8-1-1994

An empirical investigation of algorithms for the convex hull problems.

Sheri Zimmel

AN EMPIRICAL INVESTIGATION OF ALGORITHMS FOR THE CONVEX HULL PROBLEM

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

University of Nebraska at Omaha

by

Sheri Zimmel

August 1994

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College, University of Nebraska, in partial fulfillment of the requirements for the degree Master of Arts, University of Nebraska at Omaha.

Committee

Name	Department
Betty Hicks	Computer Science
Dan Scott	Industrial and Management Systems Engineering
Hans Fuss	COMPUTER SCIENCE - UNO

Betty L Hicks
Chairman

8/25/94
Date

ABSTRACT

The objective of this thesis is to implement and empirically evaluate the performance of serial and parallel algorithms for the multidimensional convex hull problem. In recent years several new serial algorithms and parallel variants thereof have been developed. However, the algorithms were implemented on machines having very different architectures with speeds differing by orders of magnitude. As a result, no comparison of the implementations can be done. Inefficiencies, ranging from major to minor in scope, are also inherent in each of these previous implementations.

In this study we designed and implemented improved versions of existing serial algorithms for the multidimensional convex hull problem, designed and implemented improved parallel versions of these algorithms, and performed extensive computational testing to determine the relative merits of each. All of the algorithms are based on mathematical programming formulations – those in which an objective and a set of limitations, or constraints, are stated in the form of mathematical functions.

The computational results, obtained on a shared-memory multiprocessor, show that the algorithms vary in performance depending on certain characteristics of the problem data. However, one of the implementations was clearly superior to the others in all cases.

ACKNOWLEDGEMENTS

I would like to express my deepest thanks to my advisor, Dr. Betty Hickman, for her support, patience, encouragement, and inspiration. I feel very fortunate to have had the opportunity to work with her and I will always cherish her friendship.

My sincere appreciation goes to Dr. Dan Scott and Dr. Hassan Farhat for graciously serving on my committee. I greatly appreciate their suggestions, time and support.

I am deeply grateful to all of my former teachers for their time and dedication. I would especially like to thank Dr. Wallace Raab of the University of South Dakota for first introducing me to operations reasearch. I would also like to thank Professor John Lushbough of the University of South Dakota for instilling in me a love of computer science. At the University of Nebraska at Omaha, I would like to thank Professor Stanley Wileman and Dr. Arunachalam Raviehandran for being excellent teachers who always found time for their students.

I owe many thanks to my friends Lorie Manzer and Vicki VanCleave who have been a source of unceasing support and encouragement. I would also like to thank Scott Ames for not only being a friend, but also being a teacher. And I would like to thank Carol Becic for making my time at the Univeristy of Nebraska at Omaha much more enjoyable. Brian and Sherrie Baumgartner, Todd and Kris Olson, Dawn Kueter and Jord Turner deserve may thanks for providing not only friendship over many years, but also undeserved understanding and patience.

Finally, I would like to express my deepest gratitude to my family. I would like to thank my parents, Darlene and Roger Slocum, and Richard and Carol Zimmel for their love and support. I would also like to thank Steve Zimmel for being a true friend. This acknowledgment would be incomplete without expressing my love and thanks to my husband, Mark, who has given me immeasurable support, inspiration, love, and encouragement. His patience, and understanding are truly appreciated. No expression of thanks can adequately convey my feelings towards these very special people.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS iii

LIST OF TABLES v

Chapter

1. INTRODUCTION 1

1.1. Overview 1

1.2. History of Convex Hull Problem 3

1.3. Applications of Convex Hull Problem 3

1.4. Notations and Definitions 6

1.5. Parallel Computing Fundamentals 7

1.5.1. Objectives of Parallel Computing 7

1.5.2. Parallel Computer Architectures 7

1.5.3. Parallel Algorithm Design and Implementation Issues . 10

1.5.4. Programming on Tightly Coupled MIMD Multiprocessors 10

1.5.5. Monitor - Based Task Scheduling 11

1.6. Accomplishments 12

2. ALGORITHMS, IMPLEMENTATIONS, AND COMPUTATIONAL RESULTS 13

2.1. Base Algorithm 14

2.1.1. Serial Case 14

2.1.2. Parallel Case 16

2.2. Preprocessor and Quadratic Programming-Based Approach . . 16

2.2.1. Serial Case 16

2.2.2. Parallel Case 24

2.3. Two-Stage Linear Programming-Based Algorithm 29

2.3.1. Serial Case 29

2.3.2. Parallel Case 33
2.4. Expanding-Hull Linear Programming-Based Approach 35
2.4.1. Serial Case 35
2.4.2. Parallel Case 39
2.5. Computational Testing 41
3. SUMMARY AND CONCLUSIONS 61
BIBLIOGRAPHY 63

LIST OF ILLUSTRATIONS

Figure Page

1.1. Set of Points in the Plane 1

1.2. Set of Points with Convex Hull Identified 2

1.3. Definitions 8

1.4. Flynn's Categories 9

2.1. Base Algorithm – Serial Version 15

2.2. Base Algorithm – Parallel Version 17

2.3. Preprocessor 1 Algorithm – Serial Version 19

2.4. Preprocessor 2 Algorithm – Serial Version 20

2.5. Preprocessor 3 Algorithm – Serial Version 21

2.6. Specialized Frank-Wolfe and Variation of Preprocessor 3 Algorithm – Serial Version 23

2.7. Preprocessor 1 Algorithm – Parallel Version 25

2.8. Preprocessor 2 Algorithm – Parallel Version 26

2.9. Preprocessor 3 Algorithm – Parallel Version 27

2.10. Specialized Frank-Wolfe and Variation of Preprocessor 3 Algorithm – Parallel Version 28

2.11. Improved Two-stage Linear Programming-based Algorithm – Serial Version 31

2.12. Improved Two-stage Linear Programming-based Algorithm – Parallel Version 34

2.13. Expanding-Hull Linear Programming-Based Approach Algorithm – Parallel Version 38

2.14. Expanding-Hull Linear Programming-Based Approach Algorithm – Parallel Version 40

2.15. Speedup vs. Number of Processors for Set A – 816 Points, 14 Dimensions 45

2.16. Speedup vs. Number of Processors for Set A – 1000 Points, 6 Dimensions 45

2.17. Speedup vs. Number of Processors for Set A – 334 Points, 19 Dimensions 46

2.18. Speedup vs. Number of Processors for Set B – 4000 Points, 6 Dimensions 46

2.19. Speedup vs. Number of Processors for Set B – 4000 Points, 8 Dimensions 47

2.20. Speedup vs. Number of Processors for Set B – 4000 Points, 10 Dimensions 47

2.21. Speedup vs. Number of Processors for Set B – 4000 Points, 12 Dimensions 48

LIST OF TABLES

Table		Page
I.	Test Problem Characteristics	49
II.	Set A – Average Run Times	50
III.	Set B – Run Times (6 dimensions)	51
IV.	Set B – Run Times (8 dimensions)	52
V.	Set B – Run Times (10 dimensions)	53
VI.	Set B – Run Times (12 dimensions)	54
VII.	Set A – Average Speedups	55
VIII.	Set B – Speedups (6 dimensions)	56
IX.	Set B – Speedups (8 dimensions)	57
X.	Set B – Speedups (10 dimensions)	58
XI.	Set B – Speedups (12 dimensions)	59
XII.	Base Times	60

CHAPTER 1

INTRODUCTION

1.1 Overview

One of the fundamental problems in computational geometry is the determination of the *extreme* or *bounding* points of a finite set of points. These extreme points define the *convex hull* of the set. In two dimensions (the plane) the convex hull of a set of points may be thought of as the area encompassed by a rubber band stretched around all the points. Figure 1.1 below shows an arbitrary set of points in the plane and Figure 1.2 shows how the "rubber band" would be stretched around those points. The points (shown in grey) that are touched by the rubber band are the extreme points.

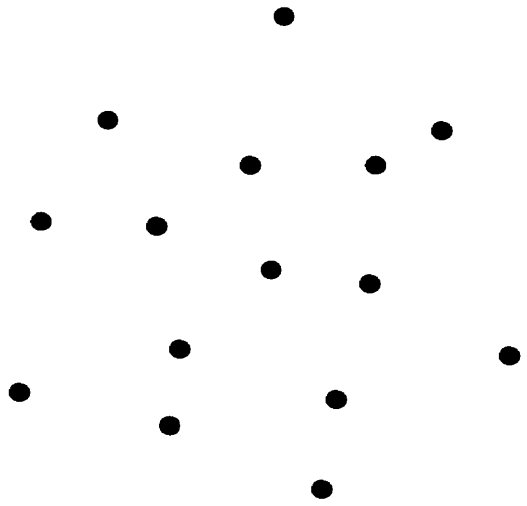

Figure 1.1 Set of Points in the Plane.

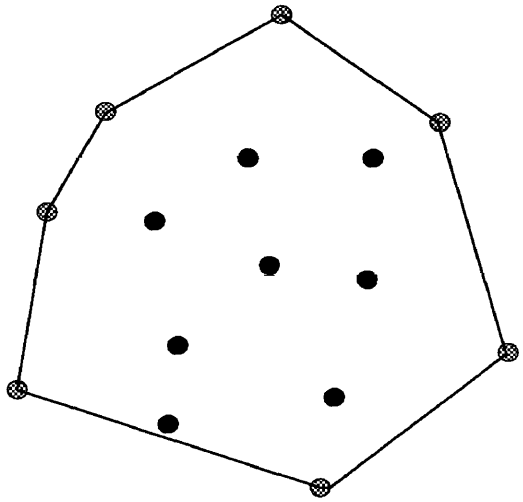

Figure 1.2 Set of Points with Convex Hull Identified.

Computer scientists have developed efficient algorithms for the convex hull problem when the points are restricted to two or three dimensions (Edelsbrunner 1987). However, the problem becomes much more difficult when the points are allowed to have dimensions greater than three – the so-called multidimensional case (Rosen, Xue, and Phillips 1991). Recent theoretical advances (Dulá and Helgason 1993) have opened the door for the development of more efficient algorithms for this case. Additionally, the availability of parallel computers provides opportunities for the solution of problems deemed unsolvable using current serial computers. In order to capitalize on the capabilities of parallel computers, new algorithms must be developed and implemented.

This thesis contributes to the development and evaluation of algorithms for the multidimensional convex hull problem. We present a discussion of several serial and parallel algorithms, their implementations, and the results of comparative computational testing.

1.2 History of Convex Hull Problem

Although it plays a supporting role in the solution of many more difficult problems, very few studies have addressed the convex hull problem directly. In fact, two different versions of the convex hull problem have been posed – the envelope and the frame (Preparata and Shamos 1985). In the envelope version, the objective is to determine a complete description of the boundary of the convex hull, whereas in the frame version we seek only to find the extreme points from the boundary of the convex hull. We will consider only the frame problem in this work.

One of the first studies was done by Wets and Witzall (1967) in which they presented a simplex-based approach to the problem of identifying the generating elements of a convex polyhedral cone – a problem equivalent to the frame problem. Wallace and Wets (1992) presented a more formal version of this procedure.

Rosen, Xue and Phillips (1991) developed and implemented serial and parallel algorithms for solving the convex hull problem that used a two stage method based entirely on the solution of linear programs. Shortly thereafter, Dulá, Helgason and Hickman (1992) presented a serial algorithm for the convex hull problem that employed a number of geometric preprocessors along with a specialized version of the Frank-Wolfe algorithm for solving intermediate quadratic programs. This was the first method for solving the convex hull problem that did not use linear programming. Recently Dulá and Helgason (1993) developed a serial algorithm which is based on linear programming along with geometric processing techniques. Dulá, Helgason, and Venugopal (1993) reported on a parallel version of this algorithm.

1.3 Applications of Convex Hull Problem

Applications of the convex hull problem appear in many different fields. In computer graphics, the convex hull problem is important (Pokorney and Gerald

1988) when clipping a B-spline curve against a window (the portion of the scene that is mapped onto the computer screen) or a view volume (a 3-D window). A B-spline curve is constructed by taking a linear combination of basic splines which are piecewise functions that form a basis for the construction of curves from a set of control points. A closed B-spline curve (one that connects to itself) has the property that it will never cross the convex hull of its defining control points. Therefore when clipping a curve one may first check if the convex hull of the control points intersects the clipping area. If it does not, then neither does the B-spline curve. The curve itself need only be examined if the convex hull intersects the clipping area.

In image processing and pattern recognition the convex hull is used for classifying and delineating objects in images (Rosenfeld 1969, Pratt 1991, Duda and Hart 1973). An irregularly shaped object can be described by constructing the smallest convex hull possible around that object and observing which points are within the convex hull but not within the object and which points are within both. The former set of points constitutes the convex deficiency of the object. Convex deficiencies are divided into two types: bays, which consist of the regions that lie between the convex hull perimeter and the object; and lakes, which are regions totally enclosed by the object. The convex hull, lakes, and bays provide a means of partitioning a complicated figure into several less complicated figures. The total figure may then be described in terms of the number of partitions and the position of each of the partitions.

The convex hull is also used in the construction of Voronoi diagrams (Graham and Yao 1990). Given a set S containing n points, a Voronoi diagram is constructed from n Voronoi regions. A Voronoi region $V(p)$ is defined for each point $p \in S$ and consists of all points in S that are closer to p than to any other point (except

themselves); i.e., $V(p) = \{q \in S \mid ||p - q|| < ||r - q|| \ \forall \ r \in S, r \neq q\}$ The n Voronoi regions that form a partition of the plane are called a Voronoi diagram of S. The problem of finding the set of n points which are the actual vertices of a Voronoi region (a convex polygon) can be modeled as a convex hull problem. Brown (1979) observed that the Voronoi diagram of a set in $\Re^d$ corresponds to the convex hull of a transformed set in $\Re^{d+1}$. Thus convex hull solutions may be used to find Voronoi diagrams.

In applied statistics, the extreme points of the convex hull of a set of observations correspond to outliers in the sample. Many estimators of parameters of a population are sensitive to the presence of outliers. One group of estimators known as the Gastwirth estimators (Gastwirth 1966) removes the effects of these outliers by using α*-trimmed* estimators. These α*-trimmed* estimators are calculated by removing the upper and lower α fraction of the points if in $\Re^1$ or by removing the extreme points of the convex hull of the points if in $\Re^d$. If outliers still exist, after the removal of the extreme points from the original set of points another application of the algorithm will remove another set of extreme points. This process of removing extreme points is analogous to peeling an onion. The estimate uses only the first layers of the full set of points, which have less than a $(1 - 2\alpha)$ fraction of the total set of points.

The convex hull problem also has applications in economics as a component of data envelopment analysis (Dulá and Helgason 1993), an analytical tool used in the assessment of relative efficiencies of businesses. Introduced by Charnes, Cooper and Rhodes (1978), data envelopment analysis is based on the identification of the inputs and outputs of a business. For each business of a particular type, these inputs and outputs are quantified and organized in vector form. A subset of the extreme points of the convex hull of these vectors defines a *frontier*. The businesses corresponding to vectors on the frontier are deemed more efficient than the others.

The convex hull problem also has applications in mathematical programming. The set of feasible solutions of a linear program is defined by a collection of linear inequality constraints. A constraint is said to be redundant if it can be omitted without affecting the set of feasible solutions. If redundant constraints are included, the problems are larger than necessary, thereby increasing the cost of obtaining a solution. Redundant constraints in a linear programming problem correspond to extraneous variables in the problem's dual. Extraneous variables are those variables that will never be basic and therefore can be eliminated from the problem. Dulá (1991) established that the problem of identifying extraneous variables in a linear program is equivalent to finding the extreme points of the convex hull defined by the columns of the coefficient matrix.

Identifying the extreme points of the convex hull of a set of points is also an important feature of recent algorithms for solving the two-stage stochastic linear program with recourse (Wallace and Wets 1989, 1992). In this problem first stage decisions must be made before values of random parameters are available, at which time second stage adjustments – recourse actions – are permitted. Identification of the extreme points of the convex hull of a set of points is a key element in guaranteeing that recourse is always available in the second stage.

1.4 Notations and Definitions

The following notation will be used throughout this thesis. Uppercase italicized Latin letters are used to denote sets. Scalars are denoted by lowercase Latin or Greek letters. Vectors are denoted by lower case bold Latin letters. The i^{th} element of a vector $\mathbf{v}$ is given by v_i. The inner product of two vectors $\mathbf{u}$ and $\mathbf{v}$ will be denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$.

Given a finite set of n points in d dimensions $X = \{\mathbf{x^1}, \ldots, \mathbf{x^n}\} \subseteq \Re^d$ with $\mathbf{x^i} = (x_1^i, \ldots, x_d^i)$, we now define terms to be used throughout this thesis in Figure 1.3.

1.5 Parallel Computing Fundamentals

1.5.1 Objectives of Parallel Computing

Parallel processing technology makes it possible for multiple processing units to simultaneously manipulate data elements to solve a single problem. Parallel processing is a method by which the "wall clock" time required to solve a problem may be significantly reduced. Since parallel processing is typically more costly than traditional serial processing, the motivation for its use springs from the need to solve existing problems faster as well as to make tractable problems that are currently unsolvable.

One way to measure the quality of a parallel code is *speedup*, $S(\cdot)$ (Quinn 1994). Although speedup has been defined in a number of different ways (see Barr and Hickman 1993), we will use the ratio of the problem's solution time using the fastest one-processor code to the time using a parallel code and p processors on the same machine. *Linear speedup*, where $S(p) = p$, is considered to be ideal. It is also possible in certain instances to have *superlinear speedup*, where $S(p) > p$. Another measure of the quality of a parallel code is efficiency (Quinn 1994). Computed as $E(p) = S(p)/p$, efficiency indicates how well the code utilizes the available processors.

1.5.2 Parallel Computer Architectures

Because computer architectures are quite varied, a number of classification methods have been developed. One popular method developed by Flynn (1966) is based on the possible number of instruction streams and data streams that can be executed concurrently. An instruction stream is the sequence of coded steps the computer is

1. The *zero vector* is a vector whose components are all equal to zero, and will be denoted by $\mathbf{0}$.

2. The *norm* of a point $\mathbf{x} \in X$ is $\|\mathbf{x}\| = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$.

3. The *Euclidean distance* between two points $\mathbf{x^i}$, $\mathbf{x^j} \in X$ is $d(\mathbf{x^i}, \mathbf{x^j}) = \|\mathbf{x^i} - \mathbf{x^j}\|$.

4. A set X in $\Re^d$ is a *convex set* if $\mathbf{x^i}, \mathbf{x^j} \in X \Rightarrow \lambda\mathbf{x^i} + (1-\lambda)\mathbf{x^j} \in X$ for all $\lambda \in [0,1]$.

5. A *convex combination* of two points $\mathbf{x^i}$, $\mathbf{x^j} \in X$ is any point of the form $\lambda\mathbf{x^i} + (1-\lambda)\mathbf{x^j}$ where $\lambda \in [0,1]$.

6. A *strict convex combination* of two points $\mathbf{x^i}$, $\mathbf{x^j} \in X$ is any point of the form $\lambda\mathbf{x^i} + (1-\lambda)\mathbf{x^j}$ where $\lambda \in (0,1)$.

7. An *extreme point* $\mathbf{x^e}$ of a convex set X is a point that cannot be represented as a strict convex combination of two distinct points $\mathbf{x^i}$, $\mathbf{x^j} \in X$. Thus, if $\mathbf{x^e} = \lambda\mathbf{x^i} + (1-\lambda)\mathbf{x^j}$ with $\lambda \in (0,1)$ then $\mathbf{x^e} = \mathbf{x^i} = \mathbf{x^j}$.

8. The *convex hull* of a set X, denoted $ConvX$, is the set of all convex combinations of elements of X. Thus, $ConvX = \{\mathbf{x} \in \Re^d \mid \mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{x^i},\ \mathbf{x^i} \in X,\ \lambda_i \geq 0,\ \sum_{i=1}^{n} \lambda_i = 1\}$.

9. The *Euclidean distance* between a point $\mathbf{p}$ and a convex hull $ConvX$ is $d(\mathbf{p}, ConvX) = \inf_{\mathbf{x^j} \in ConvX} d(\mathbf{p}, \mathbf{x^j})$.

10. The *hyperplane* $H(\mathbf{a}, b) = \{\mathbf{x} \in \Re^d \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ where $\mathbf{a}$ is a non-zero vector normal to the hyperplane and b is the level value.

Figure 1.3 Definitions.

to carry out, while a data stream is a series of data on which the instruction stream is to be executed. The four categories in Figure 1.4 result.

This research was carried out in a tightly-coupled MIMD Sequent Symmetry S81. This environment includes a common bus, shared memory, and identical processors. The basic processing model of the S81 is that of a series of individual processors on which processes, which may be interacting, are scheduled by a Unix-based operating system (Sequent 1992).

- SISD (single-instruction stream, single-data stream): SISD computers have a single control unit, execute a single instruction stream, and operate on a single piece of data at a time. Thus a SISD computer is not capable of parallel computing, although multitasking can give a SISD computer the appearance of parallelism.
- SIMD (single-instruction stream, multiple-data stream): The SIMD category includes computers which permit synchronous data-parallel computing by allowing multiple processors to execute a single instruction stream on different data streams. This form of parallelism is very powerful in cases where the same set of instructions must be executed on large sets of data.
- MISD (multiple-instruction stream, single-data stream): At the present time, no general-purpose MISD computers exist.
- MIMD (multiple-instruction stream, multiple-data stream): Multiple independent processors share data via a common memory or by passing messages. If the time required to access a particular memory location is the same for all processors the system is said to be *tightly-coupled*; otherwise, it is *loosely-coupled*. There is no global synchronization mechanism; rather synchronization must be achieved explicitly and locally. Furthermore, different operations may be performed on different data streams simultaneously.

Figure 1.4 Flynn's Categories.

1.5.3 Parallel Algorithm Design and Implementation Issues

Application-level parallel processing, where multiple processing elements can be used to solve a single problem, offer a very promising advance over serial processing. If it is possible to divide and schedule the work to solve a problem between the different processing elements for simultaneous execution, the time required to solve the problem can be dramatically reduced. The importance of matching algorithmic steps with the architecture of the target computer is amplified when developing algorithms for execution on parallel computers. Therefore, the recent evolution in computing machinery requires that there also be an evolution in algorithms and their implementations.

1.5.4 Programming Tightly Coupled MIMD Multiprocessors

To take advantage of a parallel processing environment the work associated with an algorithm must be partitioned into a series of tasks. These tasks must then be assigned to separate processes for simultaneous execution. If the tasks are assigned ideally, each process will have the same amount of work to do, causing the workload to be balanced.

Two paradigms for partitioning tasks among processes are control parallelism and data parallelism. With control parallelism different operations are performed on different data simultaneously. With data parallelism, multiple processors execute identical pieces of code on different data streams. It is not necessary for a particular implementation to use one method of partitioning or the other exclusively. In fact, many realistic applications exploit a combination of the two.

Once the tasks have been partitioned, they are assigned to processes based on one of three scheduling algorithms. The three algorithms are: prescheduling, static scheduling or dynamic scheduling. In prescheduling, the tasks are assigned

to processes by the programmer before the program is compiled. A disadvantage of prescheduling is that the programs cannot adapt to the changes in the computing load or the number of available CPU's. Prescheduling is typically appropriate for control-parallel applications in which each process is assigned a specific task.

With static scheduling tasks are assigned to processes at run time using some predetermined method. The primary advantage of static scheduling over prescheduling is that the number of processes may be changed from one run to the next and the program may automatically adapt to changes in the system when distributing the tasks among the processes. Thus, the workload may be distributed among the total number of processors available at run time. Although little overhead is associated with static scheduling, if the task lengths vary significantly, some processes may finish early and stand idle while others work to finish their tasks.

Dynamic scheduling, like static scheduling, is done at run time; however, with dynamic scheduling each process gets its tasks from a shared queue instead of from a predetermined distribution. In this manner the overhead is increased, but dynamic load balancing can occur, since each process continues working as long as there are tasks to be done.

Synchronization of the actions of multiple processes is needed whenever these processes change values stored in shared memory. One method of achieving synchronization is to use a lock, which is a simple semaphore that can be used to allow only one process at a time to access a shared data structure. Algorithms that do not require process synchronization are said to be *asynchronous*.

1.5.5 Monitor-Based Task Scheduling

One means of synchronizing parallel processes is a programming construct called a *monitor* (Hoare 1974). This self-scheduled work allocation scheme consists of a

critical section of code, a shared work list that is accessible only within the critical section, and a delay queue for idle processes. The critical section of code is guarded by a lock which allows in only one process at a time. If a process is idle it tries to acquire the lock and enter the critical section. If it obtains the lock it may then select a task from the work list and update the list accordingly. The process then exits the critical section, so that another process may enter. Once a process has exited the critical section it performs its selected task and returns to the critical section to request a new task. If no work is available when a process enters the critical section, the process is placed in the delay queue until work becomes available, at which time it is released. Termination occurs when all of the processes are in the delay queue.

A self-scheduled method such as the monitor is very effective when the number of tasks and the time requirements of each task can vary widely or are unknown. Thus, the monitor accommodates the needs of our implementations where the number of tasks often depends on the number of points and/or dimensions which varies greatly from problem to problem.

1.6 Accomplishments

This thesis contributes to the development and evaluation of serial and parallel algorithms for solving the multidimensional convex hull problem. In Chapter II four solution approaches are described. We present improved versions of these algorithms and describe their implementation in serial and parallel. Also reported are the results of extensive computational testing based on two problems sets. This testing showed that our implementation of one of the algorithms consistently out-performs all other algorithms. This implementation is also clearly superior to a previous implementation of the same algorithm. Results also illustrate the effect of certain characteristics of the problem data on the performance of the various codes.

CHAPTER 2

ALGORITHMS, IMPLEMENTATIONS, AND COMPUTATIONAL RESULTS

In the past several years, new serial and parallel algorithms have been developed to solve the convex hull problem. These algorithms were implemented on different computer systems and tested using different input data. As a result, no reliable comparisons of the algorithms could be made. Additionally, each of these algorithms suffers from varying degrees of inefficiency. At the time this research was begun parallel counterparts for two of the serial algorithms had not been developed.

The objectives of this research were to: (1) design and implement improved versions of existing serial algorithms for the convex hull problem; (2) design and implement parallel versions of these serial algorithms; and (3) perform extensive computational testing to determine the relative merits of each of the algorithms. The results, obtained on a shared-memory multiprocessor, show that the one algorithm is clearly superior and that certain characteristics of the input data affect the performance of the different algorithms.

For the purpose of describing the various algorithms, we define the set X to be a finite set of n points in d dimensions, such that $X = \{\mathbf{x^1}, \ldots, \mathbf{x^n}\} \subseteq \Re^d$ and $\mathbf{x^i} = (x_1^i, \ldots, x_d^i)$. At any point in the algorithm, X will be partitioned into three subsets: X^E – the set of points that are known to be extreme; X^N – the set of points that are known to be non-extreme; and X^U – the set of points whose status has not yet been determined. At the beginning of each algorithm, $X^U = X$ and $X^N = X^E = \emptyset$.

2.1 Base Algorithm

2.1.1 Serial Case

A standard approach for determining if a point $\mathbf{x^k} \in X$ is an extreme point of $ConvX$ is based upon fundamental polyhedral theory and linear programming. The point $\mathbf{x^k} \in X$ is extreme if and only if $\mathbf{x^k}$ cannot be expressed as a convex combination of other elements of X. This can be determined by solving the following linear program.

$$\text{P1}(k)\text{: } z_1(k) = \text{Minimize} \quad \sum_{\substack{i=1 \\ i \neq k}}^{n} \lambda_i \tag{2.1}$$

$$\text{subject to} \quad \sum_{\substack{i=1 \\ i \neq k}}^{n} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.2}$$

$$\lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{2.3}$$

Dulá and Helgason (1993) proved the following result: If P1(k) is feasible then $\mathbf{x^k} \neq 0$ is an extreme point of $ConvX$ if and only if $z_1^*(k) > 1$ where $z_1^*(k)$ is the optimal objective value. If P1(k) is infeasible then $\mathbf{x^k} \neq 0$ is an extreme point of $ConvX$.

Based on this result, a straightforward method for determining the status (extreme or non-extreme) of each point $\mathbf{x^k} \in X$ is to solve P1(k) for each $k = 1, \ldots, n$. This requires the solution of n linear programs, each of size $d \times (n-1)$. Since the problems are typically large and dense, this is a computationally intensive and time-consuming task.

Our version of the base algorithm, shown in Figure 2.1, incorporates features designed to reduce the time required to solve these n linear programs. Note that if a point is found to be non-extreme, then it must be a convex combination of other

points in X. Therefore, it will never be basic and may be eliminated from subsequent problems (Ali 1993). We may revise problem P1(k) to reflect this as follows:

$$\text{P2}(k)\text{:}\ \ z_2(k) = \text{Minimize} \quad \sum_{\substack{i=1 \\ i\neq k \\ \mathbf{x}^i \notin X^N}}^{n} \lambda_i \tag{2.4}$$

$$\text{subject to} \quad \sum_{\substack{i=1 \\ i\neq k \\ \mathbf{x}^i \notin X^N}}^{n} \lambda_i \mathbf{x}^i = \mathbf{x}^k \tag{2.5}$$

$$\lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{2.6}$$

base_serial(X)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

	X	{set of points to be examined}

Variables

	k	{index of elements in X}
	n	{number of elements in X}
	$z_2^*(k)$	{objective function value}

Compute $\|\mathbf{x}\|\ \forall\ \mathbf{x} \in X$
Sort X in ascending order of norms.
$k \leftarrow 1$
While $k \leq n$
  Solve P2(k)
  If P2(k) is infeasible, **then** $X^E \leftarrow X^E \cup \{\mathbf{x}^k\}$
  else if $z_2^*(k) > 1$, **then** $X^E \leftarrow X^E \cup \{\mathbf{x}^k\}$
    else $X^N \leftarrow X^N \cup \{\mathbf{x}^k\}$
  $X^U \leftarrow X^U - \{\mathbf{x}^k\}$
  $k \leftarrow k + 1$

Figure 2.1 Base Algorithm – Serial Version.

Each time a point is found to be non-extreme, the constraint matrix is reduced by one column. In problems having a substantial number of non-extreme points this can significantly affect the time required to process all n points. This time can be further reduced if it were possible to eliminate some of the non-extreme points early in the process. Our attempt at this is to first translate the points so that the origin is the barycenter of the points. A quick sort is used to place the elements of X in ascending order according to their norms. The points are then processed in this order. Since the points are centered around the origin, points with small norms are somewhat more likely to be non-extreme. By considering these points early in the process, there is a potential for eliminating them from much of the computation.

2.1.2 Parallel Case

Our parallel version of the base algorithm, shown in Figure 2.2, employs a dynamic-scheduling and data partitioning approach. After the norms of all points are computed concurrently, a serial quick sort is performed to place the points in ascending order of norm. Instances of problem P2 are then assigned to processes on a first-come, first-served basis. The only synchronization required is in this selection of points from the set X^U. Therefore, the processes can run almost independently. Furthermore, since the points are not partitioned among the processes in a predetermined manner, each will continue work until all of the points have been assigned for processing.

2.2 Preprocessor and Quadratic Programming-Based Approach

2.2.1 Serial Case

Dulá, Helgason, and Hickman (1992) designed and implemented a serial algorithm for the convex hull problem that utilized preprocessors and quadratic program-

ming techniques. Three preprocessors were developed based on geometric concepts and are used to identify some of the extreme points. The algorithm then proceeds to a second phase in which the remaining points are classified using a specialized quadratic programming approach.

```
base_parallel(X)

Assumption: Elements of X have been translated so that the barycenter
            of the points is the origin.
Input Parameters
        X           {set of points to be examined}
Shared
        k           {index of elements in X; initial value is 0}
        n           {number of elements in X}
Private
        α           {current index of element in X currently being}
                    {processed by a particular process}
        z*_2(k)     {objective function value}

    Compute ||x|| ∀ x ∈ X
    Sort X in ascending order of norms.
    Lock
        k ← k + 1
        α ← k
    Unlock
    While α ≤ n
        Solve P2(α)
        If P2(α) is infeasible, then X^E ← X^E ∪ {x^α}
        else if z*_2(α) > 1, then X^E ← X^E ∪ {x^α}
             else X^N ← X^N ∪ {x^α}
        X^U ← X^U − {x^α}
        Lock
             k ← k + 1
             α ← k
        Unlock
    }
```

Figure 2.2 Base Algorithm – Parallel Version.

The three preprocessing schemes are independent and all work with the original set X. Therefore, a particular point may be identified as extreme by more than one preprocessor. The preprocessors are designed to identify extreme points, but are incapable of classifying points as non-extreme.

Preprocessor 1, shown in Figure 2.3, is based on an idea presented in Preparata and Shamos (1985 - Section 4.1.2) and involves searching along each of the d axes for outliers. For each dimension $i = 1, \ldots, d$, the procedure searches all points in X for the point(s) with maximum value and the point(s) with minimum value in that component. If the maximum (minimum) value occurs at a unique point, then that point is extreme. Also, if exactly two points have the maximum (minimum) value, then both points are extreme. If more than two points are tied for maximum (minimum) value, other coordinates of the points participating in the tie must be examined to find an extreme value in another coordinate. In the best case, as many as $4d$ extreme points – two points having minimum value and two points having maximum value for each dimension – can be identified in preprocessor 1. In the worst case, only two points will be identified.

In preprocessor 2, shown in Figure 2.4, for each point $\mathbf{x} \in X$, we find the point(s) in X farthest away from $\mathbf{x}$. All points that are the maximum distance from $\mathbf{x}$ are extreme (proof in Dulá, Helgason, and Hickman 1992).

preprocessor_1_serial($X, start\_dim, stop\_dim$)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

X	{set of points to examine}
$start\_dim$	{first dimension to be examined}
$stop\_dim$	{last dimension to be examined}

Variables

k	{index of elements in X}
c	{index of coordinate}
X^{MAX}	{set of points tied for maximum}
X^{MIN}	{set of points tied for minimum}
α	{maximum coordinate value}
β	{minimum coordinate value}

$c \leftarrow start\_dim + 1$
While $c \leq stop\_dim$
  $\alpha = \max_{\mathbf{x} \in X} x_c$
  $\beta = \min_{\mathbf{x} \in X} x_c$
  $X^{MAX} = \{\mathbf{x} \mid x_c = \alpha\}$
  $X^{MIN} = \{\mathbf{x} \mid x_c = \beta\}$
  If $|X^{MAX}| > 2$, **then**
    preprocessor_1_serial $(X^{MAX}, c, c+1)$
  else
    $X^E \leftarrow X^E \cup X^{MAX}$
    $X^U \leftarrow X^U - X^{MAX}$
  If $|X^{MIN}| > 2$, **then**
    preprocessor_1_serial $(X^{MIN}, c, c+1)$
  else
    $X^E \leftarrow X^E \cup X^{MIN}$
    $X^U \leftarrow X^U - X^{MIN}$
  $c \leftarrow c + 1$

Figure 2.3 Preprocessor 1 Algorithm – Serial Version.

```
preprocessor_2_serial(X)

Assumption: Elements of X have been translated so that the barycenter
            of the points is the origin.
Input Parameters
        X           {set of points to examine}
Variables
        k           {index of elements in X}
        j           {index of elements in X}
        n           {number of elements in X}
        α           {maximum distance value}
        X^MAX       {set of points tied for maximum}

    k ← 1
    While k ≤ n
        α ← max_{j=1,...,n, j≠k} d(x^k, x^j)
        X^MAX ← {x | d(x^k, x) = α}
        X^E ← X^E ∪ X^MAX
        X^U ← X^U − X^MAX
        k ← k + 1
```

Figure 2.4 Preprocessor 2 Algorithm – Serial Version.

Preprocessor 3, shown in Figure 2.5, is a generalization of preprocessor 1. In preprocessor 1, extreme points along each axis were identified. This is equivalent to maximizing (minimizing) in the direction of a unit vector for each dimension. In preprocessor 3, different directions of optimization are explored. Specifically, each point in X is used to define a hyperplane, or a direction of optimization. The hyperplane for each point $\mathbf{x} \in X$, the hyperplane normal to $\mathbf{x}$ is evaluated at each point $\mathbf{y} \in X$ by computing the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$. If the maximum (minimum) value occurs at exactly one or two points, those points are extreme. However, if the maximum (minimum) occurs at more than two points, a tie-breaking procedure must be invoked to determine which of the points involved in the tie are actually

extreme. Dulá, Helgason, and Hickman (1992) show that this tie-breaking procedure is actually another (smaller) convex hull problem and therefore, any or all of the preprocessors can be invoked to resolve the tie.

```
preprocessor_3_serial(X)

Assumption: Elements of X have been translated so that the barycenter
            of the points is the origin.
Input Parameters
        X           {set of points to examine}
Variables
        k           {index of elements in X}
        n           {number of elements in X}
        α           {maximum coordinate value}
        β           {minimum coordinate value}
        X^MAX       {set of points tied for maximum}
        X^MIN       {set of points tied for minimum}

    k ← 1.
    While k ≤ n
        α ← max_{j=1,···,n} ⟨x^k, x^j⟩
        β ← min_{j=1,···,n} ⟨x^k, x^j⟩
        X^MAX = {x^j | ⟨x^k, x^j⟩ = α}
        X^MIN = {x^j | ⟨x^k, x^j⟩ = β}
        If |X^MAX| > 2, then
            preprocessor_1_serial(X^MAX, 0, 1)
        else
            X^E ← X^E ∪ X^MAX
            X^U ← X^U − X^MAX
        If |X^MIN| > 2, then
            preprocessor_1_serial(X^MIN, 0, 1)
        else
            X^E ← X^E ∪ X^MIN
            X^U ← X^U − X^MIN
        k ← k + 1.
```

Figure 2.5 Preprocessor 3 Algorithm – Serial Version.

After execution of the three preprocessors, some (at least two) extreme points will have been identified. These points are transferred from X^U to X^E. The set

X^N remains empty at this point and the second phase is entered to conclusively determine the status of the remaining points in X^U.

In this phase a point $\mathbf{x} \in X^U$ is projected onto $ConvX^E$. If $\mathbf{x} \in ConvX^E$, $\mathbf{x}$ is not extreme and is moved from X^U into X^N. If $\mathbf{x} \notin ConvX^E$, the projection $\mathbf{p}$ of $\mathbf{x}$ onto $ConvX^E$ defines a hyperplane with normal $\mathbf{x} - \mathbf{p}$ that separates $\mathbf{x}$ from $ConvX^E$. A variation of preprocessor three is then implemented using this hyperplane. We search for the point(s) in X at which this hyperplane takes on minimum and maximum values, i.e., $\{\mathbf{y} \mid \langle \mathbf{y}, \mathbf{x} - \mathbf{p} \rangle \geq \langle \mathbf{z}, \mathbf{x} - \mathbf{p} \rangle \ \forall \ \mathbf{z} \in X\}$ and $\{\mathbf{y} \mid \langle \mathbf{y}, \mathbf{x} - \mathbf{p} \rangle \leq \langle \mathbf{z}, \mathbf{x} - \mathbf{p} \rangle \ \forall \ \mathbf{z} \in X\}$ At least one point whose status is unknown possibly $\mathbf{x}$, will be identified in the first set. Other points may be identified in the second set, but it is possible that they may have already been classified as extreme. The distinguishing feature of this phase is that at each iteration (projection) at least one point is transfered out of X^U – if $\mathbf{x}$ is contained within the convex hull of known extreme points it is moved to X^U; otherwise one or more extreme points are identified using the variation of preprocessor 3 and moved to X^E.

In the algorithm described in Dulá, Helgason, and Hickman (1992) the projection of a point $\mathbf{x}$ onto $ConvX^E$ is formulated as a "minimum distance" problem, i.e., minimize the distance from $\mathbf{x}$ to $ConvX^E$. This is, of course, a quadratic program. A specialized version of the Frank - Wolfe algorithm (Frank and Wolfe 1956) is used to solve this problem – finding both the minimum distance and the projection point.

```
frank_wolfe_serial(X)

Assumption: Elements of X have been translated so that the barycenter
            of the points is the origin.
Input Parameters
        X          {set of points to examine}
Variables
        k          {index of elements in X^U}
        j          {index of elements in X}
        n          {number of elements in X}
        a^k        { normal to hyperplane separating x^k and ConvX^E}
        X^MAX      {set of points tied for maximum}

    k ← 1
    While k ≤ n
        Compute the projection p^k of x^k onto ConvX^E
        If p^k = x^k, then
            X^N ← X^N ∪ {x^k}
            X^U ← X^U − {x^k}
        else
            a^k = x^k − p^k
            α ← max_{j=1,...,n} ⟨x^j, a^k⟩
            X^MAX ← {x^j | ⟨x^j, a^k⟩ = α}
            If |X^MAX| > 2, then
                preprocessor_1_serial(X^MAX, 0, 1)
            else
                X^E ← X^E ∪ X^MAX
                X^U ← X^U − X^MAX
        If x^k ∉ X^U, then k ← k + 1
```

Figure 2.6 Specialized Frank-Wolfe and Variation of Preprocessor 3 Algorithm - Serial Version.

Our implementation of this algorithm, described in Figures 2.3 through 2.6, differed from the implementation of Dulá, Helgason, and Hickman (1992) in one significant area. Ties between more than two points pose problems in both preprocessor 1 and preprocessor 3. In our implementation the procedures were implemented recursively so that when such a tie is encountered, the points involved in the tie were output to the appropriate preprocessor for resolution. The preprocessors could be

repeatedly invoked until the tie was resolved. In the implementation of Dulá, Helgason, and Hickman (1992), only a single-stage tie resolution was attempted. If unsuccessful, the preprocessor was abandoned with no points leaving X^U.

2.2.2 Parallel Case

Our parallel implementation of the preprocessor and quadratic-programming-based algorithm (described in Figures 2.7 - 2.10) exploits both data parallelism and control parallelism in a dynamic scheduling approach. No other parallel implementation of this algorithm is extant. Control parallelism was used to allow different processors to execute the three preprocessors and the Frank-Wolfe procedure concurrently. Since the preprocessors are independent of each other, different processors can be working on each at the same time. Also, as long as at least one extreme point has been identified, the specialized Frank-Wolfe procedure can be executed in parallel with any of the preprocessors. Data parallelism was implemented within each of the preprocessors and in the quadratic programming procedure by partitioning the points among the processes.

preprocessor_1_parallel($X, start\_dim, stop\_dim, p$)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

X	{set of points to examine}
p	{partition size}
$start\_dim$	{first dimension to be examined}
$stop\_dim$	{last dimension to be examined}

Shared

k	{index of elements in X}
c	{index of coordinate}
d	{number of dimensions}

Private

μ	{current index of coordinate being processed}
θ	{end of local partition}
X^{MAX}	{set of points tied for maximum}
X^{MIN}	{set of points tied for minimum}
α	{maximum coordinate value}
β	{minimum coordinate value}

Lock

$\mu \leftarrow start\_dim + 1$

$c \leftarrow start\_dim + p$

Unlock

$\theta = min\{\mu + p, stop\_dim\}$

While $\mu \leq \theta$

$\alpha = \max_{\mathbf{x} \in X} x_\mu$

$\beta = \min_{\mathbf{x} \in X} x_\mu$

$X^{MAX} = \{\mathbf{x} \mid x_\mu = \alpha\}$

$X^{MIN} = \{\mathbf{x} \mid x_\mu = \beta\}$

If $|X^{MAX}| > 2$, **then**

preprocessor_1_serial$(X^{MAX}, \mu + 1, 1)$

else

$X^E \leftarrow X^E \cup X^{MAX}$

$X^U \leftarrow X^U - X^{MAX}$

If $|X^{MIN}| > 2$, **then**

preprocessor_1_serial$(X^{MIN}, \mu + 1, 1)$

else

$X^E \leftarrow X^E \cup X^{MIN}$

$X^U \leftarrow X^U - X^{MIN}$

$\mu \leftarrow \mu + 1$

Figure 2.7 Preprocessor 1 Algorithm – Parallel Version.

preprocessor_2_parallel(X)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

X	{set of points to examine}

Shared

k	{index of elements in X}
j	{index of elements in X}
n	{number of elements in X}

Private

μ	{current index of coordinate being processed}
α	{maximum distance value}
X^{MAX}	{set of points tied for maximum}

Lock
  $k \leftarrow k + 1$
  $\mu \leftarrow k$
Unlock
While $\mu \leq n$
  $\alpha \leftarrow \max_{\substack{j=1,\ldots,n \\ j \neq \mu}} d(\mathbf{x}^{\mu}, \mathbf{x}^{\mathbf{j}})$
  $X^{MAX} \leftarrow \{\mathbf{x} \mid d(\mathbf{x}^{\mu}, \mathbf{x}) = \alpha\}$
  $X^{E} \leftarrow X^{E} \cup X^{MAX}$
  $X^{U} \leftarrow X^{U} - X^{MAX}$
  Lock
    $k \leftarrow k + 1$
    $\mu \leftarrow k$
  Unlock

Figure 2.8 Preprocessor 2 Algorithm – Parallel Version.

preprocessor_3_parallel(X)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

X	{set of points to examine}

Shared

k	{index of elements in X}
j	{index of elements in X}
n	{number of elements in X}

Private

μ	{current index of coordinate being processed}
α	{maximum coordinate value}
β	{minimum coordinate value}
X^{MAX}	{set of points tied for maximum}
X^{MIN}	{set of points tied for minimum}

Lock
$k \leftarrow k + 1$
$\mu \leftarrow k$
Unlock
While $\mu \leq n$
$\alpha \leftarrow \max_{j=1,\cdots,n} \langle \mathbf{x}^{\mu}, \mathbf{x}^{\mathbf{j}} \rangle$
$\beta \leftarrow \min_{j=1,\cdots,n} \langle \mathbf{x}^{\mu}, \mathbf{x}^{\mathbf{j}} \rangle$
$X^{MAX} = \{\mathbf{x}^{\mathbf{j}} \mid \langle \mathbf{x}^{\mu}, \mathbf{x}^{\mathbf{j}} \rangle = \alpha\}$
$X^{MIN} = \{\mathbf{x}^{\mathbf{j}} \mid \langle \mathbf{x}^{\mathbf{m}}\mathbf{u}, \mathbf{x}^{\mathbf{j}} \rangle = \beta\}$
If $|X^{MAX}| > 2$, **then**
preprocessor_3_serial(X^{MAX})
else
$X^{E} \leftarrow X^{E} \cup X^{MAX}$
$X^{U} \leftarrow X^{U} - X^{MAX}$
If $|X^{MIN}| > 2$, **then**
preprocessor_3_serial(X^{MIN})
else
$X^{E} \leftarrow X^{E} \cup X^{MIN}$
$X^{U} \leftarrow X^{U} - X^{MIN}$
Lock
$k \leftarrow k + 1$
$\mu \leftarrow k$
Unlock

Figure 2.9 Preprocessor 3 Algorithm – Parallel Version.

```
frank_wolfe_parallel(X)

Assumption: Elements of X have been translated so that the barycenter
            of the points is the origin.
Input Parameters
        X           {set of points to examine}
Shared
        k           { index of elements in X^U; initial value is 0}
        j           { index of elements in X}
        n           { number of elements in X^U}
Private
        β           { current index of coordinate being processed}
        a^k         { normal to hyperplane separating x^k and ConvX^E}
        X^MAX       {set of points tied for maximum}

    Lock
        k ← k + 1
        β ← k
    Unlock
    While β ≤ n
        Compute the projection p^β of x^β onto ConvX^E
        If p^β = x^β, then
            X^N ← X^N ∪ {x^β}
            X^U ← X^U − {x^β}
        else
            a^β = x^β − p^β
            α ← max_{j=1,...,n} ⟨x^j, a^β⟩
            X^MAX ← {x^j | ⟨x^j, a^β⟩ = α}
            If |X^MAX| > 2, then
                preprocessor_1_serial(X^MAX, 0, 1)
            else
                X^E ← X^E ∪ X^MAX
                X^U ← X^U − X^MAX
        If x^β ∉ X^U, then
            Lock
                k ← k + 1
                β ← k
            Unlock
```

Figure 2.10 Specialized Frank-Wolfe and Variation of Preprocessor 3 Algorithm – Parallel Version.

2.3 Two-Stage Linear Programming-Based Algorithm

2.3.1 Serial Case

Rosen, Xue and Phillips (1991) developed both a serial and a parallel algorithm based on linear programming techniques to solve the convex hull problem. Each of the algorithms consists of two major stages. In the first stage, an attempt is made to quickly identify some of the non-extreme points by solving relatively small linear programs. In the second stage, the status of each point remaining in X^U is determined by solving larger linear programs.

The linear programs solved in each of the two stages are essentially the same as in the base algorithm. The difference is that the convexity constraint is added and therefore, instead of examining the objective function value, one only needs check feasibility. In the first stage, the columns of the constraint matrix are elements of a set S which is a subset of X. S serves as a holding area for points that have been examined, but not classified, in stage one. Initially $S = \emptyset$. The constraints of the stage one problem are of the form:

$$\text{P3}(k): \qquad \sum_{\substack{i:x^i \in S \\ i \neq k}} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.7}$$

$$\sum_{\substack{i:x^i \in S \\ i \neq k}} \lambda_i = 1 \tag{2.8}$$

$$\lambda_i \geq 0, \qquad \forall i : x^i \in S. \tag{2.9}$$

If P3(k) is feasible, then the point $\mathbf{x^k}$ is indeed a convex combination of other elements of S (and X) and is therefore not extreme. If P3(k) is infeasible, no categorization of $\mathbf{x^k}$ may occur during this stage and the set S is expanded to include $\mathbf{x^k}$. Therefore, $\mathbf{x^k}$ will be a column in the constraint matrix for the remainder of the

first stage, but its classification as extreme or non-extreme will not occur until the second stage. As we see, the first stage is an attempt to identify non-extreme points quickly, based on the solution of smaller linear programs than those solved in the base algorithm.

Points unclassified at the end of stage one are passed onto stage two in which a problem of size $d \times (|S| - 1)$ is solved for each point in S. The constraints of the stage two problem are given below.

$$\text{P4}(k): \qquad \sum_{\substack{i:x^i \in S \\ i \neq k}} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.10}$$

$$\sum_{\substack{i:x^i \in S \\ i \neq k}} \lambda_i = 1 \tag{2.11}$$

$$\lambda_i \geq 0, \qquad \forall i : x^i \in S. \tag{2.12}$$

In an attempt to identify a large portion of the non-extreme points during stage one, Rosen, Xue, and Phillips (1991) first sort the points in *descending* order of distance from the center of the smallest rectangle containing X. In this manner, S is likely to become populated with potential extreme points early in stage one, therefore increasing the probability that non-extreme points will be identified in stage one and thereby eliminated from further consideration.

Clearly the performance of this algorithm is dependent on the number of points in X that are extreme. In the worst case, all points are extreme and $2n$ linear programs must be solved – n in stage one and n in stage two. If a small portion of points are extreme, and in particular, if those points can be found early in stage one, the algorithm should perform much better than the base algorithm. Although more linear programs will be solved, they will be significantly smaller.

We have made improvements to this algorithm in two areas. This revised algorithm is given in Figure 2.11. One improvement is in stage two. In this stage, as in the base algorithm, when a point is found to be non-extreme, it may be eliminated from subsequent problems since we know it will never be basic (Ali 1993). The revised formulation of P3 and P4 are given in problems P5 and P6, respectively.

```
two_stage_serial(X)

Input Parameters
        X       {set of points to be examined}
Variables
        k       {index of elements in X}
        l       {index of elements in S}
        n       {number of elements in X}
        m       {number of elements in S}
        d       {dimension of ponts in X}

    preprocessor_1_serial(X, 0, d)
    Compute ||x|| ∀ x ∈ X
    Sort X in descending order of norms.
    k ← 1
    S ← X^E
    While k ≤ n
        If x^k ∉ S, then
            Solve P5(k)
            If P5(k) is infeasible, then S ← S ∪ {x^k}
            else X^N ← X^N ∪ {x^k}
        k ← k + 1
    l ← 1
    While l ≤ m
        If x^l ∈ S, then
            Solve P6(l).
            If P6(l) is infeasible, then X^E ← X^E ∪ {x^l}
            else X^N ← X^N ∪ {x^l}
             X^U ← X^U − {x^l}
        l ← l + 1.
```

Figure 2.11 Improved Two-stage Linear Programming-based Algorithm – Serial Version.

$$\text{P5}(k): \qquad \sum_{\substack{i:x^i \in S \\ i \neq k \\ \mathbf{x^i} \notin X^N}} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.13}$$

$$\sum_{\substack{i:x^i \in S \\ i \neq k \\ \mathbf{x^i} \notin X^N}} \lambda_i = 1 \tag{2.14}$$

$$\lambda_i \geq 0, \qquad \forall i : x^i \in S. \tag{2.15}$$

$$\text{P6}(k): \qquad \sum_{\substack{i:x^i \in S \\ i \neq k \\ \mathbf{x^i} \notin X^N}} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.16}$$

$$\sum_{\substack{i:x^i \in S \\ i \neq k \\ \mathbf{x^i} \notin X^N}} \lambda_i = 1 \tag{2.17}$$

$$\lambda_i \geq 0, \qquad \forall i : x^i \in S. \tag{2.18}$$

The second improvement is in stage one. A critical indicator of the algorithm's performance is early identification of candidate extreme points in stage one. The procedure detailed in Rosen, Xue, and Phillips (1991) for ordering the points is a good attempt at identifying likely extreme points quickly so that a majority of non-extreme points will be eliminated in stage one. However, to determine this ordering, one must compute the center of the smallest rectangle containing all the points. This requires that for each dimension, one must determine the point(s) having minimum value and the point(s) having maximum value in the coordinate. The maximum and minimum values are then averaged to determine the corresponding coordinate of the center of the rectangle. Recall, however, that this identification of points having maximum or minimum coordinate values is precisely what preprocessor 1 of the preprocessor and quadratic programming approach is designed to do since

some of these points are themselves extreme. Rather than ignoring this information, in our implementation we moved these points into the set S before entering stage one. In this manner it is possible to start stage one with up to $4d$ extreme points already identified, Thus not only eliminating the need to solve linear programs in stage one for those points, but also potentially reducing the number of candidate extreme points carried into stage two.

2.3.2 Parallel Case

The parallel implementation reported by Rosen, Xue, and Phillips (1991) was done on an NCUBE/7 hypercube with 64 processors and distributed memory. It is based on a data partitioning approach wherein each concurrently executing process is assigned a unique instance of problem P3 in stage one or P4 during stage two. A static scheduling approach is used to assign problem instances to processes. The elements of X are distributed as evenly as possible among the processors. Each processor is responsible for determining the candidates for stage two from its subset of points. After every 10 iterations, each processor shares its set of candidates with the other processors. In this manner it is hoped that more non-extreme points will be eliminated in stage one.

Our parallel implementation (see Figure 2.12) differs from that of Rosen, Xue and Phillips in two significant areas in addition to incorporating the distinguishing features of our serial implementation described previously. Instead of using a static scheduling approach, we utilized a monitor which assigns unique instances of problem P5 or P6 on a first-come, first-served basis. In this way, all processes remain busy until all instances have been assigned, thus minimizing process idle time.

The second difference is in the sharing of candidates with other processes. In our implementation, any time a candidate is found, this information is noted in

```
two_stage_parallel(X)

Input Parameters
        X       {set of points to be examined}
Shared
        k       {index of elements in X; initial value is 0}
        l       {index of elements in S; initial value is 0}
        n       {number of elements in X}
        m       {number of elements in S}
        d       {dimension of ponts in X}
        p       {partition size for preprocessor 1}
Private
        α       {index of element in X currently being processed}
        β       {index of element in S currently being processed}

    _parallel(X, 0, d, p)
    Compute ||x|| ∀ x ∈ X
    Sort X in descending order of norms.
    S ← X^E
    Lock
        k ← k + 1
        α ← k
    Unlock
    While α ≤ n
        If x^α ∉ S, then
            Solve P5(α)
            If P5(α) is infeasible, then S ← S ∪ {x^α}
            Else X^N ← X^N ∪ {x^α}
        Lock
            k ← k + 1
            α ← k
        Unlock
    Lock
        l ← l + 1
        β ← l
    Unlock
    While β ≤ m
        If x^β is not extreme, then {
            Solve P4(β)
            If P4(β) is infeasible, then X^E ← X^E ∪ {x^β}
            Else X^N ← X^N ∪ {x^β}
            X^U ← X^U − {x^β}
        Lock
            l ← l + 1
            β ← l
        Unlock
```

Figure 2.12 Improved Two-stage Linear Programming-based Algorithm – Parallel Version.

shared memory immediately and thus is instantly accessible by all other processes. By sharing this information without delay, the number of candidates identified for stage two processing may be significantly reduced.

This type of communication was also key in the second stage. In our parallel implementation (as in the serial), when a candidate in stage two is found to be non-extreme, it is never again included in the constraint matrix of the linear program assigned to any process. In the implementation of Rosen, Xue, and Phillips, this elimination does not occur, thus lengthening execution time.

2.4 Expanding-Hull Linear Programming-Based Approach

2.4.1 Serial Case

Dulá and Helgason (1993) presented a new algorithm for the convex hull problem that is based on the ideas behind the preprocessor and quadratic programming approach described previously. Instead of preprocessors, a single linear program is solved initially to identify some of the extreme points. And instead of a quadratic program, a linear program is solved to determine if each $\mathbf{x} \in X^U$ is interior to the convex hull of known extreme points. If so, $\mathbf{x}$ it is not extreme; otherwise a hyperplane separating $\mathbf{x}$ from $ConvX^E$ is found and used in a variant of preprocessor three to find one or more extreme points. In this manner, as in the preprocessor and quadratic programming-based approach, the convex hull of known extreme points is expanded until all points are categorized. A major advantage of this algorithm is that the linear programs that must be solved (except for the initial one) never have more columns than extreme points of X.

In the quadratic programming-based algorithm, preprocessors were used to identify at least two extreme points. These extreme points defined the initial polytope used in the quadratic programming stage. In the new algorithm of Dulá and Helgason

(1993), at least $d+1$ extreme points must be identified so that the initial set is guaranteed to positively span $\Re^d$. One means of accomplishing this is to solve a $d \times (n-1)$ linear program as follows. First some point $\mathbf{x^l}$ such that $\|\mathbf{x^l}\|^2 \geq \|\mathbf{x^j}\|^2 \ \forall \ j = 1, \ldots, n$ must be identified. Such a point $\mathbf{x^l}$ is necessarily extreme. Then the linear program P7(l) must be solved.

$$\text{P7}(k): \qquad \text{Minimize} \qquad \sum_{\substack{i=1 \\ i \neq k}}^{n} \lambda_i \tag{2.19}$$

$$\text{subject to} \qquad \sum_{\substack{i=1 \\ i \neq k}}^{n} \lambda_i \mathbf{x^i} = -\mathbf{x^k} \tag{2.20}$$

$$\lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{2.21}$$

Dulá and Helgason (1993) show that if the solution to P7 is unique, the basic columns at optimality are all extreme points. These basic columns, along with $\mathbf{x^l}$ provide an initial set of $d+1$ extreme points. (If the solution to P7(l) is not unique, one may solve P7(k) for points $\mathbf{x^k} \neq \mathbf{x^l}$, until one is found which generates a unique optimal basis.)

Given an initial set of at least $d+1$ known extreme points X^E we then seek to determine whether a given point $\mathbf{x^k} \in X^U$ is contained within $ConvX^E$. This may be done by solving the linear program P8(k).

$$\text{P8}(k): \ z_8(k) = \text{Minimize} \qquad \sum_{i:\mathbf{x^i} \in X^E} \lambda_i \tag{2.22}$$

$$\text{subject to} \qquad \sum_{i:\mathbf{x^i} \in X^E} \lambda_i \mathbf{x^i} = \mathbf{x^k} \tag{2.23}$$

$$\lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{2.24}$$

Dulá and Helgason (1993) proved the following result with regard to P8(k): If $z_8^*(k)$ is the optimal objective value for some $\mathbf{x^k} \neq 0$ then

1. $z_8^*(k) < 1$ if and only if $\mathbf{x^k}$ is interior to $ConvX^E$;
2. $z_8^*(k) = 1$ if and only if $\mathbf{x^k}$ is on the boundary of $ConvX^E$;
3. $z_8^*(k) > 1$ if and only if $\mathbf{x^k}$ is exterior to $ConvX^E$.

Therefore if $z_8^*(k) \leq 1$ then $\mathbf{x^k}$ is contained within the convex hull of X^E and thus is not extreme. Otherwise we seek to identify a hyperplane that separates $\mathbf{x^k}$ from $ConvX^E$. Such a hyperplane was immediately available as a result of the Frank-Wolfe procedure. Dulá and Helgason (1993) show how to find such a hyperplane from the information generated in solving P8(k). We must first examine the dual of P8(k) which is given in P9(k).

P9(k):	Maximize	$\boldsymbol{\pi}\mathbf{x^k}$		(2.25)
	subject to	$\boldsymbol{\pi}\mathbf{x^i} \leq 1$	$\mathbf{x^i} \in X^E$	(2.26)
		$\lambda_i \geq 0,$	$i = 1, \ldots, n.$	(2.27)

Given optimal solutions $\boldsymbol{\lambda}^*$ and $\boldsymbol{\pi}^*$ to the primal and dual problems, respectively, by dual feasibility we know that $\boldsymbol{\pi}^*\mathbf{x^i} \leq 1 \ \forall \ \mathbf{x^i} \in X^E$. Clearly then any point $\mathbf{q} \in ConvX^E$ also satisfies $\boldsymbol{\pi}^*\mathbf{q} \leq 1$. Also since $\mathbf{x^k} \notin ConvX^E, z_8^*(k) > 1$ and therefore $\boldsymbol{\pi}^*\mathbf{x^k} > 1$. So we have $\boldsymbol{\pi}^*\mathbf{q} \leq 1 \ \forall \ \mathbf{q} \in ConvX^E$ and $\boldsymbol{\pi}^*\mathbf{x^k} > 1$. Therefore $H(\boldsymbol{\pi}^*, 1)$ must separate $\mathbf{x^k}$ and $ConvX^E$.

Once a separating hyperplane is found, the variant of preprocessor 3 implemented in the preprocessors and quadratic programming-based approach may be executed to identify at least one extreme point.

Our implementation of this algorithm (shown in Figure 2.13) differed from that described in Dulá and Helgason (1993) in two areas. Instead of solving a $d \times (n-1)$

expanding_hull_serial(X)

Assumption: Elements of X have been translated so that the barycenter of the points is the origin.

Input Parameters

X	{set of points to examine}

Variables

k	{index of elements in X}
n	{number of elements in X}
d	{dimension of ponts in X}
$z_8^*(k)$	{objective function value}
α	{maximum coordinate value}
β	{minimum coordinate value}
$\boldsymbol{\pi}^*$	{Optimal solution to P9(k)}
X^{MAX}	{set of points tied for maximum}
X^{MIN}	{set of points tied for minimum}

preprocessor_1_serial($X, 0, d$)
Compute $\|\mathbf{x}\|$ $\forall$ $\mathbf{x} \in X$
Sort X in descending order of norms.
$k \leftarrow 1$
While $k \leq n$
  Solve P8(k)
  If $z_8^*(k) \leq 1$, **then**
    $X^N \leftarrow X^N \cup \{\mathbf{x^k}\}$
    $X^U \leftarrow X^U - \{\mathbf{x^k}\}$
  else
    $\alpha = \max_{\mathbf{x^j} \in X^U} \langle \boldsymbol{\pi}^*, \mathbf{x^j} \rangle$
    $\beta = \min_{\mathbf{x^j} \in X^U \cup X^E} \langle \boldsymbol{\pi}^*, \mathbf{x^j} \rangle$
    $X^{MAX} = \{\mathbf{x^j} \mid \langle \boldsymbol{\pi}^*, \mathbf{x^j} \rangle = \alpha\}$
    $X^{MIN} = \{\mathbf{x^j} \mid \langle \boldsymbol{\pi}^*, \mathbf{x^j} \rangle = \beta\}$
    If $|X^{MAX}| > 2$, **then**
      preprocessor_1_serial($X^{MAX}, 0, 1$)
    else
      $X^E \leftarrow X^E \cup X^{MAX}$
      $X^U \leftarrow X^U - X^{MAX}$
    If $|X^{MIN}| > 2$, **then**
      preprocessor_1_serial($X^{MIN}, 0, 1$)
    else
      $X^E \leftarrow X^E \cup X^{MIN}$
      $X^U \leftarrow X^U - X^{MIN}$
  $k \leftarrow k + 1$

Figure 2.13 Expanding-Hull Linear Programming-Based Approach Algorithm – Serial Version.

linear program to identify the initial set of $d+1$ extreme points, we used the preprocessors described previously to quickly identify extreme points. If the preprocessors did not identify at least $d+1$ extreme points then the method employing problem P7 was used.

We also sorted the points in decreasing order of norm before the hull build-up procedure. In this manner, points more likely to be extreme are considered first and therefore the convex hull of the set of known extreme points grows larger early in the process. In this manner many of the points considered later will be identified as non-extreme by problem P8, thus eliminating the need to execute the preprocessor 3 variant in those cases.

2.4.2 Parallel Case

Developed concurrently with our parallel implementation of this algorithm was an implementation by Dulá, Helgason, and Venugopal (1993). The latter implementation was based on the serial implementation of Dulá and Helgason (1993). In addition to the differences between the serial algorithms described in the previous section, the two parallel implementations differ significantly. Our implementation, shown in Figure 2.14, is based on a dynamic-scheduling approach for assigning instances of problem P8 to different processes. The Dulá, Helgason, and Venugopal implementation uses a pre-scheduling approach wherein the points are partitioned among processes apriori. Given p processes, process i is assigned problems P(i), P$(i+p)$, P$(i+2p)$, etc. If a process finishes its allotted work before all other processes are done it starts at the end of the list of points and processes points in reverse order until all points are categorized. No synchronization is done during this phase. This could easily result in multiple processes solving the same instance of problem P8, then concurrently solving another instance of P8, etc. until all points are classified.

expanding_hull_parallel(X)
Assumption: Elements of X have been translated so that the barycenter of the points is the origin.
Input Parameters
X {set of points to examine}
Shared
k {index of elements in X}
d {dimension of points in X}
p {partition size for preprocessor 1}
n {number of elements in X}
Private
μ {current index of element in X being processed}
z_8^* {objective function value}
α {maximum coordinate value}
β {minimum coordinate value}
$\boldsymbol{\pi}^*$ {Optimal solution to P9(k)}
X^{MAX} {set of points tied for maximum}
X^{MIN} {set of points tied for minimum}

preprocessor_1_parallel($X, 0, d, p$)
Compute $\|\mathbf{x}\|$ $\forall$ $\mathbf{x} \in X$
Sort X in descending order of norms.
Lock
$k \leftarrow k+1$, $\mu \leftarrow k$
Unlock
While $\mu \leq n$
Solve P8(μ)
If $z_8^*(\mu) \leq 1$, **then**
$X^N \leftarrow X^N \cup \{\mathbf{x}^\mu\}$, $X^U \leftarrow X^U - \{\mathbf{x}^\mu\}$
else
$\alpha = \max_{\mathbf{x}^j \in X^U} \langle \boldsymbol{\pi}^*, \mathbf{x}^j \rangle$
$\beta = \min_{\mathbf{x}^j \in X^U \cup X^E} \langle \boldsymbol{\pi}^*, \mathbf{x}^j \rangle$
$X^{MAX} = \{\mathbf{x}^j \mid \langle \boldsymbol{\pi}^*, \mathbf{x}^j \rangle = \alpha\}$
$X^{MIN} = \{\mathbf{x}^j \mid \langle \boldsymbol{\pi}^*, \mathbf{x}^j \rangle = \beta\}$
If $|X^{MAX}| > 2$, **then**
preprocessor_1_serial($X^{MAX}, 0, 1$)
else
$X^E \leftarrow X^E \cup X^{MAX}$, $X^U \leftarrow X^U - X^{MAX}$
If $|X^{MIN}| > 2$, **then**
preprocessor_1_serial($X^{MIN}, 0, 1$)
else
$X^E \leftarrow X^E \cup X^{MIN}$, $X^U \leftarrow X^U - X^{MIN}$
Lock
$k \leftarrow k+1$, $\mu \leftarrow k$
Unlock

Figure 2.14 Expanding-Hull Linear Programming-Based Approach Algorithm – Parallel Version.

In our dynamic-scheduled implementation, an efficient monitor is used to assign instances of P8 to processes. All processes work on different problems until all points are processed. Thus minimal overhead is incurred to insure that no redundant work is done.

The other major difference between the two implementations is primarily a result of the different optimizers used to solve the linear programs. While our implementation utilized the callable CPLEX (CPLEX 1993) routines, the implementation of Dulá, Helgason, and Venugopal used XMP (Marsten 1981). We were unable to access CPLEX source code while Dulá, Helgason, and Venugopal did have access to XMP source code. Because of this they were able to allow all processes to share the constraint matrix, thus significantly reducing the amount of memory required to assign work to extra processes. In our implementation each process had a separate copy of all the problem data and used shared memory to communicate a change in status of a point.

2.5 Computational Results

Each of the four algorithms were implemented in serial and parallel on a Sequent Symmetry S-81 shared memory multiprocessor. All of the algorithms being tested were coded in C compiled using the DYNIX C compiler running on DYNIX 3.1.2 the Sequent's proprietary implementation of UNIX. This machine has 26 Intel 80386 processors and 104 Mb of sharable memory. CPLEX (CPLEX 1993) callable library routines which were invoked from the C application programs.

Testing of the implementations was done on two problem sets. The first set, Set A, consists of the three problems reported on by Dulá and Helgason (1993) and Dulá, Helgason, and Venugopal (1993). Each of their implementations was done on a Sequent Symmetry S-81 machine housing 20 Intel 80386 processors and 32 Mb of

sharable memory, so valid comparisons of their reported results with our results may be made. The second set, Set B, consists of problems which were randomly generated on the Sequent Symmetry S-81. In generating these problems the C pseudo random number generator rand() was used. The seeds for this random number generator were chosen from the range 1 to 20. Since the function rand() generates a randomly distributed sequence of integers the expression $(float)(rand()mod656000)/10$ was used to transform the integers into floating point numbers. The characteristics of problems in both sets are given in Table I. Viewing the two sets collectively, the number of points ranges from 334 to 4000 and the number of dimensions ranges from 6 to 19. The extreme point density varied from 32% to 100%.

In attempting to solve these problems with our implementations of each of the algorithms, we found that the preprocessors and quadratic programming approach was not competitive with the other three algorithms. The preprocessors work very well and find a number of extreme points. However, the Frank-Wolfe method for computing the projection of a point onto a convex hull suffers from numerical difficulties – in particular when a point is very near the perimeter of the hull, convergence is extremely slow. None of the Set B problems could be solved in a reasonable amount of time, so no timings will be presented for this algorithm.

Each problem in Sets A and B was solved using both the serial and the parallel implementations of each of the algorithms with the number of processors varying from two to 25. The problems from Set A were each run 3 times and the times were averaged. Due to the long run times for some of the problems in Set B we were able to run each of them only once. The resulting run times for each of the problems using each algorithm (base algorithm (BASE), two-stage linear programming-based algorithm (LP2S), and hull-expansion linear programming-based approach (LPHE)

are shown in Tables II through VI. Included in the Set A times in Table II are the times reported by Dulá and Helgason (1993) and Dulá, Helgason, and Venugopal (1993) in the column denoted DH/DHV. The associated speedups for the Set A problems and some of the Set B problems are given in Tables VII through XI and are illustrated in Figures 2.15 through 2.21. These speedups were computed based on the fastest of the serial times of all of the algorithms.

An examination of this data reveals several things. It is clear that our implementation of the hull-expansion linear programming-based algorithm is superior to all other implementations. In fact, as the problems grow larger the speedup of LPHE approaches linear. An examination of LP2S versus the base algorithm performance is also interesting. When the extreme point density is low, LP2S is superior to BASE. However, when the extreme point density is high, the opposite is true. In the low-density problems, LP2S solves approximately 1.3 more linear programs than BASE, whereas in the high-density problems, LP2S solves over 1.9 more linear programs than BASE. This is reasonable because the more extreme points there are, the more candidate points will be input to stage two of the LP2S algorithm. Therefore as the density increases, the number of linear programs that must be solved in LP2S approaches $2n$ while BASE always solves n linear programs in all cases. In the low density case, even though LP2S still solves more than n linear programs, those that are solved are significantly smaller than those solved using BASE, thus leading to better solution times. Problem density did not seem to affect the number of linear programs solved in LPHE as this number remained very close to n on all problems.

Also of interest is the fact that the LPHE times are significantly faster than those of the corresponding implementation by Dulá and Helgason (1993) and Dulá, Helgason, and Venugopal (1993). In fact, even LP2S was superior to DH/DHV

in two of the three Set A problems. In an attempt to determine if the difference in optimizers played a role in the disparate results, we examined the run times of BASE which used CPLEX (1993), with the times for the implementation of the base algorithm reported by Dulá and Helgason (1993) which used XMP (Marsten 1981). Since the work to be done in the base algorithm is constant and involves only solving linear programs, such a comparison should indicate the relative efficiencies of CPLEX and XMP. Our hypothesis was that the much newer optimizer CPLEX would be far superior to the older XMP. The timing comparisons are given in Table XII. Note that, contrary to our hypothesis, BASE actually performed worse than Dulá and Helgason's implementation of the base algorithm using XMP. Therefore our implementations were actually at a disadvantage by using CPLEX.

Another factor that could affect these results is the difference in the amount of primary memory on our system versus that on which the DH/DHV codes were tested. However, since almost all problem data was shared among the processes – in particular the constraint matrix which is by far the largest block of data – the difference in memory capacity should have little, if any, effect on the relatively small Set A problems.

We believe that the runtime disparity is primarily due to the asynchronous nature of the DHV implementation. The self-scheduling approach coupled with the lack of communication among processes provides opportunity for considerable inefficiency since many instances of multiple processes solving the same problem could occur.

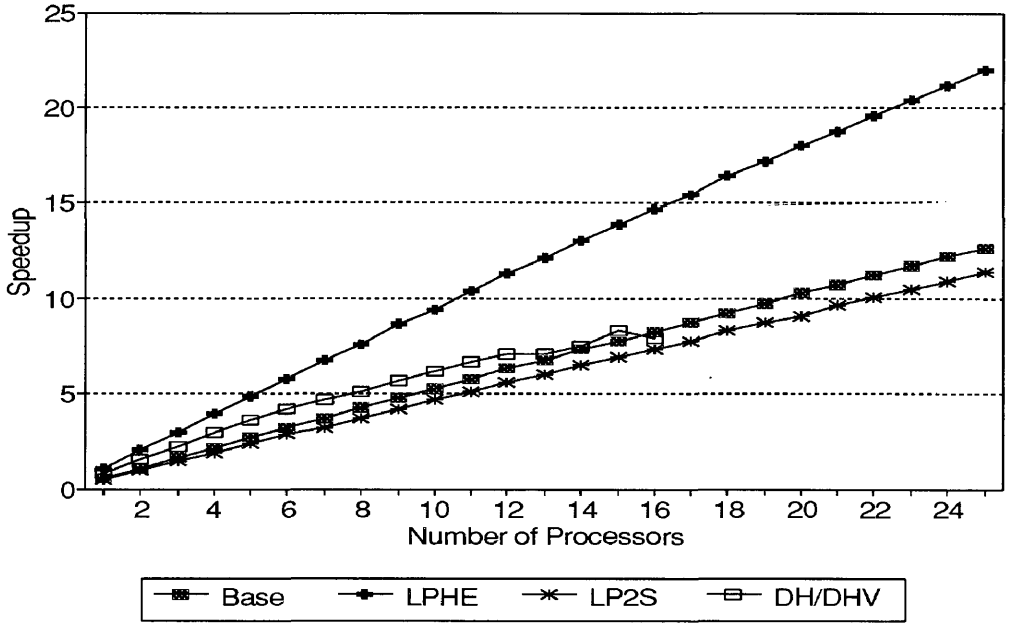

Figure 2.15 Speedup vs. Number of Processors for Set A – 816 Points, 14 Dimensions.

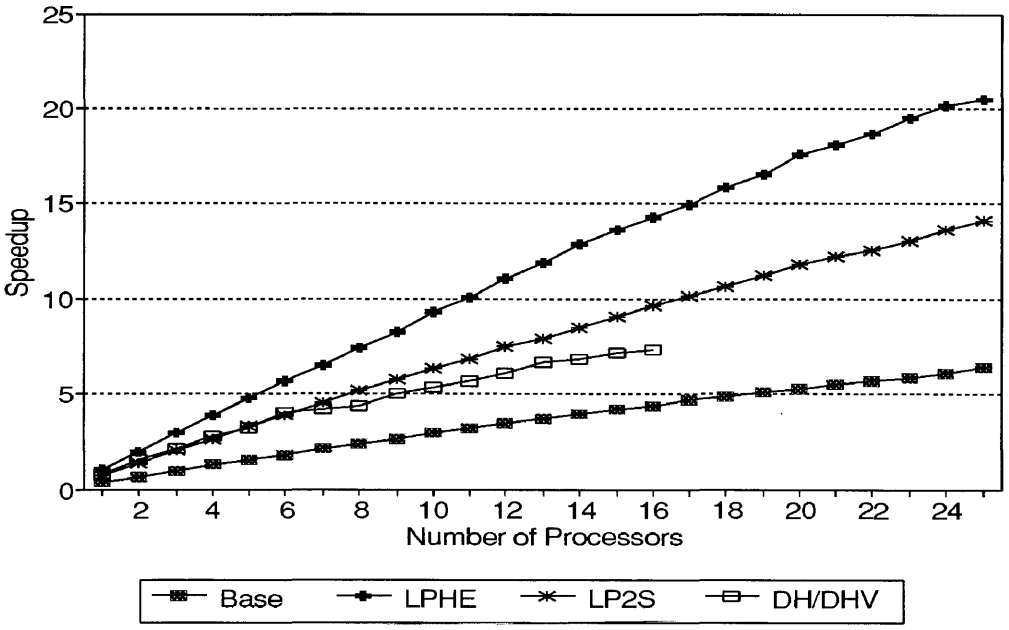

Figure 2.16 Speedup vs. Number of Processors for Set A – 1000 Points, 6 Dimensions.

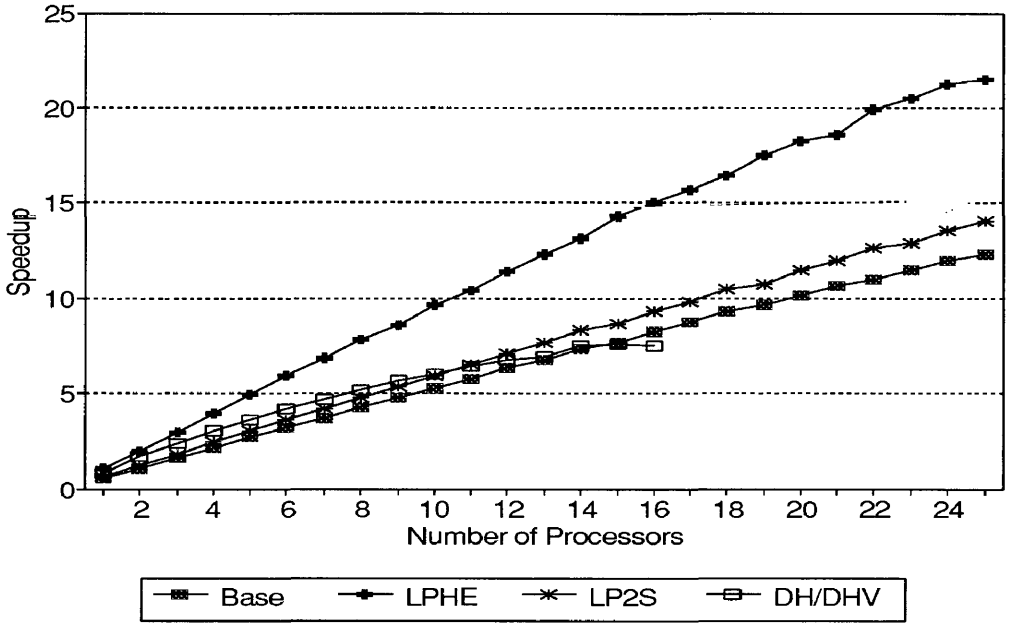

Figure 2.17 Speedup vs. Number of Processors for Set A – 334 Points, 19 Dimensions.

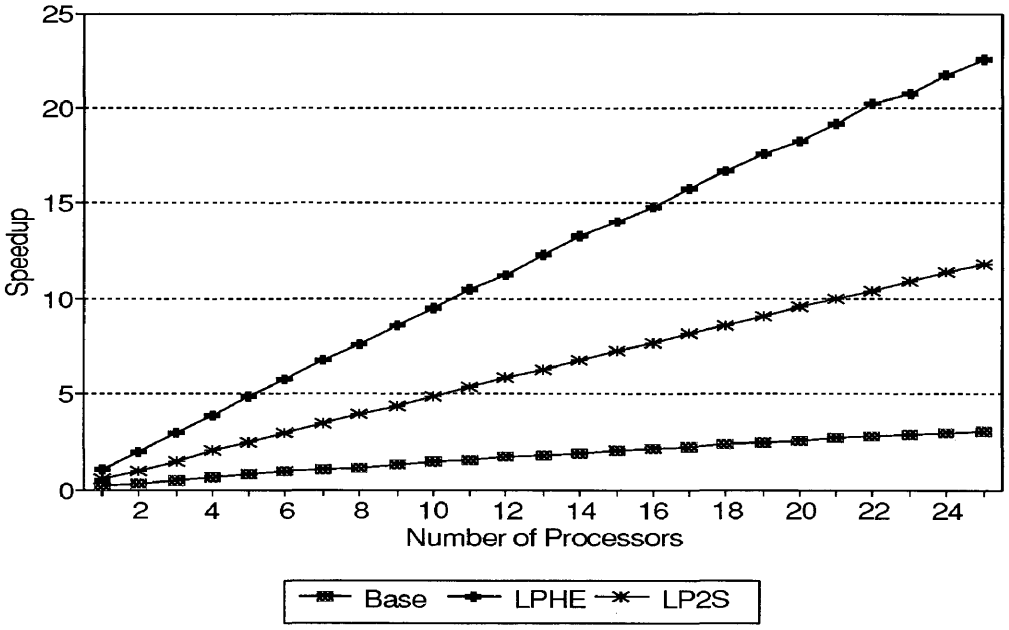

Figure 2.18 Speedup vs. Number of Processors for Set B – 4000 Points, 6 Dimensions.

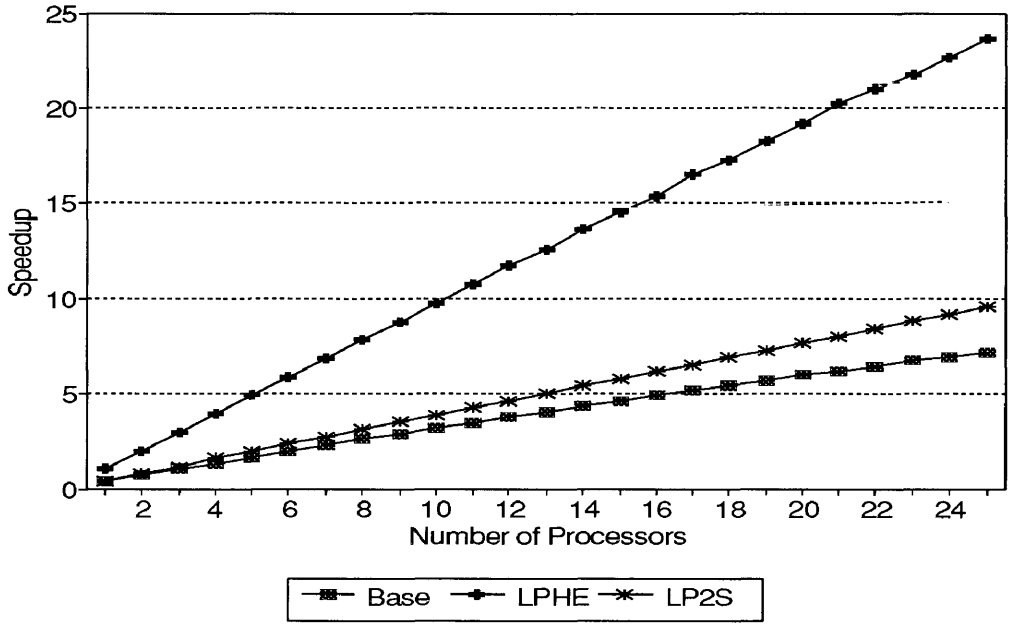

Figure 2.19 Speedup vs. Number of Processors for Set B – 4000 Points, 8 Dimensions.

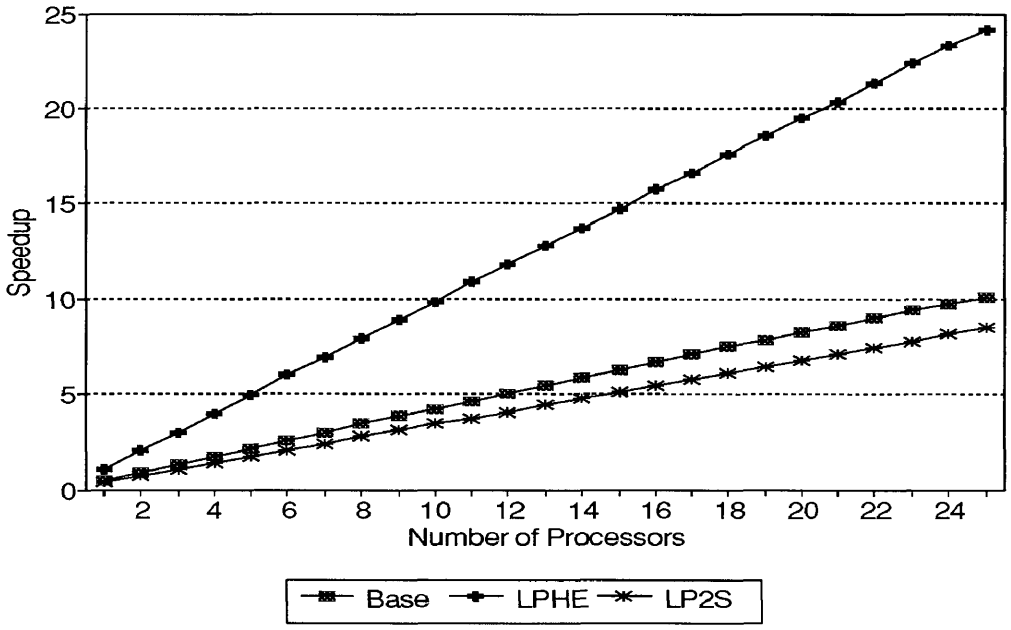

Figure 2.20 Speedup vs. Number of Processors for Set B – 4000 Points, 10 Dimensions.

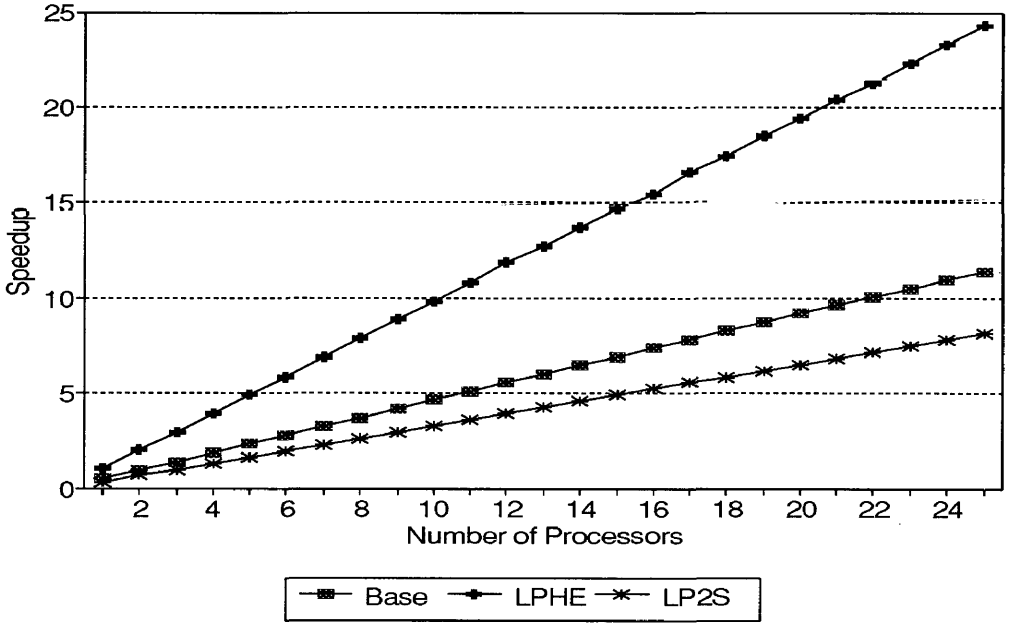

Figure 2.21 Speedup vs. Number of Processors for Set B – 4000 Points, 12 Dimensions.

Table I.--Test Problem Characteristics.

Problem Set	Number of Points	Number of Dimensions	Number of Extreme Points	Extreme Point Density
Set A	816	14	734	90%
	1000	7	340	34%
	334	19	334	100%
Set B	1000	6	484	48%
	2000	6	738	37%
	3000	6	947	32%
	4000	6	1128	38%
	1000	8	765	77%
	2000	8	1358	68%
	3000	8	1805	60%
	4000	8	2272	57%
	1000	10	933	93%
	2000	10	1749	87%
	3000	10	2438	81%
	4000	10	3203	80%
	1000	12	983	98%
	2000	12	1899	95%
	3000	12	2800	93%
	4000	12	3685	92%

Table II.-- Set A – Average Run Times.

# of	816 points, 14 dimensions				1000 points, 7 dimensions				334 points, 19 dimensions			
Processors	BASE	LPHE	LP2S	DH/DHV	BASE	LPHE	LP2S	DH/DHV	BASE	LPHE	LP2S	DH/DHV
Serial	4082.7	2187.0	4739.3	2499.0	1259.3	423.3	624.0	785.0	1697.3	908.7	1516.7	1264.0
2	2053.0	1105.3	2356.0	1231.0	668.0	216.3	315.0	432.0	853.7	465.3	754.3	593.0
3	1370.0	739.0	1556.0	848.0	449.0	145.7	212.0	314.0	571.0	303.7	505.0	426.0
4	1023.3	561.3	1167.7	640.0	339.7	111.0	161.3	234.0	424.0	232.7	379.3	331.0
5	820.7	450.3	941.7	528.0	278.3	88.7	129.3	207.0	341.0	184.7	303.0	272.0
6	687.3	377.7	779.0	449.0	232.7	74.3	109.0	160.0	283.7	154.0	252.7	238.0
7	590.3	326.3	676.7	407.0	202.0	64.6	93.7	156.0	245.0	132.3	219.3	212.0
8	516.0	286.7	584.3	371.0	178.7	57.0	82.0	148.0	214.0	116.7	192.3	192.0
9	461.3	252.7	526.0	335.0	161.3	51.3	73.7	130.0	191.7	105.7	170.7	174.0
10	416.3	231.7	469.7	307.0	145.6	45.3	67.0	122.0	173.0	94.7	153.0	165.0
11	379.3	210.0	426.0	283.0	133.3	42.0	62.0	114.0	157.0	87.0	139.7	154.0
12	348.3	193.7	394.0	268.0	123.7	38.3	56.7	107.0	144.7	79.6	128.3	147.0
13	323.3	179.6	364.7	266.0	115.3	35.7	53.3	98.0	135.0	73.7	118.7	143.0
14	299.0	168.0	337.7	251.0	107.3	33.0	50.0	95.0	124.0	69.3	109.7	133.0
15	283.0	157.7	317.7	227.0	102.3	31.0	46.7	91.0	117.7	63.6	105.0	131.0
16	264.7	148.3	298.0	240.0	96.3	29.7	44.0	89.0	110.7	60.3	97.7	132.0
17	249.3	141.3	282.0		91.3	28.3	41.7		103.7	58.0	92.3	
18	236.3	133.0	263.7		87.3	26.7	39.7		97.7	55.3	86.7	
19	223.7	127.0	250.7		83.3	25.7	37.7		94.0	52.0	85.0	
20	213.0	121.7	240.0		80.0	24.0	36.0		89.7	49.7	79.3	
21	204.3	116.3	227.7		77.0	23.3	34.7		85.7	49.0	75.7	
22	194.7	111.7	217.3		74.3	22.7	33.7		82.7	45.7	71.7	
23	187.0	107.0	208.0		72.0	21.7	32.3		79.3	44.3	70.7	
24	179.3	103.3	200.0		70.0	21.0	31.0		75.7	42.6	67.3	
25	173.3	99.3	191.7		67.0	20.7	30.0		74.0	42.3	64.7	

All times in seconds.

Table III.--Set B – Run Times (6 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	1053	361	808	3995	979	1938	8741	1704	3359	15334	2635	5362
2	541	181	401	2201	497	973	4799	867	1681	8369	1353	2683
3	365	120	269	1492	336	658	3263	585	1122	5695	902	1792
4	276	92	203	1138	254	493	2490	441	839	4326	681	1338
5	224	75	163	921	202	392	2010	352	678	3528	543	1080
6	190	63	135	774	172	329	1702	303	566	2953	459	896
7	163	53	116	673	148	281	1469	257	483	2563	392	770
8	146	47	101	596	128	246	1301	226	424	2270	348	675
9	132	43	90	531	115	220	1171	202	379	2035	308	599
10	120	39	82	485	104	198	1060	179	342	1858	278	542
11	110	35	75	443	96	182	979	166	310	1707	252	495
12	102	32	69	412	88	166	905	152	286	1581	234	453
13	95	30	64	386	81	154	846	141	265	1476	214	421
14	89	29	60	362	75	143	793	132	244	1394	193	391
15	84	26	56	341	71	135	751	124	230	1308	183	364
16	80	25	53	322	67	127	708	117	216	1235	173	342
17	76	25	50	307	64	120	673	111	203	1181	167	324
18	73	23	47	293	60	113	645	105	194	1126	153	306
19	69	21	45	280	57	107	613	100	183	1075	150	290
20	67	21	43	270	55	102	591	95	176	1035	144	277
21	64	20	41	259	52	98	568	90	166	1002	137	264
22	63	20	39	249	51	94	548	87	159	958	130	254
23	60	19	38	240	49	90	529	83	154	927	127	242
24	59	18	36	234	47	86	517	81	147	902	121	232
25	57	18	35	226	45	83	498	80	142	874	117	224

All times in seconds.

Table IV.--Set B - Run Times (8 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	Base	LPHE	LP2S	Base	LPHE	LP2S	Base	LPHE	LP2S	Base	LPHE	LP2S
Serial	1894	825	2279	7519	2827	7929	15435	5399	14450	26224	8826	22816
2	953	425	1141	3746	1435	3967	7755	2757	7197	13262	4502	11314
3	636	284	761	2509	954	2658	5204	1814	4796	8920	3014	7608
4	476	216	572	1892	722	1987	3921	1373	3610	6721	2247	5702
5	382	173	456	1512	578	1587	3154	1096	2880	5415	1794	4569
6	324	145	382	1269	481	1318	2639	920	2399	4535	1508	3811
7	275	123	328	1098	413	1132	2277	796	2058	3900	1289	3269
8	244	108	290	965	364	988	2002	685	1802	3427	1131	2846
9	217	97	258	863	325	885	1788	614	1604	3089	1006	2537
10	196	87	230	780	293	797	1632	552	1442	2782	906	2286
11	179	81	210	715	265	726	1482	506	1316	2554	825	2081
12	165	73	194	659	243	665	1364	464	1201	2348	754	1907
13	154	67	179	608	226	615	1274	429	1111	2177	702	1759
14	144	63	166	569	210	571	1187	398	1037	2045	647	1633
15	136	59	158	531	195	531	1108	376	967	1922	608	1526
16	127	57	148	501	186	502	1048	352	909	1809	575	1435
17	120	53	139	476	175	471	992	332	856	1713	535	1351
18	114	51	131	454	167	444	945	312	809	1623	510	1280
19	109	48	124	430	158	423	903	298	765	1549	484	1209
20	104	47	119	411	151	401	861	286	729	1484	460	1148
21	100	44	113	394	143	385	823	272	695	1420	436	1098
22	97	42	108	378	138	365	796	260	665	1375	420	1048
23	92	42	103	365	133	351	763	250	632	1318	405	1002
24	89	40	100	348	127	336	734	241	608	1270	389	964
25	86	38	97	338	122	324	708	233	588	1232	373	923

All times in seconds.

Table V.--Set B – Run Times (10 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	2973	1389	4204	11981	5252	16351	24739	10708	31816	42729	18353	54032
2	1485	701	2101	6006	2652	8159	12447	5421	15824	21385	9244	26968
3	1005	469	1396	4009	1784	5420	8281	3603	10554	14258	6183	18007
4	746	355	1050	3010	1340	4081	6222	2717	7931	10774	4656	13481
5	599	284	841	2423	1073	3256	4976	2160	6337	8657	3711	10793
6	498	236	701	2019	895	2715	4180	1818	5278	7202	3078	9018
7	430	203	604	1732	765	2335	3584	1544	4579	6173	2661	7712
8	378	178	525	1518	671	2030	3150	1359	3991	5436	2331	6740
9	336	160	467	1347	595	1808	2812	1206	3537	4823	2070	6008
10	302	143	423	1212	539	1619	2537	1086	3174	4362	1865	5387
11	277	132	386	1103	488	1479	2313	991	2891	3968	1684	4907
12	253	120	352	1019	447	1365	2114	909	2647	3662	1553	4505
13	233	113	327	940	414	1255	1952	840	2453	3383	1432	4160
14	219	104	303	874	387	1168	1821	783	2277	3144	1334	3860
15	204	99	283	818	362	1088	1706	735	2135	2941	1251	3608
16	193	93	265	770	342	1016	1597	694	1994	2759	1169	3383
17	181	87	250	723	323	970	1510	655	1877	2605	1103	3196
18	172	82	237	690	303	913	1434	616	1772	2465	1043	3027
19	163	78	224	654	290	859	1359	585	1679	2351	988	2858
20	155	75	214	623	273	817	1295	555	1592	2234	941	2712
21	149	72	204	594	264	782	1238	530	1520	2143	902	2589
22	143	69	194	568	250	744	1185	510	1452	2040	858	2467
23	137	66	186	547	239	720	1132	489	1386	1957	819	2363
24	132	63	180	521	233	680	1092	467	1323	1885	786	2259
25	127	62	173	504	221	654	1053	450	1278	1815	761	2174

All times in seconds.

Table VI.--Set B – Run Times (12 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	4136	2010	5867	16754	7711	24056	35980	16601	51429	61444	28718	88557
2	2074	1003	2920	8367	3873	11979	17981	8479	25755	30600	14605	43842
3	1381	656	1947	5619	2585	7953	12304	5636	17057	20543	9652	29318
4	1038	493	1454	4204	1927	5963	8986	4218	12896	15410	7243	22022
5	835	393	1172	3356	1545	4797	7201	3374	10286	12287	5793	17590
6	698	328	971	2792	1291	4012	6025	2839	8552	10279	4874	14626
7	597	289	834	2401	1106	3408	5170	2429	7380	8828	4164	12614
8	520	249	735	2106	960	2989	4540	2120	6426	7727	3649	10960
9	466	223	653	1871	860	2657	4036	1878	5713	6867	3235	9752
10	420	205	579	1679	778	2391	3651	1696	5155	6192	2926	8783
11	385	183	532	1537	699	2171	3294	1546	4704	5633	2647	7968
12	351	171	486	1414	643	1999	3034	1403	4312	5165	2417	7315
13	328	158	455	1323	603	1843	2793	1305	3959	4773	2255	6735
14	302	144	424	1209	557	1702	2605	1211	3683	4429	2096	6274
15	285	134	393	1135	519	1602	2426	1135	3439	4157	1952	5857
16	268	129	369	1063	494	1507	2278	1071	3244	3899	1854	5509
17	253	119	348	1002	462	1413	2150	1009	3047	3663	1724	5166
18	237	115	329	948	440	1330	2028	957	2874	3466	1646	4878
19	227	109	309	890	411	1271	1922	906	2731	3296	1550	4655
20	216	104	295	855	395	1206	1835	863	2594	3129	1479	4405
21	206	101	283	818	374	1147	1749	826	2465	2981	1404	4198
22	197	95	270	777	361	1093	1673	792	2360	2847	1351	4018
23	187	91	257	746	343	1056	1606	754	2252	2733	1288	3821
24	182	89	247	715	330	1006	1533	721	2176	2618	1233	3685
25	175	85	237	693	319	969	1468	694	2082	2514	1182	3522

All times in seconds.

Table VII.--Set A – Average Speedups.

# of	816 points, 14 dimensions				1000 points, 7 dimensions				334 points, 19 dimensions			
Processors	BASE	LPHE	LP2S	DH/DHV	BASE	LPHE	LP2S	DH/DHV	BASE	LPHE	LP2S	DH/DHV
Serial	0.54	1.00	0.46	0.76	0.34	1.00	0.68	0.83	0.54	1.00	0.60	0.79
2	1.07	1.98	0.93	1.54	0.63	1.96	1.34	1.51	1.06	1.95	1.21	1.68
3	1.60	2.96	1.41	2.23	0.94	2.91	2.00	2.07	1.59	2.99	1.80	2.33
4	2.14	3.90	1.87	2.95	1.25	3.81	2.62	2.79	2.14	3.91	2.40	3.00
5	2.67	4.86	2.32	3.58	1.52	4.77	3.27	3.14	2.67	4.92	3.00	3.65
6	3.18	5.79	2.81	4.21	1.82	5.70	3.88	4.07	3.20	5.90	3.60	4.18
7	3.71	6.70	3.23	4.64	2.10	6.55	4.52	4.16	3.71	6.87	4.14	4.68
8	4.24	7.63	3.74	5.10	2.37	7.43	5.16	4.39	4.25	7.79	4.73	5.16
9	4.74	8.66	4.16	5.65	2.62	8.25	5.75	5.01	4.74	8.60	5.32	5.70
10	5.25	9.44	4.66	6.16	2.91	9.34	6.32	5.32	5.25	9.60	5.94	6.03
11	5.77	10.41	5.13	6.68	3.18	10.08	6.84	5.69	5.79	10.44	6.51	6.45
12	6.28	11.29	5.55	7.06	3.42	11.04	7.47	6.07	6.28	11.41	7.08	6.76
13	6.76	12.17	6.00	7.11	3.67	11.87	7.94	6.67	6.73	12.33	7.66	6.93
14	7.31	13.02	6.48	7.52	3.94	12.83	8.47	6.85	7.33	13.11	8.29	7.47
15	7.73	13.87	6.89	8.33	4.14	13.66	9.07	7.18	7.72	14.27	8.65	7.56
16	8.26	14.74	7.34	7.88	4.40	14.27	9.62	7.31	8.21	15.06	9.30	7.50
17	8.77	15.47	7.76		4.64	14.94	10.16		8.77	15.67	9.84	
18	9.25	16.44	8.29		4.85	15.87	10.67		9.30	16.42	10.48	
19	9.78	17.22	8.73		5.08	16.49	11.24		9.67	17.47	10.69	
20	10.27	17.98	9.11		5.29	17.64	11.76		10.13	18.29	11.45	
21	10.70	18.80	9.61		5.50	18.15	12.21		10.61	18.54	12.01	
22	11.23	19.58	10.06		5.70	18.67	12.57		10.99	19.90	12.68	
23	11.70	20.44	10.51		5.88	19.54	13.09		11.45	20.50	12.86	
24	12.20	21.17	10.94		6.05	20.16	13.66		12.01	21.30	13.50	
25	12.62	22.02	11.41		6.38	20.48	14.11		12.28	21.47	14.05	

All times in seconds.

Table VIII.--Set B – Speedups (6 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	0.34	1.00	0.45	0.25	1.00	0.51	0.20	1.00	0.51	0.17	1.00	0.49
2	0.67	1.99	0.90	0.45	1.97	1.01	0.36	1.97	1.01	0.32	1.95	0.98
3	0.99	3.01	1.34	0.66	2.91	1.49	0.52	2.91	1.52	0.46	2.92	1.47
4	1.31	3.92	1.78	0.86	3.85	1.99	0.68	3.86	2.03	0.61	3.87	1.97
5	1.61	4.81	2.22	1.06	4.85	2.50	0.85	4.84	2.51	0.75	4.85	2.44
6	1.90	5.73	2.67	1.27	5.69	2.98	1.00	5.62	3.01	0.89	5.74	2.94
7	2.22	6.81	3.11	1.46	6.62	3.48	1.16	6.63	3.53	1.03	6.72	3.42
8	2.47	7.68	3.57	1.64	7.65	3.98	1.31	7.54	4.02	1.16	7.57	3.90
9	2.74	8.40	4.01	1.84	8.51	4.45	1.46	8.44	4.50	1.30	8.56	4.40
10	3.01	9.26	4.40	2.02	9.41	4.94	1.61	9.52	4.98	1.42	9.48	4.86
11	3.28	10.31	4.81	2.21	10.20	5.38	1.74	10.27	5.50	1.54	10.46	5.32
12	3.54	11.28	5.23	2.38	11.13	5.90	1.88	11.21	5.96	1.67	11.26	5.82
13	3.80	12.03	5.64	2.54	12.09	6.36	2.01	12.09	6.43	1.79	12.31	6.26
14	4.06	12.45	6.02	2.70	13.05	6.85	2.15	12.91	6.98	1.89	13.31	6.74
15	4.30	13.89	6.45	2.87	13.79	7.25	2.27	13.74	7.41	2.02	14.02	7.24
16	4.51	14.44	6.81	3.04	14.61	7.71	2.41	14.56	7.89	2.13	14.80	7.71
17	4.75	14.44	7.22	3.19	15.30	8.16	2.53	15.35	8.39	2.23	15.78	8.13
18	4.95	15.71	7.68	3.34	16.32	8.66	2.64	16.23	8.78	2.34	16.68	8.61
19	5.23	17.19	8.02	3.50	17.18	9.15	2.78	17.04	9.31	2.45	17.57	9.09
20	5.39	17.19	8.40	3.63	17.80	9.60	2.88	17.94	9.68	2.55	18.30	9.51
21	5.64	18.05	8.81	3.78	18.83	9.99	3.00	18.93	10.27	2.63	19.23	9.98
22	5.73	18.05	9.26	3.93	19.20	10.42	3.11	19.59	10.72	2.75	20.27	10.37
23	6.02	19.00	9.50	4.08	19.98	10.88	3.22	20.53	11.07	2.84	20.75	10.89
24	6.12	20.06	10.03	4.18	20.83	11.38	3.30	21.04	11.59	2.92	21.78	11.36
25	6.33	20.06	10.31	4.33	21.76	11.80	3.42	21.30	12.00	3.02	22.52	11.76

All times in seconds.

Table IX.--Set B – Speedups (8 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	Base	LPHE	LP2S	Base	LPHE	LP2S	Base	LPHE	LP2S	Base	LPHE	LP2S
Serial	0.44	1.00	0.36	0.38	1.00	0.36	0.35	1.00	0.37	0.34	1.00	0.39
2	0.87	1.94	0.72	0.76	1.97	0.71	0.70	1.96	0.75	0.67	1.96	0.78
3	1.30	2.91	1.08	1.13	2.96	1.06	1.04	2.98	1.13	0.99	2.93	1.16
4	1.73	3.82	1.44	1.49	3.92	1.42	1.38	3.93	1.50	1.31	3.93	1.55
5	2.16	4.77	1.81	1.87	4.89	1.78	1.71	4.93	1.88	1.63	4.92	1.93
6	2.55	5.69	2.16	2.23	5.88	2.15	2.05	5.87	2.25	1.95	5.85	2.32
7	3.00	6.71	2.52	2.58	6.85	2.50	2.37	6.78	2.62	2.26	6.85	2.70
8	3.38	7.64	2.85	2.93	7.77	2.86	2.70	7.88	3.00	2.58	7.80	3.10
9	3.80	8.51	3.20	3.28	8.70	3.19	3.02	8.79	3.37	2.86	8.77	3.48
10	4.21	9.48	3.59	3.62	9.65	3.55	3.31	9.78	3.74	3.17	9.74	3.86
11	4.61	10.19	3.93	3.95	10.67	3.89	3.64	10.67	4.10	3.46	10.70	4.24
12	5.00	11.30	4.25	4.29	11.63	4.25	3.96	11.64	4.50	3.76	11.71	4.63
13	5.36	12.31	4.61	4.65	12.51	4.60	4.24	12.59	4.86	4.05	12.57	5.02
14	5.73	13.10	4.97	4.97	13.46	4.95	4.55	13.57	5.21	4.32	13.64	5.41
15	6.07	13.98	5.22	5.32	14.50	5.32	4.87	14.36	5.58	4.59	14.52	5.78
16	6.50	14.47	5.57	5.64	15.20	5.63	5.15	15.34	5.94	4.88	15.35	6.15
17	6.88	15.57	5.94	5.94	16.15	6.00	5.44	16.26	6.31	5.15	16.50	6.53
18	7.24	16.18	6.30	6.23	16.93	6.37	5.71	17.30	6.67	5.44	17.32	6.90
19	7.57	17.19	6.65	6.57	17.89	6.68	5.98	18.12	7.06	5.70	18.24	7.30
20	7.93	17.55	6.93	6.88	18.72	7.05	6.27	18.88	7.41	5.95	19.19	7.70
21	8.25	18.75	7.30	7.18	19.77	7.34	6.56	19.85	7.77	6.22	20.24	8.04
22	8.51	19.64	7.64	7.48	20.49	7.75	6.78	20.77	8.12	6.42	21.01	8.42
23	8.97	19.64	8.01	7.75	21.26	8.05	7.08	21.60	8.54	6.70	21.79	8.81
24	9.27	20.63	8.25	8.12	22.26	8.41	7.36	22.40	8.88	6.95	22.69	9.16
25	9.59	21.71	8.51	8.36	23.17	8.73	7.63	23.17	9.18	7.16	23.66	9.56

All times in seconds.

Table X.--Set B – Speedups (10 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	0.47	1.00	0.33	0.44	1.00	0.32	0.43	1.00	0.34	0.43	1.00	0.34
2	0.94	1.98	0.66	0.87	1.98	0.64	0.86	1.98	0.68	0.86	1.99	0.68
3	1.38	2.96	1.00	1.31	2.94	0.97	1.29	2.97	1.02	1.29	2.97	1.02
4	1.86	3.91	1.32	1.75	3.92	1.29	1.72	3.94	1.35	1.70	3.94	1.36
5	2.32	4.89	1.65	2.17	4.90	1.61	2.15	4.96	1.69	2.12	4.95	1.70
6	2.79	5.89	1.98	2.60	5.87	1.93	2.56	5.89	2.03	2.55	5.96	2.04
7	3.23	6.84	2.30	3.03	6.87	2.25	2.99	6.94	2.34	2.97	6.90	2.38
8	3.68	7.80	2.65	3.46	7.83	2.59	3.40	7.88	2.68	3.38	7.87	2.72
9	4.13	8.68	2.97	3.90	8.83	2.91	3.81	8.88	3.03	3.81	8.87	3.06
10	4.60	9.71	3.28	4.33	9.74	3.24	4.22	9.86	3.37	4.21	9.84	3.41
11	5.01	10.52	3.60	4.76	10.76	3.55	4.63	10.81	3.70	4.63	10.90	3.74
12	5.49	11.58	3.95	5.15	11.75	3.85	5.07	11.78	4.05	5.01	11.82	4.07
13	5.96	12.29	4.25	5.59	12.69	4.19	5.49	12.75	4.37	5.43	12.82	4.41
14	6.34	13.36	4.58	6.01	13.57	4.50	5.88	13.68	4.70	5.84	13.76	4.76
15	6.81	14.03	4.91	6.42	14.51	4.83	6.28	14.57	5.02	6.24	14.67	5.09
16	7.20	14.94	5.24	6.82	15.36	5.17	6.71	15.43	5.37	6.65	15.70	5.43
17	7.67	15.97	5.56	7.26	16.26	5.41	7.09	16.35	5.71	7.05	16.64	5.74
18	8.08	16.94	5.86	7.61	17.33	5.75	7.47	17.38	6.04	7.45	17.60	6.06
19	8.52	17.81	6.20	8.03	18.11	6.11	7.88	18.30	6.38	7.81	18.58	6.42
20	8.96	18.52	6.49	8.43	19.24	6.43	8.27	19.29	6.73	8.22	19.50	6.77
21	9.32	19.29	6.81	8.84	19.89	6.72	8.65	20.20	7.05	8.56	20.35	7.09
22	9.71	20.13	7.16	9.25	21.01	7.06	9.04	21.00	7.38	9.00	21.39	7.44
23	10.14	21.05	7.47	9.60	21.98	7.29	9.46	21.90	7.73	9.38	22.41	7.77
24	10.52	22.05	7.72	10.08	22.54	7.72	9.81	22.93	8.09	9.74	23.35	8.12
25	10.94	22.40	8.03	10.42	23.77	8.03	10.17	23.80	8.38	10.11	24.12	8.44

All times in seconds.

Table XI .--Set B – Speedups (12 dimensions).

# of	1000 points			2000 points			3000 points			4000 points		
Processors	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S	BASE	LPHE	LP2S
Serial	0.49	1.00	0.34	0.46	1.00	0.32	0.46	1.00	0.32	0.47	1.00	0.32
2	0.97	2.00	0.69	0.92	1.99	0.64	0.92	1.96	0.65	0.94	1.97	0.66
3	1.46	3.06	1.03	1.37	2.98	0.97	1.35	2.95	0.97	1.40	2.98	0.98
4	1.94	4.08	1.38	1.83	4.00	1.29	1.85	3.94	1.29	1.86	3.97	1.30
5	2.41	5.12	1.72	2.30	4.99	1.61	2.31	4.92	1.61	2.34	4.96	1.63
6	2.88	6.13	2.07	2.76	5.97	1.92	2.76	5.85	1.94	2.79	5.89	1.96
7	3.37	6.96	2.41	3.21	6.97	2.26	3.21	6.83	2.25	3.25	6.90	2.28
8	3.87	8.07	2.74	3.66	8.03	2.58	3.66	7.83	2.58	3.72	7.87	2.62
9	4.31	9.01	3.08	4.12	8.97	2.90	4.11	8.84	2.91	4.18	8.88	2.95
10	4.79	9.81	3.47	4.59	9.91	3.23	4.55	9.79	3.22	4.64	9.82	3.27
11	5.22	10.98	3.78	5.02	11.03	3.55	5.04	10.74	3.53	5.10	10.85	3.60
12	5.73	11.75	4.14	5.45	11.99	3.86	5.47	11.83	3.85	5.56	11.88	3.93
13	6.13	12.72	4.42	5.83	12.79	4.18	5.94	12.72	4.19	6.02	12.74	4.26
14	6.66	13.96	4.74	6.38	13.84	4.53	6.37	13.71	4.51	6.48	13.70	4.58
15	7.05	15.00	5.12	6.79	14.86	4.81	6.84	14.63	4.83	6.91	14.71	4.90
16	7.50	15.58	5.45	7.25	15.61	5.12	7.29	15.50	5.12	7.37	15.49	5.21
17	7.95	16.89	5.78	7.70	16.69	5.46	7.72	16.45	5.45	7.84	16.66	5.56
18	8.48	17.48	6.11	8.13	17.53	5.80	8.19	17.35	5.78	8.29	17.45	5.89
19	8.86	18.44	6.51	8.66	18.76	6.07	8.64	18.32	6.08	8.71	18.53	6.17
20	9.31	19.33	6.81	9.02	19.52	6.39	9.05	19.24	6.40	9.18	19.42	6.52
21	9.76	19.90	7.10	9.43	20.62	6.72	9.49	20.10	6.74	9.63	20.45	6.84
22	10.20	21.16	7.44	9.92	21.36	7.06	9.92	20.96	7.03	10.09	21.26	7.15
23	10.75	22.09	7.82	10.34	22.48	7.30	10.34	22.02	7.37	10.51	22.30	7.52
24	11.04	22.58	8.14	10.79	23.37	7.67	10.83	23.03	7.63	10.97	23.29	7.79
25	11.49	23.65	8.48	11.13	24.17	7.96	11.31	23.92	7.97	11.42	24.30	8.15

All times in seconds.

Table XII.--BASE Times.

Problem	DH	BASE
816 points, 14 dimensions	4082.7	3585
1000 points, 7 dimensions	1259.3	954
334 points, 19 dimensions	1697.3	2042

All times in seconds.

CHAPTER 3

SUMMARY AND CONCLUSIONS

The multidimensional convex hull problem arises in many contexts such as linear programming redundancy, data envelopment analysis, and detection of statistical outliers. In the last few years several algorithms to solve this problem – both serial and parallel – have been developed and implemented. However, the disparate nature of the hardware on which the implementations were done made performance comparisons impossible.

In this thesis we described four serial algorithms for solving the multidimensional convex hull problem: (1) a base linear programming-based algorithm; (2) a preprocessor and quadratic programming-based algorithm; (3) a two-stage linear programming-based algorithm; and (4) a hull-expansion and linear programming-based algorithm. Each of these algorithms was improved and implemented. Parallel variants of each algorithm were also developed and implemented on a shared-memory multiprocessor. Extensive computational testing was performed to empirically compare the performance of each implementation.

Results of the computational testing revealed that our serial and parallel implementations of the hull-expansion and linear programming-based approach were clearly superior to our implementations of all other algorithms. Our improvements to this algorithm were also apparent as our implementation substantially out-performed a previous implementation done on the same type machine.

Results also indicated that performance of two-stage linear programming algorithm was affected by the density of the extreme points while the hull-expansion and linear programming-based algorithm and the base algorithm were not. The preprocessor and quadratic programming approach was not competitive due to numerical difficulties in solving the quadratic program. However, the preprocessors were used very successfully in two of the other algorithms.

BIBLIOGRAPHY

Ali, I. "Stream-lined Computation for DEA." *European Journal of Operational Research*, 64 (1993): 61-67.

Barr, R. S., and B. L. Hickman. "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions." *ORSA Journal on Computing*, 5 (Winter 1993): 2-18.

Brown, K. Q. "Voronoi Diagrams From Convex Hulls." *Information Processing Letters*, 9 (1979): 223-228.

Charnes, A., W. W. Cooper and E. Rhodes. "Measuring the Efficiency of Decision Making Units." *European Journal of Operational Research*, 2 (1978): 429-444.

CPLEX. *Using the CPLEX™ Callable Library and CPLEX™ Mixed Integer Library* CPLEX Optimization, Inc., (1993).

Duda, R. O., and P. E. Hart. *Pattern Classification and Scene Analysis.* New York: Wiley-Interscience, 1973.

Dulá, J. H. "Geometry of Optimal Value Functions With Applications to Redundancy in Linear Programming." Dallas, Texas: Southern Methodist University, Department of Computer Science and Engineering, 1991. Technical Report 91-CSE-3.

Dulá, J. H. "Geometry of Polyhedral Cones and Optimal Value Functions With Applications to Redundancy in Linear Programming." Dallas, Texas: Southern Methodist University, Department of Computer Science and Engineering, 1990. Technical Report 90-CSE-33.

Dulá, J. H., and R. V. Helgason. "A New Procedure for Identifying the Frame of the Convex Hull of a Finite Collection of Points in Multidimensional Space." Dallas, Texas: Southern Methodist University, Department of Computer Science and Engineering, 1993. Technical Report 93-CSE-1.

Dulá, J. H., R. V. Helgason, and B. L. Hickman. "Preprocessing Schemes and a Solution Method for the Convex Hull Problem in Multidimensional Space." *Computer Science and Operations Research: New Developments in their Interfaces.* O. Balci, ed., Pergamon Press, U. K., (1992): 59-70.

Dulá, J. H, R. V. Helgason, and N. Venugopal. "A nearly Asynchronous Parallel LP-based Algorithm for the Convex Hull Problem in Multidimensional Space." Dallas, Texas: Southern Methodist University, Department of Computer Science and Engineering, 1993. Technical Report 93-CSE-27.

Edelsbrunner, H. *Algorithms in Combinatorial Geometry.* Berlin: Springer-Verlag, 1987.

Flynn, M.J. "Very High-Speed Computing Systems." *Proceedings of the IEEE*, 54 (1966): 1901-1909.

Frank, M., and P. Wolfe. "An Algorithm for Quadratic Programming." *Naval Research Logistics Quarterly*, 3 (1956): 95-110.

Gastwirth, J. "On Robust Procedures." *Journal of the American Statistical Association*, 61 (1966): 929-948.

Graham, R., and F. Yao. "A Whirlwind Tour of Computational Geometry." *The American Mathematical Monthly*, 97 (October 1990): 687-701.

Hoare, C. A. R., "Monitors: An Operating System Structuring Concept." *Communications of the ACM*, 17 (1974): 549-557.

Marsten, R., "The Design of the XMP Linear Programming Library." *ACM Transactions on Mathematical Software*, 7 (1981): 481-497.

Pokorny, C. K., and C. F. Gerald. *Computer Graphics: The Principles Behind the Art and Science.* Irvine, CA: Franklin, Beedle Associates, 1988.

Pratt, W. K. *Digital Image Processing.* New York: Wiley-Interscience, 1991.

Preparata, F. P., and M. I. Shamos. *Computational Geometry : An Introduction.* New York: Springer-Verlag, 1985.

Quinn, Michael J. *Parallel Computing Theory and Practice.* New York: McGraw Hill Book Company, 1994.

Rosen, J. B., G. L. Xue, and A. T. Phillips. "Efficient Computation of Extreme Points of Convex Hulls in $\Re^d$." *Advances in Optimization and Parallel Computing.* P.M. Pardalos ed., (1991): 267-292.

Rosenfeld, A. *Picture Processing by Computer.* New York: Academic Press, 1969.

Sequent, *Guide to Parallel Programming on Sequent Computer Systems.* Beaverton, Oregon: Sequent Computer Systems, 1987.

Wallace, S. W., and R. J. B. Wets. "Preprocessing in Stochastic Programming: The Case of Linear Programs." *ORSA Journal of Computing*, 4 (1992): 45-59.

Wets, R. J. B., and C. Witzgall. "Algorithms for Frames and Lineality Spaces of Cones." *Journal of Research of the National Bureau of Standards - B Mathematics and Mathematical Physics*, 71B (1967): 1-7.