Natural Parallelism: Advancing Machine Learning with OpenACC

Yohandi
School of Data Science
The Chinese University of Hong Kong, Shenzhen
YOHANDI@LINK.CUHK.EDU.CN

1 Introduction

1.1 Background

With the extension concepts of parallel computing, natural parallelism can be further blended into the realm of machine learning. This project specifically focuses on accelerating softmax regression, neural networks, and convolutional neural networks using OpenACC. OpenACC offers a simplified approach to programming for parallel computing, particularly for GPU acceleration.

1.2 Objective

The project's aim is to demonstrate the significant impact of parallel computing in machine learning, especially when dealing with extensive data and complex models. By optimizing machine learning algorithms for GPUs, computation speeds can be greatly enhanced, thereby improving the overall efficiency and performance of these algorithms.

The project involves understanding the core algorithms of softmax regression and neural networks, followed by the application of OpenACC for parallel computing on GPUs. This exploration will not only provide hands-on experience with parallel computing in machine learning but also highlight the importance of effective hardware utilization to boost algorithmic performance.

2 Softmax Regression and Neural Networks: Basic Principles

2.1 Softmax Regression

Softmax Regression, a generalization of logistic regression, is particularly effective in handling multiclass classification problems. It's commonly used in scenarios where an input needs to be classified into one of several categories. The fundamental operation of softmax regression involves mapping an input vector to a probability distribution over predicted output classes.

Algorithmic Workflow:

1. **Logit Computation:** For each class, the algorithm computes a logit, which is a function of the input vector and a set of parameters (weights and biases). These logits are essentially the raw outputs of linear models for each class.
2. **Softmax Function:** The logits are then passed through the softmax function. This step involves exponentiating each logit and then normalizing these values so that they sum up to one. The output of the softmax function is a probability distribution: each element represents the probability of the input belonging to a particular class.
3. **Training Process:** During training, the softmax regression model aims to adjust its weights and biases to minimize the difference between the predicted probability distribution and the actual dis-

tribution (the true class of each input). This is typically done using a gradient descent optimization algorithm, where gradients are computed based on the softmax loss function.

Applications: Softmax regression is widely used in various fields like image recognition, natural language processing, and more, due to its simplicity and effectiveness in multiclass classification.

2.2 Neural Networks (NNs)

Neural Networks represent a more complex and powerful class of models compared to softmax regression. Particularly, this project focuses on a 2-layer Neural Network, which includes an input layer, one hidden layer, and an output layer.

Forward Propagation:

1. **Layer Processing:** Each layer in the network transforms its input via a weighted sum followed by a non-linear activation function. The input layer receives the initial data, and each subsequent layer uses the output of the previous layer as its input.
2. **Activation Functions:** Common choices for activation functions in hidden layers include ReLU (Rectified Linear Unit) and sigmoid functions. These functions introduce non-linearity to the model, enabling it to learn complex patterns.

Backward Propagation:

1. **Loss Computation:** The difference between the network's prediction and the actual output (the loss) is calculated using a loss function, like mean squared error for regression tasks or cross-entropy for classification tasks.
2. **Gradient Descent:** Backpropagation is used to compute the gradient of the loss function with respect to each weight in the network. The network's weights are then updated in the direction that most reduces the loss, typically using the SGD (Stochastic Gradient Descent) method.

Advantages of NNs: NNs can model complex patterns thanks to their deep structures and non-linear activation functions. They are widely used in various applications like speech recognition, language modeling, and complex game playing.

2.3 Convolutional Neural Networks (CNNs) [Extra Credit]

Convolutional Neural Networks are a specialized kind of neural network used primarily for processing structured grid data such as images. CNNs employ a mathematical operation called convolution in at least one of their layers.

Key Components:

1. Convolutional Layers: These layers use a set of learnable filters that scan the input data and perform convolution operations. This allows the network to automatically and adaptively learn spatial hierarchies of features from input images.
2. Pooling Layers: These layers reduce the spatial dimensions (width and height) of the input volume for the next convolutional layer. Common pooling methods include max pooling and average pooling.
3. Fully Connected Layers: Towards the end of the network, CNNs often use fully connected layers where every input is connected to every output. This is similar to the traditional multi-layer perceptron structure.

Training and Acceleration with OpenACC:

1. **Efficient Training:** CNNs require substantial computational resources for training due to the complexity and number of operations involved. This is where OpenACC comes into play, offering a way to parallelize these computations, particularly on GPU architectures.

2. **Optimizations:** Utilizing OpenACC, the convolutional and pooling operations can be significantly accelerated. Static filters, when compiling, can lead to improved performance by reducing runtime computation.

Performance Goals: For this project, the implementation of CNN with OpenACC should aim for higher accuracy compared to the 2-layer NN. This involves careful design and tuning of the network architecture, including the number and size of filters, pooling layers, and fully connected layers.

Application: CNNs have been pivotal in breakthroughs in tasks like image and video recognition, image classification, medical image analysis, and many other areas of computer vision.

3 Project Tasks and Implementation

The initial setup involves preparing the working directory with the necessary datasets, source code, and compilation scripts. The dataset comprises training and testing data for the MNIST dataset, essential for training and evaluating the softmax regression and neural network models.

3.1 Task 1: Train MNIST with Softmax Regression

This task involves training the MNIST dataset using softmax regression. The training process includes matrix operations like matrix multiplication, softmax normalization, and gradient descent optimization. The key challenge here is to effectively handle the matrix operations to optimize the softmax regression model.

3.2 Task 2: Accelerate Softmax with OpenACC

The acceleration of softmax regression using OpenACC entails applying parallel computing strategies to optimize matrix operations within the softmax regression training. The focus is on efficiently distributing computational tasks across GPU resources to expedite the training process.

3.3 Task 3: Train MNIST with Neural Network

Training a 2-layer neural network on the MNIST dataset involves implementing forward and backward propagation. The key components include matrix operations for calculating activations, applying the ReLU function, and updating weights using SGD.

3.4 Task 4: Accelerate Neural Network with OpenACC

Similar to Task 2, this task focuses on accelerating the neural network training process using OpenACC. The challenge is to parallelize the matrix operations and gradient calculations involved in neural network training, ensuring efficient utilization of GPU resources.

3.5 Task 5: Train MNIST with Convolutional Neural Network [Extra Credit]

This task involves creating and training a Convolutional Neural Network (CNN) on the MNIST dataset. CNNs are highly efficient for image recognition, leveraging spatial hierarchy in image data. The process includes defining the CNN architecture with convolutional, pooling, and fully connected layers, followed by training through forward and backward propagation. The training includes implementing activations (e.g., ReLU), loss computation (e.g., cross-entropy), and weight updates using optimization algorithms like SGD.

3.6 Task 6: Accelerate Convolutional Neural Network with OpenACC [Extra Credit]

In this task, the focus is on accelerating the training of the CNN model for the MNIST dataset using OpenACC. This involves adapting the CNN training process to leverage GPU resources for efficient computation. Key challenges include parallelizing the convolutional and pooling operations, managing memory efficiently between the CPU and GPU, and ensuring that the computational gains from acceleration do not compromise the model's accuracy or learning capability.

3.7 Anticipated Challenges and Solutions

- **Matrix Operations Optimization:** Ensuring efficient implementation of matrix operations is crucial. Solutions may include optimizing memory access patterns and employing shared memory on GPUs.
- **Balancing GPU Workload:** Achieving an optimal distribution of tasks across GPU cores is essential for maximizing parallel processing efficiency.
- **Handling Numerical Stability:** Careful handling of numerical operations, especially in softmax calculations, is necessary to avoid issues like overflow or underflow.
- **Debugging and Profiling:** Effective debugging and profiling tools will be essential in identifying performance bottlenecks and optimizing the code.

4 Experiment Results

4.1 Model Performance Metrics

In this subsection, we report on the performance metrics of the various machine learning tasks undertaken in this project. The performance metrics are critical in evaluating the effectiveness of the machine learning models post-training. The tables below will present the loss and error rates after the initial training period and after the completion of all epochs, both on the training and test datasets. These metrics are essential for understanding the models' predictive performance and generalization capabilities to new, unseen data.

Table 1: Loss and Error Metrics After the First Training Epoch

Task	Epochs	Loss		Error	
		Train	Test	Train	Test
Softmax Regression	10	0.35134	0.33588	0.10182	0.094
Neural Network	20	0.13466	0.14293	0.04023	0.0424
Convolutional Neural Network	10	0.02423	0.07163	0.00552	0.023

Table 2: Loss and Error Metrics After the Last Training Epoch

Task	Epochs	Loss		Error	
		Train	Test	Train	Test
Softmax Regression	10	0.27886	0.28442	0.07847	0.0797
Neural Network	20	0.00783	0.06655	0.00048	0.02
Convolutional Neural Network	10	0.00492	0.06553	0.00012	0.019

4.2 Execution Time Performance

The execution time is a performance metric in assessing the efficiency of the machine learning algorithms when subjected to training with and without acceleration techniques. This subsection details the time taken for each task across multiple runs (takes) and the average execution time, providing insight into the consistency and reliability of the algorithm's performance. The recorded times are instrumental in understanding the computational demands of each task and the benefits of using acceleration frameworks like OpenACC.

Table 3: Execution Time for Machine Learning Tasks

Task	Method	Epochs	Takes (ms)				
			Take 1	Take 2	Take 3	Take 4	Take 5
Softmax Regression	Sequential	10	9195	9184	9196	9201	9203
	OpenACC	10	958	957	957	961	962
Neural Network	Sequential	20	682848	-	-	-	-
	OpenACC	20	7105	7070	7097	7060	7188
Convolutional Neural Network	Sequential	10	514447	-	-	-	-
	OpenACC	10	6471	6552	6491	6520	6491

Table 4: Speedup Factors for Machine Learning Tasks

Task	Speedup Factor
Softmax Regression	9.6
Neural Network	96.1
Convolutional Neural Network	79.1

4.3 Profiling

Profiling is crucial for understanding the performance characteristics of an application. It provides insights into which parts of the code are the most time-consuming and thus potential targets for optimization. The tables below list the top 10 functions that consume the most execution time during the OpenACC acceleration of the Softmax Regression and Neural Network tasks, along with the percentage of total time, the total time in nanoseconds, the number of instances the function was called, and the average time per instance.

Table 5: Profiling Results for Softmax Regression with OpenACC

Name	Time(%)	Total Time (ns)	Instances	Average (ns)
matrix_dot_openacc	61.1	393,930,100	6,020	65,436.9
mean_softmax_loss_openacc	10.1	65,380,362	20	3,269,018.1
matrix_dot_trans_openacc	8.9	57,679,849	6,000	9,613.3
matrix_minus_openacc	4.6	29,791,016	12,000	2,482.6
matrix_softmax_normalize_openacc_130	4.0	25,970,404	6,000	4,328.4
matrix_softmax_normalize_openacc_124	3.6	22,977,930	6,000	3,829.7
vector_to_one_hot_matrix_openacc	2.8	18,239,833	6,000	3,040.0
matrix_mul_scalar_openacc	2.4	15,166,578	6,000	2,527.8
matrix_div_scalar_openacc	2.3	14,966,135	6,000	2,494.4
mean_softmax_loss_openacc_red	0.1	630,010	20	31,500.5

Table 6: Profiling Results for Neural Network with OpenACC

Name	Time(%)	Total Time (ns)	Instances	Average (ns)
matrix_dot_openacc	60.7	2,672,498,888	24,080	110,984.2
matrix_dot_trans_openacc	25.2	1,109,457,703	24,000	46,227.4
mean_softmax_loss_openacc	2.6	115,063,600	40	2,876,590.0
matrix_minus_openacc	2.5	110,670,068	36,000	3,074.2
matrix_mul_scalar_openacc	2.0	85,954,912	24,000	3,581.5
matrix_div_scalar_openacc	1.9	85,693,023	24,000	3,570.5
matrix_trans_dot_openacc	1.3	55,174,915	12,000	4,597.9
ReLU_openacc	0.9	39,469,316	12,040	3,278.2
matrix_softmax_normalize_openacc_130	0.9	37,678,597	12,000	3,139.9
matrix_softmax_normalize_openacc_124	0.7	32,391,271	12,000	2,699.3

5 Analyses and Findings

5.1 Performance Analysis Based on Execution Time

The execution time results delineate a clear advantage of utilizing OpenACC for accelerating machine learning tasks. For instance, Softmax Regression achieved a speedup factor of 9.6 when using OpenACC, cutting down the average execution time from 9195.8 ms to 959 ms. This suggests that GPU acceleration is not merely an incremental improvement but a transformative change in how quickly a model can be trained.

In more computationally intensive tasks, such as Neural Network training, the benefits of GPU acceleration become even more pronounced. The average execution time was reduced from 682848 ms to 7104 ms, yielding a speedup factor of 96.1. This dramatic decrease indicates that GPU acceleration is particularly effective for tasks with a high degree of parallelism inherent in their computation.

The Convolutional Neural Network task also saw a considerable performance improvement, with the execution time decreasing from 514447 ms to 6505 ms when utilizing OpenACC, corresponding to a speedup factor of approximately 79.1. This efficiency gain is particularly significant given the complexity of CNNs and their widespread use in real-world applications.

5.2 Profiling Results Analysis

Profiling for the Softmax Regression task revealed that matrix multiplication, represented by the function matrix_dot_openacc, is the primary computational bottleneck, taking up 61.1% of the total execution time. This was expected due to the nature of the task where dot products are a fundamental operation. The average time per instance for this function was reduced significantly with GPU acceleration, implying that the optimization of matrix operations should be a focal point for further speed improvements.

For the Neural Network task, the function matrix_dot_openacc also dominated the profile, consuming 60.7% of the total time, with an average of 110,984.2 ns per instance. The optimization of this function through OpenACC indicates a potential for even greater speedups, especially as the complexity of the network increases.

5.3 Speedup and Efficiency Analysis

The speedup factors across different tasks suggest that the efficiency of parallelization scales with the complexity of the task. While all tasks benefited from GPU acceleration, the Neural Network task, which is inherently more complex, saw the greatest improvement. This correlation suggests that as the complexity of the task increases, so does the potential for performance gains through parallel processing.

The Softmax Regression, while less complex than the Neural Network, still achieved a significant speedup. This demonstrates that even for less complex tasks, GPU acceleration can provide substantial benefits, likely due to the reduction in time spent on matrix-related operations.

The efficiency gains as indicated by the speedup factors are a compelling argument for integrating GPU acceleration into the workflow of machine learning tasks. The results clearly demonstrate that for certain tasks, particularly those that are matrix operation-heavy, the investment in GPU resources can lead to substantial improvements in performance.

5.4 Conclusion

The findings from our experiments clearly illustrate the benefits of GPU acceleration for machine learning tasks. OpenACC has proven to be a valuable tool in reducing the execution time of models, particularly for complex tasks such as Neural Network training. The analyses indicate that the more computationally intensive the task, the greater the benefit from acceleration. These results underscore the importance of parallel processing capabilities in modern machine learning and serve as a guide for future research and development in the field.

Compilation and Execution Instructions

Within the submitted zip file, a folder $\boxed{\text{project3}}$ can be found, this will be the main project folder.

1. Navigate to the folder
2. Run $\boxed{\text{run.sh}}$ or simply paste the below sequence of commands:

```
scancel -u 120040025
bash ./test.sh
squeue -o "%.18i %.9P %.25j %.9u %.2t %.10M %.6D %R"
```

3. Upon completion, one can find out the results information in the same directory.