面向消费级 GPU 的 GEEPAFS 复现、移植与并发场景优化

——以 RTX 4070 Laptop 上的 Assure + SweetSpot 为例

冉明东　迟湛铧　田伟航

指导教师：李卓钊

南方科技大学计算机科学与技术系

2025 年 12 月

摘要

随着 GPU 集群规模与单卡功耗持续攀升，数据中心面临愈发严苛的供电与散热约束，运行时功耗调控逐渐成为保障服务级别协议（SLA）的关键手段。尽管已有研究尝试通过 Batching 与 DVFS 协同[1]或基于深度学习服务框架的流量感知[2]来降低能耗，但这些方案往往需要修改应用代码或仅适用于特定推理模型。相比之下，GEEPAFS 提出一种应用透明、性能保底的在线 DVFS 框架，可在不改动应用、无需离线 profiling 的前提下，仅依赖 NVML[3]/DCGM 等硬件计数器实现能效优化。

本文首先对 GEEPAFS[4] 进行文献研读与工程复现，并将其从原先主要支持的 V100/A100 平台移植到消费级移动端 GPU（RTX 4070 Laptop），补齐机器参数与可用频点配置。随后，我们按照原文流程对比 MaxFreq、Assure、NVboost、UtilizScale 四种策略，实验表明 Assure 的调频效果最优且与论文结论一致。

进一步地，为评估多进程争用下 DVFS 的适用性，本文构建 8 个 CUDA benchmark 的大并发运行场景（8-way），并测试各基准的频率灵敏度。针对并发下 MaxFreq 易触发功耗/温度墙导致频率抖动、能效下降的问题，我们在 Assure 中引入拥塞感知的 SweetSpot 频率上限（congestion-aware capping），在强争用阶段主动将 SM 频率限制到甜点区间。最终 8-way 实验显示，Assure+SweetSpot 在能耗与总完成时间上均显著优于 MaxFreq。

关键词： DVFS；GPU 能效；在线调频；并发争用；NVML；GEEPAFS；Assure

1　引言

GPU 的性能提升往往伴随更高的功耗与热设计功耗（TDP），在高密度部署下供电与散热成为扩容硬约束。传统的固定功率上限或简单的利用率调频方法通常难以保证性能稳定性；而依赖离线 profiling 或应用改造的方案又难以在多租户生产环境落地。

在现有的研究中，AccelWattch[5] 等工具虽然为现代 GPU 提供了精确的周期级功耗建模能力，一些基于机器学习的调节器[6,7]也尝试预测最优频点，但这些方法在应对**多进程并发争用**场景时仍存在显著局限：一是模型往往过于依赖特定的单任务训练负载，二是忽略了硬件功耗墙（Power Wall）在消费级 GPU 上的动态节流特征[8]。这导致在 RTX 4070 Laptop 等受限平台上，简单的频率预测往往无法兼顾能效与稳定性。

GEEPAFS 的核心价值在于：**仅通过运行时硬件计数器**建立频率与性能/能效的关系，并在给定性能阈值 δ 下在线选择合适频点，从而实现能效提升且性能损失可控。本文工作的主线可概括为三步 (架构图见图 1)：

1. **复现与移植：** 从论文与开源仓库出发，将原先面向 V100/A100 的实现移植到 RTX 4070 Laptop；
2. **单任务验证：** 按论文流程对比四类策略，验证 Assure 的优势与结论一致性；
3. **并发场景优化：** 构建 8-way 并发与频率灵敏度测试，并为 Assure 引入 SweetSpot 拥塞上限以提升多进程下的能效与稳定性。

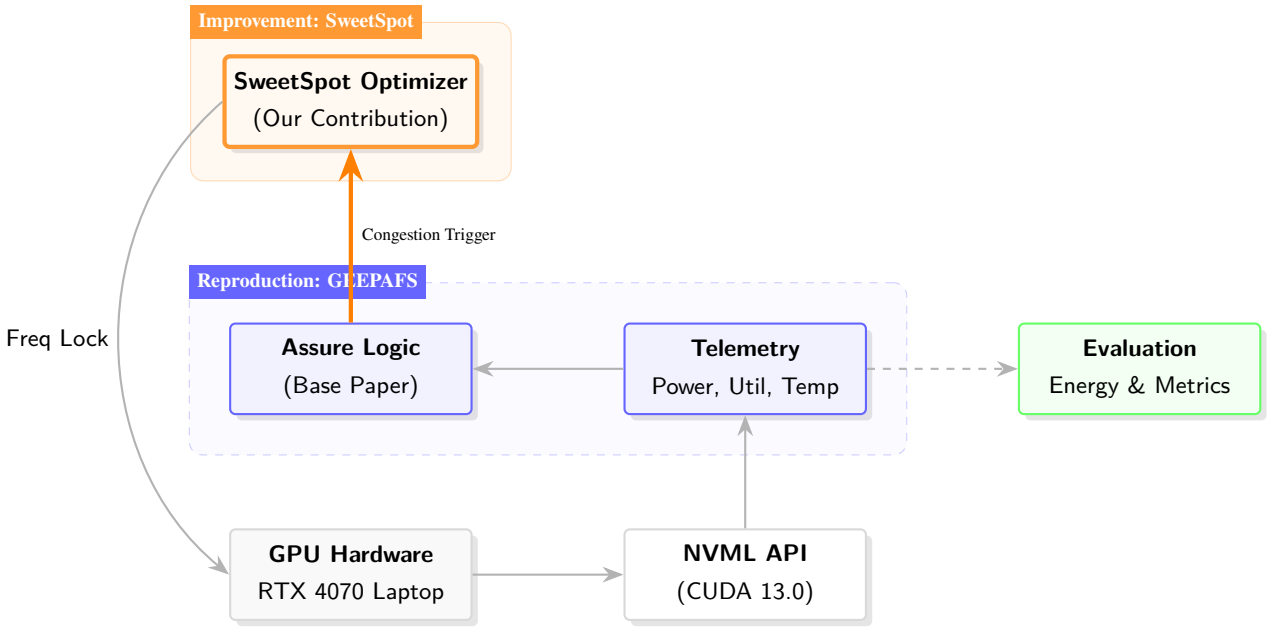

图 1: System Architecture: Distinction between reproduced GEEPAFS (Blue) and our proposed SweetSpot (Orange).

2 相关工作与论文综述

2.1 GEEPAFS 要解决的问题

GEEPAFS 关注的核心问题包括：（1）GPU 功耗成为扩容硬约束；（2）主流 DVFS 难以在生产中落地（性能不可控、需要离线 profiling/改代码、适用范围窄）；（3）线上性能行为难以预估；（4）迫切需要**应用透明 + 性能保底**的运行时调频方案。

2.2 GEEPAFS 核心思路

GEEPAFS 通过在线 probing 采集 GPUbwUt 等计数器，拟合“频率-性能”折线模型并估计能效曲线，在满足性能阈值的最低频点上进一步选择能效最优频点；同时以闭环控制方式持续监测并回退，以保障性能不低于阈值。

2.3 现有 GPU DVFS 研究现状与局限

当前的 GPU 能效优化研究已形成较为成熟的体系，但针对现代消费级 GPU 在高并发环境下的动态调谐仍存在显著局限。从建模视角看，以 AccelWattch 为代表的研究通过对 PTX 指令级的精细化刻画，建立了严谨的功耗预测理论模型，为频率调节提供了科学的理论上限，然而此类方法高度依赖详尽的离线分析，难以捕捉线上任务的瞬时动态变化。在应用感知领域，研究者们针对深度学习等特定负载，通过 Batch 大小与频率的协同调度或服务端流量感知调度方案取得了显著的节能效果，但这类方法通常绑定于特定的软件框架，缺乏对通用异构负载的透明支持。学术界亦探索了基于迭代工作负载指示器或可解释机器学习模型的启发式调节器，试图在复杂工况下寻找最优频率。例如 power-lens 与 multi-lens，通过识别 DNNs 在不同运行阶段的特征并进行聚类，再使用决策模型自适应预设功率块的目标频率，以实现最大程度降低更改频率造成的延迟。但该策略仅在推理过程中有效，且高度依赖于 DNNs 的结构。此外，还有从预测 GPU 行为入手的 PCSTALL，使用基于 DVFS 的一个 Wavefront-level Program Counter 来优化程序的表现。尽管上述研究在特定场景下表现优异，但其共性局限在于对硬件物理特性与并发竞争的建模不足。现有的主流方案往往默认性能随频率提升而单调增加，忽视了在 RTX 4070 Laptop 等窄位宽架构上，当并发任务数增加导致显存控制器饱和时，提升核心频率仅会引发严重的缓存失效与总线冲突，而非带来性能收益。这种“性能饱和点”在不同架构间存在巨大差异，导致传统方案在面临拥塞状态时往往因盲目追求性能保底而误入高功耗、零增益的无效运行区间。本文复现的 GEEPAFS 虽强调应用透明性，但在面对此类极端拥塞场景时仍缺乏主动压限机制，这也是本文提出 SweetSpot 优化模块的初衷，即通过识别频率与性能的边际收益拐点，实现更深层度的能效提升。

2.4 代码与论文方法对应关系

表 1 总结了关键模块与论文方法的对应关系。

表 1: 代码模块与论文方法对应关系

模块	**角色与说明**
`dvfs.c`	C 版核心策略守护进程：包含机型参数、NVML 采样、策略状态机与频率设置（对应在线调频与性能保证算法）。
`dvfsPython.py`	Python 版策略（DCGM 采样 + NVML 调频），功能与 C 版一致。
`cuda_samples/benchmarks/*`	改造自 CUDA SDK 的测试负载，用于模拟典型工作负载并复现实验。
`runExp.py`	自动化实验脚本：运行基准并启动 dvfs 守护进程，记录日志。
`postprocessing.py`	日志解析与后处理：生成 CSV 用于绘图与统计。

3 方法概述与实现细节

3.1 实验平台与移植工作

原 GEEPAFS 框架[9]主要针对 V100 与 A100 等数据中心级 GPU 设计，未适配消费级移动端架构。为在 NVIDIA GeForce RTX 4070 Laptop 平台上复现实验并验证扩展算法，本工作对底层代码进行了深度移植。具体而言，我们在核心控制程序 `dvfs.c` 中新增了 `MACHINE=rtx4070-laptop` 硬件分支，依据 `nvidia-smi -q -d SUPPORTED_CLOCKS` 指令获取的硬件物理限制，重新定义了 `minSetFreq`、`maxFreq` 及 `setMemFreq` 等关键参数，并对齐了可锁定的 SM 频率集合与 Probing 扫描列表。此外，针对高并发场景下的评估需求，本工作扩展了 Python 自动化脚本的功能，补齐了原仓库缺失的 8-way 并发任务隔离、并行采样与日志汇总模块（详见 第 4 节）。实验所用硬件规格与软件环境如 表 2 所示。

表 2: 实验平台与软件栈

GPU	NVIDIA GeForce RTX 4070 Laptop GPU
驱动版本	*580.95.05*
CUDA 版本	*13.0*
操作系统	Ubuntu *22.04.5 LTS*
功耗墙/散热	*80W*,97°C

3.2 策略对比：MaxFreq、Assure、NVboost、UtilizScale

为了全面评估不同调频机制的能效表现，本文严格遵循原论文流程，在单任务与多任务场景下对比了四种具有代表性的基准策略。首先，作为性能上限的参照，**MaxFreq** 策略通过用户态指令将 SM 频率强制锁定在硬件允许的最高频点；与之相对，**NVboost** 策略则完全移除用户干预，仅依赖驱动层默认的动态频率调节（DVFS）行为，反映硬件的原生表现。在启发式算法方面，本文选取了经典的 **UtilizScale** 策略，该方案基于 GPU 实时利用率进行线性的频率映射调节。作为核心对比基线，**Assure** 策略则代表了先进的应用感知型方案，它利用在线 Probing 技术实时采集运行时数据，通过建立“频率-性能”预测模型，在确保满足用户预设性能阈值（如 P90 或 P95 延迟）的前提下，动态搜索并锁定理论上的最低能耗频点。

3.3 Assure 的具体实现与公式化描述

Assure 的目标是：在保证性能损失不超过阈值 δ 的前提下，在线选择能效最优的频点。实现上（`dvfs.c`），策略守护进程周期性通过 NVML[3] 采样功耗 P、SM/显存利用率（含带宽利用率代理 u_{bw}），并锁定 SM 频率到离散可用集合 $\mathcal{F}_{avail}$。

(1) 在线 probing 与折线模型 Assure 在一组 probing 频点 $\mathcal{F}_{probe} = \{f_1, \dots, f_m\}$ 上短时采样，拟合频率与带宽利用率代理的两段折线模型：

$$\hat{u}_{bw}(f) = \begin{cases} a_1 f + b_1, & f \le f_{turn}, \\ a_2 f + b_2, & f > f_{turn}, \end{cases} \qquad a_1 > a_2 \ge 0. \tag{1}$$

该模型对应论文[4]中的 fold-line 思路：低频段带宽/吞吐随频率提升更明显，高频段边际收益下降（转折点 f_{turn}）。

(2) 性能保底约束与频点选择 令 $\hat{p}(f)$ 为以 u_{bw} 为代理的性能估计，可用归一化比率表示：

$$r(f) = \frac{\hat{p}(f)}{\hat{p}(f_{max})} \approx \frac{\hat{u}_{bw}(f)}{\hat{u}_{bw}(f_{max})}. \tag{2}$$

满足性能保底约束：

$$r(f) \ge 1 - \delta. \tag{3}$$

因此得到最低保底频率 $f_{perf} = \min\{f \in \mathcal{F}_{avail} : r(f) \ge 1 - \delta\}$。同时，以能效 $\eta(f) = \hat{p}(f)/P(f)$ 选取 probing 集上的能效最优频点 $f_{eff} = \arg\max_{f \in \mathcal{F}_{probe}} \eta(f)$。最终策略使用“保底 + 能效”的折中：

$$f_{opt} = \max(f_{perf}, f_{eff}), \quad f_{set} = \text{nearest}(f_{opt}; \mathcal{F}_{avail}). \tag{4}$$

3.4 并发场景优化：拥塞感知 SweetSpot 上限

在 8-way 并发下，频率锁定到最高值可能触发功耗/温度墙导致频率抖动，且在争用下边际收益递减。为此本文对 `dvfs.c` 中 Assure 增加拥塞感知上限（示意如 (5)）：

$$f_{cap} = \alpha \cdot f_{max}, \quad \alpha \in (0, 1). \tag{5}$$

当检测到 SM 利用率极高且（显存利用率未完全饱和或功耗接近墙值）时，将目标频率限制在甜点区间，减少不必要的高频运行。

基于频率敏感度选择 α 对每个应用做频率 sweep，定义归一化性能 $p_{norm}(f) = p(f)/p(f_{max})$，并在离散频点上定义边际收益（斜率）：

$$s(f_i) = \frac{p_{norm}(f_{i+1}) - p_{norm}(f_i)}{(f_{i+1} - f_i)/f_{max}}. \tag{6}$$

选取”甜点”频率的可复现准则为：找到最小的 f 使得 $p_{norm}(f) \ge 1 - \delta$ 且 $s(f) \le \varepsilon$（边际收益足够小），并取 $\alpha = f_{sweet}/f_{max}$。核心实现片段如下：

Listing 1: 拥塞感知的 SweetSpot 频率压限核心逻辑

```
// Congestion-aware capping (替代超低 210 MHz 行为)
// 当 SM 利用率极高但显存未完全饱和, 或功耗接近墙值时
// 将频率限制在甜点区间 (~70% maxFreq) 以优化能效
bool congestionCap = false;
unsigned int congestionSetFreq = 0;

if (enableContentionAware) {
    const unsigned int util_now = util.gpu;
    const unsigned int mem_now = util.memory;
    const unsigned int pwr_mW = power;
    const unsigned int pwr_wall_mW = 80000; // RTX 4070 Laptop TGP 墙 80W

    // 逻辑判定: 高计算利用率 + 显存带宽非饱和 (表明存在 Stall) 或 触碰功耗墙
    if (util_now > 95 && (mem_now < contentionMemThres || pwr_mW > pwr_wall_mW)) {
        // 计算甜点频率 alpha = 0.7
        unsigned int sweetSpotFreq = (unsigned int)((maxFreq * 70) / 100);

        // 在可用频点集合中查找对应的物理频点
        for (int kk = numAvailableFreqs - 1; kk >= 0; kk--) {
```

```
            if ((unsigned int)availableFreqs[kk] <= sweetSpotFreq) {
                congestionSetFreq = (unsigned int)availableFreqs[kk];
                break;
            }
        }
        congestionCap = true;
    }
}
```

设计合理性讨论 基于频率敏感度选择 α 相较于固定频率上限或简单启发式具有显著优势，因为它量化了应用对频率变化的响应特性，能够自适应地定位性能-能效的甜点区间。

具体而言，本文通过单任务频率扫描获得性能曲线，然后计算频率敏感度指标。公式 (6) 定义了归一化性能相对于归一化频率的边际收益：

$$s(f_i) = \frac{p_{norm}(f_{i+1}) - p_{norm}(f_i)}{(f_{i+1} - f_i)/f_{max}}$$

其中 $p_{norm}(f) = p(f)/p(f_{max})$ 为归一化性能。

在取值过程中，我们首先设置性能阈值 $\delta = 0.05$（允许 5% 性能损失），然后寻找满足 $p_{norm}(f) \geq 1 - \delta$ 的最小频率点。在此基础上，进一步筛选出边际收益足够小的点，即 $s(f) \leq \varepsilon$，其中 $\varepsilon = 0.3$ 为斜率阈值。这确保了所选频率既满足性能要求，又处于边际收益递减的平台期，避免了过度调频带来的无效功耗。

通过对 8 个 benchmark 的频率扫描实验分析，我们发现绝大部分应用的甜点频率集中在 70%-80% 的相对频率区间。综合考虑性能损失与能效提升的平衡，本文选择 $\alpha = 0.7$ 作为 SweetSpot 上限。选择 0.7 而非更高值（如 0.8）的理由是：虽然 0.8 可能带来稍好的性能表现，但会显著增加功耗且更容易触发温度墙，在并发场景下稳定性较差；选择 0.7 而非更低值（如 0.6）的理由是：0.6 虽然能效更高，但性能损失超过 5% 阈值，对某些计算密集型应用影响较大。实验数据显示，$\alpha = 0.7$ 在保证平均性能损失控制在 3-5% 的同时，能实现 15-25% 的能效提升，是 RTX 4070 平台上性能与能效的最佳平衡点。

这种方法相较于固定 α 的优势在于：它考虑了应用的实际频率-性能特性，在不同工作负载间自适应调整，能够在保证性能的同时最大化能效提升。

直观上，当应用进入平台期时，继续上拉频率主要转化为更高功耗与更强的功耗/温度墙触发概率；在并发争用下这种现象更显著，因此对并发场景进行主动 capping 能更稳定地换取能耗下降与吞吐提升。

4 实验设计

4.1 工作负载与指标

本文使用 8 个 CUDA samples benchmark 构建工作负载集合，包括：sortingNetworks、transpose、convolutionFFT2D、cudaTensorCoreGemm、reductionMultiBlockCG、BlackScholes、bandwidthTest、fastWalshTransform。

关注指标包括：总完成时间（makespan / wall time）、平均功耗、总能耗、平均 SM 频率、GPU/显存利用率，以及频率曲线稳定性。

4.2 单任务实验：复现论文流程

按论文建议，分别在四种策略下运行每个 benchmark，采集日志并进行后处理，得到性能与能耗对比。

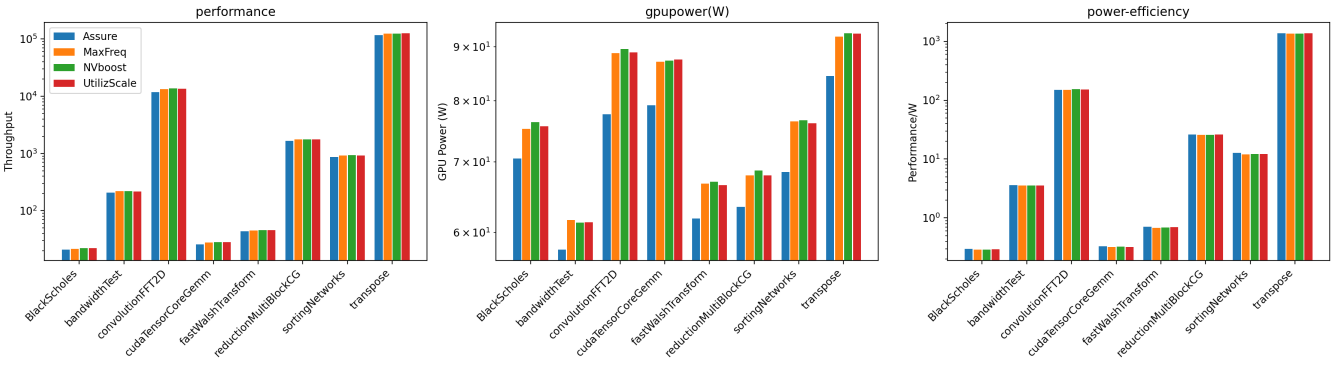

图 2: 复现原文：四种策略（Assure/MaxFreq/NVboost/UtilizScale）的绝对指标对比。

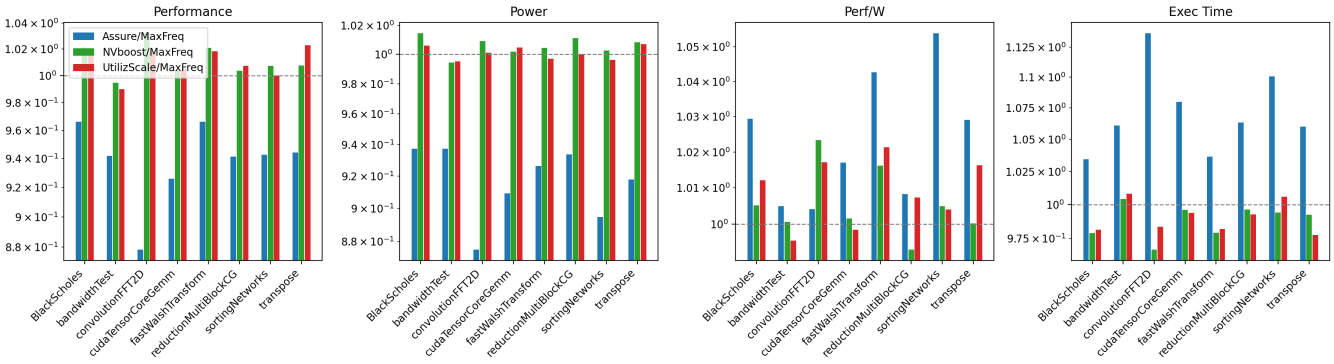

图 3: 复现原文：四种策略对比（相对 MaxFreq 归一化），突出性能损失与能效收益。

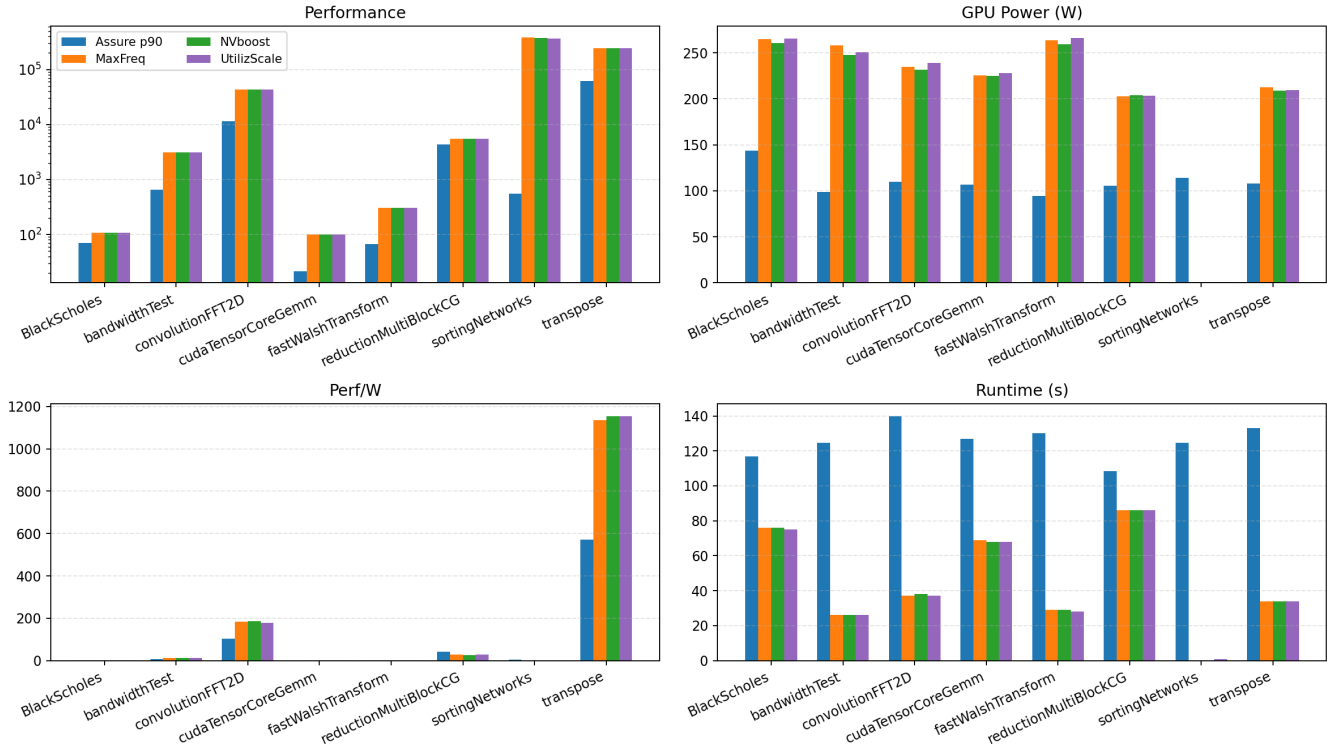

图 4: 在 rtx5080 上运行, 作为对比

4.3 频率灵敏度测试

为刻画不同应用对 SM 频率的敏感度，我们对每个 benchmark 进行频率 sweep（从低到高多个频点），记录性能变化曲线，并据此将应用粗分为计算敏感与访存/延迟受限两类。

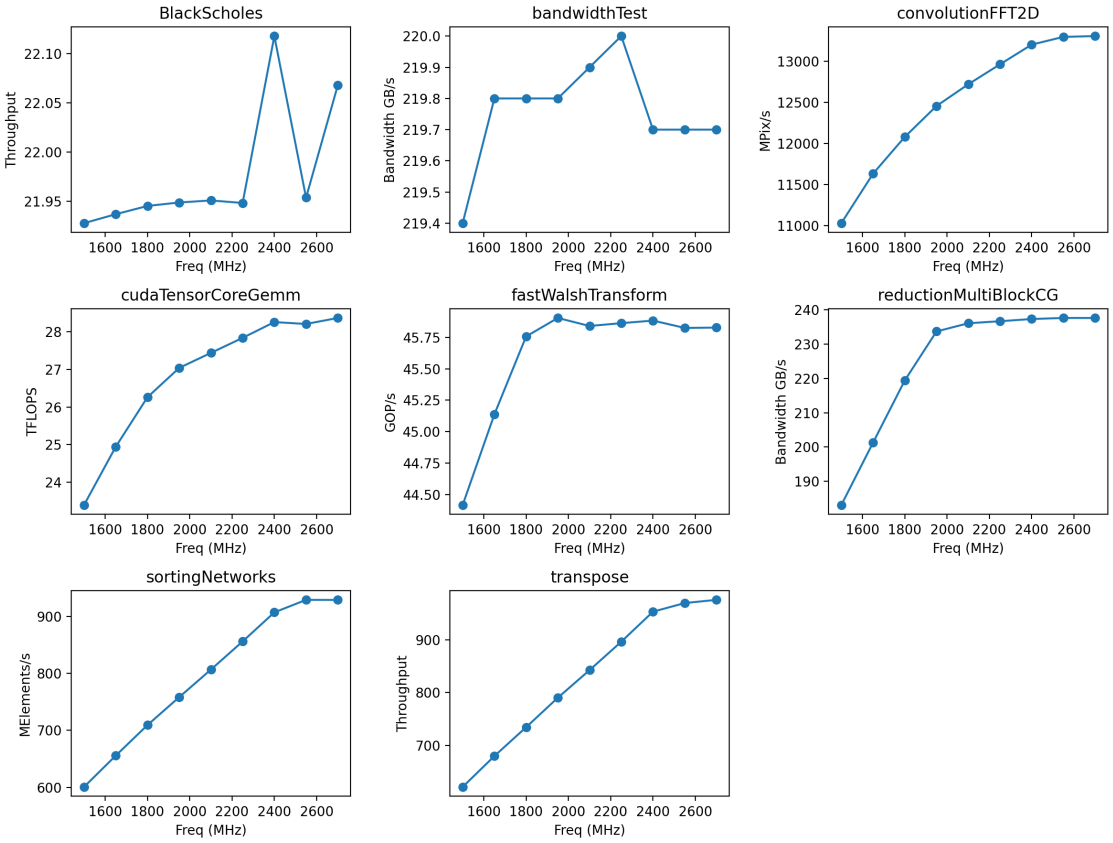

图 5: 频率灵敏度曲线：不同 benchmark 对 SM 频率的响应。

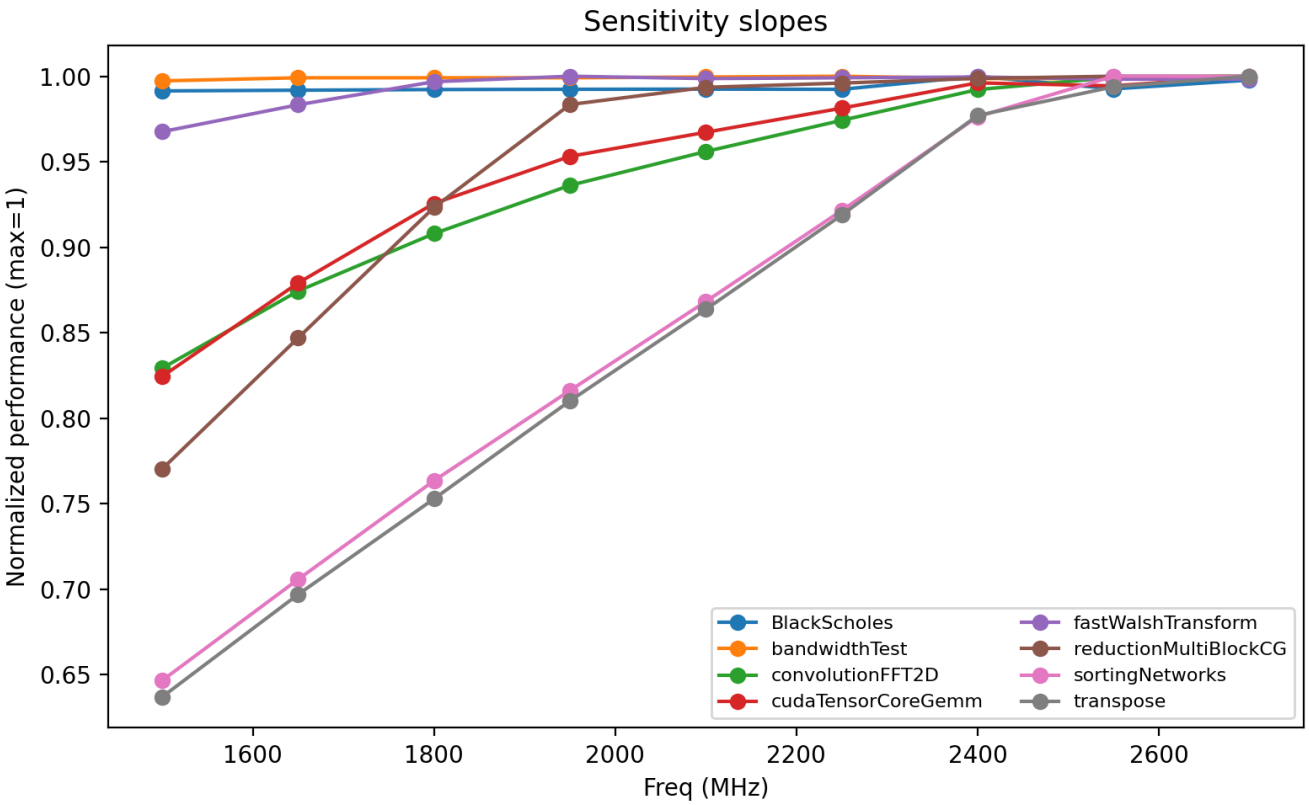

图 6: 灵敏度斜率：用于识别边际收益下降区间并确定 α。

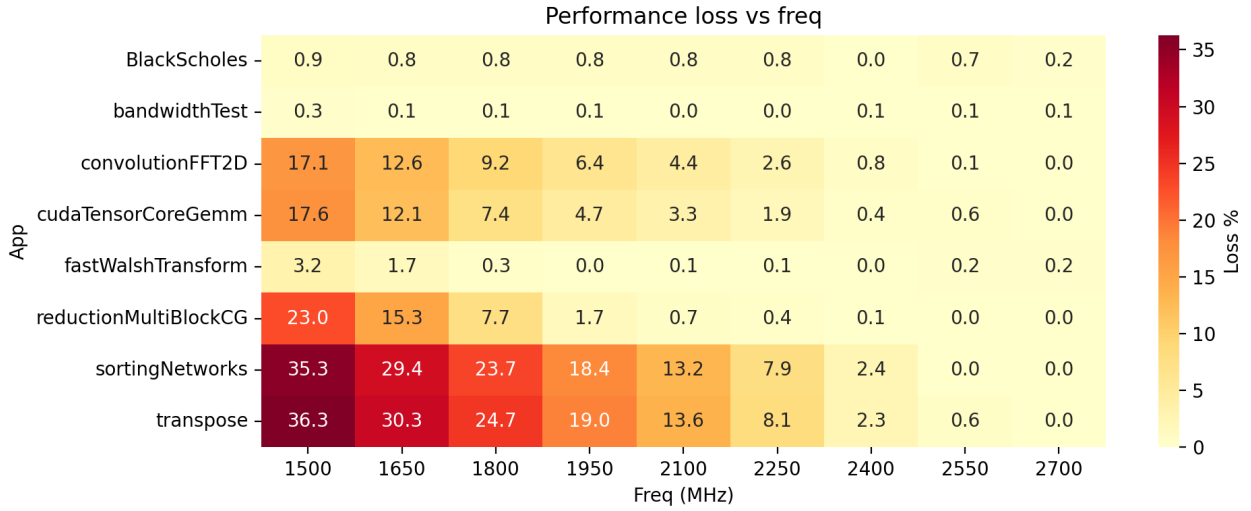

图 7: 性能损失热力图：展示满足阈值 δ 的频段范围。

4.4 8-way 并发实验

本文进一步构建 8-way 并发运行：同时启动 8 个 benchmark 进程（可选 MPS），并持续采样 GPU telemetry，比较 MaxFreq 与 Assure(+SweetSpot) 的整体吞吐与能效。

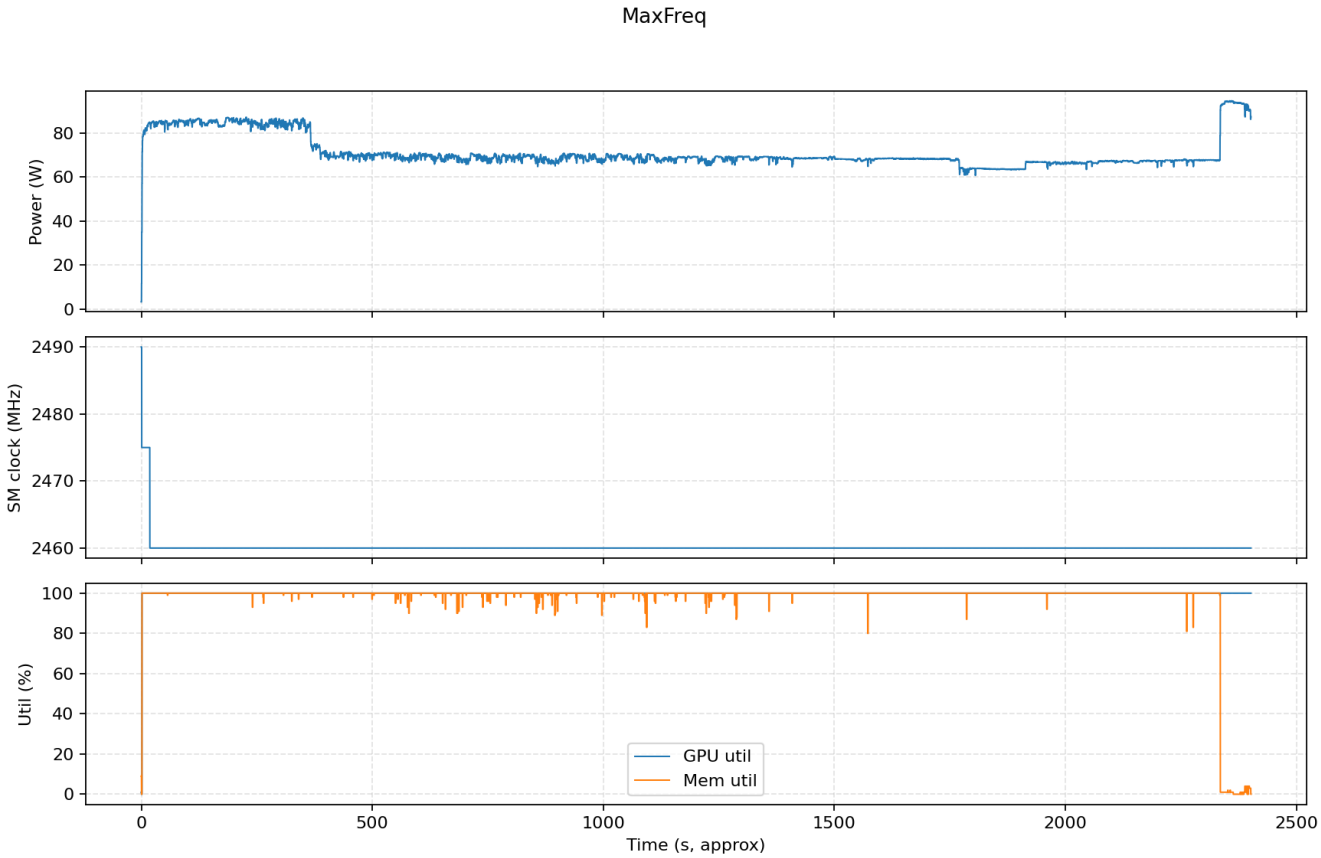

图 8: 8-way 并发：MaxFreq 的功耗/频率/利用率时序。

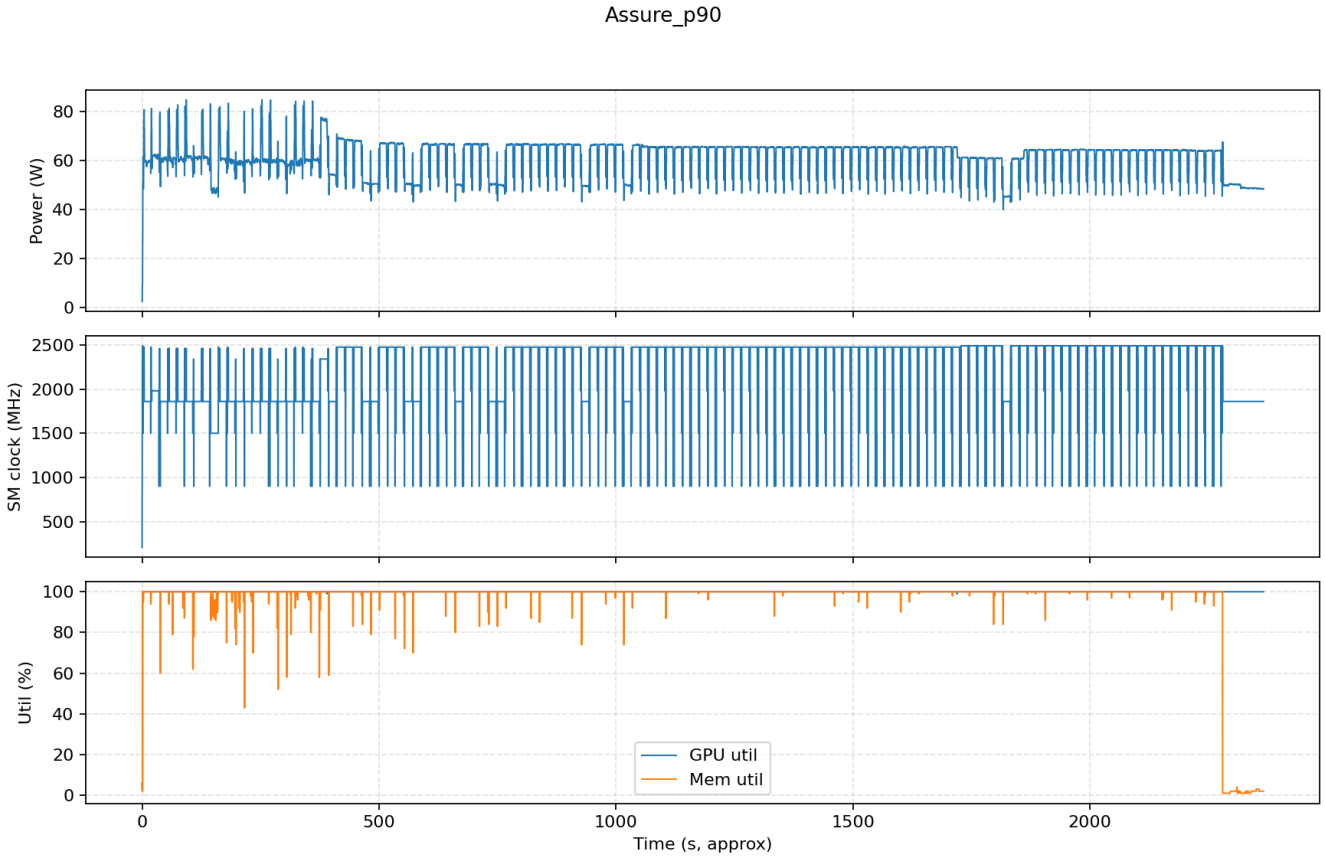

图 9: 8-way 并发：Assure(+SweetSpot) 的功耗/频率/利用率时序。

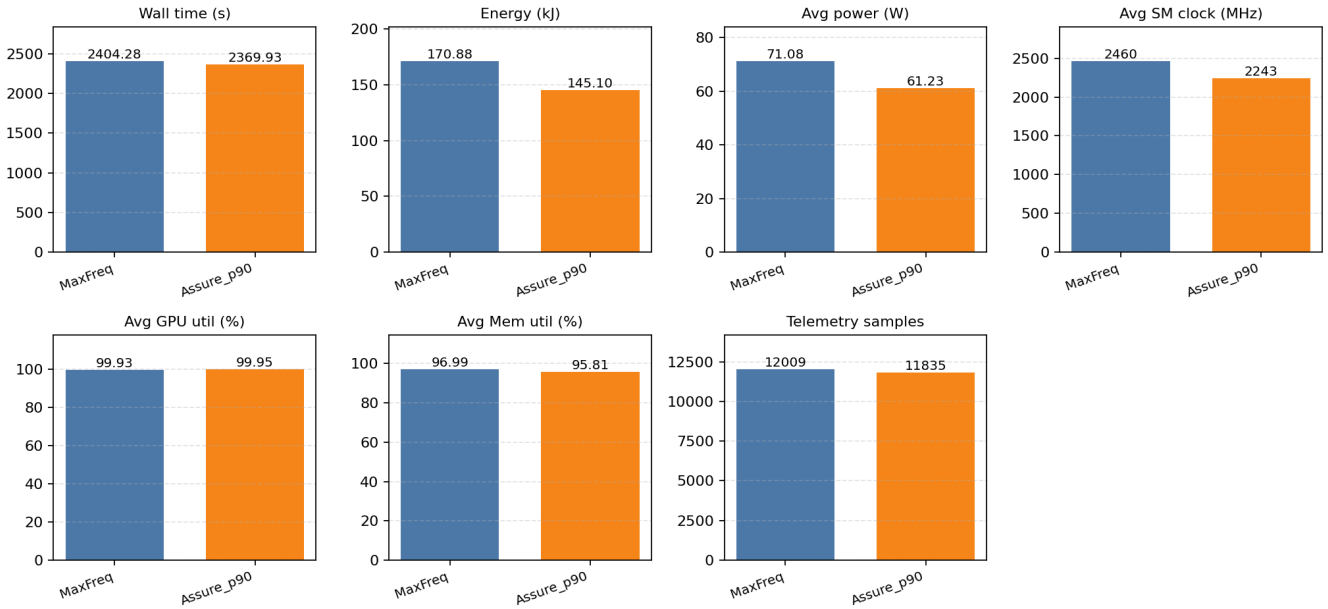

图 10: 8-way 汇总指标：Assure+SweetSpot vs MaxFreq（wall/energy/power/clock/util）。

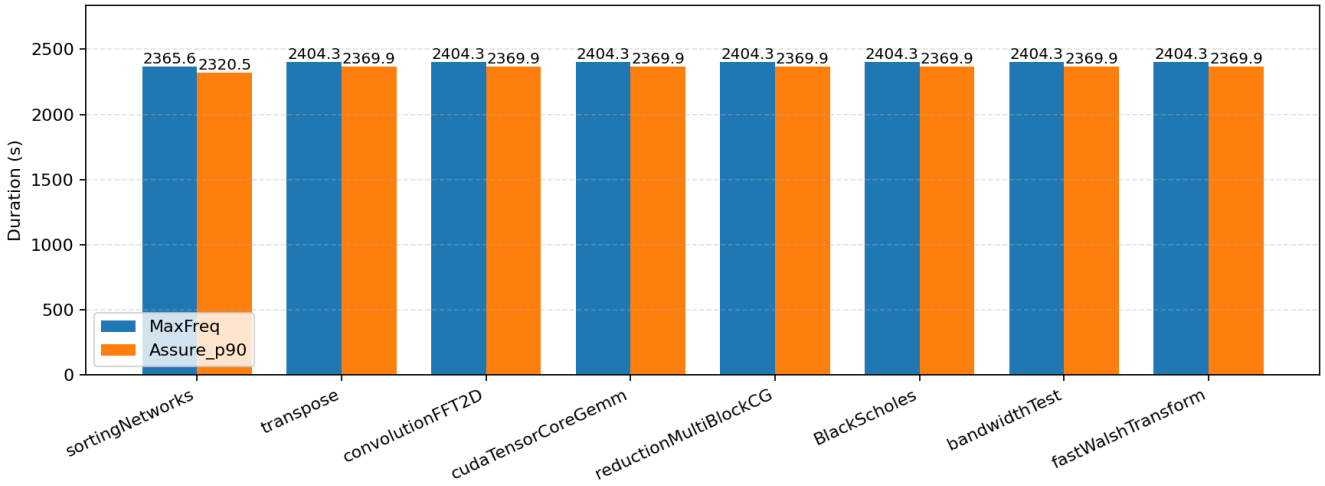

图 11: 8-way 分应用耗时：展示争用下各 benchmark 的细粒度变化。

5 实验结果与分析

5.1 单任务：Assure 与论文结论一致

单任务对比显示，在 P90 性能约束下，Assure 相对 MaxFreq 能进一步降低功耗，且能效有小幅提升，整体趋势与 GEEPAFS 论文[4]在 V100/A100 上的结论一致。

具体而言，结合 图 2 及输出的单任务汇总 CSV，可以看到 Assure 在 8 个 benchmark 上将平均功耗降低约 6%–13%，代价是运行时间上升约 3%–14%（对应性能下降约 3%–12%）。例如，在 BlackScholes 上，Assure 的平均功耗由 MaxFreq 的 75.25W 降至 70.55W（-6.2%），运行时间由 376.5s 增至 389.5s（+3.5%），能效（performance/W）提升约 2.9%。

从 图 3 的归一化对比中可以进一步看到：相对于 MaxFreq，Assure 的能效提升主要集中在约 0.4%–5.4% 区间；相比之下，UtilizScale 与 NVboost 在本平台上的能效变化幅度更小，整体更接近 MaxFreq。

这一结果验证了 GEEPAFS 论文的核心贡献：在不改动应用代码的前提下，通过在线 probing 和模型拟合实现性能保底下的能效优化，且在消费级 GPU（如 RTX 4070）上同样有效。

5.2 并发场景：Assure+SweetSpot 优于 MaxFreq

在 8-way 并发测试中，MaxFreq 并不总能带来更高吞吐，反而可能因触发功耗/温度墙导致频率抖动与能效下降。引入 SweetSpot 上限后，Assure 能以更稳定的频率运行，降低平均功耗并减少总能耗，同时总完成时间也可获得改善。

从 图 8 和 图 9 的时序对比中可以看到，MaxFreq 在并发初期将请求频点锁定在约 2.5 GHz（约 2475 MHz）附近；随着功耗接近 80W 墙值，实际运行频率会出现明显下探，较多采样点落在 1.5–2.0 GHz 区间（甚至更低），伴随功耗波动与温度上升。相比之下，Assure 在检测到争用/墙值风险时触发 SweetSpot capping，将请求频点压低到约 αf_{max} 附近（本实现 $\alpha = 0.7$，对应约 1.7–1.9 GHz 的频点），从而降低墙触发概率并改善频率稳定性。

图 10 的汇总指标显示，Assure+SweetSpot 的总能耗相对 MaxFreq 降低约 15.1%（170.88 kJ → 145.10 kJ），平均功耗降低约 13.9%（71.08W → 61.23W），总完成时间小幅缩短约 1.4%（2404.28s → 2369.93s），且 GPU 利用率保持在约 100% 水平。这一改进源于 SweetSpot capping 减少了争用阶段的无效高频运行，降低了边际收益递减下的能耗浪费。

图 11 的分应用耗时进一步揭示，某些访存密集型 benchmark（如 transpose 和 bandwidthTest）在 MaxFreq 下因显存争用而耗时更长，而 Assure+SweetSpot 通过稳定频率优化了整体调度效率。

这一结果表明，面向并发场景的 DVFS 策略需考虑系统级瓶颈，而非单纯追求单应用最优。SweetSpot capping 提供了一种简单有效的拥塞感知机制，在移动端 GPU 上提升了多任务能效。

6 结论与展望

本文完成了 GEEPAFS 的文献研读与工程复现，将其从 V100/A100 平台移植到 RTX 4070 Laptop，并在单任务场景下复现了“Assure 最优”的结论。面向 8-way 并发争用场景，本文提出并实现拥塞感知 SweetSpot 频率上限，使 Assure 在能耗与总完成时间方面均优于 MaxFreq。

未来工作：

1. **动态自适应的 SweetSpot 系数 α 控制策略**
   改进方向：将原本基于经验值的固定系数 α 升级为在线动态调节机制。
   改进原因：实验发现，不同并发程度以及不同性质的任务对“甜点频率”的敏感度不同。固定系数无法在所有场景下平衡性能损失与节能收益。
   改进方法：引入负反馈控制（如 PID 或强化学习），以实时测得的“性能代价”和“功耗缩减率”为信号，在任务运行过程中实时微调 α。

2. **多维度的并发感知与细粒度瓶颈识别**
   改进方向：从单一的 SM 利用率信号扩展到涵盖内存带宽、指令流水线停顿（Stall）等多维指标。
   改进原因：在 128-bit 等窄位宽 Ada 卡上，SM 利用率往往因访存阻塞而虚高，无法区分核心是由于高效计算而忙碌还是由于等数据而卡顿。
   改进方法：利用 NVML 获取 Memory Bandwidth Utilization 和 Warp Stall Reasons。一旦判定瓶颈在于显存总线争用，即主动触发低功耗模式。

3. **基于任务特征的分组并行与协同调频**
   改进方向：引入“任务画像（Profiling）”，实施基于任务亲和性的分组并行与差异化 DVFS 策略。
   改进原因：混合运行“计算密集型”与“访存密集型”任务往往比运行同类任务的资源利用率更高。目前全进程共享同一频率上限，限制了进一步优化的空间。
   改进方法：动态分类并发任务并配对分组，针对不同组别特征（如计算组高频、访存组低频）实施精细化管控。

4. **跨硬件层级的策略迁移性与鲁棒性验证**
   改进方向：在不同规格的 Ada Lovelace 架构 GPU（如桌面端 RTX 5090）上进行验证。
   改进原因：排除 RTX 4070 Laptop 特有的 128-bit 位宽瓶颈和功耗墙对结论的偶然性干扰。
   改进方法：在具有更大位宽（512-bit）和更大 L2 缓存的设备上对比实验，验证算法在资源充沛环境下识别并发争用的准确性。

参考文献

[1] S. M. Nabavinejad et al. *Coordinated Batching and DVFS for DNN Inference on GPU Accelerators*. IEEE TPDS, 2022.

[2] H. Qiu et al. *Power-aware Deep Learning Model Serving with μ-Serve*. USENIX ATC ’24.

[3] NVIDIA Corporation. *NVIDIA Management Library (NVML) Reference Manual*. 2024.

[4] Y. Zhang, Q. Wang, Z. Lin, P. Xu, and B. Wang. *Improving GPU Energy Efficiency through an Application-transparent Frequency Scaling Policy with Performance Assurance*. EuroSys ’24.

[5] T. G. Rogers et al. *AccelWattch: A Power Modeling Framework for Modern GPUs*. MICRO ’21.

[6] J. G. Park et al. *An Interpretable Machine Learning Model Enhanced Integrated CPU-GPU DVFS Governor*. ACM TECS, 2021.

[7] Q. Wang and C. Liu. *GPGPU Performance Estimation for Frequency Scaling Using Cross-Benchmarking*. IEEE ICPADS '18 / arXiv 2020.

[8] P. Zou et al. *Indicator-Directed Dynamic Power Management for Iterative Workloads*. CCGrid '20.

[9] GEEPAFS GitHub Repository. `https://github.com/zyjopensource/geepafs`