IoT FINAL PROJECT

REPORT

GUO Xiaofan

YIN Chenghao

Contents

1. Introduction

In this final IoT project, the objective is to gain proficiency in utilizing the multi-core capabilities of low-power development boards GAP8, to understand the basic structure of neural networks, and to learn how to analyze performance.

The project involves mastering control of the GAP8's multi-core functionality by running different functions on distinct cores, implementing a simple neural network, and deploying this network on the GAP8. The project will use simulated data for training and predictions and will conduct a detailed performance analysis of the GAP8, focusing on aspects such as cycles, frequency, and time, as outlined in the attached document.

1.1 GAP8 Introduction

The GAP8 microprocessor, crafted by GreenWaves for edge computing and IoT, features a nona-core 32-bit RISC-V architecture. This innovative processor is structured around three key components: autonomous peripherals, an ultra-low power microcontroller, and the compute engine[1].

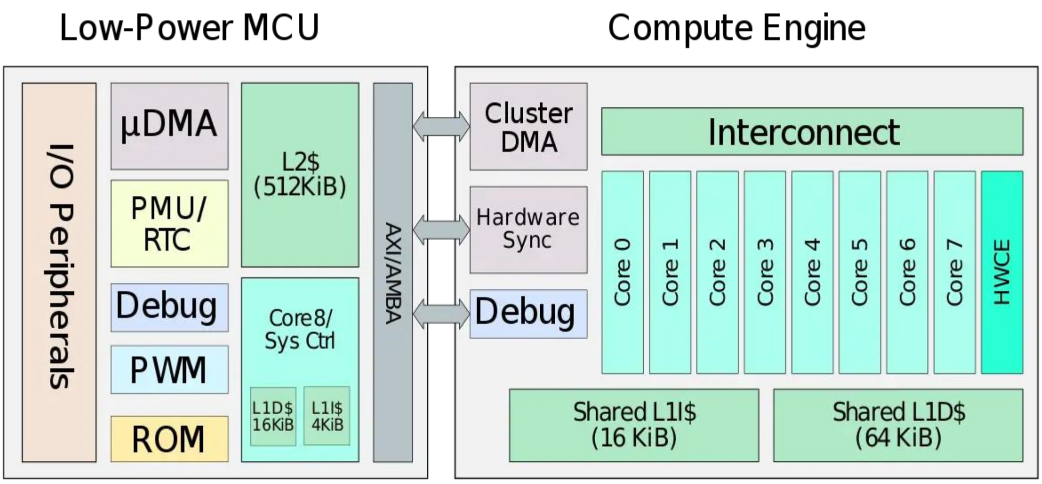


Figure 1 Gap8 Block Diagram[2]

1.1.1 Autonomous Peripherals

Autonomous Peripherals arehardware components that can operate independently of the main CPU. They manage data transfers between peripherals and memory without the need for CPU intervention, enabling efficient processing and low-power operation.

1.1.2 Ultra-low Power Microcontroller

The microcontroller block is a standard MCU (Microcontroller Unit) with many of the standard features. The MCU is situated on its own power domain with the peripherals power switchable and configurable along with the clock generator.

The chip features nine cores (1 serving in the MCU + 8 in the compute engine), with support for the integer multiplication and division instructions (M) and compressed instructions (C) standard extensions.

The GAP8 a private L1 cache for the MCU core which consists of a 16 KiB of data cache and 4 KiB of instruction cache. The compute engine has a shared level 1 cache of its own which consists of a 16 KiB instruction cache and a 64 KiB data cache. Additionally, the entire chip shares a 512 KiB level 2 cache consisting of 4 128 KiB cache banks.

1.1.3 Compute Engine

The compute engine consists of eight additional cores clustered together to form a low power but powerful computational engine. The engine sits on an entirely separate voltage and frequency domains which can be switched off when not operating or downclocked to suit a particular workload more efficiently.

1.1.4 Features of GAP8

Gap8 is focused on efficient energy management, making it suitable for battery-powered or remotely deployed devices where energy is limited. Moreover, GAP8's autonomous operation allows for reduced operational costs, as it can process and analyze data independently without constant communication with a central server.

1.2 The Neural Network

Neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer, rely on training data to learn and improve their accuracy over time[3].

- Feedforward network with a single layer of neurons:

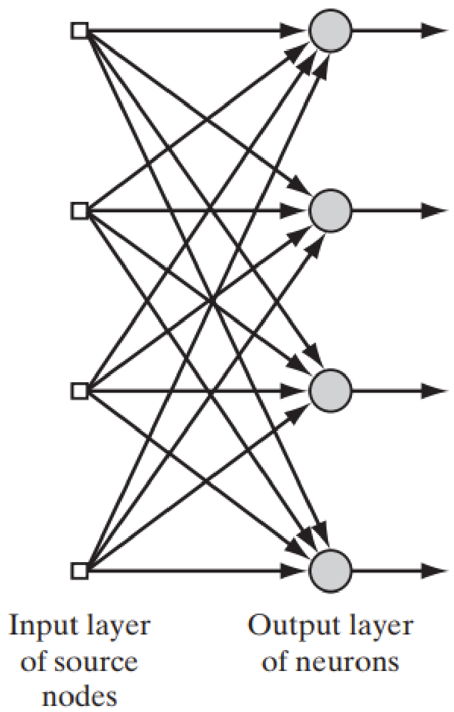


Figure 2 Feedforward network with a single layer of neurons[4].

- Fully connected feedforward network with one hidden layer and one output layer:

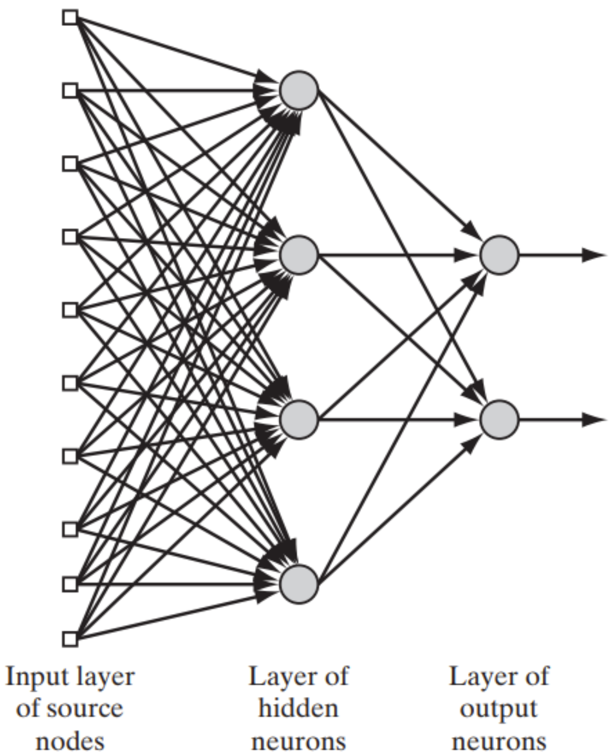


Figure 3 Fully connected feedforward network[5].

1.2.1 Input Layer

The input layer is the first layer of the neural network and is responsible for receiving input data.

- **Function:** Directly process raw input data.
- **Data transformation:** The input layer usually does not make any changes to the data and passes it directly to the next layer.

1.2.2 Hidden Layer

Hidden layers are located between the input layer and the output layer. They are where the neural network performs calculations and feature extraction. There can be one or more hidden layers.

- **Function:** Processes data and extracts features.
- **Operation:** Each hidden layer contains multiple neurons, and each neuron is connected to all nodes from the previous layer with weights. Each neuron receives inputs from the nodes of the previous layer, applies weights, and produces an output through an activation function. This output becomes the input for the next layer. This process repeats, allowing the network to learn representations of the input data by adjusting the weights.
- **Weights:** Each connection has a weight, which represents the strength of the connection. During training, these weights are optimized according to the algorithm so that the network can perform tasks accurately.
- **Activation Function:** Introduces non-linearity to the network, enabling it to learn more complex patterns. Common activation functions include sigmoid, tanh, and ReLU.

1.2.3 Output Layer

The output layer is the last layer of the neural network and is responsible for outputting calculation results.

- **Function:** Provides the final output of the network, representing the solution to the problem.
- **Operation:** Each output node corresponds to a possible output category or value.

2.Realization

2.1 Code Link

- Realize a simple neural network.
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP1.c
- Implement the simple neural network on GAP8.
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP2.c
- Implement the neural network on GAP8 with Performance counters.
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP2-monitor.c
- flexSensorData Test.
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP3.c
- flexSensorData Test with Hidden_Layer.
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP3-2.c
- Multi-core asynchronous control
  https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP4.c

2.2 Interpretation of Results

2.2.1 Realize a Simple Neural Network

A simple neural network example implemented in C language was run and the following results were obtained:

```
        *** Neural Network Test ***
Input: [0.100000, 0.120000], Target: 0.000000, Prediction: 0.011752
Input: [1.100000, 0.900000], Target: 1.000000, Prediction: 0.968554
Input: [2.100000, 1.980000], Target: 2.000000, Prediction: 2.028473
Input: [2.890000, 3.200000], Target: 3.000000, Prediction: 2.991888
```

Figure 4 The Output of a Simple NN (1).

```
        *** Neural Network Test ***
Input: [0.500000, 0.400000], Target: 0.000000, Prediction: 0.093871
Input: [1.200000, 1.000000], Target: 1.000000, Prediction: 0.804686
Input: [2.500000, 2.200000], Target: 2.000000, Prediction: 2.214163
Input: [3.000000, 2.800000], Target: 3.000000, Prediction: 2.900671
```

Figure 5 The Output of a Simple NN (2).

Looking at the overall output results, the predicted value of the neural network is quite close to the target value, which indicates that the training process is successful, and the network is able to learn the relationship between the input data and the output data.

2.2.2 Implement the Neural Network on GAP8

1) Implement the Simple Neural Network on GAP8

Running a simple neural network on GAP8[6] gives the following results:

```
        *** Neural Network Parameters ***

>>>Define:
Input Size: 2
Output Size: 1
Learning Rate: 0.010000
Epochs: 10000

>>>Neural network parameters:
Input: [0.100000, 0.120000], Prediction: 0.012210, Weight:0.685931
Input: [1.100000, 0.900000], Prediction: 0.968171, Weight:0.346192
Input: [2.100000, 1.980000], Prediction: 2.027989, Weight:0.000000
Input: [2.890000, 3.200000], Prediction: 2.992229, Weight:0.000000
bias:-0.097926

         *** Neural network based on GAP8 ***

>>>Entering main controller
[32 0] Neural network based on GAP8
Perf : 20566
cycles Timer : 28223 cycles

>>>Cluster master core entry
[0 0] Neural network based on GAP8
[0 2] Neural network based on GAP8
[0 6] Neural network based on GAP8
[0 4] Neural network based on GAP8
[0 5] Neural network based on GAP8
[0 3] Neural network based on GAP8
[0 1] Neural network based on GAP8
[0 7] Neural network based on GAP8

>>>Cluster master core exit
Test success !
```

Figure 6 The Output of a Simple Neural Network on GAP8.

Analysis of these results shows that the neural network can produce predictions close to the target value based on the training data, indicating that the learning process is effective.

However, some values of 0.0 for weights may mean that some inputs have very little impact on the output or have been ignored by the network, which can be due to some weights contributing less to error reduction during the optimization process of the network. The final deviation value is negative, which has a small adjustment effect on the final forecast value.

Additionally, "Performance Cycles" and "Timer Cycles" provide information about the efficiency of program execution. A lower number of cycles usually means higher performance.

Finally, successful tests show that the implemented training strategy can effectively implement the forward propagation and back propagation algorithms of neural networks on GAP8 hardware.

2) Implement the NN on GAP8 with Performance Counters

In order to realize the function of performance counter, based on the code of “Implement the simple neural network on GAP8.”, added the following three functions and run these in main function.

```
//uint32_t start_time;
clock_t start_time, end_time;
double total_time_in_seconds;

void start_performance_monitoring() {
    pi_perf_conf(
        1 << PI_PERF_CYCLES |
        1 << PI_PERF_ACTIVE_CYCLES |
        1 << PI_PERF_INSTR |
        1 << PI_PERF_LD_STALL |
        1 << PI_PERF_JR_STALL |
        1 << PI_PERF_IMISS |
        1 << PI_PERF_LD |
        1 << PI_PERF_ST |
        1 << PI_PERF_JUMP |
        1 << PI_PERF_BRANCH |
        1 << PI_PERF_BTAKEN |
        1 << PI_PERF_RVC |
        1 << PI_PERF_LD_EXT |
        1 << PI_PERF_ST_EXT |
        1 << PI_PERF_LD_EXT_CYC |
        1 << PI_PERF_ST_EXT_CYC |
        1 << PI_PERF_TCDM_CONT
    );

    pi_perf_reset();
    pi_perf_start();
    start_time = clock();
}

void stop_performance_monitoring() {
    pi_perf_stop();
    end_time = clock();
    total_time_in_seconds = (double)(end_time - start_time) / CLOCKS_PER_SEC;
}

//void print_performance_data(uint32_t start_time, uint32_t end_time, uint32_t total_time) {
void print_performance_data() {
    printf("Total cycles: %d\n", pi_perf_read(PI_PERF_CYCLES));
    printf("Active cycles: %d\n", pi_perf_read(PI_PERF_ACTIVE_CYCLES));
    printf("Instructions executed: %d\n", pi_perf_read(PI_PERF_INSTR));
    printf("Load data stalls: %d\n", pi_perf_read(PI_PERF_LD_STALL));
    printf("Jump register stalls: %d\n", pi_perf_read(PI_PERF_JR_STALL));
    printf("Instruction misses: %d\n", pi_perf_read(PI_PERF_IMISS));
    printf("Data memory loads: %d\n", pi_perf_read(PI_PERF_LD));
    printf("Data memory stores: %d\n", pi_perf_read(PI_PERF_ST));
    printf("Unconditional jumps: %d\n", pi_perf_read(PI_PERF_JUMP));
    printf("Branches: %d\n", pi_perf_read(PI_PERF_BRANCH));
    printf("Taken branches: %d\n", pi_perf_read(PI_PERF_BTAKEN));
    printf("Compressed instructions: %d\n", pi_perf_read(PI_PERF_RVC));
    printf("External memory loads: %d\n", pi_perf_read(PI_PERF_LD_EXT));
    printf("External memory stores: %d\n", pi_perf_read(PI_PERF_ST_EXT));
    printf("External load cycles: %d\n", pi_perf_read(PI_PERF_LD_EXT_CYC));
    printf("External store cycles: %d\n", pi_perf_read(PI_PERF_ST_EXT_CYC));
    printf("TCDM contention cycles: %d\n", pi_perf_read(PI_PERF_TCDM_CONT));
    printf("Execution time: %f seconds\n", total_time_in_seconds);
}
```

Figure 7 Three Functions of Performance Counter.

```c
int main(void) {

    initialize();

    start_performance_monitoring();

    double training_data[][INPUT_SIZE] = {{0.1, 0.12}, {1.1, 0.9}, {2.1,1.98}, {2.89, 3.2}};

     double targets[] = {0, 1, 2, 3};

     for (int epoch = 0; epoch < EPOCHS; epoch++) {

        for (int i = 0; i < sizeof(training_data) / sizeof(training_data[0]); i++) {

            train(training_data[i], targets[i]);

        }

    }

    //void stop_performance_monitoring();
    //uint32_t end_time, total_time;
    stop_performance_monitoring();

    printf("\n\t*** Neural Network Parameters ***\n");

    printf("\n>>>Define:\n");
    printf("Input Size: %d\n", INPUT_SIZE);
    printf("Output Size: %d\n", OUTPUT_SIZE);
    printf("Learning Rate: %lf\n", LEARNING_RATE);
    printf("Epochs: %d\n", EPOCHS);

     printf("\n>>>Neural network parameters:\n");

    //for (int i = 0; i < sizeof(training_data) / sizeof(training_data[0]); i++) {
    for (size_t i = 0; i < sizeof(training_data) / sizeof(training_data[0]); i++) {

        double prediction = predict(training_data[i]);

        printf("Input: [%lf, %lf], Prediction: %lf, Weight:%lf\n", training_data[i][0],
training_data[i][1], prediction, weights[i]);

    }

    printf("bias:%lf\n",bias);


    printf("\n\t *** Neural network based on GAP8 ***\n");

    print_performance_data();

    return pmsis_kickoff((void *)helloworld);
}
```

Figure 8 Called in Main Function.

However, during the running process, encountered the problem that the time function cannot be called. To capture the data of time tried using the following two functions:

1) Using function of PMSIS “pi_time_us()”

```
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/basic/helloworld/BUILD
/GAP8_V3/GCC_RISCV_FREERTOS/FP2-monitor.o: In function `start_performance_monito
ring':
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/basic/helloworld/FP2-m
onitor.c:139: undefined reference to `pi_time_us'
collect2: error: ld returned 1 exit status
make: *** [/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/rtos/freeRTOS/vendors/gwt
/rules/freeRTOS_rules.mk:419: /home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/exampl
es/gap8/basic/helloworld/BUILD/GAP8_V3/GCC_RISCV_FREERTOS/test] Error 1
```

Figure 9 Unable to Call the Function pi_time_us().

2) Using function of C language “clock ()”

```
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/basic/helloworld/BUILD
/GAP8_V3/GCC_RISCV_FREERTOS/FP2-monitor.o: In function `start_performance_monito
ring':
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/rtos/freeRTOS/vendors/gwt/gap8/pmsis
/include/cores/TARGET_RISCV_32/core_utils.h:15: undefined reference to `clock'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/rtos/freeRTOS/vendors/gwt/gap8/pmsis
/include/cores/TARGET_RISCV_32/core_utils.h:15: undefined reference to `clock'
collect2: error: ld returned 1 exit status
make: *** [/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/rtos/freeRTOS/vendors/gwt
/rules/freeRTOS_rules.mk:419: /home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/exampl
es/gap8/basic/helloworld/BUILD/GAP8_V3/GCC_RISCV_FREERTOS/test] Error 1
```

Figure 10 Unable to Call the Function clock().

The code compiles successfully but the result is not outputted due to the target function not being invoked.

2.2.3 Use flex sensor data as training and test data

1) flexSensorData Test.

The following procedures were implemented for training the neural network on GAP8 using the data in the flexSensorData file:

a) The flexSensorData.csv was opened with fopen().
b) Each line of the file was read using fscanf().
c) The parsed data were stored in the training_data and targets arrays.
d) The file was closed with fclose() upon completion of the data import.

```
    /*

    double training_data[100][INPUT_SIZE];
    double targets[100];
    int rows = 0;

#ifdef ENABLE_FILE_IO

    FILE *file;
    file = fopen("flexSensorData.csv", "r");
    if (!file) {
        printf("CAN NOT OPEN！\n");
        return 1;

    }

    while (fscanf(file, "%lf,%lf,%lf\n", &training_data[rows][0], &training_data[rows][1],
&targets[rows]) != EOF) {
        rows++;
    }

    fclose(file);

    */
```

Figure 11 Code to Import .csv Data Using Function.

However, a problem occurred during the reading process, the function could not be called accurately, terminal displayed as follows:

```
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `fopen'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `perror'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `fgets'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `strtok'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `atof'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `atof'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `strtok'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `fgets'
/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/bas
ic/helloworld/a.c:18: undefined reference to `fclose'
collect2: error: ld returned 1 exit status
make: *** [/home/hx/gap_riscv_toolchain_ubuntu/gap_sdk/rtos/f
reeRTOS/vendors/gwt/rules/freeRTOS_rules.mk:419: /home/hx/gap
_riscv_toolchain_ubuntu/gap_sdk/examples/gap8/basic/helloworl
d/BUILD/GAP8_V3/GCC_RISCV_FREERTOS/test] Error 1
```

Figure 12 Cannot be Called from Library Function.

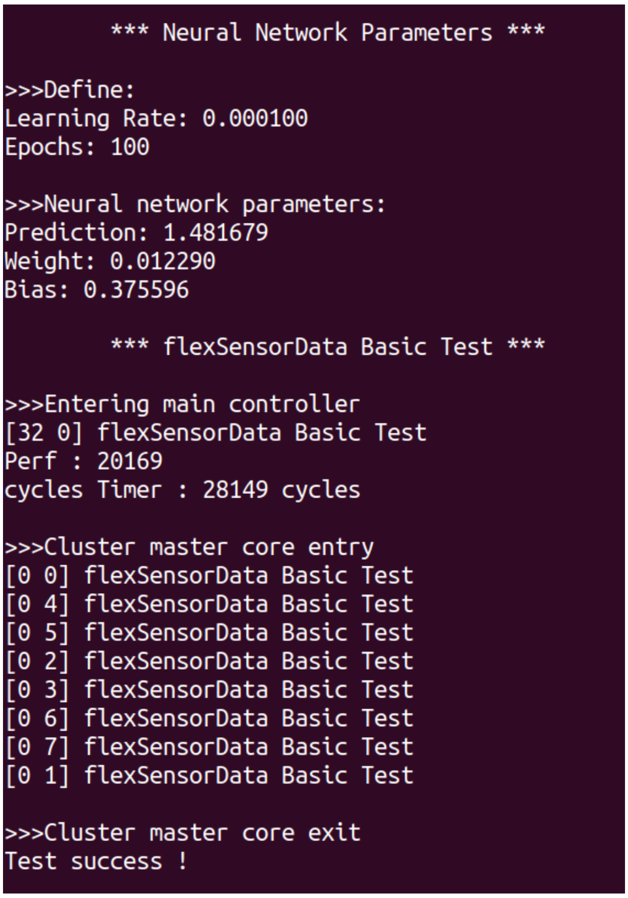
Therefore, modifications are made to directly introduce the data into the neural network calculation, the following results were obtained:

```
        *** Neural Network Parameters ***

>>>Define:
Learning Rate: 0.000100
Epochs: 10000

>>>Neural network parameters:
Prediction: 1.412866
Weight: -0.006292
Bias: 1.979180

        *** flexSensorData Basic Test ***

>>>Entering main controller
[32 0] flexSensorData Basic Test
Perf : 20095
cycles Timer : 27852 cycles

>>>Cluster master core entry
[0 0] flexSensorData Basic Test
[0 6] flexSensorData Basic Test
[0 2] flexSensorData Basic Test
[0 5] flexSensorData Basic Test
[0 7] flexSensorData Basic Test
[0 1] flexSensorData Basic Test
[0 4] flexSensorData Basic Test
[0 3] flexSensorData Basic Test

>>>Cluster master core exit
Test success !
```

Figure 13 NN Data from flexSensorData.csv (Epochs: 10000).

Considering that the weights are less than zero, the bias is greater than one, and the number of training epochs is set to a relatively high value of 10,000, there may be overfitting in the program.

Therefore, the epochs value is reduced to 100, and the results are as follows:



```
        *** Neural Network Parameters ***

>>>Define:
Learning Rate: 0.000100
Epochs: 100

>>>Neural network parameters:
Prediction: 1.481679
Weight: 0.012290
Bias: 0.375596

        *** flexSensorData Basic Test ***

>>>Entering main controller
[32 0] flexSensorData Basic Test
Perf : 20169
cycles Timer : 28149 cycles

>>>Cluster master core entry
[0 0] flexSensorData Basic Test
[0 4] flexSensorData Basic Test
[0 5] flexSensorData Basic Test
[0 2] flexSensorData Basic Test
[0 3] flexSensorData Basic Test
[0 6] flexSensorData Basic Test
[0 7] flexSensorData Basic Test
[0 1] flexSensorData Basic Test

>>>Cluster master core exit
Test success !
```

Figure 14 NN Data From flexSensorData.csv (Epochs:100).

The specific values of the weights and biases of the neural network after a certain number of trainings provide some information about the state of the network after training. Performance indicators indicate that the program can run on the cluster and the execution time is reasonable.

2) flexSensorData Test with Hidden Layer

Considering that neural networks with hidden layers have stronger learning capabilities and can theoretically approximate any continuous function, we made the following attempt.

Based on the data in the flexSensorData.csv file, a neural network with hidden layers was trained on GAP8 and the following results are obtained:

```
        *** Neural Network Parameters ***

>>>Define:
Learning Rate: 0.010000
Epochs: 10000
HIDDEN_LAYER_SIZE: 10

>>>Neural network parameters:
Weight: 1.707808
Prediction: 1.686064
Bias: 1.686064

        ***  flexSensorData HIDDEN_LAYER Test  ***

>>>Entering main controller
[32 0] flexSensorData HIDDEN_LAYER Test
Perf : 20399
cycles Timer : 28733 cycles

>>>Cluster master core entry
[0 0] flexSensorData HIDDEN_LAYER Test
[0 4] flexSensorData HIDDEN_LAYER Test
[0 5] flexSensorData HIDDEN_LAYER Test
[0 7] flexSensorData HIDDEN_LAYER Test
[0 6] flexSensorData HIDDEN_LAYER Test
[0 1] flexSensorData HIDDEN_LAYER Test
[0 3] flexSensorData HIDDEN_LAYER Test
[0 2] flexSensorData HIDDEN_LAYER Test

>>>Cluster master core exit
Test success !
```

Figure 15 NN with Hidde_Layer.

2.2.4 Multi-core asynchronous control

```c
void function_core_0(void *arg) {

    initialize();

    double training_data[][INPUT_SIZE] = {{0.1, 0.12}, {1.1, 0.9}, {2.1, 1.98}, {2.89, 3.2}};
    double targets[] = {0, 1, 2, 3};

    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        for (int i = 0; i < sizeof(training_data) / sizeof(training_data[0]); i++) {
            train(training_data[i], targets[i]);
        }
    }
        printf("\n\t***[Core 0] Training the simple NN on the GAP8***");
    for (int i = 0; i < sizeof(training_data) / sizeof(training_data[0]); i++) {
        double prediction = predict(training_data[i]);
        printf("\n[Core 0]Input: [%.2f, %.2f], Target: %.2f, Prediction: %lf\n",
               training_data[i][0], training_data[i][1], targets[i], prediction);
    }
}
```

Figure 16 Simple NN Run in the 1st Core.

```c
void function_core_1(void *arg) {

    //initialize();

    const int NUM_SAMPLES = 4;
    double random_data[NUM_SAMPLES][INPUT_SIZE];

    for (int i = 0; i < NUM_SAMPLES; i++) {
        for (int j = 0; j < INPUT_SIZE; j++) {
            random_data[i][j] = ((double)simple_rand() / RAND_MAX) * 2 - 1;
        }
    }
    printf("\n\t***[Core 1] Generating random prediction dataset***");
    for (int i = 0; i < NUM_SAMPLES; i++) {
        double prediction = predict(random_data[i]);
        printf("\n[Core 1]Random Input: [%lf, %lf], Prediction: %lf\n",
                random_data[i][0], random_data[i][1], prediction);
    }

}

```

Figure 17 Stochastic NN Run in the 2nd Core.

```c
void function_core_2(void *arg) {

    initialize();

    double weights_input_hidden[INPUT_SIZE][HIDDEN_SIZE] = {{0.15, 0.25}, {0.20, 0.30}};
    double weights_hidden_output[HIDDEN_SIZE][OUTPUT_SIZE] = {{0.40}, {0.50}};
    double input[INPUT_SIZE] = {1.0, 2.0};
    double hidden[HIDDEN_SIZE] = {0};
    double output[OUTPUT_SIZE] = {0};

    for (int i = 0; i < HIDDEN_SIZE; i++) {
        for (int j = 0; j < INPUT_SIZE; j++) {
            hidden[i] += input[j] * weights_input_hidden[j][i];
        }
        hidden[i] = activation(hidden[i]);
    }

    for (int i = 0; i < OUTPUT_SIZE; i++) {
        for (int j = 0; j < HIDDEN_SIZE; j++) {
            output[i] += hidden[j] * weights_hidden_output[j][i];
        }
        output[i] = activation(output[i]);
    }
    printf("\n\t***[Core 2] NN forward propagation function with Hidden_Layer***");
    printf("\n[Core 2]Input:(%.2f,%.2f), hidden:%f, Output: %f\n", input[0],input[1],
hidden[0], output[0]);

}
```

Figure 18 NN Forward Propagation with Hidden Layer Run in the 3rd Core.

```
void function_core_3(void *arg) {

    initialize();

    double weights[2] = {0.5, -0.5};
    double input[2] = {0.5, 0.6};
    double expected_output = 0.7;
    double output;

    output = activation(input[0] * weights[0] + input[1] * weights[1]);

    double error = expected_output - output;

    for (int i = 0; i < 2; i++) {
        double gradient = error * activation_derivative(output) * input[i];
        weights[i] += LEARNING_RATE * gradient;
    }

    printf("\n\t***[Core 3] NN back propagation function with Hidden_Layer***");
    printf("\n[Core 3]input:(%.2f,%.2f),Updated weights: %f, %f\n",input[0],input[1],
weights[0], weights[1]);
}
```

Figure 19 NN Back Propagation with Hidden Layer Run in the 4th Core.

```
void function_core_4(void *arg) {

    printf("\n\t***[Core 4] Linear Regression Model***");

    double weights = 0.0;
    double bias = 0.0;
    double inputs[] = {1.0, 2.0, 3.0, 4.0};
    double targets[] = {2.0, 4.0, 6.0, 8.0};
    int n_samples = sizeof(inputs) / sizeof(inputs[0]);

    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        for (int i = 0; i < n_samples; i++) {
            double output = inputs[i] * weights + bias;
            double error = targets[i] - output;
            weights += LEARNING_RATE * error * inputs[i];
            bias += LEARNING_RATE * error;
        }
    }
    printf("\n[Core 4]Trained weights: %f, bias: %f\n", weights, bias);
}
```

Figure 20 Linear Regression Run in the 5th Core.

```
void function_core_5(void *arg) {

   printf("\n\t***[Core 5] Logistic Regression Model***");

    double weights = 0.0;
    double bias = 0.0;
    double inputs[] = {0, 1, 2, 3};
    double targets[] = {0, 0, 1, 1};
    int n_samples = sizeof(inputs) / sizeof(inputs[0]);

    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        for (int i = 0; i < n_samples; i++) {
            double output = activation(inputs[i] * weights + bias);
            double error = targets[i] - output;
            weights += LEARNING_RATE * error * output * (1 - output) * inputs[i];
            bias += LEARNING_RATE * error * output * (1 - output);
        }
    }
    printf("\n[Core 5]Trained weights: %f, bias: %f\n", weights, bias);
}
```

Figure 21 Logistic Regression Run in the 6th Core.

```c
void function_core_6(void *arg) {

    printf("\n\t***[Core 6] Multi-Layer Perceptron Model***");

    double weights1 = 0.15, weights2 = 0.25;
    double bias1 = 0.35, bias2 = 0.45;

    double inputs[] = {0.5, 0.6};
    double target = 0.7;
    double hidden_output, final_output;

    for (int epoch = 0; epoch < EPOCHS; epoch++) {

        hidden_output = activation(inputs[0] * weights1 + inputs[1] * weights2 + bias1);
        final_output = activation(hidden_output * weights2 + bias2);

        double error = target - final_output;
        weights2 += LEARNING_RATE * error * final_output * (1 - final_output) * hidden_output;
        weights1 += LEARNING_RATE * error * final_output * (1 - final_output) * inputs[0];
        bias2 += LEARNING_RATE * error * final_output * (1 - final_output);
        bias1 += LEARNING_RATE * error * final_output * (1 - final_output);
    }

    printf("\n[Core 6]Trained weights1: %f, weights2: %f, bias1: %f, bias2: %f\n", weights1, weights2, bias1, bias2);
}

```

Figure 22 Multi-Layer Perceptron Run in the 7th Core.

```c
void function_core_7(void *arg) {

    printf("\n\t***[Core 7] Simple Classification Model***");

    double weights[] = {0.0, 0.0};
    double bias = 0.0;
    double inputs[][2] = {{0, 0}, {0, 1}, {1, 0}, {1, 1}};
    double targets[] = {0, 1, 1, 0}; // XOR-like problem
    int n_samples = sizeof(inputs) / sizeof(inputs[0]);

    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        for (int i = 0; i < n_samples; i++) {
            double output = activation(inputs[i][0] * weights[0] + inputs[i][1] * weights[1] + bias);
            double error = targets[i] - output;
            weights[0] += LEARNING_RATE * error * output * (1 - output) * inputs[i][0];
            weights[1] += LEARNING_RATE * error * output * (1 - output) * inputs[i][1];
            bias += LEARNING_RATE * error * output * (1 - output);
        }
    }
    printf("\n[Core 7]Trained weights: [%f, %f], bias: %f\n", weights[0], weights[1], bias);

}

```

Figure 23 Simple Classification Run in the 8th Core.

Run the code and get the following results:

```
>>>Entering main controller

>>>Entering cluster on core 0







        ***[Core 3] NN back propagation function with Hidden_Layer***
        ***[Core 2] NN forward propagation function with Hidden_Layer***
        ***[Core 1] Generating random prediction dataset***
[Core 2]Input:(1.00,2.00), hidden:0.550000, Output: 0.645000
[Core 1]Random Input: [-0.258156, -0.672048], Prediction: 0.309833
[Core 3]input:(0.50,0.60),Updated weights: 0.500000, -0.500000

[Core 1]Random Input: [-0.184790, 0.921812], Prediction: 0.000000

[Core 1]Random Input: [0.916379, 0.919858], Prediction: 0.067947

[Core 1]Random Input: [-0.071200, 0.602588], Prediction: 0.196470
        ***[Core 6] Multi-Layer Perceptron Model***
[Core 6]Trained weights1: 0.179153, weights2: 0.286263, bias1: 0.408306, bia
s2: 0.508306
        ***[Core 4] Linear Regression Model***
[Core 4]Trained weights: 2.000000, bias: 0.000000
        ***[Core 5] Logistic Regression Model***
[Core 5]Trained weights: 0.000000, bias: 0.000000
        ***[Core 7] Simple Classification Model***
[Core 7]Trained weights: [0.000000, 0.000000], bias: 0.000000

        ***[Core 0] Training the simple NN on the GAP8***
[Core 0]Input: [0.10, 0.12], Target: 0.00, Prediction: 0.012123

[Core 0]Input: [1.10, 0.90], Target: 1.00, Prediction: 0.968244

[Core 0]Input: [2.10, 1.98], Target: 2.00, Prediction: 2.028082

[Core 0]Input: [2.89, 3.20], Target: 3.00, Prediction: 2.992164

>>>Leaving cluster on core 0

>>>Leaving main controller
```

Figure 24 Results of Function Run in Different Cores

It can be seen from the results that the titles and output results of different functions which running in different Cores are mixed and are not displayed in the order.

Each “function_core_n(void *arg)” in the output results is executed on a different core, but their output are redirected to the same standard output stream: Terminal. Therefore, the output results are displayed continuously rather than separated by cores. In another world, their output are displayed consecutively in the order of execution, not in programming order.

3. Conclusions

In this final project learned about the architecture and functions of GAP8 and gained a preliminary understanding and application of neural networks.

The project mainly completed the successful deployment of a simple neural network on the GAP8 platform, the use data of flexSensorData.csv for training and testing, and the method of multi-core asynchronous control.

The goals were basically completed, but some problems were still encountered in the process, such as the linking and function calling of the C language function library and PMSIS function library on the GAP8.

4. References

[1] GAP8 – GreenWaves, https://en.wikichip.org/wiki/greenwaves/gap8

[2] File: gap8 block diagram.svg, https://en.wikichip.org/wiki/File:gap8_block_diagram.svg

[3] What are neural networks? https://www.ibm.com/topics/neural-networks

[4] Haykin, S. (2009). Neural networks and learning machines (3rd ed., p. 21). Upper Saddle River, NJ: Pearson Education, Inc.

[5] Haykin, S. (2009). Neural networks and learning machines (3rd ed., p. 22). Upper Saddle River, NJ: Pearson Education, Inc.

[6] https://greenwaves-technologies.com/manuals/BUILD/PULP-OS/html/index.html#section1