synchronization layer. Finally, I plan to improve the p-Compact Regions algorithm by integrating a new measure of mass compactness (Li et al. 2014c) and to exploit the applicability of the proposed model to solve other regionalization problems, such as the max-p-regions (Duque et al. 2012) and p-regions problem (Duque et al. 2011) optimally and efficiently.

Chapter 6

SPATIAL REGIONALIZATION AS AN EXPLORATORY CLASS OF PARALLEL ALGORITHM

Duque *et al.* (2012), citing Fischer (1980), defines a region as a 'set of spatially contiguous areas which show a high degree of similarity regarding a set of attributes'. Likewise, Shirabe (2009) suggests that districting is the process of 'aggregating predefined discrete geographic units into larger clusters'. Li *et al.* (2014b), in defining the p-Compact regions problem, set the attribute to be regional compactness, i.e., the likeness of a region to a circle. Within this work, regionalization is defined as the process by which atomic areal units are combined, under some set of constraints (common constraints include contiguity, compactness, or the application of some objective function), such that larger regions are generated (Duque *et al.*, 2007). Broadly, spatial regionalization algorithms, which enforce a contiguity constraint, can be divided into two classes: (1) flow based aggregation, and (2) attribute based aggregation. The former, p-Functional regions problems seek to define regions by interdependence using flows as a key means of aggregation (Duque *et al.*, 2012; Kim *et al.*, 2015). The latter, p-Regions problems, seek to define regions based on the degree of similarly (or dissimilarity), but not inter-dependence, of atomic units (Duque *et al.*, 2012) This work focuses on the p-Regions class of problem which seeks to aggregate n atomic units into p regions with $n \leq p$ (Duque *et al.*, 2011). For the remainder of this work, references to the p-Regions problem focus on attribute (in the case of the Max-P regions problem) or compactness (in the case of the p-Compact regions problem) based aggregation.

The process of spatial regionalization finds wide application in a number of domains. Wise *et al.* (1997) utilize spatial regionalization within a healthcare context in order to ensure confidentiality and Pathman *et al.* (2006) aggregates survey data from the zip code level to generate Primary Care Service Areas for analysis. Wise *et al.* (2001) suggests that regionalization is an essential exploratory spatial data analysis tool to allow users to experiment with the Modifiable Areal Unit Problem (MAUP) (Gehlke and Biehl, 1934; O'Sullivan and Unwin, 2010b), and identify outliers. Shirabe (2005) illustrate the selection of contiguous parcels in a land use planning case and Shirabe (2009) provides a generalized districting model designed for political, social, or economic regionalization. Openshaw (1977); Morril (1976); Pang *et al.* (2010) explicitly apply regionalization to the political redistricting problem. Miller and Shaw (2001) aggregate areal units to identify Transportation Analysis Zones (TAZ) and both Gonzalez-Ramirez *et al.* (2011), and Li *et al.* (2014a) utilize a compactness constrained regionalization approach to identify optimal delivery zones and TAZs, respectively. With vast increases in total spatial data sizes (Yang *et al.*, 2010) I anticipate additional application of regionalization as a means to reduce total problem size through aggregation, assuming that robust, computationally viable regionalization methods can be implemented.

Described from a computational standpoint, p-Regions algorithms seek to solve complex combinatorial problems and are known to be NP-Hard (Duque *et al.*, 2011). Mixed integer programming (MIP) formulation, used to compute exact solutions, is difficult due to the spatial contiguity constraint. Duque *et al.* (2011) describes three MIP techniques, using global and connected component sub-graph representations of a study area. Preventing cycling within a sub-graph while ensuring contiguity is maintained adds significant cost to the regionalization process. Duque *et al.* (2011) describes MIP experiments with runtimes which were capped at three hours, total problem sizes constrained to 49 atomic units ($n = 49$) and optimality achieved only with the smallest number of regions ($p \leq 6$ being the upper limit). Likewise, Kim *et al.* (2015) utilizes MIP to solve p-Functional regions problems with $n = 25$ and reports runtimes exceeding 17 hours for the computation of an exact solution, in some instances. Clearly, the addition of the contiguity constraint and the need to process larger, non-trivial datasets in a reasonable amount of time precludes the use of exact, MIP solution methods. Therefore, heuristic solution methods are employed for medium to large datasets where computational complexity is traded for, potentially, solution quality(Duque *et al.*, 2012; Li *et al.*, 2014a). In instances where the problem set is sufficiently large, exact solutions may not even be feasible.

p-Regions problems are ideally suited for distributed, parallel, implementation. Interest in parallel spatial analysis has recently increased as significant increases in data size, research in highly distributed parallel computing models, and the necessary hardware has become more widely available. It is within the context of the emergent Geospatial Cyberinfrastructure (Yang *et al.*, 2010; Wang, 2010, 2013; Wright and Wang, 2011) that I cite this work. In contrast to previous works, which have largely focused on decompose-conquer-merge strategies (Wang and Armstrong, 2005; Yang *et al.*, 2008; Padmanabhan *et al.*, 2011; Tang *et al.*, 2011; Rey *et al.*, 2013) for parallel spatial implementations, spatial regionalization is amenable to cooperative exploration of a solution space. Works focusing specifically on p-Regions problems in parallel domains are limited with Laura *et al.* (2015) providing a low communication implementation of the p-Compact-Regions problem. Widener *et al.* (2012) provides an implementation which focuses on the identification of spatial clusters, a loose analog to the p-Regions problem. Outside of the spatial analysis domain, parallel implementations have been designed to solve the quadratic assignment problem (QAP) (Gabrielsson, 2007; James *et al.*, 2009a,b), the generalized assignment problem (GAP)

(Liu and Wang, 2015) using genetic algorithms, and the multi-commodity capacitated network design problem (Crainic, 2002) using a parallel Tabu Search. Likewise, Ram *et al.* (1996) and Onbasoglu and Ozdamar (2001) provide parallel heuristic implementations to solve traveling salesman and generalized mathematical optimization problems, respectively.

This work focuses on the implementation of the Max-p regions (MPR) algorithm in a HPC environment using a cooperative heuristic implementation. I classify this implementation as *exploratory* using the taxonomy present in Chapter 2.

The remained of this work is organized as follows: In Section I 6.1 formulate the MPR problem and provide both a serial and parallel implementation. In Section 6.2 I describe a series of experiments in an HPC environment and Section 6.3 reports the results. Section 64. includes a discussion of the results and this chapter concludes with Section 6.5.

6.1 Max-p Regions Problem

In addition to the contiguity and minimization of inter-regional heterogeneity, constraints described above, MPR seeks to endogenously identify the maximum possible number of regions a given a study area can contain, i.e., p is not know apriori. This requires an additional minimum region size constraint, the floor constraint, to ensure that $n \neq p$ for all realizations.

The objective formulation, as presented by Duque *et al.* (2012) of the Max-P Regions problem statement is:

Minimize: (6.1)

$$Z = (-\sum_{k=1}^{n}\sum_{i=1}^{n} i^{k0}) * 10^h + \sum_{i}\sum_{j|j>i} d_{ij}t_{ij} \tag{6.2}$$

Subject to: (6.3)

$$\sum_{i=1}^{n} x_i^{k0} \leq 1 \forall k = 1, \ldots, n \tag{6.4}$$

$$\sum_{k=1}^{n}\sum_{c=0}^{q} x_i^{kc} = 1 \forall i = 1, \ldots, n \tag{6.5}$$

$$x_i^{kc} \leq \sum_{j \in N_i} x_j^{k(c-1)} \forall i = 1, \ldots, n; \forall k1 =, \ldots, n; \forall c = 1, \ldots, q \tag{6.6}$$

$$\sum_{i=1}^{n}\sum_{c=0}^{q} x_i^{kc} l_i \geq \text{threshold} * \sum_{i=1}^{n} x_i^{k0} \forall 1, \ldots, n \tag{6.7}$$

$$t_{ij} \geq \sum_{c=0}^{q} x_i^{kc} + \sum_{c=0}^{q} x_j^{kc} - 1 \forall i, j = 1, \ldots, n | i < j; \forall k = 1, \ldots, n \tag{6.8}$$

$$x_i^{kc} \in 0, 1 \forall i = 1, \ldots, n; \forall k = 1, \ldots, n; \forall c = 0, \ldots, q; \tag{6.9}$$

$$t_{ij} \in 0, 1 \forall i, j = 1, \ldots, n | i < j \tag{6.10}$$

where, k is a vector of potential regions, i is an area within the region, h is a scaling factor,d_{ij} is an element of a dissimilarity matrix, t_{ij} is a binary membership matrix, c is an index into a contiguity order used for maintaining the contiguity constraint, q is the maximum index in a given contiguity order, w_{ij} is a binary adjacency element, and l is a vector of attribute values for each i.

The first half of the objective function formulation $(-\sum_{k=1}^{n}\sum_{i=1}^{n} i^{k0}) * 10^h$ seeks to assign all atomic units to a region, maximize the total number of regions, and ensure that solutions with the maximum number of regions are guaranteed to provide a more optimal solution using a scaling factor (h), while the second half $\sum_i \sum_{j|j>i} d_{ij}t_{ij}$ sums global variance. This implies that solutions with more regions supersede solutions with fewer regions and that solutions with the same number of regions are rank-able by inter-regional heterogeneity, e.g. similarity. The objective function is subject to a number of constraints designed to enforce a lower threshold on region size and a contiguity constraint.

Citing Duque *et al.* (2012) constraint 6.4Max-p Regions Problemequation.6.1.4 ensures that each region contains one, and only one seed, i.e. $c = 0$ in the contiguity order. Constraint 6.5Max-p Regions Problemequation.6.1.5 ensures that each i is assigned to a single region, and a single contiguity order. In order for an atomic unit, i, to be assigned to a region, it must be spatially contiguous to some other atomic unit already in the region. Constraint 6.6Max-p Regions Problemequation.6.1.6 enforces this requirement. The minimum size threshold constraint is enforced by constraint 6.7Max-p Regions Problemequation.6.1.7. Constraint 6.8Max-p Regions Problemequation.6.1.8 ensures that pairwise dissimilarity is utilized as the metric for computing global heterogeneity and constraints 6.9Max-p Regions Problemequation.6.1.9 and 6.10Max-p Regions Problemequation.6.1.10 enforce a MIP problem.

A serial implementation of MPR consists of three distinction phases. First, preprocessing must occur in order to read a dataset into memory (incurring some Input/Output cost) and a data structure to store spatial contiguity, e.g., an adjacency matrix, must be created. Next, one or more initial feasible solutions (IFS) must be realized. Finally, a local search phase is entered which permutes the IFS, under all constraints, and seeks to improve the objective function value. Given these three distinct phases, it is possible to split the MPR objective function into two parts and assign each part to a given phase. IFS generation seeks to minimize $(-\sum_{k=1}^{n}\sum_{i=1}^{n} i^{k0}) * 10^{h}$, which, as described above, ensure that solutions with a higher number of regions are always favored. Contiguity constraints and permutation through edge reassignment ensures that p cannot vary during the local search phase. Therefore, only the minimization of $\sum_i \sum_{j|j>i} d_{ij} t_{ij}$ must be considered with each permutation.

One possible realization of a serial implementation is as follows:

```
Data: Spatial Weights Object, Attribute Vector, Floor Variable, Floor Threshold,
      Maximum Iterations, Optionally: Seeds
Result: Vector of length n + 1
regions = list
enclaves = list
candidates = sequentially increasing list of size n, e.g. [0,1,2,3,...,n]
current best p = 0
best solution = None
while maxiterations > 0 do
    while candidates do
        Randomly select a seed unit or select the first seed from user supplied seeds
        Remove the seed from the candidates list
        while Region size ¡ floor threshold do
            additions = neighbors to the current region if additions then
                randomly select a region from additions to add to the current region;
            else
                enclaves.append(seed);
                break;
            end
        end
    end
    if p in computed solution ¿ current best p then
        best solution = current solution current best p = p in current solution
    end
    maxiterations -= 1;
end
for All unassigned enclaves in best solution do
    Assign enclaves
end
```

Algorithm 3: Initial Feasible Solution Generation

Once the best IFS has been selected a local search phase, governed by Tabu Search, described below, is initiated. One possible implementation of this algorithm is:

```
Data: Spatial Weights Object, Attribute Vector, Current Solution, Max Failures,
      Optionally: Tabu List Length, Aspiration Criteria
Result: Vector of length n + 1
while maxfailure > 0 do
    valid = data structure of valid swaps Enumerate all potential edge reassignments
    for each edge reassignment do
        Check the floor constraint and add to valid if passed;
        Check the contiguity constrain and add to valid if passed;
    end
    Sort all edge reassignment for each valid reassignment do
        Check the objective function value eIfin tabu list check aspiration function;
        if aspiration true then
            make the move;
        else
            continue, do not accept move;
        end
        not in tabu list;
        make the move;
        reset the failure counter;
    end
    if no move made then
        maxfailures -= 1
    end
end
```

Algorithm 4: Local Search

Tabu Search

Tabu Search (TS) is a heuristic solution method developed to solve complex combinatorial optimization problems where some optimally definable solution exists within a finite set of potential solutions (Pham and Karaboga, 2000; Glover, 1989b,a). The application of TS suggests that fully enumerated solutions are not feasible in a given amount of time. Therefore, TS utilizes a two phase approach: (1) the generation of some initial feasible solution(s)[1] (IFS), and (2) local search. I focus on cooperative parallelization of the second phase and note only that IFS generation can have a significant impact on the final solution based on the initial selection of seed regions.

Local search leverages iterative permutation of a solution space to explore all possible neighbors to the current solution, i.e., the existing solution is slightly permuted. Within the context of MPR, this takes the form of edge reassignment, where a single atomic unit is reassigned to an adjacent region. With each iteration of the local search phase, all possible swaps are enumerated. The process of edge reassignment can become trapped in a local optima, that is a single solution within the solution space which is not the global best, but which is locally inescapable. In order to facilitate escape from local optima, TS uses an aspiration function which accepts either, the realization which most improves the global solution quality (objective function value) or accepts the solution which degenerates the global solution least Pham and Karaboga (2000). This process can introduce cycling, and a tabu list of prohibited moves is maintained. In pseudo code the local search phase of the algorithm is:

[1] See James *et al.* (2009a) for a multi-start tabu search that reseeds the IFS.

```
Data: Neighborhood
Result: Improved optima or total iterations counter incremented
best = CurrentOptima;
MaxFailures = User defined number of failures;
while CurrentFailures <= MaxFailures do
    EvaluateSwaps for the current solution;
    if swap in EvaluateSwaps > best then
        Check the TabuList;
        if not tabu then
            MakeSwap & Update CurrentOptima;
        end
        CheckAspiration Function;
        if CheckAspiration Function is True then
            MakeSwap & Update CurrentOptima;
        end
        if swap in EvaluateSwaps < best then
            CurrentFailures += 1
        end
    end
end
```

Algorithm 5: TS Local Search Phase

Local search continues until the maximum number of failures is attained. As moves are accepted they are added to the tabu list, forcing previous moves to increment out of scope. In this way a previously tabu move will become available again. In terms of this usage case, local search swaps all members of a region adjacent to another regions, checks that the swap does not violate the contiguity constraint or floor constraint, and accepts the swap which improves the objective function most.

6.1.1 Parallel Max-p Regions

Interest in parallel implementations of TS began almost immediately after initial serial publication of the heuristic (Taillard, 1991) following the initial articulation by Glover (1989b) and Glover (1989a). Crainic *et al.* (1997) developed a taxonomy of parallel implementation techniques in order to begin classifying the diverse set of implementations developed. This taxonomy provides the means to classify implementations not only by solution quality and speed, but also by specific implementation details. This taxonomy also allows new implementations to focus on those techniques which have proven most successful in providing high quality answers with fast convergence.

Crainic *et al.* (1997) classifies parallel TS implementations based on three criteria: search control, control and communication, and search differentiation. Search control indicates whether the local swap TS phase is controlled by a single processor, called 1-control, or distributed over multiple cores, p-control. A TS utilizing 1-control operates from a single solution and distributes swap computation at each iteration over available cores. In contrast, a p-control TS runs n concurrent TS, where n is the number of cores. This classification identifies the point of parallelization and the number the independent, concurrent searches which are occurring.

Control and communication describes the type of control (synchronous or asynchronous) and quantity of information passed between cores. A distinction is drawn between control and communication within the 1-control and p-control classifications. Control and communication within the 1-control category can be rigid, indicating that a single core controls communication between children and waits to synchronize processing until all cores have completed their task, e.g., bulk synchronization to a single master controller. Alternatively, a knowledge synchronized approach extends the rigid control technique by distributing information between cores at synchronization. Like the rigid control scheme bulk synchronization occurs, but the information is synchronized to both the master and all workers. Within the p-control category, collegial communication shares an improved solution asynchronously with other processes. A knowledge collegial communication method extends collegial communication by sharing not only a solution, but also solution characteristics, e.g., frequency of successful swaps, search parameters, swap history. These characteristics are then analyzed by a core and used to improve search trajectory.

Finally, search differentiation indicates how the TS local search phase is initiated. Crainic *et al.* (1997) identifies four different differentiation classes: Single Point Single Strategy (SPSS), Single Point Different Strategies (SPDS), Multiple Points Single Strategy (MPSS), and Multiple Points Different Strategies (MPDS). Single point strategies begin the local search phase with the same starting solution, while multi-points strategies begin the local search phase with a different starting solution. Single strategy and different strategy dictate whether the local TS phase utilizes the same parameters (TS list length, differentiation and aspiration criteria, total local failures, etc.) or divergent parameters. Given this taxonomy, it is possible to locate TS implementations within each of the three broad categories, search control, control and communication, and search differentiation.

Citing their work within Crainic's taxonomy, James *et al.* (2009a,b) explore variable tabu list length, diversification, intensification, and memory management issues found in parallel TS implementations. Taillard (1991) first introduced variable tabu list length and James *et al.* (2009a) find that variable TS list length diversifies solution trajectory. This can be implemented as a function of n, the total number of atomic geometries in the input data set, where total TS list length is described by:

$$\begin{aligned}
&s_{min} = n \cdot 0.9 \\
&s_{max} = n \cdot 1.1 \\
&\Delta = \{x| \in \mathbb{N}, 0 \leq x \leq (s_{max} - s_{min})\} \\
&TS_{list_length} = n \pm \Delta
\end{aligned} \tag{6.11}$$

Diversification indicates the perturbation method employed to escape a potential local optima. This processes is initiated when a core has been unable to make an improvement to the solution in a given number of failures. Intensification is the process by which the current optimal solution is propagated through the search space and explored by multiple cores. James *et al.* (2009a) implements intensification by asynchronously testing a single core solution against a shared memory solution space. If the current solution is a new global optima, it is propagated to 50% of the search space, thereby intensifying TS search in that neighborhood. A 'kick' algorithm, which performs a small number of random assignment swaps, is also used to slightly perturb the high quality solution (James *et al.*, 2009a). The addition of intensification helps reduce total parallel processing time as globally, TS does not need to run as long before the algorithm converges. Finally, memory management in SPDS and MPDS implementations is essential to avoid concurrent overwrites and provide a means of masterless communication (p-control). James *et al.* (2009a) provides strategies to facilitate complex inter-core communication using a series of locks and semaphores. These implementations provide higher quality results than more simplistic single strategy implementations.

In creating a parallel Max-p regions implementation I focus on maximizing the probability of finding a high-quality solution (minimizing the objective function) at some cost to overall performance. Tom y knowledge, quantification of this relationship has not been performed; a single method with exceptional speed and an assurance of (near) optimality has not been developed. Therefore, I have designed this implementation with the goal of maximizing solution quality. I hypothesize that is possible to increase inter-core communication during the local search phase and leverage intensification and diversification strategies to force the algorithm to converge more quickly. That is, rapid failure to make a successful solution permutation, local to a single core, and subsequent iteration to the next feasible solution within the global solution space reduces global processing time while still locating high-quality solutions.

As with the serial implementation, the parallel MPR consists of three discrete phases. First, file I/O occurs, a spatial contiguity object is generated, and the necessary attribute vectors are generated. This process is identical to the serial approach. Next, i initial feasible solutions (IFS) are generated using an embarrassingly parallel implementation. Finally, local search is performed using a cooperative tabu search (TS). The following describes phases two and three of the implementation.

Within the following section, I refer to the local solution space as the shared memory, local to each compute node, within which c solutions are stored, where c is the number of cores per node. The global solution space is the aggregate of all local solution spaces, e.g., the global solution space in an eight node, eight core per node environment would consist of 8 local storage spaces and $8 * j$ solutions in the global solution space, where j is the size of each local solution space.

I implement a hybrid distributed and shared memory implementation in order to leverage the efficiency of a low communication cost shared memory space and the scalability of message passing approaches as the total number of available nodes increases. The shared memory code utilizes Python multiprocessing and CTypes.

MPI is used to manage all message passing. Throughout, bulk synchronization is used as asynchronous MPI based synchronization proved extremely difficult to manage.

Initial Feasible Solution Generation

Each compute node generates a user defined number of IFS within some number of iterations. A shared memory space and associated lock are created local to each node with a total size equal to $jxn + 1$, where j is the number of IFS to be generated and n is the total number of atomic units. In Equation 6.12Initial Feasible Solution Generationequation.6.1.12, a sample IFS with $j = 2$ is provided. All references by position are zero offset. At positional index zero, each row holds the current number of regions, 7 and 4 in this example. The remainder of each row contains a representation of the current solution with the value representing membership within a region and the positional index representing the atomic unit ID. For example, in row one region 1 is composed of units 2 and 5. Region 2 is composed of units 1, 3, and 4.

$$\begin{array}{cccccc} 7 & 2 & 1 & 2 & 2 & 1 \\ 4 & 1 & 2 & 2 & 1 & 2 \end{array} \tag{6.12}$$

This representation provides three key benefits. First, a regular, matrix representation of the global solution space facilitates the user of shared memory spaces. Second, representation of each solution, within the local solution space, as a vector provides an efficient representation for communication between nodes. Finally, by encoding the objective function value into the solution, synchronization of two separate data structures is not required.

Once the shared solution space is initialized, each available core begins the generation of an IFS. Recall that the MPR objective function biases solutions with a larger number of regions. Therefore, the parallel generation of IFS must ensure that the local solution space contains the current best known p. Secondary to that goal, diversity should be ensured to leverage the benefits of beginning local search with a diverse set of solutions.

Algorithm 6Initial Feasible Solution Generationalgocf.6 provides a pseudocode implementation of this process. This is a two step process which iterates until a maximum number of iterations have been performed. First, each core generates an IFS as per the serial implementation without assigning enclaves. Next, the local solution space is queried for the current maximum and those indices which contain a poorer p count. These queries can be processed in one of three ways: (1) if the current solution is poorer than current maximum, the iteration counter is deincremented, (2) if the current solution bests a currently saved solution at some positional index, the solution space is locked and the current solution added to that index, or (3) if the solution is worse than the current best and poorer than all currently stored solutions the iteration counter is zeroed. Manipulation of the iteration counter local to each core, facilitates node-local load balancing. This process allows a solution space to be iteratively populated, and if a new global best is identified, iteratively repopulated, until all rows contain a solution with an equal number of regions.

```
Data: Spatial Weights Object, Attribute Vector, Floor Variable, Floor Threshold,
      Maximum Iterations, Optionally: Seeds
Result: Vector of length n + 1
generateIFS (As Above)
currentregions = number regions in IFS
poorerindices = where(local solution space $p$ ¡ current $p$)
currentmax = max(local solution space)
if $currentregions < currentmax$ then
    Deincrement iteration counter by 1
else if $length(poorerindices) > 0$ then
    Add the solution to the solution space
else
    Deincrement iteration counter to zero
end
```

Algorithm 6: Initial Feasible Solution Generation

Once all node-local iterations are completed, a bulk synchronization phase is used to standardize p across the global solution space, where p is the number of regions in a solution. First, p is broadcast to all nodes. This ensures that all nodes know the p value of all other nodes. Local to each node, p_{local} is compared to all other p, p_{global}. If $p_{local} < max(p_{global})$, a node with the maximum p_{global} is randomly selected and an asynchronous, MPI managed, `Get` request is made to copy and replace the local solution with the selected node's solution. Once all solutions within the global solution space have standardized p, enclave assignment is completed in an embarrassingly parallel manner. Within the enclave assignment phase, index zero of each solution is updated to include the current objective function value.

Cooperative Tabu Search

The implemented algorithm is defined as a p-control, Multi-Point Different Strategies, cooperative Tabu search with periodic bulk synchronization. This approach runs p concurrent tabu searches from different starting solutions, each with different, stochastically generated TS parameters, and periodically synchronizes results to propagate the 'best' solutions across all workers. Recall that IFS generation completed with each local storage space being composed of a shared $jxn + 1$ solution matrix with the zero index within each row containing the second component of the objective function, $\sum_i \sum_{j|j>i} d_{ij}t_{ij}$ and the global solution space has a standardized p for all solutions.

Prior to initiating local search five parameters are set. First, each core is parameterized with a maximum number of iterations that define the total number of independent local search phases a given core can make. Next, a maximum number of failures is provided. This value is permuted, local to each code using Equation 6.13Cooperative Tabu Searchequation.6.1.13:

$$maxfailures = maxfailures + maxfailures * U(-1.1, 1.2) \qquad (6.13)$$

where U is a uniform distribution. Stochastically computed maximum failures are shown to improve the global solution quality (James *et al.*, 2009a). Third, a constant maximum number of iterations per core is added to control the number of times that a local search will be applied to the local solution space. Fourth, an integer identifier is assigned to each core in order to facilitate traversal across the solution space, described below. Finally, each core determines tabu search list length, a differentiation of the search strategy introduced by Taillard (1991), as defined by:

$$\begin{aligned}
&s_{min} = n \cdot 0.9 \\
&s_{max} = n \cdot 1.1 \\
&\Delta = \{x| \in \mathbb{N}, 0 \leq x \leq (s_{max} - s_{min})\} \\
&TS_{list_length} = n \pm \Delta
\end{aligned} \tag{6.14}$$

James *et al.* (2009a) finds that variation of this parameter helps to diversify the trajectory of solution traversal.

The remainder of this section describes the process of solution traversal, intensification, and diversification from the perspective of a single core. This process occurs concurrently across all cores on all nodes. First, the core applies Tabu Search to the solution corresponding to the solution row with the same identifier, i.e., core one works on row one. Once completed, the core locks and queries the global solution space, comparing the newly computed solution to all other local solutions. If the solution is better than the solution currently held in the corresponding index, the local solution space is updated. If the solution is better than all other solutions in the solution space, it is intensified to some percentage of the local solution space. If the current solution is worse than all others, or the solution has not been improved by the Tabu Search, the solution is diversified and then written to the global solution space. Once completed the core identifier is incremented by 1 (or set to zero in the case where $index + 1$ is larger than j) and the next solution row processed.

```
Data: Spatial Weights Object,Attribute Vector, MaxFailures, MaxIterations,
      Optionally: IntensificationPercentage, Aspiration Criteria
Result: Local Search Object
performLocalSearch (As Above)
Lock the solution matrix
if $currentsolution > allcurrentsolution$ then
    Propagate current solution to IntensificationPercentage of the solution space
else if $currentsolution > currentsolutioninlocalsolutionspace$ then
    Add the solution to the solution space
else
    Diversify the solution
end
if $coreid + 1 > j$ then
    coreid = 0
else
    coreid += 1
end
```

Algorithm 7: Local Search using Tabu Search

Intensification and diversification provide two methods for improving the solution quality. Intensification explicitly suggests that the current best solution is either the global optima or a promising path to continue to explore. For this reason, that solution is intensified to some percentage of the solution space for a variable number of cores, with divergent TS parameters, to process. Intensification is parameterized to allow the user to define the percentage of intensification. It is possible that the assumption of high solution quality is erroneous and therefore, intensification is not to 100% of the solution space. Conversely, a non-improving solution is assumed to be a dead-end. Therefore, the solution is diversified, in order to significantly alter the current realization, prior to allowing the next core to perform local search.

Diversification is achieved by selecting the region composed of the maximum number of atomic units and iteratively reassigning edge units to adjacent regions until any additional atomic unit removal would violate the floor constraint. [2]

6.2 Experiments

Two different data generation processes were employed in order to explore the impact of varying Tabu Search parameters, intensification and diversification strategies, and the impact of spatially autocorrelated data. Within this secretion I first describe the data generation process and then present test performed.

The first test data set, Figure 6.2Experimentssection.6.2 was generated in an effort to control the regionalization process and identify a know optimal solution. To that end, both random and spatially clustered point patterns were generated over a defined extent. From these point patterns, Voronio diagrams were generated and the area of each polygon computed. Using a derivation of the IFS generation code, regions were generated using random initial seed assignment and grown until a threshold total area achieved. Once this process was completed, enclaves were assigned to the smallest adjacent region in an effort to balance total area without the need to perform a complete area based regionalization. All members of a given region were attributed the same region identifier. Recall from Equation 6.2Max-p Regions Problemequation.6.1.2 that the right hand side of the objective function describes sum of the inter-regional variance. By using the region identifier as that attribute, it is known a priori that the right hand size should sum to zero. It is not possible guarantee that the known solution is in fact an optimal space partitioning at any

[2] I also test similar logic where the least homogeneous region reassigns units with little change to the overall performance of the algorithm.

other p value, only that it is optimal at the p defined in the data generation process. Random and clustered datasets with 100 and 500 polygons were generated.

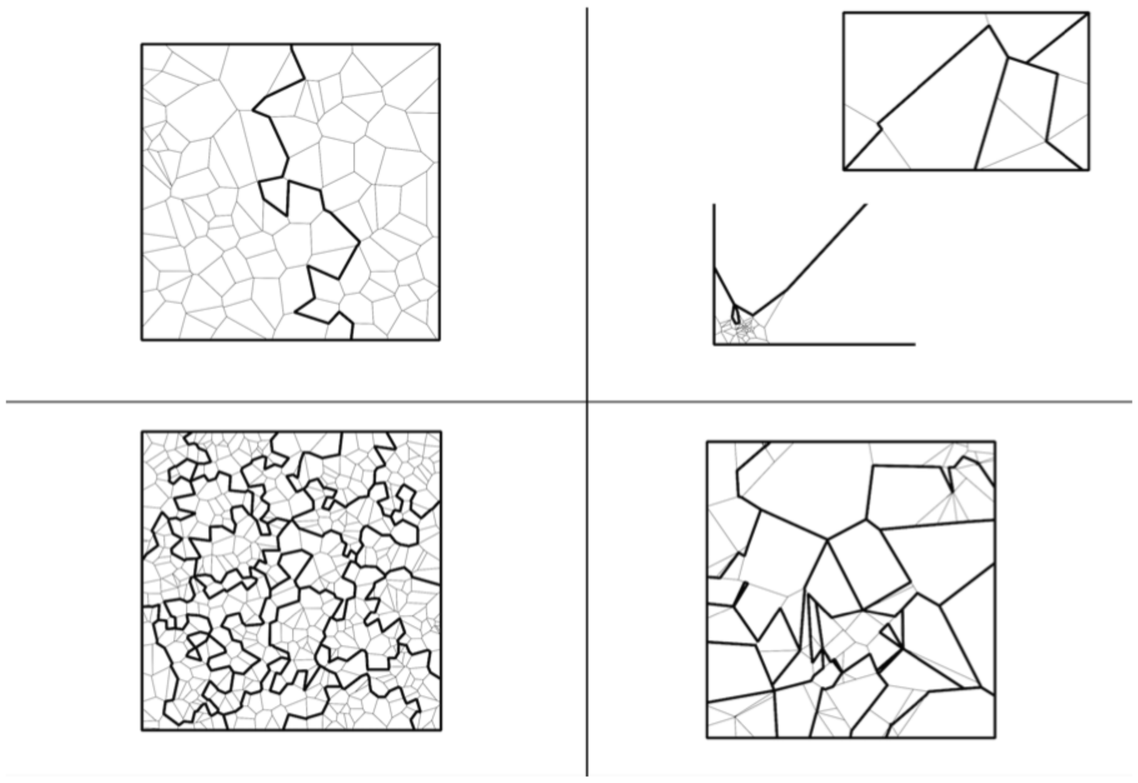


Figure 6.1: Synthetically Generated Data, Clockwise, 100 Randomly Distributed Polygons, 100 Clustered Polygons with Inset to Highlight Tight Clustering, 500 Clustered Polygons and 500 Randomly Distributed Polygons. All Figures Include Region Overlay of the Known Optimal Partitioning at a given p.

The second data set, Figure 6.2Experimentsfigure.6.1 is designed to test the algorithm when spatially autocorrelated data is present. Data sets with 100, 225, 400, and 625 elements in regular lattices were generated to use as input areal units. Each geometry is attributed with a uniformly distributed random scalar in the range [-1, 1] to be used as the threshold. Next, spatially lagged attributes were generated with $\rho = \{0, 0.1, 0.3, 0.5, 0.7, 0.9\}$. Optimal solutions for the MPR problem are not known for these data sets.

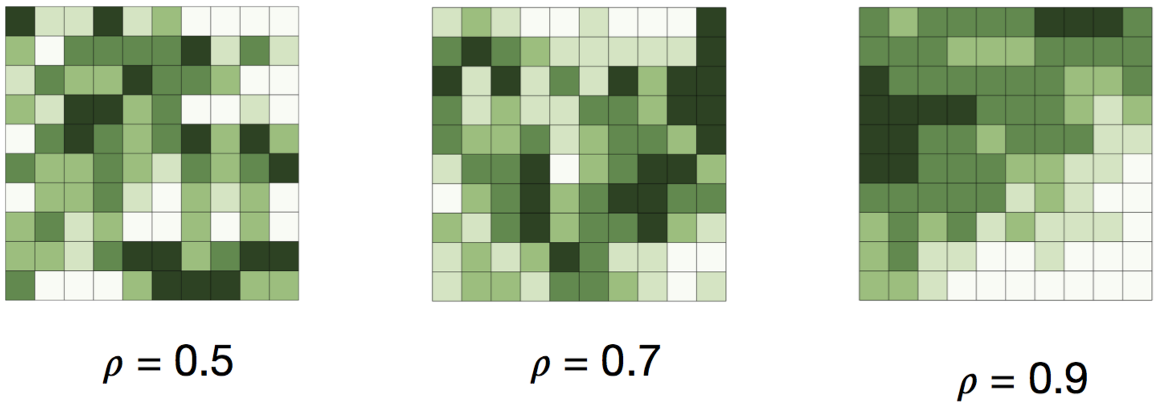


Figure 6.2: Synthetically Generated, Spatially Autocorrelated Data on a 10 X 10 Lattice.

Using both datasets five tests were performed to explore the impact of varying Tabu Search parameters, as well as the impact of increased quantity and type of inter-core communication. First, TS search parameters were allowed to vary. These parameters include the Tabu List length and total number of local search move failures before termination. Next, solution space traversal iterations from the set 1, 2, 4, 8, 16, 32, 64 were tested. That is, each processing core is allowed to iterate around the solutions space some number of times, before processing is terminated. As in the first test TS parameters are allowed to vary. Third, the maximum number of iterations was set to 16, TS parameters allowed to vary, and intensification percentages between 20% and 80% at 20% intervals tested. Next, intensification strategies were disabled and diversification enabled, meaning that a non-improving solution is permuted with the assumption that non-improvement is indicative of either poorly defined TS parameters or a local optima. Finally, TS parameters were allowed to vary, maximum iterations fixed to 16, intensification to 60%, and diversification enabled. Given the stochastic nature of the algorithm, each test was run 400 times.

All experiments were performing using four nodes from the GeoDa Center computing cluster, a homogeneous high performance computing environment. Each node is composed of two quad-core Intel Xeon X5355 processors running at 2.93GHz. All cores have access to 16 Gigabytes of shared memory RAM and are inter-conencted via a high speed infiniband network. Input shapefiles, described below, are stored on a network Lustre file system which is accessed by the rank 0, managing node.

6.3 Results

Table 6.1Maximum Iteration Results Showing Mean Objective Function Value after IFS Generation, Mean Final Solution, Mean Improvement, and Best Final Solutiontable.6.1 reports the results from testing increases to the maximum number of times a given core iterates around the solution space. The optimal partitioning is unknown for all datasets. Constraining the region growth phase of the IFS was possible for the 100 random and cluster polygon tests, resulting in a known optimal solution. Constraining p for the 500 solution tests was not possible without significantly altering the algorithm. Therefore, the known optimal solution for the 100 polygon tests is zero and parameterization performance is tested using the 100 polygon algorithm.

Iterations		100 Random	100 Clustered
1	$\overline{IFS}$	4.479761	17.746427
	$\overline{Final}$	4.070746	14.814066
	$\overline{Improvement}$	0.409016	2.932362
	Best Obj.	0.98000	13.42909
2	$\overline{IFS}$	4.622725	16.079658
	$\overline{Final}$	3.208718	15.567508
	$\overline{Improvement}$	1.414007	0.512150
	Best Obj.	0.979167	9.287855
4	$\overline{IFS}$	4.616459	15.035625
	$\overline{Final}$	2.752254	13.177657
	$\overline{Improvement}$	1.864205	1.857968
	Best Obj.	1.911111	11.520000
8	$\overline{IFS}$	4.534005	14.755537
	$\overline{Final}$	2.280461	11.217338
	$\overline{Improvement}$	2.253544	3.538199
	Best Obj.	1.911111	0.976744
16	$\overline{IFS}$	4.264563	16.931734
	$\overline{Final}$	2.112895	10.921694
	$\overline{Improvement}$	2.151668	6.010041
	Best Obj.	1.911111	0.976744
32	$\overline{IFS}$	4.753670	15.171829
	$\overline{Final}$	2.011258	10.935356
	$\overline{Improvement}$	2.742412	4.236473
	Best Obj.	1.911111	0.976744
64	$\overline{IFS}$	4.551710	17.268747
	$\overline{Final}$	1.967355	10.148520
	$\overline{Improvement}$	2.584355	7.120227
	Best Obj.	1.911111	0.976744

Table 6.1: Maximum Iteration Results Showing Mean Objective Function Value after IFS Generation, Mean Final Solution, Mean Improvement, and Best Final Solution.

Aditional iteration succeeded in decreasing the mean objective function value for all tests. The 100 random polygon tests at iterations equal one and two highlight the random nature of the algorithm as these two found the best, non-optimal solution. For all tests, significant improvement between the IFS objective function and the final objective function is observed. Figure 6.3Resultstable.6.1 shows the distribution of solutions from the 100 iterations. In the randomly distributed data, cases where a processing cores iterates 8 to 64 times around a solution space resulted in significantly higher convergence to local optima. The same phenomena is observable in the clustered data, though the convergence is not as distinct.

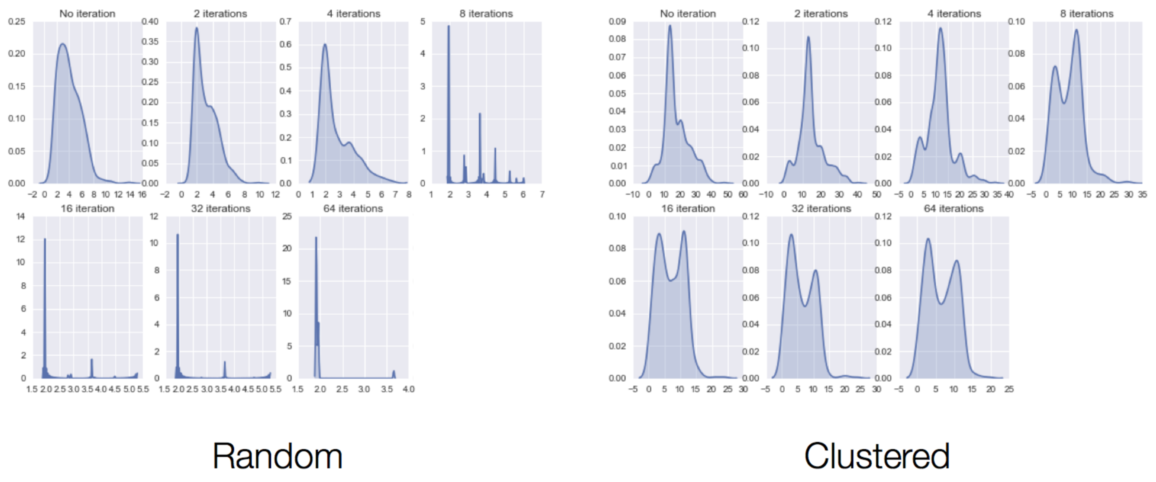


Figure 6.3: Distribution of Solution Values in Random and Clustered, 100 Polygon Datasets Achieved by Varying the Total Iteration Count.

Setting the maximum number of iterations to 16 and testing intensification between 20% and 80% at 20% intervals, Table 6.2Results of Intensification for the 100 Random and Clustered Polygon Teststable.6.2 illustrates significant improvement to the best known objective function and the frequency with which the known optima was reached. Unclustered data consistently reached the optima more frequently than clustered data. Additionally, no correlation os observed between the quantity of intensification and the frequency with which the optimal solution was achieved.

% Intensification		100 Random	100 Clustered
20%	$\overline{IFS}$	4.482461	15.587122
	$\overline{Final}$	0.014694	1.091752
	$\overline{Improvement}$	4.467767	14.495370
	Best Obj.	0	0
	Best Obj. Frequency	397	346
40%	$\overline{IFS}$	4.561280	15.069903
	$\overline{Final}$	0.004898	0.908355
	$\overline{Improvement}$	4.556382	14.161549
	Best Obj.	0	0
	Best Obj. Frequency	399	355
60%	$\overline{IFS}$	4.551993	15.825269
	$\overline{Final}$	0.014694	0.775023
	$\overline{Improvement}$	4.537299	15.050246
	Best Obj.	0	0
	Best Obj. Frequency	397	362
80%	$\overline{IFS}$	4.602588	14.941264
	$\overline{Final}$	0.021924	0.925188
	$\overline{Improvement}$	4.580665	14.016077
	Best Obj.	0	0
	Best Obj. Frequency	395	355

Table 6.2: Results of Intensification for the 100 Random and Clustered Polygon Tests.

Figure 6.3Resultstable.6.2 shows the solution distributions summarized in Table 6.2Results of Intensification for the 100 Random and Clustered Polygon Teststable.6.2. Convergence rates are significantly higher using intensification, but are attributable to the use of the previous iterative strategy. In contrast to the previous strategy, convergence rates, to the known optimal solution, are higher. Therefore, I conclude that this convergence, unseen using iteration alone, is a function of the intensification operator.

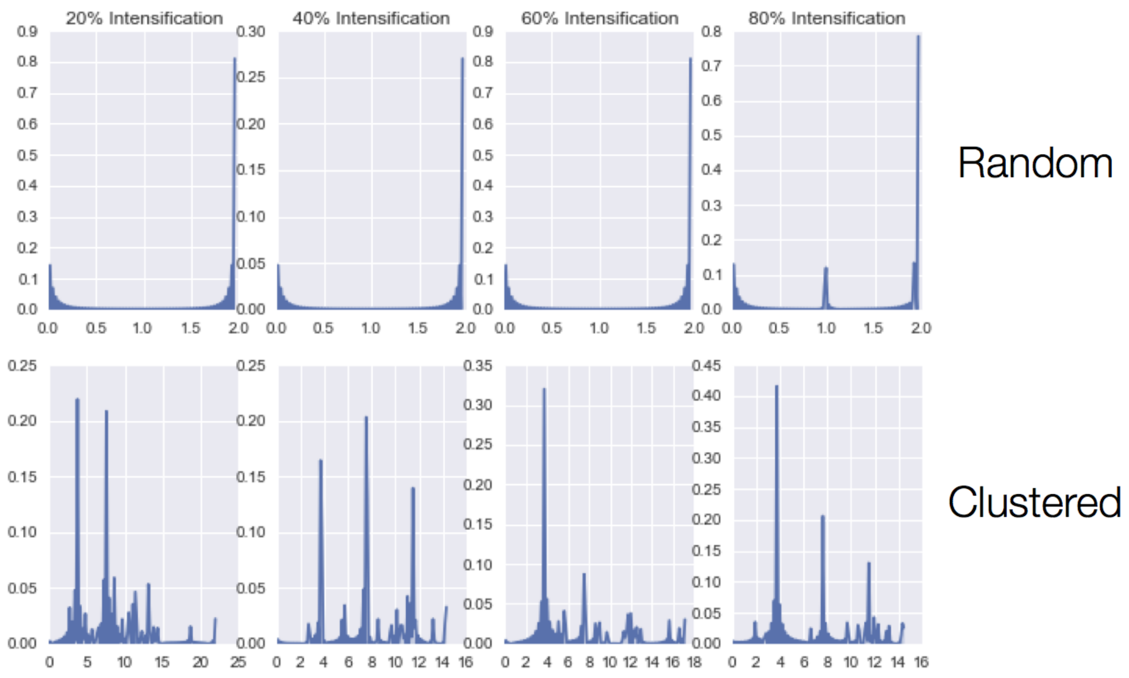


Figure 6.4: Solution Distributions with Intensification Between 20% and 80% for the 100 Random and Clustered Polygon Tests.

Removing all intensification and setting iterations to 16 it is clear that diversification alone is not sufficient to escape local optima, Table 6.3One Hundred Polygon Tests with Iterations Set to 16 and Diversification Enabledtable.6.3. In fact, the frequency with a high quality solution is found suggests that intensification is significantly more important than the diversification algorithm tested. Comparing these results to the intensification results, it is clear that the diversification operator is performing too well. That is, the solution is being diversified sufficiently that the algorithm is becoming trapped in a local optima, $1.9\bar{1}$ and 0.9767 for random and clustered data, respectively.

	100 Random	100 Clustered
$\overline{IFS}$	4.506932	14.80696
$\overline{Final}$	2.028983	7.176011
$\overline{Improvement}$	2.47795	7.630949
Best Obj.	1.911111	0.976744
Best Obj. Frequency	267	25

Table 6.3: One Hundred Polygon Tests with Iterations Set to 16 and Diversification Enabled.

Finally, Table 6.4Results for Full Cooperative, Parallel Tabu Searchtable.6.4 shows the results of testing solution space iteration, intensification, and diversification for the 100 and five hundred polygon data sets. Optimality is being regularly achieved in the 100 polygon tests and low mean final solution values. The 500 polygon tests show significant improvement due to local search with over 50% improvement in the 500 clustered tests.

	100 Random	100 Clustered	500 Random	500 Clustered
$\overline{IFS}$	4.485775	14.510799	8223.544158	6404.556761
$\overline{Final}$	0.095361	1.402004	5978.578674	3089.399333
$\overline{Improvement}$	4.390414	13.108795	2244.965483	3315.157427
Best Obj.	0	0	4292.370605	1750.955933
Best Obj. Frequency	383	288	1	1

Table 6.4: Results for Full Cooperative, Parallel Tabu Search.

Figure 6.3Resultstable.6.4 reports the distribution of testing the algorithm across regular lattice data sets with varying degrees of spatial autocorrelation. For all tests greater than 100 polygon units, diversification, and the fully cooperative method are consistently outperformed by methods utilizing iteration and / or intensification. This is inline with previous tests; the diversification operator is over performing and local optima are not escaped. This behavior is consistent across all value of ρ. Interestingly, at $n = 10^2$ polygon units, it appears that as ρ increases, the dependence upon the diversification operator appears to also increase.

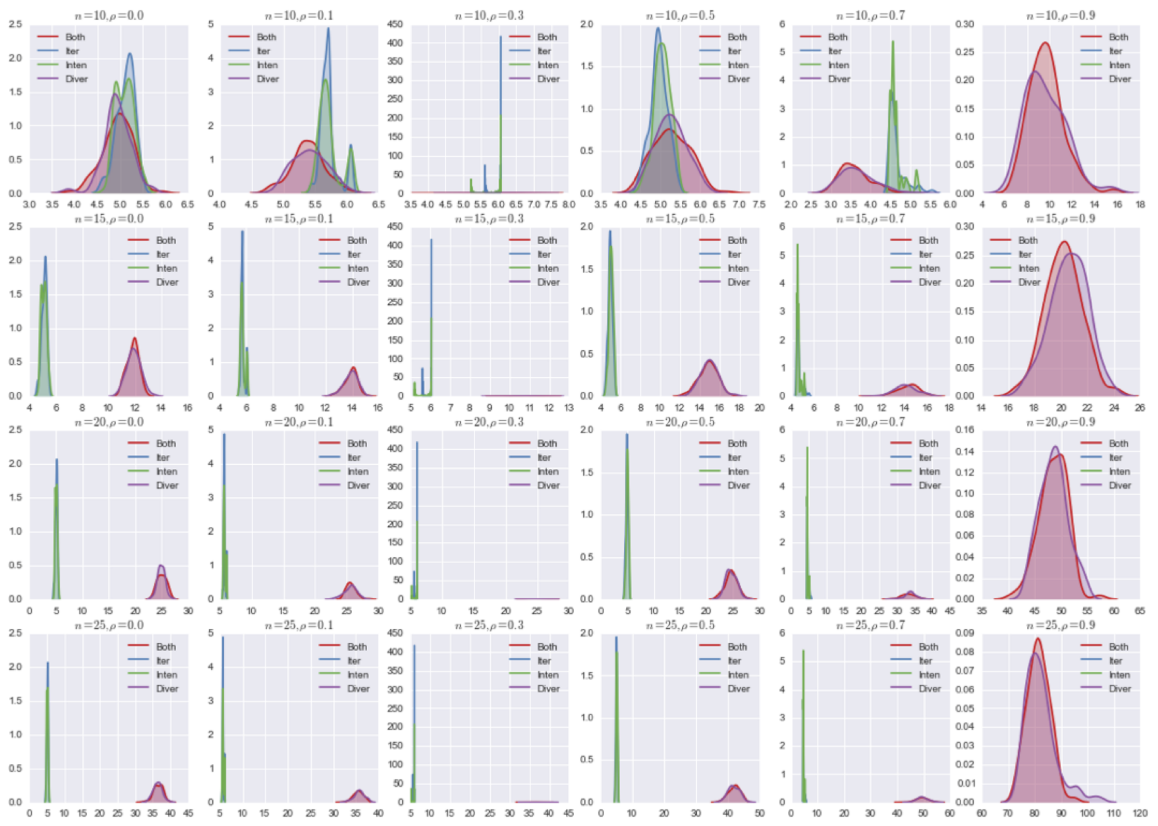


Figure 6.5: Results Regular Lattice Tests Varying ρ.

6.3.1 Speed

While the goal of this work was not to reduce the overall compute time of the algorithm, it is still essential to discuss the impact to overall speed. Local search requires greater than 90% of the total compute time for all tests run. Therefore, increases in the maximum number of times a given core will perform local search can significantly increase total compute time. This increase is linear. Communication incurred by the standardization of IFS and periodic locks during local search are extremely small, requiring less than 1% of total compute time. This may increase as the total problem size increase and the network becomes more saturated.

6.4 Discussion

Intuitively, spatial regionalization is a stochastically driven process. In a highly parallel environment, a frequent question is: why apply local search, the most computationally expensive process, when a higher number of IFS could be generated in the same amount of time? Using stochastically driven seed selection, region growth, and enclave assignment processes, I show significant improvement with the application of local search. To my knowledge, no study has been performed to quantify the component contribution of each aspect of IFS generation, and until such a study is completed, local search is an essential component of high-quality solution generation.

The first experiment, testing the maximum number of iterations around the solution space also shows that simply varying Tabu Search parameters is insufficient in locating an optimal solution. In instances where cooperative complexity would be too high, simple iteration across solutions with different TS search parameters does more frequently drive the algorithm to higher quality solutions.

Combined with iteration, intensification significantly improves the performance of a cooperative Tabu Search driven regionalization. In contrast, diversification alone is insufficient in permuting the solution space. I suggest that additional testing is required using varied methods of diversification as the operations research literature suggests that diversification is an essential component of successful algorithms.

The cumulative inclusion of iteration across the solution space, intensification at 40%, and diversification illustrate the potential gains achievable using a parallel, cooperative search strategy. The methods described, at the parameters tested, do not preclude the algorithm becoming trapped in a local optima, but 6.4Results for

Full Cooperative, Parallel Tabu Searchtable.6.4 shows that the likely of convergence to a (near) optimal solution is significantly increased. I suggest that diversification, as implemented, has a negative impact on the regionalization process. This may be because of the extent of diversification, i.e. shrinking the largest region to minimal size may be too drastic of a diversification, is too large, or that the diversification is rapidly reversed returning to a local optima.

The second set of tests, performed by varying levels of spatial autocorrelation illustrate the overall performance of the diversification operators in all but the smallest problem sizes. It is at these small problem sizes that insight into tuning the diversification operator may be found. For example, if the total size of each region is near the threshold, the diversification operator can make limited alteration to the current solution. Quantifying the diversification potential as a function of the solution quality distributions may allow for limited diversification, sufficient to escape local optima, but incapable of driving a solution so far from the global optima that reverie is not possible.

6.5 Conclusion

This work has focused on the application of a cooperative, multi-start parallel Tabu Search implementation to solve synthetically generated spatial regionalization problems. Solution quality has been the paramount goal. In the short term, future work will focus on testing inter vs. intra node intensification. That is, is it more efficient to perform bulk synchronization at some interval during local search across all nodes, or more efficient to keep intensification local to each individual node. Longer term work will focus on improvements to speed, without attention to solution quality, the quantification of the speed - quality relationship, and finally the development of a hybrid approach which allows the end user fine-grained control over where along the speed - quality boundary they wish to process. From here, it may be possible to identify convergence criteria to allow a highly communicative implementation to collectively terminate computation due to a non-improving solution. Additional work leveraging spatial regionalization will focus on an analysis of regionalization as a preprocessing step for other spatial analysis tasks, e.g. map classification. Through this work, I sought to apply the above methods and quantify potential performance gains, and the associated accuracy reductions, achievable through regionalized data size reduction.

Chapter 7

CONCLUSION

The continued exponential growth in depth and breadth of digital data capture with accompanying spatial data is unquestionable. Across all sciences, strategies for storing, processing, analyzing, and synthesizing these data sets continue to be active areas of research. The emergent Cyber Infrastructure paradigm, and derived Geospatial Cyber Infrastructure, will continue to be essential tools to facilitate scientific discovery leveraging these increased data sizes and increasingly complex process models. A view of the current landscape of parallel spatial analysis algorithms shows that algorithm development has been diverse in terms of implementation methodology and hardware environments. This diversity is healthy but lacks a common, unifying vocabulary from which future implementations can be driven. In implementing many parallel algorithms, the number of dead-ends, e.g. implementations techniques which fail, are significantly higher than the number of techniques which exceed expectations. For this reason the National Science Foundation (2012), issued a call for reusable, maintainable software frameworks to support common research tasks across diverse, inter-disciplinary research teams. This dissertation seeks to answer this call, not through the development of a software framework, but through the contribution of a theoretical classification mechanism, a taxonomy of vector spatial analysis algorithms. Leveraging this taxonomy, and the common vernacular it suggests, maintainable, extensible, interoperable software frameworks can be specified and developed.

The development of said taxonomy is the primary contribution of this dissertation. The taxonomy succeeds in providing high level representations that inform, but do not dictate, implementation. This is achieved by avoiding tight coupling to hardware and those classification criteria that can cause a devolution to a one-to-one mapping. I suggest that, from an implementation perspective domain decomposition, communication granularity, file I/O, and performance are all essential decisions. Inclusion of these criteria cause the taxonomy to dictate implementation as a function of hardware, data, and algorithm; this results in a cookbook style set of directives. By existing at a higher level and identifying generalities in communication patterns, drawn from an array of successful parallelization efforts, I suggest that the taxonomy avoids this pitfall. Finally, the taxonomy succeeds in being comprehensive across the spatial analysis stack, and in those instances where it may not be, is extremely extensible. This dissertation contributes a means to drive the development of parallel vector spatial analysis software frameworks.

Leveraging the classification scheme of Asanovic *et al.* (2006) I map the concept of computational dwarfs, or those core computational methods and the associated distributed communication requirements, to algorithms within the spatial analysis stack. I suggest that the existing Dense Linear Algebra, Sparse Linear Algebra, and MapReduce dwarfs are ideally suited for classifying some algorithms within the spatial analysis stack. Both Dense and Sparse Linear Algebra dwarfs leverage the fact that data vectors and matrices are composed of contiguously stored, homogeneous data. This representation drives point-to-point and local communication that is generally fine-grained. Vectorization plays a key role in providing processor-level parallelism during local computation. The MapReduce dwarf maps to a wide range of spatial analysis methods where no inter-core communication is required, i.e. embarrassingly parallel implementations. In addition to the three aforementioned classes, I propose Geometric, Topological, and Exploratory dwarfs. These dwarfs leverages the additional information inherent in spatial data; spatial is special. Geometric and

Topological dwarfs share significant commonalities, focusing on GIScience problems largely drawn from the computational geometry domain. I draw a key distinction in the quantity and frequency of communication, with the former generally utilizing point-to-point communication, e.g. parallel sorting, and the latter requiring the global communication of the data topology, e.g. a tree search.

Case studies both inform and are informed by the taxonomy. The development of the Geometric and Topological dwarfs are a prime example of the former. An analytical approach suggested that atomic computational operations focusing on geometric comparison were largely identical. Communication patterns would be data dependent, a function of the underlying density. Through implementation and simulation, it became clear that the previously identified thresholds, at which communication would shift from sort based methods to global methods, no longer held. The case studies directly informed the development and identification of the topological dwarf. In contrast, dwarfs drawn from Asanovic *et al.* (2006) rest upon a wealth of literature from the Computer Science domain. The Dense Linear Algebra and MapReduce case studies were directly informed by this previous work. The taxonomy and case studies are symbiotic.

I implement a three-phase, Fisher-Jenks optimal choropleth map classification algorithm with both the computation of dense distance and error matrices using vectorization and parallel computing paradigms. The addition of a lockless, shared memory space removes the previous need for in-memory data duplication and further improves overall scalability. This implementation is deployed to SMP computing environments and shows significant speedup over all other existing methods. The implementation is not without drawbacks, and still scales quadratically ($O(n^2)$) in memory requirements. Within the context of the taxonomy, this is a classic Dense Linear Algebra problem where a set of operands are applied to a decomposition of a dense matrix. This model is mappable to Sparse Linear Algebra problems with the addition of element position metadata, e.g. the location of non-zero entries.

I show that the classification of algorithms as either Geometric or Topological is contingent upon the underlying data structure as well as the quantity and timing of required communication. Point based Nearest Neighbor Search (NNS) and polygon adjacency algorithms are implemented using methods classifiable as both Geometric and Topological. Static grid based and adaptive decomposition methods are applied to spatial data sets with varying underlying densities. I show that resultant data structures map well to communication requirements in parallel environments; this requires the definition of two computational dwarfs. Regular gridded decomposition and global parallel sorting, a geometrically classifiable method, are shown to significantly outperform topological approaches in the case of polygon adjacency. In contrast, the same geometrically classifiable methods are significantly slower than the generation of a tree based, topological structure for NNS. I show that KD-Trees outperform global sorting methods across all problem sizes. The classification of methods is invariant to data clustering, suggesting that these classifications are robust.

Two spatial regionalization algorithms, p-Compact regions (PCR) and Max-p region (MPR), are implemented under the MapReduce and Exploratory dwarfs, respectively. I show that even though largely driven by stochastic processes, the probability that a PCR implementation yields high-quality results can be improved through a zero communication, MapReduce approach. This work serves as a benchmark for other parallel implementations as communication overhead is limited to initial data transmission and final data aggregation. The relationship between cooperative methods (moving the implementation to the exploratory classification), solution quality, and overall performance can be measured against this initial implementation. The PCR case study also highlights the potential to deploy closely related implementations to both SMP and HPC environments, thereby reducing the overall development time. Again, this is a trade-off between raw performance and more qualitative concerns, e.g. implementation time, code, readability and maintainability.

I present an MPR implementation that leverages Exploratory computational and communication techniques within the local search phase. Bulk synchronization is utilized to reduce the implementation complexity while still providing the necessary functionality to allow for a highly cooperative local search phase. I show that intensification, diversification, and controlled randomization in algorithm parametrization are essential drivers of the search process and consistently locate the highest quality solutions. Specifically, the process of intensification is shown to have the largest impact of overall solution quality across all tested data sets. It should be noted that this implementation highlights the composite nature of dwarfs, with the generation of the adjacency structure being classified as a geometric dwarf, Initial Feasible Solutions (IFS) being modeled as a MapReduce dwarf, and local search being realized as an exploratory dwarf.

It is within the MPR case study, that the power and scalability of this classification method becomes apparent. Low level classifications rapidly map at a one-to-one ratio to spatial analysis methods. The ability to chain elements in the classification becomes an exercise in individual algorithm parallelization. Invariant to low level implementation concerns, the presented taxonomy offers enough theoretical scaffolding to logically decompose an algorithm and suggest low-level implementation specifics without any stringent requirement.

Potential for future work within this domain is vast. Constrained to the context of the CyberGIS middleware layer a wide array of algorithms still require initial parallel implementations. I see the development of fully distributed sparse and dense linear algebra operations as an essential component to continued spatial algorithm scalability. This process is not composed of just the relatively trivial requirement that a matrix be generated across many memory address, but also the requirement that subsequent dwarfs leverage the distributed representations . Future work seeks to realize this in the case of the Fisher-Jenks algorithm where lookup within the distance matrix for error matrix computation is non-trivial. Within the context of Geometric and Topological algorithms additional work is clearly required to explore more efficient topological data structures, such that polygon adjacencies can be rapidly computed via a tree-based data structure in an HPC environment. This work is distinct from the current push to more efficiently leverage spatial databases and the accompanying indexing structures, as it seeks to focus on efficient techniques for commonly applied 'one-off' processing. Scalable spatial regionalization algorithms, classifiable as exploratory, remain relatively unstudied and significant improvement to the techniques of intensification and diversification across other heuristic solutions methods is required. Finally, the process of accessing and utilizing HPC resources, the primary computational environment for these methods, remains challenging. A future research goal target is the development of interfaces to support the utilization of these high-performance methods through an easy to use abstraction. In this way, the developed methods can be integrated more fully into the CyberGIS stack.

REFERENCES

Aggarwal, A., B. Chazelle, L. Guibas, C. O'Dunlaing and C. Yap, "Parallel computational geometry", Algorithmica **4**, 1-4, 293–327 (1988).

Akman, W., W. R. Franklin, C. Kankanhalli and C. Narayanaswami, "Geometric computing and uniform grid technique", Computer-Aided Design **21**, 7, 410–420 (1989).

Alted, F., "The starving cpu problem or why should I care about memory access", Presentation (2013).

Amdahl, G. M., "Validity of the single-processor approach to achieving large scale computing capabilities", in "AFIPS Conference Proceedings", pp. 483–485 (1967).

Andrienko, G., N. Andrienko, P. Bak, D. Keim, S. Kisilevich and S. Wrobel, "A conceptual framework and taxonomy of techniques for analyzing movement", Journal of Visual Languages and Computing **22**, 3, 213–232 (2011).

Anselin, L., *Spatial econometrics: Methods and models* (Matrinus Nijhoff, Dordrecht, the Netherlands, 1988).

Anselin, L., "The Moran scatterplot as an ESDA tool to assess local instability in spatial association", in "Spatial Analytical Perspectives on GIS", chap. 8, pp. 111–125 (Taylor and Francis, London, UK, 1996).

Anselin, L. and S. J. Rey, "Spatial econometrics in an age of CyberGIScience", International Journal of Geographical Information Science **26**, 12, 2211–2226 (2012).

Anselin, L., S. J. Rey and W. Li, "Metadata and provenance for spatial analysis: The case of spatial weights", International Journal of Geographical Information Science **28**, 11, 2261 – 2280 (2014).

Anselin, L. and O. Smirnov, "Efficient algorithms for constructing proper higher order spatial lag operators", Journal of Regional Science **36**, 1, 67 – 89 (1996).

Arjomandi, E., *A study of parallelism in graph theory*, Ph.D. thesis, University of Toronto, Department of Computer Science (1975).

Armstrong, M. and R. Marciano, "Massively parallel processing of spatial statistics", International Journal of Geographical Information Systems **9**, 2, 169–189 (1995).

Armstrong, M. and R. Marciano, "Local interpolation using a distributed parallel supercomputer", International Journal of Geographical Information Systems **10**, 6, 713–729 (1996).

Armstrong, M., C. Pavlik and R. Marciano, "Parallel processing of spatial statistics", Computers & Geosciences **20**, 2, 91–104 (1993).

Armstrong, M. P., “Geography and computational science”, Annals of the Association of American Geographers **90**, 1, 146–156 (2000).

Armstrong, M. P., M. K. Cowles and S. Wang, “Using a computational grid for geographic information analysis: A reconnaissance”, The Professional Geographer **57**, 2005, 365–375 (2005).

Armstrong, M. P. and P. J. Densham, “Domain decomposition for parallel processing of spatial problems”, Computers, Environment and Urban Systems **16**, 6, 497–513 (1992).

Asanovic, K., R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams and K. A. Yelick, “The landscape of parallel computing research: A view from berkley”, Tech. Rep. UCB/EECS-2006-183, Eletrical Engineering and Computer Sciences University of California at Berkeley (2006).

Atallah, M. and M. Goodrich, “Efficient parallel solutions to some geometric problems”, in “Proc. IEEE International Conference on Parallel Processing”, pp. 411–417 (1984).

Atallah, M. J., R. Cole and M. T. Goodrich, “Cascading divide-and-conquer: A technique for designing parallel algorithms”, SIAM Journal on Computing **18**, 3, 499–532 (1989).

Atallah, M. J. and M. T. Goodrich, “Efficient plane sweeping in parallel”, Tech. rep., Purdue University (1985).

Atkins, D., *Revolutionizing science and engineering through cyberinfrastructure: Report of the National Science Foundation blue-ribbon advisory panel on cyberinfrastructure* (National Science Foundation, 2003).

Barbini, P., G. Cevenini and M. R. Massai, “Nearest-neighbor analysis of spatial point patterns: application to biomedical image interpretation”, Computers and Biomedical Research **29**, 482 – 493 (1996).

Bell, G. C. and J. Gray, “The revolution yet to happen”, in “Beyond Calculation: The Next Fifty Years of Computing”, edited by P. J. Denning and R. Metcalf (Copernicus, 1997).

Blelloch, G. and J. Little, “Parallel solutions to geometric problems on the scan model of computation”, MIT Artificial Intelligence Laboratory Memo 952 (1988).

Brewer, C. A. and L. Pickle, “Evaluation of methods for classifying epidemiological data on choropleth maps in series”, Annals of the Association of American Geographers **92**, 4, 662–681 (2002).

Burgess, J. and A. Bruns, “Twitter archives and the challenges of ‘big social data’ for media and communication research”, M/C Journal **15**, 5 (2012).

Burrough, P. and R. McDonnell, *Principles of geographical information systems* (Oxford University Press, 1998).

Buzbee, B., “A strategy for vectorization”, Parallel Computing **3**, 187–192 (1986).

Chow, A. L., *Parallel Algorithms for Geometric Problems*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (1980).

Clark, J., Philip and F. C. Evans, “Distance to nearest neighbor as a measure of spatial relationships in populations”, Ecology **35**, 4, 445–453 (1954).

Clematis, A., B. Falcidieno and M. Spagnuolo, “Parallel processing on hetereogeneous networks for GIS applications”, International Journal of Geographical Information Systems **10**, 6, 747–767 (1996).

Clematis, A., M. Mineter and R. Marciano, “High performance computing with geographical data”, Parallel Computing **29**, 1275–1279 (2003).

Colella, P., “Defining software requirements for scientific computing”, Slide of 2004 presentation included in David Pattern’s 2005 talk (2004).

Cormen, T., C. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms* (The MIT Press, Cambridge, MA USA, 2001), 3 edn.

Cova, T. and R. L. Church, “Contiguity constraints for single-region site search problems.”, Geographical Analysis **32**, 4, 306 – 329 (2000).

Crainic, M., Teodor Gabriel anf Gendreau, “Cooperative parallel tabu search for capacitated network design”, Journal of Heuristics **8**, 601 – 627 (2002).

Crainic, T. G., M. Toulouse and M. Gendreau, “Towards a taxonomy of parallel tabu search heuristics”, INFORMS Journal on Computing **9**, 1, 61–72 (1997).

Cramer, B. and M. P. Armstrong, “An evaluation of domain decomposition strategies for parallel spatial interpolation of surfaces”, Geographical Analysis **31**, 2, 148–168 (1999).

Cressie, N. A. C., *Statistics for Spatial Data* (John Wiley & Sons, Ltd, 1990).

Dehne, F., A. Fabri and A. Rau-Chaplin, “Scalable parallel computational geometry for coarse grained multicomputers”, International Journal of Computational Geometry & Applications **6**, 3, 379–400 (1996).

Ding, Y. and P. J. Densham, “Spatial strategies for parallel spatial modelling”, International Journal of Geographical Information Systems **10**, 6, 669 (1996).

Dowers, S., B. M. Gittings and M. J. Mineter, “Towards a framework for high-performance geocomputation: handling vector-topology within a distributed service environment”, Computers, Environment and Urban Systems **24**, 471–486 (2000).

Duque, J. C., L. Anselin and S. J. Rey, “The Max-P-Regions problem”, Journal of Regional Science **52**, 3, 397–419 (2012).

Duque, J. C., R. L. Church and R. S. Middleton, "The p-Regions problem", Geographical Analysis **43**, 1, 104–126 (2011).

Duque, J. C., R. Ramos and J. Surinach, "Supervised regionalization methods: A survey", International Regional Science Review **30**, 195 – 220 (2007).

Eckstein, D., *Parallel Graph Processing Using Depth-First Search and Breadth-First Search*, Ph.D. thesis, University of Iowa, Department of Computer Sciencee (1977).

Egenhofer, M. J., J. Glasgow, O. Gunther, J. R. Herring and D. J. Peuquet, "Progress in computational methods for representing geographical concepts", International Journal of Geographical Information Science **13**, 8, 775–796 (2010).

Fischer, M. M., "Regional taxonomy: A comparison of some hierarchic and non-hierarchic strategies", Regional Science and Urban Economics **10**, 4, 503–537 (1980).

Flynn, M. J., "Some computer organizations and their effectiveness", IEEE Transactions on Computers **C-21**, 9, 948–960 (1972).

Forum, M. P., "MPI: A message passing interface", Tech. rep., University of Tennessee, Knoxville, TN, USA (1994).

Gabrielsson, S., *A Parallel Tabu Search Algorithm for the Quadratic Assignment Problem*, Master's thesis, University of Toronto (2007).

Gehlke, C. E. and K. Biehl, "Certain effects of grouping upon the size of the correlation coefficient in census tract material", Journal of the American Statistical Association **29**, 185A, 169 – 170 (1934).

Glover, F., "Tabu search—part I", ORSA Journal on computing **1**, 3, 190–206 (1989a).

Glover, F., "Tabu search—part II", ORSA Journal on Computing **2**, 1, 4–32 (1989b).

Gonzalez-Ramirez, R., N. R. Smith, R. G. Askin, P. Miranda and J. Sanchez, "A hybrid metaheuristic approach to optimize the districting design of a parcel company", Journal of Applied Research and Technology **9**, 1, 19–35 (2011).

Goodchild, M., "Geographical information science", International Journal of Geographical Information Systems **6**, 1, 31–45 (1992).

Goodchild, M., "Citizens as sensors: the world of volunteered geography", GeoJournal **69**, 4, 211–221 (2007).

Goodchild, M., "Twenty years of progress: GIScience in 2010", Journal of Spatial Information Science **1**, 1, 3–20 (2010).

Grama, A., A. Gupta, G. Karypis and V. Kumar, *Introduction to Parallel Computing* (Addison-Wesley Publishing Company, 2003).

Griffith, D., “Supercomputing and spatial statistics: a reconnaissance.”, The Professional Geographer **42**, 4, 481–492 (1990).

Gropp, W., “MPI at exascale: Challenges for data structures and algorithms”, in “Recent Advances in Parallel Virtual Machine and Message Passing Interface”, edited by M. Ropo, J. Westerholm and J. Dongarra, vol. 5759 of *Lecture Notes in Computer Science* (Springer Berlin Heidelberg, 2009).

Gustafson, J. L., “Reevaluating amdahl’s law”, Communications of the ACM **31**, 5, 532 – 533 (1988).

Gutman, A., “R-trees: A dynamic index structure for spatial searching”, in “Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data”, p. 47 (1984).

Harding, T., R. Healey, S. Hopkins and S. Dowers, “Vector polygon overlay”, in “Parallel Processing Algorithms for GIS”, edited by R. Healey, S. Dowers, B. Gittings and M. Mineter, chap. 13, pp. 265–288 (Taylor and Francis, Bristol, P.A., 1998).

Hartigan, J. A., “Partition by Exact Optimization”, in “Clustering Algorithms”, chap. 6, pp. 130 – 142 (Wiley, New York, New York, USA, 1975), 1 edn.

Hawick, K. A., P. Coddington and H. James, “Distributed frameworks and parallel algorithms for processing large-scale geographic data”, Parallel Computing **29**, 1297 – 1333 (2003).

Healey, R., S. Dowers, B. Gittings and M. Mineter, eds., *Parallel Processing Algorithms for GIS* (Taylor and Francis, Bristol, P.A., 1998).

Hill, M. D. and M. R. Marty, “Amdahl ’s Law in the multicore era”, Computer **41**, 7, 33–38 (2008).

Hirschberg, D. S., “Parallel algorithms for the transitive closure and the connected component problems”, in “Proc. 8th ACM Symposium on Theory of Computing”, pp. 55–57 (1976).

Hoel, E. and H. Samet, “Data-parallel polygonization”, Parallel Computing **29**, 1381–1401 (2003).

Intel, “Intel Hyper-Threading technology”, Tech. Rep. January, Intel Corporation (2003).

Jain, A., M. Murty and P. Flynn, “Data clustering: A review”, ACM Computing Surveys **31**, 3, 264–323 (1999).

James, T., C. Rego and F. Glover, “A cooperative parallel tabu search algorithm for the quadratic assignment problem”, European Journal of Operational Research **195**, 3, 810–826 (2009a).

James, T., C. Rego and F. Glover, “Multistart tabu search and diversification strategies for the quadratic assignment problem”, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **39**, 3, 579–596 (2009b).

Kaltofen, E. L., “The ”Seven Dwarfs” of symbolic computation”, Tech. rep., Department of Mathematics, North Carolina State University (2014).

Kelbert, A., “Science and Cyberinfrastructure: The chicken and egg problem”, Eos, Transactions American Geophysical Union **95**, 458–459 (2014).

Kim, H., Y. Chun and K. Kim, “Delimitation of functional regions using a p-regions problem approach”, International Regional Science Review **38**, 2, 235–263 (2015).

Langtangen, H. P. and X. Cai, “On the efficiency of python for high-performance computing: A case study involving stencil updates for partial differential equations”, in “Modeling, Simulation and Optimization of Complex Processes: Proceedings of the third International Conference on High Performance Scientific Computing”, edited by H. G. Bock, E. Kostina, X. P. Hoang and R. Rannache, pp. 337–357 (Springer, 2008).

Laura, J., W. Li, S. J. Rey and L. Anselin, “Parallelization of a regionalization heuristic in distributed computing platforms – a case study of parallel-p-compact-regions problem”, International Journal of Geographical Information Science **29**, 4, 536–555 (2015).

Laura, J. and S. J. Rey, “Improved parallel optimal choropleth map classification”, in “Modern Accelerator Technologies for Geographic Information Science”, edited by X. e. a. Shi, pp. 197–212 (Springer Science+Business Media, New York, 2014).

LeSage, J. P., “What regional scientists need to know about spatial econometrics”, The Review of Regional Studies **44**, 1, 13–32 (2014).

Leskovec, J., A. Rajaraman and D. Ullman, Jeffrey, *Mining of Massive Datasets* (Cambridge University Press, 2014).

Li, W., R. L. Church and M. F. Goodchild, “An extendable heuristic framework to solve the p-compact-regions problem for urban economic modeling”, Computers, Environment and Urban Systems **43**, 1–13 (2014a).

Li, W., R. L. Church and M. F. Goodchild, “The p-compact-regions problem”, Geographical Analysis **46**, 3, 250–273 (2014b).

Li, X., W. Li, L. Anselin, S. Rey and J. Koschinsky, “A MapReduce algorithm to create contiguity weights for spatial analysis of big data”, in “Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data”, BigSpatial ’14, pp. 50–53 (ACM, New York, NY, USA, 2014c), URL `http://doi.acm.org/10.1145/2676536.2676543`.

Liu, Y., K. Wu, S. Wang, Y. Zhao and Q. Huang, “A MapReduce approach to Gi*(D) spatial statistic”, in “Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems”, HPDGIS ’10, pp. 11–18 (ACM, New York, NY, USA, 2010), URL `http://doi.acm.org/10.1145/1869692.1869695`.

Liu, Y. Y. and S. Wang, “A scalable parallel genetic algorithm for the Generalized Assignment Problem”, Parallel Computing **46**, 98–119 (2015).

Malkov, Y., A. Ponomarenko, A. Logvinov and V. Krylov, “Approximate nearest neighbor algorithm based on navigable small world graphs”, Information Systems **45**, 61 – 68 (2014).

Maneewongvatana, S. and D. M. Mount, “It’s okay to be skinny, if your friends are fat”, in “4th Annual CGC Workshop on Computational Geometry”, (1999).

Manovich, L., “Trending: the promises and the challenges of big social data”, in “Debates in the Digital Humanities”, edited by M. K. Gold (University of Minnesota Press, 2012).

Marimont, R. B. and M. B. Shapiro, “Nearest neighbor searches and the curse of dimensionality”, IMA Journal of Applied Mathematics (1979).

McAllister, M., *The Computational Geometry of Hydrology Data in Geographic Information Systems*, Ph.D. thesis, The University of British Columbia (1999).

McKenney, P. E., ed., *Is parallel programming hard, and, if so, what can you do about it?* (IBM, IBM Beaverton, 2013), january 31, 2015 edn.

Miller, H. and S. Shaw, *Geographic Information Systems for Transportation: Principles and Applications* (Oxford University Press, 2001).

Moore, G. E., “Cramming more components into integrated circuits”, Electronics **38**, 8, 114–117 (1965).

Morril, R., “Redistricting revisited”, Annals of the Association of American Geographers **66**, 548 – 566 (1976).

Nagy, G. and S. Wagle, “Computational geometry and geography”, The Professional Geographer **32**, 3, 343–354 (1980).

National Science Foundation, “A vision and strategy for software for science, engineering, and education: Cyberinfrastructure framework for the 21st century”, Tech. rep., National Science Foundation (2012).

Oliphant, T., *Guide to NumPy*, Provo, UT, URL `http://www.tramy.us/` (2006).

Onbasoglu, E. and L. Ozdamar, “Parallel simulated annealing algorithms in global optimization”, Journal of Global Optimization **19**, 27-50 (2001).

OpenMP, “OpenMP: Application program interface version 4.0”, Tech. rep., OpenMP Architecture Review Board (2013).

Openshaw, S., “A geographical solution to scale, and aggregation problems in region-building, partitioning, and spatial modeling”, Transactions of the Institute of British Geographers pp. 169 – 184 (1977).

O'Sullivan, D. and D. J. Unwin, *Geographic Information Analysis* (Wiley, Hoboken, New Jersey, 2010a).

O'Sullivan, D. and D. J. Unwin, "The pitfalls and potential of spatial data", in "Geographic Information Analysis", chap. 2 (John Wiley & Sons, Ltd, 2010b).

Padmanabhan, A., S. Wang and J.-P. Navarro, "A CyberGIS gateway approach to interoperable access to the National Science Foundation TeraGrid and the Open Science Grid", in "Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery", No. 42 in TG '11, pp. 42:1–42:8 (ACM, 42:1–42:8, 2011).

Pang, S., H. He, Y. Li, T. Zhou and K. Xing, *An Approach to Redistricting based on Simple and Compactness*, vol. 6145 of *Lecture Notes in Computer Science*, pp. 415 – 424 (Springer Berlin Heidelberg, 2010).

Pathman, D., T. Ricketts III and T. Konrad, "How adults' access to outpatient physician services relates to the local supply of primary care physicians in the rural southeast", Health Research and Educational Trust (2006).

Pervasive Technology Institute, "Cyberinfrastructure: Pervasive technology institute: Uits research technologies division", URL `http://internal.pti.iu.edu/ci/iu-cyberinfrastructure-news-march-2007` (2007).

Pham, D. and D. Karaboga, *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks* (Springer, London, UK, 2000).

Puri, S. and S. K. Prasad, "Output-sensitive parallel algorithms for polygon clipping", in "28th IEEE International Parallel and Distirbuted Processing Symposium", (2014).

Ram, D., T. Sreenivas and K. Subramaniam, "Parallel simulated annealing algorithms", Journal of Parallel and Distributed Computing **37**, 207–212 (1996).

Rey, S., L. Anselin, R. Pahle, X. Kang and P. Stephens, "Parallel optimal choropleth map classification in PySAL", International Journal of Geographical Information Science **27**, 5, 1023–1039 (2013).

Rey, S. J. and L. Anselin, "PySAL: A Python library of spatial analytical methods.", in "Handbook of Applied Spatial Analysis", edited by A. Fischer, M.M ; Getis, pp. 175–193 (Springer, 2010).

Samet, H., "The quadtree and related hierarchical data structures", Computing Surveys **16**, 2, 187 – 260 (1984).

Savage, C. D., *Parallel Algorithms for Graph Theoretic Problems*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science (1978).

Shirabe, T., "A model of contiguity for spatial unit allocation", Geographical Analysis **37**, 1, 2–16 (2005).

Shirabe, T., "Districting modeling with exact contiguity constraints", Environment and Planning B: Planning and Design **36**, 6, 1053–1066 (2009).

Shook, E., S. Wang and W. Tang, "A communication-aware framework for parallel spatially explicit agent-based models", International Journal of Geographical Information Science (2013).

Slocum, T., R. McMaster, F. Kessler and H. Howard, *Thematic cartography and geovisualization.* (Prentice Hall., 2008).

Stewart, C. A., S. Simms, B. Plale, M. Link, D. Y. Hancock and G. C. Fox, "What is Cyberinfrastructure", in "Proceedings of the 38th Annual ACM SIGUCCS Fall Conference: Navigation and Discovery", SIGUCCS '10, pp. 37–44 (2010).

Stojmenovic, I. and D. J. Evans, "Comments on two parallel algorithms for the planar convex-hull problem", Parallel Computing **5**, 3, 373–375 (1987).

Stratton, J. A., C. Rodrigues, I.-J. Sung, L.-W. Chang, N. Anssari, G. Liu, W.-m. W. Hwu and N. Obeid, "Algorithm and data optimization techniques for scaling to massively threaded systems", IEEE Computer **45**, 8, 26 – 32 (2012).

Sui, D. and M. F. Goodchild, "The convergence of GIS and social media: challenges for GIScience", International Journal of Geographical Information Science **25**, 11, 1737–1748 (2011).

Taillard, E., "Robust taboo search for the quadratic assignment problem", Parallel Computing **17**, 4-5, 443–455 (1991).

Tang, W., "Parallel construction of large circular cartograms using graphics processing units", International Journal of Geographical Information Science **27**, 11, 2182–2206 (2013).

Tang, W., D. Bennett and S. Wang, "A parallel agent-based model of land use opinions", Journal of Land Use Science **6**, 121 – 135 (2011).

Trienekens, H. and A. Bruin, "Towards a taxonomy of parallel branch and bound algorithms", Report eur-cs-92-01, Erasmus University, Rotterdam (1992).

Valiant, L. G., "A bridging model for parallel computation", Communications of the ACM **33**, 8, 103–111 (1990).

van Rossum, G. and F. Drake, "Python Reference Manual", (2013).

von Neumann, J., "First draft of a report on the EDVAC", Tech. rep., Unites States Army Ordnance Department and University of Pennsylvania (1945).

Wang, S., "A cybergis framework for the synthesis of Cyberinfrastructure, GIS, and spatial analysis", Annals of the Association of American Geographers **100**, 3, 535–557 (2010).

Wang, S., “CyberGIS: Blueprint for integrated and scalable geospatial software ecosystems”, International Journal of Geographical Information Science **27**, 11, 2119–2121 (2013).

Wang, S., L. Anselin, B. Bhaduri, C. Crosby, M. Goodchild, Y. Liu and T. Nyerges, “CyberGIS software: A synthetic review and integration roadmap”, International Journal of Geographical Information Science **27**, 11, 2122–2145 (2013).

Wang, S. and M. P. Armstrong, “A quadtree approach to domain decomposition for spatial interpolation in Grid computing environments”, Parallel Computing **29**, 10, 1481–1504 (2003).

Wang, S. and M. P. Armstrong, “A theory of the spatial computational domain”, in “Proceedings of the 8th International Conference on GeoComputation”, (2005).

Wang, S., M. P. Armstrong, J. Ni and Y. Liu, “Gisolve: A grid-based problem solving environment for computationally intensive analysis”, in “Challenges of Large Applications in Distributed Environments. CLADE 2005. Proceedings.”, (2005).

Wang, S., M. K. Cowles and M. P. Armstrong, “Grid computing of spatial statistics: using the teragrid for gi*(d) analysis”, Concurrency and COmputation: Practice and Experience **20**, 1697–1720 (2008).

Ward, M. D. and K. S. Gleditsch, “An introductiuon to spatial regression models in the social sciences”, Https://web.duke.edu/methods/pdfs/SRMbook.pdf (2007).

Waugh, T. C., “A response to recent papers and articles on the use of quadtrees for geographic information systems”, in “Proceesings of the Second International Symposium on Spatial Data Handling”, vol. 2, pp. 33–37, International Geographical Uninion. Commission on Geographical Data Sensing and Processing and the International Cartographic Association (International Geographical Union, COmmision on Geographical Data Sesngin and Processing, 1986).

Waugh, T. C. and S. Hopkins, “An algorithm for polygon overlay using cooperative parallel processing”, International Journal of Geographical Information Systems **6**, 6, 457–467 (1992).

Widener, M. J., N. C. Crago and J. Aldstadt, “Developing a parallel computational implementation of amoeba”, International Journal of Geographical Information Science **26**, 9, 1707 – 1723 (2012).

Wise, S., R. Haining and J. Ma, *Regionalisation tools for exploratory spatial analysis of health data*, pp. 83–100 (Springer, 1997).

Wise, S., R. P. Haining and J. Ma, “Providing spatial statistical data analysis functionality for the GIS user: the SAGE project”, International Journal of Geographical Information Science **15**, 3, 239 – 254 (2001).

Wright, D. J. and S. Wang, “The emergence of spatial cyberinfrastructure”, PNAS **108**, 14, 5488–5491 (2011).

Yang, C., W. Li, J. Xie and B. Zhou, “Distributed geospatial information processing: sharing distributed geospatial resources to support Digital Earth”, International Journal of Digital Earth **1**, 3, 259–278 (2008).

Yang, C. and R. Raskin, “Introduction to distributed geographic information processing research”, International Journal of Geographical Information Science **23**, 5, 553–560 (2009).

Yang, C., R. Raskin, M. Goodchild and M. Gahegan, “Geospatial Cyberinfrastructure: Past, present and future”, Computers, Environment and Urban Systems **34**, 4, 264–277 (2010).