Machine Learning Model Performance Report

Executive Summary

In this project, we (Usman and Ahmad) implemented and evaluated three different versions of a binary classification model: serial CPU processing, parallel CPU processing, and GPU-accelerated processing. All implementations achieved the same prediction results, but with dramatically different processing times:

Implementation	Processing Time	% Improvement vs Serial
GPU-accelerated	7.70 seconds	**94.3%**
Parallel CPU	126.73 seconds	**5.7%**
Serial CPU	134.44 seconds	baseline

Our GPU implementation successfully exceeded our required performance target of 70% time reduction, achieving a **94.3%** reduction compared to the serial implementation.

Processing Time Analysis

I

Key Observations:

- **GPU Implementation**: We achieved a remarkable 94.3% reduction in processing time compared to our serial implementation, completing the task in just 7.7 seconds.
- **Parallel CPU Implementation**: This yielded only a marginal improvement of 5.7% over our serial implementation, reducing processing time from 134.44 to 126.73 seconds.
- **Performance Target**: Our requirement of at least 70% reduction in processing time was significantly exceeded by our GPU implementation.

Our Implementation Approaches

Serial CPU Implementation

- **Technology**: We used scikit-learn pipelines with GridSearchCV
- **Parallelization**: None (single-threaded execution)
- **Optimization**: Standard scikit-learn implementation

- **Processing Time**: 134.44 seconds (baseline)
- **Key Components**:
  - Standard logistic regression with grid search
  - Sequential preprocessing pipeline
  - No parallelization in model training or hyperparameter tuning

Parallel CPU Implementation

- **Technology**: We enhanced our approach with scikit-learn and joblib parallelization
- **Parallelization**: We implemented threading with `n_jobs=-1` (all available cores)
- **Optimization**: Thread-based parallelization of GridSearchCV
- **Processing Time**: 126.73 seconds (5.7% improvement)
- **Key Components**:
  - We added parallel processing with `parallel_backend('threading')`
  - We utilized all available CPU cores for parallel grid search
  - We maintained the same preprocessing pipeline as our serial implementation
  - We added CPU count detection to automatically utilize all cores

GPU-accelerated Implementation

- **Technology**: We implemented TensorFlow with GPU acceleration
- **Parallelization**: GPU-based parallel processing
- **Optimization**: Multiple TensorFlow-specific optimizations
- **Processing Time**: 7.70 seconds (94.3% improvement)
- **Key Components**:
  - We implemented mixed precision computation (`mixed_float16`)
  - We created an optimized data pipeline using `tf.data.Dataset`
  - We enabled JIT compilation (`jit_compile=True`)
  - We configured GPU memory growth for optimal resource usage
  - We added early stopping to reduce unnecessary computation
  - We implemented data caching and prefetching
  - We optimized batch processing

Hardware Utilization Analysis

CPU Utilization

- **Serial Implementation**: Our initial approach was limited to a single core, leaving most CPU resources idle
- **Parallel Implementation**: We achieved better CPU utilization, but saw diminishing returns
- **Bottlenecks We Identified**:
  - I/O operations during data loading
  - Thread synchronization overhead
  - Memory bandwidth limitations

GPU Utilization

- **Implementation Benefits**:
  - We leveraged massive parallelization of matrix operations
  - We utilized dedicated high-bandwidth memory
  - We took advantage of specialized hardware for floating-point operations
- **Optimizations We Used**:
  - Mixed precision to leverage GPU Tensor Cores
  - Memory growth management to optimize VRAM usage
  - Data pipeline optimizations to keep GPU fed with data

Performance Optimization Strategies

Data Processing Optimizations

- **GPU Implementation**:
  - We used TensorFlow's `tf.data` API for efficient data loading
  - We implemented data caching to reduce preprocessing overhead
  - We added prefetching to overlap computation and data preparation
  - We used batching to optimize throughput

Computational Optimizations

- **GPU Implementation**:
  - Our mixed precision computation reduced memory usage and increased throughput
  - We enabled JIT compilation to optimize execution of TensorFlow operations
  - We designed a simple model architecture to minimize unnecessary computations

- We implemented early stopping to prevent wasted computation cycles

Parallel Processing Differences

- **Parallel CPU**: We found this was limited by threading overhead and memory bandwidth
- **GPU**: We leveraged its massively parallel architecture specifically designed for matrix operations

Our Recommendations

1. **Infrastructure Investment**:
   - We recommend standardizing on GPU-accelerated implementations for time-critical machine learning workloads
   - If GPUs are not available, we suggest investigating distributed computing solutions that may offer better scaling than threading
2. **Implementation Improvements**:
   - For CPU-only environments, we recommend exploring alternative parallel backends like 'multiprocessing' instead of 'threading'
   - We suggest optimizing data loading and preprocessing steps which may be bottlenecks in the pipeline
   - Consider smaller grid search spaces when using CPU implementations
3. **Future Optimization Opportunities**:
   - We believe exploring specialized hardware like TPUs could provide further acceleration
   - We recommend investigating quantization techniques to reduce model size and increase inference speed
   - For larger datasets, we suggest considering distributed training across multiple GPUs

Conclusion

Our GPU-accelerated implementation successfully exceeded the performance target by reducing processing time by 94.3% compared to our serial implementation. This dramatic improvement demonstrates the significant benefits ofleveraging GPU acceleration for machinelearning tasks, even with relatively simple models like logistic regression. Our parallel CPU implementation showed only modest improvements, suggesting that thread-based parallelization may not be sufficient for achieving substantial performance gains in this particular workflow.