NUMERICAL ANALYSIS OF A PARALLEL DOMAIN DECOMPOSITION METHOD FOR LINEAR ACOUSTIC PROBLEMS ON A CAVEMAN NETWORK

by

Maria Streater

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE

Major Subject: MATHEMATICS

West Texas A&M University

Canyon, Texas

August, 2017

Approved:

[Co-Chairman, Thesis Committee]	[Date]
[Co-Chairman, Thesis Committee]	[Date]
[Member, Thesis Committee]	[Date]
[Member, Thesis Committee]	[Date]
[Head, Major Department]	[Date]
[Dean, Major School]	[Date]
[Dean, Graduate School]	[Date]

ABSTRACT

In this work, transport on a network and the advantages of using a parallel architecture for computations for three distinct network configurations are explored. The aim is to exploit mathematical methods coupled with modern computing techniques to arrive at an efficient and effective solution. Results obtained on serial and parallel architectures are presented and compared on the configurations. The three dimensional transport problem is modeled as a two dimensional network problem to reduce dimensionality, a common practice when dealing with complicated systems. This system is represented on a given domain as a graph. The domain decomposition method is implemented for the linear acoustic problem on three network configurations. The convergence of this method is examined in order to determine an *a priori* estimate for the total number of iterations to convergence. The numerical solution of the multidimensional problem is justified by algebraic conditions that model the multi-dimensional effects at the network junctions. The results demonstrate an approximate bound on the number of iterations for a given network and provide insight into the advantages of parallel processing on networks of a certain size.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computational power, which has increased significantly, still has limits. Mathematical models of natural processes involve significant complexity. Computation of these models are affected by limits on computational power and such models of these processes cannot always incorporate all dimensions. A common practice when trying to mitigate computational time is to employ dimensional reduction. This method provides a suitable model with a dimension dependency removed via assumptions about the solution. In this work, transport on a class of networks is modeled. A network is a large multidimensional system that simplifies to transport on an edge via dimensional reduction. Systems can include such concepts as laser drilling or gas flow through a nozzle [4]. After dimensional reduction is applied, these systems are transformed into one-dimensional domains and are represented by edges on a network. This network appears as a graph. This is now a problem that can be solved in reasonable time and with less computational demand. In addition, application of the domain decomposition method and computations of the simulation completed on a parallel architecture to further decrease run time and increase output are studied. The do-

main decomposition method is used for a linear transport type problem defined on a specific type of network with three configurations. The premise of parallel computing is to simultaneously use multiple processors for computation where available. Most modern computers employ this tactic by using dual or quad core central processing units (CPUs) [26] and use optimization software to handle load balancing among the processors.

In Chapter 2, preliminary concepts regarding laws of conservation, an introduction the numerical methods used in this work, and some basic concepts of parallel computing are presented. Chapter 3 details network problems and the application of to the network problem. Chapter 4 details the convergence analysis. Numerical methods and results are further explored in Chapter 5. A summary of the results from this work, conclusions and avenues for future work are presented in Chapter 6

CHAPTER II

PRELIMINARIES

Hyperbolic partial differential equations (PDEs) appear in many problems. PDEs where wave motion or advective transport is important are the focus of the work herein. These PDEs arise in a variety of fields such as gas dynamics, acoustics, optics, and geophysics [11, 29, 18]. This includes systems of PDEs in a wide range of fields that are not limited to those in a purely scientific arena. In other fields such systems are used to explain stocks, air traffic, water flows, and feedback [22]. All of these systems obey the laws of conservation. In this chapter, a brief introduction of the concepts used throughout this work is given. Preliminary and fundamental concepts for advective transport on a network and the parallel computing aspect studied in this work are introduced. The general theory of conservation laws, advection, and characteristics are detailed in section 2.1. Section 2.2 and numerical methods used in this work are discussed. The basics of parallel computing are introduced in section 2.3; for specifics on the cluster structure and hardware, see Appendix A and B.

2.1 Conservation Laws

Fundamental concepts for this work begin with the conservation laws, which are an important class of homogeneous hyperbolic equations [18]. Quantities such as energy and momentum are conserved, meaning that the given substance or quantity is neither created nor destroyed. Thus models of physical laws give rise to the conservation laws. These laws are commonly derived in an integral form that is then converted to the differential form. Their derivation begins with physical principles that lead to development of the conservation laws.

Consider, as in [18], some conserved quantity, q, flowing through a one-dimensional pipe with some known velocity α, which is assumed to vary only with x, the distance along the pipe, and time t . Then the total mass of the conserved quantity in any given section $[x_1, x_2]$ is represented by

$$\int_{x_1}^{x_2} q(x,t)dx. \tag{2.1}$$

Since the substance is conserved within any given section the total mass within $[x_1, x_2]$ can change only with respect to the flux of particles through the endpoints x_1 and x_2 of the section which gives

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = f_1(t) - f_2(t), \tag{2.2}$$

where the flux $+f_1(t)$ and $-f_2(t)$ represent the flux into the section. Equation (2.2) is the integral form of the conservation law.

Now consider how the flux functions are related to q. Flux is given by

$$f(x,t) = \alpha q(x,t), \tag{2.3}$$

where α represents a constant velocity. This transforms (2.2) into

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = f(q(x_1,t)) - f(q(x_2,t)), \tag{2.4}$$

which says that the flux at any point and time can be determined from the value of the conserved quantity at that point and does not depend on the location of the point. This type of equation is called autonomous [18]. Now further suppose that f and q are smooth then with some modifications (2.4) becomes

$$\int_{x_1}^{x_2} \left[\frac{\partial}{dt}q(x,t) + \frac{\partial}{\partial x}f(q(x,t))\right] dx = 0, \tag{2.5}$$

which implies that the integrand is identically zero. Since x_1 and x_2 are arbitrary this implies

$$\frac{\partial}{dt}q(x,t) + \frac{\partial}{\partial x}f(q(x,t)) = 0. \tag{2.6}$$

As in [18], a substitution of (2.3) transforms the above equation into

$$\frac{\partial}{dt}q(x,t) + \alpha\frac{\partial}{\partial x}q(x,t) = 0, \tag{2.7}$$

which is is the differential form for the conservation laws. For another version of the derivation of the conservations laws see [4, 20], where flux is represented as a vector and the Divergence Theorem is employed. The discontinuous solution is explored in[4, 18, 6, 29] among many others, but this work only considers a solution that is continuous.

2.2 Advection Equation

The flux given by $f(q) = \alpha q$, where α is a constant, becomes the advection equation. The scalar advection equation is

$$q_t + \alpha q_x = 0, \tag{2.8}$$

and models advection or translation of a substance or data at a constant velocity. It is a scalar, linear, constant-coefficient hyperbolic partial differential equation. Although not used in the model for this work, the Cauchy problem, which is the simplest case of (2.8), provides insight into the type of solution that should be expected. For a unique solution, an initial condition is required along with boundary conditions. The expectation is that the initial condition to be advected across the domain at speed α,

so given some initial condition

$$q(x, 0) = \eta(x), \tag{2.9}$$

the exact solution is of the form

$$q(x, t) = \eta(x - \alpha t). \tag{2.10}$$

It is easy to verify that this holds for any smooth function and satisfies (2.8).

2.2.1 Characteristics

The consideration of conservation laws here, is limited to the one dimensional case. To discuss the properties of characteristic curves. The solution of $q(x, t)$ is constant along these characteristic curves in the x-t plane, that is $q(x, t)$ is constant for any $x - \alpha t$ that is constant in the x-t plane. From [18], this means that the characteristic curves for a PDE are curves along which $q(x, t)$ simplifies in some manner. Physically, the expectation is that the values of q simply advect with constant velocity α, and indeed this is what occurs. The flux function given by (2.3) is a linear function and the characteristic curves are particularly useful here. Since $f(q)$ is a linear function, $f'(q)$ will be constant, hence all characteristics are lines which all have the same slope in the the x-t plane. These characteristics can be used to solve (2.7) in conjunction with initial condition (2.9). Again, the exact solution is given by (2.10) and the conclusion is that the initial profile travels with constant speed α. In the case where

$\mathbf{q}(x,t) : \mathbb{R}$ X $\mathbb{R}^+ \to \mathbb{R}^n$ is a vector valued function, the conservation law is given by

$$\frac{\partial}{\partial t}\mathbf{q} + \mathbf{A}\frac{\partial}{\partial x}\mathbf{q} = 0,\ \mathbf{A} \in \mathbb{R}^{n \text{ x } n}. \tag{2.11}$$

This linear system can be solved in a manner much like that of the scalar case, but in order to handle this coupled conservation law system convert from the state variables $\mathbf{q}$ to characteristic variables, w and z, as in [18].

Characteristic Variables

As in [12] consider the first order linear system in (2.11) where $\mathbf{q}$ is a n-vector and $\mathbf{A}$ is an n x n constant matrix. First assume that the system is strictly hyperbolic.

Definition 2 .1. A system of hyperbolic partial differential equations given in the form

$$\frac{\partial}{\partial t}\mathbf{q} + \mathbf{A}\frac{\partial}{\partial x}\mathbf{q} = 0, \tag{2.12}$$

is called strictly hyperbolic if all of the eigenvalues of $f'(\mathbf{q}) = \mathbf{A}$ are real and distinct [30].

This work only considers the linear case, hence $f'(\mathbf{q}) = \mathbf{A}$, which implies that $\mathbf{A}$ is diagonalizable with real eigenvalues. Using the methods presented in [18, 4] decompose $\mathbf{A}$ in the following manner,

$$\mathbf{A} = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1}, \tag{2.13}$$

where

$$\mathbf{\Lambda} = diag(\lambda_i, \lambda_2, \ldots, \lambda_n), \tag{2.14}$$

$$\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n], \tag{2.15}$$

$$\mathbf{R}^{-1} = [\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n]^T, \tag{2.16}$$

where λ_i represent the eigen values of $\mathbf{A}$ and $\mathbf{l}$ and $\mathbf{r}$ represent the left and right eigen vectors of A. Multiply (2.11) by $\mathbf{R}^{-1}$ and obtain

$$\frac{\partial}{\partial t}\mathbf{w} + \mathbf{\Lambda}\frac{\partial}{\partial x}\mathbf{w} = 0, \tag{2.17}$$

where

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{q}, \tag{2.18}$$

which are the characteristic variables. Thus the linear system decouples into a system of advection equations which have speed of sound λ_i. Thus solve these advection equations as previously discussed and then convert back to the state variables $\mathbf{q}$. The idea is that of superposition [11], which simplifies complex problems by breaking them into smaller subproblems which may be solved one at a time. The results are combined together to obtain the solution of the system. The solution is a superposition of waves

and characteristic variables given by

$$\mathbf{q}(x,t) = \sum_{p=1}^{n} w^p(x,t) r^p. \tag{2.19}$$

So $\mathbf{q}(x,t)$ is a linear combination of the right eigenvectors at each point of the x-t plane. Requiring the system (2.11) to be hyperbolic ensures that the eigenvectors of $mathbf{A}$ n vectors are linearly independent and that every $q \in \mathbf{q}$ is unique. These curves are the characteristics, which as previously stated, are straight lines in the case of a constant-coefficient system. Note that by letting $\mathbf{L} = \mathbf{R}^{-1}$ be the left eigenvectors of $\mathbf{A}$,

characteristics $\mathbf{w}^p(x,t)$ can be written as

$$\mathbf{w}^p(x,t) = \mathbf{l}^p \mathbf{q}(x,t). \tag{2.20}$$

Incorporating the initial data (2.9), the solution (2.19) may be written as

$$\mathbf{q}(x,t) = \sum_{p=1}^{n} [\mathbf{l}^p \eta\left(x - \lambda^p t\right)]] \mathbf{r}^p. \tag{2.21}$$

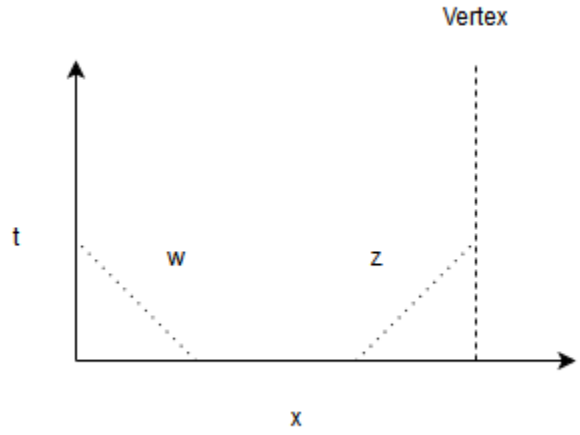

Figure 2.1: Characteristic lines

Note that this work only considers the case where the solution to the conservation laws are smooth. For more information regarding considerations for a weak solution see [20, 4] and the references therein.

2.2.2 Boundary Conditions

Consider a bounded domain, then (2.2) indicates how to set the boundary conditions for the one-dimensional conservation laws. Due to the direction in which the characteristic lines propagate, one of the two characteristic variables can be determined from initial conditions. The other characteristic is set by boundary conditions. Lions in [20] indicates that the number of boundary conditions needed depends on the structure of the characteristics at that boundary. This means that the number of boundary conditions needed depends on the direction of the characteristic lines at the junction. The lines that can be traced back into the domain, similar to an up-

wind method [19], will be determined by the initial condition. Determination of the characteristics directed away from the boundary requires boundary conditions. Since the problem is linear, the characteristics are determined by the eigenvalues of $\mathbf{A}$ and are constant.

2.3 Numerical Methods

This work requires a numerical discretization scheme in order to attain results. The numerical scheme that will be implemented throughout this work is the Lax-Friedrichs method from [19].

2.3.1 Lax-Friedrichs

Recall that, the solution to the advection equation (2.8) is given by (2.10). There are issues that can arise in the discretization of a hyperbolic equation, such as stability and accuracy. These are particularly visible with the advection equation. Define q as the exact solution of the PDE and the exact solution Q of q as the exact solution of the discrete problem, that is,

$$Q_j^{n+1} \equiv q\left(j\Delta x, n\Delta t\right) + \mathcal{O}\left(\Delta x^2\right) + \mathcal{O}\left(\Delta t\right) \tag{2.22}$$

With given quantities Δx, Δt, and constant speed α along with the approach of LeVeque [19], consider the discretization method using a entered difference in space and the forward difference in time given by,

$$q_x(x,t) = \frac{q(x+\Delta x,t)-q(x-\Delta x,t)}{2\Delta x} + \mathcal{O}\left(\Delta x^2\right), \tag{2.23}$$

$$q_t(x,t) = \frac{q(x,t+\Delta t)-q(x,t-\Delta t)}{\Delta t} + \mathcal{O}\left(\Delta t\right). \tag{2.24}$$

These give the method,

$$\frac{Q_j^{n+1}-Q_j^n}{\Delta t} = -\frac{\alpha}{2\Delta x}\left(Q_{j+1}^n - Q_{j-1}^n\right), \tag{2.25}$$

which may be rewritten as

$$Q_j^{n+1} = Q_j^n - \frac{a\Delta t}{2\Delta x}\left(Q_{j+1}^n - Q_{j-1}^n\right). \tag{2.26}$$

This method gives the stencil in Figure (2.2), but this method has issues with stability and so is not useful. Replacing Q_j^n on the right hand side of (2.26) by the average $\frac{1}{2}\left(Q_{j-1}^n - Q_{j+1}^n\right)$ gives the Lax-Friedrichs method,

$$Q_j^{n+1} = \frac{1}{2}\left(Q_{j-1}^n - Q_{j+1}^n\right) - \frac{\alpha\Delta t}{2\Delta x}\left(Q_{j+1}^n - Q_{j-1}^n\right), \tag{2.27}$$

which is a much more useful method as it is stable.

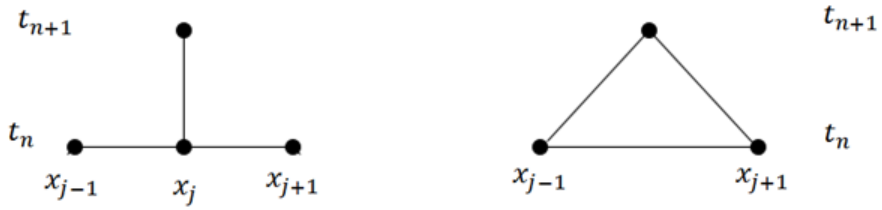

Figure 2.2: Stencils for methods (2.25) and (2.26)

Although this lower order method, it is Lax-Richtmyer stable provided

$$\left|\frac{a\Delta t}{\Delta x}\right| \leq 1. \tag{2.28}$$

It is important to note that the LaxFriedrichs method introduces significantly more diffusion than is required. As such it gives numerical results that are commonly badly smeared unless a highly refined grid is used [18]. The condition set forth in(2.28) is the Courant-Friedrichs-Lewy Condition (CFL).

Definition 2 .2. CFL Condition: A numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the PDE, at least in the limit as Δt and Δx both approach zero [18].

The CFL condition is a necessary condition that must be satisfied by a finite volume or finite difference method in order to expect it to be stable and thus converge to the solution of the differential equation as the grid is refined. Physically, this means

that the method must be used in a way that information propagates at the physical speeds determined by the eigenvalues of $f'(q)$.

2.4 Parallel Computing

Traditional serial programming runs a problem or job on a single computer having a single central processing unit (CPU). The CPU breaks the problem or jobs into a discrete series of instructions which are executed sequentially. Only one instruction may execute at any moment in time with this type of computing.

From Figure 2.3, it is clear that in a serial approach, all processing happens one after the other, or sequentially. The next job is not triggered until the previous job has finished. Speed improvement on serial processes is limited by an inability to increase transfer speeds of operands between the data buffers and the functional units [21]. In the execution of a parallel program, tasks are simultaneous and work load is split up on multiple CPUs in order to achieve more rapid results. The idea is that the process of solving a problem can usually be divided into several smaller tasks, which may be carried out concurrently with some communication overhead. It is important to note that overall performance depends heavily on the overhead associated with reads and writes to the global memory. Subjects such as barrier synchronization and dynamic scheduling, while not central to this work, are central in the field of parallel computing. Refer to [13] and the references therein for in depth discussion of these and other concepts.

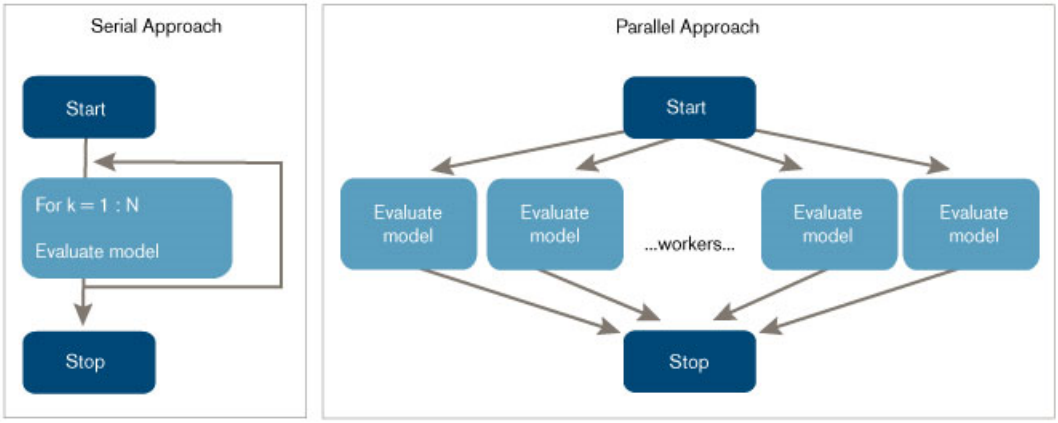

Figure 2.3: Serial versus Parallel Processing

Coordinating communication between the various processors can create bottlenecks and data integrity issues, which leads to slower execution or inaccuracies. Parallel programming is not simply an extension of serial programming; programs need to "think in parallel" [21]. Thus parallel algorithms are more complex and require more work and time from a programming standpoint. If one is not careful race conditions can arise within a simulation. This occurs when the sequence or timing of processes or threads depend on a shared state, which can create data corruption when events "collide" or happen in an unintended order. Deadlock is another possible issue. This happens when two or more competing actions are waiting for the other to finish, resulting in neither ever completing.

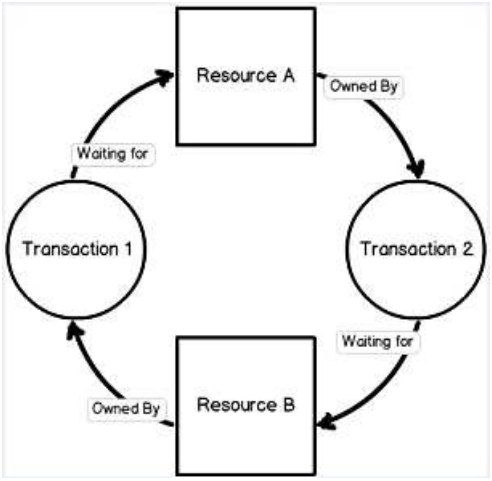

Figure 2.4: Deadlock [26]

Sequential processing is an sufficient method for processing data one step at a time and is ideal in situations when the CPU is performing a calculation that depends on the result of the previous calculation. Processing calculations, that cannot be parallelized will slow down a parallel program. There are data dependency issues with the parallelization of this type of simulation; however, this may be addressed with the domain decomposition method discussed in Chapter 3.

In theory using additional resources will shorten completion time, thereby creating potential computational cost savings. Since parallel clusters can be built from inexpensive commodity grade components, they are ideal for gaining speedup without sacrificing cost. A single computing resource can only complete one task at a time, while multiple computing resources can work on many tasks simultaneously. Parallel processing is much faster than sequential processing for repetitive calculations on substantial amounts of data [26]. A parallel processor is capable of multitasking on a large scale and can, therefore, simultaneously process several streams of data.

Compared with serial systems, parallel systems permit more freedom of expression in problem analysis and programming [21]. Such freedom expands the type and depth of problems that can approached and analyzed.

Flynn's taxonomy dating from 1966 does not adequately reflect current architectural designs it is nevertheless a solid guideline and remains useful today [21]. Flynn's taxonomy distinguishes multi-processor computer architectures by classification along the two independent dimensions of instruction stream and data stream. Each of these dimensions can have only one of two possible states: single or multiple.

Single Instruction, Single Data (SISD): A serial computer with the CPU acting on one instruction stream and using one data stream as input, both in any one clock cycle. The SISD systems have deterministic execution and are the oldest type of computers. Some examples are older generation mainframes, minicomputers, workstations and single processor/core PCs.

Single Instruction, Multiple Data (SIMD): A type of parallel computer where all processing units execute the same instruction at any given clock cycle. Each processing unit can operate on a different data element. This is best suited for specialized problems characterized by a high degree of regularity. This type has synchronous and deterministic execution with two varieties: processor arrays and vector pipelines.

Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution.

Multiple Instruction, Single Data (MISD): A type of parallel computer in which a single data stream is fed into multiple processing units. Each processing unit operates on the data independently by separate instruction streams. Few, if any, actual examples of this type of parallel computer exist or existed.

Multiple Instruction, Multiple Data (MIMD): A type of parallel computer in which every processor may be executing a different instruction stream and may be working with a different data stream. The execution can be synchronous or asynchronous, deterministic or non-deterministic. Currently, this is the most common type of parallel computer. Examples include most current supercomputers, networked parallel computer clusters and multi-core PCs. The cluster used for this study is MIMD.

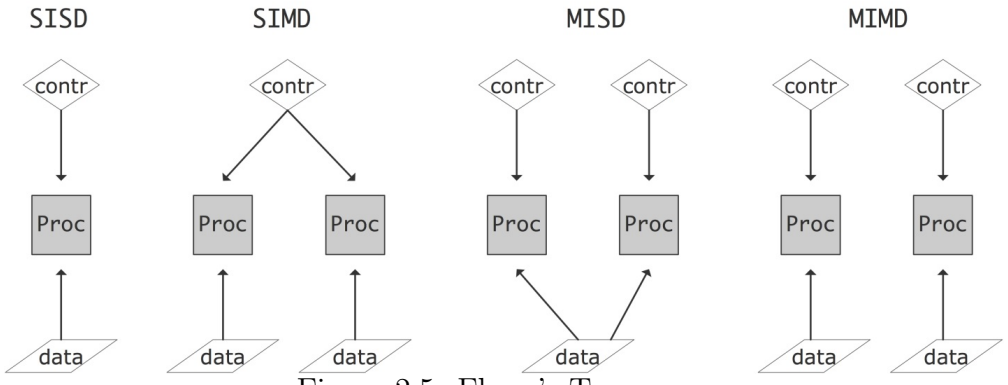

Figure 2.5: Flynn's Taxonomy

2.4.1 Limits

An obvious limit for small scale clustering, i.e. an at home scenario, can be limited by funding and the necessary space to set up and maintenance the hardware. For large scale clustering overall energy consumption to run and cool the CPUs present concerns about energy consumption and its' related costs.

A theoretical index, speedup, has been used for measuring the performance of parallel algorithms[21]. Amdahl's law, also known as Amdahl's argument, is named after computer architect Gene Amdahl. It is used to find the upper limit of the expected improvement to an overall system when the system is only partially improved. Often, it is used in parallel computing to predict theoretical maximum observed speedup using multiple processors. Amdahl's law states that the possible performance improvement to be gained from using a faster mode of execution is limited by the fraction of the time the faster mode can be used [21]. Every parallel algorithm has a sequential component that will eventually limit speedup. According to Amdahl's Law, potential speedup is defined by the fraction of code that can be parallelized.

$$speedup = \frac{1}{1-p} \ , \tag{2.29}$$

where no speedup corresponds to $p = 0$ and if all of the code is parallelized $p = 1$. Theoretically speedup is then infinite. With the introduction of the number of

processors performing the parallel fraction of work, the relationship can be modeled by

$$speedup_n = \frac{1}{1 - p + \frac{p}{n}} , \tag{2.30}$$

where n is the number of processors.

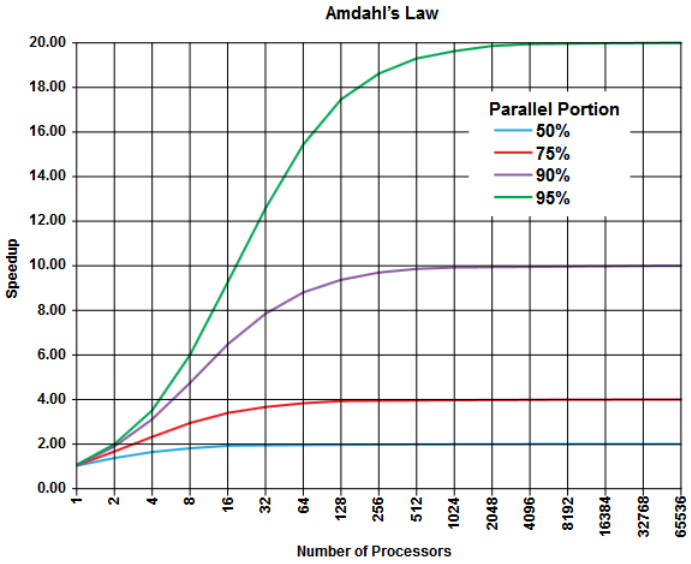

Figure 2.6: Amdahl's Law

Amdahl's law is an algorithm that uses the law of diminishing returns that determines the overall speedup of the program. The limit of $speedup_n$ as n approaches infinity will simply be *speedup*, that is as n tends to infinity, the p/n term will tend to zero. See [13] for a more rigorous definition of speed up and efficiency.

It is obvious that there are limits to the scalability of parallelism. For example:

n	$p = .50$	$p = .90$	$p = .99$
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1,000	1.99	9.91	90.99
10,000	1.99	9.91	99.02
100,000	1.99	9.99	99.9

Table 2.1: Speedup

It follows that problems that increase the percentage of parallel time with size are more scalable than problems with a fixed percentage of parallel time.

CHAPTER III

TRANSPORT ON NETWORKS

A set of items, vertices with edge connections linking them is a network, commonly termed a graph [28], see Figure (3.1). Networks can take widely varied forms and are studied in a variety of different fields such as biological networks, social networks, information networks, and technological networks. In fact the computer cluster used for the numerical results in this work is a network of stand alone computers linked together with an ethernet cable.

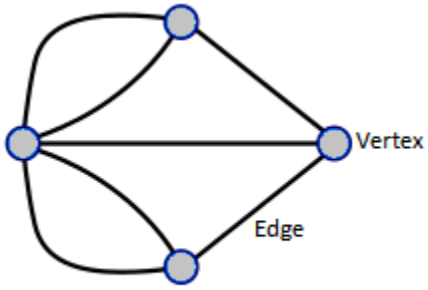

Figure 3.1: A sample network: Seven Bridges of Königsberg

Common networks that most people use include the Internet or social media platforms. Some networks are manufactured or constructed such as electrical transmission lines, road systems and the flow of traffic [3, 29], or irrigation systems [1], while other

networks such as our cardiovascular system, ecosystems, and rivers [8] are natural.

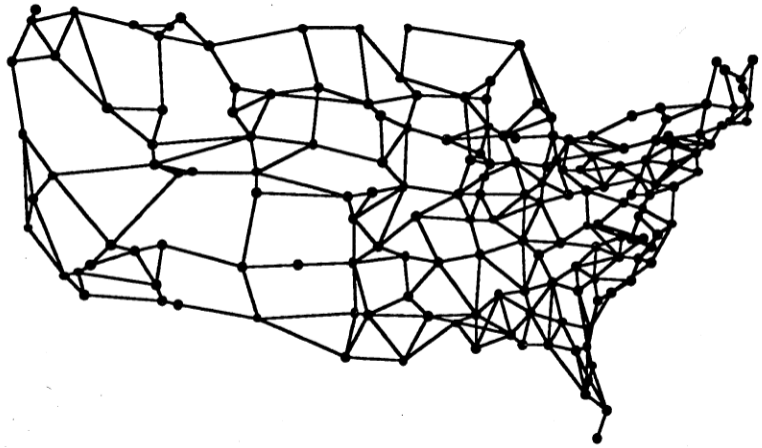

Figure 3.2: A traffic network [14]

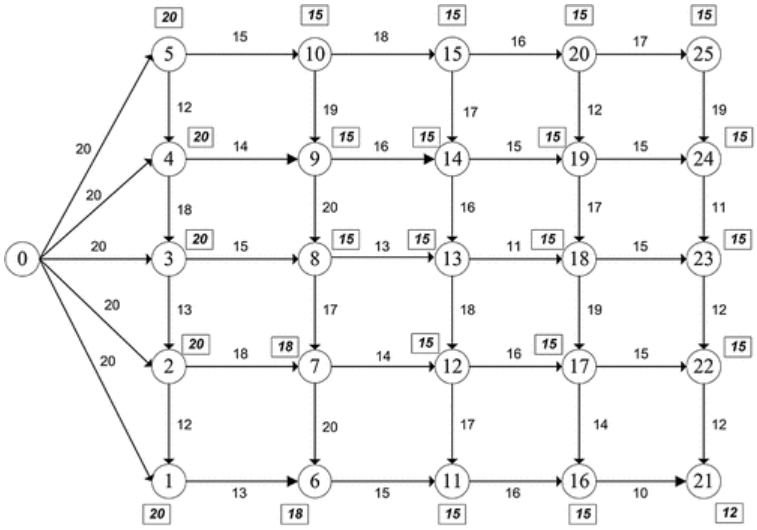

Figure 3.3: Design of an irrigation network system [1]

More interesting still are networks composed of a combination of man made structures which interpret or use biological structures. In [9] a brain to brain interface is constructed to relay neural activity between mice through inserted electrical wiring to form a neural link that shares sensorimotor impulses.

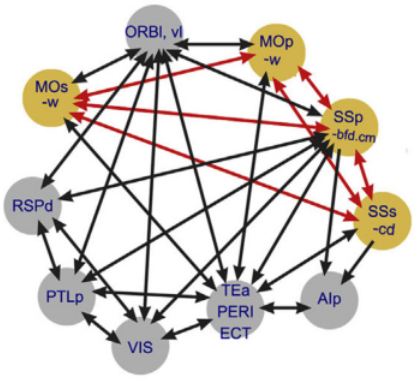

Figure 3.4: Sensorimotor whisker subnetwork of mice [10]

This could be modeled such that each mouse acts as a vertex and the mechanical connections between them as edges. The vertices represent the complex mouse neocortex, which are difficult to model, while the edges represent the less complex transfer of information from mouse to mouse. The simplified network for this is Figure 3.5.

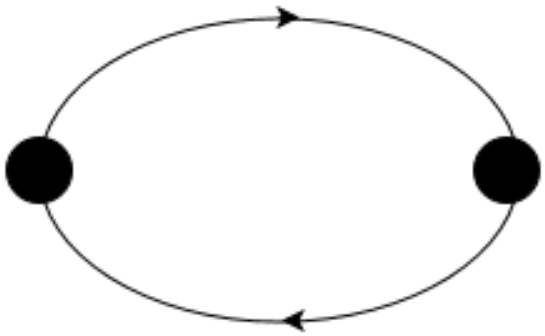

Figure 3.5: Simple oval network

In [28], an object that consists of a vertex set and an edge set is defined as a graph. Then any two elements of the vertex set are connected by one element of the edge set and edges that connect to the same vertex are said to be adjacent to one another.

Graphs are helpful in the sense that they can be represented with a drawing and allow us to see connections and envision movement or flow throughout the network. These ideas are formalized with the following definitions.

Definition 3 .1. A graph is a pair $\mathcal{G} = (V, E)$ such that $E \subseteq V \times V$. Thus each element of E is related to two elements of V. Elements of V are called vertices and elements of E are called edges [7]

Two additional definitions [24], which prove helpful when describing network properties and problems are added.

Definition 3 .2. A restriction $\mathcal{G}|_v$ of a graph $\mathcal{G}$ to a vertex v is defined by,

$$\mathcal{G}|_v = \{e \in E | e = (w, v) \text{ or } e = (v.w)\}. \tag{3.1}$$

This represents the set of all edges which connect to the vertex v

Definition 3 .3. The index $\mathcal{G}_v$ of a vertex v is defined by

$$\mathcal{G}(v) = |\mathcal{G}|_v|. \tag{3.2}$$

Thus the index of v is equal to the number of edges that connect to v on the graph. Graph theory is a robust and well developed field. Mathematical theory of graphs has become increasingly important as research has sifted from small group networks towards large scale networks driven by big data and higher computing power. Graphs

provide information about connectedness and clustering at a glance. When networks are large and complicated, the graphical representation of a given network can become less useful due to the high visual complications, as in Figure 3.6.

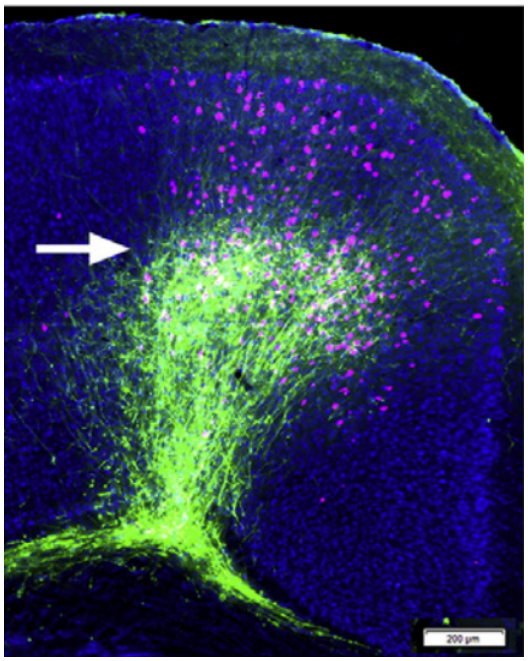

Figure 3.6: Multi-fluorescent labeled afferent and efferent pathways of a mouse neocortex [10]

For the problem addressed in this section, consider one specific class of network in which some substance or material is transported between vertices. The transport is physically multi-dimensional, so the differential equation used to model this transport can also be multi-dimensional. Here it is beneficial to use a network with edges and vertices to represent portions of the domain. The edges define portions of the domain that can be approximated by a one-dimensional equation while the more complex regions of the domain that cannot be well approximated by one-dimensional equations are defined as the vertices. Assume that the number of elements of the vertex set

is small compared to the edge set. Consider a network transporting a trace amount of fertilizer through irrigation pipes, as in Figure 3.3 where the majority of the edge domain consists of straight pipes with small junctions, the vertices, connecting adjacent pipes together. Here consider a tracer to be an amount so small as to not effect the fluid dynamics of the system. The flow of the tracer through the straight pipes can be modeled with one-dimensional differential equations. At the pipe connections, the vertices, enforce algebraic conditions that model the multi-dimensional effects of the junctions. These conditions are referred to as junction conditions throughout this work. The proper specification of junction conditions is always an important part of the setup of a problem. In this work, the network of consideration is one in which the edges engender linear hyperbolic equations to which the domain decomposition method is applied to this linear hyperbolic network problem. Then resulting problem is solved numerically using serial and parallel architectures. Application of the numerical and computational methods to resolve a specific class of network, similar to that of Figure 3.3, is implemented to resolve such networks throughout this work.

3.1 Previous Work

Networks are useful and as such have been heavily applied. Biological networks have been studied, including neural networks in [9, 23, 10] where maps of neural activity have been produced through the use of tracer injections of chemicals and bio-florescent materials. The statistical properties of "small world" type networks

such as electrical power grids and air traffic routes are examined in [17]. Small world networks are those in which travel from vertex to vertex is accomplished by the use of few edges relative to the size of the overall network and are characterized by finite dimensionality [17]. In [4], a model of laser drilling and that of gas flow through a pipe are presented along with an analysis of each problem. Determination of simple and accurate junction conditions to model the physical properties involved at vertices is a subject of major research in network transport problems. Much of the work on network type problems is directed at determining junction conditions and the well-posedness of the problem. In [4], the treatment of these types of problems is explored in a mathematical sense. Particular attention is paid to the numerical method used to approximate the multi-dimensional physics where junction conditions, numbering at least $G|_v$, $v \in V$, are needed at each vertex and it is assumed the conditions are linear in the state variables. If these conditions were not linear, they could be linearized about density ρ and velocity u. The linear state variable junction conditions are coupled and relate the network edges to one another.

3.1.1 Linear Acoustics Equations on a Network Domain

In this section the linear acoustic equation detailed in [18] and the notation of [5] are employed. These equations and notations are used throughout the remainder of this work. Define a network $\mathcal{G} = (V, E)$ as in (3 .1) and further let there be N edges, denoted e_i, on the unit interval $[a_{e_i}, b_{e_i}]$. Note that the interval can be parameterized

if necessary. The linear acoustic equations for any $e_i \in E$ of G are

$$\left(\frac{\partial}{\partial t} + \begin{bmatrix} 0 & K_{e_i} \\ \frac{1}{\rho_{e_i}} & 0 \end{bmatrix} \frac{\partial}{\partial x} \right) \begin{bmatrix} p_{e_i} \\ u_{e_i} \end{bmatrix} (x,t) = 0 \ , \ x \in (a_{e_i}, b_{e_i}) \ , \ t > 0, \tag{3.3}$$

where p_{e_i} is the pressure and u_{e_i} is the velocity. The quantity K_{e_i} is the bulk modulus and p_{e_i} is the density of the medium where both physical quantities are constant and positive. The initial conditions are

$$p_{e_i}(x,0) = \phi_{e_i}(x) \ , \ x \in [a_{e_i}, b_{e_i}], \tag{3.4}$$

$$u_{e_i}(x,0) = \psi_{e_i}(x) \ , \ x \in [a_{e_i}, b_{e_i}]. \tag{3.5}$$

Note that the eigenvalues of the coefficient matrix in (3.3) are $\lambda_1 = c_{e_i}$ and $\lambda_2 = -c_{e_i}$ where $c_{e_i} = \sqrt{K_{e_i}/\rho_{e_i}} > 0$ is the speed of sound. Observe that (3.3) is a coupled system. A change of variables [18] is applied, to determine the characteristic variables

$w_{e_i} = \frac{1}{2Z_{e_i}}(-p_{e_i} + Z_{e_i} u_{e_i})$,

$z_{e_i} = \frac{1}{2Z_{e_i}}(p_{e_i} + Z_{e_i} u_{e_i})$,

in terms of pressure, velocity, and impedance $Z_{e_i} = \rho_{e_i} c_{e_i}$. This then turns (3.3) into the decoupled characteristic system

$$\frac{\partial}{\partial t} \begin{bmatrix} w_{e_i} \\ z_{e_i} \end{bmatrix} (x,t) + \begin{bmatrix} -c_{e_i} & 0 \\ 0 & c_{e_i} \end{bmatrix} \frac{\partial}{\partial x} \begin{bmatrix} w_{e_i} \\ z_{e_i} \end{bmatrix} (x,t) = 0. \tag{3.6}$$

Recall that a and b correspond to the vertices a_{e_i} and b_{e_i} of edge e_i. Then denote e_i^a as the vertex on edge e_i corresponding to a_{e_i} with an analogous notation for e_i^b. The adjacent edges to e_i are designated in the following manner:

incoming adjacent edges at vertex a_{e_i} as $\mathcal{A}_{e_i} = \left\{e_j \in E : e_j^b = e_i^a\right\}$

outgoing adjacent edges at vertex a_{e_i} as $\mathcal{B}_{e_i} = \left\{e_l \in E : e_l^a = e_i^a\right\}$

incoming adjacent edges at vertex b_{e_i} as $\mathcal{C}_{e_i} = \left\{e_m \in E : e_m^b = e_i^b\right\}$

outgoing adjacent edges at vertex b_{e_i} as $\mathcal{D}_{e_i} = \left\{e_n \in E : e_n^a = e_i^b\right\}$.

Figure 3.7 illustrates a sample network with adjacent edges to e_i. Thus the structure at any vertex can be defined using the sets

$$\mathcal{A}_{e_i} = \left\{e_j \in E : e_j^b = e_i^a\right\}, \tag{3.7}$$

$$\mathcal{B}_{e_i} = \left\{e_l \in E : e_l^a = e_i^a\right\}, \tag{3.8}$$

$$\mathcal{C}_{e_i} = \left\{e_m \in E : e_n^b = e_i^b\right\}, \tag{3.9}$$

$$\mathcal{D}_{e_i} = \left\{e_n \in E : e_n^a = e_i^b\right\}. \tag{3.10}$$

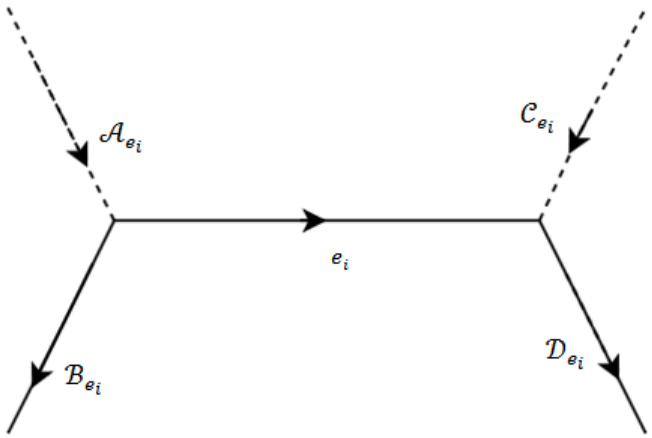

Figure 3.7: Adjacent edge sets

There is always one positive eigenvalue and one negative eigen value for the system in (3.6), so only one boundary condition is necessary at each endpoint on any given edge e_i. The number of junction conditions is equal to the number of edges connected at any given vertex v. The physical junction conditions that are imposed are continuity of pressure and conservation of momentum at v. The boundary conditions for edge e_i are then given by

$$\left.\begin{aligned} p_{e_i}(a_{e_i},t) &= p_{e_j}(b_{e_j},t)\ \forall e_j \in \mathcal{A}_{e_i} \\ p_{e_i}(a_{e_i},t) &= p_{e_l}(b_{e_l},t)\ \forall e_l \in \mathcal{B}_{e_i} \\ p_{e_i}(b_{e_i},t) &= p_{e_m}(b_{e_m},t)\ \forall e_m \in \mathcal{C}_{e_i} \\ p_{e_i}(b_{e_i},t) &= p_{e_n}(b_{e_n},t)\ \forall e_n \in \mathcal{D}_{e_i} \end{aligned}\right\} \forall t > 0, \text{ and } i = 1,\ \ldots N, \tag{3.11}$$

for continuity of pressure and

$$\left.\begin{aligned} u_{e_i}(a_{e_i,t}) &= \sum_{e_j \in \mathcal{A}_{e_i}} u_{e_j}(b_{e_j}, t) - \sum_{e_l \in \mathcal{B}_{e_i}} u_{e_l}(a_{e_l}, t) \\ u_{e_i}(b_{e_i,t}) &= \sum_{e_n \in \mathcal{D}_{e_i}} u_{e_n}(a_{e_n}, t) - \sum_{e_m \in \mathcal{C}_{e_i}} u_{e_m}(b_{e_m}, t), \end{aligned}\right\} \forall t > 0, \text{ and } i = 1, \ \ldots N, \tag{3.12}$$

for conservation of momentum. The problem defined in (3.3)-(3.5) with junction conditions (3.11)-(3.12) is well posed, see [15, 16], where a e the proof of the existence and uniqueness of this general problem is presented.

3.2 Linear Transport Problems on a Caveman Network

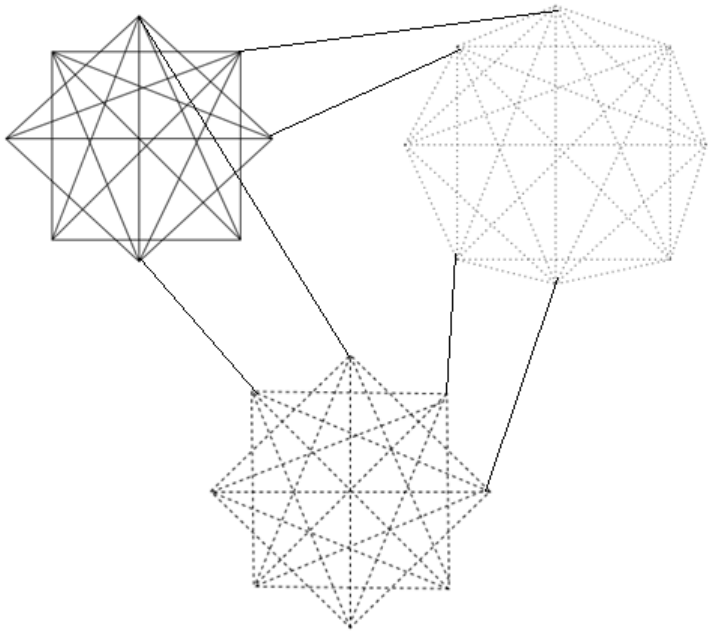

Figure 3.8: Sample caveman network

A caveman network is one where there are areas of high connectivity, represented as caves, joined by areas of low connectivity, represented as lines. The linear transport problem studied in this work represents a simplified caveman network where the oval sections act as the caves or areas of high complexity, and the straight edge connections act as areas of low complexity. Representation of this specific example as a cluster which allows the complicated network to be visualized in an approachable manner.

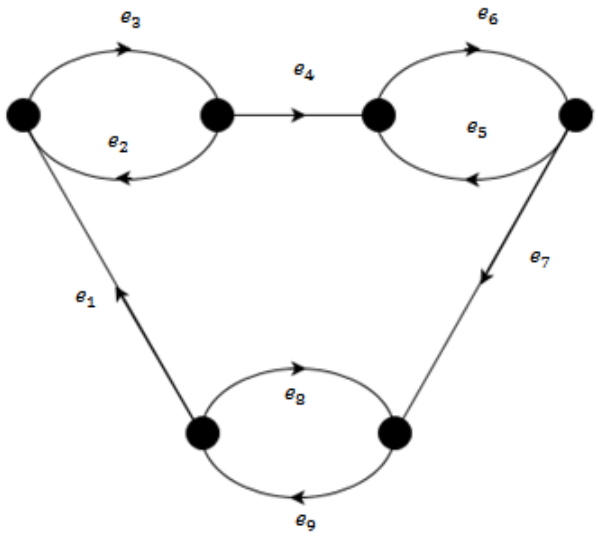

Figure 3.9: Simplified cluster network

To solve the serial problem described in (3.3)-(3.5) with junction conditions (3.11)-(3.12), the interior portion of the domain is updated for a given edge using a given numerical method where the junction conditions determine the boundary values for that edge. Note that these calculations are completed using the characteristic variables since the original system is coupled. This process is repeated for each time step until final time T. This entire process is repeated until the solution converges is a

predetermined way.

To solve this problem in parallel, the solution for each edge is solved on a CPU node of the parallel computer with the same algorithm as above. The master CPU node collates and combines all of the edge data to form a complete solution.

Now consider the particular class of network problems similar to those depicted in Figures (3.10) and (3.9).

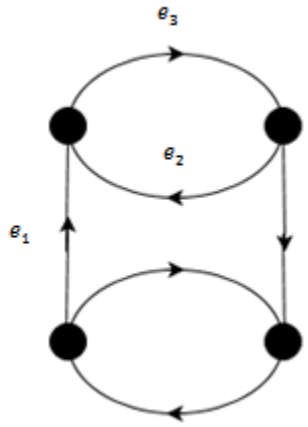

Figure 3.10: "Barrel" network

For the class of networks under consideration the degree of each vertex is always exactly 3, that is, $\mathcal{G}_3$ at every vertex $v \in V$. Given constant matrix $J \in \mathbb{R}^{3\times 3}$, which is determined by the conditions at each junction, the system of three equations that relate the state variables

$$\mathbf{q}_{e_i} = \begin{bmatrix} p_{e_i} \\ u_{e_i} \end{bmatrix}, \tag{3.13}$$

of the three edges that are connected at vertex v is the system,

$$\mathbf{J}\begin{bmatrix}\mathbf{q}_{e_1}\\ \mathbf{q}_{e_2}\\ \mathbf{q}_{e_3}\end{bmatrix} = 0. \tag{3.14}$$

As in (2.12) write $\mathbf{A}_{e_i} = \mathbf{R}_{e_i}\mathbf{\Lambda}_{e_i}\mathbf{R}_{e_i}^{-1}$ and then convert (3.3) into characteristic variables and obtain the decoupled system

$$\frac{\partial}{\partial t}w_{e_i}(x,t) - c_{e_i}\frac{\partial}{\partial x}w_{e_i}(x,t) = 0, \tag{3.15}$$

$$\frac{\partial}{\partial t}z_{e_i}(x,t) + c_{e_i}\frac{\partial}{\partial x}z_{e_i}(x,t) = 0, \tag{3.16}$$

which can then be solved using traditional methods.

3.2.1 Junction Conditions

In general junction conditions are difficult to set for an entire network. Junctions can have a varying number of edges connected at that vertex and the characteristic structure of the edges at the vertex further complicate generalization. The junction conditions for the acoustic equations are given by physical constraints. For this work, the junction conditions derived in [4] are applied. The first condition is continuity of pressure, that is, the pressure p is required to be continuous from edge to edge across

any given junction,

$$p_{e_j} = p_{e_i} \ \forall \ e_i \in \mathcal{G}_v \setminus \{e_j\} \ j = 1, \cdots, N. \tag{3.17}$$

Secondly, require that the momentum coming into a junction and the momentum leaving a junction are equal. Thus, enforce the condition that momentum is conserved across a junction. In [4], Collins indicates that there are two ways in which to denote this condition. Either the edges are of unit length or the a parameterization is assigned. This work only considers unit edges though edges can have a flow that is towards the vertex or away from the vertex. The edges whose flow is towards the vertex are incoming and edges whose flow is leaving the vertex are outgoing. As in [4], the set of incoming edges to v are denoted $I_p(v)$ and $O_p(v)$ denotes the set of outgoing edges to v. The conservation of momentum relation becomes

$$\sum_{e_i \in I_p(v)} \left| u_{e_i} \right| = \sum_{e_j \in O_p(v)} \left| u_{e_j} \right|. \tag{3.18}$$

The conditions set forth in (3.17) and (3.18) provide exactly three conditions at each vertex. Since three edges connect at each vertex, there are six characteristics to be calculated in order to solve the system. Three of the characteristics can be interpolated back into an edge domain while the remaining three are determined by the junction conditions. Thus the system is readily determined at each junction. The specific class of networks examined here is simplified in characteristic structure

and limits the number of edges connected at a vertex. Recall that the number of edges connected at each vertex is exactly three and that due to the direction of flow on each, there are only two cases to consider when handling the characteristics. At each junction, the characteristic variables with a direction of flow that is towards the vertex can be determined by the methods in [2]. The value of either the w or z characteristic at the junction is linearly interpolated by tracing the characteristic lines using data from the previous time step. Then a system of equations is solved to determine the remaining three characteristic values, which are then used to solve for the state variables p and u on the boundary at each time step. The state variables are related to the characteristic variables by

$$p_{e_i}(v,t) = Z_{e_i}(z_{e_i}(v,t) - w_{e_i}(v,t)) \tag{3.19}$$

$$u_{e_i}(v,t) = w_{e_i}(v,t) + z_{e_i}(v,t), \tag{3.20}$$

where $Z_{e_i} = \rho_{e_i} c_{e_i}$ is the impedance for $i = 1, 2, 3$. Using (3.17) - (3.20) the junction conditions may be rewritten in terms of the characteristics variables. Noting that the incoming variables z_{e_1}, z_{e_2}, and w_{e_3} may be calculated at the boundary by interpolation to the known data at the previous time step [18, 29, 30]. Some rearranging gives

the invertible system

$$\begin{bmatrix} -Z_{e_1} & 0 & -Z_{e_3} \\ 0 & -Z_{e_2} & -Z_{e_3} \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} w_{e_1}(v,t) \\ w_{e_2}(v,t) \\ z_{e_3}(v,t) \end{bmatrix} = \begin{bmatrix} -Z_{e_1} z_{e_1}(v,t) - Z_{e_3} w_{e_3}(v,t) \\ -Z_{e_2} z_{e_2}(v,t) - Z_{e_3} w_{e_3}(v,t) \\ -z_{e_1}(v,t) - z_{e_2}(v,t) + w_{e3}(v,t) \end{bmatrix}. \quad (3.21)$$

The boundary conditions are defined on each edge for all time in this network. The state variables on the boundary can then be calculated from the characteristic variables at each junction at each time step. It is worth noting that for networks with configurations different from that under study here, the junction conditions yet hold so that a system similar to (3.21) can be solved at each vertex to determine the state variables at a boundary.

3.3 Domain Decomposition on a Network

To numerically solve a problem, such as that previously discussed, a computer is employed. For the network in this study, the problem is solved for each of the interior domain nodes associated with each edge and then the solution for the boundaries on each edge may be resolved. This process is repeated for each time step until the final time is reached. This serial computational approach is an inefficient way to solve this type of problem. Typically most simulations have areas where work can be distributed and run simultaneously, i.e. in parallel. This is one of the reasons most computers in use have new generation processors that run in parallel. Specifically,

computers are commonly used that have dual or quad cores, meaning that they have multiple processors. Assuming that a code used to solve a numeric problem is as efficient as it can be, parallel processing is the only way to achieve a faster computational, unless it is possible to remove or manipulate data dependencies within an algorithm. Memory requirements for direct solvers are too costly, as they require $\mathcal{O}(m^3)$ work [27] and iterative solvers are not robust enough. Different iterative and direct solver methods are the needed. One such class of options are domain decomposition methods. Domain decomposition is an approach that is often used for large scale computational problems by splitting the domain into smaller sub-domains. This method works by finding a solution of the problem on each sub-domain separately. In particular, interest in these methods has peaked due to their applicability to the numerical resolution of boundary value problems [20]. Implementation of the domain decomposition method requires one to divide up the domain of the problem. In splitting the domain, artificial boundaries are created and boundary conditions must be imposed along these new boundaries. This is where it is possible to remove data dependencies and take advantage of the iterative process of this method. Rather than using the current values on the adjacent sub-domain, the data from the previous iterate is used, which means that the work is reduced to $\mathcal{O}(m^2)$ and in some special cases to $\mathcal{O}(m)$. The procedure begins with some initial iterate for the solution on all sub-domains that is not the solution for all space and time. Then at each time step on each sub-domain, solve the system at each interior domain node along with the associated

boundary problem. This process is repeated until a predetermined stopping criteria, commonly a small change from one iteration to the next, is achieved. Some of the benefits of using domain decomposition include, ease when dealing with complicated geometries, the ability to use different discretization on different sub-domains, and that splitting the problem up into smaller problems without data dependency lends itself to parallelization [4]. The proof of convergence results for many well-known problems is presented in [25]. Domain decomposition for time-dependent problems can be handled in different ways. In [20], the author presents an approach of temporal discretization where a spatial problem is solved at each time step. Then, at each time step, boundary data must be communicated across sub-domains. This means that many small packets of data are communicated, at each time step, which may dramatically increase overhead [21]. In parallel processing it is more efficient to transfer larger packets of information less frequently than it is to require frequent and small communication across processors [26]. For this reason it is necessary to implement an approach that partitions the spatial domain and then applies the time discretization to each sub-domain. This allows a solution to be determined at each edge to final time T on a separate CPU node. The entire solution on that sub-domain is then sent back to the master CPU node. This approach greatly reduces overhead and allows for full exploitation of the domain decomposition method and parallel processing speeds. In this work only the linear acoustic equations on a network are considered, to which domain decomposition is applied such that each sub-domain is one edge. In partic-

ular the additive Schwarz [25] domain decomposition algorithm is employed, which has been proven to converge using the maximum principal [20]. A further benefit for large systems may be achieved through splitting the domain into into multi-edges per sub-domain [20] solved with parallel algorithms. Resolution of the linear acoustic problem is initiated by defining an initial iterate

$$\mathbf{q}_{e_i}^0(x,t) = \mathbf{h}_{e_i}(x,t)\ \forall e_i \in E,\ x \in (a_{e_i}, b_{e_i}),\ 0 < t \leq T \tag{3.22}$$

as the solution on each sub-domain, i.e. every edge. This initial iterate is defined for all time t less than the final time T. The network is partitioned into S disjoint connected sub-domains $\mathcal{H}_n$

$$\mathcal{G} = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \cdots \cup \mathcal{H}_S \tag{3.23}$$

$$\mathcal{H}_n = (V_n, E_n)\ \ \forall n = 1,2,\ \ldots,S. \tag{3.24}$$

Then solve

$$\frac{\partial}{\partial t}\mathbf{q}_{e_i}^k + \mathbf{A}_{e_i}\frac{\partial}{\partial x}\mathbf{q}_{e_i}^k = 0 \; e_i \in \mathcal{H}_n,\, x \in (a_{e_i}, b_{e_i}),\, 0, t \leq T \tag{3.25}$$

$$\mathbf{J_a}\begin{bmatrix} \mathbf{q}_{\hat{e}_1}^k \\ \vdots \\ \mathbf{q}_{\hat{e}_L}^k \\ \mathbf{q}_{\tilde{e}_1}^{k-1} \\ \vdots \\ \mathbf{q}_{\tilde{e}_N}^{k-1} \end{bmatrix} = 0 \; \hat{e}_l \in \mathcal{G}|_a \cap \mathcal{H}_n \; , \; \tilde{e}_l \in \mathcal{G}|_a \backslash \mathcal{H}_n \tag{3.26}$$

$$\mathbf{J_b}\begin{bmatrix} \mathbf{q}_{\hat{e}_1}^k \\ \vdots \\ \mathbf{q}_{\hat{e}_L}^k \\ \mathbf{q}_{\tilde{e}_1}^{k-1} \\ \vdots \\ \mathbf{q}_{\tilde{e}_L}^{k-1} \end{bmatrix} = 0 \; \hat{e}_l \in \mathcal{G}|_b \cap \mathcal{H}_n \; , \; \tilde{e}_l \in \mathcal{G}|_b \backslash \mathcal{H}_n \tag{3.27}$$

where a and b correspond to the vertices a_{e_i} and b_{e_i} of edge e_i. The matrices $\mathbf{J_a}$ and $\mathbf{J_b}$ correspond to the junction conditions at the endpoints of edge e_i and relate the state variables at iterate k with the ones on adjacent edges at iterate $k-1$. These junction conditions are the generalized conditions from [4, 5]. Equation (3.21) may

then be rewritten in general as

$$\begin{bmatrix} -Z_{e_1} & 0 & -Z_{e_3} \\ 0 & -Z_{e_2} & -Z_{e_3} \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} w_{e_1}^k(v,t) \\ w_{e_2}^{k-1}(v,t) \\ z_{e3}^{k-1}(v,t) \end{bmatrix} = \begin{bmatrix} -Z_{e_1} z_{e_1}(v,t) - Z_{e_3}^{k-1} w_{e_3}^{k-1}(v,t) \\ -Z_{e_2}^{k-1} z_{e_2}^{k-1}(v,t) - Z_{e_3}^{k-1} w_{e_3}^{k-1}(v,t) \\ -z_{e_1}(v,t) - z_{e_2}^{k-1}(v,t) + w_{e3}^{k-1}(v,t) \end{bmatrix} \tag{3.28}$$

Thus the boundary conditions are determined by interpolating the characteristics whose flow is into the vertex followed by solving the invertible system in (3.28) for the remaining characteristics. The state variables $\mathbf{q}_{e_i}$ at vertex v are recovered via

$$\mathbf{q}_{e_i} = \begin{bmatrix} p_{e_i} \\ u_{e_i} \end{bmatrix} = \mathbf{R} \begin{bmatrix} w_{e_i} \\ z_{e_i} \end{bmatrix} \tag{3.29}$$

This process is repeated until the solution converges. Since the previous iterate from the adjacent edges is used to determine the solution for edge e_i, data transfers only at the beginning and end of each iteration rather than at each time step. This greatly reduces overhead and allows us to distribute computations for each edge to a CPU node to solve, thereby completing the computations for each edge simultaneously leading to decreased run time to solve the network problem. Here lies the power of parallel computing.

CHAPTER IV

CONVERGENCE ANALYSIS

In this chapter, previous analysis for general networks from [5, 4] is examined and then these results are applied to the specific network class studied in this work. The network class under consideration is any network of similar structure to figure (3.9) or (3.10). That is any caveman type network with clustered areas joined by areas of low connectivity. The caves are represented by the oval regions and the straight edges act as the low connectivity regions. Note that any size network can be constructed from any number of caves with the same number of connecting edges. For the analysis of this work, limit the number of edges connecting the caves is limited to one so that there are only three edges connected to any given vertex.

4.1 Previous Work

Some convergence results for domain decomposition methods convergence are examined in [4]. Further, intuition as to how convergence occurs hs been addressed by the author. For simplicity the two edge oval network is considered and defined to have unit edge length and unit speed of sound and is partitioned so that there is one edge

per sub-domain. This simple example is explored to examine convergence and then the ideas are expanded to networks with varying edge length and speed of sound.

4.1.1 Convergence of Domain Decomposition

Since the advection equation is a hyperbolic problem, each edge only receives boundary data on one boundary. Also, the speed of propagation is positive indicating that data would enter on the left side boundary. Assume that the initial iterate is not the solution for all space and time and further that given exact boundary conditions exact solutions are obtained. That is, assume exact solutions so that any error other than that of the initial iterate can be ignored. The latter is due to the assumption that the only error induced will come from the initial iterate. Certainly, this is not entirely realistic as the numerical solution is an approximate solution due in part to computer representation of numbers. By nature, all numeric methods introduce some error and most numbers can not be represented accurately by computers [27, 2] which use floating point representations of numbers. Floating point numbers can also create catastrophic cancellation, which is an extreme loss of significant figures [13]. Collins in [4] determines convergence properties of the domain decomposition method under the above assumptions. The error is expected to decrease with each successive iteration and the interval of exactness will increase along with the reduction of error.

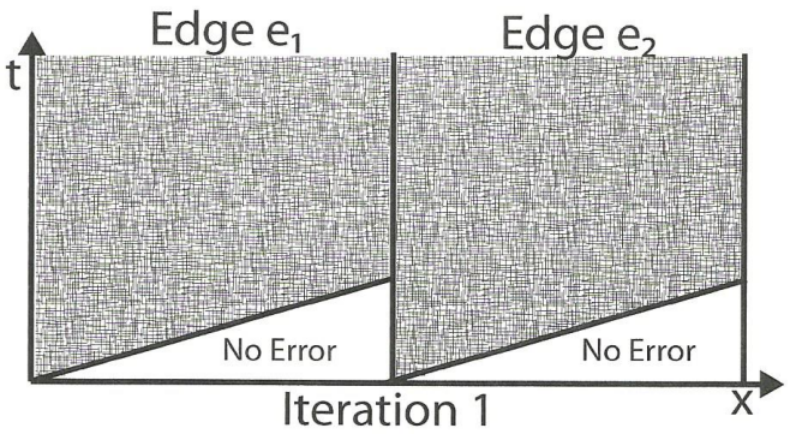

Figure 4.1: Error after one iteration of the oval network [4]

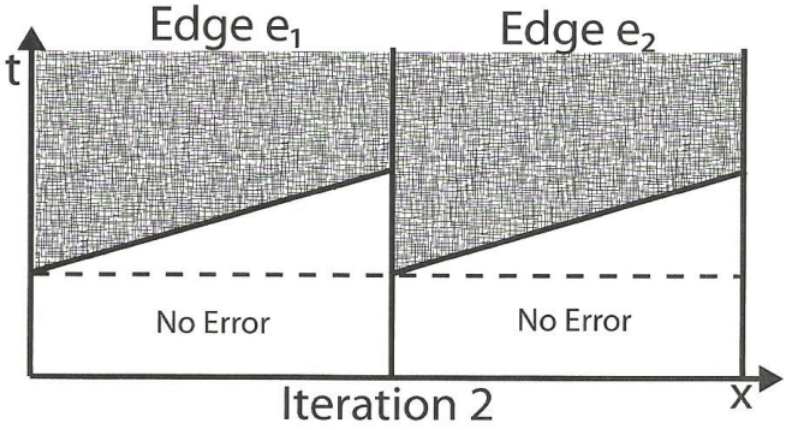

Figure 4.2: Error after two iterations of the oval network [4]

Since the problem is hyperbolic the speed of propagation is finite. Also boundary data is only taken from one boundary. Thus the domain decomposition method is expected to converge in a finite number of iterations.

4.1.2 Generalization to Networks

Consider the general network problem introduced in [4]. In particular, the network $\mathcal{G}$ is any connected graph with unit edge length and varying constant speed of sound

c_{e_i} on each unit length edge $e_i \in E$. As indicated in [4], a $\mathcal{G}$ with varying edge length, L_{e_i}, can be parametrized with the change of variables $\hat{c}_{e_i} = \frac{x - a_{e_i}}{L_{e_i}}$. The network is further partitioned such that each sub-domain has exactly one edge. The convergence variables are defined as follows

$$\tau_{e_i}^k \quad = sup\left\{t > 0 : z_{e_i}\big(x,t\big) - z_{e_i}^k\big(x,t\big) = 0 \ \forall x \in \big(a_{e_i}, b_{e_i}\big)\right\}, \tag{4.1}$$

$$\sigma_{e_i}^k \quad = sup\left\{t > 0 : w_{e_i}\big(x,t\big) - w_{e_i}^k\big(x,t\big) = 0 \ \forall x \in \big(a_{e_i}, b_{e_i}\big)\right\}. \tag{4.2}$$

They correspond to the largest time for which convergence has been attained on any edge e_i after k iterations for each characteristic variable w_{e_i} and z_{e_i}. The convergence variables may be thought of as the height of the region of exactness [4]. As the number of iterations increase, so do these times and the height of this region. The area of exactness propagates through the domain in an advective manner. With each successive iteration this area increases as the exact solution propagates from the left boundary into the domain. The convergence variables may be denoted

$$\tau_{e_i}^k \quad \mapsto \mathbf{w}_{e_i}^k\left(c\right) =: z_{e_i}^k \tag{4.3}$$

$$\sigma_{e_i}^k \quad \mapsto \mathbf{w}_{e_i}^k\left(-c\right) =: w_{e_i}^k \tag{4.4}$$

where $\mathbf{w}_{e_i}^K(c)$ and $\mathbf{w}_{e_i}^K(-c)$ are the characteristic variables associated with eigenvalue c and $-c$. From [4], these convergence variables are further defined as

$$\tau_{e_i}^{k+1} = min_{e_j \in \mathcal{A}, e_l \in \mathcal{B}} \{\tau_{e_j}^k, \sigma_{e_l}^k\} + \frac{1}{c} \tag{4.5}$$

$$\sigma_{e_i}^{k+1} = min_{e_m \in \mathcal{C}, e_n \in \mathcal{D}} \{\tau_{e_m}^k, \sigma_{e_n}^k\} + \frac{1}{c} \tag{4.6}$$

The following two lemmas are needed in later convergence analysis and represent how the error propagates through the domain, that is they can be used to provide information about τ_{e_i} and σ_{e_i}.

Lemma 4 .1. *Let $\mathcal{E}(x,t)$ satisfy the following differential equation*

$$\begin{cases} \frac{\partial}{\partial t}\mathcal{E}(x,t) + \alpha\frac{\partial}{\partial x}\mathcal{E}(x,t) = 0 \ \forall x \in [0,1], t > 0 \\ \mathcal{E}(x,0) = 0 \ \forall x \in [0,1] \\ \mathcal{E}(0,t) = \psi_E(t) \ t > 0 \end{cases}$$

where $\alpha > 0$. If $\mathcal{E}(x,t) = 0 \ \forall x \in [0,1]$ and $t < \omega$ then $\mathcal{E}(1,t) = 0$ for $t < \omega + \frac{1}{\alpha}$ [5]

Lemma 4 .2. *Let $\mathcal{F}(x,t)$ satisfy the following differential equation*

$$\begin{cases} \frac{\partial}{\partial t}\mathcal{F}(x,t) + \beta\frac{\partial}{\partial x}\mathcal{F}(x,t) = 0 \ \forall x \in [0,1], t > 0 \\ \mathcal{F}(x,0) = 0 \ \forall x \in [0,1] \\ \mathcal{F}(0,t) = \psi_F(t) \ t > 0 \end{cases}$$

where $\beta < 0$. If $\mathcal{F}(x,t) = 0 \ \forall x \in [0,1]$ and $t < \omega$ then $\mathcal{F}(1,t) = 0$ for $t < \omega + \frac{1}{|\beta|}$ [5]

These lemmas are used in the proof of the following theorem.

Theorem 4 .3. *Convergence Variable Update Formula: Given a connected network $\mathcal{G}$ with the acoustic network problem as defined in [5, 4], if the network problem is solved with the additive Schwarz domain decomposition algorithm with the network partitioned such that each sub-domain contains exactly one edge, the convergence variables satisfy the update formula*

$$\tau_{e_i}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_j}^k + \frac{1}{c_{e_j}}\ ,\ \sigma_{e_l}^k + \frac{1}{c_{e_l}},\ \tau_{e_m}^k + \frac{1}{c_{e_m}} + \frac{1}{c_{e_i}},\ \sigma_{e_n}^k + \frac{1}{c_{e_n}} + \frac{1}{c_{e_i}}\right\}$$

$$\sigma_{e_i}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_m}^k + \frac{1}{c_{e_m}}\ ,\ \sigma_{en}^k + \frac{1}{c_{e_n}},\ \tau_{e_j}^k + \frac{1}{c_{e_j}} + \frac{1}{c_{e_i}},\ \sigma_{e_l}^k + \frac{1}{c_{e_l}} + \frac{1}{c_{e_i}}\right\}$$

For proof of Lemma 4 .1, Lemma 4 .2, and Theorem 4 .3 see [5].

4.2 Variable Speed and Constant Edge Length

The network $\mathcal{G}$ that studied in this work differs in that the degree of any vertex is always 3, denoted as $\mathcal{G}|_3$ with the number of varying constant speeds limited to two, c_1 and c_2 with $c_1 < c_2$. Here c_1 corresponds to the speed of sound on the straight edges of the network and c_2 corresponds to the speed of sound on the curved outer and inner oval edges of Figure 4.3. There are three possible flow configurations at each vertex; two adjacent edge flows entering the vertex, two adjacent edge flows leaving the vertex, and a combination of an adjacent edge flow entering while the other adjacent edge flow is leaving the vertex. As such three cases are considered for the convergence variables. Then denote each edge e_i with a superscript O,I, or S to represent the outside curved edges, inside curved edges, and the straight edges.

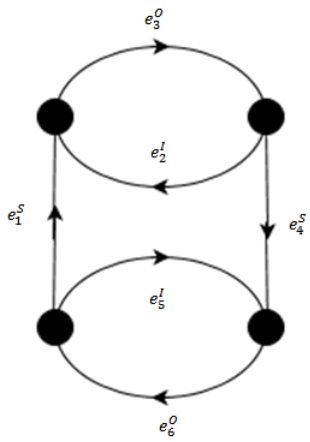

Figure 4.3: Example network with edge designation

Corollary 4 .4. *Convergence Variable Update Formula for Variable Speed and Constant Edge Length. Given a connected network with an acoustic network problem defined as above, then solve it with the Additive Schwarz domain decomposition algorithm with exactly one edge per sub-domain; the convergence variables satisfy the following update formulas:*

$$\tau_{e_i{}^O}^{k+1} \geq min_{\substack{e_j\in\mathcal{A}\\ e_n\in\mathcal{D}}} \left\{\tau_{e_{j_1}}^k + \tfrac{1}{c_1}\ ,\ \tau_{e_{j_2}}^k + \tfrac{1}{c_2}\ ,\ \sigma_{e_{n_1}}^k + \tfrac{1}{c_1} + \tfrac{1}{c_2}\ ,\ \sigma_{e_{n_2}}^k + \tfrac{1}{c_2} + \tfrac{1}{c_2}\right\} \quad (4.7)$$

$$\sigma_{e_i{}^O}^{k+1} \geq min_{\substack{e_j\in\mathcal{A}\\ e_n\in\mathcal{D}}} \left\{\sigma_{e_{n_1}}^k + \tfrac{1}{c_1},\ \sigma_{e_{n_2}}^k + \tfrac{1}{c_2},\ \tau_{e_{j_1}}^k + \tfrac{1}{c_1} + \tfrac{1}{c_2},\ \tau_{e_{j_2}}^k + \tfrac{1}{c_2} + \tfrac{1}{c_2}\right\} \quad (4.8)$$

$$\tau_{e_i{}^I}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_j}^k + \tfrac{1}{c_2}\ ,\ \sigma_{e_l}^k + \tfrac{1}{c_1},\ \tau_{e_m}^k + \tfrac{1}{c_1} + \tfrac{1}{c_2},\ \sigma_{e_n}^k + \tfrac{1}{c_2} + \tfrac{1}{c_2}\right\} \quad (4.9)$$

$$\sigma_{e_i{}^I}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_m}^k + \tfrac{1}{c_1}\ ,\ \sigma_{e_n}^k + \tfrac{1}{c_2},\ \tau_{e_j}^k + \tfrac{1}{c_2} + \tfrac{1}{c_2},\ \sigma_{e_l}^k + \tfrac{1}{c_1} + \tfrac{1}{c_2}\right\} \quad (4.10)$$

$$\tau_{e_i^S}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_j}^k + \tfrac{1}{c_2}\ ,\ \sigma_{e_l}^k + \tfrac{1}{c_2},\ \tau_{e_m}^k + \tfrac{1}{c_2} + \tfrac{1}{c_1},\ \sigma_{e_n}^k + \tfrac{1}{c_2} + \tfrac{1}{c_1}\right\} \quad (4.11)$$

$$\sigma_{e_i^S}^{k+1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau_{e_m}^k + \tfrac{1}{c_2}\ ,\ \sigma_{e_n}^k + \tfrac{1}{c_2},\ \tau_{e_j}^k + \tfrac{1}{c_2} + \tfrac{1}{c_1},\ \sigma_{e_l}^k + \tfrac{1}{c_2} + \tfrac{1}{c_1}\right\} \quad (4.12)$$

Proof. The convergence variable update formulas for this network are a direct application of Lemma 4 .1 and Lemma 4 .2 with speed of propagation c_1 or c_2 depending on edge e_i and a special application of Theorem 4 .3. The construction of the network $\mathcal{G}$ means that three cases must be considered for the general convergence variable update formulas. Denote the inside and outside curved edges as e_i^I and e_i^O respectively with the straight edges denoted as e_i^S. Suppose that the speed of sound on any curved edge e_i^O or e_i^I is c_2 and the speed of sound on any straight edge e_i^S is c_1, with $c_1 < c_2$. Observe that $\mathcal{E}^k_{e_i^O}$, $\mathcal{F}^k_{e_i^O}$, $\mathcal{E}^k_{e_i^I}$, $\mathcal{F}^k_{e_i^I}$, $\mathcal{E}^k_{e_i^S}$, and $\mathcal{F}^k_{e_i^S}$ satisfy lemmas (4 .1)-(4 .2) respectively with the boundary conditions $\psi_{E_O}(t)$, $\psi_{F_O}(t)$, $\psi_{E_I}(t)$, $\psi_{F_I}(t)$, $\psi_{E_S}(t)$, and $\psi_{F_S}(t)$. Consider the curved outer edges of the oval, e_i^O. Here the case is where both adjacent edge flows are either entering or leaving the vertex. Without loss of generality supposed that the configuration is as in Figure 4.3 and denote multiple entering or leaving edges with subscripts 1 and 2. The boundary condition $\psi^{k+1}_{E_O}(t)$ is obtained by solving the linear system of junction conditions similar to (3.28). This leads to

$$\tau^{k+1}_{e_i^O} \geq min_{\substack{e_j\in\mathcal{A}\\ e_n\in\mathcal{D}}} \left\{ \tau^k_{e_{j_1}} + \tfrac{1}{c_1} \, , \, \tau^k_{e_{j_2}} + \tfrac{1}{c_2} \, , \, \sigma^k_{e_{n_1}} + \tfrac{1}{c_1} + \tfrac{1}{c_2} \, , \, \sigma^k_{e_{n_2}} + \tfrac{1}{c_2} + \tfrac{1}{c_2} \right\} \quad (4.13)$$

$$\sigma^{k+1}_{e_i^O} \geq min_{\substack{e_j\in\mathcal{A}\\ e_n\in\mathcal{D}}} \left\{ \sigma^k_{e_{n_1}} + \tfrac{1}{c_1}, \, \sigma^k_{e_{n_2}} + \tfrac{1}{c_2}, \, \tau^k_{e_{j_1}} + \tfrac{1}{c_1} + \tfrac{1}{c_2}, \, \tau^k_{e_{j_2}} + \tfrac{1}{c_2} + \tfrac{1}{c_2} \right\} \quad (4.14)$$

The remaining two cases are derived in a similar fashion. Consider the curved inner edges of the oval, denoted e_i^I. Here the case is where one adjacent edge flow is entering the vertex while the other adjacent edge flow is leaving the vertex depicted in Figure (4.3), so that

$$\tau^{k+1}_{e_i^I,c_2} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau^k_{e_j}+\tfrac{1}{c_2}\ ,\ \sigma^k_{e_l}+\tfrac{1}{c_1},\ \tau^k_{e_m}+\tfrac{1}{c_1}+\tfrac{1}{c_2},\ \sigma^k_{e_n}+\tfrac{1}{c_2}+\tfrac{1}{c_2}\right\} \quad (4.15)$$

$$\sigma^{k+1}_{e_i^I,c_2} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau^k_{e_m}+\tfrac{1}{c_1}\ ,\ \sigma^k_{e_n}+\tfrac{1}{c_2},\ \tau^k_{e_j}+\tfrac{1}{c_2}+\tfrac{1}{c_2},\ \sigma^k_{e_l}+\tfrac{1}{c_1}+\tfrac{1}{c_2}\right\}. \quad (4.16)$$

Lastly consider the update formulas for the straight edges and note that any incoming flow and any outgoing flow will be from a curved edge with speed of sound c_2. The update formulas for this case are

$$\tau^{k+1}_{e_i^S,c_1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau^k_{e_j}+\tfrac{1}{c_2}\ ,\ \sigma^k_{e_l}+\tfrac{1}{c_2},\ \tau^k_{e_m}+\tfrac{1}{c_2}+\tfrac{1}{c_1},\ \sigma^k_{e_n}+\tfrac{1}{c_2}+\tfrac{1}{c_1}\right\}, \quad (4.17)$$

$$\sigma^{k+1}_{e_i^S,c_1} \geq min_{\substack{e_j\in\mathcal{A},e_l\in\mathcal{B}\\ e_m\in\mathcal{C},e_n\in\mathcal{D}}} \left\{\tau^k_{e_m}+\tfrac{1}{c_2}\ ,\ \sigma^k_{e_n}+\tfrac{1}{c_2},\ \tau^k_{e_j}+\tfrac{1}{c_2}+\tfrac{1}{c_1},\ \sigma^k_{e_l}+\tfrac{1}{c_2}+\tfrac{1}{c_1}\right\}. \quad (4.18)$$

□

Theorem 4 .5. *Given an acoustic network problem defined as in Corollary* (4 .4) *where the speed of sound on the curved edges is constant denoted by c_2 and the speed of sound on the straight edges is constant denoted by c_1 with $c_1 < c_2$. Then the total number of iterations, k, needed for the convergence is bounded by $k \leq Tc_2 + 1$, where T is the final time.*

To prove Theorem 4 .5 an additional lemma is necessary.

Lemma 4 .6. *For all $k > 0$ and $0 < c_1 < c_2$, $\frac{k-2}{c_2} + \frac{1}{c_1} > \frac{k-1}{c_2}$*

Proof. We wish to show that $\frac{k-2}{c_2} + \frac{1}{c_1} > \frac{k-1}{c_2}$

We have $c_1 < c_2$, then

$$c_1 k - c_1 > c_1 k - c_2$$

$$c_1 k - 2c_1 > c_1 k - c_1 - c_2$$

$$c_1(k-2) > c_1(k-1) - c_2$$

$$c_1(k-2) + c_2 > c_1(k-1)$$

$$\frac{c_1(k-2) + c_2}{c_1 c_2} > \frac{k-1}{c_2}$$

$$\frac{k-2}{c_2} + \frac{1}{c_1} > \frac{k-1}{c_2}$$

□

Theorem 4 .5

Proof. We show by induction that the convergence variables satisfy

$$\tau_{e_i}^k \geq \frac{k-1}{c_2} \tag{4.19}$$

$$\sigma_{e_i}^k \geq \frac{k-1}{c_2} \tag{4.20}$$

for all edges e_i and $k > 0$. Since the initial iterate is assumed to not be the solution for all space and time, thus for all curved and straight edges $\tau_{e_i^O}^k = \tau_{e_i^I}^k = \tau_{e_i^S}^k = \sigma_{e_i^O}^k = \sigma_{e_i^I}^k = \sigma_{e_i^S}^k = 0$. Thus the above inequalities hold for $k = 1$. Assume these hold for $k-1$ and using the the convergence variable formulas from Theorem (4 .4) and lemma

(4 .6), we have

$$\tau_{e_i^O}^k \geq min_{\substack{e_j \in \mathcal{A} \\ e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_1} \, , \, \frac{k-2}{c_2} + \frac{1}{c_2} \right\} = \frac{k-1}{c_2}$$

$$\sigma_{e_i^O}^k \geq min_{\substack{e_j \in \mathcal{A} \\ e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_1} \, , \, \frac{k-2}{c_2} + \frac{1}{c_2} \right\} = \frac{k-1}{c_2}$$

$$\tau_{e_i^I}^k \geq min_{\substack{e_j \in \mathcal{A}, e_l \in \mathcal{B} \\ e_m \in \mathcal{C}, e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_2} \, , \, \frac{k-2}{c_2} + \frac{1}{c_1} \right\} = \frac{k-1}{c_2}$$

$$\sigma_{e_i^I}^k \geq min_{\substack{e_j \in \mathcal{A}, e_l \in \mathcal{B} \\ e_m \in \mathcal{C}, e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_1} \, , \, \frac{k-2}{c_2} + \frac{1}{c_2} \right\} = \frac{k-1}{c_2}$$

$$\tau_{e_i^S}^k \geq min_{\substack{e_j \in \mathcal{A}, e_l \in \mathcal{B} \\ e_m \in \mathcal{C}, e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_2} \, , \, \frac{k-2}{c_2} + \frac{1}{c_2} \right\} = \frac{k-1}{c_2}$$

$$\sigma_{e_i^S}^k \geq min_{\substack{e_j \in \mathcal{A}, e_l \in \mathcal{B} \\ e_m \in \mathcal{C}, e_n \in \mathcal{D}}} \left\{ \frac{k-2}{c_2} + \frac{1}{c_2} \, , \, \frac{k-2}{c_2} + \frac{1}{c_2} \right\} = \frac{k-1}{c_2}$$

Then setting $\tau_{e_i^O}^k = \tau_{e_i^I}^k = \tau_{e_i^S}^k = \tau_{e_i}^k = T$ and $\sigma_{e_i^O}^k = \sigma_{e_i^I}^k = \sigma_{e_i^S}^k = \sigma_{e_i}^k = T$ and then solving for k, we obtain the desired inequality. □

Thus an *a priori* bound on the number of required iterations for convergence that is dependent on the faster speed c_2 and final time T has been determined. It is important to note that Theorem 4 .5 holds for any caveman network with any number of caves where there are exactly two vertices per cave and when all vertices connect exactly three edges. Thus Theorem 4 .5 extends to any network in the caveman class. As any error except for the error in the initial iterate was not taken into consideration, the *mathita priori* bound is an estimate of the true bound. Therefore it is expected that the numerical results will require an actual number of iterations that may be somewhat higher than the estimate.

CHAPTER V

NUMERICAL RESULTS

In this chapter comparisons of the numerical results to the convergence analysis of chapter 4 are presented. In addition comparisons of aspects of the results due to serial or parallel computation are discussed. Using the Lax-Friedrichs method the linear acoustic equations are solved across three configurations of a given network class using the additive Schwarz domain decomposition algorithm. All configurations are solved with serial and parallel architectures in order to compare run time and to identify conditions such that parallel computation is advantageous. Theorem 4 .5 provides an estimate of the number of iterations to convergence for each of the three configurations. The infinity norm [13] determines convergence of network. The stopping criteria of the numerical algorithm is

$$||q^k - q^{k-1}|| < 10^{-6}, \tag{5.1}$$

where here

$$||V|| = ||V||_\infty = max_i\{|V_i|\}. \tag{5.2}$$

This measure being satisfied indicates a small change from iteration to iteration. Collins addresses justification of the update formulas of Theorem 4 .3 and by extension Theorem 4 .4, [4]. Graphical plots of the numerical results at various iterations are presented supporting the conjecture that error flows from the left boundary with a finite constant speed. These results match the graphical results of Figures (4.1) and (4.2).

5.1 Varying Speed and Constant Edge Length

For all numerical results, ρ_{e_i} and K_{e_i} are chosen such that the speed of sound on any straight edge is $c_1 = 1$ and is $c_2 = 2$ on any curved edge. These speeds were chosen for simplicity, but could be any numeric value. All runs used 100 domain nodes per spatial edge with a CFL condition of $\frac{1}{2}$. To eliminate any run time discrepancies, twenty simulation runs were completed in serial and in parallel for each configuration. The linear acoustic equations

$$\left(\frac{\partial}{\partial t} + \begin{bmatrix} 0 & K_{e_i} \\ \frac{1}{\rho_{e_i}} & 0 \end{bmatrix} \frac{\partial}{\partial x} \right) \begin{bmatrix} p_{e_i} \\ u_{e_i} \end{bmatrix} (x,t) = 0 \ , \ x \in (a_{e_i}, b_{e_i}) \ , \ t > 0, \tag{5.3}$$

$$p^0_{e_i}(x,0) \quad = \sin(2\pi x) \ e_i \in E, \ x \in (0,1) \tag{5.4}$$

$$u_{e_i}(x,0) \qquad = 0 \ e_i \in E, \ x \in (0,1) \tag{5.5}$$

for $i = 1 \cdots N$ are then solved over the network configurations represented in (3.5), (3.10), and (3.9). Observe that the initial conditions are continuous and satisfy the junction conditions for every vertex. The junction conditions of continuity of pressure and conservation of momentum discussed previously are enforced. The initial iterate is chosen to be

$$p^0_{e_i}(x, 0) = 5 \quad x \in (0, 1),\, t > 0 \tag{5.6}$$

$$u^0_{e_i}(x, 0) = 10 \quad e_i \in E,\, x \in (0, 1),\, t > 0. \tag{5.7}$$

As this is not the solution for all space and time, the choice assures that there will be initial error. Results for three network configurations are presented. All three configurations are small in scale but provide confirmation of the above analysis and are consistent with that of [5], where Collins notes in that it is advantageous to implement the domain decomposition method in parallel with P sub-domains and N iterations to convergence provided

$$N < P. \tag{5.8}$$

All networks are solved to final time $T = 1$.

5.1.1 Comparison of Numerical Results with Iteration Estimates

Results are presented for the three configurations of the caveman network class. The results were computed using the methods and criteria above. In Table 5.1 the it-

eration count results for all three configurations are presented. For the purely oval configuration speed of sound used was $c = 1$. The analytical iteration estimate is compared to the actual iteration count for serial and parallel architectures. The projected iteration count to resolved the problem on an oval network is the actual number of iterations to achieve convergence. The resolution of the problem defined over the barrel and cluster networks required two additional iterations numerically than what was projected. Since any error other than that introduced from the initial iterate was ignored, these additional iterations can be attributed to the numerical error introduced from the numerical method, that is, discretization error. The extra iterations were needed to address the error associated with the method. The problem defined over the oval network required less additional iterations to achieve convergence than that of the barrel or cluster networks. This is due to the simplicity of the oval network connections. The iteration count is close to the estimated count, which shows that the estimate is applicable for the caveman class of networks. Note that there is no difference in the number of iterations needed for convergence to a solution for either computational architecture implemented. A difference was not expected and was corroborated by the results presented.

	Oval	Barrel	Cluster
Analytical Estimate	2	3	3
Actual Serial Iteration Count	2	5	5
Actual Parallel Iteration Count	2	5	5

Table 5.1: Numeric Results for the Oval, Barrel, and Cluster networks

Since the network is partitioned such that there is exactly one sub-domain per edge, the number of sub-domains are equal to the total number of edges in any given network configuration. Using (5.8) and considering the oval case, $N = 2$ and $P = 2$, which requires $2 < 2$, but $2 \not< 2$, which suggests that parallel processing is not advantageous in this case. For the barrel case, $N = 5 < 6 = P$, thus it is expected that the run time would decrease when processing in parallel. The expectation is the same for the cluster where the number of iterations and processors is related by $5 < 9$.

Each simulation was run twenty times. Use of a mean run time avoids inconsistent data. The mean run time results given in seconds are presented in Table 5.2.

	Oval	Barrel	Cluster
Serial Run Time (s)	39.34915	190.2321	118.847
Parallel Run Time (s)	15.89011	43.01524	62.68962
Percent change	-59.62	-77.39	-47.25

Table 5.2: Run Time Results

The above results demonstrate that run time decreased to solve the problem defined on each of the configurations when implemented on a parallel architecture. The results for the barrel and the cluster networks were expected but the results for the oval network contradict the previous conclusion. This is possibly due to the fact that the network is so small and simple that there is limited overhead. The overhead is further limited by the small number of iterations to convergence. It is likely that this is not an inconsistency in the estimate $N < P$, but rather a consequence of the simplicity and rapid convergence of the problem defined on an oval configuration. Notice that the decrease in run time for the barrel configuration is large. The barrel network simulation was run twenty times per trial and the trial was run five times with similar results. It is possible that the decrease is so dramatic on this configuration because the network is large enough to reap the benefits of parallel computation and maybe because the part of the problem farmed out to each CPU node requires minimal message passing. Now further examine the run time graphically.

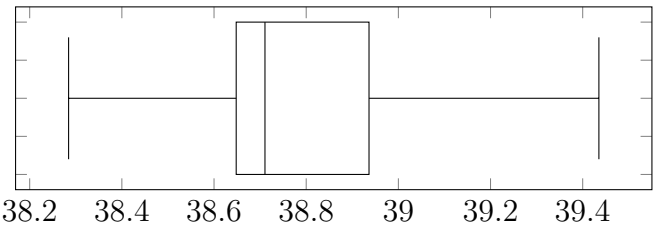

Figure 5.1: Oval Serial run time in seconds

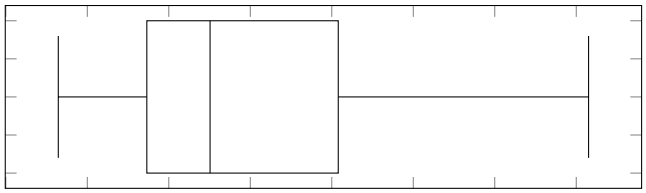

Figure 5.2: Oval Parallel run time in seconds

Here the run time results of computation for the oval network case on serial and parallel architectures are presented. All of the times are clustered tightly together such that the interquartile range is within the magnitude of 10^{-4}, thus all of the simulations reached convergence in approximately the same amount of time. The upper whisker for the parallel plot is extended due to several run times being higher in this configuration including two outliers at approximately 230 seconds. There are similar clustering results when various initial iterates are selected, suggesting that the initial iterate does not need to be accurate in order for convergence to occur. These results are similar for the barrel and cluster configurations.

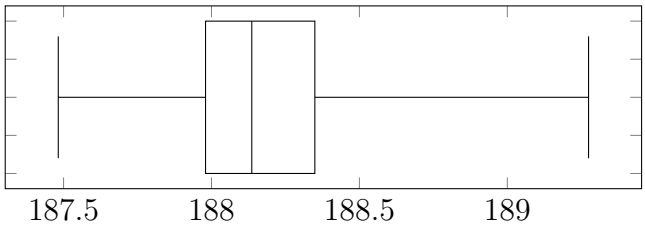

Figure 5.3: Barrel Serial run time in seconds

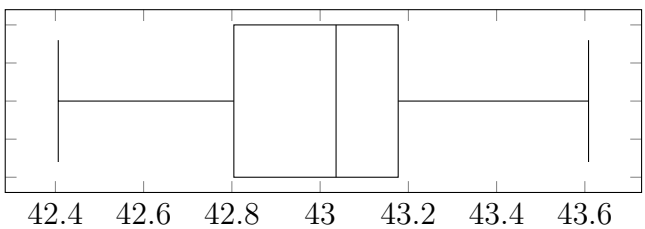

Figure 5.4: Barrel Parallel run time in seconds

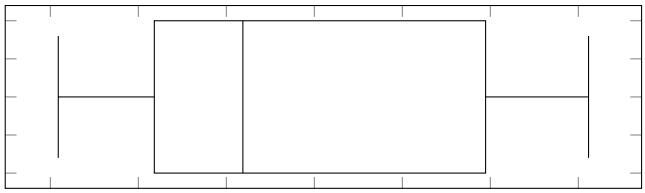

Figure 5.5: Cluster Serial run time in seconds

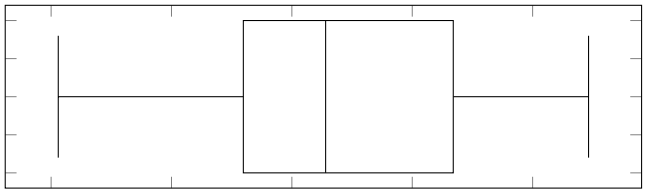

Figure 5.6: Cluster Parallel run time in seconds

In Figure 5.7 the run time for all of the configurations is compared using the median from the box plots above. The number of edges increases the overall run time as well. This is particularly visible for the parallel run times on the three configurations. These parallel times increase in a step like fashion as the the network complexity increased even though convergence of the solutions for the problems defined for the barrel and cluster networks occurred with the same number of iterations.

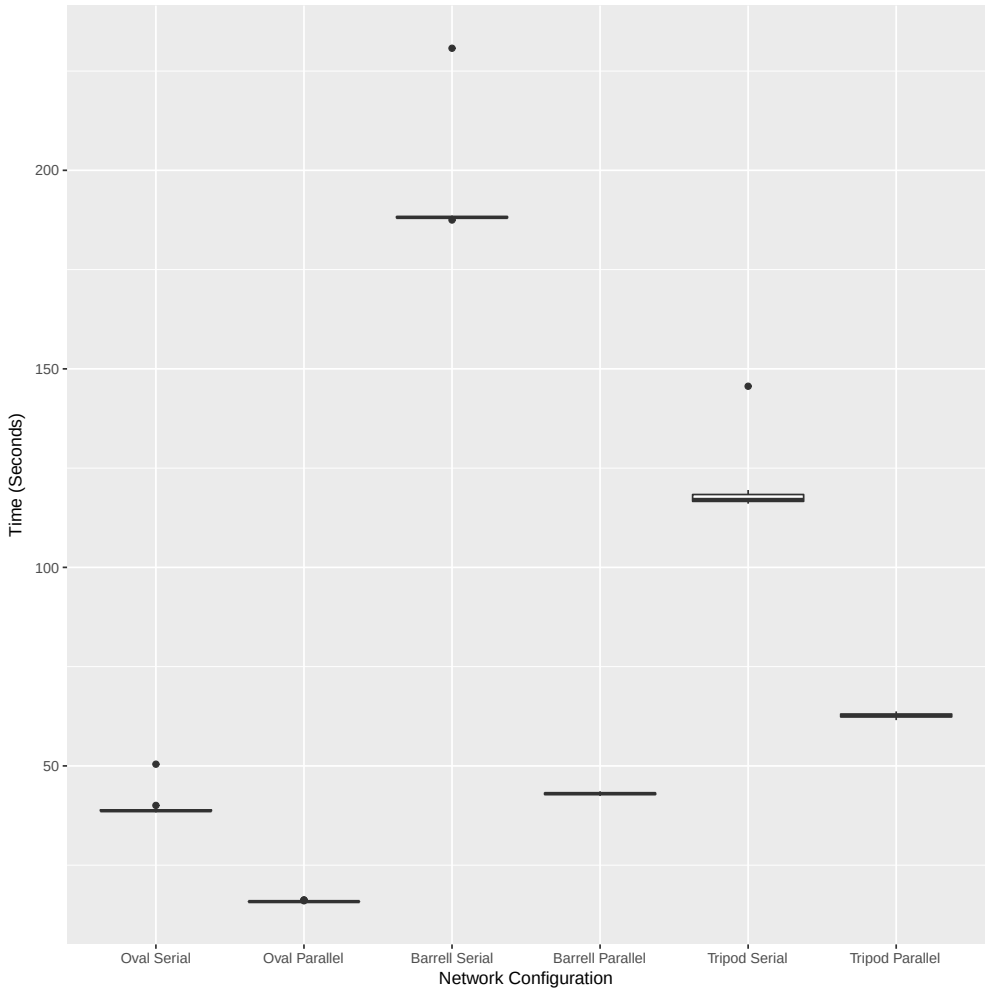

Figure 5.7: Run time comparison for all configurations

CHAPTER VI

CONCLUSIONS

In this work transport on a caveman network is considered on three configurations. The dimensionality of the network was reduced to simplify computation. Junction conditions for the networks were implemented that are consistent with those of previous investigations. The results achieved indicate that network problems solved using the domain decomposition methods can be solved on parallel architectures to increase computational efficiency. The convergence estimate determined analytically provides an *a priori* estimate on the number of iterations required for convergence of the discrete problem to to the exact solutions. However, the computations illustrate that due to the introduction of error the actual iteration count will only be nominally higher than predicted. Thus the estimate is meaningful and useful. Additionally, the iteration count is directly proportional to the final time T and the largest speed of propagation on the edges of the network. The iteration estimate is particularly helpful in the assessment of whether one should consider application of the method on a serial or parallel architecture. If resolution of the problem defined over the entire network takes N iterations, then computation on a serial architecture costs $\mathcal{O}(1)$

work. However, if computed on a parallel architecture with P processors under the constraint of (5.8), the work is reduced to $\mathcal{O}\left(\frac{N}{P}\right)$. Therefore, as long as constraint (5.8) is satisfied, it is advantageous to solve the problem defined over a network on a parallel architecture. The results in Chapter 5, indicate (5.8) holds, thus provide guidelines to determine when it is advantageous to resolve a problem using parallel architecture.

6.1 Future Research

The scope of this work is narrow and there were many aspects and avenues that were not explored. As mentioned in Chapter 1, the Lax-Friedrichs method is low order so implementation of a better algorithm could reduce run time as well as iterations to convergence. The networks studied in this work are simplified in structure, variation of speed of sound, and edge length. Though these networks can extend into "cave-man" like networks where there are high levels of clustering separated by single or few connections, further research into more complex and widely applicable networks is needed. In parallel computing, network structure plays a key role in designing optimal partitions for the parallelization process. As such, in depth analysis of optimal partitioning of sub-domains of a give network should be pursued. Network structure can be studied to identify key features in the domain geometry that affect the solution. Other avenues that can be explored include optimization of the algorithm and reduction of overhead transfer. The program used to run the simulation should

be further optimized for efficiency. Exploration of memory allocation and use could target areas of opportunity to further decrease run time.

References

[1] D. Alexiou and C. Tsouros. Design of an irrigation network system in terms of canal capacity using graph theory. *Journal of Irrigation and Drainage Engineering*, 143(6), 2017.

[2] K. E. Atkinson. *An Introduction to Numerical Analysis*, pages 11–38,131–158. John Wiley & Sons, Inc, 2 edition, 1989.

[3] G. M. Coclite and B. Piccoli. Traffic flow on a road network. eprint, 2002.

[4] J. B. Collins. *Dimension Reduction: Modeling and Numerical Analysis of Two Applied Problems*. phdthesis, North Carolina State University, Raleigh, NC, 2010.

[5] J. B. Collins. Analysis of domain decomposition method for linear transport problems on networks. *Applied Numerical Mathematics*, 109:61–72, 2017.

[6] R. M. Colombo and F. Marcellini. Smooth and discontinous junctions in the p-system. *J. Math. Anal. Appl.*, 361:440–456, 2009.

[7] R. Diestel. *Graphy Theory*, pages 1–6. Springer-Verlag, 2006.

[8] P. S. Dodds and D. H. Rothman. Geometry of river networks. iii. characterizaton of component connectivity. *Physical Review E*, 63, 2000.

[9] M. Pais-Vieira et al. A brain-to-brain interface for the real-time sharing of sensorimotor information. *Scientific Reports*, 3:1319, 2013.

[10] Zinng et al. Neural networks of the mouse neocortex. *Cell*, 156(5):1096–1111, 2014.

[11] S. J. Farlow. *Partial Differential Equatons for Scientists and Engineers*, pages 4,198. Dover Publications, 1993.

[12] E. Godlewski and P. A. Ravioart. *Numerical Approximations of Hyperbolic Systems of Conservation Laws*, chapter 1,5. Springer, New York, 1996.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*, pages 54–67, 275–289. Johns Hopkins University Press, 1996.

[14] H. Holden and N. H. Risebro. A mathematical model of traffic flow on a network of unidirectional roads. *SIAM Journal on Mathematical Analysis*, 26:999–1070, 2006.

[15] G. Leugering J. E. Lagnese, E. J. Schmidt. *Modeling, Analysis and Control of Dynamic Elastic Multi-Link Structures*. Birkhauser, Boston, 1994.

[16] J. E. Lagnese and G. Leugering. *Domain Decomposition Methods in Optimal Control of Partial Differential Equations*, chapter 9. Spinger Basel AG, 2004.

[17] M. Barthelemy L.A.N. Amaral, A. Scala and H.E. Stanley. Classes of small world networks. *Proc. Natl. Acad.Sci. USA*, 97:1114911152, 2000.

[18] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*, pages 1–157. Cambridge University Press, Cambridge, 2004.

[19] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*, pages 181–289. SIAM, 2007.

[20] P. L. Lions. On the schwartz alternating method. *Proc. First International Sympos. on Domain Decomposition Method for Partial Differential Equations, SIAM Philadelphia, PA*, 1988.

[21] J. J. Modi. *Parallel Algorithms and Matrix Computations*, pages 3–55. Oxford University Press, 1988.

[22] F. V. Moulder. *Social Problems of the Modern World*, pages 355–369. Wadsworth, 2000.

[23] V. Pieribone and D. F. Gruber. *Aglow in the dark: the revolutionary science of biofluorescence*. Belknap Press, 2006.

[24] Y. V. Pokornyi and A. V. Borovskikh. Differential equations on networks (geometric graphs). *Journal of Mathematical Sciences*, 119:691–718, 2004.

[25] A. Quarteroni and A. Valli. *Domain Decomposition Method for Partial Differential Equations*. Oxford University Press, New York, 1999.

[26] A. S. Tanenbaum and H. Bos. *Modern Operating Systems*, chapter 1-6. Pearson, New Jersey, 4 edition, 2015.

[27] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*, pages 97–249. SIAM, 1997.

[28] R. J. Trudeau. *Introduction to Graph Theory*, pages 19–20. Dover Pub. New York, 1993.

[29] E. C. Zachmanoglou and D. W. Thoe. *Introduction to Partial Differential Equations with Applications*, pages 72–76,112–152,361–367. Dover, New York, 1976.

[30] E. Zauderer. *Partial Differential Equations of Applied Mathematics*, pages 117–120. John Wiley & Sons, 1998.

Appendix A

BEOWULF CLUSTER

The Beowulf computer cluster is comprised of nearly identical, commodity-grade computer hardware resulting in a parallel computing cluster made from inexpensive personal computer hardware. Each processor is autonomous and may operate as a standalone computer. Ubuntu was installed on all CPU nodes of the cluster. Ubuntu is an open source operating system, that is a Debian-based Linux operating system for personal computers and other A net install was used to choose which software would be downloaded and installed. A minimal install with no GUI was used for the CPU nodes. The CPU nodes were all connected through a local area network(LAN) with a switch. The LAN allowed the configuration of Secure Shell (SSH) and Network File System (NFS).

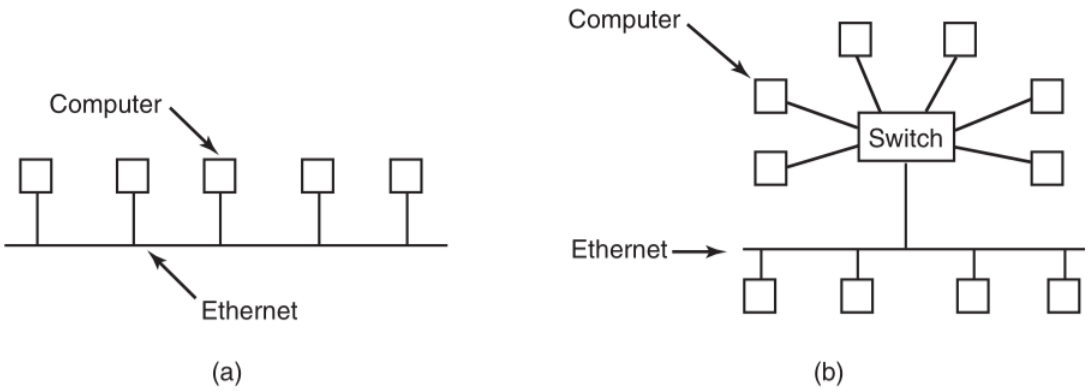

Figure 1.1: a) Classic Ethernet connection b) Switched Ethernet connection

On each CPU node a static IP and hostname was assigned. SSH is a network protocol for initializing text-based shell sessions on remote machines. This makes the operation of the cluster very easy. All the CPU nodes can be controlled through the head CPU node using SSH. Open MPI requires that the head CPU node can SSH into each machine with no password. To setup SSH, some cryptographic keys were created on the head CPU node, and then copied to each CPU node. With these keys, the master can SSH to any CPU node with no password necessary. NFS allows the CPU nodes to access a directory on the master. This seems to be the easiest way to transfer files between CPU nodes. To setup NFS, a directory on the head CPU node was created, changes were made to its security to allow access, and then the directory was exported. For the CPU nodes, a place to mount the shared directory was created and changed to mount the directory automatically during the boot process.

A message passing interface (MPI) is a communications protocol used for programming parallel computers. Open MPI is an open source messaging passing interface implementation that is developed in a true open source fashion and maintained by a consortium of academic, research, and industry partners. Open MPI is able to pool resources, technologies, and expertise from across the computing community and thus provides a high performance message passing library. As such it is used in many TOP500 supercomputers. Open MPI provides a wrapper for compilers and some simple commands to run a process on the cluster. Open MPI is the backbone of the cluster. MPI also handles all of the load balancing which is nontrivial see [13].

Appendix B

SPECIFICATIONS

Model	CPU MHz	Cache Size	CPU Cores
AMD Athlon IIx4 630 Processor	800	512 KB	4
Intel Core 2 Duo CPU E7500 @ 2.93GHz	2926.2333	3072 KB	2
Intel Core 2 Duo CPU E7500 @ 2.93GHz	2926.20	3072 KB	2
Intel Core 2 Duo CPU E7500 @ 2.93GHz	2926.01	3072 KB	2
Intel Core 2 Duo CPU E7500 @ 2.93GHz	2926.01	3072 KB	2
Intel Core 2 Duo CPU E8400 @ 3GHz	2000	6144 KB	2
Intel Core 2 Duo CPU E8400 @ 3GHz	2000	6144 KB	2
Intel Core 2 Duo CPU E7500 @ 2.93GHz	1600	3072 KB	2
Intel Core 2 Duo CPU E7500 @ 2.93GHz	1600	3072 KB	2

Table 2.1: Cluster CPU Specifications

The AMD Athlon processor is not a true quad core processor. It is referred to as a hyper-threaded core which is different that a true quad core in that a hyper-threaded core share the execution resource.