A parallel algorithm to solve the mathematical problem "double coset enumeration of $S_{24}$ over $M_{24}$"

Elena Yavorska Harris

Follow this and additional works at: http://scholarworks.lib.csusb.edu/etd-project

Part of the Theory and Algorithms Commons

Recommended Citation
Harris, Elena Yavorska, "A parallel algorithm to solve the mathematical problem "double coset enumeration of $S_{24}$ over $M_{24}$"" (2003). Theses Digitization Project. 2432.
http://scholarworks.lib.csusb.edu/etd-project/2432

This Thesis is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
A PARALLEL ALGORITHM TO SOLVE THE MATHEMATICAL PROBLEM

"DOUBLE COSET ENUMERATION OF S_{24} OVER M_{24}"

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Elena Yavorska Harris

December 2003
A PARALLEL ALGORITHM TO SOLVE THE MATHEMATICAL PROBLEM

"DOUBLE COSET ENUMERATION OF S_{24} OVER M_{24}"

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

by
Elena Yavorska Harris

December 2003

Approved by:

Ernesto Gomez, Chair, Computer Science

Zahid Hasan

Arturo Concepcion

Keith Schubert

12/1/2003
ABSTRACT

A few programs are known that perform double coset enumerations. However, these programs do not solve problems that involve very large permutation groups. A new parallel algorithm is presented and evaluated in this thesis. The algorithm computes all single cosets in the double coset $M_{24} \cdot P \cdot M_{24}$, where $P$ is a permutation on $n$ points of a certain cycle structure, and $M_{24}$ is the Mathieu group related to a Steiner system $S(5, 8, 24)$ as its automorphism group. The purpose of this work is not to replace the existing algorithms, but rather to explore a possibility to extend calculations of single cosets beyond the limits encountered when using currently available methods. Sequential and parallel programs that use this algorithm are written and tested. The performance of the sequential program is compared with the performance of a program that uses functions of Magma, one of the most common software applications for solving group theory problems, and the performance of the parallel program is evaluated. The results of the tests show that the proposed algorithm works slower for the cases when the number of single cosets in the double coset $M_{24} \cdot P \cdot M_{24}$ is less than 98, and faster if this number is equal to or greater than 98. Moreover, the
proposed algorithm allows this problem to be solved for the case when \( P \) is a permutation on 15 points of cycle structure [15], where Magma allows solving this program for the case when \( P \) is a permutation on 11 points of cycle structure [11].
ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who contributed to this research directly or indirectly. The support of the National Science Foundation under the award 9810708 is gratefully acknowledged.

I would like to acknowledge with utmost thanks and deep appreciation the continuous help of Drs. Hasan and Gomez, my primary advisors. Without their generosity, expertise and technical support this work would have been impossible. I also would like to most sincerely thank Drs. Concepcion and Schubert, members of my committee, for their insightful suggestions and help.

I would like to take this opportunity to express my profound gratitude to the faculty of the Department of Computer Science who contributed to my knowledge: Dr. Botting, Dr. Concepcion, Dr. Georgio, Dr. Gomez, Dr. Karant, Dr. Mendoza, Dr. Murphy, and Dr. Zemoudeh.

And last but not least, I would like to thank my husband, Keith Harris, and my mother, Vera Agapova, for their eternal support and encouragement.
6.2.2 Parallel Program .................... 43
6.3 Results ................................. 44
   6.3.1 Sequential Program ............... 44
   6.3.2 Parallel Program ................. 64
6.4 Time and Resources .................... 81

CHAPTER SEVEN: CONCLUSIONS AND FUTURE RESEARCH .... 85
APPENDIX A: SETS OF THE TYPE \( U_n \) ................. 87
APPENDIX B: MAGMA’S PROGRAM ................... 90
REFERENCES ....................................... 92
LIST OF TABLES

Table 1. Amount of Work for Sequential Program .......... 44
Table 2. Amount of Work in the Start Part ............... 45
Table 3. Comparison of Theoretical and Experimental Values of $W_{\text{work}}$ .......... 50
Table 4. $W_{\text{start}}$ and $W_{\text{start}_\text{Magma}}$ .......... 56
Table 5. $W_{\text{work}}$ and $W_{\text{work}_\text{Magma}}$ .......... 56
Table 6. $W_{\text{total}}$ and $W_{\text{total}_\text{Magma}}$ .......... 57
Table 7. Comparison of Running Time $T_{\text{seq}}$ and $T_{\text{Magma}}$ ....... 59
Table 8. Comparison of $T_{\text{start}}$ and $T_{\text{start}_\text{Magma}}$ .......... 60
Table 9. Comparison of $T_{\text{work}}$ and $T_{\text{work}_\text{Magma}}$ .......... 60
Table 10. Single Cosets for $U_{12}$, $U_{13}$, and $U_{14}$ .......... 63
Table 11. $T_{\text{exc}}$ in Seconds, Parallel Program .......... 65
Table 12. Speedup($n$, $N_p$) .......... 66
Table 13. Efficiency, $\varepsilon(n$, $N_p)$ .......... 66
Table 14. $W_{\text{total}_\text{per}_\text{proc}}$ for $U_{13}$ .......... 68
Table 15. Amount of Work and Number of Messages with $N_p = 2$ for $U_{7}$, ..., $U_{14}$ .......... 70
Table 16. Estimated Work and Number of Messages for $U_{7}$, ..., $U_{20}$ .......... 73
Table 17. Running Time with Fixed Number of Calculated Single Coset Representatives for $U_{19}$, $U_{19}$, and $U_{20}$ .......... 76
Table 18. $t_{\text{real}}$, Speedup, and $\varepsilon$ for $U_{11}$, ..., $U_{20}$ and $N_p = 2$ .......... 77
Table 19. Speed\textsubscript{10\textsuperscript{-3}} sec/single coset) for U\textsubscript{15},..., U\textsubscript{20} s ........................ 82

Table 20. Estimated T\textsubscript{exc} in Days ........................ 83

Table 21. Estimated T\textsubscript{exc} in Years ........................ 83

Table 22. Hard Disk Space in GB Needed for Output ........................ 84
LIST OF FIGURES

Figure 1. Data Partitioning and Data Structures . . . . 32
Figure 2. Message Passing . . . . . . . . . . . . . . . . 34
Figure 3. Pseudocode for the Start Part . . . . . . . . 36
Figure 4. Pseudocode for the Work Part . . . . . . . . 37
LIST OF GRAPHS

Graph 1. Start Work as Function of Stabilizer's Order .................. 46
Graph 2. Order Comparison: n!, (n-1)!, and W_{total}(n) ........ 53
Graph 3. W_{start} and W_{start,Magma} .......................... 57
Graph 4. W_{work} and W_{work,Magma} .............................. 58
Graph 5. W_{total} and W_{total,Magma} .............................. 58
Graph 6. Comparison of Running Time T_{seq} and T_{Magma} ....... 59
Graph 7. Comparison of T_{start} and T_{start,Magma} ............. 61
Graph 8. Comparison of T_{work} and T_{work,Magma}
for U_6, U_7, and U_8 ............................................. 62
Graph 9. Comparison of T_{work} and T_{work,Magma}
for U_8, U_9, and U_10 ............................................. 62
Graph 10. W_{total_per_proc} for U_{13} as a Function of N_p ....... 69
Graph 11. Work Decrease and Message Increase .................... 71
Graph 12. Estimated Work and Messages for N_p = 2 ............ 74
Graph 13. Efficiency as Function of Stabilizer's Order .......... 78
CHAPTER ONE

INTRODUCTION

Many researchers have shown an interest in double cosets within computational group theory. However, in general, problems such as finding all double cosets in a large permutation group remains unsolved.

One of the most common software applications for solving group theory problems is Magma, which allows users to find all single coset representatives in the double coset $\mathbb{M}_{24} P \mathbb{M}_{24}$, where $P$ is a permutation on $n$ points of a certain cycle structure, and $\mathbb{M}_{24}$ is the Mathieu group related to a Steiner system $S(5, 8, 24)$ as its automorphism group. There is a limitation to such calculations. For example, if $P$ is a permutation of the cycle structure $[n]$, then Magma performs calculations for $n$ less than or equal to 11. The limitation on calculating the single coset representatives for a larger $n$ is due to the limitation of RAM.

Advances in modern technology and the availability of parallel programming enable us to produce faster programs for problems that involve large amounts of calculations. The objective of this research is to investigate the
possibilities for computing single coset representatives in a double coset in parallel.

The proposed algorithm uses techniques that allow the solution of the space problem during run time; however, the resulting file with calculated single coset representatives still requires many gigabytes of storage. The parallel approach of the proposed algorithm is implemented and tested with a set of permutations, each of which is of the cycle structure \([n]\), where \(n\) varies from 6 to 14. This research provides an evaluation of the test results along with an estimation of run time and the space resources needed to find the single coset representatives for larger \(n\).

1.1 Background

The problem considered in this thesis, double coset enumeration of \(S_{24}\) over \(M_{24}\), is a problem belonging to Group Theory. \(S_{24}\) is the symmetric group of degree 24, and the Mathieu group \(M_{24}\) is a simple sporadic group, related to a Steiner system \(S(5, 8, 24)\) as its automorphism group. A Steiner system \(S(5, 8, 24)\) is a collection of 8-element subsets of a 24-element set such that any 5 elements of the 24 elements belong to exactly one 8-element subset.
The Mathieu group $M_{24}$ is one of the first simple sporadic groups that were discovered by Emile Leonard Mathieu, a French mathematician. $M_{24}$ plays an important role in the discovery of simple groups since it is involved in 20 out of 26 sporadic simple groups. Studying the Mathieu group $M_{24}$ helps with the study of sporadic simple groups.

Another important aspect of the Mathieu group $M_{24}$ is the place it takes in the field of coding theory. The Mathieu group $M_{24}$ is the Automorphism group of the binary Golay code (the group of permutations of the set of points from 1 to 24 that sends codewords to codewords).

The double coset enumeration of $S_{24}$ over $M_{24}$ gives a better understanding the geometrical structure of the Mathieu group $M_{24}$.

There is a one-to-one correspondence between the orbits of $S_{24}$ on the ordered pairs of Steiner systems, the orbits of $M_{24}$ on the set of all Steiner systems, and the double cosets $M_{24} \times M_{24}$ in $S_{24}$. An orbit of $M_{24}$ on the set of all Steiner systems, $T$, is $\{S_a | a \in M_{24} \times M_{24}, S \in T\}$. A double coset $M_{24} \times M_{24}$ is $\{M_{24} \times m | x \in S_{24}, \text{ and } m \in M_{24}\} = \{Mm^{-1} \times m | x \in S_{24}, \text{ and } m \in M_{24}\} = \{M^{-1} \times m | x \in S_{24}, \text{ and } m \in M_{24}\}$. 
1.2 Previous Work Done

The history of enumerating cosets goes back to 1936 when Todd and Coxeter provided "a practical method for enumerating cosets of a finite abstract group" in [9]. In 1973, Cannon et al. described the Todd-Coxeter lookahead algorithm and its implementation and analysis in [3]. A report of the basic techniques with worked examples was given by Leech in 1984, in [6]. Given defining relations for a group G and generators of its subgroup H of finite index, applications of the Todd-Coxeter algorithm enumerate the cosets of H in G using a coset multiplication table. The limitation of these early implementations of Todd-Coxeter is the space required to store the coset table in computer memory. The need to overcome this limitation has inspired researchers to find new methods of calculating coset enumeration. As a result, in 1984, Conway offered to use enumerating double cosets of subgroups H and K in a group G as a technique of significant space-saving in his paper [4]. And later, in 1991, Linton implemented this technique and reported the results in [7]. The latter author provided the description of single and double coset enumeration, as well as implementation techniques. Linton pointed out that the main restriction on a group K is the
space required to store look-up tables that describe the structure of K. The double coset table is used to keep track of double coset representatives. This double coset enumeration technique later was incorporated into GAP and Magma.

Another approach to finding canonical double coset representatives, given a permutation group G acting on a set S with a base b, subgroups H and K of G given by a base, a strong generating set, and an element g of G, is provided in [2]. This algorithm is restricted to permutation groups of degree in the thousands.

The algorithm to find double coset representatives that is used in this thesis differs significantly from all the previously described work since it uses a modified direct approach of calculating single coset representatives in a double coset. The most relevant paper that describes a direct approach of calculating double cosets is [1], where Butler describes Dimino's algorithm that uses an explicitly stored list of elements of a group. The restriction of this algorithm is the size of secondary storage and the time required to access and search stored elements. The author states that this algorithm is restricted to very small groups.
The following is the organization of this thesis.

Chapter 2 gives basic definitions and examples of the related concepts in computational group theory.

Chapter 3 describes the problem of double coset enumeration and specifies a straightforward procedure that solves this problem.

Chapter 4 provides the description of a new algorithm and some techniques of its implementation.

Chapter 5 discusses the parallel algorithm.

Chapter 6 provides the results, analysis and evaluation of the tests obtained by running sequential and parallel programs. The rough estimation of run time and space usage required to run the same program for cycles of larger length is provided as well.

Chapter 7 provides conclusions and suggestions for future research.
CHAPTER TWO
DEFINITIONS AND EXAMPLES

This chapter provides the definitions and examples necessary for understanding the remainder of this work. The reference that is used for the definitions and examples is Rotman [8].

2.1 Permutation. Let \( Y \) be a non-empty set. A bijective mapping (one-one and onto) \( f: Y \rightarrow Y \) is called a permutation of \( Y \).

Example.
Let \( Y = \{1, 2, 3\} \), then \( f = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix} \)
is a permutation of \( Y \), where \( f(1) = 3, f(2) = 1, \) and \( f(3) = 2; \)

2.2 Symmetric Group. Let \( S \) be a set of \( n \) elements. The set of all permutations of \( S, S_n, \) forms a group under the operation of composition of mappings. \( S_n \) is known as the symmetric group of degree \( n \).

2.3 Order of a Group. The order of a group is the number of elements in the group.

2.4 K-cycle. A permutation \( \alpha \) in \( S_n \), such that \( \alpha(i_1) = i_2, \alpha(i_2) = i_3, \ldots, \) and in general \( \alpha(i_{r-1}) = i_r, \) and \( \alpha(i_k) = \alpha(i_1) \) is called a \( k \)-cycle and denoted by
\[ \alpha = \{i_1, i_2, \ldots, i_k\} \] or
\[ \alpha = (i_1 i_2 \ldots i_k). \]

**Example.**

\[ \beta = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix} \]
and when written as a 3-cycle, \( \beta = (1 \ 3 \ 2). \)

2.5 **Length of a Cycle.** The length of a \( k \)-cycle is \( k. \)

2.6 **Disjoint Cycle.** Two cycles are disjoint if they have no elements in common.

**Example.**

Let \( \alpha_1 = (1 \ 2 \ 3), \) and \( \alpha_2 = (2 \ 4 \ 5). \) Then \( \alpha_1 \) and \( \alpha_2 \) are not disjoint since they have the element "2" in common.

Let \( \beta_1 = (1 \ 2 \ 3), \) and \( \beta_2 = (4, 5, 6). \) Then \( \beta_1 \) and \( \beta_2 \) are disjoint cycles.

2.7 **Product of Two Cycles.** Let \( \alpha = (i_1 \ i_2 \ldots \ i_k) \) and \( \beta = (j_1 \ j_2 \ldots \ j_k) \) be two cycles, where \( i_r \) and \( j_t \) belong to \( S, \) and not all \( i_r \) and \( j_t \) are distinct. Then the product of these two cycles, \( \gamma, \) is found by the following procedure.

**Step 1.** Form a list of the elements of the given cycles, \( A = (i_1, i_2, \ldots, i_k, j_1, j_2, \ldots, j_k). \)

**Step 2.** Choose one element from \( A, \) say \( i_1, \) and it is the first element of \( \gamma, \) i.e. so far, \( \gamma = (i_1). \)

**Step 3.** Find \( a = \beta(i_1), \) and then find \( b = \alpha(a); \) \( b \) is the next element of \( \gamma, \) so now \( \gamma = (i_1, b). \)
Step 4. Next, find $c = \beta(b)$ (to which element $\beta$ sends $b$), and find $d = \alpha(c)$ (to which element $\alpha$ sends $c$); $d$ is the next element of $\gamma$, so now $\gamma = (i_1, b, d)$.

Continue until the last element of $\gamma$ is sent to the first element of $\gamma$; when that condition occurs, close the right parenthesis, open the left parenthesis, and start with an element from $A$ that is not in $\gamma$. Continue until all the elements of $A$ have participated in this procedure.

Example.
Let $\alpha = (1 \ 3 \ 5 \ 4)$ and $\beta = (2 \ 1 \ 6 \ 3)$. Find their product.

$A = \{1, 2, 3, 4, 5, 6\}$. $\gamma = \alpha \ast \beta = (1)$

$\beta(1) = 6$, $\alpha(6) = 6$ ($\alpha$ does not have 6, so it fixes 6), and $\gamma = (1, 6)$

$\beta(6) = 3$, $\alpha(3) = 5$, and $\gamma = (1, 6, 5)$

$\beta(5) = 5$, $\alpha(5) = 4$, and $\gamma = (1, 6, 5, 4)$

$\beta(4) = 4$, $\alpha(4) = 1$, but 1 is the first element of $\gamma$, so close the right parenthesis, and $\gamma = (1, 6, 5, 4)$. Pick the element from $A$ that is not in $\gamma$: 2, and start the left parenthesis: $\gamma = (1, 6, 5, 4)(2)$

$\beta(2) = 1$, $\alpha(1) = 3$, and $\gamma = (1, 6, 5, 4)(2, 3)$

$\beta(3) = 2$, $\alpha(2) = 2$, and $\gamma = (1, 6, 5, 4)(2, 3)$

Check whether all the elements of $A$ are in $\gamma$. They are, so stop. And the result is
\[ \gamma = \alpha^* \beta = (1 \, 3 \, 5 \, 4)^* (2 \, 1 \, 6 \, 3) = (1, \, 6, \, 5, \, 4)(2, \, 3). \]

2.8 Inverse of a Cycle. Let \( \beta = (1 \, 2 \, 3 \ldots \, k) \) cycle. Then the inverse of \( \beta \) is \( \beta^{-1} = (k \ldots \, 3 \, 2 \, 1) \), and the product of \( \beta \) and \( \beta^{-1} \) is the identity.

Example.

Let \( \beta = (1 \, 3 \, 4 \, 6) \). Then \( \beta^{-1} = (6 \, 4 \, 3 \, 1) \), and
\[
\beta \ast \beta^{-1} = (1 \, 3 \, 4 \, 6) \ast (6 \, 4 \, 3 \, 1) = (1)(3)(4)(6) = 1
\]

2.9 Product of a Form \( x^m \). \( x^m = m \ast x \ast m^{-1} \), where \( x \) and \( m \) are cycles. \( m \ast x \ast m^{-1} \) is obtained from \( x \) by applying \( m \) to the symbols of \( x \). \( m \ast x \ast m^{-1} \) is a conjugate of \( x \).

Example.

Let \( x = (1, \, 3, \, 4, \, 5) \), and \( m = (2, \, 5, \, 1, \, 4) \).

Then \( m \ast x \ast m^{-1} = (m(1), \, m(3), \, m(4), \, m(5)) = (4, \, 3, \, 2, \, 1) \)

2.10 Steiner System \( S(5, \, 8, \, 24) \). The Steiner system \( S(5, \, 8, \, 24) \) is a collection of 8-element subsets of a 24 element set, \( \Omega \), such that any 5 of 24 elements belong to exactly one 8-element subset. There are 759 elements in \( S \).

The set of all Steiner systems with parameters \( (5, \, 8, \, 24) \) is denoted as \( T \).

2.11 Homomorphism, Isomorphism, and Automorphism. Let \( G \) together with an operation \( + \) be a group, and \( H \) together with an operation \( \ast \) be a group.
A mapping \( f: G \rightarrow H \) is called a homomorphism if
\[
f(g+h) = f(g) * f(h), \text{ where } g, \text{ and } h \in G.
\]

A mapping \( f: G \rightarrow H \) is called an isomorphism if it is a homomorphism and one-one, and onto.

A mapping \( f: G \rightarrow G \) is called an automorphism if it is an isomorphism.

**2.12 Mathieu Group \( M_{24} \).** The set of all automorphism of a group \( G \) is a group denoted by \( \text{Aut}(G) \). \( \text{Aut}(G) \) is a subgroup of \( S_6 \), the group of all permutations of \( G \).

The Mathieu group \( M_{24} \) is \( \text{Aut}(S(5, 8, 24)) \).

**2.13 Coset.** Let \( G \) be a group and \( H \) be a subgroup in \( G \). Then a left coset of \( H \) in \( G \) is the set \( gH \), where \( g \in G \). Similarly a right coset of \( H \) in \( G \) is \( Hg \), where \( g \in G \).

**Example.**
Let \( G \) be a group, \( H = \{1, h_1, h_2, h_3\} \) be a subgroup in \( G \), and \( g \in G \). Then \( gH = \{g, gh_1, gh_2, gh_3\} \).

**Theorem.**
Two cosets are either identical or do not have elements in common.

**2.14 Double Coset.** Let \( G \) be a group and \( H \) be a subgroup in \( G \). Then a double coset of \( H \) in \( G \) is \( H \ g \ H \), where \( g \in G \) is called a double coset representative.
Example

Let $x \in S_{24}$, and $M = M_{24}$. Then the double coset of $M_{24}$ in $S_{24}$ is $M \times M = \{Mx^m \mid m \in M\}$, where $x$ is a double coset representative.

2.15 Double Coset Enumeration of $S_{24}$ over $M_{24}$. The double coset enumeration of $S_{24}$ over $M_{24}$ is the problem of finding all single coset representatives in the double coset $M_{24} \times M_{24}$, where $x$ is in $S_{24}$.

2.16 Stabilizer of a Cycle in $M_{24}$. A stabilizer $St$ of a cycle $x$ in $M_{24}$ is the set of all elements $m$ of $M_{24}$ such that $x^m$ has only the symbols of $x$ in it.

Example.

Let $\alpha = (1 \ 2 \ 4 \ 5)$ be a cycle, and $m = (1 \ 6 \ 3 \ 7)$. Then $\alpha^m = (m(1), m(2), m(4), m(5)) = (6 \ 2 \ 4 \ 5)$ has the symbol "6" in it, which is not in $\alpha$; therefore, the element $m = (1 \ 6 \ 3 \ 7)$ is not in the stabilizer of $\alpha$.

Let $m_1 = (1 \ 4 \ 2 \ 5)(6 \ 3 \ 7)$. Then $\alpha^{m_1} = (m_1(1), m_1(2), m_1(4), m_1(5)) = (4 \ 5 \ 2 \ 1)$ has only the symbols that are in $\alpha$; therefore, $m_1$ is in the stabilizer of $\alpha$.

2.17 k-transitive. Let $G$ be a permutation group on a set $S$. $G$ is $k$-transitive on $S$ if for every pair of $k$-tuples of distinct elements of $S$, say $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$, where $x_i$ and $y_i$ are elements of $S$, there exists a
permutation $g$ in $G$ that takes $x_i$ to $y_i$, i.e. $g(x_i) = y_i$, for $i = 1, \ldots, k$.

$M_{24}$ is 5-transitive on $\Omega = \{1, 2, \ldots, 24\}$.

2.18 Special Sets $U_n$, $S_n$, and $T_n$. Let $\Omega = \{1, 2, 3, \ldots, 24\}$. The subsets of $\Omega$ fall into 49 orbits under the action of $M_{24}$ [7]. These 49 orbits are of the three types: $S_n$, $U_n$, and $T_n$.

"In general, a set of cardinal $n \leq 12$ is called special ($S_n$) if it contains or is contained in a special octad, otherwise umbral ($U_n$) if it is contained in an umbral dodecad, and tranverse ($T_n$) if not... Sets of more than 12 points are described by the same adjectives as their complements." [7]
CHAPTER THREE
PROBLEM DESCRIPTION

Given $U_n$ set and its stabilizer in $M_{24}$, find all single coset representatives (one for each orbit) of a double coset $M_{24} \times M_{24}$, where $x$ is a cycle formed from the elements of $U_n$. The following procedure describes the process of finding the required single coset representatives.

**Step 1.** The stabilizer of $U_n$ in $M_{24}$, $St$, is found using Magma.

**Step 2.** The set of all permutations of $U_n$ in $S_{24}$, $P_n$, is found.

**Step 3.** An element $\alpha$ of $P_n$ is conjugated by the stabilizer $St$ forming a single coset $O$, i.e., $\alpha^m$ is found for all $m$ in $St$. $O = \{\alpha^m \mid m \in St\}$.

**Step 4.** $O$ is subtracted from $P_n$. $P_n = P_n \setminus O$.

**Step 5.** Steps 3 and 4 are repeated until $P_n$ is empty.

The set of all elements $\alpha$ of $P_n$, for which single cosets $O$ were calculated in the step 3, is the set of desirable single coset representatives.

The following example illustrates the implementation of the steps described above for an imaginary $U_5$ and an imaginary stabilizer $St$ of $U_5$ in $M_{24}$.
Example.

Suppose \( U_5 = \{1, 2, 3, 4, 5\} \) is given. The set of all single coset representatives \( Q \) must be found. Let \( u = (1 2 3 4 5) \) be a cycle formed from the elements of \( U_5 \). The stabilizer \( St \) of \( U_5 \) in \( M_{24} \) is found by calculating \( g = u^m \) for all \( m \) in \( M_{24} \). \( g = (m(1), m(2), m(3), m(4), m(5)) \). If any element of the \( m(1), m(2), m(3), m(4), m(5) \) does not belong to \( U_5 \), then \( g \) is not in the \( St \). The assumption is made that the order of the \( St \) is 8.

The set of all permutations of \( U_5 \) is

\[
P_5 = \left\{ (1 2 3 4 5), (1 3 2 4 5), (1 4 2 3 5), (1 5 2 3 4), \\
(1 2 3 5 4), (1 3 2 5 4), (1 4 2 5 3), (1 5 2 4 3), \\
(1 2 4 3 5), (1 3 4 2 5), (1 4 3 2 5), (1 5 3 2 4), \\
(1 2 4 5 3), (1 3 4 5 2), (1 4 3 5 2), (1 5 3 4 2), \\
(1 2 5 3 4), (1 3 5 2 4), (1 4 5 2 3), (1 5 4 2 3), \\
(1 2 5 4 3), (1 3 5 4 2), (1 4 5 3 2), (1 5 4 3 2) \right\}
\]

Note, that there are \( 5! = 120 \) permutations of \( U_5 \), but some of them are identical in terms of cycles. For example, the cycles \( (1 2 3 4 5) \) and \( (3 4 5 1 2) \) are equivalent. That is why when all the permutations of \( U_5 \) are calculated, only one of the equivalent cycles are taken. Therefore, there are \( (5-1)! = 24 \) permutations of \( U_5 \) in total.

To calculate a single coset \( O_\alpha \) for an element of \( P_5 \), the cycle \( \alpha = (1 2 3 4 5) \) is chosen from \( P-5 \).
Then \( O_\alpha = \{ \alpha^m \mid m \in S \}. \)

Let \[
O_\alpha = \left\{ (1 \ 2 \ 3 \ 4 \ 5), (1 \ 2 \ 4 \ 5 \ 3), (1 \ 3 \ 4 \ 2 \ 5), \\
(1 \ 3 \ 5 \ 4 \ 2), (1 \ 4 \ 3 \ 2 \ 5), (1 \ 4 \ 5 \ 2 \ 3), \\
(1 \ 5 \ 2 \ 3 \ 4), (1 \ 5 \ 3 \ 4 \ 2) \right\}
\]

The single coset \( O_\alpha \) must be removed from \( P_5 \). The highlighted elements of \( P_5 \) are the elements of \( O_\alpha \). See below:

\[
P_5 = \left\{ (1 \ 2 \ 3 \ 4 \ 5), (1 \ 3 \ 2 \ 4 \ 5), (1 \ 4 \ 2 \ 3 \ 5), (1 \ 5 \ 2 \ 3 \ 4), \\
(1 \ 2 \ 3 \ 5 \ 4), (1 \ 3 \ 2 \ 5 \ 4), (1 \ 4 \ 2 \ 5 \ 3), (1 \ 5 \ 2 \ 4 \ 3), \\
(1 \ 2 \ 4 \ 3 \ 5), (1 \ 3 \ 4 \ 2 \ 5), (1 \ 4 \ 3 \ 2 \ 5), (1 \ 5 \ 3 \ 2 \ 4), \\
(1 \ 2 \ 4 \ 5 \ 3), (1 \ 3 \ 4 \ 5 \ 2), (1 \ 4 \ 3 \ 5 \ 2), (1 \ 5 \ 3 \ 4 \ 2), \\
(1 \ 2 \ 5 \ 3 \ 4), (1 \ 3 \ 5 \ 2 \ 4), (1 \ 4 \ 5 \ 2 \ 3), (1 \ 5 \ 4 \ 2 \ 3), \\
(1 \ 2 \ 5 \ 4 \ 3), (1 \ 3 \ 5 \ 4 \ 2), (1 \ 4 \ 5 \ 3 \ 2), (1 \ 5 \ 4 \ 3 \ 2) \right\}
\]

After subtracting \( O_\alpha \) from \( P_5 \), \( P_5 \) is:

\[
P_5 = \left\{ (1 \ 2 \ 3 \ 5 \ 4), (1 \ 3 \ 2 \ 4 \ 5), (1 \ 4 \ 2 \ 3 \ 5), (1 \ 5 \ 2 \ 4 \ 3), \\
(1 \ 2 \ 4 \ 3 \ 5), (1 \ 3 \ 2 \ 5 \ 4), (1 \ 4 \ 2 \ 5 \ 3), (1 \ 5 \ 3 \ 2 \ 4), \\
(1 \ 2 \ 5 \ 3 \ 4), (1 \ 3 \ 4 \ 5 \ 2), (1 \ 4 \ 3 \ 5 \ 2), (1 \ 5 \ 4 \ 2 \ 3), \\
(1 \ 2 \ 5 \ 4 \ 3), (1 \ 3 \ 5 \ 2 \ 4), (1 \ 4 \ 5 \ 3 \ 2), (1 \ 5 \ 4 \ 3 \ 2) \right\}
\]

\( \alpha \) is the first found single coset representative of the set \( Q \). Now, \( Q = \{ \alpha \} \).

The next cycle from \( P_5 \) is chosen, say \( \beta = (1 \ 2 \ 3 \ 5 \ 4) \), and the next single coset \( O_\beta \) is calculated as \( \{ \beta^m \mid m \in S \} \).
Assume that the highlighted elements of \( P_5 \) below are the elements of \( O_\beta \).
After subtracting \( O_\beta \) from \( P_5 \), \( P_5 \) is:

\[
P_5 = \left\{ (1 \, 2 \, 3 \, 4 \, 5), \ (1 \, 3 \, 2 \, 4 \, 5), \ (1 \, 4 \, 2 \, 3 \, 5), \ (1 \, 5 \, 2 \, 4 \, 3), \right. \\
\left. (1 \, 2 \, 4 \, 3 \, 5), \ (1 \, 3 \, 2 \, 5 \, 4), \ (1 \, 4 \, 2 \, 5 \, 3), \ (1 \, 5 \, 3 \, 2 \, 4), \right. \\
\left. (1 \, 2 \, 5 \, 3 \, 4), \ (1 \, 3 \, 4 \, 5 \, 2), \ (1 \, 4 \, 3 \, 5 \, 2), \ (1 \, 5 \, 4 \, 2 \, 3), \right. \\
\left. (1 \, 2 \, 5 \, 4 \, 3), \ (1 \, 3 \, 5 \, 2 \, 4), \ (1 \, 4 \, 5 \, 3 \, 2), \ (1 \, 5 \, 4 \, 3 \, 2) \right\}
\]

The cycle \( \beta \) is the next single coset representative of the set \( Q \). \( Q = \{ \alpha, \beta \} \).

The next cycle is chosen from \( P_5 \), say \( \gamma = (1 \, 2 \, 4 \, 3 \, 5) \). The single coset \( O_\gamma \) is calculated as \( \{ \gamma^m \mid m \in \text{St-5} \} \). \( \gamma \) is the next single coset representative that is put into \( Q \). \( Q = \{ \alpha, \beta, \gamma \} \). Suppose \( O_\gamma \) has the same elements that are left in \( P_5 \). Then after the subtraction of \( O_\gamma \) from \( P_5 \), \( P_5 \) does not have any elements, and the process of finding all orbit representatives of \( M_{24} \) in the set of all permutations of \( U_5 \) is completed. The resulting set is

\[
Q = \{ \alpha, \beta, \gamma \} = \{(1 \, 2 \, 3 \, 4 \, 5), \ (1 \, 2 \, 3 \, 5 \, 4), \ (1 \, 2 \, 4 \, 3 \, 5) \}. 
\]
CHAPTER FOUR

NEW ALGORITHM

In this chapter, a review of a new algorithm is given, as well as a description of the implementation of the main functionalities.

4.1 Overview

A new algorithm is offered to find all single coset representatives in the double coset $M_{24} \times M_{24}$, where $x$ is a cycle formed from the elements of $U_n$. The algorithm has two main steps, or functionalities: calculating a single coset (Subsection 4.2.1), and subtracting this single coset from the set of all permutations of $U_n$ (Subsection 4.2.2).

The implementation of these functions is provided in this section. Also, some techniques are used to speed up run time or to save space. At the beginning of this section, the explanation of the techniques is presented, followed by a description of implementation of the main functionalities.

4.1.1 Enciphering

An input to the problem is a set of the type $U_n$ and the stabilizer of a cycle formed from elements this set in $M_{24}$. 
Each element of the stabilizer is a product of disjoint cycles. For example, \( m = (1, 4, 22, 6)(2, 16, 15, 8)(3, 20, 13, 12)(5, 14, 23, 24)(7, 18, 19, 11)(9, 10, 21, 17) \in M_{24} \). 

\( U_n \) is a special set of \( n \) elements, where each element belongs to \( \Omega \), the set of 24 elements, \( \Omega = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24\} \). To perform any necessary calculations, the elements of \( U_n \) and the elements of its stabilizer are placed into data structures as arrays or vectors of integers. As was described above, the set of all permutations of \( U_n \) is calculated and kept in the second storage. To save a single \( n \)-cycle in a file, at least \( 2n \) bytes are required (one byte is required to store an integer, and 1 byte is needed to separate two integers). If integers are replaced with characters, then \( n \) bytes are required to save one \( n \)-cycle in a file (characters do not need to be separated for the purpose of reading from a file). That is why one-to-one and onto mapping is used to convert integers to characters.

\[
\begin{align*}
1 & \leftrightarrow a \\
2 & \leftrightarrow b \\
3 & \leftrightarrow c \\
4 & \leftrightarrow d \\
5 & \leftrightarrow e \\
6 & \leftrightarrow f \\
7 & \leftrightarrow g \\
8 & \leftrightarrow h \\
9 & \leftrightarrow i \\
10 & \leftrightarrow j \\
11 & \leftrightarrow k \\
12 & \leftrightarrow l \\
13 & \leftrightarrow m \\
14 & \leftrightarrow n \\
15 & \leftrightarrow o \\
16 & \leftrightarrow p \\
17 & \leftrightarrow q \\
18 & \leftrightarrow r \\
19 & \leftrightarrow s \\
20 & \leftrightarrow t \\
21 & \leftrightarrow u \\
22 & \leftrightarrow v \\
23 & \leftrightarrow w \\
24 & \leftrightarrow x
\end{align*}
\]

The character “z” represents the identity of \( M_{24} \). This mapping is used to encipher elements of stabilizers of \( U_n \) as well as \( U_n \).
For example, \( m = (1, 4, 22, 6)(2, 16, 15, 8)(3, 20, 13, 12)(5, 14, 23, 24)(7, 18, 19, 11)(9, 10, 21, 17) \) is enciphered into \( m’ = \text{advfabpohbctmlcenwxegrskgijuqi} \). The fact that \( m \) in \( M_{24} \) is a product of disjoint cycles helps to write \( m \) as a single string. To show that the last element of a cycle goes to the first element, when enciphering, the character ‘\( \)’ is enciphered into the first character of this cycle. To check the correctness of such a representation, one can test out where a middle element of a cycle of \( m \) is sent, as well as to which symbol the element at the end of a cycle is sent in this string. For example, \( m(4) = 22 \). Find the corresponding character in \( m’ \), \( 4 \rightarrow d \). Find where \( d \) is sent in \( m’ \), look for \( d \) in \( m’ \) starting from the left of the string. \( m’(d) = v \). Find the integer corresponding to \( v \), \( v \rightarrow 22 \). So, \( m’ \) sends \( 4 \) to \( 22 \) as required. Pick the last element of a cycle, \( m(8) = 2 \); \( 8 \rightarrow h \); \( m’(h) = b \); \( b \rightarrow 2 \); \( m’(8) = 2 \) as required. A conversion of integers to characters not only saves space, but also significantly improves the run time of calculations since strings are processed faster than vectors of integers.

4.1.2 Imaginary Set of Permutations

One part of the problem is to find the set of all permutations of \( U_n \). The size of the set of all permutations
is \((n-1)\)!. Each permutation is represented as a string of \(n\) characters, so the total space needed to save the set is \([\(n-1\)!](n+1)\) bytes. For example, the set for \(U_{13}\) requires 6.7 GB (one permutation consists of 14 characters, and one character takes one byte), and the set for \(U_{24}\) needs 6.46\(\times10^{14}\) GB. To solve the space problem, a so-called imaginary set of permutations is used in the calculations. The set of permutations has a recursive pattern, so it is possible to calculate the position that a particular permutation takes in an imaginary file of permutations, and it is possible to calculate from the given position the corresponding permutation. So, instead of keeping a file of permutations in a second storage, this program uses the calculations of permutations from positions and vise versa. A similar technique is provided with C++ function libraries; it is "next_permutation()" call of the <algorithm> library. "next_permutation()" calculates a permutation given the original string of characters and the position of this permutation relative to the original string [10, p. 545]. This function call cannot be used in this program, since the call works with \(n!\) permutations of the string with \(n\) characters, but the program works with \((n-1)!\) permutations of \(n\)-string.
The use of this technique is detailed in the section that describes the implementation of subtraction a single coset from a set of all permutations.

4.2 Main Functionalities

4.2.1 Calculating a Single Coset

To calculate a single coset, the two inputs are needed: a cycle and the stabilizer of $U_n$. As was described before, a single coset is the set of $m \times m^{-1}$, where $m$ is an element of the stabilizer, and $x$ is the input cycle. There are two different implementations of how to calculate $m \times m^{-1}$, which are described below. Recall, that if $x = (i_1, i_2, \ldots, i_k)$, then $m \times m^{-1} = (m(i_1), m(i_2), \ldots, m(i_k))$.

Approach 1. For each element $m$ of a stabilizer, process each symbol, character $c$, of the given cycle in the following way: find the character $c$ in $m$ by comparing $c$ to each character of $m$ starting from the left of the string representing $m$. After $c$ is found in $m$, read in the next character of $m$ that follows $c$, call it "next". "next" is $m(c)$.

Let $St$ be the stabilizer of $U_n$, and let the order of $St$ is equal to $d$. How many operations are needed to calculate $m \times m^{-1}$? For a single element $m \in St$ with length 1, there
are from 1 to 1 comparisons for each symbol of a cycle. The length of m varies from 25 to 32. Let $l_{\text{ave}}$ be an average length of m, then $l_{\text{ave}} = (25 + 32)/2 = 29$. And let $\text{comp}_{\text{ave}} = (29(29+1)/2)/29 = 15$ be the average number of comparisons of one character of the cycle to the characters of m. Then the total number of comparisons that is needed to be performed to calculate $m \times m^{-1}$ for all m in St is equal to $\text{comp}_{\text{ave}} * n * d = 15n$ where n is the number of characters in x, and d is the number of elements m in St. In the best case (number of comparisons is 1 for each m in St), the total number of comparisons is $nd$, and in the worst case (the number of comparisons is equal to the average length of m) the total number of comparisons is $29nd$.

**Approach 2.** Let $u = (i_1 \ i_2 \ i_3 \ ... \ i_n)$ be the first cycle in the set of all permutations of $U_n$. For this approach, $m \ u \ m^{-1}$ is calculated as was described in Approach 1. The resulting single coset $O_u = \{m \ u \ m^{-1} \mid m \text{ is in St}\}$ is saved, and is used to calculate the rest of the single cosets. All the subsequent single cosets are calculated as follows. For each symbol of a cycle x, find the position of this symbol in u, where u is the cycle described above. Keep the found positions in a vector "winner". For each element m in St,
build a new cycle of a single coset by the rule (m u m⁻¹
[winner[0]], m u m⁻¹[winner[1]], ..., m u m⁻¹[winner[n-1]]),
where winner[i] is the saved position of the iᵗʰ character
of x in u, and m u m⁻¹[winner[i]] is the (winner[i])ᵗʰ symbol
of m u m⁻¹. This is illustrated in the following example.

Example.
Let m = advfabpohbctmlcenwxegrskgijuqi, and
u = (4 8 14 17 23 24) = dhmqwx, then
m u m⁻¹ = (m(d), m(h), m(n), m(q), m(w), m(x)) = vbwixe.
Let x = (4 14 24 23 8 17) = dnxwhq.

For each symbol of a cycle x, find the position of
this symbol in u, and form the vector "winner":

<table>
<thead>
<tr>
<th>position of</th>
<th>position of</th>
</tr>
</thead>
<tbody>
<tr>
<td>winner, i</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>iᵗʰ symbol</td>
<td>0 2 5 4 1 3</td>
</tr>
<tr>
<td>of x in u</td>
<td>0 2 5 4 1 3</td>
</tr>
</tbody>
</table>

Note that the vector "winner" is calculated once for x,
and does not depend on the order of St.

m x m⁻¹ = (m u m⁻¹[0], m u m⁻¹[2], m u m⁻¹[5], m u m⁻¹[4], m u m⁻¹[1], m u m⁻¹[3]) = vwexbi.
Compare this result with \( m \times m^{-1} \) calculated as in the first approach:

\[ m \times m^{-1} = (m(d), m(n), m(x), m(w), m(h), m(q)) = \text{vwexbi}. \]

The main purpose of this procedure is to calculate the positions of the characters of \( x \) in \( u \) once, and then using this information read the characters of the corresponding positions of \( m \times u \times m^{-1} \) for all \( m \) in \( St \), so there are no comparison operations for each \( m \) in \( St \) that were needed for the first approach.

The length of the first permutation of \( U_n, u \), is known and equal to \( n \). The average number of comparisons needed to find a character of a cycle \( x \) in \( u \) is \( \frac{(13 \times (13+1)/2)}{13} = 7 \). The total number of comparisons to calculate \( m \times m^{-1} \) is \( 7 \times n \). (Compare with \( 15 \times n \times d \).) The number of operations of other kinds is approximately equal in the 1st and 2nd approaches.

4.2.2 Subtraction Scheme

To subtract a single coset from the set of all permutations of \( U_n \), the following scheme has been worked out. The main purpose of this scheme is to keep track of which single cosets are found and not to pick elements of the found single cosets to calculate a new single coset. There are two issues related to the subtraction function.
The first is that it is not possible to keep the whole set of permutations in RAM, even if many processes are used and the set is divided between them. So, the subtraction must be performed in parts: after a single coset is calculated and kept in memory, a part of the set of permutations is read into memory, the subtraction of elements of single cosets from this part is performed, and another part of the set of permutations is read, and so on until the whole set is processed. However, this procedure is very inefficient because to subtract a relatively small set of permutations from a set that is thousands or millions of times larger requires a lot of time. The remedy to this problem is the following: the subtraction of a single coset is performed only from one subset of the set of permutations, and a new single coset representative is chosen from this subset; the process continues until the subset is empty. While processing the first subset, the program keeps track of the size of the single cosets found. After the first subset is processed, the program checks whether it needs to take another subset: if there were found p single cosets of the size r, the total permutations that would have been subtracted are \( P \times r \), and if the size of the whole set of permutations is \( P \times r \), then there is no need to process other
subsets since all the single coset representatives are found in the first subset. The described procedure is demonstrated in the following example.

**Example.**

Recall the example that explains the problem's procedure for $U_5$. In this example, $U_5$ and its $St$ are used again. An additional control value $W$ is used that is incremented by the size of the calculated single coset.

The subset $A_5$ of $P_5$ is read into memory:

$$A_5 = \begin{cases} (1\ 2\ 3\ 4\ 5), \\
(1\ 2\ 3\ 5\ 4), \\
(1\ 2\ 4\ 3\ 5), \\
(1\ 2\ 4\ 5\ 3), \\
(1\ 2\ 5\ 3\ 4), \\
(1\ 2\ 5\ 4\ 3) \end{cases}$$

A single coset representative is chosen to be a cycle from $A_5$, $\alpha = (1\ 2\ 3\ 4\ 5)$. Then $O_\alpha = \{(1\ 2\ 3\ 4\ 5), (1\ 2\ 4\ 5\ 3), (1\ 3\ 4\ 2\ 5), (1\ 3\ 5\ 4\ 2), (1\ 4\ 3\ 2\ 5), (1\ 4\ 5\ 2\ 3), (1\ 5\ 2\ 3\ 4), (1\ 5\ 3\ 4\ 2)\}$ is calculated as was described earlier. $W$ is incremented by the size of $O_\alpha$, $W = 8$. Then each element of $O_\alpha$ is subtracted from $A_5$.

The subset $A_5$ is

$$\begin{cases} (1\ 2\ 3\ 5\ 4), \\
(1\ 2\ 4\ 3\ 5), \\
(1\ 2\ 5\ 3\ 4), \\
(1\ 2\ 5\ 4\ 3) \end{cases}$$

27
Pick another orbit representative from $A_5$, $\beta = (1\ 2\ 3\ 5\ 4)$. Calculate $O_\beta$; increment $W$, $W = 16$. After subtraction $A_5$ is:

\[
\begin{cases}
(1\ 2\ 4\ 3\ 5), \\
(1\ 2\ 5\ 3\ 4)
\end{cases}
\]

Pick another orbit representative from $A_5$, say $\gamma = (1\ 2\ 4\ 3\ 5)$. Calculate $O_\gamma$; increment $W$, $W = 24$. After subtraction $O_\gamma$, $A_5$ is empty. Check whether $W$ is equal to the size of the original set of all permutations, $P_5$. The size of $P_5$ is 24, and $W = 24$. So, all the single coset representatives are found, and there is no need to process other subsets of $P_5$.

The second issue of subtraction function is how big should a subset of permutations be to fit into memory. The bigger the subset, the more chances of not repeating the calculations for a new subset. But there is a technical restriction to the size of the subset: the size of memory. In order to fit a larger subset into memory, the data structure array of bits is used. Each bit of the array represents one permutation. If a bit is set to 1, the corresponding permutation has not been subtracted, and if a bit is set to 0, then the corresponding permutation has been subtracted. A new orbit representative is chosen among...
bits that are set to 1. The position of a bit in the array represents a position of the corresponding permutation in the set of permutations.

The following procedure provides the steps of the substraction scheme.

**Step 1.** Find a bit that is set to 1 in the array of bits, b.

**Step 2.** Convert a position of the bit b to the position in the set of all permutations, i.

**Step 3.** Calculate a permutation s from the position i.

**Step 4.** Calculate a single coset 0s as described in the Approach 2 of Section 4.2.1.

**Step 5.** Convert the elements of 0s to the corresponding positions in the set of all permutations, call the set of positions D.

**Step 6.** Find positions in the bit set that correspond to the positions of the set D and set the corresponding bits to 0.

The advantage of this scheme is that it solves the space problem. There is no need to keep huge file of the set of all permutations of Un in the second storage. Also, this scheme permits a larger subset of permutations to be
kept in memory, since only one bit is needed to represent one permutation.

This scheme has some drawbacks. The calculations of permutations from the positions and vice versa are used which is time consuming. To decrease the number of such calculations, at Step 4 described above, the program checks whether the $m \times m^{-1}$ is in the range of the processing bitset: if it is, then it is converted to the position to be subtracted from the bitset, otherwise, it is ignored. Another disadvantage is that if not all single coset representatives are found in one subset, then for the next subset the program first calculates single cosets for the identified single coset representatives, and subtracts them from the new subset, and after that it chooses new single coset representatives from those elements of the new subset that have not been subtracted. However, the process of subtracting one single coset from the whole set of all permutations takes more time than performing the same calculations a few times. Despite these disadvantages, this scheme allows for the solving of the space problem and, to some extent, the time problem.
CHAPTER FIVE
PARALLEL ALGORITHM

N processes participate in this parallel algorithm. Each process has its own bitset (an array of bits) that represents a subset from the set of all permutations of $U_n$. N processes cover N equal different subsets. A particular $U_n$ and the file that holds its stabilizer are the input to the program. Since the larger part of the work is in calculating a single coset, the stabilizer is divided by N parts, and each process reads in the corresponding part of the stabilizer. Data partitioning and the data structures used by each process are shown in Figure 1.
The processes choose a new single coset representative, $x$, from their bitset in turn. After being selected, $x$ is sent via MPI to all processes and to the file "orbits.txt".

Next, each process calculates its part of a single coset, after which a process converts the resulting permutations to the corresponding positions of the strings.
in the set of all permutations, and exchanges the results between other processes. Then each process subtracts its own results from the bitset, waits for the rest of processes to finish with their calculations (MPI_Barrier is used), and subtracts the results of the others from the bitset. Now the processes are ready to choose a new orbit representative from among those permutations that are left after subtraction in their bitsets. The calculations continue in a loop until the bitsets of all the processes are empty (when all bits in a bitset are set to 0). This process is considered to be one round. The processes have certain control values that tell the processes whether new subsets from the set of permutations are needed to be chosen or the calculations have been completed in one round. The message passing scheme is presented in Figure 2.
Calculate and send a new single coset representative

Conjugate the new representative by stabilizer in parallel

Exchange calculations

Subtract permutations from set_bit in parallel

Exchange control values

Figure 2. Message Passing
The pseudocode for the Start and Work parts of the proposed parallel algorithm is presented in Figures 3 and 4. The pseudocode is given for a single process that participates in the parallel algorithm with $N_p$ processes. The message exchange is highlighted.
Each process has its identification number, my_ID, that varies from 0 to N_p-1. Read in the input k-cycle and encipher it into the string S. Read in the corresponding to this process part of the input stabilizer, encipher it, and conjugate S by the enciphered stabilizer, and put the result into the vector stabilizer. Initialize necessary control values:

- array chunks has (k-1) entries, and each entry keeps track how many cycles are found in the corresponding subset of the size (k-2)!, Chunk, of the set of all permutations of the input cycle, Permut;
- array subset has (k-2) entries, and each entry keeps track how many cycles are found in the subset of the size (k-3)!, Sub_chunk, of the currently processing Chunk;
- array set_bit is of the size (k-3)! if this value is less than maximum allowable threshold value (that may vary dependent on the size of RAM), or divide (k-3)! by N_p until the resulting value is less than the threshold value;
- set the control value set_empty to the size of set_bit; set_empty keeps track how many there are bits set to 1 in the set_bit; when set_empty is 0, then all permutations have been found and subtracted from set_bit;

Figure 3. Pseudocode for the Start Part
Figure 4. Pseudocode for the Work Part

While the sum of the entries of chunks is not equal to the size of the set of all permutations of the input cycle, (k-1)!
do
    Set entries of chunks to 0.
    Set entries of subset to 0.
    Calculate the start position of set_bit as:
    \[ \text{pos}\_\text{start} = (\text{my}\_\text{ID} \times \text{size}\_\text{of}\_\text{set}\_\text{bit}) \]
    \[ + (\text{current}\_\text{round} \times N_p \times \text{size}\_\text{of}\_\text{set}\_\text{bit}). \]
    Calculate the end position of set_bit as:
    \[ \text{pos}\_\text{end} = \text{pos}\_\text{start} + \text{size}\_\text{of}\_\text{set}\_\text{bit} - 1. \]
    Set entries of set_bit to 1.
    While all "orbits.txt" files are not processed
        do
            Open current file with single coset representatives.
            While the end of the current file is not encountered
                do
                    Read in a current single coset representatives, cos_repr.
                    Conjugate cos_repr by stabilizer;
                    increment the corresponding entries of chunks and subset.
                    Put conjugates that belong to set_bit*N_p into vector coset.
                    Calculate positions of conjugates of coset in Permut, put them into vector positions.
                    Send the vector positions to each of N_p processes.
                    Receive a vector positions from each of Np processes.
                    For each received positions, set entries of my set_bit corresponding to positions to 0.
                    Decrease set_empty by the number of bits that set to 0.
                end do
            Close file,
        end do
    end do
While set_empty is not 0
do
    Find the first bit that is set to 1 in my set_bit.
    Calculate the position of this bit in Permut.
    Calculate permutation that corresponds to this position;
    if it is my turn, send this permutation to each of Np
    processes. This permutation is the new representative.
    Conjugate this new representative by stabilizer;
    increment the corresponding entries of chunks and subset.
    Put conjugates that belong to set_bit into vector coset.
    Calculate positions of conjugates of coset in Permut,
    put them into vector positions.
    Send the vector positions to each of Np processes.
    Receive a vector positions from each of Np processes.
    For each received positions, set entries of my set_bit
    corresponding to positions to 0.
    Decrease set_empty by the number of bits that set to 0.
end do
Use the Reduce and Broadcast functions of MPI to add the
values of entries of chunks of all processes.
Check whether the sum of entries of chunks of all processes
is equal to (k-l)! 
Find the first not full Chunk; it is current now.
Use the Reduce and Broadcast functions of MPI to add the
values of entries of subset of all processes.
Find not full Sub_chunk in subset; it is current set_bit.
end do

Figure 4. Pseudocode for the Work Part (Continued)
CHAPTER SIX
RESULTS AND DISCUSSION

To analyze the proposed algorithm, sequential and parallel programs are developed in this thesis. The performance of the sequential program is compared with the performance of Magma’s program (Appendix B), and the performance of the parallel program is evaluated. The results of experiments depend on many variables such as length of a cycle x used to calculate M x M, size of the set of all permutations of x, order of the stabilizer of x in M_{24} and the number of single cosets in the double coset M x M.

In Section 6.1, the measures that are used in these experiments are introduced. In Section 6.2, an overview of the experiments is provided. In Section 6.3, results are presented and evaluated. Finally, in Section 6.4, the estimation about time execution and resources needed to solve the problem for a larger order of cycles is given.

6.1 Measurements

To obtain an execution profile of both parallel and sequential codes, time spent in initialization, time spent
in different stages of calculation, and time spent in message passing are measured explicitly in CPU seconds. Another measurement that is used to evaluate the time complexity of the algorithm is the number of arithmetic operations performed by the program; this measurement is obtained by implementing a counter in the code. Arithmetic operations are used as criteria of measurement of amount of work rather than float point operations since the program does not use the latter operations. The number of messages is counted for the different number of processes and different k-cycles.

Other measures are calculated indirectly. Knowing a k-cycle’s length and the order of its stabilizer, $St$, one can calculate the size of the set of all permutations, $P$, of this cycle by the formula:

$$|P| = (k-1)!$$  \hspace{1cm} (1)

The approximate number of single cosets, $Q$, is calculated as follows:

$$|Q| = |P|/|St|$$  \hspace{1cm} (2)

The speed of calculating single cosets per second is found by the formula:

$$\text{Speed}_{\text{orb}} = |Q| / T_{\text{exc}},$$  \hspace{1cm} (3)
where $T_{exc}$ is the execution time required to find $|Q|$ single cosets.

Speedup is calculated by the formula:

$$\text{Speedup} = \frac{T_{seq}}{T_{exc}},$$

(4)

where $T_{seq}$ is the execution time on one process, and $T_{exc}$ is the execution time on $N_p$ processes.

The efficiency of the parallel program is calculated as:

$$\varepsilon = \frac{\text{Speedup}}{N_p}$$

(5)

6.2 Overview Of Experiments

The set of experiments is carried out to evaluate both sequential and parallel implementations of the proposed algorithm. All the experiments described in this thesis are done on the Raven Cluster if otherwise is not stated.

The Raven Cluster is a cluster of machines connected via switched Gigabit network. Each machine in the cluster has two 1.4 MHz Pentium® III processors and 256 MB of RAM.

Both sequential and parallel programs use two input files: the first one, "input_cycle.txt", contains a $k$-cycle, $(1, 2, 3, ..., k)$, and the second file, "stabilizer.txt", contains the stabilizer of this cycle in $M_{24}$ calculated by Magma. It is important to put the elements of the cycle in the ascending order. The output, single
coset representatives, is in the output files "orbitsX.txt", where X varies since the maximum number of single coset representatives allowed to be put in a single file is fixed. After one output file is full, another file is opened for the output. Before testing the program with a different k-cycle, the output files "orbitsX.txt" calculated previously must be deleted or renamed.

6.2.1 Sequential Program

The purpose of experiments is to evaluate the time complexity of the sequential program, and to compare the performance of this program with the functioning of Magma's program.

Logically, the proposed program is subdivided into two main parts: the Start part and the Work part. In the Start part, the input cycle and its stabilizer are read into memory and enciphered. Then the stabilizer is transformed as is described in Subsection 4.2.2, i.e., the input k-cycle is conjugated by the enciphered stabilizer producing as a result the "transformed stabilizer". Finally, initialization of the necessary variables and data structures takes place. In the Work part, the main functionalities described in Section 4.2 are performed in a loop.
The purpose of the first test is to calculate the arithmetic operations in the Start and Work parts as well as to explicitly measure the time of Start, \(T_{\text{start}}\), the time of Work, \(T_{\text{work}}\), and the time of execution, \(T_{\text{seq}}\). The results for different cycles are presented in the next section.

The next experiment is to compare the two algorithms: the straightforward one that uses Magma's function calls, and the proposed one. This test is performed for different cycles, using an Intel Celeron® 586 processor running Windows 98 SE, and the results are provided in the next section.

6.2.2 Parallel Program

To analyze the parallel program in terms of efficiency and scalability, a number of experiments is performed.

The time of execution, \(T_{\text{exec}}\), the time of the sequential part of the program, \(T_{\text{start}}\), the time of calculation, \(T_{\text{calc}}\), and the time of communication, \(T_{\text{comm}}\) are measured explicitly for the cycles of different lengths and different number of processes. These values are used to calculate speedup and efficiency, and analyze the results.

To study how communication scales, the test is done where the number of processes is fixed and the number of messages is measured as a function of problem size. This
test shows whether the number of messages increases faster or slower than the amount of work.

6.3 Results

6.3.1 Sequential Program

In the following table the results of counting arithmetic operations for both the Start and Work parts of the sequential program are given for different inputs, \( U_n \).

Table 1. Amount of Work for Sequential Program

| n  | |St| | # of orbits | # of arithmetic operations | # of arithmetic operations |
|----|---|---|-------------|--------------------------|--------------------------|
|    |   |   |             | Start                  | Work                  | Total                    |
| 7  | 720| 1 | 144,052     | 2,592                  | 146,644                |
| 8  | 384| 18| 69,451      | 65,950                 | 135,401                |
| 9  | 432| 98| 91,845      | 159,652                | 251,497                |
| 10 | 1,440| 264| 259,346    | 1,318,308              | 1,577,654              |
| 11 | 7,920| 460| 1,451,242  | 12,053,591             | 13,504,833            |
| 12 | 95,04| 446| 19,941,561 | 137,716,357            | 157,657,918           |
| 13 | 7,920| 60,480| 1,715,104  | 1,572,008,685          | 1,573,723,78          |

The results of Table 1 imply that the amount of work spent in the Start part of the sequential program depends on the order of the input stabilizer. To verify this conjecture, the next experiment is done for the cycles of
larger length. The number of arithmetic operations in the Start part is counted, and then the program exits the execution. The results are given in Table 2.

Table 2. Amount of Work in the Start Part

<table>
<thead>
<tr>
<th>Length of a cycle</th>
<th>Stabilizer’s order</th>
<th># arith. operat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1440</td>
<td>470,256</td>
</tr>
<tr>
<td>15</td>
<td>432</td>
<td>132,960</td>
</tr>
<tr>
<td>16</td>
<td>384</td>
<td>122,094</td>
</tr>
<tr>
<td>17</td>
<td>720</td>
<td>254,228</td>
</tr>
<tr>
<td>18</td>
<td>2160</td>
<td>694,585</td>
</tr>
<tr>
<td>19</td>
<td>5760</td>
<td>1,950,080</td>
</tr>
</tbody>
</table>

Comparing the results of Tables 1 and 2, it is observed that the amount of work spent in the Start part is the function of the two variables: the input stabilizer’s order and the length of the input cycle. Graph 1 shows the amount of the Start work as the function of the order of a stabilizer. The lower curve is drawn for $U_8$, $U_9$, $U_7$, $U_{10}$, $U_{11}$, and the upper curve for $U_{16}$, $U_{15}$, $U_{17}$, $U_{14}$, and $U_{13}$ (the cycles are taken in the ascending order of the corresponding stabilizers). The results of Tables 1 and 2 are used for Graph 1.
Graph 1. Start Work as Function of Stabilizer’s Order

The major contribution to the Start amount of work is due to enciphering and transforming the input stabilizer. To encipher a stabilizer, it is necessary to read every character of the input file. The approximate estimation of the amount of work required to encipher a stabilizer of order |St| is calculated as the product of |St| and the number of characters in one line of the input file. The line is represented as a product of disjoint cycles on 24 numbers separated by comma and space. So, the total number of characters in one such line is 24*3. This approximation results in an overestimation since many lines have less than 24 numbers. To transform the enciphered stabilizer,
(24/2)*n arithmetic operations are needed for one line of the stabilizer since it is necessary to find n characters in a line of the length 24. So, the total number of the operations to transform the stabilizer is calculated as |St|*(24/2)*n. However, experiments show that this approximation also gives an overestimation. The contribution of the initialization is negligible and is ignored. So, theoretically, the amount of Start work is represented as the function of the two variables, the stabilizer’s order (|St|) and the input cycle’s length (n), in the following formula:

\[ W_{\text{start}} = |St|*24*3 + |St|*(24/2)*n \]

or

\[ W_{\text{start}} = |St|*[72 + 12*n] \]  \hspace{1cm} (6)

For the cycles of length n equal to 6, 7, 9, 10, 11, 12 or 13, it is sufficient to use only one subset (of order (n-3)!) of the set of all permutations to find all single coset representatives in the double coset \( M_{24} U_n M_{24} \). If only one subset is required then we say only one round of calculations is required. If only one round is required to calculate all single coset representatives, then the next approximation can be used to compute the amount of work spent in the Work part. The amount of work needed to
conjugate a cycle by the transformed stabilizer is the function of the two variables: |St| and n defined above.

$$W_{orb} = \left(\frac{(n-1)!}{|St|}\right) \times (|St| + \frac{n^2}{2} + 2n)$$

$$+ n^*(n-2)! + (n-3)!$$

(7)

$W_{orb}$ is the amount of work required to conjugate $[(n-1)!/|St|]$ single coset representatives by the transformed stabilizer. The number of arithmetic operations evaluated by $(|St| + \frac{n^2}{2} + 2n)$ is required for each single coset representative. And the rest of the operations, $(n^*(n-2)! + (n-3)!)$, is required for the calculation of all single coset representatives. The value $(n-3)!$ is the size of the subset of the set of all permutations used in the calculations. The function that conjugates a single coset representative with the transformed stabilizer has an addition comparison operation that allows sorting out only those conjugates that belong to the currently processing subset of the set of all permutations. Consequently, this allows the saving of work and time on the calculation of the positions in the imaginary file of all permutations. The amount of work needed to calculate the positions from the strings is estimated by the formula:

$$W_{calc} = (n^2 - 1.5n)*(n-3)!$$

(8)
And the last contribution to the amount of work in the Work part is the number of comparisons needed to perform the subtraction of conjugates from the subset of the set of all permutations:

\[ W_{\text{subtr}} = (n-3)! \]  \hspace{1cm} (9)

So, the total amount of work of the Work part is

\[
W_{\text{work}} = W_{\text{orb}} + W_{\text{calc}} + W_{\text{subtr}}, \text{ or using (7), (8), and (9)}:
\]

\[
W_{\text{work}} = \left[(n-1)! / \mid St\mid \right] \times (\mid St\mid + n^2/2 + 2*n) + n*(n-2)!
\]
\[
+ (n-3)! + (n^2 - 1.5*n)(n-3)! + (n-3)!, \text{ or}
\]

\[
W_{\text{work}} = \left[(n-1)! / \mid St\mid \right] \times (\mid St\mid + n^2/2 + 2*n)
\]
\[
+ (n-3)! \times (2*n^2 - 3.5*n + 2) \hspace{1cm} (10)
\]

The results of comparing the two values of \( W_{\text{work}} \) are presented in Table 3: the first value is obtained using formula (10) and the second value is taken from Table 1.
Table 3. Comparison of Theoretical and Experimental Values of $W_{\text{work}}$

<table>
<thead>
<tr>
<th>n</th>
<th># of arithmetic operations in Work part</th>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2,571</td>
<td>2,592</td>
</tr>
<tr>
<td>8</td>
<td>18,288</td>
<td>65,950</td>
</tr>
<tr>
<td>9</td>
<td>141,507</td>
<td>159,652</td>
</tr>
<tr>
<td>10</td>
<td>1,222,200</td>
<td>1,318,308</td>
</tr>
<tr>
<td>11</td>
<td>11,958,908</td>
<td>12,053,591</td>
</tr>
<tr>
<td>12</td>
<td>129,951,360</td>
<td>137,716,357</td>
</tr>
<tr>
<td>13</td>
<td>1,554,366,240</td>
<td>1,572,008,685</td>
</tr>
</tbody>
</table>

As it can be observed from Table 3, the values of $W_{\text{work}}$ obtained theoretically and experimentally are close. The exception is the values for $U_8$, which occurs because the single coset representatives for $U_8$ are not calculated in one round; three subsets of the size $(n-3)!$ (in this case, $(8-3)!$) of the set of all permutations are needed to calculate all single coset representatives. In other words, there are repetitions of the same calculations.

The calculation of single coset representatives for $U_8$ is a good example of what happens if the size of the subset is not sufficient to find all the single coset representatives. For $U_8$, 10 single coset representatives are
found in the first round, then the second round is started
by taking the next subset and, having 10 identified single
coset representatives, by recalculating single cosets for
them and subtracting the latter from the new subset. Seven
new single coset representatives are found in the second
round. In the third round, recalculations are performed for
17 single coset representatives and 1 new single coset is
found. So, to calculate the amount of work in the Work part
using formula (10), one needs to adjust the number of
single coset representatives for which the work is done.
Instead of the number of single coset representatives
calculated by the formula
\[
(n-1)! / |St| = (8-1)!/384 = 14,
\]
there are \((10) + (10 + 7) + (17 + 1) = 45\) single coset
representatives, for which work is performed. Moreover, in
formula 10, the number of arithmetic operations calculated
as \((n-3)! \ast (2n^2 - 3.5n + 2)\) should be multiplied by the
number of rounds, three, in this case. Plugging these
values in the formula 10, we obtain
\[
W_{work} = [(n-1)! / |St|] \ast (|St| + n^2/2 + 2*n)
+ 3 \ast (n-3)! \ast (2n^2 - 3.5n + 2)
\]
\[
W_{work} = 45 \ast (384 + 64/2 + 16) + (8-3)! \ast (128 - 28 + 2)
\]
\[
W_{work} = 56,160
\]
This value approximates the value from Table 1, 65,950.

The total amount of work required to calculate single coset representatives is given by the formula

\[
W_{\text{total}} = W_{\text{start}} + W_{\text{work}}, \text{ or}
\]

\[
W_{\text{total}} = |St|*[72 + 12*n] + [(n-1)! / |St|]*(|St| + n^2/2 + 2*n) + (n-3)! * (2*n^2 - 3.5*n + 2) \quad (11)
\]

It follows from (11) that \(W_{\text{total}}\) is a function of order at least \((n-1)!\), i.e.,

\[
W_{\text{total}}(n) = \Omega((n-1)!)\]

For smaller cycles of length from 6 to 13, the total amount of work is the function of order at most \(n!\):

\[
W_{\text{total}}(n) = O(n!)
\]

Graph 2 shows the \(\ln\) of the three functions: \(W_{\text{total}}(n)\), \((n-1)!\) and \(n!\) for the values of \(n\) equal 7, 8, 9, 10, 11, 12, and 13.
In $W_{\text{total}}(n)$

$$f(n) = \ln(n!)$$

$$g(n) = \ln((n-1)!)$$

Graph 2. Order Comparison: $n!$, $(n-1)!$, and $W_{\text{total}}(n)$

However, it is impossible theoretically to determine the value for the order at most for cycles with length greater than 13. Problems of larger size inevitably involve repetitive calculations. Since single coset representatives are distributed irregularly in the set of all permutations of $U_n$, and the program processes this set part by part, one cannot know in advance how many rounds of recalculations are required. As the result of such irregularity, the amount of work as well as execution time can be estimated only roughly in terms of possible minimum.

To compare the two algorithms (one that uses Magma’s function calls and the one proposed in this thesis), first
we discuss the program that uses Magma’s function calls, and then compare the results obtained using both programs.

Magma’s program is a sequential program. In the Start part of Magma’s program, the set of all permutations and the stabilizer of the input cycle are calculated; in the Work part of the Magma’s program, the following routing is performed in a loop until the set of all permutations is empty:

- Conjugate the first cycle of the set by the stabilizer producing the resulting subset Q;
- Subtract the subset Q from the set of all permutations;

Information about the implementation of Magma’s function calls is not available, so to estimate the order of the Magma’s program, we use the least possible amount of arithmetic operations needed to execute this program. In the Start part, to generate the set of all permutations for a given cycle of length n, it is assumed that at least one arithmetic operation is needed per set element. The calculation of the stabilizer for a given cycle is ignored. Hence, the amount of work required for the Start part of
the Magma’s program is the function of the order of the set of all permutations:

\[ W_{\text{start Magma}} = (n - 1)! \]  \tag{12}

In the Work part of Magma’s program, to conjugate a cycle of length \( n \) by a stabilizer of order \(|St|\), at least \(|St|*n*(24 / 3) \) arithmetic operations are required. To subtract a subset of order \(|St|\) from the set of all permutations of order \((n-1)!\), at least \(|St|*\log_2((n-1)!)\) arithmetic operations are needed. The arithmetic operations mentioned above for both the function calls in the Work part must be performed at least \((n-1)!/|St|\) times. This estimation results in the following formula for the amount of work in the Work part of the Magma’s program:

\[ W_{\text{work Magma}} = \left\{ (n-1)! / |St| \right\} \times \{ |St|*n*(24 / 3) + |St|*\log_2((n-1)!) \}, \text{ or} \]

\[ W_{\text{work Magma}} = (n-1)! * \{8*n + \log_2((n-1)!)) \} \]  \tag{13}

It is calculated that for \( n \) equal or greater than five, \( \log_2((n-1)!) \) is equal to or greater than \( n \). Then it follows that for \( n \) equal to or greater than five, the number of arithmetic operations for \( W_{\text{work Magma}} \) is greater than \((n!)\). The total amount of work needed for Magma’s program is calculated using formulas 12 and 13 as:

\[ W_{\text{total Magma}} = W_{\text{start Magma}} + W_{\text{work Magma}}, \text{ or} \]
\[ W_{\text{total}_\text{Magma}} = (n - 1)! + (n-1)! \times \{8n + \log_2[(n-1)!]\} \quad (14) \]

Thus, it follows that \( W_{\text{total}_\text{Magma}} \) is the function of order at least \( n! \):

\[ W_{\text{total}_\text{Magma}}(n) = \Omega(n!), \]

and of order at most \( 12\times(n!) \) since \( n \) is not larger than 24.

Tables 4, 5, and 6 compare the values of \( W_{\text{start}} \) and \( W_{\text{start}_\text{Magma}}, W_{\text{work}} \) and \( W_{\text{work}_\text{Magma}}, \) and \( W_{\text{total}} \) and \( W_{\text{total}_\text{Magma}} \) obtained using the formulas 10, 11, 13, and 14. \( W_{\text{work}} \) for \( U_8 \) is calculated taking into account recalculations.

**Table 4. \( W_{\text{start}} \) and \( W_{\text{start}_\text{Magma}} \)**

<table>
<thead>
<tr>
<th>( U_n )</th>
<th>( U_6 )</th>
<th>( U_7 )</th>
<th>( U_8 )</th>
<th>( U_9 )</th>
<th>( U_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_{\text{start}} )</td>
<td>1,658,880</td>
<td>112,320</td>
<td>64,512</td>
<td>77,760</td>
<td>276,480</td>
</tr>
<tr>
<td>( W_{\text{start}_\text{Magma}} )</td>
<td>120</td>
<td>720</td>
<td>5,040</td>
<td>40,320</td>
<td>362,880</td>
</tr>
</tbody>
</table>

**Table 5. \( W_{\text{work}} \) and \( W_{\text{work}_\text{Magma}} \)**

<table>
<thead>
<tr>
<th>( U_n )</th>
<th>( U_6 )</th>
<th>( U_7 )</th>
<th>( U_8 )</th>
<th>( U_9 )</th>
<th>( U_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_{\text{work}} )</td>
<td>20,852</td>
<td>2,571</td>
<td>56,160</td>
<td>141,507</td>
<td>1,222,200</td>
</tr>
<tr>
<td>( W_{\text{work}_\text{Magma}} )</td>
<td>6,709</td>
<td>47,874</td>
<td>389,588</td>
<td>3,560,224</td>
<td>36,095,359</td>
</tr>
</tbody>
</table>
Table 6. \( W_{\text{total}} \) and \( W_{\text{total, Magma}} \)

<table>
<thead>
<tr>
<th>( U_n )</th>
<th>( U_6 )</th>
<th>( U_7 )</th>
<th>( U_8 )</th>
<th>( U_9 )</th>
<th>( U_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_{\text{total}} )</td>
<td>1,679,732</td>
<td>114,891</td>
<td>120,692</td>
<td>219,267</td>
<td>1,498,680</td>
</tr>
<tr>
<td>( W_{\text{total, Magma}} )</td>
<td>6,829</td>
<td>48,594</td>
<td>394,628</td>
<td>3,600,544</td>
<td>36,458,239</td>
</tr>
</tbody>
</table>

Graphs 3, 4, and 5 shows ln of the values of Tables 4, 5, and 6 respectively.

Graph 3. \( W_{\text{start}} \) and \( W_{\text{start, Magma}} \)
Graph 4. $W_{\text{work}}$ and $W_{\text{work_Magma}}$

Graph 5. $W_{\text{total}}$ and $W_{\text{total_Magma}}$
Table 7. shows the results of the explicitly measured running time of the proposed sequential and Magma programs.

Table 7. Comparison of Running Time $T_{seq}$ and $T_{Magma}$

<table>
<thead>
<tr>
<th>$U_n$</th>
<th>$U_6$</th>
<th>$U_7$</th>
<th>$U_8$</th>
<th>$U_9$</th>
<th>$U_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{seq}$, CPU sec</td>
<td>34.43</td>
<td>2.14</td>
<td>1.75</td>
<td>3.34</td>
<td>19.39</td>
</tr>
<tr>
<td>$T_{Magma}$, CPU sec</td>
<td>0.393</td>
<td>0.613</td>
<td>0.978</td>
<td>14.513</td>
<td>335.099</td>
</tr>
</tbody>
</table>

Graph 6 shows a comparison of the results of Table 7.

Graph 6. Comparison of Running Time $T_{seq}$ and $T_{Magma}$

Tables 8 and 9 show the distribution of execution time between the Start part and Work part for the two programs.
Table 8. Comparison of \( T_{\text{start}} \) and \( T_{\text{start,Magma}} \)

<table>
<thead>
<tr>
<th>( U_n )</th>
<th>( U_6 )</th>
<th>( U_7 )</th>
<th>( U_8 )</th>
<th>( U_9 )</th>
<th>( U_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{\text{start}} ), CPU sec</td>
<td>34.16</td>
<td>2.09</td>
<td>1.21</td>
<td>1.36</td>
<td>4.18</td>
</tr>
<tr>
<td>( T_{\text{start,Magma}} ), CPU sec</td>
<td>0.149</td>
<td>0.205</td>
<td>0.319</td>
<td>0.556</td>
<td>4.599</td>
</tr>
</tbody>
</table>

Table 9. Comparison of \( T_{\text{work}} \) and \( T_{\text{work,Magma}} \)

<table>
<thead>
<tr>
<th>( U_n )</th>
<th>( U_6 )</th>
<th>( U_7 )</th>
<th>( U_8 )</th>
<th>( U_9 )</th>
<th>( U_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{\text{work}} ), CPU sec</td>
<td>0.27</td>
<td>0.05</td>
<td>0.54</td>
<td>1.98</td>
<td>15.21</td>
</tr>
<tr>
<td>( T_{\text{work,Magma}} ), CPU sec</td>
<td>0.244</td>
<td>0.408</td>
<td>0.659</td>
<td>13.957</td>
<td>330.5</td>
</tr>
</tbody>
</table>

Graph 7 demonstrates the results of Table 8, and Graphs 8 and 9 the results of Table 9.
Comparing Graphs 3 and 7, it is observed that both the amount of work in the Start part and the start time of the both programs behave in the same way. $W_{\text{start}}$ and $T_{\text{start}}$ of the proposed program is greater than $W_{\text{start\_Magma}}$ and $T_{\text{start\_Magma}}$ of Magma’s program for $U_6, \ldots, U_9$, and smaller for $U_{10}$. It happens because $W_{\text{start\_Magma}}$ is of order at least $(n-1)!$ and $W_{\text{start}}$ depends on a stabilizer's order. Thus, $W_{\text{start\_Magma}}$ grows faster for a larger $n$, and comparatively less for a smaller $n$. The proposed program uses techniques that require more work in the beginning but help to save work and running time for larger problems.
Graph 8. Comparison of $T_{\text{work}}$ and $T_{\text{work-Magma}}$ for $U_6, U_7, \text{and } U_8$

Graph 9. Comparison of $T_{\text{work}}$ and $T_{\text{work-Magma}}$ for $U_8, U_9, \text{and } U_{10}$
Comparing Graphs 4 and 8, and 9, it is observed that both the amount of work and the time in the Work part of the both programs behave in the same way. Time spent in the Work part of the proposed program is slower than of the Magma’s program for $U_6$, and faster for $U_7$, $U_8$, $U_9$, and $U_{10}$.

It is possible to calculate the single coset representatives for $U_{11}$ using Magma’s program, but for $U_{12}$, there is not enough memory to keep the set of all permutations.

Using the proposed program, the single coset representatives are calculated for $U_{12}$, $U_{13}$, and $U_{14}$. The calculations are performed on the Raven Cluster, and the results are presented in Table 10.

Table 10. Single Cosets for $U_{12}$, $U_{13}$, and $U_{14}$

<table>
<thead>
<tr>
<th></th>
<th>$U_{12}$</th>
<th>$U_{13}$</th>
<th>$U_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># single cosets</td>
<td>single cosets' order</td>
<td># single cosets</td>
</tr>
<tr>
<td>400</td>
<td>95,040</td>
<td>60,480</td>
<td>7,920</td>
</tr>
<tr>
<td>12</td>
<td>31,680</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>47,520</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15,840</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.3.2 Parallel Program

Table 11 contains the results of the running time of the parallel program, $T_{exc}$. The measurements are taken for $Un$, where $n$ varies from 7 to 13, and for the different number of processes.
Table 11. $T_{\text{exc}}$ in Seconds, Parallel Program

<table>
<thead>
<tr>
<th># proc $N_p$</th>
<th>$U_7$</th>
<th>$U_8$</th>
<th>$U_9$</th>
<th>$U_{10}$</th>
<th>$U_{11}$</th>
<th>$U_{12}$</th>
<th>$U_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.367</td>
<td>0.542</td>
<td>0.438</td>
<td>1.154</td>
<td>7.17</td>
<td>82.29</td>
<td>882.2</td>
</tr>
<tr>
<td>2</td>
<td>0.658</td>
<td>0.812</td>
<td>0.618</td>
<td>1.355</td>
<td>4.783</td>
<td>43.71</td>
<td>522.8</td>
</tr>
<tr>
<td>3</td>
<td>0.702</td>
<td>0.714</td>
<td>0.893</td>
<td>1.344</td>
<td>4.241</td>
<td>31.42</td>
<td>410.9</td>
</tr>
<tr>
<td>4</td>
<td>0.843</td>
<td>1.036</td>
<td>1.088</td>
<td>1.972</td>
<td>5.344</td>
<td>26.67</td>
<td>457</td>
</tr>
<tr>
<td>5</td>
<td>0.898</td>
<td>1.407</td>
<td>1.568</td>
<td>2.268</td>
<td>5.192</td>
<td>23.48</td>
<td>409.6</td>
</tr>
<tr>
<td>6</td>
<td>0.916</td>
<td>1.05</td>
<td>1.923</td>
<td>2.847</td>
<td>5.369</td>
<td>22.16</td>
<td>448.6</td>
</tr>
<tr>
<td>7</td>
<td>1.327</td>
<td>1.501</td>
<td>2.182</td>
<td>3.019</td>
<td>5.733</td>
<td>23.24</td>
<td>636.1</td>
</tr>
</tbody>
</table>

Observation of the results in Table 11 suggests that running in parallel does not improve the execution time for the sets of small size, for $U_7, \ldots, U_{10}$. The sequential part of the parallel program corresponds to the Start part of the sequential program. With a small problem size, the sequential part of the parallel program contributes the most to the total amount of work, making parallelism inefficient.

The purpose of the parallel program is to speed up the running time of larger sets. To analyze the results given in Table 11, the speedup and efficiency are calculated as
is described in Section 6.1 for $U_{11}$, $U_{12}$, and $U_{13}$, and presented in Tables 12 and 13.

Table 12. Speedup($n$, $N_p$)

| $U_n$ | $|St|$ | Speedup($n$, $N_p$) |
|-------|-------|---------------------|
|       |       | Number of processes, $N_p$ |
|       |       | 2  | 3  | 4  | 5  | 6  | 7  |
| $U_{11}$ | 7,920 | 1.50 | 1.69 | 1.34 | 1.38 | 1.34 | 1.25 |
| $U_{12}$ | 95,040 | 1.88 | 2.62 | 3.08 | 3.50 | 3.71 | 3.54 |
| $U_{13}$ | 7,920 | 1.69 | 2.15 | 1.93 | 2.15 | 1.97 | 1.39 |

Table 13. Efficiency, $\varepsilon(n, N_p)$

| $U_n$ | $|St|$ | Efficiency, $\varepsilon(n, N_p)$ |
|-------|-------|----------------------------------|
|       |       | Number of processes, $N_p$ |
|       |       | 2  | 3  | 4  | 5  | 6  | 7  |
| $U_{11}$ | 7,920 | 0.75 | 0.56 | 0.34 | 0.28 | 0.22 | 0.18 |
| $U_{12}$ | 95,04 | 0.94 | 0.87 | 0.77 | 0.70 | 0.62 | 0.51 |
| $U_{13}$ | 7,920 | 0.84 | 0.72 | 0.48 | 0.43 | 0.33 | 0.20 |
To understand the experimental results of the parallel program, one needs to examine how the amount of work is divided between the processes. The amount of work needed for the sequential part of the parallel program is calculated as the amount of work for the Start part of the sequential program:

\[ W_{\text{start}} = |St|*[72 + 12*n] \]

Recall that the amount of work required for the Work part of the sequential program is calculated by the formula

\[
W_{\text{work}} = [(n-1)! / |St|] * (|St| + n^2/2 + 2*n) + n*(n-2)!
+ (n-3)! + (n^2 - 1.5*n)*(n-3)! + (n-3)!, \text{ or}
\]

\[
W_{\text{work}} = [(n-1)! / |St|] * (|St| + n^2/2 + 2*n)
+ (n-3)! * (2*n^2 - 3.5*n + 2)
\]

Not all arithmetic operations considered in the above formula are divided between \( N_p \) processes. When executed in parallel, the amount of work per one process is:

\[
W_{\text{work\_per\_proc}} = [(n-1)! / |St|] * (|St|/N_p + n^2/2 + 2*n)
+ {(n-3)! * (2*n^2 - 3.5*n + 2)}/ N_p \quad (15)
\]

Then the total amount of work done by a single process is

\[
W_{\text{total\_per\_proc}} = W_{\text{start}} + W_{\text{work\_per\_proc}}, \text{ or}
\]

\[
W_{\text{total\_per\_proc}} = |St|*[72 + 12*n]
+ [(n-1)! / |St|] * (|St|/N_p + n^2/2 + 2*n)
+ {(n-3)! * (2*n^2 - 3.5*n + 2)}/ N_p \quad (16)
\]
Table 14 compares the total amount of work per process obtained experimentally and using formula 16 for $U_{13}$.

Table 14. $W_{\text{total\_per\_proc}}$ for $U_{13}$

<table>
<thead>
<tr>
<th># processes, $N_p$</th>
<th>amount of work (# arith. op.)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>experimental</td>
<td>by formula 16</td>
</tr>
<tr>
<td>1</td>
<td>1,138,643,458</td>
<td>1,556,172,000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>570,694,237</td>
<td>782,330,400</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>388,667,409</td>
<td>524,383,200</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>307,043,642</td>
<td>395,409,600</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>248,938,990</td>
<td>318,025,440</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>204,302,979</td>
<td>266,436,000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>177,519,755</td>
<td>229,586,400</td>
<td></td>
</tr>
</tbody>
</table>

Although the values of $W_{\text{total\_per\_proc}}$ obtained experimentally and using formula 16 differ, the average rate of change of $W_{\text{total\_per\_proc}}$ as a function of $N_p$ is very close for both experimental results and those obtained by formula 16. This suggests that one can use formula 16 to generate results when questions about general tendency of rate of change $W_{\text{total\_per\_proc}}$ are discussed.

Graph 10 shows $W_{\text{total\_per\_proc}}$ for $U_{13}$ as a function of $N_p$; the results of Table 14 are used.
To make formula 16 more precise, it is adjusted to the experimental results by multiplying it by the constant, 1.32:

\[
W_{\text{total per proc}} = 1.32 \times \left\{ |St| \times [72 + 12n] + \frac{(n-1)!}{|St|} \times \left( |St|/N_p + n^2/2 + 2n \right) + \frac{(n-3)! \times (2n^2 - 3.5n + 2)}{N_p} \right\}
\]  

(17)

The results of Table 14 for the experimental \(W_{\text{total per proc}}\) and \(W_{\text{total per proc}}\) calculated by formula 17 are used to calculate the estimation error:

\[
\text{Error} = 100\% \times \frac{\text{experimental} - \text{by formula 17}}{\text{by formula 17}}
\]

does not exceed 4%.
The number of messages is another important measurement for the purpose of studying the performance of the parallel program. How does the number of messages increase as the number of processes grows? How does the number of messages change as a function of problem size? To answer these questions the following experiments are done.

The number of messages and the amount of work are measured explicitly in the program for $U_7, \ldots, U_{14}$ with the number of processes fixed and equal to two. When the number of messages is counted, MPI calls: Barrier, Broadcast, and Reduce, are counted as one message per a call. The results are provided in Table 15.

Table 15. Amount of Work and Number of Messages with $N_p = 2$ for $U_7, \ldots, U_{14}$

<table>
<thead>
<tr>
<th>$U_n$</th>
<th>amount of work, # arith. op.</th>
<th># messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_7$</td>
<td>5,689</td>
<td>15</td>
</tr>
<tr>
<td>$U_8$</td>
<td>21,282</td>
<td>249</td>
</tr>
<tr>
<td>$U_9$</td>
<td>69,432</td>
<td>887</td>
</tr>
<tr>
<td>$U_{10}$</td>
<td>531,166</td>
<td>2,383</td>
</tr>
<tr>
<td>$U_{11}$</td>
<td>4,468,991</td>
<td>4,146</td>
</tr>
<tr>
<td>$U_{12}$</td>
<td>50,459,701</td>
<td>4,020</td>
</tr>
<tr>
<td>$U_{13}$</td>
<td>570,694,237</td>
<td>544,326</td>
</tr>
<tr>
<td>$U_{14}$</td>
<td>38,197,866,776</td>
<td>162,586,584</td>
</tr>
</tbody>
</table>
The results of Table 15 are illustrated in Graph 11. The horizontal axis represents the value of \( n \), the length of the input cycle, and the vertical axis shows \( \ln \) of the values in the second and third columns of Table 15.

Graph 11. Work Decrease and Message Increase

From Graph 11, the number of messages increases slower than the amount of work for the values of \( n \) from 7 to 12, and faster for the values of \( n \) from 12 to 14. This suggests that if the number of messages increases faster than the amount of work for larger cycles, then parallelism would not work well on the problems of larger size. To determine if this is the case, the following model of calculation of the number of messages as a function of the number of processes and the length of the input size is proposed.
The sequential part requires two MPI calls. The number of MPI calls to calculate a single coset is equal to 
\((5+2*N_p)\), where \(N_p\) is the number of processes. And at the end of a single round there are four MPI calls. The number of messages, \(M_{sg}\), required to calculate \(|Q|\) single coset representatives in a single round is the function of the three variables: the length of the input cycle, \(n\), the order of the input stabilizer, \(|St|\), and the number of processes, \(N_p\).

\[
M_{sg} = 2 + |Q| \times (5 + 2 \times N_p) + 4, \text{ or }
\]

\[
M_{sg} = 6 + [(n-1)! / |St|] \times (5 + 2 \times N_p) \tag{18}
\]

Table 16 shows the number of messages calculated by formula 18 and the amount of work per one process calculated by formula 17 for two processes and \(U_7, ..., U_{20}\). It is assumed that the work is done in one round.
Table 16. Estimated Work and Number of Messages for U₇, ..., U₂₀

<table>
<thead>
<tr>
<th>Uₙ</th>
<th>Wtotal_per_proc, by formula 17</th>
<th>Msg, by formula 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>U₇</td>
<td>149,985</td>
<td>15</td>
</tr>
<tr>
<td>U₈</td>
<td>97,392</td>
<td>124</td>
</tr>
<tr>
<td>U₉</td>
<td>199,426</td>
<td>846</td>
</tr>
<tr>
<td>U₁₀</td>
<td>1,183,248</td>
<td>2,274</td>
</tr>
<tr>
<td>U₁₁</td>
<td>10,046,203</td>
<td>4,130</td>
</tr>
<tr>
<td>U₁₂</td>
<td>112,892,314</td>
<td>3,786</td>
</tr>
<tr>
<td>U₁₃</td>
<td>1,032,676,128</td>
<td>544,326</td>
</tr>
<tr>
<td>U₁₄</td>
<td>13,918,566,182</td>
<td>38,918,886</td>
</tr>
<tr>
<td>U₁₅</td>
<td>221,795,048,724</td>
<td>1,816,214,406</td>
</tr>
<tr>
<td>U₁₆</td>
<td>3,464,589,966,520</td>
<td>30,648,618,006</td>
</tr>
<tr>
<td>U₁₇</td>
<td>50,604,382,955,174</td>
<td>261,534,873,606</td>
</tr>
<tr>
<td>U₁₈</td>
<td>784,411,085,814,682</td>
<td>1,482,030,950,40</td>
</tr>
<tr>
<td>U₁₉</td>
<td>13,625,596,844,994,800</td>
<td>10,003,708,915,2</td>
</tr>
<tr>
<td>U₂₀</td>
<td>253,798,096,671,679,000</td>
<td>47,517,617,347,2</td>
</tr>
</tbody>
</table>

Graph 12 demonstrates the results of Table 16. The horizontal axis is the length of the input cycle, n, and the vertical axis is ln of the values of the second and third columns of Table 16.
Graph 12. Estimated Work and Messages for $N_p = 2$

The shape of the two curves in Graph 12 retains the same pattern for $n$ from 7 to 14 as that in Graph 11. For the values of $n$ from 12 to 15 the number of messages increases faster than the amount of work, for $n$ from 15 to 17 the number of messages and the amount of work have the same rate of change, and for $n$ from 17 to 20 the amount of work increases faster than the number of messages. This observation suggests that parallelism might be more efficient for the cycles of length larger than 17. The following experiment provides an additional proof for the conclusion above.
The number of single coset representatives to be found is fixed in the experiment, and the program is executed with different number of processes for each of $U_{15}, \ldots, U_{20}$. The number of processes increases until there is no improvement in the running time, i.e. until the running time with $N_{p+1}$ process is greater than the running time with $N_p$ processes. The experiment shows that the running time with the number of processes equal to two for $U_{15}, \ldots, U_{17}$ is greater than the running time with a single process for these input cycles. The results of the experiment for $U_{18}, \ldots, U_{20}$ are given in Table 17. For $U_{18}$ and $U_{19}$, the number of calculated single coset representatives is 10,000, and for $U_{20}$, it is 1,000.
Table 17. Running Time with Fixed Number of Calculated Single Coset Representatives for $U_{18}$, $U_{19}$, and $U_{20}$

<table>
<thead>
<tr>
<th># processes, $N_p$</th>
<th>running time in sec, $t_{real}$, $U_{18}$</th>
<th>running time in sec, $t_{real}$, $U_{19}$</th>
<th>running time in sec, $t_{real}$, $U_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47.3</td>
<td>126.5</td>
<td>54.04</td>
</tr>
<tr>
<td>2</td>
<td>37.8</td>
<td>77.5</td>
<td>29.7</td>
</tr>
<tr>
<td>3</td>
<td>38.8</td>
<td>65.1</td>
<td>22.3</td>
</tr>
<tr>
<td>4</td>
<td>52.2</td>
<td>71.5</td>
<td>19.2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>66.2</td>
<td>18.6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>69.2</td>
<td>17.9</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>154.5</td>
<td>19.8</td>
</tr>
</tbody>
</table>

The observation of the results of the latter experiment implies that the less the order of the input stabilizer, the less efficiency is achieved by parallelism. Table 18 provides the running time in seconds for $U_{11}$, ..., $U_{20}$ for one and two processes, and evaluates speedup and efficiency for these cases. For $U_{11}$, ..., $U_{13}$ all single coset representatives are found. For $U_{15}$ and $U_{16}$, the number of calculated single coset representatives is 100,000; for $U_{14}$, $U_{17}$, $U_{18}$, and $U_{19}$, the number of calculated single coset representatives is 10,000; and for $U_{20}$, it is 1,000.
Table 18. \( t_{\text{real}} \), Speedup, and \( \varepsilon \) for \( U_{11}, \ldots, U_{20} \) and \( N_p = 2 \)

| \( U_n \) | \( |\text{St}| \) | \( N_p \) | \( t_{\text{real}} \), sec | Speedup | \( \varepsilon \) |
|-----|-----|-----|-----------|--------|--------|
| \( U_{11} \) | 7,920 | 1 | 7.17 | 1.50 | 0.75 |
| | | 2 | 4.783 | | |
| \( U_{12} \) | 95,040 | 1 | 82.298 | 1.88 | 0.94 |
| | | 2 | 43.714 | | |
| \( U_{13} \) | 7,920 | 1 | 882.26 | 1.69 | 0.84 |
| | | 2 | 522.88 | | |
| \( U_{14} \) | 1,440 | 1 | 33.97 | 1.14 | 0.57 |
| | | 2 | 29.7 | | |
| \( U_{15} \) | 432 | 1 | 112.5 | 0.58 | 0.29 |
| | | 2 | 193.5 | | |
| \( U_{16} \) | 384 | 1 | 114.9 | 0.58 | 0.29 |
| | | 2 | 198.0 | | |
| \( U_{17} \) | 720 | 1 | 19.3 | 0.83 | 0.42 |
| | | 2 | 23.2 | | |
| \( U_{18} \) | 2,160 | 1 | 47.3 | 1.25 | 0.63 |
| | | 2 | 37.8 | | |
| \( U_{19} \) | 5,760 | 1 | 126.5 | 1.63 | 0.82 |
| | | 2 | 77.5 | | |
| \( U_{20} \) | 23,040 | 1 | 54.0 | 1.82 | 0.91 |
| | | 2 | 29.7 | | |

Graph 13 uses the results of Table 18 to show efficiency, \( \varepsilon \), as a function of an order of a stabilizer.
Graph 13. Efficiency as Function of Stabilizer's Order

Graph 13 shows that the efficiency enhances as an order of a stabilizer increases. However, it is only a suggestion based on the condition used in this experiment. For example, when the efficiency for $U_{14}$ is measured with 10,000 calculated orbit representatives, the efficiency is 0.57. When the program is executed with 2 processes, and all orbit representatives (4,325,472) are found, the efficiency is 0.74. So, efficiency depends not only on a stabilizer's order but also on a problem size. From Table 13, the stabilizers for $U_{11}$ and $U_{13}$ have the same order, but efficiency is greater for $U_{13}$ because the amount of work for $U_{13}$ is larger.
Speedup and efficiency are calculated using the running time. The running time depends on the two major factors: the speed of the calculating single cosets, Speed\textsubscript{orb}, and the number of rounds. Speed\textsubscript{orb} calculated by the formula 3 is not the actual speed but the average speed, which is significantly less than the actual speed. For example, to calculate the actual speed for U\textsubscript{14} on a single process, the values from Table 13 are used. To calculate 10,000 orbit representatives, it is required 33.97 seconds on a single process for U\textsubscript{14}. Then the actual speed is equal to 10,000/33.97, or 294.37 single coset representatives per second. The running time to calculate all single coset representatives for U\textsubscript{14}, 4,325,472, is 14 hours and 45 minutes. Thereby, because of the recalculation, the average speed calculated using formula 3 is 81.45 orbit representatives per second. To improve the average speed, Speed\textsubscript{orb}, it is necessary to decrease the number of rounds of recalculation. It is possible to achieve this by making the bitset described in Chapter 5 as large as possible. The limitation of the choice of a bitset’s size is memory size. Another possibility to increase a bitset’s size is to use a larger number of processes; however, taking into account that each process
has a bitset of the maximum allowed size, the processes must run on different machines. For example, if a single process has a bitset with the size $L$, then $N_p$ processes can process the bitset of the size $L \times N_p$, which might reduce the number of rounds, and consequently improve the running time.

The experiments and the theoretical discussion considered so far suggest that utilization of the larger number of processes for the $U_n$ with stabilizer's order less than 1000 does not improve parallelism. The remedy to this situation is not dividing the stabilizer between the processes. The number of messages will decrease, and processing the larger bitset of the size $L \times N_p$, will decrease the number of rounds, improving the execution time.

So far, the efficiency and speedup of the parallel program have been discussed. Another important performance metric of a parallel program is scalability. A parallel program is said to be scalable if efficiency is the same when the amount of work per processor is the same. In our case, the closer the values of $n$, the length of the input cycle, the closer the values of $W_{\text{total\_per\_proc}}$, the amount of work per process calculated by formula 16. For example, from Table 16, $W_{\text{total\_per\_proc}}$ for $U_{12}$ and two processes is

80
85,524,480. Using formula 16, it is determined that to achieve the same value of $W_{\text{total per proc}}$ for $U_{13}$ and two processes, 20 processes are necessary. From Table 18, the efficiency for $U_{12}$ and $N_p = 2$ is 0.94, and the efficiency for $U_{13}$ and $N_p = 7$ is 0.20. From Table 13, for $U_{13}$, the efficiency decreases as $N_p$ increases. It follows that the efficiency for $U_{13}$ and $N_p = 20$ will be less than 0.2. Comparing $\varepsilon(U_{12}, N_p = 2) = 0.94$ and $\varepsilon(U_{13}, N_p = 20) < .20$, it is obvious that the efficiency for these cases is not the same. For the cases when $n$ differs more than one digit, more processes are needed to make the amount of work per process approximately equal. As can be seen from the discussion in this subsection, efficiency decreases as the number of processes increases. Therefore, this parallel program is not scalable.

6.4 Time and Resources

To estimate the minimum running time to find all single coset representatives in the double coset $M_{24} U_n M_{24}$ for $n$ equal to 15, 16, 17, 18, 19, and 20; the results of Table 17 are used. It is assumed that for $U_{15}$, $U_{16}$ and $U_{17}$ a single process is used, for $U_{18}$ two processes, for $U_{19}$ three processes, and for $U_{20}$ six processes. It is assumed that all
single coset representatives are found in one round. The speed of calculation a single coset is estimated by the formula

$$\text{Speed}_{\text{icos}} = \frac{T_{\text{exc}}}{|Q|}$$  \hspace{1cm} (19),

where $|Q|$ and $T_{\text{exc}}$ are the number of single cosets and execution time respectively from Table 17 for $U_{18}$, $U_{19}$, and $U_{20}$, and from Table 18 for $U_{15}$, $U_{16}$ and $U_{17}$. $\text{Speed}_{\text{icos}}$ is measured in seconds per a single coset.

Table 19 contains the results of calculation $\text{Speed}_{\text{icos}}$ for $U_{15}, \ldots, U_{20}$.

<table>
<thead>
<tr>
<th>$U_{15}$</th>
<th>$U_{16}$</th>
<th>$U_{17}$</th>
<th>$U_{18}$</th>
<th>$U_{19}$</th>
<th>$U_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.125</td>
<td>1.149</td>
<td>1.930</td>
<td>3.780</td>
<td>6.510</td>
<td>17.900</td>
</tr>
</tbody>
</table>

Then the execution time is found as a product of speed of calculation a single coset and approximate number of single cosets in the double coset $M_{24} U_n M_{24}$, where the latter is estimated as $(n-1)!/|St|$. Table 20 shows the execution time estimated by formula:

$$T_{\text{exc}} = \text{Speed}_{\text{icos}} \times \frac{(n-1)!}{|St|}$$  \hspace{1cm} (20),

where $\text{Speed}_{\text{icos}}$ is taken from Table 19, and $|St|$ is from Table 18. Table 20 gives the results of $T_{\text{exc}}$ converted in
days, and Table 21 shows the same results measured in years.

Table 20. Estimated $T_{exc}$ in Days

<table>
<thead>
<tr>
<th></th>
<th>$U_{15}$</th>
<th>$U_{16}$</th>
<th>$U_{17}$</th>
<th>$U_{18}$</th>
<th>$U_{19}$</th>
<th>$U_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.62</td>
<td>45.28</td>
<td>649.12</td>
<td>7,204.31</td>
<td>83,750.18</td>
<td>1,093,834.04</td>
</tr>
</tbody>
</table>

Table 21. Estimated $T_{exc}$ in Years

<table>
<thead>
<tr>
<th></th>
<th>$U_{17}$</th>
<th>$U_{18}$</th>
<th>$U_{19}$</th>
<th>$U_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.78</td>
<td>19.74</td>
<td>229.45</td>
<td>2,996.81</td>
</tr>
</tbody>
</table>

Table 22 presents the estimation of hard disk resources in gigabytes needed to keep the output files for $U_{15}, \ldots, U_{20}$. It is assumed that it is needed $(n+1)$ bytes to save a single coset representative presented as a string of $n$ characters, and there are $(n-1)!/|St|$ such representatives for each $U_n$. 
Table 22. Hard Disk Space in GB Needed for Output

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_{15}$</td>
<td>$U_{16}$</td>
<td>$U_{17}$</td>
<td>$U_{18}$</td>
<td>$U_{19}$</td>
<td>$U_{20}$</td>
</tr>
<tr>
<td>3.23</td>
<td>57.89</td>
<td>523.07</td>
<td>3,128.73</td>
<td>22,230.46</td>
<td>110,874.44</td>
</tr>
</tbody>
</table>

The actual execution time must be at least ten times greater than the estimated time given in Table 20 due to recalculations. It is feasible to complete the calculations for $U_{15}$ and $U_{16}$ in terms of execution time, but in terms of disk space, the Raven Cluster has a total of 20 GB of hard disk space, and it is not sufficient to hold the output for $U_{16}$. 
The important contribution of this research is that it has been demonstrated that there are ways of improving a straightforward algorithm that finds single coset representatives in the double coset $M_{24} U_n M_{24}$.

The limitation of the straightforward algorithm that uses Magma’s functions is that it cannot find single coset representatives in the double coset $M_{24} U_n M_{24}$ for $U_n$ with $n$ equal to or greater than 12. The work of this thesis shows that it is possible to find single coset representatives in the double coset for $U_n$ with $n$ greater than 11. Using the proposed programs, the problem has been solved for $U_{12}$, $U_{13}$, $U_{14}$, and $U_{15}$. For example, 201,802,032 single coset representatives have been found in the double coset $M_{24} U_{15} M_{24}$, which required 14 days and 6 hours to run using four processors on the Raven Cluster.

The proposed program works slower than Magma’s program for $n$ smaller than 9. The results show that the proposed program works faster than Magma’s program for $U_n$ with $n$ equal to or greater than 9. For example, for $U_9$ the proposed
program is 4.34 times faster, and for $U_{10}$ is 17.28 times faster than Magma's program.

Even though the parallel program is not scalable, it is possible to achieve Speedup of 2 for $U_n$ with the stabilizer's order larger than 7,000.

The program for the proposed algorithm is written for the cycles of structure $[n]$. Future studies may explore the algorithm for the cycles of more complex structure.

Finally, this research has shown that even an improved straightforward algorithm does not help to solve the problem of finding the single coset representatives in the double coset $M_{24} U_n M_{24}$ for $U_n$ of order larger than 16 due to the limitation of second storage. This suggests that future studies might investigate other ways of solving this problem.
APPENDIX A

SETS OF THE TYPE $U_n$
In this appendix, sets of the type $U_n$ that are used in the proposed program and the corresponding stabilizers' orders, $|\text{St}|$, are provided.

| $U_n$ | The elements of $U_n$ set | $|\text{St}|$ |
|-------|-----------------------------|-----------|
| $U_6$ | 4, 8, 14, 17, 23, 24        | 1152      |
| $U_7$ | 4, 7, 8, 14, 17, 23, 24     | 720       |
| $U_8$ | 2, 4, 7, 8, 14, 17, 23, 24  | 384       |
| $U_9$ | 2, 4, 7, 8, 14, 17, 22, 23, 24 | 432   |
| $U_{11}$ | 1, 2, 4, 7, 8, 14, 17, 22, 23, 24 | 1440  |
| $U_{11}$ | 1, 2, 4, 7, 8, 12, 14, 17, 22, 23, 24 | 7920  |
| $U_{11}$ | 1, 2, 4, 7, 8, 12, 14, 17, 21, 22, 23, 24 | 9504  |
| $U_{11}$ | 3, 5, 6, 9, 10, 11, 13, 15, 16, 18, 19, 20, 21 | 7920  |
| \( U_n \) | The elements of \( U_n \) set | \(|\text{St}|\) |
|---|---|---|
| \( U_1 \) | 3, 5, 6, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20, 21 | 1440 |
| \( U_1 \) | 1, 3, 5, 6, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20, 21 | 432 |
| \( U_1 \) | 1, 3, 5, 6, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20, 21, 22 | 384 |
| \( U_1 \) | 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18, 20, 21, 23 | 720 |
| \( U_1 \) | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18, 20, 21, 22 | 2160 |
| \( U_1 \) | 1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22 | 5760 |
| \( U_2 \) | 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22 | 2304 |
| \( U_2 \) | 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22 | 0 |
APPENDIX B

MAGMA'S PROGRAM
In this appendix an example of Magma's program used in this thesis is provided. This program calculates and prints into an output file single coset representatives in the double coset $M_{24} U_8 M_{24}$.

```plaintext
s:=SymmetricGroup(24);
a:=s!(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24);
b:=s!(1,2);
s:=sub<s|a,b>;
alp:=s!(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23);
bet:=s!(15,7,14,5,10,20,17,11,22,21,19)(3,6,12,1,2,4,8,16,9,18,13);
del:=s!(1,18,4,2,6)(8,16,13,9,12)(14,17,11,19,22)(20,10,7,5,21);
M:=sub<s|alp,bet,gam,del>;
u8:={2,4,7,8,14,17,23,24};
yu8:=s!(24,23);
S8:=SymmetricGroup(8);
S8:=sub<s|u8, yu8>;
D:=Classes(S8);
X:={};
X:=X join {D[19][3]};
set_all_permut:=Class(S8, Setseq(X)[1]);
stab:=Stabiliser(M, u8);
N:=1;
while set_all_permut ne {} do
    single_coset:={};
single_coset:=Setseq(set_all_permut)[1]^stab;
set_all_permut:= set_all_permut diff single_coset;
N:=N+1;
coset_repres:=Setseq(single_coset)[1];
PrintFileMagma("orbits_8_magma.txt", coset_repres);
end while;
```
REFERENCES


