第一章　实战案例：128芯Decode部署

第1条　背景信息

1）**模型**：DeepSeek-V3 (671B, W8A16)

2）**芯片**：SG2260E

a）算力：128 TFLOPS (Int8)

b）DRAM：64 GB

c）DRAM带宽：273 GB/s × 0.893 = 244 GB/s

3）**网络拓扑**：两层Switch（TP Group内64 GB/s，Group间16 GB/s）

4）**目标**：4K Input，TPS per Batch ≥ 10

第2条　步骤1：定义芯片架构

```
# SG2260E 架构配置
arch = TPUArch(
    core=8,                          # 单芯片核心数
    flops=128*1e12,                  # 128 TFLOPS (Int8)
    dram_bw=273e9*0.893,             # DRAM 有效带宽（理论带宽 × 利用率）
    intra_bw=64e9,                   # TP Group 内直连带宽
    inter_bw=16e9                    # TP Group 间 PCIe 带宽
)
```

关键决策：网络拓扑假设

拓扑方案	TP Group 内	TP Group 间	性能影响	适用场景
一层 Switch	64 GB/s	64 GB/s	最优	128 芯及以下，有高速互联
两层 Switch	64 GB/s	16 GB/s	-10~15%	256 芯，或交换机端口受限

⚠️**易错点**：

1）**不要低估带宽影响**：`inter_bw`从64降到16 GB/s会导致10-15%性能下降

2）**确保PCIe Link≥16GB/s**：如果只有8GB/s，性能会大幅下降，应避免

3）**DRAM带宽利用率**：通常取0.7-0.9，过于乐观会高估性能

第3条　步骤2：确定搜索空间与策略

1）芯片数量选择决策不同芯片数量的性能差异巨大：**关键发现**：

芯片数	推荐配置	TPS/Chip	总吞吐量	相对效率	适用场景
16	TP4+SP4+EP16,B4	462	7,392	427%	**Prefill**(FTL=2.2s)
32	TP4+SP8+EP32,B8	407	13,024	376%	**Prefill**(FTL=2.5s)
64	TP4+DP16+EP64,B8	69	4,416	64%	❌ 不推荐 Decode
128	**TP4+DP32+EP128,B10**	**108**	**13,824**	**100%**	✅ **Decode 推荐**

256	TP4+DP64+EP256,B12	138	35,328	128%	高吞吐 Decode

a) **16/32芯**：Prefill单芯效率极高（400%+），但Decode无法支撑大Batch

b) **64芯**：Decode效率仅64%，不推荐单独部署

c) **128芯**：性能与成本最佳平衡点⭐

d) **256芯**：总吞吐量翻倍，单芯效率提升28%

2) 并行策略决策**策略1：是否使用TBO？**

TBO（Two-Batch Overlap）通过双批次交替执行掩盖All2All通信。

判断标准：结论：❌ MoE不做TP（设置`moe_tp=1`，仅做EP）

```
moe_tp = 1  # MoE不做TP，仅做EP
```

策略3：MLA_TP值选择？

不同TP的实测对比（128芯）：**结论：**✅**TP=4是最优选择**

```
tp_candidates = [2, 4, 8]  # 重点测试TP=4
```

⚠️**策略决策易错点总结：**

a) 需要Batch B满足：不考虑All2All通信下能达到20 TPS per Batch

b) **SG2260E实测**：单芯Batch>4时仅能达到12-15 TPS per Batch

c) **结论：**❌ 不满足TBO条件，不使用TBO

```
将 tbo = False  # SG2260E不使用TBO
```

策略2：MoE部分是否做TP？

做TP的收益与损失对比：

MOE_TP	收益	损失	128 芯实测 TPS/Chip
1（仅 EP）	无额外通信	无	**108**⭐
2（TP+EP）	选中 TP Group 内专家时减少 All2All	Dispatch/Combine 需 All Gather	98 (-9.3%)
4（TP+EP）	同上	通信开销更大	93 (-13.9%)

MLA_TP	DP	Batch	TPS/Chip	通信占比	评价
2	64	10	101	低	计算效率低
4	**32**	**10**	**108**	**中**	**最优平衡**⭐
8	16	10	84	高	通信开销大

d) ❌**错误1**：盲目使用TBO → 实际上只有高带宽芯片且Batch能做大才有效

e) ❌**错误2**：MoE做大TP（TP>4）→ 通信开销抵消收益，性能反而下降

f） ❌**错误3：**TP设置过大（TP=16/32）→ AllReduce通信成为瓶颈

g） ✅**正确做法：**先用粗略规则缩小搜索空间，再精确评估

3） 生成候选配置

基于策略决策，生成所有可行配置：**128芯配置空间：**

```
def generate_configs(num_chips=128,
                     tp_candidates=[2, 4, 8],
                     batch_candidates=[6, 8, 10, 12, 16]):
    """
    生成所有可行的部署配置

    约束条件：
        tp × dp = num_chips (MLA 部分)
        moe_tp × ep = num_chips (MoE 部分)
    """
    configs = []
    moe_tp = 1  # 固定策略：MoE 不做 TP
    ep = num_chips

    for tp in tp_candidates:
        if num_chips % tp != 0:
            continue
        dp = num_chips // tp

        for batch in batch_candidates:
            configs.append({
                'bs': batch,
                'tp': tp,
                'dp': dp,
                'moe_tp': moe_tp,
                'ep': ep
            })

    return configs
```

MLA_TP	MLA_DP	MOE_EP	单芯 Batch 候选	配置数
2	64	128	[6, 8, 10, 12, 16]	5
4	**32**	**128**	**[6, 8, 10, 12, 16]**	**5**
8	16	128	[6, 8, 10, 12, 16]	5

第4条　步骤3：批量性能评估

```
def evaluate_all_configs(arch, configs, is_prefill=False, tbo=False):
```

```
    """批量评估所有配置的性能"""
    results = []

    for cfg_dict in configs:
        # 1. 创建 Config 对象
        cfg = Config(**cfg_dict)

        # 2. 创建模型并评估
        model = DeepSeekModel(arch, cfg, is_prefill=is_prefill)
        tps, flops, elaps, mfu, dram_occupy = model.eval(tbo=tbo)

        # 3. 计算关键指标
        total_batch = cfg.bs * cfg.tp * cfg.dp
        decode_time_ms = elaps / 1000.0
        tps_per_batch = 1000.0 / decode_time_ms
        tps_per_chip = tps / (cfg.tp * cfg.dp)
        dram_gb = dram_occupy / (1 << 30)

        # 4. 记录结果
        results.append({
            'Batch': cfg.bs,
            'MLA_TP': cfg.tp,
            'MLA_DP': cfg.dp,
            'MOE_TP': cfg.moe_tp,
            'MOE_EP': cfg.ep,
            'Total_Batch': total_batch,
            'Decode_Time_ms': decode_time_ms,
            'TPS_per_Batch': tps_per_batch,
            'TPS_per_Chip': tps_per_chip,
            'Total_TPS': tps,
            'MFU': mfu * 100,
            'DRAM_GB': dram_gb
        })

    return pd.DataFrame(results)
```

执行评估：

```
configs_128 = generate_configs(num_chips=128)
df_128 = evaluate_all_configs(arch, configs_128, is_prefill=False, tbo=False)
```

评估结果示例（128 芯，部分数据）：

Batch	MLA_TP	MOE_EP	Total_Batch	TPS/Batch	TPS/Chip	DRAM(GB)	MFU(%)

6	4	128	768	**11.6**	89	12.2	11.8
8	4	128	1024	**11.3**	90	12.7	12.1
10	**4**	**128**	**1280**	**10.3**	**103**	**13.1**	**12.6**
12	4	128	1536	9.4 ❌	112	13.5	13.0
16	4	128	2048	7.8 ❌	125	14.2	13.8

关键发现：

1) **TPS per Batch与Batch的反比关系**：Batch越大 → 延迟越长 → TPS/Batch越低
2) **TPS per Chip与Batch的正比关系**：Batch越大 → 总吞吐越高 → TPS/Chip越高
3) **SLO约束是硬边界**：Batch≥12时，TPS/Batch < 10，不满足SLO，必须被过滤
4) **访存受限特征**：MFU仅12-14%，说明瓶颈在DRAM带宽和通信，不是计算

评估技巧：

5) 先看一组配置（如TP=4）在不同Batch下的表现，找到SLO边界
6) 再对比不同TP（2/4/8）在相同Batch下的差异，确定最优TP
7) 最后在满足SLO的配置中，选择TPS/Chip最大的

第5条　步骤4：筛选与优选

1) 应用约束条件

```
# 约束 1: TPS per Batch >= 10 (SLO 要求)
feasible_df = df_128[df_128['TPS_per_Batch'] >= 10].copy()

# 约束 2: DRAM 容量检查 (SG2260E 为 64GB)
feasible_df = feasible_df[feasible_df['DRAM_GB'] <= 60].copy()

print(f"原始配置数: {len(df_128)}")
print(f"满足约束的配置: {len(feasible_df)}")
```

a) **筛选结果：**

原始配置数：15

满足TPS per Batch ≥ 10：8个 ✅

满足DRAM ≤ 60GB：8个 ✅（全部满足）

被过滤的配置：7个（主要是Batch≥12的配置）

b) **筛选注意事项：**

预留DRAM余量：实际可用容量通常只有标称的70-80%，需预留30-50%

给框架

SLO要预留Margin：建议实际TPS per Batch比SLO高10-20%（考虑框架开销）

不要只看TPS per Chip：必须先满足SLO，再优化单芯吞吐

2) 选择最优方案

```
# 按 TPS per Chip 降序排序
optimal_df = feasible_df.sort_values('TPS_per_Chip', ascending=False)
optimal_config = optimal_df.iloc[0]
```

a) **最优配置（128芯Decode方案）：**

配置参数：

MLA_TP: 4

MLA_DP: 32

MOE_TP: 1

MOE_EP: 128

单芯Batch: 10

总Batch: 1280

性能指标：

TPS per Batch: 10.30 (SLO≥10 ✅)

TPS per Chip: 108.00

总吞吐量: 13,824.00 TPS

MFU: 12.6%

DRAM占用: 12.70 GB (容量64GB ✅)

TPOT: 97.1 ms

b) **选择原因：**

✅**满足SLO且不浪费：**TPS per Batch=10.3，刚好满足≥10

✅**单芯效率最优：**在满足SLO的配置中，TPS per Chip达到最高值108

✅**TP=4最佳：**平衡了计算分摊和通信开销

✅**EP=128充分利用：**所有芯片参与专家并行，减少MoE通信

3) Top 5配置对比

Rank	TP	Batch	TPS/Batch	TPS/Chip	特点	适用场景

1⭐	4	10	10.30	**108.00**	刚好满足 SLO，TPS/Chip 最高	**标准 Decode 部署**
2	2	10	10.10	101.00 (-6.5%)	通信开销小	网络带宽受限
3	4	8	11.30	90.00 (-16.7%)	TPS/Batch 更高	延迟极敏感场景
4	4	6	11.60	89.00 (-17.6%)	TPS/Batch 最高	实时交互场景
5	2	8	10.80	86.00 (-20.4%)	通信少+延迟低	低负载+低带宽

4） **关键权衡：**

a） **Rank 1：**刚好满足SLO（10.3≥10），无性能浪费，单芯效率最高

b） **Rank 2-5：**虽然TPS/Batch更高（用户体验更好），但牺牲了总吞吐量（成本效益下降）

c） **为什么不选Batch=12？**TPS/Batch=9.4 < 10，违反SLO约束

第6条　步骤5：输出结果与验证

1） 保存评估结果

```
import os

output_dir = 'deployment_results'
os.makedirs(output_dir, exist_ok=True)

# 1. 保存完整扫描结果（CSV）
df_128.to_csv(f'{output_dir}/full_results_128chips.csv', index=False)

# 2. 保存可行配置（CSV）
feasible_df.to_csv(f'{output_dir}/feasible_results_128chips.csv', index=False)

# 3. 保存最优配置（JSON）
import json
optimal_json = {
    "chip_count": 128,
    "model": "DeepSeek-V3-671B-W8A16",
    "optimal_config": {
        "MLA_TP": int(optimal_config['MLA_TP']),
        "MLA_DP": int(optimal_config['MLA_DP']),
        "MOE_TP": int(optimal_config['MOE_TP']),
        "MOE_EP": int(optimal_config['MOE_EP']),
        "batch_per_chip": int(optimal_config['Batch']),
        "total_batch": int(optimal_config['Total_Batch'])
    },
    "performance": {
        "tps_per_batch": round(optimal_config['TPS_per_Batch'], 2),
        "tps_per_chip": round(optimal_config['TPS_per_Chip'], 2),
        "total_tps": round(optimal_config['Total_TPS'], 2)
    }
```

```
}

with open(f'{output_dir}/optimal_config_128chips.json', 'w') as f:
    json.dump(optimal_json, f, indent=2)
```

2) 性能详细分析

a) **Block耗时分析**（每层）：

Block	Time(μs)	占比(%)	瓶颈类型
Embedding	7	0.2	访存
MM_QKV	264	7.9	计算
MLA	379	11.4	访存
Attention_FC	173	5.2	计算
Dense_MLP	589	17.7	访存
MoE	634	**19.0**	通信+访存
LM_Head	1326	**39.8**	访存（词表大）

b) **性能瓶颈：**

LM_Head占比最高(39.8%)：词表大(128K)，输出矩阵乘访存密集

MoE占比第二(19.0%)：EP128时通信开销+访存

Dense_MLP占比第三(17.7%)：前馈网络访存密集

MFU仅12.6%：说明为访存受限，非计算受限

c) **优化建议：**

如果有256芯：选择EP256可将MoE耗时从634μs降至433μs（-32%）

LM_Head优化：考虑Top-K采样减少输出矩阵计算

网络拓扑：如果能做到TP Group间直连，可提升10-15%性能

3) 结果验证（暂无）

a) ✅ 主要误差来源：通信模型（实际网络拓扑差异）、访存模型（DRAM带宽利用率动态变化）

b) ✅ 建模可用于方案选型和趋势分析

c) ⚠️ 最终方案建议在实际硬件上验证