Improving Performance and Processing Efficiency of GAN Neural Networks using ARM64 Process Architecture

C. Strommen

LibreCS, January 2023

TABLE OF CONTENTS

INTRODUCTION

Generative Adversarial Networks (GANs) are a powerful class of neural networks that have been used to generate a wide range of outputs, including images, videos, and text. They are composed of two main components: a generator and a discriminator. The generator generates new samples, and the discriminator attempts to distinguish these generated samples from real samples. Training a GAN requires significant computational resources, and the choice of hardware and software can have a significant impact on the efficiency of training. In this whitepaper, we will explore the theoretical efficiency of training GANs on arm64 processors using the software in the repository https://github.com/LibreCS/vqgarm.

There are several advantages of using arm64 processors over x86 processors for machine learning workloads:

1. Power efficiency: arm64 processors are known for their power efficiency, which can be beneficial for running machine learning workloads that require a lot of computational power. This can help reduce power consumption and costs.
2. Cost-effective: arm64 processors are generally less expensive than x86 processors, which can make it more cost-effective to use them for machine learning workloads.
3. Smaller form factor: arm64 processors are often smaller in size than x86 processors, which can be beneficial for applications that require a smaller form factor, such as mobile or embedded devices.
4. Scalability: arm64 processors can be used in a variety of different devices, from small embedded systems to large servers, allowing for more scalability in machine learning workloads.
5. Software compatibility: arm64 processors have good software compatibility with a wide range of operating systems and programming languages, which makes it easier to develop and run machine learning workloads on arm64 processors.
6. Hardware specific optimization: arm64 processors have different instruction sets and therefore different libraries that are optimized for them. It's important to use hardware-specific libraries and frameworks to make the best use of the arm64 processors for machine learning workloads.

It's worth noting that the advantages of one architecture over another will vary depending on the specific workload, task and hardware configuration. It's always important to test and evaluate the performance of the different architectures for a given task.

BACKGROUND

Arm64 processors are a type of processor architecture that is used in a wide range of devices, including smartphones, tablets, and servers. They are designed to be energy-efficient and have a smaller form factor compared to traditional x86 processors. However, the performance of arm64 processors in training GANs has not been well studied.

The theoretical efficiency of training GANs on arm64 processors can be determined by comparing the performance of arm64 processors to other types of processors. One way to do this is to measure the number of floating-point operations per second (FLOPS) that can be performed by the processor.

The software in the repository https://github.com/LibreCS/vqgarm is specifically designed to optimize the training of GANs on arm64 processors. It uses vector quantization to reduce the dimensionality of the generator and discriminator, making it possible to train GANs on arm64 processors with less computational resources.

A Generative Adversarial Network (GAN) is a type of deep learning model that consists of two neural networks: a generator and a discriminator. The generator creates new data samples that are similar to a given training set, while the discriminator tries to distinguish the generated samples from the real samples. The two networks are trained simultaneously in an adversarial way, with the generator trying to produce samples that can fool the discriminator and the discriminator trying to correctly identify the generated samples. The training process can be formalized using two loss functions, one for the generator and one for the discriminator. The generator's loss function is typically defined as the negative log-likelihood of the discriminator's output on the generated samples, while the discriminator's loss function is the sum of the log-likelihoods of its output on the real samples and the log-likelihoods of its output on the generated samples. These loss functions are used to update the generator and discriminator's weights via backpropagation.

PRACTICAL CONSIDERATIONS

The theoretical efficiency of training GANs on arm64 processors using this software library will depend on the specific architecture of the GAN, as well as the size and complexity of the dataset being used. Additionally, the quality of the generated samples and the convergence rate of the model will also play a role in determining the overall efficiency of the training process.

It is important to note that this software is under active development, and the efficiency may vary based on the current version. The accuracy of the generated samples and the convergence rate of the model will also play a role in determining the overall efficiency of the training process. It is always recommended to test different configurations and use cases, and compare the results before making any conclusion.

There are several practical considerations for large-scale GAN neural network training on both x86 and arm64 processors:

1. Memory and storage: GANs require a large amount of memory to store the model parameters and the generated samples. Make sure to have enough memory and storage to handle the size of the training dataset and the model.
2. Computational power: GANs are computationally intensive, especially during the training process. Make sure to have enough computational power to handle the training process. This can be achieved by using powerful processors like x86 or arm64 processors, or by using specialized hardware like GPUs or TPUs.
3. Network architecture: GANs are highly dependent on the network architecture of both generator and discriminator. Be sure to choose an architecture that is suitable for the task at hand and make sure it's optimized for x86 or arm64 processors.
4. Optimization techniques: GANs are sensitive to the choice of optimization algorithms, learning rates, and other hyperparameters. It's recommended to perform a thorough hyperparameter search to find the best settings for the specific task and hardware.
5. Distributed training: GANs can be trained on multiple machines simultaneously, which can speed up the training process. Make sure to use a distributed training framework that is compatible with both x86 and arm64 processors.
6. Hardware specific optimization: x86 and arm64 processors have different instruction sets and therefore different libraries that are optimized for them. It's important to use hardware-specific libraries and frameworks to make the best use of the hardware.
7. Monitoring and debugging: GANs can be difficult to train and debug. It's important to monitor the training process and use tools such as Tensorboard to keep track of the model's performance, and to be able to identify and fix problems.

PROCEDURES

It is difficult to quantitatively describe the results of a theoretical test between x86 and arm64 processors based on a GAN neural network workload without knowing the specific details of the test, such as the dataset, the network architecture, and the hardware configuration. However, in general, the results of such a test would likely depend on the specific workload and the optimization techniques used.

Some general considerations for such a test would be:

- Training time: The training time for a GAN on x86 and arm64 processors could differ due to the different instruction sets and the specific optimization techniques used.
- Accuracy: The accuracy of the generated samples by a GAN on x86 and arm64 processors could also differ, depending on the specific optimization techniques used and the network architecture.
- Power consumption: Power consumption during training could be different between x86 and arm64 processors, as some x86 processors are generally more power-efficient than others.
- Memory and storage usage: the GANs trained on x86 or arm64 processors could have different memory and storage usage depending on the specific architecture and optimization techniques used.

It's important to note that a benchmark test should be run with the same configuration and dataset in both x86 and arm64 processors to get fair comparison. The results of such a test would provide a better understanding of the relative performance of x86 and arm64 processors for GAN workloads.

When benchmarking the performance of arm64 processors against x86 processors for processing machine learning workloads, it is important to control the input variables to ensure a fair comparison. Input variables refer to the specific parameters, settings, and configurations that are used during the benchmarking process. Failure to control input variables can lead to inaccurate or misleading results.

Some examples of input variables that should be controlled when benchmarking arm64 and x86 processors for machine learning workloads include:

- Network architecture: The architecture of the neural network being used should be the same for both arm64 and x86 processors.
- Dataset: The dataset used for training and testing should be the same for both arm64 and x86 processors.
- Optimization techniques: The optimization techniques used, such as the optimizer, learning rate, batch size, and regularization should be the same for both arm64 and x86 processors.
- Hardware configuration: The specific hardware configurations, such as the number of cores and the amount of memory, should be the same for both arm64 and x86 processors.

By controlling these input variables, it is possible to obtain accurate and reliable results that can be used to compare the performance of arm64 and x86 processors for machine learning workloads. Additionally,

it is important to monitor and measure other variables such as power consumption and memory usage during the benchmarking process as these can have an impact on the performance of the processors.

For the testing conducted, the following generation phrase was used:

```
python3 generate.py -cd cpu -i 500 -s 400 400 -p "A painting of a wizard
riding a white horse into the sunset"
python3 generate.py -cd cpu -i 500 -s 400 400 -p "A drawing of a fiery
bull fighting a white unicorn"
python3 generate.py -cd cpu -i 500 -s 400 400 -p "A sketch of a yellow
bird perched on a green tree"
```

All testing was conducted using automated code testing frameworks such as Tox and pytest to control any other environmental and processing unknowns.

RESULTS

Using incremental GAN generation times as benchmarks for overall processing speed can provide valuable insights into the performance of different processors for machine learning workloads. GANs are computationally intensive models that require significant processing power and memory. By measuring the time it takes to generate new samples at different stages of the GAN training process, it is possible to obtain an estimate of the overall processing speed of a particular processor.

For example, one could measure the time it takes to generate a set number of samples after a certain number of training iterations, such as every 100 iterations. By comparing the generation times across different processors, it is possible to determine which processor is faster and more efficient at handling GAN workloads.

Additionally, by measuring the generation time at different stages of the GAN training process, it is also possible to determine how the processing speed of a particular processor changes over time, which can be useful for identifying bottlenecks in the training process and for determining the optimal training conditions.

It is important to note that incremental GAN generation time alone should not be used as the only benchmark for overall processing speed as the time it takes to generate new samples may not be the only metric that matters, other metrics such as power consumption and memory usage could also be considered.

When benchmarking the performance of a software or hardware, it is important to avoid data ramp and deramp effects. Data ramp refers to the initial period of operation where the system is still adjusting to the workload and not performing at its optimal level. Deramp refers to the period at the end of the operation where the system is not performing at its optimal level, usually due to cache effects.

One way to avoid the data ramp and deramp effects when benchmarking is to calculate performance data using results from the middle of running the software. By measuring the performance of the system in the middle of the operation, you can eliminate the effects of data ramp and deramp and obtain a more accurate representation of the system's performance.

For example, if you are running a GAN training process, you could measure the generation time every 100 iterations, but only consider the results from iterations 1000 to 2000 to avoid data ramp and deramp.

By measuring the performance of the system in the middle of the operation, it is possible to obtain a more accurate representation of the system's performance, which can be used to compare the performance of different systems or to identify bottlenecks in the software or hardware.

Additionally, it is important to use representative test cases to benchmark the performance, as the ramp and deramp effects may vary depending on the specifics of the workload.

For a nominal run using x86 processors:

```
213  100it [08:01,  4.72s/it]
214  101it [08:04,  4.93s/it]
215  102it [08:09,  4.85s/it]
216  103it [08:13,  4.80s/it]
217  104it [08:18,  4.75s/it]
218  105it [08:23,  4.75s/it]
219  106it [08:27,  4.72s/it]
```

For a nominal run using arm64 processors:

```
196  52it [13:00, 12.65s/it]
197  53it [13:13, 12.85s/it]
198  54it [13:25, 12.60s/it]
199  55it [13:37, 12.45s/it]
200  56it [13:49, 12.40s/it]
201  57it [14:01, 12.29s/it]
202  58it [14:14, 12.34s/it]
```

Normalizing the computing power against the power consumption of each processor (~9.8W for arm64 and ~128W for x86) the following calculations were made.

$$\eta_{x86} = P_{nominal} \cdot t_{avg} \cdot n_{iterations} = 128\,W \cdot \frac{4.8556\,s}{3600\,\frac{s}{hr}} \cdot 400i \approx 69.057\,Wh$$

$$\eta_{arm64} = P_{nominal} \cdot t_{avg} \cdot n_{iterations} = 9.8\,W \cdot \frac{12.5114\,s}{3600\,\frac{s}{hr}} \cdot 400i \approx 13.6235\,Wh$$

Given that generating urgency is not a priority, arm64 processors provide a 507% power efficiency advantage over x86 processors, in turn reducing overall power consumption for large GAN generator processes.

CONCLUSION

In conclusion, training GANs on arm64 processors using the software in the repository https://github.com/LibreCS/vqgarm is a promising approach to increasing the efficiency of GAN training on arm64 processors. The vector quantization technique used in this software library can significantly reduce the computational resources required for training GANs on arm64 processors. However, the theoretical efficiency will depend on the specific architecture of the GAN, as well as the size and complexity of the dataset being used. Further research and testing is needed to fully understand the efficiency of training GANs on arm64 processors using this software library.

Key Takeaways

- Training GANs on arm64 processors using the software in the repository https://github.com/LibreCS/vqgarm is a promising approach to increase the efficiency of GAN training on arm64 processors.
- The vector quantization technique used in this software library can significantly reduce the computational resources required for training GANs on arm64 processors.
- The theoretical efficiency of training GANs on arm64 processors using this software library will depend on the specific architecture of the GAN, as well as the size and complexity of the dataset being used. Further research and testing is needed to fully understand the efficiency of training GANs on arm64 processors using this software library.