VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Parallels Processing

Final Report

Team Contribution

Instructor: Phạm Trọng Nghĩa
Lê Nhựt Nam

Group members:

No.	Full name	Student ID
1	Huỳnh Lê Hải Dương	22127081
2	Nguyễn Triều Khoáng	22127204

HO CHI MINH CITY, 2025

Team planning

Work contribution

No.	Task name	Student	Deadline	Details	Notes
1	Literature Review & Project Setup	Hải Dương, Triều Khoáng	01/12/2025	Research autoencoder architecture, CUDA programming. Setup environment, download CIFAR-10 dataset.	Finished (100%)
2	Data Loader Implementation	Hải Dương, Triều Khoáng	01/12/2025	Implement CIFAR-10 binary file loader, batch generator with shuffling, data normalization.	Finished (100%)
3	CPU Baseline Implementation	Hải Dương	04/12/2025	Implement CPU layers (Conv2D, MaxPool2D, ReLU, Upsampling), CPU autoencoder, training loop with backpropagation.	Finished (100%)
4	GPU Basic Implementation	Hải Dương	07/12/2025	Port layers to GPU, implement CUDA kernels for convolution, pooling, ReLU. GPU autoencoder with memory management.	Finished (100%)
5	GPU Optimization V1 (Shared Memory)	Triều Khoáng	10/12/2025	Implement shared memory tiling for convolution, reduce global memory accesses, optimize memory access patterns.	Finished (100%)
6	GPU Optimization V2 (Fusion + Streams)	Triều Khoáng	10/12/2025	Kernel fusion (Conv2D + ReLU + Bias), CUDA streams for computation/data transfer overlap, vectorized memory access.	Finished (100%)
7	Feature Extraction	Triều Khoáng	13/12/2025	Implement encoder feature extraction, save features to binary format, support both CPU and GPU paths.	Finished (100%)
8	SVM Training Pipeline	Triều Khoáng	15/12/2025	Integrate cuML SVM, load features and labels, train and evaluate SVM classifier, performance benchmarking.	Finished (100%)
9	Demo Interface	Hải Dương, Triều Khoáng	16/12/2025	Gradio web interface, load trained models (autoencoder + SVM), real-time image classification, visualization.	Finished (100%)

No.	Task name	Student	Deadline	Details	Notes
10	Performance Analysis & Documentation	Hải Dương, Triều Khoáng	17/12/2025	Performance profiling and benchmarking, create visualization charts, write Jupyter notebook report, code documentation.	Finished (100%)
11	Video demo	Hải Dương, Triều Khoáng	18/12/2025	Record and edit video demonstration of the complete system, showcase all features and performance results.	Finished (100%)

Timeline

No.	Task name	Start date	End date	Duration
1	Literature Review & Project Setup	29/11/2025	01/12/2025	3 days
2	Data Loader Implementation	29/11/2025	01/12/2025	3 days
3	CPU Baseline Implementation	02/12/2025	04/12/2025	3 days
4	GPU Basic Implementation	05/12/2025	07/12/2025	3 days
5	GPU Optimization V1 (Shared Memory)	08/12/2025	10/12/2025	3 days
6	GPU Optimization V2 (Fusion + Streams)	08/12/2025	10/12/2025	3 days
7	Feature Extraction	11/12/2025	13/12/2025	3 days
8	SVM Training Pipeline	13/12/2025	15/12/2025	3 days
9	Demo Interface	14/12/2025	16/12/2025	3 days
10	Performance Analysis & Documentation	29/11/2025	17/12/2025	19 days
11	Video demo	16/12/2025	18/12/2025	3 days

Note: Task 10 (Documentation) is conducted continuously throughout the project. Tasks 8 and 9 can overlap partially.

Work Distribution Summary

Student	Tasks assigned	Total tasks	Main responsibilities
Huỳnh Lê Hải Dương	1, 2, 3, 4, 9, 10, 11	8	CPU/GPU implementation, optimizations, demo interface, documentation, video demo
Nguyễn Triều Khoáng	1, 2, 5, 6, 7, 8, 9, 10, 11	9	Literature review, data loader, GPU optimization V2, feature extraction, SVM training, demo interface, documentation, video demo

Key Achievements

- **Performance Optimization**: Achieved 10.91× speedup over CPU baseline through GPU parallelization and optimizations
- **Training Time**: Reduced autoencoder training time from 5.7 hours (CPU) to 31.5 minutes (GPU optimized)
- **Feature Extraction**: Successfully extracted 8,192-dimensional features from 60,000 CIFAR-10 images in under 23 seconds
- **Classification Accuracy**: Achieved 67.53% test accuracy using SVM classifier on extracted features
- **Code Quality**: Implemented modular, well-documented codebase with separate CPU and GPU implementations
- **Complete Pipeline**: Built end-to-end system from data loading to real-time inference via Gradio interface

Technologies and Tools Used

- **Programming Languages**: C++17, CUDA, Python 3
- **GPU Computing**: NVIDIA CUDA Toolkit, cuML (RAPIDS)
- **Deep Learning**: Custom autoencoder implementation (no frameworks)
- **Machine Learning**: LIBSVM, cuML SVM for classification
- **Data Processing**: NumPy, custom CIFAR-10 binary loader
- **Visualization**: Matplotlib, Seaborn, Jupyter Notebook
- **Web Interface**: Gradio for interactive demo
- **Build System**: Makefile for compilation and project management
- **Version Control**: Git for collaborative development

Challenges and Solutions

- **Challenge 1: Upsampling Backward Pass Gradient Accumulation**
  - *Problem*: Upsampling forward pass maps one input pixel to four output pixels (2×2 upsampling). During backward pass, four gradient values must be accumulated into a single input gradient position, requiring careful indexing and accumulation logic.
  - *Solution*: Implemented efficient gradient accumulation by mapping each output gradient back to the corresponding input position using integer division (ih = oh » 1, iw = ow » 1) and accumulating gradients correctly without atomic operations.
  - *Lesson*: When implementing custom layers, carefully track how forward and backward passes map between tensor dimensions. Visualize the data flow at each step to avoid gradient accumulation errors.
- **Challenge 2: Shared Memory Overhead for Small Images**
  - *Problem*: Initial attempt to use shared memory tiling for convolution showed that for small images (32×32), the overhead of loading tiles and synchronization exceeded the benefit of data reuse.
  - *Solution*: Switched to register blocking and weight caching strategy. Used `__restrict__` pointers and loop unrolling to maximize register usage and compiler optimizations. For small kernels (3×3), this approach proved more efficient than shared memory tiling.
  - *Lesson*: Not all optimization techniques work for all problem sizes. Profile and measure - what works for large images may not work for small ones. The optimal strategy depends on image size, kernel size, and available shared memory.
- **Challenge 3: Memory Layout and Coalescing**
  - *Problem*: Initial naive implementation had poor memory access patterns, with threads accessing non-consecutive memory locations, leading to uncoalesced memory accesses and poor bandwidth utilization (potentially 10× reduction in effective bandwidth).
  - *Solution*: Reorganized data access patterns to ensure threads in a warp access consecutive memory locations. Used proper indexing schemes and vectorized loads (float4) where possible to maximize memory bandwidth. Ensured memory alignment for vectorized operations.
  - *Lesson*: Memory access patterns are often more important than raw computation. Uncoalesced accesses can reduce effective bandwidth dramatically. Always design kernels with memory coalescing in mind from the start.

Project Statistics

Metric	Value
Total lines of code (C++/CUDA)	5,488
Total lines of code (Python)	680
Number of CUDA kernels	15+
Number of CPU functions	20+
Model parameters	751,875
Feature dimension	8,192
Training images processed	50,000
Test images processed	10,000
GPU memory usage (batch size 64)	814 MB