Optimizing Performance Under Power Constraints

Ahmet Inci | NeurIPS | December 2nd 2025

Agenda

- **Rethinking fundamental design metrics in the age of energy and power limits**
  - Interplay of performance, power, and energy
  - How power limits translate to application performance
- **How workload behavior affects performance under power**
  - Workload behavior and sustained performance
  - Impact of data distribution on performance
  - How short-timescale vs. long-timescale power limits shape performance
- **Optimizing performance under power constraints**
  - Performance (Max) vs. Energy (Sum)
  - SW optimization considerations
    - Case study: Energy-aware kernel selection
    - Case study: Utilization-only optimizations
  - Practical optimization guidelines
  - Measuring LLM Inference Efficiency in TRT-LLM
    - Why efficiency matter in real-deployments
    - GPU energy profiling in TRT-LLM

System Design: Performance (Ethereal) & Energy (Physical)

Performance (The Goal)

An abstract, emergent metric of **how well** a system functions. It's the conceptual result.

Metrics: Speed (Latency), Throughput

Energy (The Cost)

The **tangible, physical resource cost** of all operations. It's the measurable input as rate of Energy consumption

Metrics: Joules (J), Watts (W)

Efficiency (The Bridge)

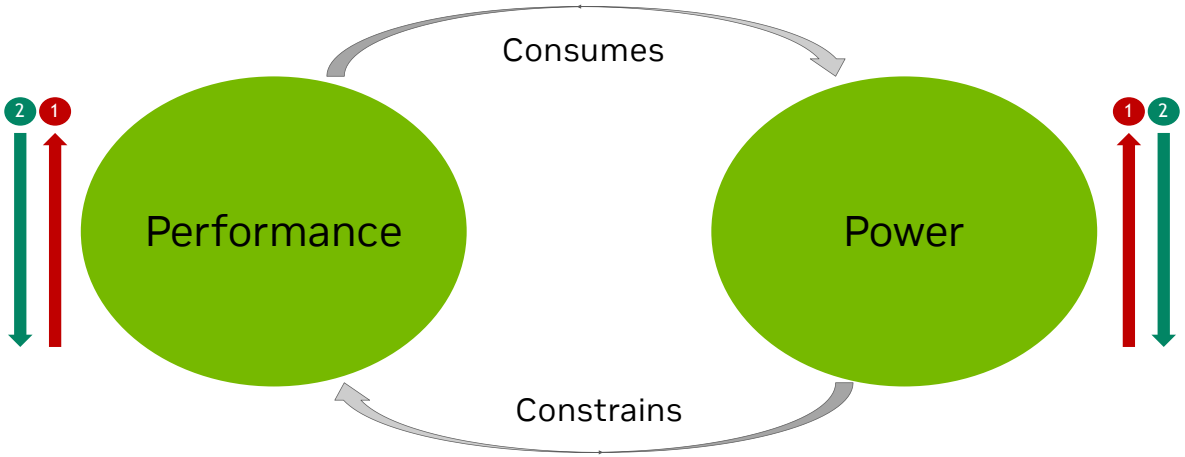
Efficiency

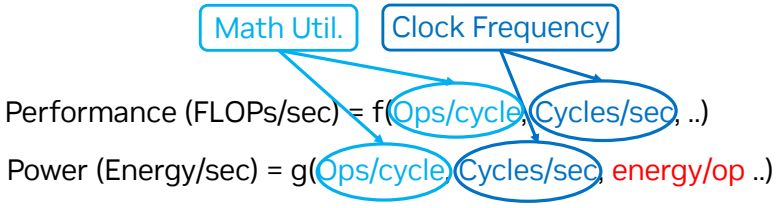
$$= \frac{\text{Performance (Useful Work/sec)}}{\text{Energy/sec Consumed}}$$

Connects the abstract goal to the physical cost. This is the key system **trade-off**.

Performance and Power are Interdependent

Computing consumes energy; rate of computing (performance) consumes power

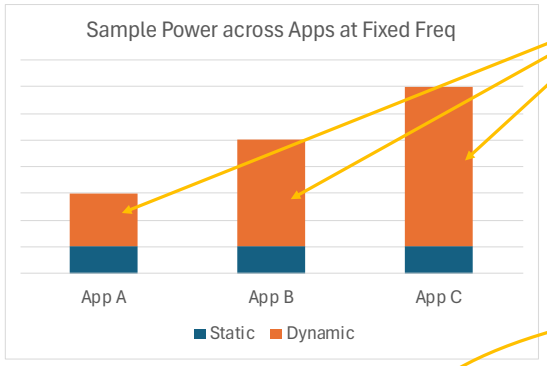




*Clock frequency = How many cycles GPU runs per second

"Constraints" driven by: Power Delivery, Thermal, Reliability, etc.

Performance, Energy and Power



App = Accelerated AI Model

Dynamic power is different due to different resource utilization in GPUs

Dynamic -> App dependent

Static -> App independent

Static Power = Energy cost of running the GPU

Dynamic Power = Dynamic Energy / Execution Time

Dynamic Energy = Energy per {op, byte} * # {ops, bytes}

Consider what happens when we optimize SW to reduce execution time?
Case A: Reduce total work done i.e. # {ops, bytes} (e.g. Kernel fusion)
Case B: Do not reduce work (e.g. Overlap compute & communication)

*Simplified view with fixed frequency, voltage etc.

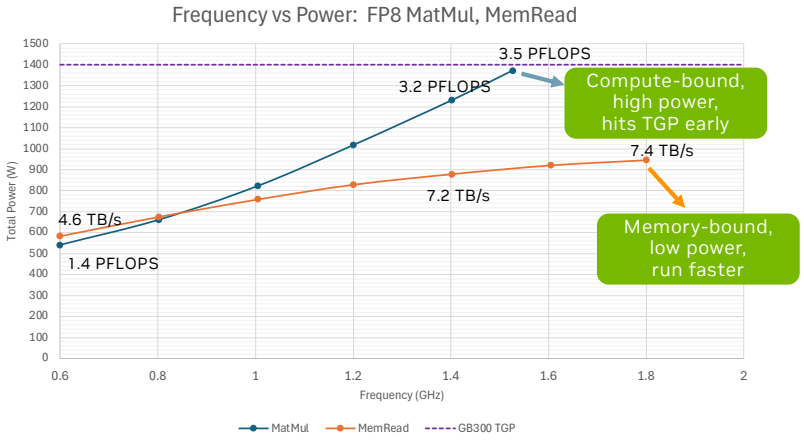
Workload Behavior Under Power Constraints

Workload Characteristics and Sustained Performance

Why different workloads land at different operating frequencies?

- **Power Budget**
  - **TGP (Total Graphics Power)** = total allowed power for GPU
  - Enforced and shared across *all subsystems*
- **Power Behavior**
  - Power draw depends on workload and clock frequency
  - For long-running workloads, average power must stay within TGP
  - GPU may exceed TGP briefly (e.g. up to 1820 W on GB300)
- **Workload Performance**
  - Sustained frequency differs across workloads
  - Dynamic power $P \approx utilization\ x\ frequency$
    - Utilization differs across workloads:
      - MatMul - high tensor core util. -> high power
      - MemRead – low util. -> low power

Different workloads land at different operating points

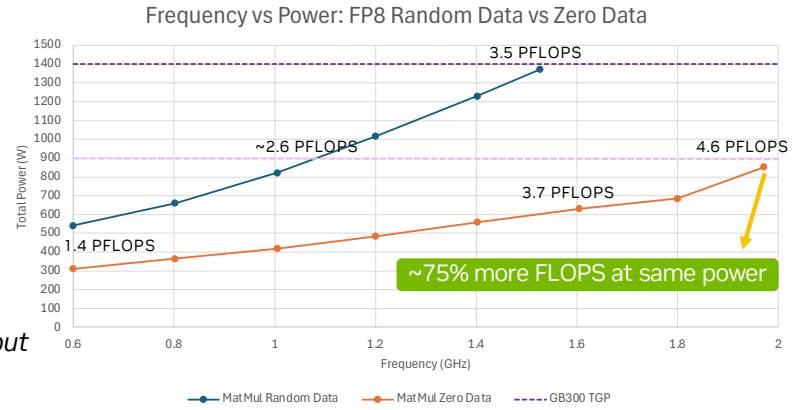


- **Key insights**
  - *Compute-bound workloads throttle earlier to stay within power budget*
  - *Memory-bound workloads run at higher clocks because their low compute utilization can't saturate TGP*
  - *Utilization drives power, and power dictates sustained frequency*

Data Distribution Affects Performance

Input sparsity changes switching activity and sustained performance

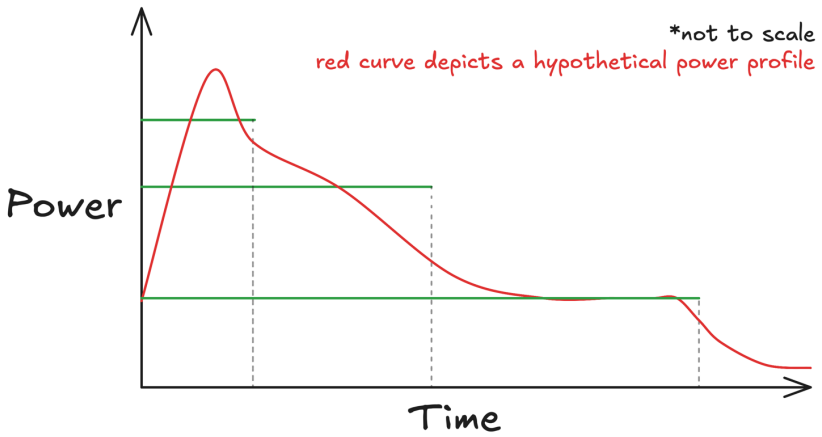
- Even with the same MatMul kernel, data distribution matters
  - Zeros cause fewer bit flips, so dynamic power is lower
  - With random data, switching activity increases, dynamic power rises and GPU throttles frequency to stay within TGP
- **Key insights**
  - *Always profile with representative data distributions*
    - *Zero data underestimates power and overestimates sustained clocks and throughput*



Power Constraints Across Timescales

Short vs. long timescale power limits and how they shape workload performance?

- Power constraints aren't a single number (e.g. GB300 TGP = 1400 W)
  - Power = rate of energy consumption (Joules / sec)
  - Limits are enforced as moving averages, not instantaneous caps
    - Constraints define how fast we're allowed to spend that energy
- Short timescale (Electrical Design Point: 100s of us to 10s of ms)
  - Peak transient power over a short moving average window
- **Key insights**
  - Short timescale limits shape kernel-level performance
  - Compute intensive kernels may momentarily run at higher frequency
    - Brief boost allowed, but frequency is clamped if short-timescale energy budget is exceeded

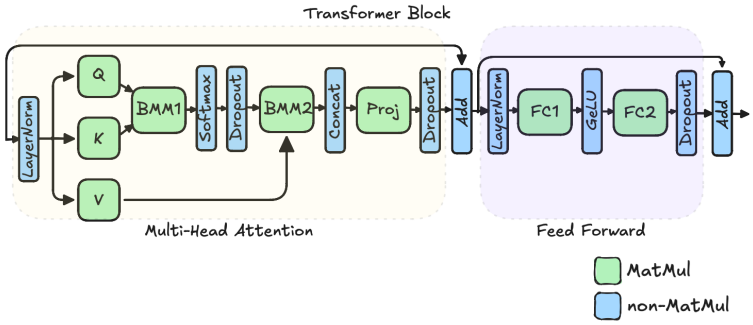


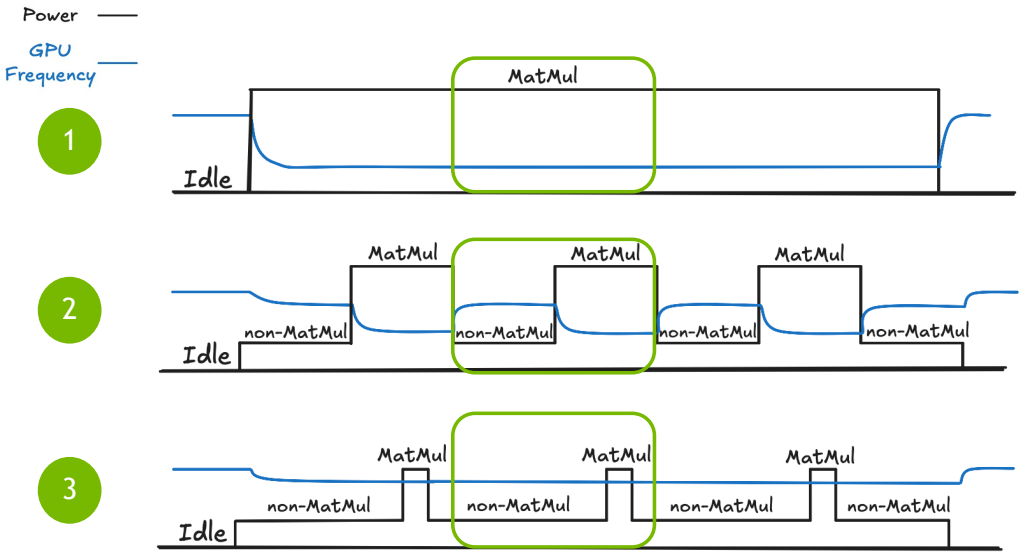
Short timescale limits shape instantaneous kernel behavior;
Long timescale limits determine where workloads settle over time

Workload Transitions and Perf under Power

How GPUs adapt to workload compute-intensity

- GPU power controller operates over a moving average power window
  - GPU adjusts frequency to keep average power ≤ TGP, not instantaneous power
- Short MatMul can draw high power as long as the moving average power ≤ TGP;
  - Power controller doesn't react – GPU sustains its frequency
- Each transformer block cycles through these high-power (MatMul) and low-power (non-MatMul) phases
  - End-to-end performance is shaped by workload composition, not individual kernel clocks





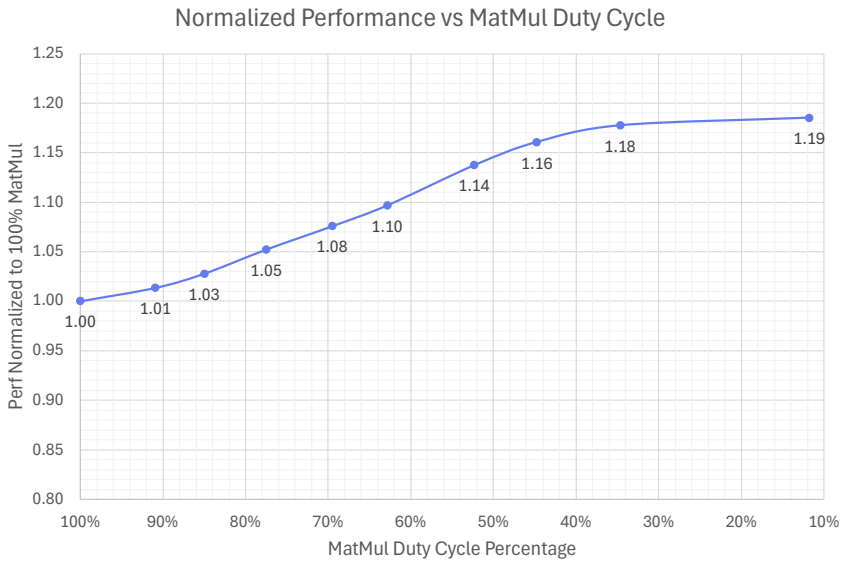
High MatMul activity pushes average power to TGP -> GPU lowers frequency to stay within TGP

Non-MatMul phases reduce average power -> GPU allows higher frequency

Low overall activity -> average power stays well below TGP -> GPU sustains near ~ F_{max}

How can we study this?

Interleaved MatMul and MemRead micro-benchmarking to mimic end-to-end application behavior



*Speedups vary based on TGP/EDPp ratios, based on GPU

Micro-benchmark details

- **MatMul** = highly optimized MatMul kernel (>90% util, random data)
- **MemRead** = kernel reading from memory (~90% util, random data)
- Kernels are interleaved 

- MatMul duty cycle = % of time in MatMul

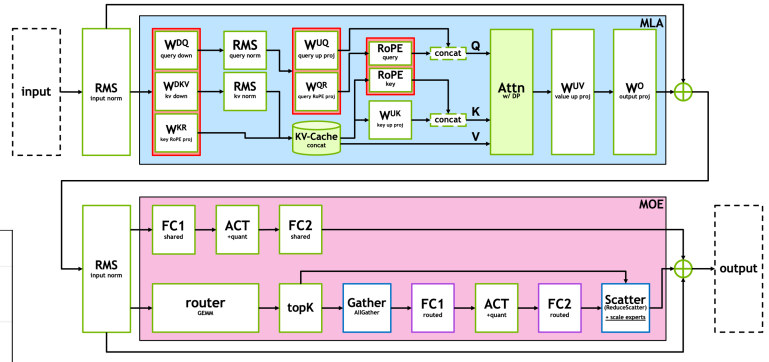
Observations

- 100% MatMul is the most power-limited case
  - Worst-case delivered FLOPS at TGP
  - Pessimistic since most workloads have non-MatMul phases
- Interleaving MemRead increases MatMul performance
  - Low-power phases (MemRead) let MatMul run faster in short bursts
    - During MatMul, power exceeds TGP but gets averaged out to TGP eventually
- Diminishing returns as duty cycle decreases
  - Average power must satisfy short-timescale limits

Real workloads outperform pure MatMul microbenchmarks because low-power phases reduce average power and allow higher performance for MatMuls

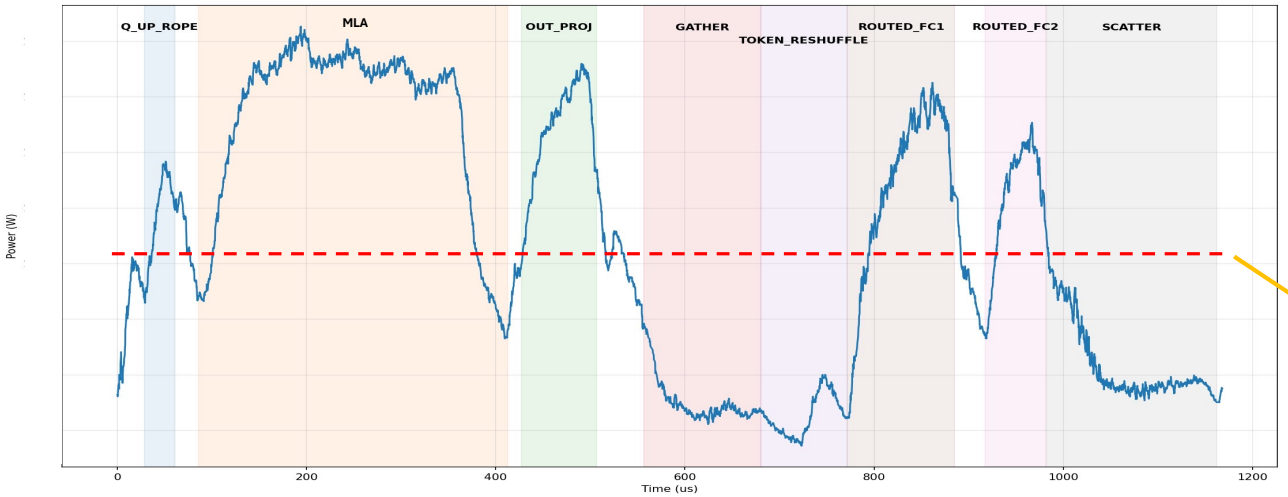
Transformer Block as an Example

Power varies across kernels, but GPU power controller sees only the averaged behavior



DeepSeek-v3 Transformer Block

DeepSeek-v3 Inference | FPROP of a Transformer Block



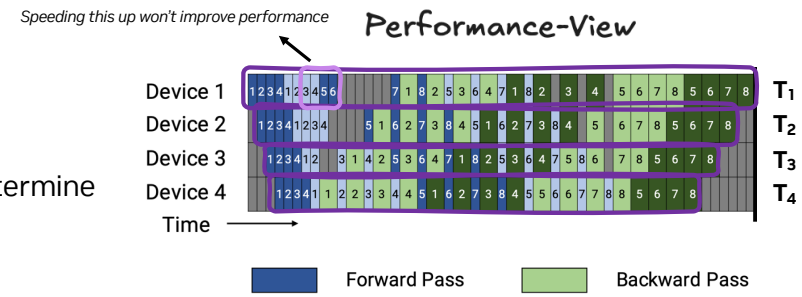
Fluctuations at small timescales get averaged out over a transformer block

Optimizing Performance Under Power Constraints

Performance (Max) vs. Energy (Sum)

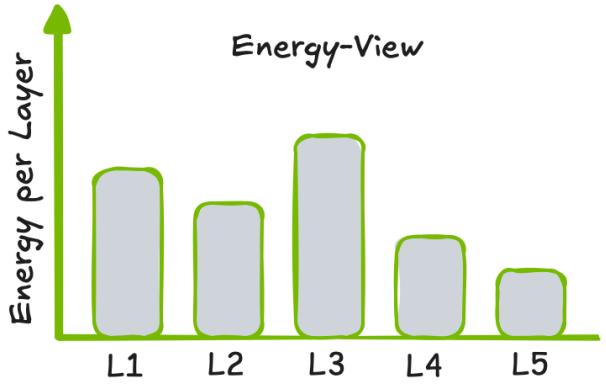
Performance depends on the slowest component; Energy depends on all components

- Performance = $max\ f(x)$
  - End-to-end performance is determined by the critical path
  - Speeding up non-critical work does not improve performance
  - Slowest component at any level {warp -> block -> kernel -> layer} determine
    - $T_{total} = max\ (T_1, T_2, .., T_N)$



- Energy = $sum\ f(x)$
  - Energy is accumulated over all kernels
  - Reducing energy in any kernel improves overall efficiency
    - $E_{total} = \sum_i P_i \times t_i$

- Max vs. Sum applies at every level
  - Within kernels -> across kernels -> across layers -> across the model



Performance is gated by the max; energy is driven by the sum
This asymmetry is why energy optimization is always useful, but performance optimization is selective

SW Optimization Considerations

When SW reduces actual work, the GPU runs faster and more efficiently

- Case Study: MatMul Tile Shapes
  - Default kernel (K1): **128x128** tile
  - Alternate kernel (K2): **128x256** tile

MNK:1024x128000x1024 measured on H100-SXM	K1	K2	Delta
Runtime @ 1400 MHz (us)	**277.9**	291	-4.5%
Power @ 1400 MHz (W)	598	510	17.3%
Runtime @ 700 W (us)	**267.2**	**267.4**	-0.07%
Frequency @ 700 W (MHz)	1462	**1530**	-4.5%
Runtime @ 400 W (us)	370.1	**335.8**	10.2%
Frequency @ 400 W (MHz)	1045	**1215**	-14%

At same frequency, K1 consumes more power and is faster

At fixed power of 700 W, runtime difference is negligible

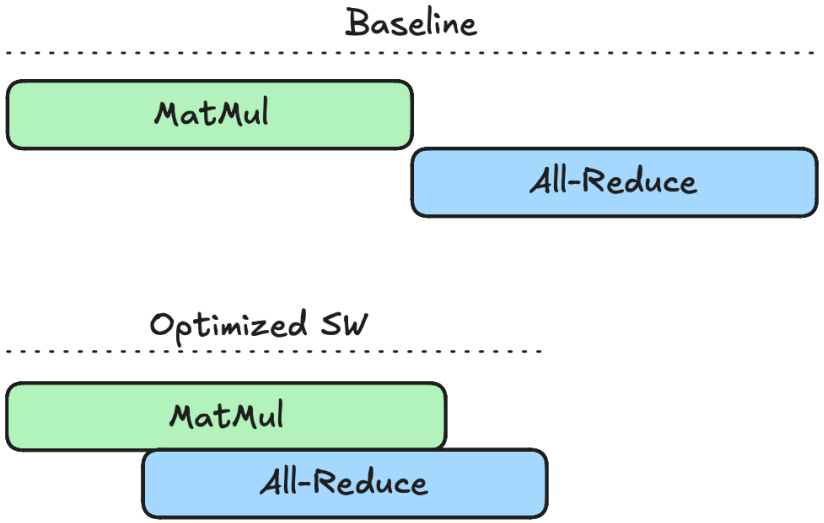
At a lower power limit, K2 is ~10% faster

- Why does K2 behave differently?
  - Larger tiles improve data reuse
  - Fewer L2 bytes -> lower energy -> faster at lower power (perf/W ↑)

What happens when you optimize a piece of SW?

Increasing utilization without reducing work has limited benefit

- Baseline
  - Compute and communication run sequentially
  - Power swings between high (MatMul) and low (All-Reduce)
  - ~60-70% Math utilization, runs at frequency = F_1
- After SW optimization (Compute-Communication Overlap)
  - Overlapping compute-communication increases utilization
  - Combined activity increases average power
  - What happens to overall performance?
    - Performance improves, but less than expected utilization gain
      - Higher average power -> lower sustained frequency $\mathbf{F_2 < F_1}$
  - Did the optimization reduce total work?



True performance gains come from reducing work, not just keeping the GPU busy

Optimization Guidelines

- Simple optimization guidelines:
  - Objective 1: Reduce cycles (increase utilization)
  - Objective 2: Reduce work (fewer ops, fewer bytes -> lower power)

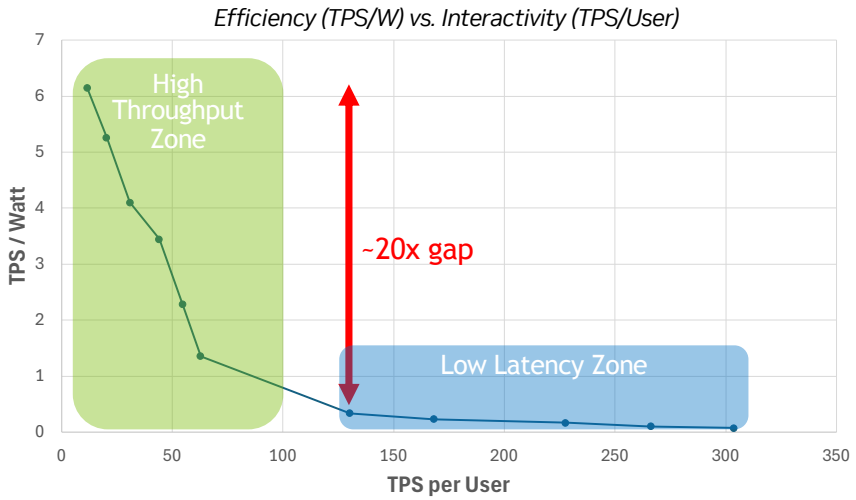
- This is business as usual for GPU developers:
  - Understand and overcome throughput bottlenecks (blocking, data reuse, etc.)
  - Hide latencies (software pipelining, etc.)
  - Overlap compute and non-compute work
  - Improve algorithms to minimize work

- Optimizations that reduce cycles usually also reduce energy
  - Things that hurt performance also hurt energy – wasted ops, extra memory traffic, etc.
    - Example: tuning MatMul traversal order to increase L2 hit rates, reduces DRAM bytes fetched, saves cycles and energy
    - Counter-examples: speculation (with high misprediction rates); operations that are off the critical path

You Can’t Optimize What You Can’t Measure: Energy Monitoring in TRT-LLM

Why tokens/sec/Watt matter for real-world deployment?

- First step to optimize: Measure the cost of inference
  - Efficiency = Tokens per Joule = Tokens/sec/Watt
- Understand your use case:
  - What is the cost of interactivity?
    - Low-latency scenarios sacrifice efficiency
    - High-throughput scenarios provide best Perf/W
  - How much efficiency are you willing to trade for responsiveness?
- Datacenters operate under fixed power budgets
  - Perf/W determines the "factory" throughput in DC power budget
    - Higher Perf/W -> more GPUs -> higher overall TPS



Running GPUs at their best Perf/W point enables more GPUs within the same DC power budget and higher overall throughput

GPU Energy Profiling in TRT-LLM

How to measure GPU energy and identify efficiency improvements?

Goals

- Enable on-the-fly power and energy tracking in TRT-LLM
- Identify Pareto-optimal configs for performance and energy

How to Use

- Collect energy data using public NVML APIs:
  - **GetTotalEnergyConsumption**
    - reports energy consumed since last driver call
- Automatically logs energy if **pynvml** is available
  - default in most NGC containers
- Logged with other performance metrics:

```
-- Energy Metrics -------------------------------------

Total Energy (J):                                160237.5460
Output Tokens per Joule (tokens/J):              13.0878
Average GPU Power (W):                           991.4987
```

Shout out to ML.Energy team for their related work in energy profiling!

https://ml.energy/zeus

Thank you!

We’re Hiring!
Help Us Shape the Future

Backup

Sample Commands

How to use *nvidia-smi*

- **Measure**
  - Sample nvidia-smi command:
    - `nvidia-smi --query-gpu=timestamp, clocks.sm, power.draw, power.draw.instant, temperature.gpu --format=csv -lms 100`

- **Control**
  - Change power limit
    - `sudo nvidia-smi –pl 400`
  - Adjust on-chip clock ratios
    - `sudo nvidia-smi boost-slider --vboost {0, 1, 2, ..}`
  - Lock clocks
    - `sudo nvidia-smi –lgc 1500`

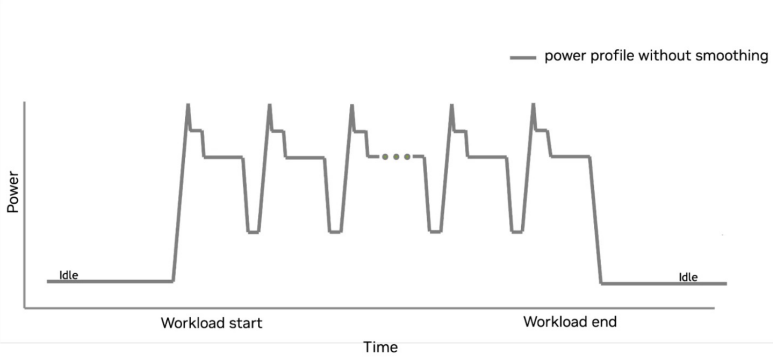
- To learn more https://docs.nvidia.com/deploy/nvidia-smi

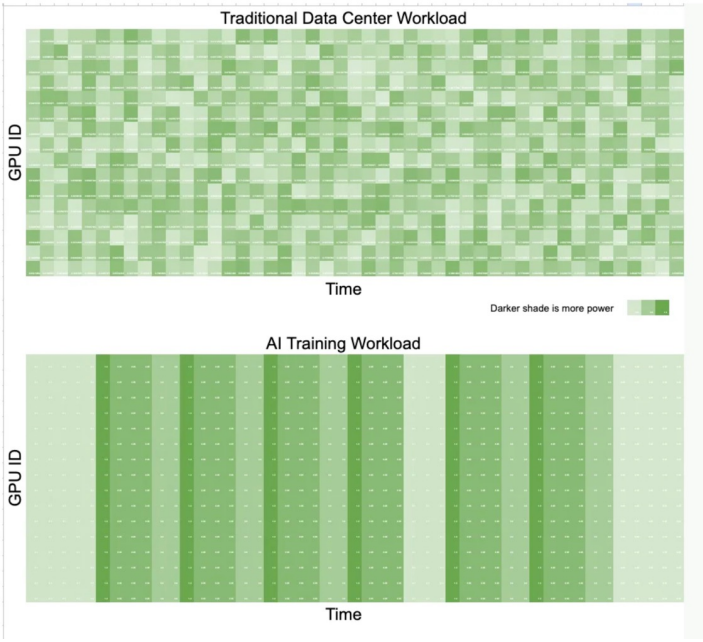
Power Stabilization for AI Datacenters

In the Age of Gigawatt Datacenters

- AI workloads cause large power fluctuations in datacenters
  - ~1000s of GPUs consume power and idle simultaneously
  - Harder for datacenters to provision power
  - Leads to throttling and lower total throughput

- Synchronous nature of training workloads
  - High power / compute-heavy phase
    - GPUs work on local data (e.g. MatMul)
      - Tensor cores operate at high utilization, drawing power close to TGP
  - Low power / communication phase
    - GPUs sync on the data (e.g. All-Reduce, checkpointing)
      - Utilization drops sharply, power collapses





Unlike traditional DC jobs, in AI workloads GPUs operate in lockstep, ~1000s of GPUs breathing power in and out

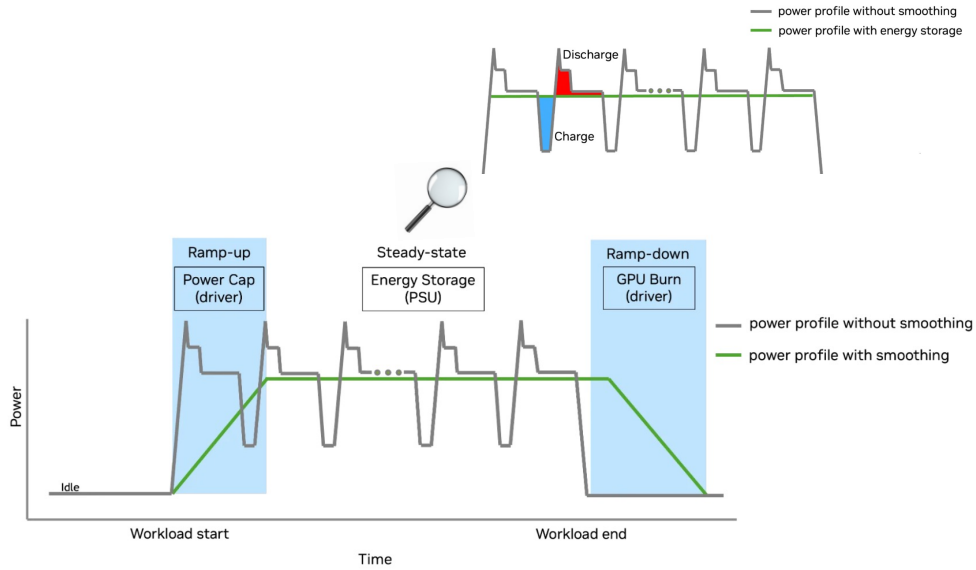
Steady Power Vision

Stable rack-level power improves sustained ML performance

- Energy buffering smooths bursty ML workload power
  - Charge during low-utilization phases (communication, gradient sync)
  - Discharge during high power bursts (MatMul-heavy compute)
  - Keeps rack-level power stable even when GPU activity is spiky

- **Why should ML practitioners care?**
  - Enables provisioning for average, not peak power
    - More GPUs can fit within the same datacenter power budget
    - Reduces throttling and improves overall TPS/Watt

- **How can SW support steady power?**
  - Design models with balanced compute and communication
    - Avoid big power swings from back-to-back MatMul or All-Reduce
    - Overlap compute and communication
  - Profile per layer performance and power
    - Identify kernels that cause transient peaks
  - Favor scheduling strategies that flatten utilization across GPUs
    - Interleaved pipeline scheduling (1F1B, ZeroBubble, DualPipe)



https://developer.nvidia.com/blog/how-new-gb300-nvl72-features-provide-steady-power-for-ai/

Stabilizing power at the DC is not just an electrical problem, it is a system-level enabler for scaling ML performance

GB300 PSU Measured Benefits

Energy buffering reduces peak power, allowing racks to stay within provisioned limits