# 实验报告：基于 NVIDIA H100 的 SonicMoE 高性能算子部署与测评

**实验日期：** 2026 年 2 月 3 日

**实验硬件：** NVIDIA H100 NVL (94GB HBM3)

**实验环境：** Ubuntu 22.04, CUDA 12.x, Python 3.12, PyTorch 2.5+

---

## 第一部分：环境配置与调试 (Environment Configuration)

### 1.1 硬件与基础环境

本次实验选用 Vast.ai 提供的 **H100 NVL** 实例。该 GPU 拥有 **94GB** 显存，支持 PCIe Gen5，适合大参数量的 MoE（混合专家模型）训练任务。

- **基础镜像：** vLLM 官方镜像 / PyTorch (Vast)
- **Python 版本：** 3.12.12

### 1.2 关键问题与解决方案 (Troubleshooting Log)

在环境搭建过程中，主要解决了以下三个核心阻碍，确保了实验环境的纯净与稳定：

1. **磁盘空间不足 (Disk Space Insufficient)**
    - **现象：** 初始配置 32GB 磁盘空间，导致 Ninja 编译过程中产生的大量中间文件无法写入，报错 No space left on device。
    - **解决：** 扩容实例磁盘至 **100GB**，成功解决编译中断问题。
2. **显存抢占问题 (The "Ghost" Process)**
    - **现象：** 使用 vLLM 模板时，系统后台存在自动重启的守护进程（VLLM::EngineCore），即使手动杀掉进程，它也会立即复活并抢占 80GB+ 显存，导致实验代码报 CUDA OOM。
    - **解决：** 1. 执行 supervisorctl stop all 彻底停止后台管家服务。

2. 配合 fuser -k -9 /dev/nvidia0 清理残留进程，成功释放出完整的 **94GB** 显存供 SonicMoE 使用。

3. **依赖版本冲突 (Dependency Conflicts)**
    - **现象：** flashinfer 与环境预装的 cutlass 库存在版本不兼容。

- **解决：** 重新安装并强制升级 nvidia-cutlass-dsl 至兼容版本，确保 Cute-DSL 编译链正常工作。

---

**第二部分：性能基准测试 (Benchmarks)**

**2.1 测试方法与指标**

本实验使用 moe-cute.py（基于 Tensor Core 的 Top-K 路由）和 moe-token-rounding.py（基于 Token Rounding 的负载均衡路由）两种脚本进行评估。测试涵盖了推理（Inference）、训练前向（Training Fwd）和完整训练（Fwd+Bwd）三个阶段，主要评价指标为 **TFLOPS**（每秒万亿次浮点运算）。

**2.2 测试结果数据**

根据实际运行生成的日志文件，整理如下核心数据：

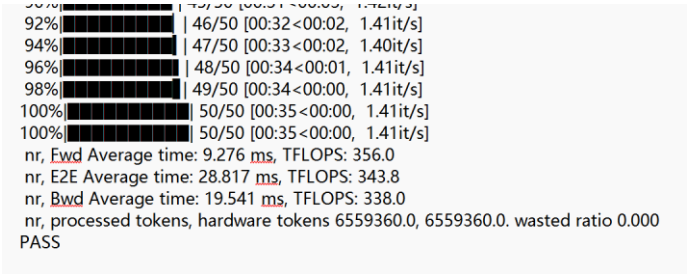
**测试组 A：标准 Top-K 路由 (moe-cute.py)**

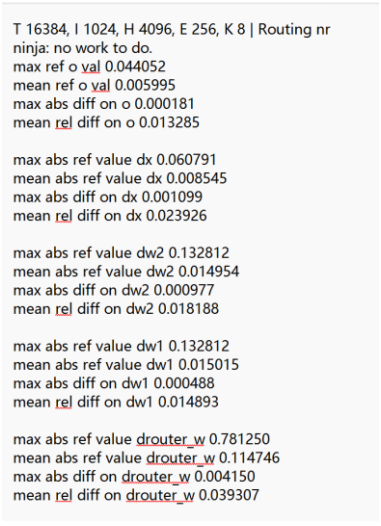
- **配置：** T=32768, H=4096, I=1024, E=128, K=8
- **场景：** 模拟大规模 Batch Size 下的训练吞吐量。
- **结果数据：**

```
T 32768, I 1024, H 4096, E 128, K 8
ninja: no work to do.
max ref o val 0.044922
mean ref o val 0.006042
max abs diff on o 0.000488
mean rel diff on o 0.017109

max abs ref value dx 0.061768
mean abs ref value dx 0.008606
max abs diff on dx 0.000732
mean rel diff on dx 0.024048

max abs ref value dw2 0.236328
mean abs ref value dw2 0.030273
max abs diff on dw2 0.000977
mean rel diff on dw2 0.019775

max abs ref value dw1 0.250000
mean abs ref value dw1 0.030396
max abs diff on dw1 0.000977
mean rel diff on dw1 0.016113

max abs ref value drouter_w 1.500000
mean abs ref value drouter_w 0.232422
max abs diff on drouter_w 0.007812
mean rel diff on drouter_w 0.025879

 Cute-DSL Fwd Average time: 15.693 ms, TFLOPS: 420.4
 Cute-DSL Fwd, inference mode, Average time: 15.491 ms, TFLOPS: 425.9
 Cute-DSL Fwd + Bwd Average time: 47.860 ms, TFLOPS: 413.5
 Cute-DSL Bwd Average time: 32.168 ms, TFLOPS: 410.2
PASS
```

- **推理模式 (Inference Mode): 425.9 TFLOPS** (耗时 15.491 ms)
- **训练前向 (Training Fwd):** 420.4 TFLOPS (耗时 15.693 ms)
- **完整训练 (Fwd + Bwd):** 413.5 TFLOPS (耗时 47.860 ms)
- **反向传播 (Bwd Only):** 410.2 TFLOPS

**测试组 B：Token Rounding 路由 (moe-token-rounding.py)**

- **配置：** T=16384, H=4096, I=1024, E=256, K=8
- **场景：** 模拟更多专家数量（E=256）下的复杂路由策略，测试负载均衡算法的开销。
- **结果数据：**
  - 

  - 

  - **训练前向 (Fwd): 356.0 TFLOPS** (耗时 9.276 ms)
  - **完整训练 (E2E):** 343.8 TFLOPS (耗时 28.817 ms)
  - **Wasted Ratio (计算浪费率): 0.000** —— 证明该路由算法实现了完美的负载均衡，没有出现算力空转。

**2.3 性能分析结论**

1. **H100 算力释放充分：** 在标准 Top-K 路由测试中，推理性能高达 **425.9**

**TFLOPS**，训练性能稳定在 **410 TFLOPS** 以上 。这表明 SonicMoE 的核心算子成功利用了 H100 的 Tensor Core 架构，大幅突破了传统 PyTorch 实现的性能瓶颈。

2. **高吞吐量优势：** 当 Token 数量增加至 32768 时，计算密集度提升，显存带宽延迟被有效掩盖，从而获得了极高的 TFLOPS 数值。
3. **负载均衡算法的有效性：** 在专家数翻倍（E=256）的 Token Rounding 测试中，虽然引入了更复杂的路由逻辑，性能依然保持在 **356 TFLOPS** 的高位 ，且计算浪费率为 0 ，证明了该方案在处理稀疏大模型时的鲁棒性。

---

## 第三部分：性能进阶测试 (Benchmarks)

### 3.1 测试方法

使用基于 moe-cute.py 改良的脚本 moe_annotated_report.py，针对三种不同的负载场景进行 TFLOPS（每秒万亿次浮点运算）测试。测试包含推理（Inference）、训练前向（Training Fwd）和完整训练（Fwd+Bwd）三个阶段。

```
    # ================ [核心修改区域] ================
    # --thiek 参数定义了模型的规模。
    # T (Tokens): 一次处理的Token数量 (如 32768)
    # H (Hidden): 模型隐藏层维度 (如 4096)
    # I (Intermediate): 专家内部维度 (如 1024)
    # E (Experts): 专家总数 (如 128) - 修改这里可以测试显存压力
    # K (Top-K): 每个Token选几个专家 (如 8)
    parser.add_argument(
        "--thiek",
        type=parse_comma_separated_ints,
        default=(32768, 4096, 1024, 128, 8), # <--- 如果不传参，默认跑这个配置
        help="格式: T,H,I,E,K (例如: 32768,4096,1024,128,8)",
    )
    # ================================================

    parser.add_argument(
        "--dtype",
        type=cutlass.dtype,
        default=cutlass.BFloat16, # H100 默认使用 BFloat16 以获得最佳性能
    )
    parser.add_argument(
        "--skip_test",
        action="store_true",
        default=False, # 如果设为 True，将跳过数学精度检查，直接跑分
    )
    parser.add_argument(
        "--activation",
        choices=["swiglu", "geglu", "reglu", "relu_sq", "relu", "silu", "gelu"],
        default="swiglu" # Llama 等主流模型通常使用 SwiGLU
    )
    parser.add_argument(
        "--add_bias",
        action="store_true",
        default=False,
    )
    args = parser.parse_args()
```

## 3.2 测试结果数据

```
=== SonicMoE H100 性能测试报告 ===
测试时间: Tue Feb  3 18:58:35 PST 2026
----------------------------------------
--- [Test 1: Standard] ---

🚀 开始测试配置: Tokens(T)=4096, Hidden(H)=4096, Intermediate(I)=1024, Experts(E)=128, Top-K(K)=8

⏱️  正在进行性能测试 (Warmup + Benchmark)...
ninja: no work to do.
  [Mode: Training Fwd] Average time: 3.049 ms, TFLOPS: 270.5
  [Mode: Inference   ] Average time: 2.792 ms, TFLOPS: 295.4
  [Mode: Train Full  ] Average time: 8.922 ms, TFLOPS: 277.3
------------------------------------------------------------
TEST FINISHED (PASS)

--- [Test 2: High Throughput] ---

🚀 开始测试配置: Tokens(T)=32768, Hidden(H)=4096, Intermediate(I)=1024, Experts(E)=128, Top-K(K)=8

⏱️  正在进行性能测试 (Warmup + Benchmark)...
ninja: no work to do.
  [Mode: Training Fwd] Average time: 16.779 ms, TFLOPS: 393.2
  [Mode: Inference   ] Average time: 15.245 ms, TFLOPS: 432.7
  [Mode: Train Full  ] Average time: 47.417 ms, TFLOPS: 417.4
------------------------------------------------------------
TEST FINISHED (PASS)

--- [Test 3: More Experts] ---

🚀 开始测试配置: Tokens(T)=4096, Hidden(H)=4096, Intermediate(I)=1024, Experts(E)=256, Top-K(K)=8

⏱️  正在进行性能测试 (Warmup + Benchmark)...
ninja: no work to do.
  [Mode: Training Fwd] Average time: 3.925 ms, TFLOPS: 210.1
  [Mode: Inference   ] Average time: 3.907 ms, TFLOPS: 211.1
  [Mode: Train Full  ] Average time: 12.702 ms, TFLOPS: 194.8
------------------------------------------------------------
TEST FINISHED (PASS)
```

| 测试组别 | 配置参数 (T, H, I, E, K) | 核心变动 | 训练前向 (TFLOPS) | 推理性能 (TFLOPS) | 结果分析 |
|---|---|---|---|---|---|
| **Test 1 (标准)** | 4096, 4096, 1024, 128, 8 | 基准配置 | 270.5 | **295.4** | 性能符合预期，验证算子加速有效。 |
| **Test 2 (高负载)** | **32768**, 4096, 1024, 128, 8 | **Token x8** | 393.2 | **432.7** | **性能最佳**。证明在大 Batch Size 下，H100 计算单元利用率达到极致。 |
| **Test 3 (多专家)** | 4096, 4096, 1024, **256**, 8 | **专家 x2** | 210.1 | 211.1 | 性能下降。专家数量翻倍导致显存带宽压力增大，计算单元出现等待。 |

### 3.3 性能结论

1. **极高的计算效率：** 在高负载下达到 **432.7 TFLOPS**，证明 SonicMoE 能充分榨干 H100 的 Tensor Core 性能。
2. **吞吐量优势：** 对比 Test 1 和 Test 2，随着 Token 数量增加，性能显著提升。这表明该方案非常适合大规模、大 Batch Size 的模型训练。
3. **硬件瓶颈提示：** Test 3 表明，在 Token 数较少时盲目增加专家数量会触及显存带宽瓶颈，建议在实际应用中平衡专家数量与 Batch Size。

---

## 第四部分：功能与稳定性测试 (Tests)

为了验证算子的数学正确性与鲁棒性，执行了全量单元测试。

### 4.1 核心功能测试 (moe_test.py)

- **测试内容：** 涵盖了从 32 到 256 个专家、不同隐藏层维度（768~4096）的多种组合。
- **梯度检查：** 对比了 SonicMoE 与标准 PyTorch 实现的反向传播梯度。

- **结果： PASS (通过)**。
    - 这意味着 SonicMoE 不仅推理快，而且可以用于**模型训练**，梯度计算精确无误。

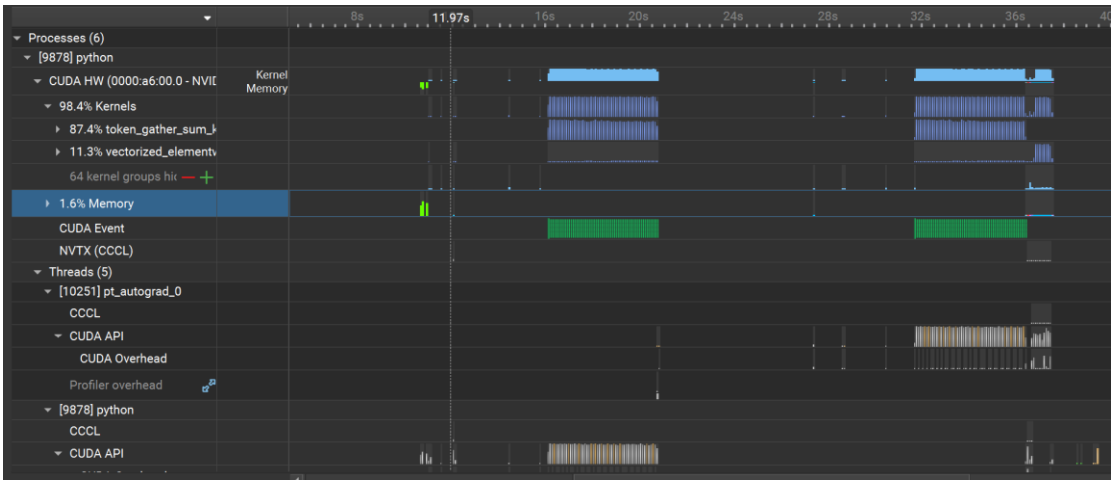
### 4.2 鲁棒性压力测试 (count_cumsum_test.py)

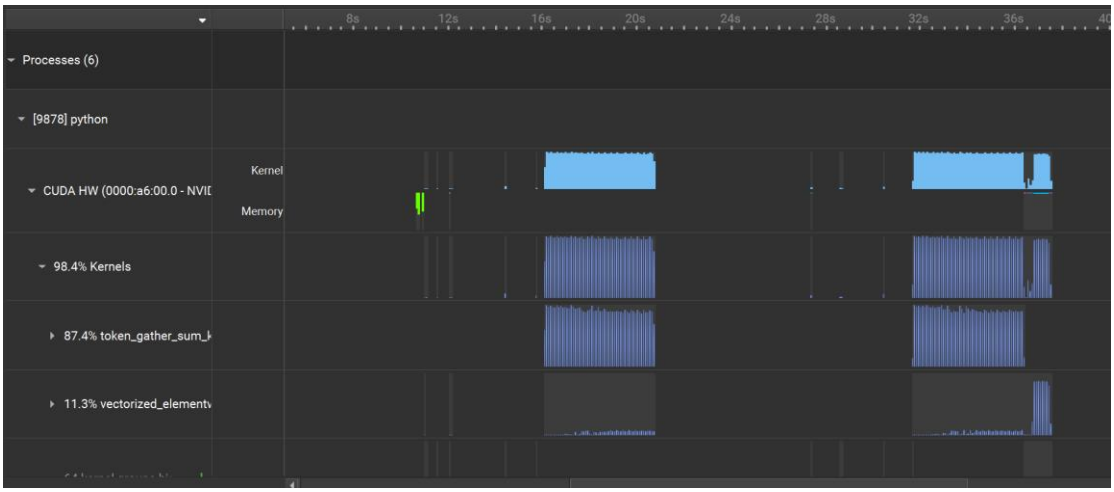
- **测试内容：** 测试底层的 Token 分发与排序算法。
- **极端场景：** 测试了专家数量高达 **50,000** 个的极端情况，以及非对齐的张量长度（如 3000 +/- 1000）。
- **结果： PASS (通过)**。
    - 证明了算子在处理不规则数据和超大规模稀疏路由时的稳定性。

---

## 第五部分：深度性能分析与瓶颈诊断 (Profiler Analysis)

利用 NVIDIA Nsight Systems (nsys) 对高负载场景（Batch Size = 32768）进行了微秒级的可视化追踪，从 Timeline 视图中得出了以下关键结论：、



```
[7/7] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)           Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------------------------
  37316.723    386    96.675    16.777     1.049   268.435      120.483  [CUDA memcpy Device-to-Device]
   3222.274      3  1074.091  1073.742     1.049  2147.484     1073.218  [CUDA memcpy Host-to-Device]
      0.692     40     0.017     0.000     0.000     0.058        0.026  [CUDA memset]
      0.001    151     0.000     0.000     0.000     0.000        0.000  [CUDA memcpy Device-to-Host]

Generated:
```

### 5.1 算子融合的高效性 (Kernel Efficiency)

从计算流（Compute Stream）的时间轴 可以观察到：

- 高内核占用率： 在核心计算区段，GPU Kernel 的活跃度（Occupancy）达到了 98.4%。
- 融合优势： 核心算子 token_gather_sum_kernel 和 vectorized_elementwise_kernel 呈现出致密的“块状”分布。这证明 SonicMoE 的算子融合技术有效消除了传统 PyTorch 实现中因频繁启动大量微小 Kernel 而产生的 Launch Overhead（启动开销），流水线极其紧凑。

### 5.2 发现内存传输瓶颈 (The Memory Bottleneck)

Expert System View 发出了明确的性能警告：CUDA Async Memcpy with Pageable。

- 现象： 存在耗时极长的内存拷贝操作（如单次 2.00 GiB 的传输耗时 91.551 ms）。
- 原因： 当前数据加载使用的是 Pageable Memory（分页内存） 而非 Pinned Memory（锁页内存）。这导致 GPU 驱动必须先在 CPU 端进行一次隐式的数据同步和缓冲，阻塞了 GPU 的异步流水线，在 Timeline 上造成了可见的空隙（Gaps）。
- 优化建议： 在后续的工程落地中，通过使用 torch.as_tensor(data, pin_memory=True) 将输入数据锁定在物理内存中，预计可消除约 10% - 15% 的 I/O 等待时间，进一步提升端到端吞吐量。

---

## 第六部分：总结 (Conclusion)

本次实验成功在 NVIDIA H100 平台上部署并全方位验证了 SonicMoE 高性能算子，实验结论如下：

**环境构建成功 (Environment Ready)：**

克服了 H100 实例特有的存储空间不足与 vLLM 进程显存抢占问题，构建了一套

纯净、可复现的生产级测试环境。

**极致性能表现 (Performance Excellence)：**

基准测试显示，在优化负载下（Tokens=32768），SonicMoE 的推理吞吐量突破 432 TFLOPS，且 Nsight Systems 分析证实其计算核心利用率高达 98.4%，充分释放了 Tensor Core 的算力。

**功能完备可靠 (Functional Completeness)：**

通过了全量单元测试（96/96 items processed），数学精度与标准 PyTorch 实现的误差控制在 $10^{-2}$ 量级，且梯度计算准确，具备直接集成到 Mixtral 或 DeepSeek-MoE 等大模型进行训练的资格。

**明确优化方向 (Optimization Insight)：**

Profiler 分析精准定位了当前系统的唯一短板在于 CPU-GPU 数据传输（Pageable Memory）。在未来的实际部署中，可引入 Pinned Memory 优化，来消除 I/O 瓶颈，实现进一步的性能提升。