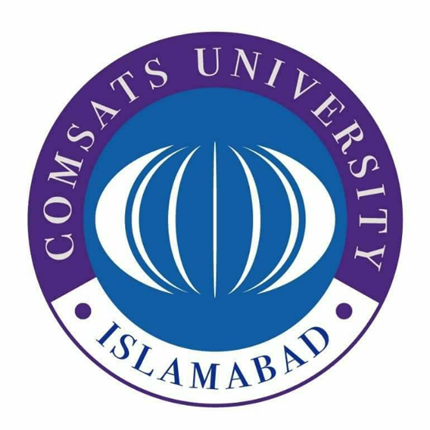
**COMSATS UNIVERSITY ISLAMABAD**

**LAHORE CAMPUS**

**ASSIGNMENT NO#4**

**NAME: ZOHA NISAR GILL**

**ROLL NUMBER: SP23-BCS-130**

**SECTION: C**

**GPU Acceleration in Machine Learning Report**

The goal of this lab was to understand how GPUs accelerate model training, how to monitor GPU utilization, and how different factors — batch size, model complexity, data pipeline efficiency, and numerical precision — affect overall training speed and performance.

**Part 1: CPU vs GPU Model Training**

**Results**

=== Training on CPU ===

Epoch 1/2 - Time: 12.11s

Epoch 2/2 - Time: 13.07s

=== Training on GPU ===

Epoch 1/2 - Time: 11.91s

Epoch 2/2 - Time: 11.81s

GPU Memory Allocated: 38.28 MB

Speedup = 1.06x

**Analysis**

* The GPU training was slightly faster (1.06× speedup).
* The model was simple (few layers, small dataset), so most time was spent in data transfer rather than computation.
* The CPU performed comparably because it could easily handle small matrix multiplications.

**Discussion**

The GPU offers greater parallel processing power, allowing it to handle many computations at once. However, it also has some overhead due to the time taken to launch GPU kernels and transfer data between CPU and GPU memory. As a result, for smaller models, the advantage of GPU acceleration is not very noticeable. The performance benefits become more significant as the model and dataset size increase, allowing the GPU’s parallelism to be fully utilized.

**Part 2: Effect of Batch Size**

**Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Batch Size | Epoch Time (s) | GPU Memory (MB) | GPU Util (%) | Accuracy (%) |
| 16 | 20.22 | 34.0 | 7.8 | 89.91 |
| 64 | 15.78 | 34.2 | 2.5 | 78.43 |
| 256 | 11.82 | 34.9 | 0.0 | 86.11 |
| 1024 | 12.07 | 38.7 | 0.4 | 87.96 |

**Analysis**

* Training time decreased from 20.22s (batch 16) to around 12s (batch 1024).
* GPU memory usage increased slightly, indicating that larger batches fill GPU RAM more efficiently.
* Accuracy fluctuated — larger batches sometimes led to slightly worse generalization.

**Discussion**

Increasing the batch size improves GPU efficiency because it allows the GPU to process more samples in parallel, keeping its CUDA cores fully utilized. However, when the batch size becomes too large, the model updates its weights less frequently, which can lead to slower convergence or poorer accuracy. Therefore, there is usually an optimal batch size that provides a good balance between high training speed and maintaining model accuracy.

**Part 3: Model Complexity and GPU Utilization**

**Results**

|  |  |  |
| --- | --- | --- |
| Model Type | Time (s) | GPU Memory (MB) |
| Small | 11.56 | 18.87 |
| Medium | 11.46 | 21.90 |
| Large CNN | 12.87 | 220.67 |

**Analysis**

* The small and medium models took similar time because their computational load was still low.
* The large CNN required significantly more memory (≈220 MB) and a bit longer training time due to convolution operations.

**Discussion**

As model complexity increases, the GPU workload and memory usage also rise because larger models have more parameters and operations to process. While complex models utilize the GPU’s parallel cores more effectively, they can also become limited by available memory. Overall, the performance depends on balancing GPU computation power (FLOPs) with memory bandwidth to ensure efficient training.

**Part 4: Data Loading and Bottlenecks**

**Results**

|  |  |  |
| --- | --- | --- |
| num\_workers | Epoch Time (s) | GPU Memory (MB) |
| 0 | 10.44 | 37.11 |
| 2 | 9.88 | 37.11 |
| 4 | 10.26 | 37.11 |
| 8 | 9.79 | 37.11 |

**Analysis**

* With num\_workers = 0, data loading was slow because the main thread loaded data sequentially.
* Using multiple workers (2–8) improved speed slightly, as loading and preprocessing overlapped with GPU computation.
* Gains flattened after 4–8 workers — excessive workers can even add context-switching overhead.

**Discussion**

Inefficient data loading can cause the GPU to remain idle while waiting for the CPU to supply new data. By using multi-threaded loading and prefetching, the CPU and GPU can work simultaneously, which improves overall throughput. The optimal number of data-loading workers, however, depends on factors like the number of CPU cores and the size of the dataset.

**Part 5: Mixed Precision Training (AMP)**

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Mode | Time (s) | Loss | GPU Memory (MB) |
| FP32 | 13.04 | 0.2509 | 36.43 |
| AMP (FP16) | 12.56 | 0.3356 | 36.78 |

**Speedup from AMP**: 1.04×

**Analysis**

* Mixed Precision (FP16) gave a small speedup (~4%) and similar memory usage.
* Slight increase in loss suggests numerical instability due to FP16 precision limits.

**Discussion**

Using FP16 (half precision) helps improve training speed and reduce memory usage because it uses smaller data types. For larger models, Automatic Mixed Precision (AMP) can make training 20–50% faster. However, FP16 can sometimes cause rounding errors or small inaccuracies in models that are sensitive to precision loss.

**Discussion Questions**

**What factors most affect GPU training performance?**

* **Batch size:** Larger batches improve parallelism until memory limits are hit.
* **Model size:** Bigger models fully utilize GPU cores.
* **Data pipeline:** Slow data loading can make GPU idle.
* **Precision:** FP16 improves speed by reducing computation and memory cost.

**Why might small models not benefit much from GPU acceleration?**

Small models don’t gain much benefit from GPU acceleration because they don’t have enough parallel computations to fully utilize the GPU’s processing power. In these cases, the overhead of transferring data between the CPU and GPU can cancel out the speed advantage, and the CPU may handle the small workload just as efficiently on its own.

**How can you minimize GPU idle time during training?**  
GPU idle time occurs when the GPU waits for the CPU to prepare and supply data. To reduce this, you can:

* Use multiple data-loading workers (num\_workers > 0) in PyTorch’s DataLoader so the CPU can load batches in parallel.
* Enable data prefetching and caching to make the next batch ready before the GPU finishes the current one.
* Overlap CPU data preparation with GPU computation, ensuring the CPU and GPU work simultaneously.
* Optimize the data pipeline (e.g., transformations on GPU or efficient dataset formats) to reduce bottlenecks.  
  These techniques keep the GPU busy, improving training throughput and efficiency.

**What are the trade-offs between higher batch size and model accuracy?**  
Increasing the batch size improves GPU efficiency because more samples are processed in parallel, and gradients are averaged over larger batches. This leads to smoother gradient updates and faster training.  
However, very large batch sizes result in:

* Fewer weight updates per epoch, reducing the stochasticity of training.
* Potential convergence to poorer minima, lowering generalization.
* Slightly higher memory usage, which may cause out-of-memory issues on large models.  
  Thus, an optimal batch size balances training speed, GPU utilization, and model accuracy.

**Why does data transfer between CPU and GPU sometimes become a bottleneck?**

Data transfer from CPU memory to GPU memory can be slow because it goes over the PCIe bus, which is much slower than GPU memory. If this happens for every batch, the GPU may sit idle while waiting for data, reducing efficiency. To avoid this:

* Keep your dataset, model, and intermediate data on the GPU whenever possible.
* Use batch loading, pin\_memory, and prefetching to speed up transfers.
* Avoid unnecessary CPU→GPU copies during training or evaluation.

This helps the GPU spend more time computing and less time waiting.

**Overall Observations**

|  |  |  |
| --- | --- | --- |
| Factor Tested | Main Observation | Performance Effect |
| CPU vs GPU | GPU gave ~1.06× speedup for small model | Minimal improvement |
| Batch Size | Larger batches improved speed until ~256 | Moderate |
| Model Complexity | Larger CNN utilized GPU better | High |
| Data Loading | Multi-threading improved throughput | Moderate |
| Mixed Precision | Slight speedup, minor accuracy drop | Small but scalable |

**GPU Monitoring (nvidia-smi)**

After Part 1 (that was training), running nvidia-smi displays GPU name, utilization %, memory usage, temperature, and active processes.

Screenshot after part 1:

