**South Dakota Mines**

**Fall 2022**

**CENG 325 - CPU & GPU Organization and Architecture**

# Hamming Code - Explanation and Python Implementation

## Hamming Code Deliverable Document

**Chami Senarath**

# Contents

# 1 Program Description

Hamming code is a set of error-correction codes that detect and correct errors occurring during transmission or stored from the sender to the receiver. This is a technique developed by R. W. Hamming for error correction. It was mostly used in the telecommunication world. Hamming worked at Bell Labs in the 1940s on the Bell Model V computer, an electromechanical relay-based machine with cycle times in seconds. The input was fed in on punched cards, which would invariably have read errors. During weekdays, special codes would find errors and flashlights so the operators could correct the problem. During after-hour periods and on weekends, when there were no operators, the machine simply moved on to the next job. Hamming worked on weekends and grew increasingly frustrated with having to restart his programs from scratch due to the unreliability of the card reader. Over the next few years, he worked on the problem of error-correction, developing an increasingly powerful array of algorithms. In 1950, he published what is now known as Hamming Code, which remains in use today in applications such as ECC memory. Hamming codes are perfect codes, that is, they have achieved the highest possible rate for codes with their block length and minimum distance 3.

In this program, we are implementing the algorithm developed by R. W. Hamming. We will be looking at creating generator matrices, parity matrices, and decoding matrices for different numbers of data bits to simulate the Hamming Code process.

# 2 Libraries & Algorithms

## 2.1 Libraries

1. **NumPy**
   NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays. In this program, a variety of routines are being utilized to accommodate various needs within algorithms. They are:

   - `np.matmul(matrix1, matrix2)`
     This function is used to multiply to matrices together and return a resultant matrix.
   - `np.zeros(rows, columns)`
     This function is used to create a zero matrix when the number of rows and columns are provided. I also used `.astype(int)` to have the resultant matrix to be only integers.
   - `np.delete(array, object, axis)`
     This function is used to delete a row or a column in a matrix.
   - `np.identity(number)`
     This function is used to created an identity matrix (which is always going to be square matrix).
   - `np.transpose(matrix)`
     This function is used to reverse or permute the axes of an array. It returns the modified array.
   - `np.ndarray.flatten()`
     This function is used to return a copy of the array collapsed into one dimension.
   - `np.random.randint(low, high, (size))`
     This function is used to return random integers from low (inclusive) to high (exclusive) for a matrix.

2. **Random**
   The random library implements pseudo-random number generators for various distributions. For integers, there is a uniform selection from a range.

   - `random.randint(low, high)`
     This function returns a random integer, both low and high inclusive.

### 2.2 Algorithms

1. **Algorithm for H Matrix**
    i. The user enters how many data bits they want.
    ii. Calculate the number of parity bits and the number of total bits, which is parity bits plus data bits.
    iii. Create a zero matrix for H.
    iv. Rows are indexed by parity bits.
    v. Columns are indexed by total bits.
    vi. If the column index + 1 is bitwise and with two to the power of row index and it is still equal to the two to the power of row index, set that spot in the H matrix to 1.

2. **Algorithm for G Matrix**
    i. Create a copy of the H matrix. (Let's call it the dummy matrix)
    ii. If the column index of the dummy matrix is a power of two, delete it.
    iii. Create a zero matrix for G.
    iv. Rows are indexed by total bits.
    v. Columns are indexed by data bits.
    vi. Create an identity matrix with the size of data bits.
    vii. If the row index + 1 in G is a power of two and the dummy matrix's row index is less than the parity bits, the row of the G matrix is the same as the row in the dummy array.
    viii. If not, the row if the G matrix is the same as the row in the identity matrix.

3. **Algorithm for R Matrix**
    i. Create a zero matrix for R.
    ii. Rows are indexed by total bits.
    iii. Columns are indexed by data bits.
    iv. Create an identity matrix with the size of data bits.
    v. If the row index + 1 is not a power of two, the R matrix's row is the same as the row in the identity matrix.

## 3 Program Structure & Functions

Functions used in this program and a brief description of each are given below.

1. `def hamming_d()`
   Provides the minimal viable product for Hamming(7, 4) - tier D. G, H, R, and p are hard-coded. Creates the message vector and performs the first parity check. No error vectors and error corrections are done here.

2. `def hamming_c()`
   Provides the minimal passing grade for Hamming(7, 4) - tier C. G, H, and R are hard-coded. Randomly creates the p vector. Randomly creates an error bit in the error vector. Performs an error correction if needed, then decodes the message.

3. `def hamming_b()`
   Tier B. Same as tier C but there is an option to select Hamming(15, 11).

4. `def hamming_a()`
   Tier A. User inputs number of data bits. Generates G, H, and R accordingly. Creates the message, encodes it, adds an error, decodes it, and prints the results.

5. `num_of_parity(data_bits) -> int`
   It counts the number of parity bits needed for a given number of data bits.

6. `is_pow_of_two(num) -> bool`
   If the number is a power of two, returns true.

# 4 Testing & Verification Process

This was my first time working on a major Python project. Hence, working through each tier aided with testing and verifying several parts of the program. On top of that, it helped me to become comfortable with Python.

1. Tier D: Familiarizing with the NumPy library and Python code structure.

```
D Tier
Message (p):  [1 0 1 1]
Send vector (x):  [0 1 1 0 0 1 1]
Received Message (r):  [0 1 1 0 0 1 1]
Parity Check (z):  [0 0 0]
>>> main
```

Figure 1: D Tier Results

2. Tier C: Introducing error and decoding the message.

```
C Tier
Message:  [0 0 1 0]
Send vector:  [0 1 0 1 0 1 0]
Received Message:  [0 1 0 0 0 1 0]
Parity Check:  [0 0 1]
Corrected Message:  [0 1 0 1 0 1 0]
Decoded Message:  [0 0 1 0]
>>> main
```

Figure 2: C Tier Results

3. Tier B: Creating Hamming codes for Hamming(15, 11)

```
B Tier
Enter Mode:
H1511
Message:  [0 1 1 0 1 1 1 0 1 0 0]
Send vector:  [0 1 0 1 1 1 0 0 1 1 1 0 1 0 0]
Received Message:  [0 1 0 1 1 1 0 0 1 1 1 0 1 0 0]
Parity Check:  [0 0 0 0]
Corrected Message:  [0 1 0 1 1 1 0 0 1 1 1 0 1 0 0]
Decoded Message:  [0 1 1 0 1 1 1 0 1 0 0]
>> main
```

Figure 3: B Tier Results

4. Tier A: Using different algorithms to verify matrix generation $HG^T = 0$.

```
Tier A
Enter number of databits:
8
Message:  [0 1 0 0 0 1 1 1]
Send vector:  [0 0 0 0 1 0 0 1 0 1 1 1]
Received Message:  [0 0 0 1 1 0 0 1 0 1 1 1]
Parity Check:  [0 0 1 0]
Corrected Message:  [0 0 0 0 1 0 0 1 0 1 1 1]
Decoded Message:  [0 1 0 0 0 1 1 1]
>> main
```

Figure 4: A Tier Results

# 5 Build Process

Extract `Hamming.py` from my zip file, then run `Py Hamming.py` in the command line. In case you do not have numpy install, run `pip install numpy`.

# 6 Level to be Graded

The level to be graded is A.