GROUP-226

S.NO	NAME	BITS ID	CONTRIBUTION %
1	Suresh M	2024ac05271	100%
2	Chatrathi Lavanya	2024ac05261	100%
3	Ravi Shankar	2024ac05260	100%
4	Disha Gaikwad	2024ac05424	100%
5	Midhun S	2024ad05463	100%

In [1]:
```python
import numpy as np
import time


# ============================================================
# Configuration
# ============================================================

K = 3                  # Pipeline stages
D = 4                  # Data parallel replicas
M = 6                  # Micro-batches
GRAD_SIZE = 120        # Must be divisible by D
COMM_MODE = "INT8"     # "FP16" or "INT8"
LR = 0.01


# ============================================================
# Quantization
# ============================================================

def quantize_int8(t):
    max_val = np.max(np.abs(t))
    scale = max_val / 127.0 if max_val != 0 else 1.0
    q = np.round(t / scale).astype(np.int8)
    return q, scale

def dequantize_int8(q, scale):
    return q.astype(np.float32) * scale


# ============================================================
# Device
# ============================================================

class Device:
    def __init__(self, rank, stage):
        self.rank = rank
        self.stage = stage
        self.weight = np.ones(GRAD_SIZE, dtype=np.float32)
        self.local_grad = np.zeros(GRAD_SIZE, dtype=np.float32)
```

```
        self.comm_bytes = 0


# ============================================================
# Hybrid Simulator
# ============================================================

class HybridSimulator:

    def __init__(self):

        self.devices = []
        self.step_matrix = []

        for dp in range(D):
            for stage in range(K):
                rank = dp * K + stage
                self.devices.append(Device(rank, stage))

        self.stage_groups = {
            s: [d for d in self.devices if d.stage == s]
            for s in range(K)
        }

    # ------------------------------------------------------------
    # 1F1B Pipeline Scheduling
    # ------------------------------------------------------------
    def run_pipeline(self):

        total_steps = M + K - 1

        for step in range(total_steps):

            step_events = []

            for d in self.devices:

                stage = d.stage
                warmup = K - stage - 1
                event = "-"

                # Forward
                if step < M and step >= stage:
                    mb = step - stage
                    event = f"F{mb}"

                # Backward
                if step >= warmup:
                    mb = step - warmup
                    if 0 <= mb < M:
                        event = f"B{mb}"
                        grad = np.random.randn(GRAD_SIZE).astype(np.float32)
                        d.local_grad += grad
```

```
            step_events.append((d.rank, event))

        self.step_matrix.append(step_events)

# ------------------------------------------------------------
# Chunked Ring AllReduce
# ------------------------------------------------------------
def ring_all_reduce(self, stage):

    group = self.stage_groups[stage]
    grads = np.stack([d.local_grad for d in group])
    chunks = np.split(grads, D, axis=1)

    # Reduce-Scatter
    for step in range(D - 1):
        for i in range(D):
            src = i
            dst = (i + 1) % D
            chunk_idx = (i - step) % D

            send_chunk = chunks[chunk_idx][src]

            if COMM_MODE == "FP16":
                payload = send_chunk.astype(np.float16)
                group[src].comm_bytes += payload.nbytes
                recv_chunk = payload.astype(np.float32)
            else:
                q, scale = quantize_int8(send_chunk)
                group[src].comm_bytes += q.nbytes
                recv_chunk = dequantize_int8(q, scale)

            target_idx = (dst - step - 1) % D
            chunks[target_idx][dst] += recv_chunk

    # All-Gather
    for step in range(D - 1):
        for i in range(D):
            src = i
            dst = (i + 1) % D
            chunk_idx = (i - step) % D

            send_chunk = chunks[chunk_idx][src]

            if COMM_MODE == "FP16":
                payload = send_chunk.astype(np.float16)
                group[src].comm_bytes += payload.nbytes
                recv_chunk = payload.astype(np.float32)
            else:
                q, scale = quantize_int8(send_chunk)
                group[src].comm_bytes += q.nbytes
                recv_chunk = dequantize_int8(q, scale)
```

```
            target_idx = (dst - step - 1) % D
            chunks[target_idx][dst] = recv_chunk

    final = np.concatenate(chunks, axis=1) / D

    for i, d in enumerate(group):
        d.local_grad = final[i]

# ------------------------------------------------------------
def optimizer_step(self):

    for stage in range(K):
        for d in self.stage_groups[stage]:
            d.weight -= LR * d.local_grad
            d.local_grad[:] = 0

# ------------------------------------------------------------
def evaluate_accuracy(self):

    np.random.seed(42)
    X = np.random.randn(300, GRAD_SIZE).astype(np.float32)
    true_w = np.random.randn(GRAD_SIZE).astype(np.float32)

    logits = X @ true_w
    y_true = (logits > 0).astype(int)

    model_w = self.stage_groups[0][0].weight
    preds = X @ model_w
    y_pred = (preds > 0).astype(int)

    return np.mean(y_pred == y_true) * 100

# ------------------------------------------------------------
def total_comm_bytes(self):
    return sum(d.comm_bytes for d in self.devices)

# ------------------------------------------------------------
def print_generation_timeline(self):

    print("\n================ PIPELINE GENERATION TIMELINE ================

    header = "Step | " + " | ".join(
        [f"R{d.rank}(S{d.stage})" for d in self.devices]
    )
    print(header)
    print("-" * len(header))

    for step, events in enumerate(self.step_matrix):
        row = f"{step:<4} | "
        for _, event in events:
            row += f"{event:^10}| "
        print(row)
```

```
        print("\n============================================================\n")

    # ---------------------------------------------------------
    def verify(self):

        print("Replica Consistency Check:")
        for stage in range(K):
            weights = [d.weight for d in self.stage_groups[stage]]
            diffs = [np.linalg.norm(weights[0] - w) for w in weights]
            print(f"  Stage {stage}: Max Weight Difference = {max(diffs)}")

    # ---------------------------------------------------------
    def run(self):

        print(f"\nHybrid Parallel Simulation (K={K}, D={D})")
        print(f"Total Nodes = {K * D}")
        print(f"Communication Mode = {COMM_MODE}\n")

        start_time = time.perf_counter()

        self.run_pipeline()

        for stage in range(K):
            self.ring_all_reduce(stage)

        self.optimizer_step()

        end_time = time.perf_counter()
        runtime = end_time - start_time

        accuracy = self.evaluate_accuracy()

        print("\n================ PERFORMANCE REPORT ================\n")
        print(f"Runtime (seconds): {runtime:.6f}")
        print(f"Prediction Accuracy: {accuracy:.2f}%")
        print(f"Total Communication Bytes: {self.total_comm_bytes()}")
        print("\n====================================================\n")

        self.verify()
        self.print_generation_timeline()


# ============================================================
if __name__ == "__main__":
    HybridSimulator().run()
```

```
Hybrid Parallel Simulation (K=3, D=4)
Total Nodes = 12
Communication Mode = INT8


================ PERFORMANCE REPORT ================

Runtime (seconds): 0.016619
Prediction Accuracy: 49.00%
Total Communication Bytes: 2160

====================================================

Replica Consistency Check:
  Stage 0: Max Weight Difference = 0.0003009646898135543
  Stage 1: Max Weight Difference = 0.0002646037610247731
  Stage 2: Max Weight Difference = 0.0002457082155160606

================ PIPELINE GENERATION TIMELINE ================

Step | R0(S0) | R1(S1) | R2(S2) | R3(S0) | R4(S1) | R5(S2) | R6(S0) | R7(S1) |
R8(S2) | R9(S0) | R10(S1) | R11(S2)
-------------------------------------------------------------------------------
-----------------------------------
0    |     F0    |     -     |     B0    |     F0    |     -     |     B0    |
F0    |     -     |     B0    |     F0    |     -     |     B0    |
1    |     F1    |     B0    |     B1    |     F1    |     B0    |     B1    |
F1    |     B0    |     B1    |     F1    |     B0    |     B1    |
2    |     B0    |     B1    |     B2    |     B0    |     B1    |     B2    |
B0    |     B1    |     B2    |     B0    |     B1    |     B2    |
3    |     B1    |     B2    |     B3    |     B1    |     B2    |     B3    |
B1    |     B2    |     B3    |     B1    |     B2    |     B3    |
4    |     B2    |     B3    |     B4    |     B2    |     B3    |     B4    |
B2    |     B3    |     B4    |     B2    |     B3    |     B4    |
5    |     B3    |     B4    |     B5    |     B3    |     B4    |     B5    |
B3    |     B4    |     B5    |     B3    |     B4    |     B5    |
6    |     B4    |     B5    |     -     |     B4    |     B5    |     -     |
B4    |     B5    |     -     |     B4    |     B5    |     -     |
7    |     B5    |     -     |     -     |     B5    |     -     |     -     |
B5    |     -     |     -     |     B5    |     -     |     -     |

============================================================
```

In [ ]: