Optimal Golomb Rulers: An Exhaustive Parallel Search Implementation

William T. Rankin (wrankin@ee.duke.edu)

Duke University
Department of Electrical Engineering
P.O. Box 90291
Durham, NC 27708-0291

Master's Thesis submitted to Duke University Department of Electrical Engineering
December 1993

Contents

List of Figures

List of Tables

Abstract

Golomb Rulers are numerical sequences that find application in a variety of engineering fields. The computation of these series, however, is prohibitively costly in terms of computer resources. Thus the discovery of new rulers over the past two decades has been limited by the available computing resources. This thesis describes a parallel implementation of an Optimal Golomb Ruler search algorithm. This algorithm provides a high utilization of available computer resources, and is portable to a variety of computer platforms. Reliability and fault tolerance issues are addressed by algorithm design. With the new algorithm, two previously unpublished rulers were independently discovered and verified.

Chapter 1

Introduction

The term "Golomb Ruler" derived its name from the relevant work by Professor Solomon W. Golomb of the University of Southern California [4]. These rulers are numerical sequences or graphs with application in a wide variety of fields including radio communications, x-ray crystallography, coding theory, and radio astronomy.

The search for Golomb Rulers is a combinatorial algorithm whose bounds can be shown to grow geometrically with respect to the solution size [21]. This has been a major limitation in the discovery of new rulers since each new ruler is by necessity larger that its predecessor. The search is bounded, however, and therefore solvable [13]. By carefully reviewing existing sequential algorithms and applying parallel programming techniques, the problem can be reduced to a workable solution.

This thesis will review the design and implementation of a parallel search algorithm for Golomb Rulers. Chapter 1 introduces the definition of a Golomb Ruler, and presents some of the applications for these sequences. Chapter 2 presents an overview of a number of search algorithms used in finding Golomb Rulers. Chapter 3 provides an analysis of performance and other efficiency metrics for two of the sequential algorithms. Chapter 4 presents a parallel implementation of one of the sequential algorithms and analyzes its performance in a distributed processing environment. Chapter 5 summarizes the results found from the parallel algorithm. Finally, Chapter 6 discusses future directions for work in this area.

1.1 Definitions

It is necessary to define some basic terms used to describe the characteristics of Golomb Rulers. Following this, a formal definition of a Golomb Ruler is given. A more complete discussion of the characteristics of Golomb Rulers is given in Appendix A.

1.1.1 Marks and Length

A Golomb Ruler consists of ordered series of integer numbers [8]. These numbers are referred to as *marks*, and correspond to positions on a linear scale. The difference between the values of any two marks is called the *distance* between those marks. The difference between the largest and smallest number is referred to as the *length* of the ruler, and corresponds to the largest distance for that ruler. The first mark of the series is by convention at position zero.

1.1.2 Difference Triangle

In evaluating Golomb Rulers, it is very helpful to establish a data structure to contain all the distances measured by that ruler. A difference triangle is a grid of numbers where each number

$$
\begin{array}{ccccc}
d1_1 & & d1_2 & & d1_3 \\
 & d2_1 & & d2_2 & \\
 & & d3_1 & &
\end{array}
$$

Figure 1.1: A Difference Triangle

$$
\begin{array}{ccccccc}
0 & & 1 & & 4 & & 6 \\
\hline
 & 1 & & 3 & & 2 & \\
 & & 4 & & 5 & & \\
 & & & 6 & & &
\end{array}
$$

Figure 1.2: A Difference Triangle Example

represents the distance between a specific pair of marks. For a ruler of m marks, there will be $m(m-1)/2$ entries in the difference triangle. An example of a difference triangle is shown in Figure 1.1.

The terms $d1_x$, $d2_x$ and $d3_x$ are the first order, second order and third order differences respectively. These differences are defined by the equations :

$$d1_x = M_{x+1} - M_x \tag{1.1}$$

$$d2_x = d1_x + d1_{x+1} \tag{1.2}$$

$$d3_x = d2_x + d2_{x+1} - d1_{x+1} \tag{1.3}$$

The equations for higher order differences are simply extensions of the third order difference equation. The first order differences are the distances measured between every pair of adjacent marks in the ruler. The second order differences are the distances measured between marks placed two apart on the ruler. A ruler with m marks will have $m-1$ first order differences, $m-2$ second order difference, and so on, up to a single $m-1$ order difference.

Figure 1.2 shows an example of a difference triangle for a simple ruler with four marks. The first row of the figure represents the mark positions for the ruler. The difference triangle elements are located below the horizontal line.

1.1.3 Golomb Ruler

A Golomb Ruler is defined as an ordered numeric sequence whose corresponding difference triangle contains no duplicate elements. In other words, the distance between any two marks in a Golomb Ruler is unique for that ruler. It is important to note that while this definition does not place any restriction on the length of the ruler, the cases that are of interest to researchers are the rulers with minimum lengths.

1.1.4 Perfect Golomb Ruler

A Perfect Golomb Ruler is defined as a Golomb Ruler whose difference triangle contains all numbers between one and the length of the ruler. Since for an N mark ruler, there are $\begin{pmatrix} N \\ 2 \end{pmatrix}$ entries in the difference triangle, the length of an N mark Perfect Golomb Ruler is $(N-1)N/2$.

Proof of Maximum Perfect Ruler

Perfection is very rare among Golomb Rulers. Golomb has presented a simple proof that no Perfect Golomb Rulers exist with more than four marks [4]. This proof is described below:

1. Given an N mark ruler, there are $\binom{N}{2}$ possible distances measured.

2. The overall length of a ruler defined as the maximum length measured by the ruler. A perfect ruler is defined as measuring all distances from one to the maximum length of the ruler. If no distances are omitted from the series then the maximum length of a perfect N mark ruler will be the $(N-1)N/2$.

3. Looking at the distances, the first $(N-1)$ distances sum up to the total allowed length of the ruler
$$\sum_{i=1}^{N-1} i = \frac{(N-1)N}{2}$$
and therefore, these distances must form the first row of the difference triangle.

4. In order to place the distance of length one in the first row of the difference triangle, it must be positioned so that when combined with one of the spaces adjacent to it, it does not form a second order distance less than length N, since all distances from 1 to $N-1$ are already contained in the first row.

5. Therefore the only possible space for the distance one is adjacent to the $N-1$ space, forming a distance of N in the second row of the difference triangle. Since the distance one cannot be adjacent to any other spaces, it must therefore be positioned one one end of the first row.

6. The distance of length two must now be placed in the first row. It cannot be placed next to a distance of less than $N-2$, since that would form a second order distance less than N. Furthermore, it cannot be placed next to an $N-2$ distance, since this would produce a second order distance of N, which already exists.

7. Therefore the only possible position for the distance of length two is adjacent to the $N-1$ space. Like the one distance, the two distance must also be on one end of the first row.

8. Since the one and two length distances must both be adjacent to the $N-1$ space, then they must be located on either side of that space.

9. Since the one and two length distances must also lie on the ends of the ruler, no more first order distances may exist.

10. Thus the Perfect Golomb Ruler can have at most three distances in the first row of the difference triangle, which is produced by a four mark ruler. Rulers of greater than four marks cannot be Perfect, since it will violate one of the above conditions.

1.1.5 Optimum Rulers

Since Perfect Rulers beyond four marks cannot exist, longer rulers are described in terms of whether or not they are Optimum. An Optimum Golomb Ruler is defined as being the shortest length for a given number of marks. As will be shown, there may exist multiple different Optimum Golomb Rulers for a given number of marks.

There also exist what are known as Near-Optimum Rulers which are simply Golomb Rulers whose length is within several units of the corresponding Optimum Ruler.

1.2 Applications

So far, we have established the characteristics of a Golomb Ruler, but we have not indicated the purpose of these rulers beyond that of a mathematical curiosity. In fact, the qualities of a Golomb Ruler that we have established find their application in a wide variety of fields. In this section we will review a selection of papers on the application of Golomb and Near Golomb Rulers to real world applications.

1.2.1 Radio Communications

Perhaps the earliest reference to Golomb Rulers, although not by that name, was in the 1953 article by W. Babcock regarding third and fifth order intermodulation distortion among consecutive channels in a radio band [2]. Babcock explores methods of channel placement to reduce or eliminate third and fifth order distortion. After reviewing several algebraic and random channel distribution methods, Babcock presents the concept of placing each channel within the frequency spectrum on intervals corresponding to the marks on a Golomb Ruler. Third order interference between channels is eliminated through this approach. Fifth order interference may also be substantially reduced [14].

An interesting aside is the fact that Babcock presents a table for Golomb rulers up to ten marks, and all of the rulers up to length eight are optimum. The nine and ten mark rulers that he presents are near optimum.

1.2.2 X-Ray Crystallography

X-ray analysis of crystal structures can produce ambiguities when two different crystal lattice structures produce identical diffraction patterns. Golomb Ruler sequences were used to model the lattice structure and resolve these ambiguities, based upon the empirical observation that no two equal length Golomb Rulers will have the same difference set [3].

Although this observation has been disproved for a single pair of six mark rulers, no counter examples for rulers of greater than six marks has been found, and this approach is still being used. In fact, Professor Golomb has offered an award of one hundred dollars to anyone who can produce a counter-example with greater than six marks [5].

1.2.3 Self-Orthogonal Codes

Golomb Rulers have been used extensively in the field of coding theory for error detection and correction. Sets of rulers are used to generate self-orthogonal codes that play an important role in communications [13][12].

These codes use what is known as a *difference triangle set* (DTS). A DTS consists of a group of rulers all having the same number of marks. A DTS has the additional quality that the difference triangle for each of the members of the set contains values unique to the set. An example of a DTS is shown in Figure 1.3 for the ruler set $\{\{0, 6, 11, 13\}, \{0, 8, 17, 18\}, \{0, 3, 5, 19\}\}$.

$$
\begin{array}{|ccccc|ccccc|ccccc|}
6 & & 11 & & 13 & 8 & & 17 & & 18 & 3 & & 15 & & 19 \\
& 5 & & 7 & & & 9 & & 10 & & & 12 & & 16 & \\
& & 2 & & & & & 1 & & & & & 4 & & \\
\end{array}
$$

Figure 1.3: A Difference Triangle Set

The individual rulers that constitute DTS's are used as parity encoding vectors, with marks of the rulers corresponding to bit positions. No two vectors will contain two bits set in the same bit positions, since to do so would mean that the vectors contained a common difference. Further discussion on DTS's can be found in [20] and [11].

1.2.4 Linear Arrays

Linear antenna and sensor arrays have played an important part in a wide variety of applications, most notably communications and radio astronomy.

Radio Astronomy makes use of Golomb Rulers in the following manner. In locating a distant radio source, it is essential to determine the angle between the incoming wavefront and the antenna array baseline. The antennas are all set to receive the same frequency, and the exact time of reception of the incoming signal can be measured and compared between all pairs of antennas. The direction of the incoming signal can be determined by analyzing the phase difference between the pairs of antennas.

Maximum accuracy is gained when no two antennas are the same distance apart, that is, when all the elements of the their difference triangle are unique. By placing the antennas at the marks of a Golomb Ruler, this condition is guaranteed. Furthermore, by using optimum or near optimum Golomb Rulers, the total length of the array element is minimized. This is an important concern when we consider the geographical distances utilized in radio astronomy, as well as the cost of the individual array elements [16][7].

Another application of Golomb Rulers to radio astronomy is in the selection of scanning frequencies. The purpose of this application is not so much to locate distant sources, as it is the location of the antennas themselves. Using a single pair of antennas, and scanning a set of frequencies from a distant point source the location of the antenna elements can be determined to an accuracy of centimeters across continental distances [4]. By assigning the scanning frequencies to correspond to the marks on a Golomb Ruler, maximum accuracy is guaranteed for a minimum number of scanning frequencies.

1.2.5 PPM Communications

The growing use of optical communications has produced an increased interest in pulse phase modulation (PPM) as a form of communications. In PPM communications, a serial input sequence is divided up into a series of equal length *frames*. Each frame is subdivided into a number of *slots*. Within each frame a single pulse may be placed within one of the slots. The position of the pulse within the frame communicates the information.

As we can see, proper frame synchronization is essential to assure error free communications. Frame synchronization is achieved through comparing the input stream with a stored sequence and generating a correlation factor based upon the summation of the compared inputs.

A desirable PPM sequence maximizes this correlation factor when the frames are synchronized and minimizes the correlation for any off-shifted sequence. Gagliardi, Robbins and Taylor [17] present the case that maximum correlation can be achieved if no spacing between any pair of incoming pulses is repeated for any other pair. This is first requirement of a Golomb Ruler. The authors go on to show how optimum and near optimum rulers maximize the correlation factor while minimizing the overall sequence length.

The correlation of serial pulse trains also finds application in radar coding [3].

1.3 Summary

As we have seen, Golomb Rulers find application in a wide field of subjects. However, while many rulers have been found and documented for these applications, the actual generation and verification of these rulers has not been discussed. In fact, the complexity involved in generating these sequences has prevented the verification of optimum rulers beyond eighteen marks as of September 1993, although only Optimum Golomb Ruler up to sixteen marks have been published [21]. In the following section, we will review the algorithms used to search for optimum and near optimum rulers.

Chapter 2

Algorithms

This chapter will present some of the various algorithms used to generate and verify Optimum Golomb Rulers.

2.1 Scientific American Algorithm

This algorithm was presented in the December 1985 issue of Scientific American [4]. The basic algorithm is composed of two sections, a ruler generation procedure and a Golomb verification procedure. The generation procedure constructs a sequence of rulers each of which are passed to the verification procedure which determines whether or not the sequence meets the criteria of a Golomb Ruler. The following sections describe each routine in turn.

2.1.1 Exhaust

The generation procedure, called Exhaust, requires two parameters. The first parameter is the number of marks contained in the desired Golomb Ruler. The second parameter sets an upper bound on the length of the Golomb Ruler being searched for. If this boundary is set too low, no ruler will be found.

The Exhaust procedure is recursive in nature. A Golomb Ruler candidate is constructed by taking an existing n mark Golomb Ruler and appending a new mark onto the right side of the ruler, generating an $n + 1$ mark ruler. An example of this is shown in Figure 2.1.

The Exhaust procedure does not keep track of the mark positions, but rather the spaces between adjacent marks, which are stored in an array called *SPACES*. These values represent the first row of the difference triangle for that ruler.

Exhaust begins by initializing the first element in *SPACES* to the distance 1. It then proceeds to the next distance in *SPACES* and, starting at a value of 1, increments this value until the distances measured by the first two entries are unique. It then repeats this process for the next element and so on. If at any point the total distance measured by the elements in *SPACES* exceeds the maximum ruler length, then the algorithm will back up one element and increment that element, and add new distances from there.

When the procedure places its last mark and finds a ruler of the target length, it simply prints this information out and continues the search, thus guaranteeing an exhaustive traverse of all possible ruler combinations.

A flowchart of the Exhaust algorithm is shown in Figure 2.2.

The Exhaust procedure is guaranteed to produce all combinations of marks within the search space determined by the number of marks and the limiting length of the ruler being searched for. The major drawback of this algorithm is that it requires a maximum value to limit the search space.

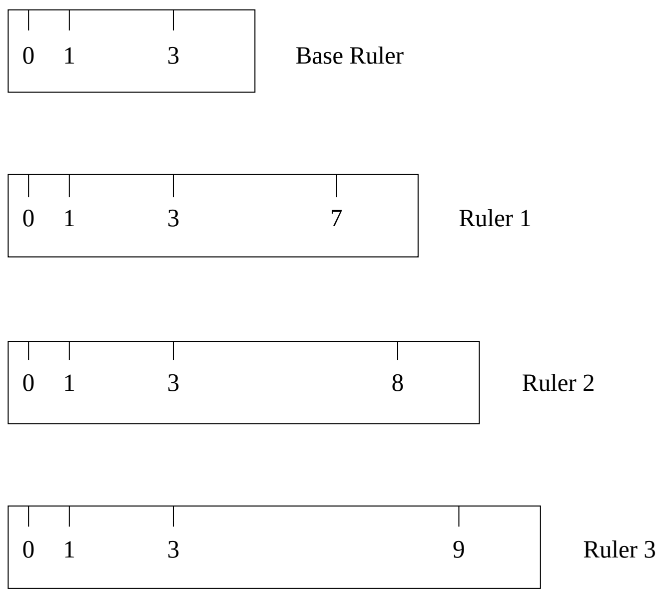

Figure 2.1: Example of Ruler Generation in Scientific American Algorithm

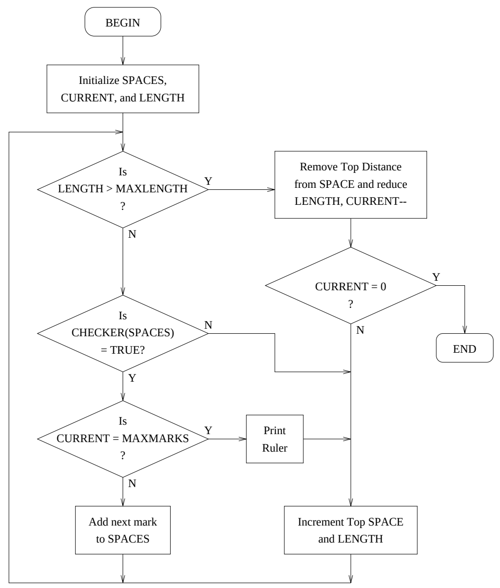

Figure 2.2: Flowchart of Exhaust Algorithm

If this maximum is too low, a ruler will never be found. If it is too high, then large amounts of processing will be wasted in traversing needless search space.

2.1.2 Checker

The ruler verification procedure, Checker, is the routine called by Exhaust to check if a series of marks meet the first requirement for Golomb Rulers.

The body of Checker is two nested loops that will compute every possible distance measures by the first N elements of the $SPACES$ array. It uses the distance computed as an index into an array of boolean values. If the array element indexed by the distance is already set to true, then the distance being checked has already been measured by that set of marks and the sequence is not a Golomb Ruler. The procedure aborts at this point, returning a result of false. If the distance array element is clear, then the procedure sets that element to true and goes on to process the next pair of marks. If there are no conflicts after all the distances have been computed, then Checker returns a value of true.

The drawback of this routine is the fact that for every ruler, Checker must generate the complete set of distances for all the N marks. As we will see later, it is more efficient to compute only those distances added to the ruler by the new mark.

2.2 Token Passing Algorithm

This algorithm was created by Professor Apostolos Dollas at Duke University. Like the Scientific American Algorithm, it consists of two parts. The first part of the code generates a series of potential rulers with the second part checking to see if they are Golomb Rulers.

Unlike the previous algorithms, this one does not build up a ruler by appending new marks on to the end of an existing ruler. Rather, it manipulates all of the specified marks simultaneously, progressing through all possible mark positions until one is found that meets the criteria for a Golomb Ruler.

A flowchart for the for the algorithm is shown in Figures 2.3 and 2.4.

The outer loop of the main routine consists of two procedures. The Iterate procedure generates a potential Golomb Ruler. The Diffncheck procedure checks to see it the proposed ruler meets the criteria for a Golomb Ruler. If Diffncheck returns a true result then an Optimum Golomb Ruler has been found and the routine prints the results and exits. If a ruler is not found, then Iterate is called again to generate the next ruler.

The Iterate subroutine starts with a set of marks, where each mark is at its minimum position. The minimum position for a mark is the length of the Optimum Golomb Ruler that corresponds to that mark position. For example, the minimum placement for the sixth mark would be at position seventeen. The position of the TOKEN mark is at the highest, or rightmost mark.

The TOKEN position is decremented to the previous mark. This mark is checked to see if it can be incremented. If it can then the position is incremented, all marks less than the current mark are reset to their initial positions, the TOKEN position is decremented to the previous mark and the procedure returns.

Otherwise, if the TOKEN is passed all the way to the first mark, then the token is passed forward to the lowest mark whose position can be incremented. This mark is incremented, all marks less than the current mark are reset to their initial positions, the TOKEN position is decremented to the previous mark, and the procedure exits.

The result of this algorithm is a sequence of rulers where each mark in the ruler is incremented only after the lower marks have progressed through all possible combinations of positions. In this way, Iterate is guarenteed to generate all possible mark combinations.

The Diffncheck testing procedure takes the set of marks and generates each row of the difference triangle. After the distances in one row are computed, they are individually compared with the

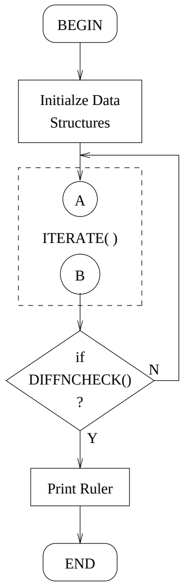

Figure 2.3: Flowchart for Token Passing Algorithm - Main Loop

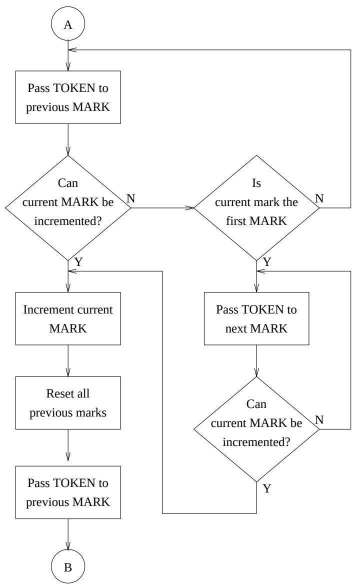

Figure 2.4: Flowchart for Token Passing Algorithm - ITERATE Subroutine

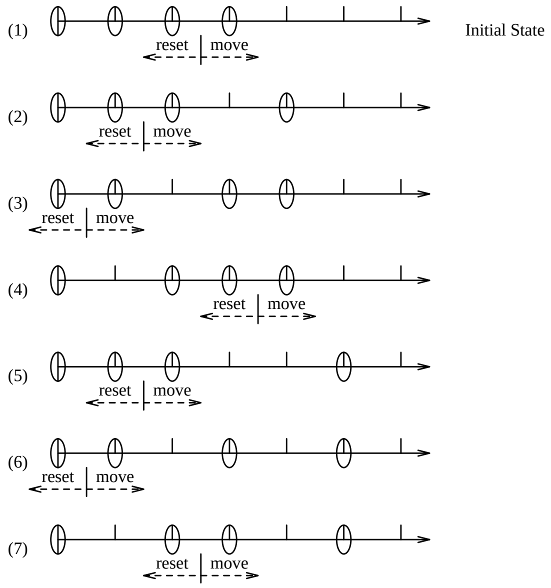

Figure 2.5: Example of Token Passing Algorithm

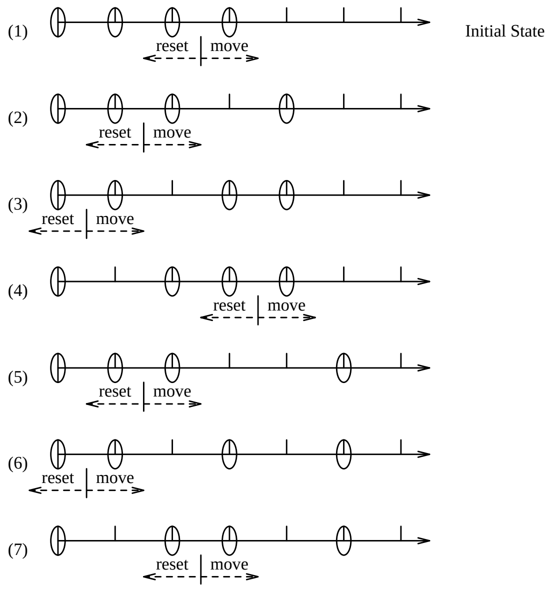

Figure 2.6: Example of Token Passing Algorithm (cont.)

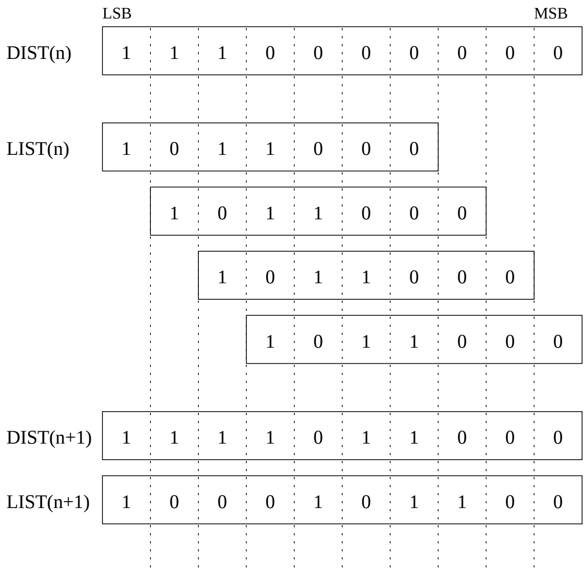

Figure 2.7: Example of Shift Algorithm

contents of the difference triangle. If two distances are found to be the same, then the procedure immediately exits with a failed status. Otherwise, the procedure exits after generating and comparing all the individual values in the difference triangle.

This process is illustrated in Figures 2.5 and 2.6 for the four mark ruler case. The ovals represent marks on the ruler, with position zero being the leftmost mark. For each cycle through the algorithm, a single mark is moved, and all marks to the left of that mark are reset to their initial positions. The lowest mark is always held at the zero position. For example purposes, the initial position for each mark n is at the $n+1$ location. In the actual implementation of the algorithm, these marks would be reset to their corresponding Golomb Ruler lengths, as described earlier.

This algorithm has the advantage that it will always produce the shortest ruler for a given number of marks first. It does not require an upper bound on the search space like the Scientific American and Shift algorithms do. However, the big disadvantage of this algorithm is in the use of the distance generation algorithm for ruler detection, which is computationally expensive.

2.3 Shift Algorithm

This algorithm was created by David McCracken as part of his undergraduate research project at Duke University [15]. The essence of the algorithm is taken from the Scientific American Algorithm, but with major improvements in the area of ruler checking.

The first improvement comes from the fact that if you save the array of distances between computations, then when a new mark is added only the new distances created by that mark need to be added and checked against the array. So instead of $N(N-1)/2$ distances to compute, only N distances need to be computed.

The next improvement reduces the checking and generation overhead even further. The distance array is stored as a bitmap, rather than as an array of boolean values. Each bit position in this distance vector (which we will call *DIST*) represents a distance measured by the current ruler. The list of marks is also stored in a bitmap vector, called *LIST*. Each set bit in *LIST* represents not so

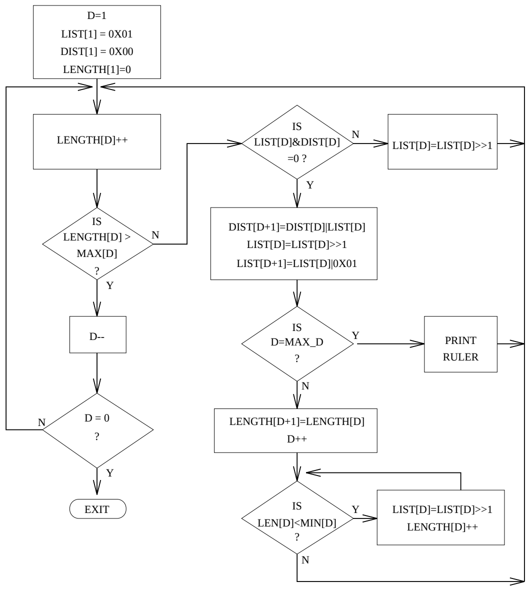

Figure 2.8: Flowchart for Shift Algorithm

much the position of a mark, but rather the distance from each mark to a single non-existent point one position past the last mark of the current ruler. So *LIST* can be pictured as the mirror image (shifted by one bit) of the ruler currently being evaluated.

In order to add a new mark to the ruler, the *LIST* vector is shifted out until the distances in that vector do not intersect any distances already measured in the *DIST* vector. This check is accomplished by simply AND-ing the two vectors together and checking to make sure that the resulting bit values are all zeros. In order to add the new point and all of its associated distances, the contents of *LIST* are OR-ed into *DIST* and the then lowest bit of *LIST* is set to add the new point. These improvements reduce the computational complexity to add and check a new point from $O(n)$ to $O(1)$.

An example of this process is shown in Figure 2.7. The initial contents of the *LIST* vector are for the three mark ruler $0, 1, 3$. The *LIST* vector is shifted times until it no longer has any set bits in common with those of the *DIST* vector. At this point the current *LIST* and *DIST* vectors are saved and the contents of the new *DIST* vector are updated by OR-ing in the contents of *LIST*, and the new mark is added to *LIST* shifting the vector and setting the first bit. The end result is that *LIST* now contains a representation of the four mark ruler $0, 1, 3, 7$ and *DIST* contains the set of distances, $1, 2, 3, 4, 6, 7$, measured by this ruler.

When a valid $N + 1$ mark ruler is formed, the existing N mark ruler is pushed onto a stack, and the algorithm attempts to add another mark to the new top ruler. If no more marks can be added to the current ruler, then the stack is popped and a new ruler is generated. The search space for a Golomb Ruler can be modeled as a large tree structure. The nodes at level N represent valid N mark rulers. The children of that node represent the set of all possible Golomb Rulers that can be formed by appending a single mark to the set of marks in the parent ruler. The Shift Algorithm traverses this search space in a depth-first manner, utilizing the stack to provide backtracking. Discovering a ruler is simply a case of finding a node that exists at the desired depth in the tree.

A flow chart of the algorithm is shown in Figure 2.8.

2.4 Tree Algorithm

As pointed out earlier, one of the problems with the Scientific American and Shift algorithms is the need to establish an upper bound on the search space prior to searching. If this upper bound is too low, then the Golomb Ruler will never be found. If this bound is too high, much time will be wasted in searching this extra space.

One algorithm that attempts to avoid this problem is the Tree Algorithm [15]. The search space is represented as a general tree structure, with nodes representing marks and the edges representing distances. Finding a valid mark consists of locating a node at the proper depth. The tree is built up by applying the Shift Algorithm to each of the nodes of the tree, checking to see if new nodes can be added.

The outer loop of the algorithm sets the maximum ruler length to a minimum known value. A subroutine is then called which adds nodes to the tree structure. If a ruler is not found at the desired depth in the tree, the subroutine exits. The outer loop then increments the maximum length and calls the subroutine again, which adds more nodes to the existing tree. Since the data in the tree is preserved between calls to the subroutine, this algorithm can reuse the existing data to generate only the sections of the search space that have not been traversed previously.

The advantages of this approach is that the process is fairly fast. Also, this algorithm will find the minimum length ruler without having a predetermined maximum bound on the search space.

However, the big disadvantage is the size of the data structures needed to hold the tree is proportional to the search space, which grows geometricly with each additional mark. Use of this algorithm to find rulers greater that about ten marks is prohibitive.

2.5 Search Space Reduction

As pointed out earlier, the Shift Algorithm is as efficient as the author knows in the generation and checking of rulers. In order to improve the execution time, the search space must be reduced. There are a number of restrictions that can be made that will drastically reduce this space.

Search space reduction techniques involve "preclusion" or "branch pruning" of the search tree in order to eliminate as many false branches as early as possible in the search algorithm [6]. Effectively, all the preclusion techniques used here involve "bounding" the positions of the marks by establishing upper and lower limits for each mark.

2.5.1 First Mark Preclusion

One of the earliest published uses of preclusion techniques is given for the Scientific American Algorithm [4]. By starting the second mark at the distance 2 instead of 1, the search space could be reduced by about ten percent for shorter rulers, while still maintaining an exhaustive search. This technique relies on the fact that if a ruler has its second mark placed at the first position (measuring a distance of 1 unit), then the ruler's mirror image cannot have a mark also at the first position. The algorithm may not detect the first ruler, if indeed it has a mark at distance 1, but is still guaranteed to find the mirror image of that ruler.

2.5.2 Midpoint Reduction

A variation on this method relies on the fact that the mirror image of a ruler still measures the same distances, as we stated earlier. Therefore, we can limit the position of the middle mark of the ruler to one side of the center. We are still guaranteed to find a ruler, or the corresponding mirror image, because one of the two must have its middle mark before the geometric center of the ruler. This one restriction effectively cuts the search space in half [5]. Note, however, that for rulers with even numbers of marks, this technique does not exclude the possibility of "discovering" both rulers of a mirrored pair. The reduction factor for even mark rulers is thus less than for the odd mark case. Note also that this technique cannot be combined with the First Mark Preclusion technique described in Section 2.5.1.

2.5.3 Maximum Position Reduction

The next space reduction technique relies on the fact that for any mark on the ruler, the remaining larger marks cannot occupy a space smaller than the corresponding Optimum Golomb Ruler for those marks. Thus there is a minimum distance that any mark can be from the end of the ruler, based upon the maximum ruler size and the mark position. This technique can be further improved by realizing that any distance measured by the first part of the ruler cannot be measured in the second part. If we keep a table of minimum ruler lengths as a function of their length and distances measured, the upper bounds for marks can be reduced even further [15]. This table is referred to as the *CHOOSE* array.

2.5.4 Minimum Ruler Preclusion

The third method is based on the fact that for a ruler with a given number of marks, certain distances have to be measured. For example, the minimum 11 mark ruler is of length 72, so we will limit the 11 mark of our search space to no less than 72. Now, all 11 mark rulers of length 72 or less measure the distance 1. Therefore, when attempting to append the 12 mark onto a length 72, 11 mark ruler, this mark cannot be placed at position 73. By keeping a table with this information, we can avoid these conditions when generating rulers [15]. This table is referred to as the *TCHOOSE* array.

2.6 Summary

There are a number of algorithms for generating and checking Golomb Rulers. The Shift Algorithm is by far the most efficient in term of both performance and resources, although the Tree Algorithm also shows very good performance, at the expense of memory size. However, both algorithms require that maximum boundary in order to limit their search space. The Token Passing Algorithm does not have this restriction. The key to finding and verifying large rulers, however, will not be so much the speed at which rulers can be checked, but rather the ability to trim down the search space as much as possible. The methods mentioned in this section succeed in this area; hopefully, additional refinements and new techniques may be added to this set.

Chapter 3

Evaluation of Algorithms

This chapter will provide a quantitative analysis of the major search algorithms presented. Algorithms will be analyzed in terms of their processing times and their internal iteration counts. Relative performance of these algorithms on different processors will be presented. Finally, the selection of the algorithm for parallel implementation will be reviewed.

3.1 Processing Time Analysis

The two algorithms being considered are the Shift Algorithm and the Token-Passing Algorithm. The base platform for all of our performance measurements in this section is the Sun SparcServer 1000 workstation. Section 3.3 will contain a comparison of the relative performance of both algorithms on a variety of platforms.

3.1.1 Shift Algorithm

While the Token-Passing Algorithm finds the minimum length ruler first, the Shift Algorithm must be provided with an upper bound for its search space. If this upper bound is too high, then significant time may be spent traversing useless search space. If the limit is too low, then a ruler will never be found and significant amounts of processing time will be wasted since the same space will be searched again, with a new, larger boundary.

In order to understand the effect that the upper bound limit has on the Shift Algorithm performance, timing measurements were taken with maximum limits set to lengths two and four marks greater than the optimum ruler length. Figure 3.1 shows the resulting execution times for the Shift Algorithm. These figures are consistent with previous timing measurements of this algorithm [15][1].

As shown in this figure, the Shift Algorithm is characterized by an exponential growth in execution time with respect to ruler length. This is characteristic of problems with combinatorial searches such as Golomb Rulers.

Figure 3.1 also shows that the execution time increases with an increase in the upper boundary limit. The lowest curve on the graph shows the execution time when the maximum bound is set to the length of the known optimum ruler. The other two curves show the execution times when the maximum boundary is set to two or four units above the optimum ruler length.

Increasing the maximum bound by N units has the effect of increasing the range of possible positions for each mark in the ruler by N units. The combinational growth of the search space results in a much larger increase in the overall search space for this relatively small increase in the boundary value. Figure 3.2 shows this increase in execution time, normalized to the time value for the optimum ruler case. For the twelve mark ruler case, increasing the maximum bounds from 72 to 76 units increases the overall execution time by a factor of greater than three. This graph

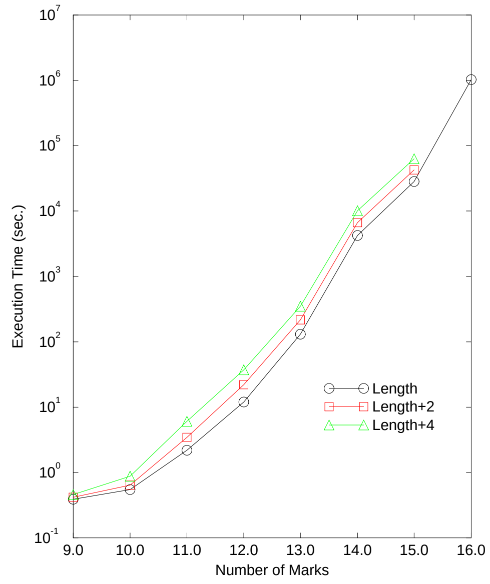

Figure 3.1: Execution Times for Shift Algorithm

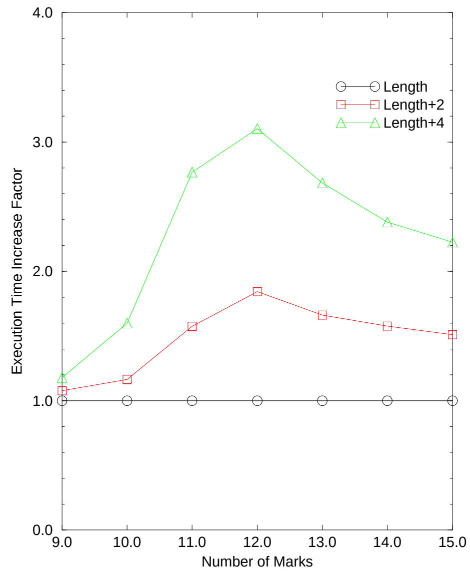

Figure 3.2: Execution Times Increase for Shift Algorithm

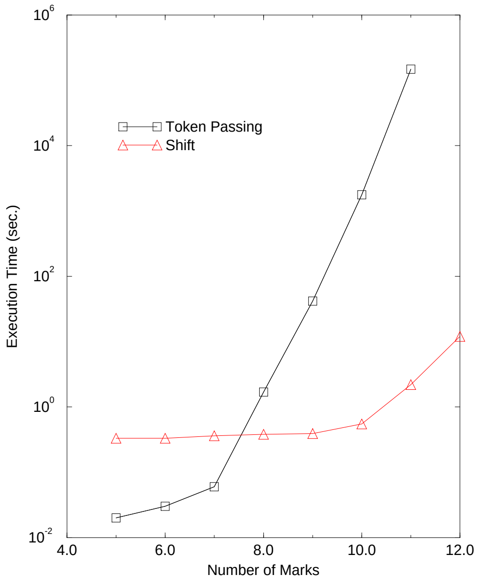

Figure 3.3: Execution Times for Token Passing Algorithm

also shows that for greater than 12 marks, the execution time factor is decreasing as the ruler size increases. This stems from the fact that the relative increase in ruler length is becoming smaller as the ruler length increases. Therefore, as ruler lengths become longer, establishing a maximum boundary within several marks of the actual Optimum Golomb Ruler length becomes less likely.

For rulers with less than 12 marks, the total execution time of the algorithm is influenced by the initialization process, which is independent of the ruler size. Thus large increases in the size of the search space have a much smaller effect on the total execution time.

3.1.2 Token-Passing Algorithm

The performance of the Token Passing Algorithm, as shown in Figure 3.3, demonstrates the same exponential growth curve as the Shift Algorithm. However, it is also noted that the execution time for the Token Passing Algorithm is a factor of ten greater than the best Shift Algorithm times. This is largely due to the computational cost of the difference generation and checking on each iteration.

The token passing algorithm finds the shortest ruler for a given number of marks. No maximum limit is needed for the algorithm to function. As a result of this, no data is given for different maximum limits.

3.2 Iteration Analysis

Sample Total CPU time is not always the best metric by which to measure and quantify algorithms. A slow algorithm may be slow because the processing time for each iteration is be substantial. Another very important metric is the number of processing iterations each algorithm needs to complete execution. Comparing the growth trends for the iteration counts of both algorithms provides a better estimate for the performance of the algorithm for very large processing runs.

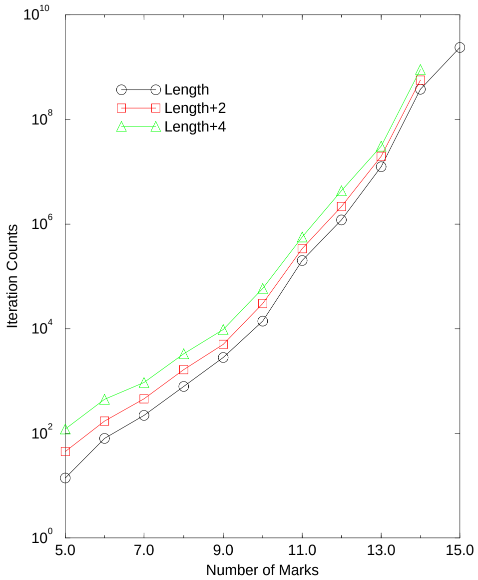

Figure 3.4: Iteration Counts for Shift Algorithm

3.2.1 Shift Algorithm

In order to analyze the number of iterations performed by the Shift Algorithm, a counter was added to the existing code. This counter in incremented whenever the *LIST* and *DIST* vectors do not contain any set bits in the same bit positions, indicating that a new mark may be added onto the *LIST* vector to create a new rulers. At this point the algorithm descends a level of the search tree. The value that the counter returns at the end of the search is the actual number of nodes that constitute the search space.

The value of the counter as a function of the number of marks is shown in Figure 3.4. As with the previous timing measurements from Figure 3.1, separate curves are given for optimum and near optimum maximum values.

Figure 3.5 shows the increase in the search space size normalized by the count for the optimum boundary value. The graph data shows similar behavior that of the execution times in Figure 3.2. The peaks corresponding to the even numbered rulers are the result of the greater efficiency of the midpoint reduction technique for odd numbered rulers, as discussed in Section 2.5.2.

3.2.2 Token-Passing Algorithm

The iteration counter for the Token Passing Algorithm is incremented whenever a mark is moved and a new potential ruler is generated and checked. This count represents the "outer loop" of the algorithm as described in the previous chapter. The results of this count are shown in Figure 3.6. Iteration counts are also shown for the Shift Algorithm for comparison.

The Token Passing Algorithm also exhibits geometric growth, but at a much higher growth rate than the Shift Algorithm. This indicates that the search space reduction techniques implemented in the Shift Algorithm are more efficient at reducing the number of program iterations. Whereas similar preclusion techniques could be used with the Token Passing Algorithm, the original computation

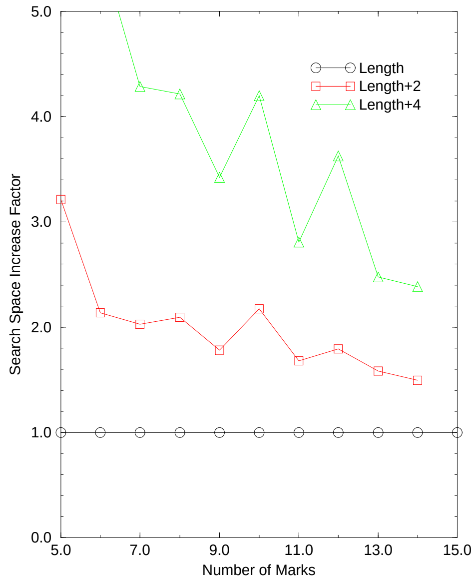

Figure 3.5: Iteration Count Increases for Shift Algorithm

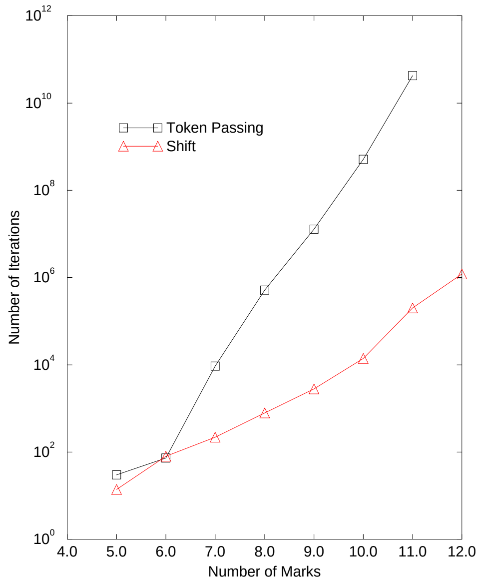

Figure 3.6: Iteration Counts for Token Passing Algorithm

Processor	Algorithm	
	Shift	Token
DecStation 2100	1.00	1.00
DecStation 3100	1.33	1.34
DecStation 5000	2.00	2.05
Sun SparcStation IPX	1.90	2.99
Sun SparcStation 10	2.92	4.07
Sun Sparc Classic	3.07	4.27
Sun SparcServer 1000	6.25	7.33
IBM RS6000/350	1.35	3.21
Intel 486-SX25	1.34	1.62

Table 3.1: Relative Processor Performance

times combined with the iteration trends did not warrant investing research time in this area.

3.3 Platform Analysis

It is important to recognize that the performance of a program is directly related the the specific processor and system architecture that the process is running on. Information about the relative performance of algorithm implementations running on different systems is also very useful for analysis of the eventual parallel implementation of these algorithms.

Table 3.1 provides measurements of the relative performance of the Shift and Token Passing Algorithms on several different processors. The numbers represent the relative performance increases, and are inversely proportional to the execution times. The measurements are normalized to the value of the slowest processor. For the Shift Algorithm, average timings were taken over four runs for 12 and 13 mark rulers. For the Token Passing Algorithm, the average of four timings was taken for 9 mark ruler search. Execution time was measured through the use of the Unix system `time` command. Full compiler optimizations were used in all cases.

All three DecStations demonstrate uniform performance increase for both algorithms. This is expected since all three machines share the same processor architecture. However, the Sun workstations show a greater speedup when running the Token Passing Algorithm over the Shift Algorithm. The IBM workstation exhibits the same behavior.

It is important to note that different versions of the same model of workstation may provide very different performance ratings. Specificly, the IBM RS6000 and Sun Sparc 10 processors have many different versions of the same model, each displaying different performance.

3.4 Selection of Algorithm

From the results of these measurements, it is clear that the Shift Algorithm offers the best performance in searching for Optimum Golomb Rulers. As seen from the timing measurements, the Shift Algorithm is the most efficient at processing individual rulers.

While the Token Passing Algorithm has the benefit of producing the shortest ruler first, without the need to establish a maximum limit, it suffers from the overhead of having to recompute the difference tree at each iteration. In addition, some of the search space reduction techniques used by the Shift Algorithm do not have direct application to the Token Passing Algorithm.

In summary, the Shift Algorithm proves to be the best prospect for parallel implementation.

Chapter 4

Parallel Implementation

While certain algorithms for generating optimum Golomb Rulers have been shown to be very efficient in traversing the search space, they still exhibit a geometric growth rate with respect to the number of marks being searched for. The speed of the algorithms reviewed in the last chapter are limited by the performance of the uniprocessor that they execute on. While the performance of uniprocessors has increased dramatically in recent years, it has not kept up with the rate at which new rulers are being searched for. Thus the execution times for these algorithms quickly becomes prohibitively excessive as the number of marks increase.

However, by taking advantage of the inherent parallelism of certain of these search algorithms, it is possible to distribute the workload among a large group of processors and achieve a significant increase in speed over the uniprocessor case.

This chapter will describe the parallel implementation of one of the algorithms previously reviewed, together with an analysis of the performance gain in the parallel implementation over that of the similar sequential algorithm.

4.1 Programming Environment

One of the initial considerations that must be made is determining the parallel/distributed environment under which the algorithms will be implemented. For our experiments, we use the PVM system [10]. PVM is a set of libraries and Unix daemons that utilize a networked group of workstations to implement a distributed parallel system.

For our application, PVM offers many advantages over other parallel systems.

- Scalability. PVM can support as many processors as are available on the workstation network.
- Homogeneous Environment. PVM allows the use of different processor architectures simultaneously, transparent to the application.
- Fault Tolerance. The failure of a single processor on the network will not cause PVM to fail. PVM will log the processor failure and continue with the reduced set of processors.
- Availability. As in most institutions, access to large amounts of CPU time on a Parallel Supercomputer is rather restricted. However, as we will cover in a later section, Workstation CPU time is more generally available and if the proper precautions are taken, processing can take place with minimal impact to the other users of the system.
- Portability. PVM his been ported to a variety of different platforms, from aging Sun-3 and IBM RT workstations to CM-5 supercomputers. This allows the programmer to take advantage of new processors as they become available.

There are also some disadvantages to the PVM system, which may make it unsuitable for certain applications.

- Communications Overhead. The communications medium for PVM version 3 is ethernet and Unix TCP/UDP sockets. Depending on the network topology, there may be significant and non-uniform communications overhead between processors. This overhead must be taken into account by the programmer.
- Load Balancing. PVM does not provide for load balancing among its processor group. Load balancing must be implemented at the user level. In addition, each processor on the network will have differing performance depending on the workload of that particular workstation. Synchronization of many processes may introduce significant overhead into the overall execution time.

These advantages and disadvantages will have to be taken into account for the design of the parallel implementation.

4.2 Algorithm Selection

Now that the parallel programming environment has been defined, the appropriate algorithm must be selected.

4.2.1 Goals

In order to review any algorithms it is necessary to establish a set of goals or metrics against which the algorithm can be judged. These metrics are:

1. Performance.

   Good sequential performance does not necessarily assure good parallel performance. However, if a highly optimized sequential algorithm scales easily to a parallel implementation then it may well provide the best parallel performance also.

2. Scalability.

   The parallel implementation of the algorithm should take maximum advantage of all available processors and not limit itself to a fixed number.

3. Portability.

   The algorithm should not be dependent on any specific processor architecture or feature.

4. Low Communications Overhead.

   Since the processor groups that are used by PVM might not be tightly coupled, there may be significant overhead involved in interprocess communications. Any parallel implementation should strive to minimize this overhead.

Of the sequential algorithms we looked at, the Shift algorithm provides the highest sequential performance, and at the same time lends itself well to partitioning.

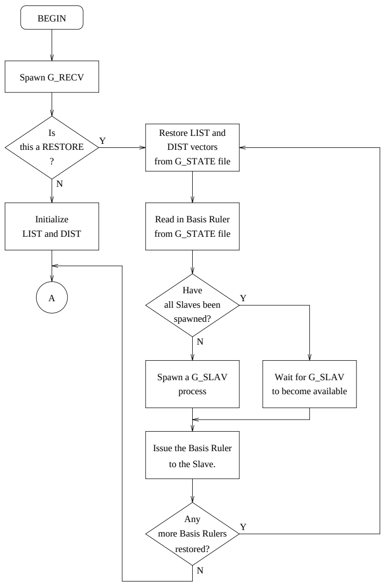

Figure 4.1: Flowchart of Master Process, g_mast

4.2.2 Description

As shown in Chapter 2, the Shift Algorithm generates Golomb Rulers by taking an existing N mark ruler and adding new points to the end of the ruler. The search space for the ruler can be viewed as a large tree where each node at depth N represents an N mark ruler and all the child nodes represent the $N+1$ mark rulers formed from adding points to the parent ruler. The search process is simply a depth-first traverse of the tree, generating new nodes as needed.

The parallel implementation of this algorithm uses a master process, g_mast, which generates N mark rulers, called *basis* rulers, and passes each ruler to a slave process that uses that ruler as the starting point from which to continue the search. The master process traverses the search tree in a depth first manner until it descends to a specific level at which point it passes the current node to a slave process, g_slav, which completes the search of the subtree identified by that node. Flowcharts of the master and slave routines are shown in Figures 4.1, 4.2 and 4.3. In addition to the slave processes, the master process also spawns a monitoring process, g_recv, which handles processing of result data from the slaves. A flowchart for the g_recv process is shown in Figure 4.4 below.

The slave process begins by initializing itself and then waiting for basis ruler data to be received from the master process. When the basis ruler data arrives, the slave process uses the Shift Algorithm to search for optimum rulers, using the basis ruler as its starting point. When the slave process has completed the search process for a specific basis ruler, it sends a message to the master process indicating that it is now idle and waiting to receive the next basis ruler. If the slave process discovers a valid ruler, it sends a message containing the ruler data to the g_recv process.

The g_recv process handles a number of administrative tasks. The process receives data from the slave processes containing valid solution rulers. These rulers are stored in an output file. The g_recv process also monitors for the termination of the master process and waits for all outstanding slave processes to complete before terminating itself. Finally, the g_recv process monitors slave termination and maintains a count of the number of outstanding slave processes.

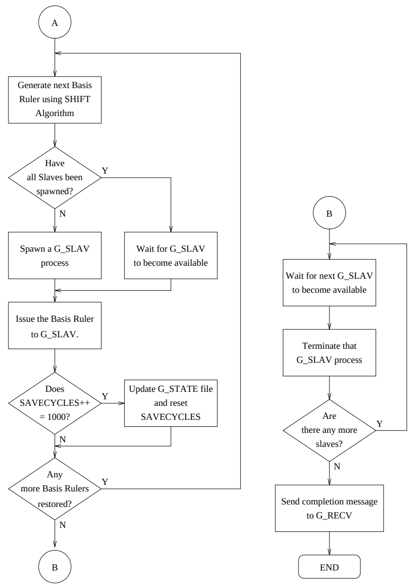

Figure 4.2: Flowchart of Master Process (cont)

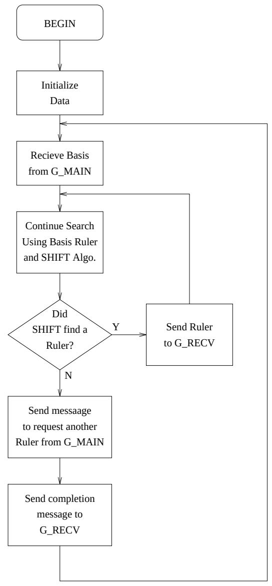

Figure 4.3: Flowchart of Slave Process, g_slav

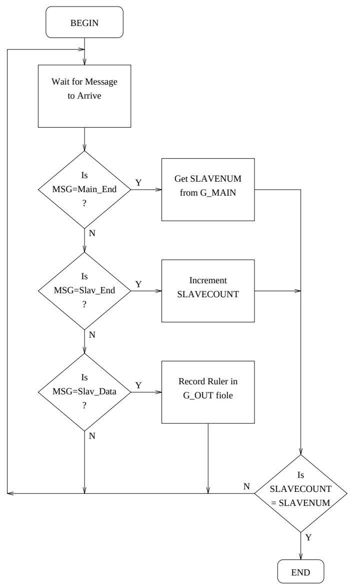

Figure 4.4: Flowchart of Receive Process, g_recv

The specifics on the Shift algorithm implementation have already been discussed and will not be presented here.

4.3 Parallel Issues

4.3.1 Load Balancing

One of the design goals in implementing a parallel algorithm is to maximize the use of all available processors. The parallel algorithm described here implements a load balancing strategy based upon the concept of a Processor Pool [10]. The slave processes can be imagined as existing in a pool which is managed by the master process. When a slave completes a job, it reports this to the master process, which immediately issues another job to that slave. As long as the master process can produce and issue jobs as fast as the slaves processes can complete them, all slave processes will be busy and maximum usage of available resources will be maintained.

4.3.2 Granularity

Another execution parameter that affects algorithm performance is the relative size, or granularity, of each of the slave processes when compared to the overall problem size. If the granularity is too fine, then the slave processes may spend a large part of their overall processing time requesting new jobs from the master process, reducing the aggregate throughput. In addition, there is the potential that the master process will be unable to produce new basis rulers at the rate that they are being processed by the slaves, resulting in a serial bottleneck. If the granularity is set too large, then failure of a slave processor can result in a large loss of corresponding processing.

In the parallel algorithm, the granularity of the search process is determined by the number of marks in the basis rulers that are sent to the slave processes. The larger the basis ruler generated by

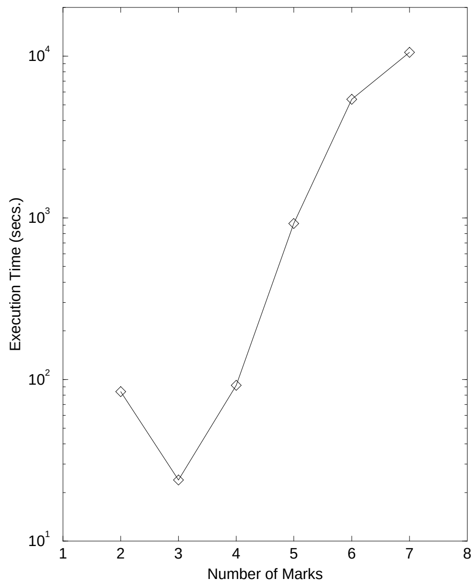

Figure 4.5: Effects of Basis Ruler Length on Performance for a 13-Mark Ruler

the master program, the smaller the corresponding search space that the slave must process. The number of marks in the basis ruler is set at run time and may be adjusted to account for the number and type of slave processors available.

Figure 4.5 shows the effects of basis ruler length on the performance of the parallel algorithm. A 13-mark ruler of length 106 was computed using 10 identical processors. The basis ruler length was varied and the resulting execution times were measured. For basis ruler lengths of greater than three marks, the total execution time grows significantly. This is a result of the fact that as the basis ruler length is increased, the size of the search space that the slave must traverse decreases. The amount of processing time that the slave spends of a single basis ruler decreases proportionally. Thus the percentage of time that the slave spends in communications overhead increases and reduces the parallel efficiency of the algorithm.

In practice, the g_main process spends most of its time in an idle state waiting for g_slav processes to request new basis rulers. While running a 19 mark ruler search using 6 mark basis rulers, the g_main process used less than 0.5 percent of the available CPU time as compared to the g_slav process.

4.3.3 Restartability

Due to the fact that the total duration of the search may be on the order of weeks or even months, it is necessary that the state of the program should be saved to a file at specific intervals and that the program should be able to restart itself from the data contained in a statefile. This restart should be performed with a minimum loss of processing time and still guarantee an exhaustive traverse of the ruler search space, with no loss of data.

Two sets of data from the parallel algorithm are saved into this state file, which is referred to as the G_STATE file. The first set of data saved is the current position of the master program in the search space. This is accomplished by writing out the contents of the $LIST$, $DIST$ and $COUNT$

stacks to the G_STATE file. This allows the master process to re-establish its position in the search space for issuing the next basis ruler once it is restarted.

In addition, it is necessary to guarantee that the basis rulers being processed at the time of the last saved state are reissued and the processing on them is completed. This safeguard is employed to assure that the corresponding areas of the search space are not lost. An array is maintained by the master process which contains the basis ruler data issued to each slave. There is one entry in the array for each slave process. The contents of the array are appended onto the G_STATE file so a record of all outstanding slave processes is maintained.

Restarting the master process consists of reloading the *LIST* and *DIST* stacks from the contents of the G_STATE file. Slave processes are started and initialized using the outstanding basis ruler set. Once all of these basis rulers have been processes by the slave pool, then the master begins generating new basis rulers.

The G_STATE file is updated after a fixed number of basis rulers have been sent out by the master process. This number is called *SAVECYCLES* and is set to a value of 1000. So, as shown in Figure 4.2, after every 1000 basis rulers are sent to the slave processes, the current state is saved.

4.3.4 Fault Tolerance

Since the distributed algorithm is being run across a great number of workstations, it is necessary for the program to be able to continue working even after the loss of many processing elements. A very simple approach is used by the master process to allow for the loss of slave processes.

If a slave process dies, then it never reports completion back to the master process. As a result, the basis ruler being processed by that slave is never removed from the G_STATE array and will be reissued when the master process is restarted. Note that this means that the master process will never complete at this time since the slave processes will always remain outstanding. It is necessary to restart the master process once a slave has gone down in order to complete the ruler search. The PVM system provides an interactive facility to monitor the status of all processes. PVM also provides the user with information on the status of the individual processors, as well as the capability to add and delete processors from the current virtual machine.

If the master process terminates unintentionally, then there is no recovery short of terminating all of the slave processes and restarting the system from the last G_STATE file. Again, the PVM system provides the capability to reset or kill all current slave processes, at which point the master process may be restarted from the Unix shell command line.

4.4 Performance Analysis

In any parallel adoption of a sequential algorithm, there will invariably exist an overhead cost associated with the parallel setup and termination phases. This section will review the efficiency of the parallel implementation of the Shift Algorithm.

4.4.1 Test Platform

The performance measurements were made on a homogeneous cluster of twelve Sun SparcClassic workstations. No other users were logged in when these measurements were taken. Measurements were taken of the real, or "wall clock" time needed for the master process to complete, and due to the light load of the cluster are representative of the true parallel execution times.

4.4.2 Test Procedure

A number of different size rulers were computed using a varied number of processors. The resulting execution times were compared to the execution time for the sequential code running on a single

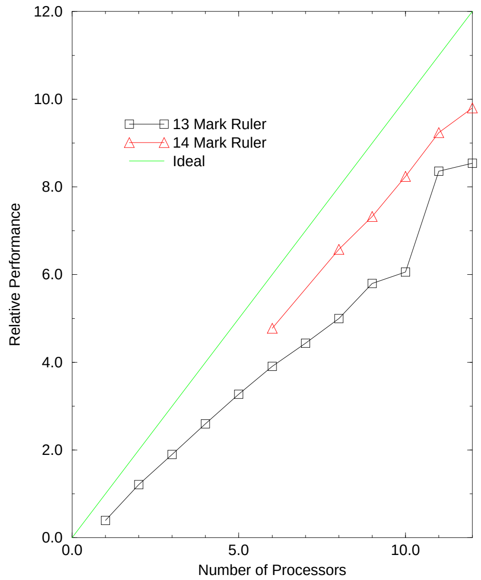

Figure 4.6: Performance of Parallel Algorithm

processor.

4.4.3 Test Results

The results from the performance runs are summarized in Figure 4.6. Timing information was taken for thirteen and fourteen mark rulers. Each ruler was run multiple times with a fixed number of CPUs and the minimum result was taken. In order to get the relative performance figure, the execution time of the sequential algorithm was divided by the execution time for the parallel job, resulting in the relative performance.

The relationship between relative performance and number of CPU's is linear for both cases, as expected. However, for the smaller ruler case, there is an incremental performance improvement of 65addition for one processor, as compared to an 85ruler. This stems from the fact that for the smaller ruler, the search space that the slaves must traverse for each basis ruler is much smaller. Thus the relative amount of time each slave spends in communication overhead is proportionally higher.

Another issue potentially affecting performance is the ability of the master processes to produce basis rulers at a rate of at least that of the aggregate processing rate of all the slaves. If the master process cannot produce basis rulers fast enough, then each slave will spend proportionally more time waiting for the master to supply a basis ruler. However, in practice, we have seen that master process spends the vast majority of its time waiting for the slaves to finish processing. Because the master process spends most of its time waiting for data requests from the slaves, a slave process is also spawned on the same processor as the master process, maximizing use of all available processors.

Chapter 5

Results

Having verified the correctness of the parallel implementation of the Shift algorithm, the program is run with ruler lengths for which the optimum solutions were not known. The results of these runs are presented in this chapter.

5.1 New Rulers

Optimum seventeen and eighteen mark Golomb Rulers were generated in separate runs on different workstation clusters. The rulers are shown in Tables 5.1 and 5.2.

In deriving the 17 and 18 mark rulers, however, we were not the first to computationally complete the optimality search. Both rulers were derived first by Mr. Olin Sibert in April and June 1993, respectively [22], whereas we derived the corresponding rulers in August and September 1993. Mr. Sibert's results, however, have not been published to date and were relayed to us by Prof. Golomb.

5.2 Performance

As shown in the previous chapter, the parallel algorithm exhibits a high processor utilization and almost linear performance gain. The following sections summarize the execution times for the seventeen and eighteen mark search runs.

5.2.1 Golomb Seventeen

The optimum seventeen mark ruler was determined from a run on a homogeneous cluster of DEC workstations. Up to 92 workstations were utilized simultaneously, although this number varied widely during the period of execution. Timing results for this run were inconclusive due to a number of factors.

1. Problems with the AFS and NFS file systems caused significant setbacks. Often state and timing data for several days of execution time were lost and the search position had to be reset from the last available state.

2. A bug in the parallel code caused a significant performance loss. This bug was fixed during the course of the search. There was no loss of any portion of the search space in the process of correcting this bug.

The total time spent running the parallel shift algorithm was approximately one month. A majority of this time was spent recovering, investigating and debugging problems brought about by the behavior of the AFS and NFS file system.

0	8	31	34	40	61	77	99	118	119	132	143	147	182	192	194	199
8	23	3	6	21	16	22	19	1	13	11	4	35	10	2	5	
31	26	9	27	37	38	41	20	14	24	15	39	45	12	7		
34	32	30	43	59	57	42	33	25	28	50	49	47	17			
40	53	46	65	78	58	55	44	29	63	60	51	52				
61	69	68	84	79	71	66	48	64	73	62	56					
77	91	87	85	92	82	70	83	74	75	67						
99	110	88	98	103	86	105	93	76	80							
118	111	101	109	107	121	115	95	81								
119	124	112	113	142	131	117	100									
132	135	116	148	152	133	122										
143	139	151	158	154	138											
147	174	161	160	159												
182	184	163	165													
192	186	168														
194	191															
199																

Table 5.1: Difference Triangle for Optimum 17 Mark Ruler

0	11	24	28	49	63	68	86	118	127	133	134	160	163	194	206	214	216
11	13	4	21	14	5	18	32	9	6	1	26	3	31	12	8	2	
24	17	25	35	19	23	50	41	15	7	27	29	34	43	20	10		
28	38	39	40	37	55	59	47	16	33	30	60	46	51	22			
49	52	44	58	69	64	65	48	42	36	61	72	54	53				
63	57	62	90	78	70	66	74	45	67	73	80	56					
68	75	94	99	84	71	92	77	76	79	81	82						
86	107	103	105	85	97	95	108	88	87	83							
118	116	109	106	111	100	126	120	96	89								
127	122	110	132	114	131	138	128	98									
133	123	136	135	145	143	146	130										
134	149	139	166	157	151	148											
160	152	170	178	165	153												
163	183	182	186	167													
194	195	190	188														
206	203	192															
214	205																
216																	

Table 5.2: Difference Triangle for Optimum 18 Mark Ruler

The seventeen mark ruler was also independently verified by a sequential version of the Shift algorithm run on a Sun SparcServer 1000 workstation. The total execution time was approximately 720 hours, i.e. roughly one month, the same calendar time as the 92 machine cluster combined. The factor of 6 performance difference (see Table 3.1), the 100% utilization of the Sun SparcServer 1000 processor, and the lack of need for backtracking on the dedicated serial run account for the similarity of these figures. This result is significant because it represents the limit of serial execution in absolute terms, as well as identfying potential problems in parallel execution.

5.2.2 Golomb Eighteen

The optimum eighteen mark Golomb Ruler was verified in a parallel run on a cluster of one Sun SparcServer 1000 and twelve Sun SparcClassic workstations. Other than the Golomb Ruler application, the workstations were largely unused during this period, allowing a high percentage of CPU utilization for the course of the run.

The total elapse time for the run was approximately 840 hours, or 35 days. The timing information supplied by the master process indicates that the 615 hours of this time was spent actively searching for rulers. The remaining 225 hours were spent in a non-processing state.

Chapter 6

Summary

The parallel implementation of the Shift Algorithm has been successful in identifying two previously unpublished Optimum Golomb Rulers, and is currently searching for the 19 mark Golomb Ruler.

An important research result is the demonstration of reliable parallel computation for extended execution times. Fault tolerance and restartability issues are addressed and have been shown to be effective in assuring processing reliability for program runs in excess of a month. Finally, previously unpublished results were found and verified against independent findings from another researcher, Mr. Olin Sibert.

6.1 Additional Work

While significant effort has been made to the programs, there are several areas of work that could provide additional performance improvement to the existing programs.

6.1.1 Optimization of Code

One area for performance improvement lies in the hand optimization of the inner loops of the program. Up to this point, all optimization has been carried out by the compiler. The reason for the lack of hand optimization was to ensure the validity of the results during algorithm development. Initial trials at hand optimization for the serial Shift Algorithm indicate that having the compiler dump the assembly language programs and hand ordering the instructions for the inner loops of the algorithm, it could be possible to increase performance by 10-20% over the compiler optimized case.

6.1.2 Search Space Reduction

Another area where performance could be improved is in the search space reduction techniques. The Maximum Position Reduction techniques, referred to in Section 2.5.3 could be expanded to include more marks. Due to processing and time limitations, the *CHOOSE* array elements are only computed for rulers up to twelve marks. The remaining elements in the *CHOOSE* array are simply set to the minimum Golomb Ruler length for that level.

Improved performance would be achieved by computing *CHOOSE* array elements for thirteen or greater marks. One possible method to achieve this would be to increase the search limits for the current program and allow it to run for a longer period of time. Alternately, since the program that generates the *CHOOSE* array is simply a variant of the Shift Algorithm, it would be possible to implement a parallel version of this program and generate much larger *CHOOSE* array data.

Analysis of the effects of Maximum Position Reduction on the serial Shift Algorithm indicate that a potential performance increase of 10-15% could be realized for each additional *CHOOSE* element added.

6.1.3 Longer Word Length

A major limitation for the Shift Algorithm is the implementation of the *LIST* and *DIST* vectors. On most current workstations shift and logical bit operations are limited by the maximum register size. For the majority of workstations, this is thirty-two bits. Thus the shift and logical bit operations performed on these vectors is done in a piecemeal fashion, breaking up the vector into a series of sequential operations.

The next generation of processors which are becoming available are primarily based around a sixty-four bit register. By utilizing sixty-four bit registers, the number of bit-vector operations, which form a significant portion of the innermost processing loop, would be effectively halved. Some initial testing of this code on a sixty-four bit processor indicates a 25% performance gain due to increased register size.

6.1.4 G_RECV Improvements

The `g_recv` process, which monitors results from the slave process, was originally implemented under PVM version 2.4. Because of the limitations of the PVM 2.4 library, the `g_recv` process is implemented as a polling loop rather than blocking for incoming messages. As a result, the `g_recv` process steals half of the cycles from the slave process that shares the same CPU. If the `g_recv` process were updated to block on incoming messages, it would dramatically reduce the CPU load on the corresponding processor. This in turn would free up cycles that the slave process could utilize.

6.2 Future Rulers

A nineteen mark ruler is currently being processed with results expected in the first few months of 1994. The best known nineteen mark ruler is shown in Table 6.1.

0	4	13	15	42	56	59	77	93	116	126	138	146	174	214	221	240	245	246
4	9	2	27	14	3	18	16	23	10	12	8	28	40	7	19	5	1	
13	11	29	41	17	21	34	39	33	22	20	36	68	47	26	24	6		
15	38	43	44	35	37	57	49	45	30	48	76	75	66	31	25			
42	52	46	62	51	60	67	61	53	58	88	83	94	71	32				
56	55	64	78	74	70	79	69	81	98	95	102	99	72					
59	73	80	101	84	82	87	97	121	105	114	107	100						
77	89	103	111	96	90	115	137	128	124	119	108							
93	112	113	123	104	118	155	144	147	129	120								
116	122	125	131	132	158	162	163	152	130									
126	134	133	159	172	165	181	168	153										
138	142	161	199	179	184	186	169											
146	170	201	206	198	189	187												
174	210	208	225	203	190													
214	217	227	230	204														
221	236	232	231															
240	241	233																
245	242																	
246																		

Table 6.1: Difference Triangle for Best Known 19 Mark Ruler

Bibliography

[1] Emily S. Angerer. Parallel and Sequential Timings of Golomb Ruler Calculation Codes. Senior Thesis, Duke University, Durham, NC, December 16 1992.

[2] W. C. Babcock. Intermodulation Interference in Radio Systems. *Bell Systems Technical Journal*, pages 63–73, January 1953.

[3] Gary S. Bloom and Solomon W. Golomb. Applications of Numbered Unidirected Graphs. *Proceedings of the IEEE*, 65:562–570, April 1977.

[4] A. K. Dewdney. Computer Recreations. *Scientific American*, pages 16–26, December 1985.

[5] A. K. Dewdney. Computer Recreations. *Scientific American*, pages 14–21, March 1986.

[6] J Nievergelt E. M. Reingold and N. Deo. *Combinational Algorithms: Theory and Practice*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[7] E. J. Blum F. Biraud and J. C. Ribes. On Optimal Sythetic Linear Arrays with Applications to Radioastronomy. *IEEE Transactions on Antennas and Propagation*, AP-22:108–109, 1974.

[8] M. Gardner. Mathematical Games. *Scientific American*, 226:108–112, March 1972.

[9] M. Gardner. Mathematical Games. *Scientific American*, 226:114–118, June 1972.

[10] Geist, Beguelin, et. al. *PVM 3 User's Guide and Reference Manual*, May 1993.

[11] E. J. Kleiber. Some Difference Triangles for Constructing Self-Orthogonal Codes. *IEEE Transactions on Information Theory*, IT-16:237–238, 1969.

[12] Torleiv Klove. Bounds on the Size of Optimal Diffence Triangle Sets. *IEEE Transactions on Information Theory*, IT-34:355–361, 1988.

[13] Torleiv Klove. Bounds and Construction for Difference Triangle Sets. *IEEE Transactions on Information Theory*, IT-35:879–886, July 1989.

[14] N. Santoro M. D. Atkinson and J. Urrutia. Integer Sets with Distinct Sums and Differences and Carrier Frequency Assignments for Nonliner Repeaters. *IEEE Transactions on Communications*, COM-34:614–617, June 1986.

[15] David McCracken. Minimum Redundancy Linear Arrays. Senior Thesis, Duke University, Durham, NC, January 23 1991.

[16] Alan T. Moffet. Minimum-Redundancy Linear Arrays. *IEEE Transactions on Antennas and Propogation*, AP-16:172–175, March 1968.

[17] J. Robbins R. Gagliardi and H. Taylor. Acquisition Sequences in PPM Communications. *IEEE Transactions on Information Theory*, 33:738–744, Sept 1987.

[18] W. T. Rankin. Optimal Golomb Rulers: An Exhaustive Parallel Search Implementation. Master's thesis, Duke University, 1993.

[19] J. P. Robinson. Optimal Golomb Rulers. *IEEE Transactions on Computers*, C-28:943–944, December 1979.

[20] J. P. Robinson and A. J. Bernstein. A Class of Binary Recurrent Codes with Limited Error Propogation. *IEEE Transactions on Information Theory*, IT-13:106–113, January 1967.

[21] James B. Shearer. Some New Optimum Golomb Rulers. *IEEE Transactions on Information Theory*, IT-36:183–184, January 1990.

[22] W. Olin Sibert. In letter to H. Taylor and S. Golomb, April 1993.

Appendix A

Tutorial Introduction to Golomb Rulers

by Aviral Singh[1]

A.1 Golomb Rulers

The Golomb Ruler got its name from Prof. Solomon W. Golomb, a professor of Mathematics and Electrical Engineering at the University of Southern California. The rulers were first discussed by W.C. Babcock, albeit not under the same name in 1952 [2]. He discussed them not as a concept but more as a set of numbers which gave him the results he wanted.

A.1.1 What is a Ruler

A normal *Ruler* is used to measure distances. It has marks at equidistant points and the distances are measured from one point to another. For our purposes, we will consider rulers with marks at integer locations. Let us assume that the ruler has m marks and the length of the ruler is n. For a normal ruler, $n = m - 1$. Such a ruler is shown in figure A.1.

The ruler in the figure has four marks at 0, 1, 2 and 3. Thus, it can measure distances up to three units in length. The distances are marked on the ruler. As shown distance one can be measured in three different ways - from 0 to 1, from 1 to 2 and from 2 to 3. Similarly, distance two can be measured in two distinct ways. This is good for a ruler since the measurement for any length (up to the maximum possible) can be taken starting from the first mark (mostly at '0'). But, the removal of this property is essential for many other applications.

A.1.2 What is a Golomb Ruler

Consider a ruler in which one distance can be measured only in one way. In other words, the distance between any two points on the ruler is unique. Such a ruler with four marks is shown in figure A.2. The distance between each pair of marks is also shown. As can be seen, they are all distinct. How do we test a ruler for this property? The easiest way to do this is to construct a *difference triangle*. Let the marks of ruler be at M_1, M_2, M_3 and M_4. Then the difference triangle is constructed as follows :-

[1]Reproduced with permission

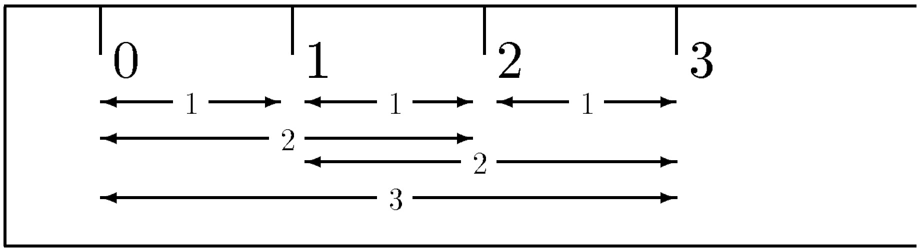

Figure A.1: A normal *ruler*

$$\begin{array}{ccccc} d1_1 & & d1_2 & & d1_3 \\ & d2_1 & & d2_2 & \\ & & d3_1 & & \end{array}$$

where $d1_x$, $d2_x$ and $d3_x$ are the first order, second order and third order differences respectively. The relationships are as follows :-

$$d1_x = M_{x+1} - M_x$$

$$d2_x = d1_x + d1_{x+1}$$

$$d3_x = d2_x + d2_{x+1} - d1_{x+1}$$

The equation for higher order differences (for larger rulers) follows the format of the equation for third order differences. The triangle contains the distances between each pair of marks. If all the numbers are distinct, the above property is satisfied. Let us construct the triangle for the ruler in figure A.2.

$$\begin{array}{ccccc} 1 & & 3 & & 2 \\ & 4 & & 5 & \\ & & 6 & & \end{array}$$

As we can see, all the numbers in the triangle are distinct. Therefore, the ruler if figure A.2 does satisfy the property. This property is a requirement for a ruler to be Golomb. This property is hereafter referred to as the first requirement.

A.1.3 Optimum Golomb Rulers

Golomb Rulers are divided into many different classes. All classes satisfy the first requirement but they are different in other aspects.

A closer look at the distances of the ruler in figure A.2 will reveal that the ruler can measure all distances from 1 to 6 (the length of the ruler). This is possible because the distances 1, 2 and 3 (1 to m - 1) are the first order distances (the distances between adjacent marks). Such rulers are called '*Perfect* Golomb Rulers'.

Perfection is rare among Golomb Rulers. In fact Golomb has supplied a proof that says that no ruler with more than four marks can attain perfection [4]. The proof is loosely stated here. If the Golomb Ruler is perfect, distances 1 to m - 1 must constitute the first order distances of the ruler. If the distance 1 is next to any distance other than m - 1, a violation will occur. This is because the sum of these distances (a second order distance) will already be present as a first order distance.

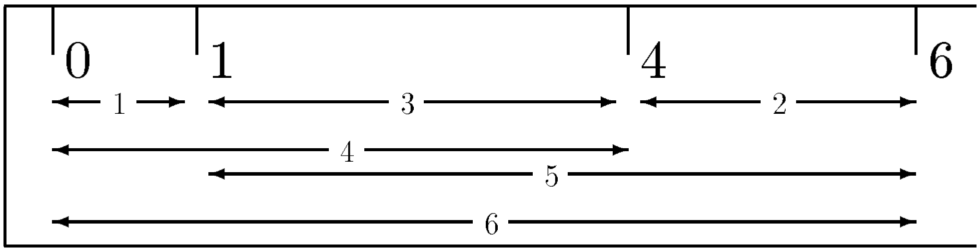

Figure A.2: A Golomb Ruler

Thus, distance 1 has to be at one end of the ruler and next to distance m - 1. The second order distance for this pair is thus m. Now, the distance 2 must also be next to m - 1 and at a corner since all distances from 1 to m are already present. This is possible only when there are only three first order distances or only four marks present in the ruler.

Thus, we cannot search for perfection in Golomb rulers with a higher number of marks. In this range, we search for what are termed '*Optimum* Golomb Rulers'. The second requirement these rulers must satisfy is that no ruler of shorter length and with the same number of marks can fulfill the first requirement. The absence of perfection implies that some distances (within the maximum limit) cannot be measured by the ruler. Let us look at a Golomb Ruler with five marks - 0, 1, 4, 9, 11. Let us draw the difference triangle of the ruler.

```
1     3      5     2
   4      8     7
      9      10
         11
```

The triangle indicates that the ruler satisfies the first requirement. The fact that the distance 6 is absent from the triangle precludes perfection. Since no shorter five mark ruler satisfies the first requirement, the ruler is an Optimum Golomb Ruler.

Another class of Golomb Rulers is *Non-Optimum* Golomb Rulers. These rulers satisfy the first condition but do not satisfy the optimality. i.e. there do exist shorter rulers with the same number of marks which satisfy the first condition.

Perfection or optimality is not unique in Golomb Rulers. In other words there may be more than one ruler (with the same number of marks), each of which satisfies the requirements of a Perfect or Optimal Golomb Ruler. Mirror images always provide a second option but totally different rulers may also satisfy the requirements.

A.1.4 Mirror Images

Mirror images are exactly what the name indicates. If we keep a mirror at the end of a ruler, an image will be formed in the mirror. In the image, the mark at n will be to the extreme left of the image. Let us make this mark the origin of the ruler. Since the origin is always at *0*, the positions of the marks will change with respect to the origin. The new position of each mark, originally at, say x, will now be at n - x. Thus, the rules to generate a mirror image of a ruler are:

1. Move the last mark at n to 0.

2. For every other mark x, generate the new mark using n-x.

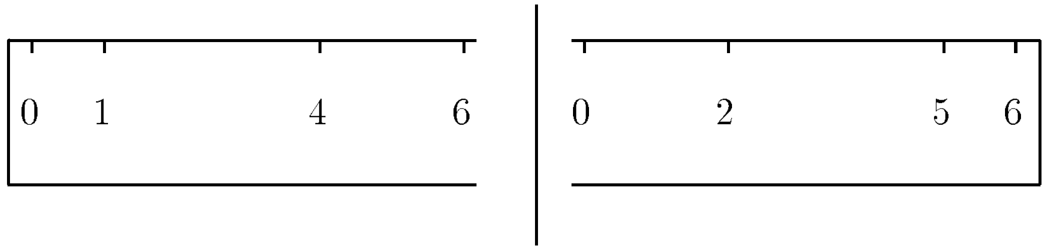

Figure A.3: A Mirror Image

Figure A.3 shows the mirror image of the four mark ruler used earlier. There is one point to note about mirror images. The first order differences are identical for mirror image pairs. This means that the set of distances that can be measured by each ruler is identical. This is important because no pair of rulers with greater than six marks have been found to have the property unless they are mirror images. This is true for both optimum and non-optimum rulers.

A.2 Important Terms

A lot of new concepts have been introduced in the preceding pages. Some of the important concepts, which will be used throughout the text are repeated here.

- Golomb Rulers. They are rulers in which no two pairs of marks have the same distance between them. i.e. the distance between any two marks on the ruler is unique.
- *Nth* Order Difference. A difference of order N indicates the distance between two marks having N - 1 marks between them. 1st order differences are also called *primary* differences.
- Perfect Golomb Rulers. These rulers are Golomb Rulers with m marks which have primary distances from 1 to m - 1. As a result they contain all distances from 1 to n (the length of the ruler). Perfect Golomb Rulers with greater than four marks cannot exist.
- Optimum Golomb Rulers. These rulers are Golomb Rulers with the added restriction that shorter Golomb Rulers with the same number of marks cannot exist. This class of Golomb Rulers is a superset of Perfect Golomb Rulers.
- Non-optimum Golomb Rulers. These rulers are Golomb Rulers which are neither perfect nor optimum. In other words, shorter Golomb Rulers with the same number of marks can be found.
- Mirror Images. A mirror image of a Golomb Ruler can be generated by calculating the new position of each mark using the formula ($n - old\,position$). Actually, a mirror image is exactly that, the image of the ruler in a mirror. The properties of mirror images are that they have identical first order differences with the result that they measure an identical set of differences.
- Golomb m. This form will be used throughout the text to represent an Optimum Golomb Ruler with m marks.

A.3 Uses of Golomb Rulers

The question that now arises is that what is the use of having a ruler with the above properties. Golomb Rulers are not very useful in measuring distances since the distance would already have to be known to predict the marks which would measure the distance. However, the fact that all distances are unique can be used in many diverse applications. Optimal rulers are important because in many applications, the rule is - the shorter, the better.

The applications of Golomb Rulers often have nothing to do with normal rulers. In these applications, the ruler is considered to be a set of ordered numbers with the properties required of a Golomb Ruler. Some of the important applications are disussed here briefly.

- Radio Systems. W.C.Babcock in [2] indicated that the rulers could be applied to the removal of 3rd order intermodulation interference to an acceptable level. The way to do this was, he said, to divide the frequency spectrum into channels of constant bandwidth. Then, allocate channels according to marks on a ruler. Further refinements were shown to remove 5th order interference too.

- Coding theory. Sets of difference triangles can be used to generate self-orthogonal codes which are important in communications [3]. The property of these sets is that the distances in all the triangles are distinct. As an example

$$(0,6,11,13),(0,8,17,18),(0,3,15,19)$$

  forms a (3,3) Difference Triangle Set (DTS). An (I,J)-DTS means that the set consists of I Golomb Rulers with J + 1 marks each (the mark at 0 is assumed and is not included in J). Further discussions on DTS's can be found in [13]. The codes have the property of limiting the propagation of any errors occurring during transmission or due to a malfunction in the encoder/decoder [20].

- X-Ray Crystallography. X-ray analysis of crystal structures is sometimes ambiguous because two distinct crystal structures produce identical X-ray diffraction patterns. The fact that no two Golomb Rulers (barring mirror images) measure exactly the same set of distances was used to resolve these inconsistencies. Even though one counterexample with six marks has been found [3], the hypothesis is still extensively used in the process.

- Computer Communication Network. Computer communication networks today specify both source and destination addresses so that routers may successfully deliver messages across the network. If the nodes on a network were allotted addresses according to the marks on a Golomb Ruler, only the difference between the addresses of the origin and destination will have to be transmitted. Since the differences are unique, together with the direction of arrival, will immediately identify both the origin and the destination node [3]. This application will probably become useful and will gain more importance once rulers with many more marks have been discovered.

- Linear Arrays. Often, many elements are arranged in a linear fashion to obtain an antenna. These arrays may be used to generate beams (in order to obtain the desired shape and frequency distribution) or to receive information (such as radio telescopes). Though the requirements and specifications for both applications are distinct, Golomb rulers are effective in increasing the efficiency of both. This is achieved by placing elements at point represented by marks on a Golomb ruler. In the first case, Perfect Golomb Rulers provide the highest possible resolution for a given aperture length. However, larger arrays (with more than four elements) do not have access to perfect Golomb rulers. Thus, *'Minimum Redundancy Linear Arrays'* [16] are used for deciding the placement. These arrays are a variation on Optimal Golomb Rulers in

No. of Marks	Length	No. of Marks	Length
1	0	10	55
2	1	11	72
3	3	12	85
4	6	13	106
5	11	14	127
6	17	15	151
7	25	16	177
8	34	17	199
9	44	18	216

Table A.1: Lengths of Known Optimum Golomb Rulers

that they may have distances repeated in them along with the absence of certain distances. The redundancy is kept at a minimum balanced with a preference for an array with a short total length. In the second case, the placement helps maximize the information collected for a fixed cost. Since the information in such a device is based mostly on the phase difference of the incoming signal between each pair of elements, placement at Golomb marks ensures that no piece of data will be duplicated [4].

Thus, Golomb Rulers have many practical and diverse applications. However, actual use depends upon further success in the search for larger optimum Golomb rulers.

A.4 The Search For Rulers

As already mentioned, Golomb Rulers were first discussed in 1952. This indicates that the search for rulers has been going on for a long time. But, the success achieved to date has been limited. Optimal rulers have been found only up to 18 marks in length. This is due mainly to the complexity involved in the search process.

A.4.1 Combinatorial Explosion

The concept of Golomb Rulers is pretty straightforward. But, to prove optimality, all possible combinations of shorter lengths have to be proved to be non-Golomb. This is further complicated by the fact that the length of optimal rulers increases very with an increase in the number of marks. This can be seen in table A.1 which tabulates the lengths of all known optimal Golomb rulers. Let us see the actual amount of work required for this process. Consider a ruler of length n with m marks. Further, let the ruler be optimal. The number of combinations (x_1) that have to be tested for any particular length k is equal to the number of ways $m - 1$ marks can be arranged in $k - 1$ positions (since the first mark is always at 0).

$$x_1 = C(k-1, m-1) \tag{A.1}$$

Also, the number of combinations tested would be minimum if no ruler of a longer length was considered. Thus, the total number of combinations (x_2) that would have to be tested would be

$$x_2 = \sum_{i=m}^{n} x_1 = \sum_{i=m}^{n} C(k-1, m-1) \tag{A.2}$$

Equation A.2 clearly shows that the increase in m (causing a corresponding increase in n) rapidly increases the number of combinations that have to be tested in order to prove optimality. To give

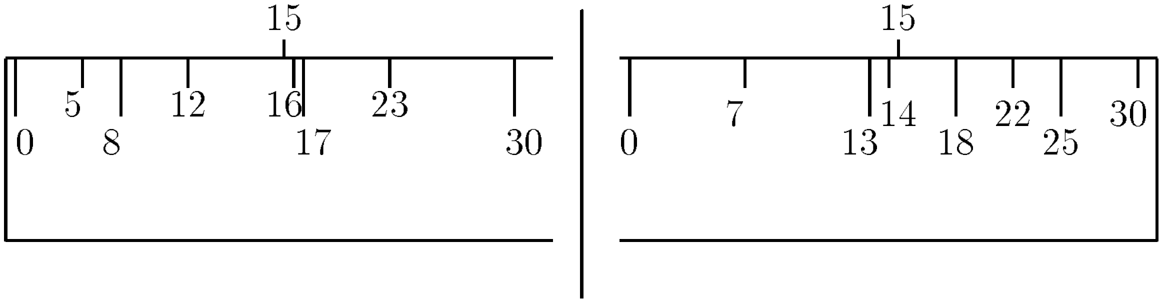

Figure A.4: An 8 mark ruler with middle marks on either side of the half length mark and its mirror image

an idea of the increase, proving optimality for Golomb 5 requires only 462 combinations to be tested while proving optimality for Golomb 11 requires the testing of approximately $2.4 * 10^{22}$ combinations. This is called *Combinatorial Explosion.*

Further, the testing of a combination is not a one step process. All the distances in the ruler have to be generated and the compared to each other. This requires $C(m, 2)$ subtractions to generate the differences and $C[C(m, 2), 2]$ comparisons to prove or disprove the first requirement of Golomb Rulers. (This is the maximum for a ruler and a value of half of this can be assumed to be the average case.)

Combinatorial explosion, both in the generation of combinations and their testing produces a problem which has not been solved with great success even today. The search for new ground in this field is beset with the problem of excessive computer time and other resources required.

A.4.2 The Process

The previous section gave information on the process involved in the search for Golomb Rulers. There are two processes running together. The first process generates the possible sequences of an m mark ruler. These sequences are then tested for the first property of Golomb Rulers by the second process. Optimality can be guaranteed by testing all possible sequences of a given length before moving on to the next length.

Equation A.2 was earlier used to show the number of sequences that had to be tested before finding the optimal Golomb ruler. That equation assumes that the whole search space has to be searched. Various search space reduction techniques can be used to reduce the numbers generated by the equation. Some of these techniques are explained below.

We know that mirror images of all rulers, whether Golomb or not, always measure the same distances. We are thus not interested in testing for both rulers. The problem is to keep mirror images from being generated in the first place. The solution to this is surprisingly simple. We force the middle mark of the ruler to stay on one side of the half length position. (The middle mark of a ruler with an even number of marks would be the one closer to the end. e.g. the middle mark of a ruler with 6 marks would be the 4th mark). For example, if we were testing rulers with 8 marks at length 30, we would force the 5th mark to stay at or above 15. (Forcing the mark to stay at or below 15 would have the same effect). This eliminates the generation of any mirror images for rulers with an odd number of marks (except those in which the middle mark is exactly at the half length position), reducing the search space almost in half. For a ruler with an even number of marks, this eliminates mirror images for all rulers but those in which the two middle marks (e.g. the 4th and

5th marks of a ruler with 8 marks) are on either side of the half length position. Such an example is shown in figure A.4. We can see that the rulers are mirror images but the 5th mark (middle mark for the rulers) in both rulers is beyond the halfway point (15 in this case since the total length of each ruler is 30). Both rulers will have to be generated and tested, even after using this search space reduction technique. Thus, for rulers with an even number of marks, this technique gives a search space reduction of slightly less than half.

The second search space reduction technique gives a lower improvement factor as compared to the first technique, but any help that can be obtained is useful. In the previous section, sequence generation had started by keeping all marks adjacent to each other and at the beginning of the spectrum. i.e. the marks for a ruler with m marks were initially arranged from 0 to $m-1$. But, we know that the kth mark cannot exist (in a Golomb ruler) at a position lower than the length of Golomb k. Thus, we can initialize all marks at positions indicated by table A.1. This means that all sequences having marks at lower positions do not have to be generated and tested. Further refinements to this search space reduction technique are explained in the text of W. Rankin's thesis[18].

The testing of sequences can also be made simpler than shown earlier. Our first choice was to construct the whole difference triangle requiring $C(m,2)$ subtractions. Then, we had to compare each item in the triangle with every other item requiring $\frac{C[C(m,2),2]}{2}$ comparisons at an average. The number of subtractions and comparisons can be substantially reduced by using a hashing array. The process is as follows. We have an array indexed by the difference. The array is initialized to zero. As the differences are generated, the array item indexed by the generated difference, is set. If for a certain difference, we see that the array item has already been set, we know that the difference has been repeated and the sequence need not be tested further. This reduces the required operations to $\frac{C(m,2)}{2}$ subtractions and $\frac{C(m,2)}{2}$ comparisons at an average. Further refinements in the technique are explained in [18].

The actual algorithms which were used to generate the Golomb rulers followed the above trend and utilized the search space reduction techniques.

A.4.3 Known Rulers

The realm of knowledge in Optimal Golomb rulers is not a vast one. As already stated, as of October 1993, optimality has been proved only for rulers up to 18 marks in length. The earliest set of rulers was published in 1952 by W.C. Babcock [2]. He published rulers up to 8 marks in length. It is not very clear as to who first published the ruler with 9 marks. John P. Robinson in 1979 [19][9] published rulers with 10, 11 and 12 marks. Douglas S. Robertson was responsible for discovering the ruler with 13 marks [4]. Finally, in 1990, J. B. Shearer published rulers with 14, 15 and 16 marks [21]. Sibert discovered rulers of 17 and 18 marks in yet unpublished results[22], the last advance made in the search. Table A.2 lists the rulers and other pertinent information.

Marks	Ruler	Discovered By	Date
3	0,1,3	W. C. Babcock	1952
4	0,1,4,6	W. C. Babcock	1952
5	0,1,4,9,11	W. C. Babcock	1952
6	0,1,4,10,12,17	W. C. Babcock	1952
7	0,1,4,10,18,23,25	W. C. Babcock	1952
8	0,1,4,9,15,22,32,34	W. C. Babcock	1952
9	0,1,5,12,25,27,35,41,44		
10	0,1,6,10,23,26,34,41,53,55	J. P. Robinson	1979
11	0,1,4,13,28,33,47,54,64,70,72	J. P. Robinson	1979
12	0,2,6,24,29,40,43,55,68,75,76,85	J. P. Robinson	1979
13	0,2,5,25,37,43,59,70,85,89,98,99,106	D. S. Robertson	
14	0,5,28,38,41,49,50,68,75,92,107,121,123,127	J. B. Shearer	1990
15	0,6,7,15,28,40,51,75,89,92,94,121,131,147,151,	J. B. Shearer	1990
16	0,1,4,11,26,32,56,68,76,115,117,134,150,163		
	168,177	J. B. Shearer	1990
17	0,8,31,34,40,61,77,99,118,119,132,143,147,		
	182,192,194,199	O. Sibert	1993
18	0,11,24,28,49,63,68,86,118,127,133,134,160,		
	163,194,206,214,216	O. Sibert	1993

Table A.2: Known Optimum Golomb Rulers