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


# ============================================================
# Configuration (Modify for Experiments)
# ============================================================

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"


# ============================================================
# Quantization Utilities
# ============================================================

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 Definition
# ============================================================

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 Parallel Simulator
# ============================================================

class HybridSimulator:

    def __init__(self):

        self.devices = []
        self.timeline = []
        self.logical_time = 0

        # Create K × D devices
```

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

        # Group by pipeline stage (Data Parallel groups)
        self.stage_groups = {
            s: [d for d in self.devices if d.stage == s]
            for s in range(K)
        }

    # ------------------------------------------------------
    def log(self, rank, event):
        self.timeline.append((self.logical_time, rank, event))
        self.logical_time += 1

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

        total_steps = M + K - 1

        for step in range(total_steps):

            for d in self.devices:

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

                # Forward Phase
                if step < M and step >= stage:
                    self.log(d.rank, f"F{step - stage}")

                # Backward Phase
                if step >= warmup:
                    mb = step - warmup
                    if 0 <= mb < M:
                        self.log(d.rank, f"B{mb}")

                        # Gradient accumulation
                        grad = np.random.randn(GRAD_SIZE).astype(np.float32)
                        d.local_grad += grad

    # ------------------------------------------------------
    # True Chunked Ring All-Reduce (Deterministic)
    # ------------------------------------------------------
    def ring_all_reduce(self, stage):

        group = self.stage_groups[stage]

        # Stack gradients (D × GRAD_SIZE)
        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

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

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

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

        for d in self.stage_groups[stage]:
            self.log(d.rank, "OPT")
            d.weight -= 0.01 * d.local_grad
            d.local_grad[:] = 0

    # ------------------------------------------------------
```

```
    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")

        # 1F1B Pipeline
        self.run_pipeline()

        # Data Parallel Synchronization
        for stage in range(K):
            self.ring_all_reduce(stage)
            self.optimizer_step(stage)

        print("Simulation Completed.\n")

        self.verify()
        self.report_comm()
        self.print_timeline()

    # ------------------------------------------------------------
    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 report_comm(self):

        total = sum(d.comm_bytes for d in self.devices)
        print("\nCommunication Statistics:")
        print(f"  Total Communication Bytes = {total}")

    # ------------------------------------------------------------
    def print_timeline(self):

        print("\n================ EXECUTION TIMELINE ================\n")

        header = f"{'Time':<6} | {'Rank':<6} | {'Stage':<6} | Event"
        print(header)
        print("-" * len(header))

        for t, rank, event in self.timeline:
            stage = self.devices[rank].stage
            print(f"{t:<6} | {rank:<6} | {stage:<6} | {event}")

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


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

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

Simulation Completed.

Replica Consistency Check:
  Stage 0: Max Weight Difference = 0.00024210211995523423
  Stage 1: Max Weight Difference = 0.0003216343466192484
  Stage 2: Max Weight Difference = 0.0002757864131126553

Communication Statistics:
  Total Communication Bytes = 2160

================ EXECUTION TIMELINE ================

Time   | Rank   | Stage  | Event
---------------------------------
0      | 0      | 0      | F0
1      | 2      | 2      | B0
2      | 3      | 0      | F0
3      | 5      | 2      | B0
4      | 6      | 0      | F0
5      | 8      | 2      | B0
6      | 9      | 0      | F0
7      | 11     | 2      | B0
8      | 0      | 0      | F1
9      | 1      | 1      | F0
10     | 1      | 1      | B0
11     | 2      | 2      | B1
12     | 3      | 0      | F1
13     | 4      | 1      | F0
14     | 4      | 1      | B0
15     | 5      | 2      | B1
16     | 6      | 0      | F1
17     | 7      | 1      | F0
18     | 7      | 1      | B0
19     | 8      | 2      | B1
20     | 9      | 0      | F1
21     | 10     | 1      | F0
22     | 10     | 1      | B0
23     | 11     | 2      | B1
24     | 0      | 0      | F2
25     | 0      | 0      | B0
26     | 1      | 1      | F1
27     | 1      | 1      | B1
28     | 2      | 2      | F0
29     | 2      | 2      | B2
30     | 3      | 0      | F2
31     | 3      | 0      | B0
32     | 4      | 1      | F1
33     | 4      | 1      | B1
34     | 5      | 2      | F0
35     | 5      | 2      | B2
36     | 6      | 0      | F2
37     | 6      | 0      | B0
```

38	7	1	F1
39	7	1	B1
40	8	2	F0
41	8	2	B2
42	9	0	F2
43	9	0	B0
44	10	1	F1
45	10	1	B1
46	11	2	F0
47	11	2	B2
48	0	0	F3
49	0	0	B1
50	1	1	F2
51	1	1	B2
52	2	2	F1
53	2	2	B3
54	3	0	F3
55	3	0	B1
56	4	1	F2
57	4	1	B2
58	5	2	F1
59	5	2	B3
60	6	0	F3
61	6	0	B1
62	7	1	F2
63	7	1	B2
64	8	2	F1
65	8	2	B3
66	9	0	F3
67	9	0	B1
68	10	1	F2
69	10	1	B2
70	11	2	F1
71	11	2	B3
72	0	0	F4
73	0	0	B2
74	1	1	F3
75	1	1	B3
76	2	2	F2
77	2	2	B4
78	3	0	F4
79	3	0	B2
80	4	1	F3
81	4	1	B3
82	5	2	F2
83	5	2	B4
84	6	0	F4
85	6	0	B2
86	7	1	F3
87	7	1	B3
88	8	2	F2
89	8	2	B4
90	9	0	F4
91	9	0	B2
92	10	1	F3
93	10	1	B3

```
94     | 11     | 2      | F2
95     | 11     | 2      | B4
96     | 0      | 0      | F5
97     | 0      | 0      | B3
98     | 1      | 1      | F4
99     | 1      | 1      | B4
100    | 2      | 2      | F3
101    | 2      | 2      | B5
102    | 3      | 0      | F5
103    | 3      | 0      | B3
104    | 4      | 1      | F4
105    | 4      | 1      | B4
106    | 5      | 2      | F3
107    | 5      | 2      | B5
108    | 6      | 0      | F5
109    | 6      | 0      | B3
110    | 7      | 1      | F4
111    | 7      | 1      | B4
112    | 8      | 2      | F3
113    | 8      | 2      | B5
114    | 9      | 0      | F5
115    | 9      | 0      | B3
116    | 10     | 1      | F4
117    | 10     | 1      | B4
118    | 11     | 2      | F3
119    | 11     | 2      | B5
120    | 0      | 0      | B4
121    | 1      | 1      | B5
122    | 3      | 0      | B4
123    | 4      | 1      | B5
124    | 6      | 0      | B4
125    | 7      | 1      | B5
126    | 9      | 0      | B4
127    | 10     | 1      | B5
128    | 0      | 0      | B5
129    | 3      | 0      | B5
130    | 6      | 0      | B5
131    | 9      | 0      | B5
132    | 0      | 0      | OPT
133    | 3      | 0      | OPT
134    | 6      | 0      | OPT
135    | 9      | 0      | OPT
136    | 1      | 1      | OPT
137    | 4      | 1      | OPT
138    | 7      | 1      | OPT
139    | 10     | 1      | OPT
140    | 2      | 2      | OPT
141    | 5      | 2      | OPT
142    | 8      | 2      | OPT
143    | 11     | 2      | OPT

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

In [ ]: