Title

NeuromorphicLLM: Energy-Efficient Large Language Model Inference via Neuromorphic Computing

Problem Statement

Current GPU-based inference for large language models (LLMs) is energy-inefficient and struggles to handle the sparse, event-driven nature of natural language processing. Existing hardware accelerators for LLMs are primarily designed for dense matrix operations and don't fully exploit the sparsity in activations and weights.

Motivation

Neuromorphic computing architectures, inspired by the human brain, are naturally suited for sparse, event-driven computation and could potentially offer significant energy efficiency gains for LLM inference. By leveraging spiking neural networks (SNNs) and memristive devices, we can design a hardware-software co-design approach that better matches the computational patterns of LLMs while dramatically reducing energy consumption.

Proposed Method

We propose NeuromorphicLLM, a novel hardware-software co-design approach for energy-efficient LLM inference. On the hardware side, we design a custom neuromorphic chip with SNNs optimized for sparse tensor operations. The chip features a crossbar array of memristive devices for efficient vector-matrix multiplications and implements a spike-based attention mechanism. On the software side, we develop a compilation framework to map pre-trained LLMs to the SNN architecture, including a novel algorithm for converting continuous-valued activations to spike trains while preserving information. We also introduce a spike-based beam search algorithm for efficient decoding. To handle the increased sparsity, we implement an adaptive synaptic pruning mechanism that dynamically adjusts the network connectivity based on activation patterns.

Step-by-Step Experiment Plan

Step 1: Neuromorphic Hardware Design

Design the neuromorphic chip architecture using a hardware description language (e.g., Verilog). Implement the crossbar array of memristive devices, spike-based attention mechanism, and on-chip memory hierarchy. Use a cycle-accurate simulator (e.g., Gem5) to model the hardware behavior.

Step 2: SNN Conversion Algorithm

Develop an algorithm to convert pre-trained LLM weights and activations to SNN representations. Implement rate-based and temporal coding schemes for spike generation. Optimize the conversion process to minimize information loss and maintain model accuracy.

Step 3: Spike-based Attention and Beam Search

Implement spike-based versions of the attention mechanism and beam search algorithm. Ensure these algorithms can operate efficiently on the neuromorphic hardware architecture.

Step 4: Adaptive Synaptic Pruning

Develop an algorithm for dynamic synaptic pruning based on activation patterns. Implement this mechanism in the hardware simulation and the software framework.

Step 5: Compilation Framework

Create a software toolchain to map pre-trained LLMs onto the neuromorphic hardware. This should include modules for model parsing, SNN conversion, pruning, and hardware-specific optimizations.

Step 6: Baseline Implementation

Implement GPU-based inference for the same LLM architectures using PyTorch or TensorFlow. This will serve as the baseline for performance and energy comparisons.

Step 7: Dataset Preparation

Prepare datasets for language modeling (e.g., WikiText-103) and text generation tasks (e.g., CNN/Daily Mail for summarization). Ensure the datasets are properly formatted for both the baseline and NeuromorphicLLM implementations.

Step 8: Performance Evaluation

Run inference on both the baseline GPU implementation and the NeuromorphicLLM simulation. Measure inference speed (tokens/second), energy consumption (Joules/token), and output quality (perplexity for language modeling, ROUGE scores for summarization).

Step 9: Scalability Analysis

Evaluate the performance of NeuromorphicLLM across different model sizes (e.g., 125M, 1.3B, 13B parameters) to assess scalability.

Step 10: Ablation Studies

Conduct ablation studies to quantify the impact of individual components (e.g., spike-based attention, adaptive pruning) on overall performance and energy efficiency.

Step 11: Error Analysis

Analyze cases where NeuromorphicLLM performs worse than the baseline. Identify potential sources of error in the SNN conversion or hardware implementation.

Step 12: Optimization and Refinement

Based on the results and error analysis, refine the hardware design, SNN conversion algorithm, and compilation framework to improve performance and energy efficiency.

Test Case Examples

Baseline Prompt Input (GPU-based Inference)

Summarize the following news article: [Input a full news article from the CNN/Daily Mail dataset]

Baseline Prompt Expected Output (GPU-based Inference)

[A coherent summary of the input article, typically 3-4 sentences long]

Proposed Prompt Input (NeuromorphicLLM Inference)

Summarize the following news article: [Input the same news article as used in the baseline]

Proposed Prompt Expected Output (NeuromorphicLLM Inference)

[A coherent summary of the input article, ideally of similar quality to the baseline output]

Explanation

The NeuromorphicLLM output should be of comparable quality to the baseline GPU-based inference, while consuming significantly less energy. We expect the NeuromorphicLLM to potentially have slightly lower quality due to the SNN conversion process, but this should be offset by the massive gains in energy efficiency.

Fallback Plan

If the proposed NeuromorphicLLM approach fails to achieve the expected energy efficiency gains or suffers from significant quality degradation, we can pivot the project in several directions. First, we could focus on a more in-depth analysis of where and why the SNN conversion process leads to information loss, potentially developing novel techniques to mitigate these issues. Second, we could explore hybrid approaches that combine neuromorphic computing with traditional digital circuits, leveraging the strengths of both paradigms. Third, we could investigate the use of our neuromorphic architecture for specific sub-components of LLM inference (e.g., only for the feed-forward layers) rather than the entire model. Finally, we could expand our focus to include training of LLMs on neuromorphic hardware, which might lead to models that are inherently more suited for spike-based computation.

Ranking Score: 6