dapat diakses melalui http://ejournal.unsrat.ac.id/index.php/jmuo

Construction of Error-Correction Code Application by Applying Finite Field and Hadamard Matrix Theory

Robinson Pongoh[a]*, Benny Pinontoan[a], Winsy Weku[a]

[a]Jurusan Matematika, FMIPA, Unsrat, Manado

KATA KUNCI

Kode perbaikan-galat
Matriks
Lapangan

ABSTRAK

Dalam dunia elektronik dan digital, informasi dapat dengan mudah ditransfer melalui saluran komunikasi. Pada data yang ditransfer galat dapat muncul dikarenakan oleh berbagai akibat. Untuk menghindari masalah ini diperlukan kode perbaikan-galat bersama aplikasinya. Konsep dari kode perbaikan-galat adalah untuk menambahkan bit-tambahan pada data agar disaat pengiriman, data tersebut lebih kuat dalam menghadapi gangguan yang hadir di saluran komunikasi. *Random Parity Code* (RPC) yang dikemukakan oleh Hershey dan Tiemann (1996) adalah salah satu dari kode yang dimaksud. Artikel ini menunjukan pembuatan aplikasi kode perbaikan-galat yang dibuat berdasarkan konsep RPC dengan bantuan teori Lapangan Terbatas dan Matriks Hadamard. Aplikasi dibuat menggunakan Metode *Rapid Application Development* (RAD). Aplikasi dihasilkan dalam bentuk perangkat lunak komputer. Perangkat lunak tersebut menjadi lebih efisien dengan menerapkan konsep algoritma “*Divide and Conquer*”.

KEYWORDS

Error-correction code
Matrix
Field

ABSTRACT

In the world of digitals and electronics, information could be easily transferred via communication channel. The transferred data might be or not containing error that rises by variety of causes. To avoid this matter, error-correction code is needed along with its efficient application. Error-correction coding is a concept to add extra data bits to makes the transmission of data or information more robust to disturbances present on the communication channel. One of them is the *Random Parity Code* (RPC) introduced by Hershey and Tiemann (1996). This article shows the construction of error-correction code application that is made based on RPC concept with the help of finite field and hadamard matrix theory. The application is made using Rapid Application Development (RAD) method. The produced application is in form of computer software. The software become more efficient by applying the divide and conquer algorithm concept.

1. Introduction

Most of people now use computer, people like to use computer because of its advantage to be more efficient and effective for doing any tasks. Another advantage of using computer is, any files (this include information and communication) within the computer could be easily managed and shared or transferred by copying or sending via communication channel such as cable, wireless network, hardware drive or other devices. But, despite of its capability to transfer file easily, the transfer process of files (or information or communication) not always come in handy. The files had to going through a communication channel when it progressing the transfer process. The file resulted from going through this communication channel might be different from the initial file. The transferred file may or may not contain error. Errors from the communication channel are rises by variety of causes such as lightning or weather failure, disc scratches, hardware failure, etc. To avoid this matter, error-correction code was invented.

Error correction code consists of encoding and decoding process. When encoding, the message we decide to encode will be encoded into codeword using

*Corresponding author: Jurusan Kimia FMIPA UNSRAT, Jl. Kampus Unsrat, Manado, Indonesia 95115; Email address: ynzonronald@gmail.com

Published by FMIPA UNSRAT (2012)

encoding method. This codewords will be sent to go through a channel, this channel assumed to be noisy. The channel could be a cable or wireless network, hardware drive or other devices. When the code received, it may contain errors or perhaps not. The received code then will be decoded. Due to redundancy and decoding method, original message could be recovered. The illustration of communication channel with and without coding process can be seen in Figure 1 and Figure 2.

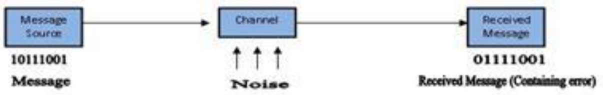


Figure 1 – Normal communication channel.

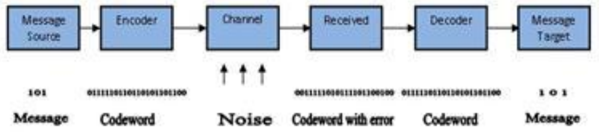


Figure 2 – Communication channel with coding.

In this research an error-correction code application is made, the theory for making the application based on Random Parity codes of J. Hershey and J. Tiemann introduced in 1996. The Encoding and Decoding concept can be seen in Figure 3 and Figure 4.

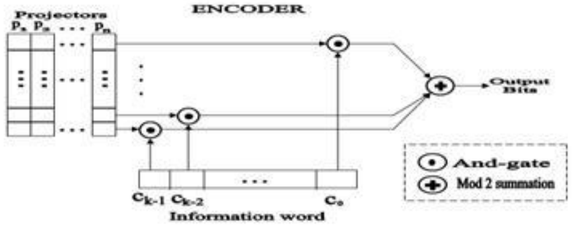


Figure 3 – Random parity encoder (Hershey and Tiemann, 1996).

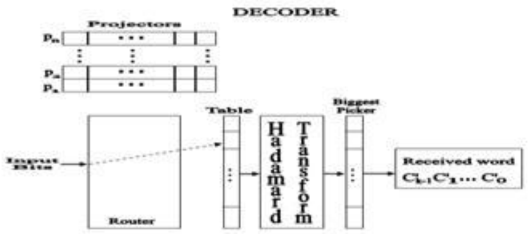


Figure 4 – Random parity decoder (Hershey and Tiemann, 1996).

In this research, the error-correcting code application will be made by applying some coding theory with the support of matrix, and field theory. In the matrix theory there is Hadamard matrices, a square matrix of order n that is an $n \times n$ matrix with integer entries ± 1 satisfying: $H_nH_n^T = H_n^TH_n = nI$; H_n^T is the transpose of H_n and I is the $n \times n$ identity matrix. Hadamard matrices is an application of finite field that have a special form and are named to J. Hadamard, because of the property of their determinants which attain a bound originally given by J. Hadamard in 1893 (Agaian, 1985)

The objectives of this research are to produce an error-correction code application and to see the application efficiency after implemented with divide and conquer algorithm concept.

The results of research hopefully will have some benefits, such as the produced application can be used to avoid error on file transmission, the produced application can be implemented on any devices that require transmission, the produced application can be used as a base design or blueprint to make another error-correction code or even its implementation.

2. Methods

2.1. Software Development Method

The error-correction code application in this research developed using one of the Software Development Method (SDM)). The method used in this research is Rapid Application Development (RAD), developed by James Martin during the 1980s at IBM and finally formalized it by publishing *Rapid Application Development* book in 1991. The model can be seen in Figure 5.

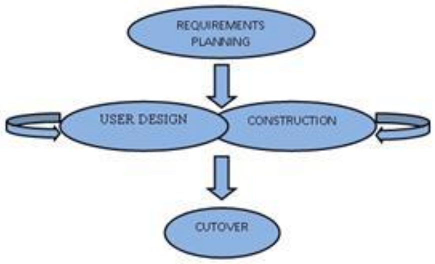


Figure 5 – RAD model.

2.2. Compiler

The compiler used for constructing the application is Borland Delphi 7 (Build 4.453), based on Pascal Programming Language.

3. Results and Discussion

3.1. Planning and Requirements

The target user is the computer user that wanted to avoid corrupted file as a result of the file transfer. This research requires computer plus its operating system and compiler.

3.2. Design

The design separated into design diagram that visualize the model of the program, and design description. The design diagram of the RPC application modeled using Unified Modeling Language (UML).

3.2.1. Design Diagram

The design diagram in UML is represented into two different views: the static (or structural) view and dynamic (or behavioral) view.

Structural Diagram

There are 4 types structural diagram for modeling the program, they are:

a. Class Diagram

This diagram describes the structure of RPC program by showing its classes, attributes, operations, and the relationships among the classes, as in Figure 6 to 9.

b. Package Diagram

This diagram describes how a system is split up into logical groupings by showing the dependencies among these groupings, as in Figure 10.

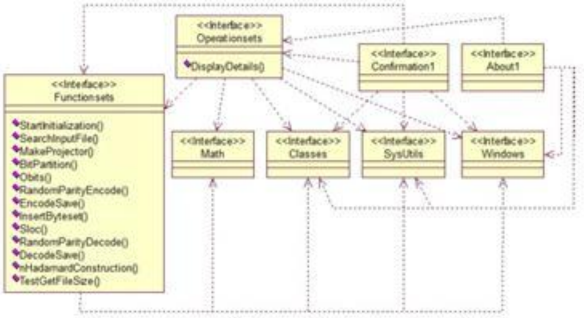


Figure 6 – Primary class diagram.

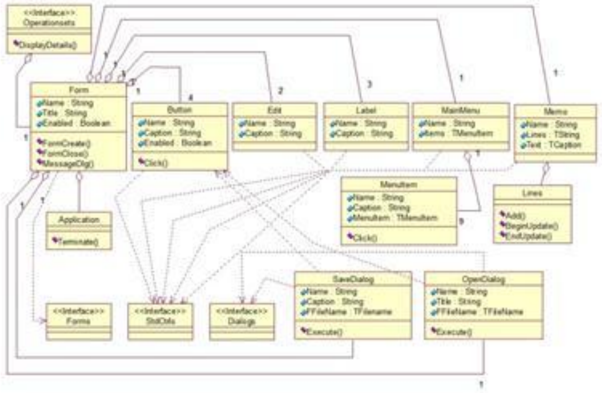


Figure 7 – Operationsets.pas class diagram.

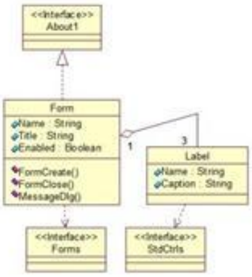


Figure 8 – About1.pas class diagram.

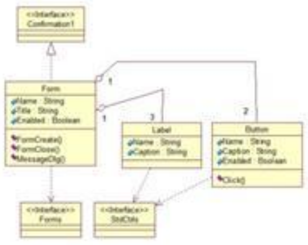


Figure 9 – Confirmation1.pas class diagram.

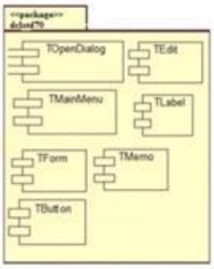


Figure 10 – Package Diagram.

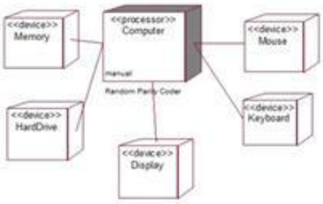


Figure 11 – Deployment diagram.

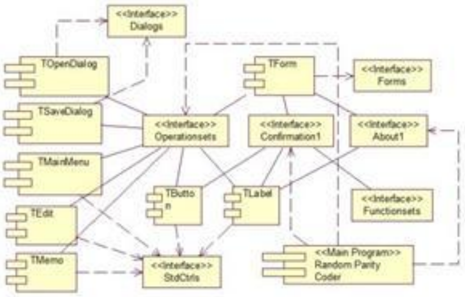


Figure 12 – Component diagram.

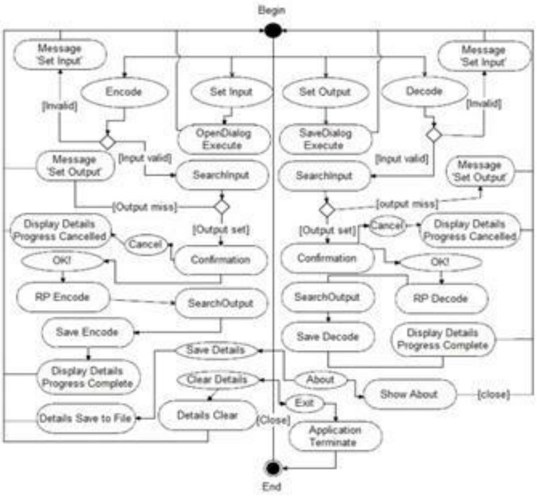


Figure 13 – Activity diagram.

c. Deployment Diagram

This diagram describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware, as in Figure 11.

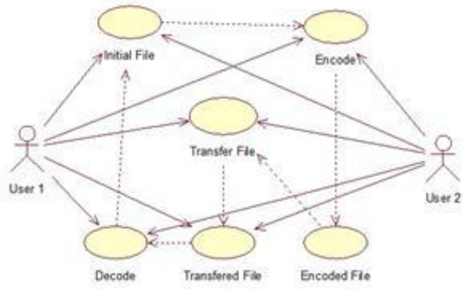


Figure 14 – Use case diagram.

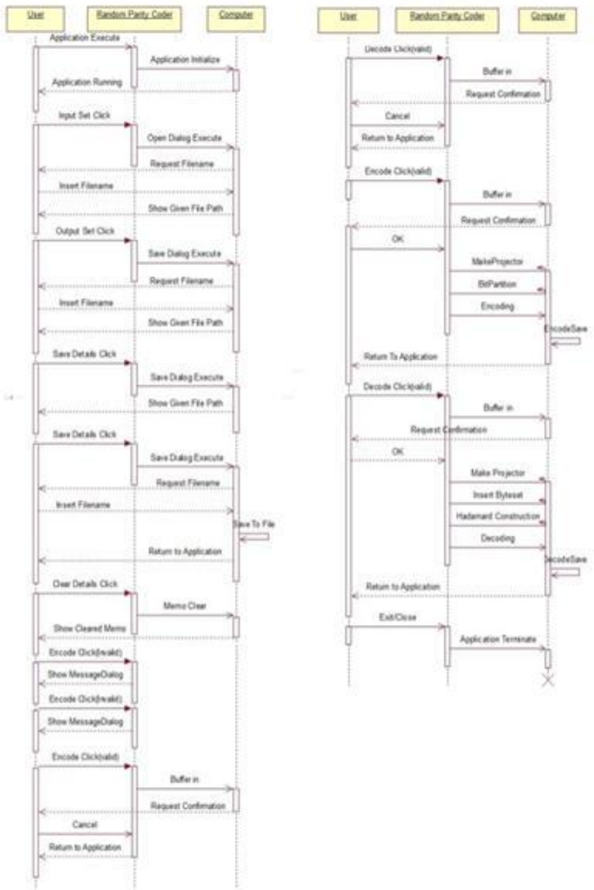


Figure 15 – sequence diagram.

d. Component Diagram

This diagram describes how random parity code is split up into components and shows the dependencies among these components, as in Figure 12.

Behavior Diagram

There are 3 types behavior diagram for modeling the RPC application, they are:

a. Activity Diagram

This diagram describes the operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control, as in Figure 13.

b. Use Case Diagram

This diagram describes the functionality provided by RPC in terms of actors and their goals that represented as use cases, also any dependencies among them, as in Figure 14.

c. Sequence Diagram

This diagram shows how objects communicate with each other in terms of a sequence of messages. Also indicates the life spans of objects relative to those messages, as in Figure 15.

3.2.2. Design Description

The design of the application contain of the *program* as the main executor and the *unit* as a part of the main program.

Program Description

Main program is the main executor of the application that initializes the application and run all units within it.

Unit Description

Units are the individual source code modules that make up a program or in other words it is the constructor of main program. A unit is a place to group functions and procedures that can be called from main program. A unit consists of at least 3 parts such as: unit statement as the identifying part, interface part as the connector part to other unit or program, implementation part as the definer of procedures and functions. In this program 4 unit construct the main program: operationsets, functionsets, confirmation1 and about1. Unit functionsets is the main focus of the research, especially the procedure Random Parity Encode and Random Parity Decode as the implementation of RPC.

```
procedure RandomParityEncode;
var
i,j,k,l:integer; tempbits:array[0..7] of byte;
tmpbyte:byte;
begin
makeprojector; bitpartition;
for i:=0 to ibytesread-1 do
begin for k :=0 to 3 do
begin for j:=0 to 7 do
begin
tempbits[j]:=obits(projector[j],zlocation[i,k]);
end; l:=1; tmpbyte:=0;
tmpbyte:= tmpbyte or tempbits[0];
repeat
tmpbyte:=(tmpbyte shl 1) or tempbits[l];l:=l+1;
until l=7;
bufferin[(i·4)+k]:=tmpbyte;
end; end; end;
```

```
procedure RandomParityDecode;
var
bitsin:array[0..7] of byte; Table:array[0..3] of
integer; resultvector:array[0..3] of integer;
i,j,k,m:integer;
begin
makeprojector;insertbyteset;
for j:=0 to ibytesread-1 do
begin
for k:=0 to 3 do
begin
zeromemory(@bitsin,sizeof(bitsin));
bitsin[0]:= (byteset[j,k] and 1);
for i:=1 to 7 do bitsin[i]:= ((byteset[j,k] shr i)
and 1);
for i:=0 to 3 do Table[i]:=0;
for i:=0 to 7 do begin
if bitsin[i]=0 then
Table[projector[i]]:=Table[projector[i]]+1 else
if bitsin[i]=1 then
Table[projector[i]]:=Table[projector[i]]-1;end;
nhadamardconstruction;
for i:=0 to 3 do begin
resultvector[i]:= hadamard[i,0] · Table[0];
for m:=0 to 3 do
resultvector[i]:=resultvector[i] +(hadamard[i,m] ·
Table[m]); end;
zlocation[j,k]:=byte(sloc(resultvector));
if zlocation[j,k]=1 then zlocation[j,k]:=3 else if
zlocation[j,k]=3 then zlocation[j,k]:=1; end;end;
for i:=0 to ibytesread-1 do
begin bufferin[i]:=0; bufferin[i]:=(bufferin[i] or
zlocation[i,3]);
for m:=2 downto 0 do bufferin[i]:=(bufferin[i] shl
(2)) or zlocation[i,m]; end;end;
```

3.3. Construction

The RPC application was built using Compiler in this case Delphi 7 and program source code based on Pascal Programming Language. The application constructed based on the design that have been explained before.

3.4. Cutover and Finalization

This section describes implementation.

Implementation

a. Changes from the original concept

To show what changes implementations are made to the original concept, the original one is shown first.

1. Original concept:

i. Binary input for info word is any of the bit size (1-bit to *n*-bit).

ii. Projector address *x* generated with conditions:

- Pseudo randomly generated.
- Cannot be a zero or 0.
- $x \in A$ with element of *A* from 1 to *n*-bit data size – 1 or $A=\{1,\ldots,2^n-1\}$

Example: $n = 2$; Size(n) = $2^n = 2^2 = 4$; $A = \{1,2,3\}$.

$n = 8$; Size(n) = $2^8 = 256$; $A = \{1,2,3,\ldots, 255\}$

iii. Number of Projector : any (affected by the input, usually at least 2^n+1)

b. Changes and implementation:

- Data type used in program is set to 8-bit
- Number of bit input for info word is 2-bit in 4 parts, then $n = 2$.
- Projector addresses generated = {1, 2, 3}.
- Projector generated = $2^{n+1} = 2^3 = 8$

b. Algorithms Implementation

There are some algorithms implemented to the program. Total of 10 pseudocodes of the algorithms can be seen below.

```
//Divide Input data n-bit to m parts of c-bit and
allocate it to array.
//Input: Integer or byte
//Output: array of byte/integer

ByteDivision(B)
For i ← 0 to m do P[i] ← Bitcheck(B,(c*i),c)
```

```
//Return Output bits 1 or 0 by processing 2 inputs
//Input: two integer or byte x and y
//Output: 1 or 0

Outputbits(x,y)
B ← x and y
For i ← 0 to n-1 do OB[i] ← Bitcheck(B,i,1)
Return (Sum(OB)) mod 2
```

```
//Generate n Projector with address for Random
Parity Coding process
//Input: integer n
//Output: array P of byte/integer

MakeProjector(n)
For i ← 0 to n-1 do P[i] ← ProjectorAddress
```

```
//Return Sum of array
//Input: an Array A
//Output: integer or byte

Sum(A[0..n-1])
S ← 0
Repeat
S ← S + A[i]
i ← i + 1
Until i = n
Return S
```

```
//Divide the B array by m part and allocate it to
2-dimension array
//Input: Array B of integer/byte
//Output: 2-dimension Array BS of integer/byte

ByteSet(B[0..n-1)
For i ← 0 to (n-1)div m do
For j ← 0 to m-1 do
BS[i,j] ← B[(m*i)+j]
```

```
//Return location of maximum value of an Array
//Input: Array A of integer/byte
//Output: integer or byte

SearchMaxLocation(A[0..n-1)
m ← Max(A[0..n-1)
For i ← 0 to n-1 do If A[i] ← m then loc ← i
Return loc
```

```
//Encode Input Infoword into Output Codeword
//OB binary length = 2^IB+1 binary length
//Input: integer of byte B
//Output: integer or byte OB

RandomParityEncode(B)
For i ← 0 to n-1 do IB[i] ← OutputBits(B,P[i])
For i ← n-1 downto 0 do OB ← Insertbits(IB[i])
Return OB
```

```
//Decode Input codeword into Output infoword
//Input: integer or byte B
//Output: Location on Maximum value, integer or
byte

RandomParityDecode(B)
For i ← 0 to n-1 do
Table[i] ← 0
For i ← 0 to n-1 do
If Bitcheck(B,i,1)= 1 then dec(Table[P[i]]) else
If Bitcheck(B,i,1)= 0 then inc(Table[P[i]])

RV ← MatrixMultiplication(Table,H)
Return SearchMaxLocation(RV)
```

```
//Construct Hadamard Matrix order n
//Input: integer n
//Output: Matrix H order n

HadamardConstruction(n)
H[0,0] ← 1
If n<2 then return H else
k ← 1
Repeat
a ← (2^k)-1
For i ← 0 to a-1 do
For j ← 0 to a-1 do
            Begin
            H[i,(j+a)] ← H[i,j]
            H[(i+a),j] ← H[i,j]
            H[(i+a),(j+a)] ← -H[i,j]
            End
k ← k+1
Until k = n
```

```
//Multiplies Matrix X times Matrix Y
//Input: 2 m × n Matrices X and Y
//Output: Matrix Z = XY

MatrixMultiplication(X,Y)
For i ← 0 to m-1 do
For j ← 0 to n-1 do
Z[i,j] ← 0
            For k ← 0 to p-1 do
            Z[i,j] ← Z[i,j] + X[i,k] * Y[k,j]
Return Z
```

Software Correctness and Efficiency

The correctness verification will be done using experimental analysis. The efficiency will be determined by seeing the program run-time and storage usage capability.

a. Software correctness verification (using experimental analysis)

The process is correct if $d = 0$ for $e \leq 1$, with d = binary distance of encode input and decode output; e = number of error; n = number of letter input.

Tabel 1 – Correctness Table.

Input	Output	*n*	e	D
123	123	3	0	0,0,0
123abc	123abc	6	0	0,0,0,0,0,0
123abc!@#	123abc!@#	9	0	0,0,0,0,0,0,0,0,0
123	123	3	1	0,0,0
123abc	123abc	6	1	0,0,0,0,0,0
123abc!@#	123abc!@#	9	1	0,0,0,0,0,0,0,0,0

According to the Table 1, the whole process is correct because $d = 0$ for $e \leq 1$

b. Software Efficiency

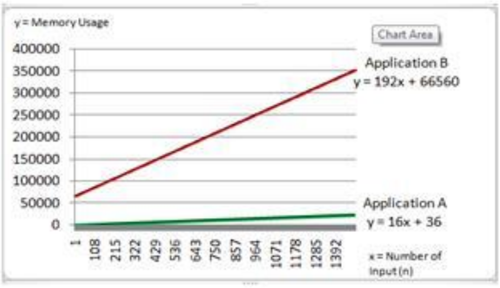
This section shows that the software become more efficient by implementing the Byte Division algorithm than before implementing it, Application A with implementation and Application B without it.

1. Storage/Memory Usage

The memory usage of each application is calculated according to procedures, functions and operations within each of it.

Tabel 2 – Memory usage of application A and B

Procedure	A	B
Projector	8	512
Codeword buffer out	4(*n*)	64(*n*)
Byte Division	4(*n*)	0
Codeword buffer in	4(*n*)	64(*n*)
Byte Set	4(*n*)	64(*n*)
Table	4	256
Result Vector	4	256
Hadamard Matrix	16	65536
Total	36+16(*n*)	66560+192(*n*)
Initial Memory usage (*n* = 1)	**52**	**66752**



Gambar 1 – Comparison chart of application A and B memory usage growth with n input.

From the Table 2 and chart in Figure 16 can be seen that Application A has less memory usage than Application B even with increasing number of input. Then, App A is better and more efficient than App B. So it is sufficient just to show that implementation of Byte Division algorithm have better run-time and storage/memory usage than without implementing the Byte Division algorithm to the application. In this section the

2. Run-time

Besides the Byte Division algorithm that only implemented to Application A, other algorithms implemented to Application A and B is the same algorithm. Byte Division algorithm makes number of input and step processed by Application A and B different on each following procedures: Buffering, Projector Generator, Encode, Decode, and Hadamard Matrix Construction. With the number of input and step increased, so is the run-time. Because of conditions above the total run-time of each process become:

$T_A(n) = T_1(n) + T_2(n) + \ldots + T_k(n) + T_{Comb}(n)$

$T_B(n) = T_1(n) + T_2(n) + \ldots + T_k(n) + T_{Comb}(n)$

$T_A(n)$, $T_B(n)$:Application A and B total run-time with n input
$T_k(n)$: run-time of algorithm k with n input
$T_{Comb}(n)$: run time of Byte Division and Byte Set algorithm
n : number of input

It is important to show that Byte Division algorithm have no significant influence to total run-time of application.

From the analysis based on the pseudocode, the run-time for each algorithm T_1 (Byte Division) and T_2 (Byte Set) , with m as the number of input for algorithm implementation are:

$T_1(n) = 5m(n)$, $T_2(n) = 3{\cdot}m(n)$;

$T_{Comb} = T_1(n) + T_2(n)$

The value of m according to implementation on Application A, m_1,m_2 = {4,4} and Application B, m_1,m_2 = {0,64}

$$T_{Comb1} = 5m_1n + 3m_2n = 5 \cdot 4 \cdot n + 3 \cdot 4 \cdot n$$
$$= 20n + 12n = 32n$$
$$T_{Comb2} = 5m_1n + 3m_2n = 5 \cdot 0 \cdot n + 3 \cdot 64 \cdot n$$
$$= 0 + 64n = 64n$$

Then, $T_{Comb1} < T_{Comb2}$ and comparison 1 : 2 for every n, with n = 1, 2, ...

Tabel 3 – Comparison of application A and B number of step and input.

Algorithm implementation	Comparison (A : B)
Projector	8:512 = 1:64
Random Parity Encode	32:512 = 1:16
Buffering codeword out	4:64 = 1:16
Buffering codeword in	4:64 = 1:16
Hadamard Construction	1:8
Random Parity Decode	4:64 = 1:16

According to Table 3, Application A have less comparison value than Application B, then $T_A - T_{comb1} < T_B - T_{comb2}$. Under the condition $T_{Comb1} < T_{Comb2}$, the total run-time condition become $T_A < T_B$. Because of the linearity of Total run-time equation, the condition works for every n input.

T_A is 2 to 64 times more efficient than T_B. Taking the average of comparison value, T_A is 19.714 times more efficient than T_B.

4. Conclusion

According to results and discussions, some conclusions are made. The application can be produced in form of computer program with some abilities and weaknesses. By applying the Divide and Conquer algorithm concept, the application become more efficient than before applying the concept either in term of memory usage and run-time. In term of memory usage for initial memory usage (52 and 66752), the application is 1283.692 times more efficient. And for each input n, memory usage increased by 16 and 192, the application is 12 times more efficient.

References

Agaian, S.S. 1985. *Hadamard Matrices and Their Applications*. Springer-Verlag.

Hershey, J. and Tiemann, J. 1996. “Random Parity Coding” in *International Conference on Communications*. Vol. 1, pp. 122-126.

.