AN13001

Glow Memory Analysis

Rev. 0 — November 2020

Application Note

Contents

1 Introduction....1
2 Glow bundle....1
3 Glow project size....4
4 Conclusion....6
5 Revision history....6

1 Introduction

Glow is a machine learning compiler for neural network graphs. It is designed to optimize the neural network graphs and generate code for a targeted hardware device. This code can then be integrated into a MCUXpresso Software Development Kit (SDK) project which provides a framework that allows the integration of the generated bundle.

This document covers how to understand the Glow memory information generated by the Glow compiler and calculate the memory required for a particular model. This compiler can then be used to determine the minimum memory size that is needed to run the model.

2 Glow bundle

A Glow bundle is the output of the Glow Ahead-of-Time (AOT) compiler. There are 4 files that are generated into the directory specified by the *-emit-bundle* argument when compiling with the **model-compiler** Glow tool. This document uses the LeNet MNIST model as an example.

- **lenet_mnist.o** - the bundle object file (code).
- **lenet_mnist.h** - the bundle header file (API).
- **lenet_mnist.weights.bin** - the model weights in binary format.
- **lenet_mnist.weights.txt** - the model weights in text format as C text array.

The **weights.bin** and **weights.txt** files contain the exact same data but in two different formats and only one needs to be used in a project that uses Glow. For very large models with numerous weight data, using the .bin file in the project often results in faster IDE compilation time than using the .txt file since the data is already in binary format and does not need to be parsed.

The **lenet_mnist.o** file is the object file that is integrated into the MCUXpresso IDE project and contains the compiled model code that is executed when inferencing. Note that the size of this file on the hard drive is larger than the Flash size required on the embedded system to use this library.

The **lenet_mnist.h** file contains #defines for memory usage and will be the focus of this application note.

2.1 Glow memory usage

Glow does not use dynamically allocated memory. Therefore, the memory requirements can be determined by looking at the output in the header file that is generated by Glow.

Inside the header file, there are three key definitions starting near line 59 (using the MNIST model as an example):

```
// Memory sizes (bytes).
#define LENET_MNIST_CONSTANT_MEM_SIZE 431360
#define LENET_MNIST_MUTABLE_MEM_SIZE 3200
#define LENET_MNIST_ACTIVATIONS_MEM_SIZE 20992
```

Let us look at these definitions in more detail:

2.1.1 LENET_MNIST_CONSTANT_MEM_SIZE

Defines the size of the model weights. It exactly matches up with the size of the **weights.bin** file, and contains the number of elements in the **weight.txt** array. During inferencing, the weights can be read from either non-volatile memory or from RAM. The performance implications of having the weights read from non-volatile memory vs RAM will be covered later in this document. From a memory standpoint, the weights will always take up the specified amount of Flash, and if read from RAM, will also require that much RAM as well. In the example below, the weights are put into RAM.

In Glow project:

```
// Statically allocate memory for constant weights (model weights) and initialize.
            GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
            uint8_t constantWeight[LENET_MNIST_CONSTANT_MEM_SIZE] = {
            #include "lenet_mnist.weights.txt"
            };
```

2.1.2 LENET_MNIST_MUTABLE_MEM_SIZE

Defines the amount of memory required for both the input and output data buffers. The memory must be allocated in RAM. This value remains constant for a particular model regardless of the compiler arguments used because the input and output dimensions for a model are static. In the header file, you will also notice that there are two #defines for the address offsets of the input and output buffers. These are offset values from the start of the **LENET_MNIST_MUTABLE_MEM_SIZE** buffer.

```
#define LENET_MNIST_data 0
#define LENET_MNIST_softmax 3136
```

Note that the output data offset is exactly 3,136 because the input data is a 28x28 pixel input image that uses 4 bytes to represent the floating point value of each monochromatic pixel, thus requiring a 28*28*1*4=3,136 byte buffer to hold the input data.

In Glow project:

```
// Statically allocate memory for mutable weights (model input/output data).
            GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
            uint8_t mutableWeight[LENET_MNIST_MUTABLE_MEM_SIZE];

            // Bundle input data absolute address.
            uint8_t *inputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_data);

            // Bundle output data absolute address.
            uint8_t *outputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_softmax);
```

2.1.3 LENET_MNIST_ACTIVATIONS_MEM_SIZE

Defines the amount of scratch memory required for intermediate computations needed by the model. This buffer must be located in RAM.

In Glow project:

```
// Statically allocate memory for activations (model intermediate results).
            GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
            uint8_t activations[LENET_MNIST_ACTIVATIONS_MEM_SIZE];
```

2.1.4 Glow memory summary

All three of these buffers that were allocated are used when running the inference:

```
lenet_mnist(constantWeight, mutableWeight, activations);
```

A summary of the Glow memory usage defined by the header file can be found below:

Table 1. Glow memory summary

Type	Constant Name in <network_name>.h	Location	Description
Model Weights	CONSTANT_MEM_SIZE	RAM or Flash	Size required for Neural Network weight data
Input/Output Data	MUTABLE_MEM_SIZE	RAM	Buffer size required for input data and results output
Scratch Data	ACTIVATIONS_MEM_SIZE	RAM	Buffer size required for intermediate computation

2.2 Options that affect Glow memory requirements

The input/output buffer size is static regardless of what Glow compile arguments are used because the input and output dimensions for a model are static. However, some compile options can have significant impacts on memory required for the weights and scratch data.

2.2.1 Quantization

Quantization is transforming the model from its original 32-bit floating point weights to 8-bit fixed point weights. This means that the model will require roughly ¼ of the space for weights. Also, fixed point math is faster than floating point math so quantizing a model often, though not always, results in a faster inference time.

A model can be quantized with the Glow compiler by first generating a quantization profile with the model-profiler or image-classifier tool. Then, when running the **model-compiler** Glow tool, the ***-load-profile=profile.yml*** argument is used to quantize the model. The effect can be seen in the LENET_MNIST_CONSTANT_MEM_SIZE value, as it is reduced by a factor of 4. This means less Flash is required, and if the weights are loaded into RAM, less RAM will be required. Also, because of the smaller data, the amount of scratch RAM required will be reduced by a factor of 4 as well.

No Quantization:

```
#define LENET_MNIST_CONSTANT_MEM_SIZE          1724672
#define LENET_MNIST_MUTABLE_MEM_SIZE           3200
#define LENET_MNIST_ACTIVATIONS_MEM_SIZE       57600
```

Quantized (default settings):

```
#define LENET_MNIST_CONSTANT_MEM_SIZE              433152
#define LENET_MNIST_MUTABLE_MEM_SIZE               3200
#define LENET_MNIST_ACTIVATIONS_MEM_SIZE           15232
```

Note that it will not be exact ¼ the size as not all aspects of the model can be quantized. The MUTABLE_MEM_SIZE remains same as expected since the input and output dimensions have not been modified when doing the quantization.

2.2.2 CMSIS-NN on memory and performance

Some Glow compiler arguments like ***-use-cmsis*** can also affect the memory requirements. However, the amount of change will be less significant than the quantization parameter. The exact amount of change will depend on the particular model. Also, the *-use-cmsis* option will only have an effect if the model is also quantized during compile time with the *-load-profile* argument.

The performance gains from using the CMSIS-NN option will often make the RAM trade-off worth it, but this will be application and model specific. See Table 2 below for data from an example LeNet MNIST model:

Table 2. CMSIS-NN Glow compile options

Compile Options	Weights	Input/ Output	Activations	Compiled Library	Total Flash	Total RAM	Inference Time on RT1060
Quantized w/ Symmetric Power 2	431,360	3,200	15,232	8,380	461,704	461,434	26 ms
Quantized w/ Symmetric Power 2 with CMSIS-NN	431,360	3,200	20,992	23,204	465,560 (+3,856)	467,184 (+5,750)	10 ms

2.2.3 HiFi4 on memory and performance

The RT685 has the option of compiling with HiFi4 extensions, which can dramatically decrease the inference time. However, this requires the HiFi4 neural network library. This library requires a total of 676 KB of Flash, and is then loaded into the RT685 RAM requiring an additional 676 KB of RAM as well. The performance gains from using the HiFi4 DSP will often make the memory trade-off worth it, but this will be application and model specific. Note that the RT685 has 4.5 MB of on-chip SRAM.

The HiFi4 neural network library can be removed from the project by setting **DSP_IMAGE_COPY_TO_RAM** to **0** in the Preprocessor project settings. The following data is collected with the LeNet MNIST model on the RT685, with weights in RAM and removing the **input.bin** and **output.bin** test images found by default in the RT685 Glow SDK project. It also includes the RAM required for the HiFi4 DSP library for the configurations that use the HiFi4 compile argument.

Table 3. HiFi4 Glow compile options

Glow Compile Options	Weights	Input/Output	Activations	Compiled Library	Total Project Flash	Total Project RAM	Inference Time on RT685
Quantized w/ Symmetric Power 2 No CMSIS-NN – No HiFi4	431,360	3,200	15,232	8,400	475,244	507,932	60.39 ms
Quantized w/ Symmetric Power 2 with CMSIS-NN – No HiFi4	431,360	3,200	20,992	23,236	479,116 (+3,872)	513,692 (+5,760)	28.52 ms
Quantized w/ Symmetric Power 2 with HiFi4 – No CMSIS-NN	432,832	3,200	23,872	7,020	1,156,168 (+677,052)	1,194,428 (+680,736)	2.51 ms
Quantized w/ Symmetric Power 2 with HiFi4 and CMSIS-NN	432,832	3,200	23,872	6,956	1,155,988 (-180)	1,194,428 (+0)	2.49 ms

3 Glow project size

Now let us explore how Glow memory requirements impact an example Glow application in MCUXpresso IDE. We will use LeNet MNIST as an example with the RT1060 MNIST Glow project as our baseline. All buffers have been set to zero to start with, and this data is captured using MCUXpresso IDE 11.2 with RT1060 SDK 2.8.0 using the “Release” high optimization compile settings. The goal is that by looking at the Glow output bundle results, it can be determined if a particular model could fit on a particular board or device based on memory requirements.

The lenet_mnist.h file contains the following defines:

```
#define LENET_MNIST_CONSTANT_MEM_SIZE 431360
#define LENET_MNIST_MUTABLE_MEM_SIZE 3200
#define LENET_MNIST_ACTIVATIONS_MEM_SIZE 20992
```

Table 4. MCUXpresso SDK compiled project size for Glow

Description	Flash (bytes)	RAM (bytes)	Change (bytes)	Details
Bare-Bones	21,424	8,496	Baseline	Baseline SDK project with PRINTF support
Adding in .o	31,064	8,496	+9,640 Flash	The Glow compiled library .o file. Note that this is less than the size on the .o file on the PC hard drive.
Adding input/ output and scratch buffers	31,064	32,688	+24,192 RAM	Adding LENET_MNIST_MUTABLE_MEM_SIZE and LENET_MNIST_ACTIVATIONS_MEM_SIZE requires 24 KB more RAM.
Adding weights in Flash	462,432	32,688	+431,368 Flash	If weights are read from Flash, this will not affect RAM usage but will require 431,360 bytes (LENET_MNIST_CONSTANT_MEM_SIZE) of non-volatile memory. The extra 8 bytes are for alignment.
Adding weights in RAM	462,424	464,048	+431,360 RAM	If weights are read from RAM, the project will require LENET_MNIST_CONSTANT_MEM_SIZE additional bytes of RAM. This is optional but may decrease inference time.
Adding memory for static input image and including image	465,560	467,184	+3,136 Flash +3,136 RAM	Allocating space for a 28*28 pixel monochromatic (1 color channel) image, with each pixel being a 4-byte wide floating point value: 28*28*1*4=3,136. If image is not statically included, then will save this amount of space.

The minimum memory requirements for a particular model can then be found by using a simple formula using numbers found in the Glow bundle header file:

- Flash: **Base Project + CONSTANT_MEM_SIZE + .o library File**
- RAM: **Base Project + MUTABLE_MEM_SIZE + ACTIVATIONS_MEM_SIZE**

3.1 Effect of weights read from RAM vs Flash

The weight data can be read directly from non-volatile Flash during inferencing by adding a “const” in front of the weight array definition:

```
uint8_t const constantWeight[LENET_MNIST_CONSTANT_MEM_SIZE] = {
#include "lenet_mnist.weights.txt"
};
```

This decreases the amount of RAM required compared to having the weights copied to RAM. The trade-off is that the inference time will often be slower. The amount of impact on inference time is very dependent on the model, with some models having a very negligible impact while others have a significant impact. It is recommended to experiment with your particular model to see what the trade-off is for that model. The effect on the amount of Flash required is negligible since the weights need to be stored in the Flash in both scenarios.

Below is an example for some models on the RT1060 using the default SDK demos. You can see that having weights in RAM makes a much larger difference in inference time for the LeNet MNIST model than for the CIFAR10 model.

Table 5. Comparing weights read from Flash or RAM

	LeNet MNIST	CIFAR10
Inference Time Weights read from Flash	17 ms	30 ms
Inference Time Weights read from RAM	10 ms	29 ms
RAM required for Weights read from Flash (bytes)	35,824	104,816
RAM required for Weights read from RAM (bytes)	467,184 (+431,360)	138,288 (+33,472)

4 Conclusion

This document describes how to calculate the Glow memory requirements using the generated Glow bundle. The MCU and external memory requirements can then be known ahead of time for a particular model. It also explored the impact of quantization on memory usage and the impact of reading weight data from Flash vs RAM.

5 Revision history

Rev	Date	Description
0	11/2020	Initial Release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject tounidentified vulnerabilities. Customers are responsible for the design and operation of theirapplications and products to reduce the effect of these vulnerabilities on customer’s applicationsand products, and NXP accepts no liability for any vulnerability that is discovered. Customersshould implement appropriate design and operating safeguards to minimize the risks associatedwith their applications and products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: November 2020
Document identifier: AN13001