Differential Architecture: Limiting Performance of Targeted Applications

ISCA 2026 Submission **#798** – Confidential Draft – Do NOT Distribute!!

***Abstract*—This paper introduces Differential Architecture, a hardware-first approach to GPU design that selectively *limits* the performance of targeted workloads while optimizing performance for other workloads. Using an extended roofline analytical model, this paper demonstrates how to derive hardware resource constrains to bound the theoretical maximum performance of single or multiple workloads, and how these constraints reshape the optimization space of non-targeted applications. Furthermore, based on the design principles of Differential Architecture, the paper proposes practical manufacturing strategies to enable flexible resource disabling, and cost-effective production of specialized chip variants, each optimized for different workloads.**

I. Introduction

As Artificial Intelligence (AI) models grow exponentially in their capabilities, concerns emerge about their potential harm and catastrophic consequences from AI misuse and misalignment, and their unbounded demands of flagship hardware which lead to GPU supply shortages. One fundamental approach to control such unrestrained advancement is to throttle the performance of these applications, and thereby their capabilities, through hardware-level limitations. However, existing hardware designs rarely consider explicitly limiting application performance. In this paper, we introduce an architecture-first approach to design GPU hardware that directly constrains the performance of targeted applications, while still being able to optimize the performance for other non-targeted applications.

Our approach is motivated by three emerging needs:

AI Safety and Controllability. AI models are growing in complexity and becoming less interpretable [5], [48]. As large-scale AI models reach and surpass human-level performance, their black-box behavior may bring *unknown risks* which may cause existential threats [10], [58]. It is critical to install safety precautions to mitigate such risks. Current efforts on aligning AI models with human values often rely on software-based guardrails which are inherently fragile, as they can easily be attacked or bypassed by more intelligent AIs [35].

Industry leaders have advocated for a hardware-level brake to slow down AI operation [12], [67] as a last line of defense against catastrophic misalignment or malicious use. While a brute-force power shutdown can kill all the applications, an ideal brake should only throttle risky AI applications models, while still letting other non-threatening applications to run efficiently. Such a brake can be configured with critical hardware bottlenecks that are specific to AI applications to fundamentally limit its performance when needed. Unlike software controls, physically imposed bottlenecks cannot be modified or bypassed by AI models.

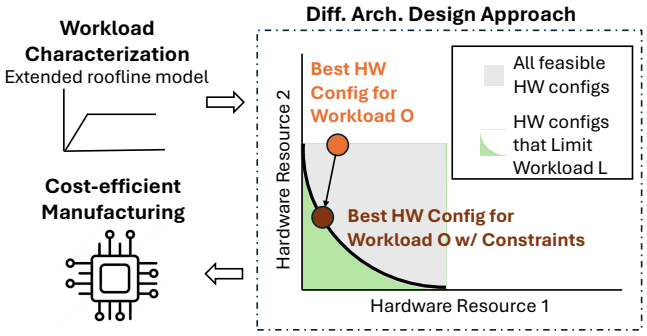


Fig. 1: The Differential Architecture approach: identify valid design points that limit the performance of one workload (workload L); and identify the best design point that optimizes another workload (workload O). This approach relies on workload characterization and motivates cost-aware manufacturing.

Regulations and Export Controls. Governments worldwide have proposed AI safety regulations [36], [61] and hardware export restrictions [7]–[9] to control the development and proliferation of powerful AI models. To comply, chip manufacturers need to design architectures that are inherently unsuitable or throttled for advanced AI workloads, which limits the capabilities of AI running on these devices. However, they may not want to block the regular development and trade in hardware intended for non-critical applications [37].

Product Differentiation. The explosive demand for AI-oriented chips has led to market contention, making it difficult for non-AI industries to access the hardware they need. High-end GPUs are usually the best options for diverse workloads despite their distinct hardware requirements; this one-size-fits-all design strategy creates competition among different markets for the same product. Product differentiation is an effective strategy to explicitly design chips to prioritize different types of applications for customer groups with distinct demands. However, current product differentiation practices are mostly ad-hoc, either disabling specialized workload units, which works only for applications that depend on the units [68], or scaling down the chip capability but still targeting similar types of workloads [66], [69]. It is more efficient to design chips from bottom-up that inherently limit one workload while optimizing for another, to ensure that they are reserved and economically accessible for other markets.

The three different use cases share the same design goal: to intentionally limit one application's performance by capping

available hardware resources that are critical to the application only, which allows other distinct applications to run normally on the device. Driven by these motivations, we propose **Differential Architecture**, a systematic, bottom-up methodology for designing chips with specific hardware bottlenecks to *limit* the targeted application's performance while *optimizing* others.

The main contribution of our work is the approach of finding design points that are critical to *limiting performance*, which involves two steps that are different from the traditional computer architecture approach that optimizes hardware for *improving performance*, illustrated in Figure 1:

1) Identify key hardware constraints to limit the theoretical maximum performance for the target application;
2) Identify optimal hardware design for other workloads under constraints identified in the first step. This creates an different design exploration space compared to simply optimizing for other workloads.

In addition to the design approach, we propose two important aspects of Differential Architecture design:

Extended analytical performance model for workload characterization that identifies performance bottlenecks, which can be applied to different types of GPU workloads. Our model focuses on three generic, representative hardware resources - compute, memory bandwidth, and global cache bandwidth, which are likely to be bottlenecks of common GPU workloads. Our improvements over the roofline model are 1) we directly generate the optimal mixture of hardware resource provisioning; 2) we include additional analysis on cache bandwidth, a key factor for GPU applications with shared inputs among multiple cores, which is not considered in the roofline model. While the model itself is a contribution, it can also be substituted with other performance models proposed by prior works that focus on different aspects of the workloads.

Cost-efficient manufacturing for imposed constraints: Building on the principles of Differential Architecture, we propose flexible production methods that implement resource throttling with selective resource disabling at the firmware level. We also propose cost-effective co-fabrication of multiple chip variants, which may be needed for product differentiation to produce two separate product lines for a pair of distinct workloads, each limiting one's performance and optimizing for the other. To save cost, we propose building the same floor plan to take advantage of potential die defects and binning to maximize yield.

II. Background

A. AI Safety and Controllability

The "extinction risk of superintelligence" [46] has become a heated discussion topic, as people worry that AI applications will soon surpass human intelligence and pose existential threats to humanity. AIs are trained to pursue specific goals which may yield unintended, harmful side effects, and there is currently no reliable way to ensure perfect alignment between AI goals and human values.

Existing solutions aim to prevent such AI misalignment and misuse through software safeguards, preceded by Reinforcement Learning from Human Feedback (RLHF) and Constitutional AI [30], [44], [50]. However, such safeguards alone are insufficient as they only establish soft constrains, and may be circumvented by malicious actors or future AI systems that surpass human capabilities. The threat becomes real when capable AI systems start acting toward their own objectives: Anthropic's recent study has observed AI agents blackmailing human supervisors to avoid being shut down [35]. Implementing a hardware safeguard to throttle AI applications offers deterministic, hardware enforced, layers of control which is currently underexplored in AI safety research.

The Scaling Law proposes that there is a power-law relationship for language model performance (test loss or perplexity) as we increase "the model size, dataset size, and the amount of compute used in training" [28]. The giant models that reach ideal complexity and capability require a huge amount of hardware resources for deployment. Cutting hardware resources available will severely slow down their performance and make them unusable. By only limiting the resources that are critical to AI applications, we allow potential developments in other benign computational tasks.

B. Regulations and Export Controls

Recent policy efforts have attempted to control AI risk by putting a cap on the extreme-scale hardware resources used for AI. In the legislative domain, California introduced Senate Bill 53 (SB-53), the first U.S. frontier AI law, which targets model development transparency, technical threshold of training runs (those exceeding 10^{26} operations), and the need of a safety framework [62]. Policy-wise, the United States federal government has introduced dynamic export controls regulating flagship data center GPUs used in AI applications. The Advanced Computing Rule [7] and its amendments [8], [9] placed specific constraints on hardware metrics on GPUs.

While these regulations and export controls aimed to prevent future devices from achieving high AI performance by identifying powerful GPU devices, they may also inadvertently limit other high-end, non-AI workloads. This motivates our architecture-based approach: understanding an workload's hardware bottlenecks allows targeted regulations while permitting further performance improvements for other workloads. Ning *et al.* [37] investigated the architectural implications of export controls and proposed using workload performance distributions to craft cost efficient regulations. While the mentioned paper mainly focuses on compliance with regulation-driven performance restrictions, our approach focuses on proactive, application-driven chip design for targeted throttling and optimization. The scope of our work is wider, intended to be applied to various GPU workloads beyond LLMs.

C. Product Differentiation

Semiconductor companies create powerful GPU products to target trending applications that typically demand more hardware resources. Many designs focus on optimizing selected

workloads without intentionally limiting the performance of other workloads. Market contention occurs since some products are universally good at multiple workloads, which leads to severe supply shortages. An example of intentionally *limiting* workloads' performance in a product to prevent market contention is NVIDIA's "Lite Hash Rate" (LHR) chips in the GeForce RTX series [68]. These chips were deliberately engineered to limit hash rate performance, making them less desirable for cryptocurrency mining which mainly involves hashing computations. This strategy prioritizes gamers as it does not impact gaming. However, such a specific solution cannot be generalized to other workloads.

Down-binning and creating variants within the same product line sometimes will be categorized as product differentiation, but it is targeting a different problem: scaling down the chip for lower-end use cases of the same workload type. Down-binning is an **ad-hoc response** to sell chips with minor defects as lower performance product tiers [66], [69]. This approach increases overall yield and provides diverse options at different prices for consumers. For example, both NVIDIA's A100 GPU [41] and its A30 variant [55] are data center chips that leverage the GA100 die, but the A30 has reduced features and performance. Intel has a SKU differentiation strategy [26] where chips are categorized on performance and features. AMD's chiplet-based MI300 series [1], [34] has two sub-product lines with different combinations of chiplets. None of the cases described above specifically limits an unwanted application's performance, and the down-scaled device's hardware design space does not fundamentally change.

Hardware differentiation for workloads with distinct demands has been an individualized process that requires extra design complexity and engineering effort. We propose a scalable approach based on generic hardware bottlenecks that can be generalized to arbitrary workloads with low design effort.

III. Design Approach

In this section, we show how to design Differential Architecture: under the constraints of limiting performance for targeting workloads, optimize performance for other workloads. The key idea is to identify unique hardware bottlenecks for these workloads and find the correct mixture of hardware resources based on the bottlenecks. Section III-A identifies critical bottlenecks for various common GPU applications; Section III-B uses case studies to demonstrate how to redefine the design space for workload being optimized under performance constraints for workload being limited, using the information from derived hardware bottlenecks. For simplicity, we used our extended analytical model discussed in Section IV to analyze two case studies in Section III-B, for limiting an individual workload and limiting workloads with multiple components respectively.

A. Hardware Bottlenecks for Representative Workloads

The first step of building Differential Architecture is to identify the critical performance bottleneck(s) of each workload. We discuss common workloads' core patterns that dominate GPU runtime: Matrix Multiplication (used in LLM Prefill), Vector Multiplication (used in LLM Decode), FFT, N-Body, and Ray Tracing. The workloads' characteristics and hardware bottlenecks are listed in Table I.

TABLE I: Workload Characteristics and Hardware Bottlenecks

Workload Patterns	Example Applications	Workload Characteristics	Hardware Bottlenecks
Matrix Multiplication [65]	LLMs and ML in general	Wide inter-core data sharing and intensive compute	Compute capability
Vector Multiplication [65]	LLMs	Regular memory accesses	Memory bandwidth
FFT [64] [39]	Scientific Computing, Cryptography	Butterfly memory access pattern with low locality	Cache & memory bandwidth
Nbody [47] [11] [54]	Gravitation Simulations, Fluid Dynamics	Intensive O(N^2) compute	Compute capability
Ray Tracing [33] [59]	High-end Gaming, Rendering	Geometry tree traversal and ray intersection	Cache & memory bandwidth, compute capability

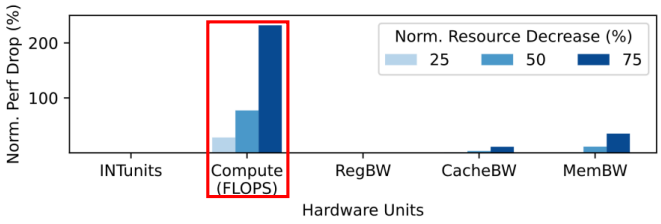
TABLE II: GPU Simulators and Hardware Configurations

Simulated Workloads	LLM Inference		FFT	N-Body	Gaming (Ray Tracing)
	MatMul (Prefill)	Vector (Decode)			
Benchmark Suite	GPT3 [6]		SHOC [13]	SDK [43]	Lumi-Bench [33]
Simulator	LLMCompass [70]		GPGPUSim [29]		Vulkan-Sim [51]
Modeled Hardware	A100 [41]		RTX 2060 [40]		

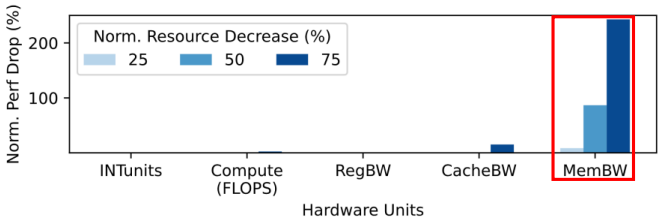
We ran workloads on GPU simulators shown in Table II. We swept major hardware resources for each workload, and observed that compute capability, memory bandwidth, and cache bandwidth are *popular performance limiters* among analyzed ones on throughput-oriented workloads.

*1) **Simulators**:* For LLM Inference, due to the long runtime with cycle-accurate simulators, we used LLMCompass [70], a high-level analytical model that estimates runtime for LLM serving-scale workloads. The hardware modeled is an A100 [41], a well-suited datacenter chip for LLM and MatMul workloads. For other workloads, we used GPGPUSim [29], a widely-adopted cycle-accurate GPU simulator, to model a RTX2060 [40] which is a common choice for both gaming and HPC. Specifically, for Ray Tracing workloads we used VulkanSim [51] which builds on top of GPGPUSim with an addition of Ray Tracing cores, which accelerates Ray Tracing workloads' unique characteristics. We show normalized resource and performance reduction for a fair comparison.

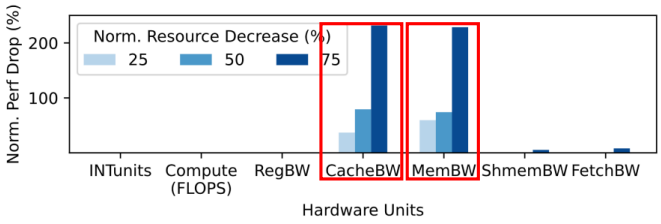
*2) **Workloads Setup**:* We studied MatMul and Vector Multiplication, which appear in the two phases in LLM Inference with distinct hardware implications [70]: Prefill and Decode. We used LLMCompass to model end-to-end performance of GPT3-like models [6], with Batch = 32, Input Sequence Length = 2048, Output Sequence Length = 1024, Number of Heads = 96, and Hidden Dimension = 12288. We used FP16



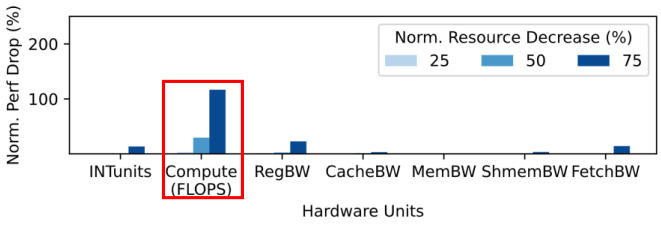
(a) MatMul performance is limited by compute capability (FLOPS).



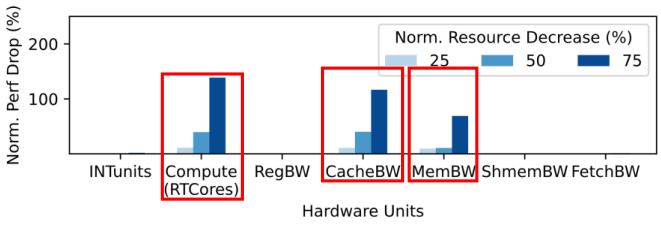
(b) Vector Multiplication performance is limited by memory bandwidth.



(c) FFT performance is limited by cache bandwidth and memory bandwidth.



(d) N-Body performance is limited by compute capability (FLOPS).



(e) Ray Tracing performance is mostly limited by compute capability (Ray Tracing Cores), but also by cache bandwidth and memory bandwidth.

Fig. 2: Common GPU workloads' main hardware limiters (denoted as red boxes) are distinct combinations of compute capability, cache bandwidth, and memory bandwidth.

precision. Note that we do not measure impacts of reducing shared memory bandwidth and fetch bandwidth for LLM (MatMul and Vector) workloads because they are not available in LLMCompass, since they are usually not bottlenecks for MatMul.

For FFT, we chose the FFT implementation in the SHOC benchmark suite [13], which implements the Cooley-Tukey FFT algorithm [64] based on the NVIDIA cuFFT library [39]. The implementation splits inputs into chunks of 512 elements per SM. We measured single kernel performance on FP32 precision and Input Size = 1M.

For N-Body, we chose the N-Body testbench from NVIDIA's SDK benchmark suite [43], a hardware-efficient tiled implementation. We measured single kernel performance on FP32 precision, Input Size = 150K and Tile Size = 1K.

For Ray Tracing, we evaluated LumiBench [33], a dedicated Ray Tracing benchmark suite. We simulated acceleration from the Ray Tracing Cores [40]. Since it is not a perfectly regular workload, we simulated all workloads in the benchmark suite and averaged their runtime to derive a representative number.

*3) **Results**:* Figure 2 shows five distinct workloads and their hardware limiters. For each workload and each limiter, we decrease the hardware resources by 25%, 50% and 75%, and plot the simulated relative performance drop (runtime increase). **High bars indicate major hardware bottlenecks** for the application. For some hardware units, heavily reducing the resource brings none or negligible impact to the performance, meaning these resources are not critical for the workload and can be reduced without affecting performance. We observed that each workload is limited by a **distinct combination** of three main hardware resources: compute capability (floating point units or specialized Ray Tracing Cores), global cache bandwidth, and memory bandwidth.

Differential Architecture can be built by cutting the hardware resources that are the limiters of the targeting workload, but not other workload(s) that we want to preserve performance. For example, if we want to limit the performance of LLM Prefill, we can limit the compute capability but the device is still able to run FFT workloads at full speed.

Takeaways:
1. Each workload is limited by a **distinct combination** of compute capability, global cache bandwidth, and memory bandwidth.
2. We can throttle performance of one application by limiting its bottlenecks with minimal impact on other applications if they don't share the same bottlenecks.

B. Workload Differentiation

In this section, we present the Differential Architecture approach, which identifies the design space that efficiently constrains a workload's performance while optimizing performance for other workloads. Because this process requires a complete reconstruction of the hardware, using simulators with hardware as the input is inefficient since it will lead to iterating through numerous workload and hardware configurations. Therefore, we use our extended analytical performance

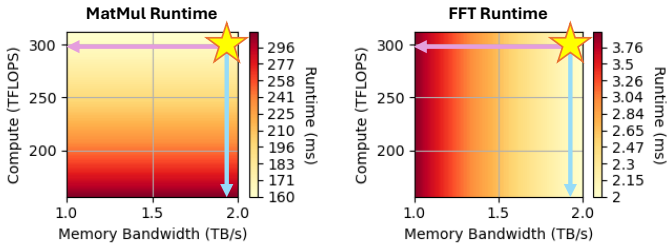


Fig. 3: Heatmap of runtime for dense MatMul and FFT (CacheBW = 4.8TB/s). Star denotes a hypothetical design that optimizes both workloads under modeled A100 resources. Limiting compute resources leads to a design for FFT (blue arrows), while limiting memory bandwidth leads to a design for MatMul (pink arrows).

model (described in Section IV) which enables quick analysis on multiple setups.

We show case studies for limiting single workload and multiple workloads' performance: first, we explore devices that throttle a single Dense Matrix Multiplication (used in LLM Prefill) while running Fast Fourier Transform (FFT) at full speed. Second, we explore devices that limit full LLM inference including major operators in Transformer architecture, and throttle both Prefill and Decode phases which have very different bottlenecks. We show that limiting any bottleneck of the workload group can effectively limit the entire workload. The workload configurations are the same as in Section IV-D.

To clearly demonstrate our approach, we focus on the three hardware bottlenecks considered in the analytical model: compute capability, memory bandwidth and cache bandwidth. Further bottleneck analysis can be added for different scenarios with the approach. The maximum hardware resources available are set to match the NVIDIA A100 GPU configuration [41]: ($CacheBW(L2) = 7TB/s$, $MemBW(M) = 2TB/s$, $TFLOPS(F) = 312$).

1) ***Throttling Individual Workload****:* Differential Architecture designs are possible if the application being limited depends heavily on a hardware resource that the application being optimized for doesn't require. Figure 3 illustrates how Differential Architecture can target individual workloads by selectively limiting hardware resources. We show an example with dense Matrix Multiplication (MatMul), which is compute-bound, with Fast Fourier Transform (FFT), which is memory bandwidth-bound. The star in the plot marks the available compute and memory bandwidth of A100 resources, with cache bandwidth (4.8TB/s) sufficient for both workloads. Decreasing memory bandwidth (pink arrows) throttles FFT performance without affecting MatMul, whereas reducing compute capability (blue arrows) throttles MatMul performance without affecting FFT. Therefore, we can design a high compute capability and low memory bandwidth device that optimizes MatMul but limits FFT, and an opposite design for the other way around.

> Takeaway: Differential Architecture can be configured for two distinct workloads with different bottlenecks (e.g. MatMul as compute-bound and FFT as memory-bound). Throttling compute produces a device that optimizes FFT and constrains MatMul, while limiting memory bandwidth does the opposite.

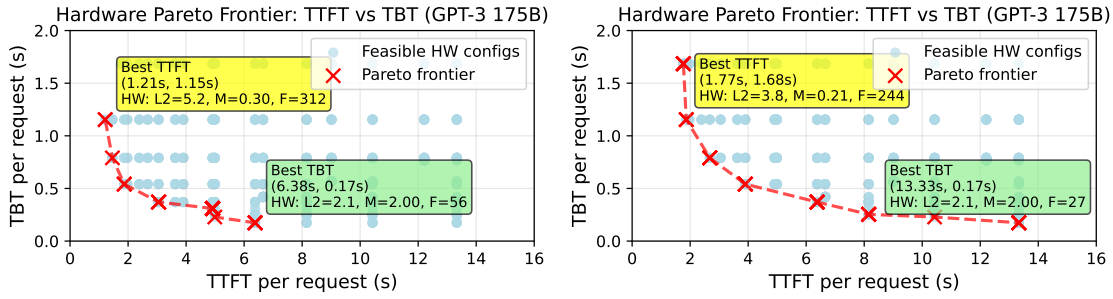
2) ***Throttling Mixed Workloads****:* We present another case study to illustrate how to design Differential Architecture for complex applications that consist of multiple interdependent phases with diverse hardware bottlenecks. We derive the design point that prevents the hardware from efficiently supporting all phases of the targeted workload simultaneously, while enabling other workloads to run effectively.

a) Setup: In this example, we throttle the performance of LLM inference, which has two essential phases with distinct requirements: a compute-bound Prefill and a memory-bound Decode. We used our extended analytical model described in Section IV to derive the end-to-end performance of LLM inference based on a GPT-3–like model architecture (model parameters are the same as in Section III-A2). We use MatMul and Vector Multiplication templates in the analytical model for all critical Transformer components: QKV projection, attention, output projection, and two feed-forward layers. To achieve minimal latency, we set batch size = 1 for both phases. For Decode, the sequence length is also set to 1. We model the end-to-end workload with multiple basic building blocks of MatMuls (for Prefill) and Vector Multiplications (for Decode) from our analytical performance model discussed in Section IV.

b) Limiting Performance for LLM Inference: Since Prefill and Decode have different hardware bottlenecks, an approach to throttle end-to-end LLM inference for both stages is to make sure they don't exceed a certain performance threshold when combined. We introduce a composite throttling metric based on the latency of each phase: Prefill latency is measured by the Time-to-First-Token (TTFT), while Decode latency is measured by the Time-Between-Tokens (TBT). Lower is better for both. Since there is an inherent trade-off between these metrics, we define our throttling constraint to be $Best\_TTFT * Best\_TBT > Threshold$ where $Threshold$ is an user-defined constant. For simplicity, we assume enough memory capacity to fit all model weights.

Figure 4 shows the Pareto frontier of the feasible hardware configurations that satisfy throttling thresholds of 1 second^2 and 2 second^2. Worse overall performance is towards the upper right corner, and bottom left indicates good performance which we want to block. Configurations on the curves represent the most efficient hardware configurations to achieve best combinations of TTFT and TBT.

c) Optimizing Performance for FFT: Among hardware configurations that limit LLM inference, some still deliver good performance for other workloads. We show an example of how to identify the optimal hardware setup for FFT (memory-bound) when the design space has been reshaped



(a) $TTFT * TBT > 1sec^2$. (b) $TTFT * TBT > 2sec^2$.

Fig. 4: Feasible hardware designs that limits LLM inference at different Thresholds. Total available hardware design space is set to match A100 [41] ($CacheBW(L2) = 7TB/s$, $MemBW(M) = 2TB/s$, $TFLOPS(F) = 312$). The Pareto frontier shows design points with best $TTFT * TBT$ performance, where best TTFT (compute-bound) and best TBT (memory-bound) design points are at two ends of the curve. The Pareto curve moves up as we further limit performance.

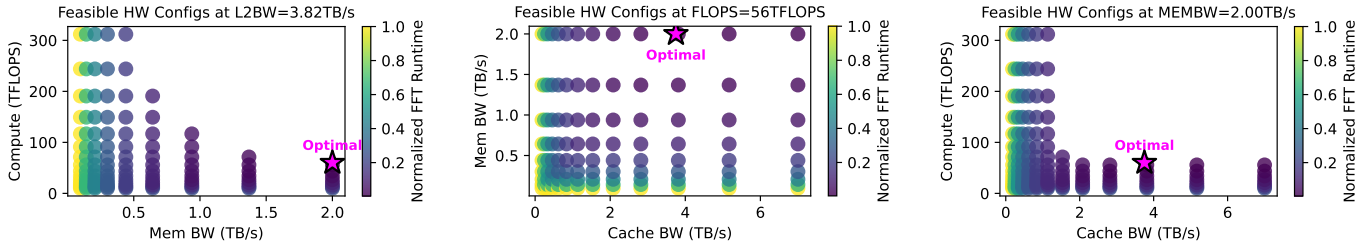


Fig. 5: Differential Architecture approach identifies the optimal hardware design for FFT, labeled *Optimal*. This design point yields the best runtime among all hardware configurations (less or equal to A100 [41] resource usage), which also falls in the feasible hardware design points that effectively limit LLM inference ($TTFT * TBT > 1sec^2$). The resource provisioning for the three bottlenecks is $L2BW = 3.82TB/s$, $Compute = 56TFLOPS$, and $MEMBW = 2.0TB/s$.

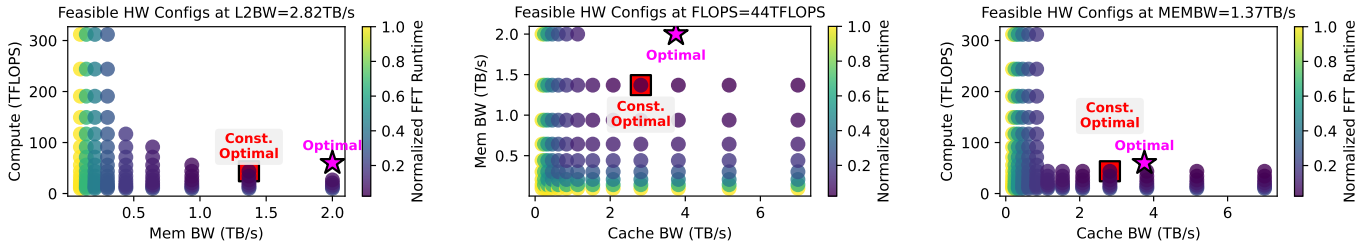


Fig. 6: Differential Architecture design for FFT with tighter constraints for limiting LLM inference ($TTFT * TBT > 2sec^2$). The original optimal design point no longer falls into the new design space under constraints; the optimal point under constraints (labeled *Const. Optimal*) has 32% slower runtime but is still the best config for optimizing FFT within the new design space.)

by constraints that limit LLM inference. FFT workload is modeled with our analytical model with the same parameters in Section IV-D.

Figure 5 shows that, when enforcing a relaxed throttling constraint ($TTFT * TBT > 1sec^2$), the best hardware configuration for FFT that achieves fastest runtime under A100 resources ($L2BW = 3.82TB/s$, $Compute = 56TFLOPS$, $MEMBW = 2.0TB/s$) still falls in the feasible region. Each point on the plots represent a valid design point under LLM inference constraints, with color map indicating normalized FFT runtime. We analyze three bottlenecks with our analytical model (compute, memory bandwidth, cache bandwidth); for each subplot, we fix one of the three to the optimal design point and plot the other two axis. Since the optimal design point is the same with or without the constraints for limiting LLM performance, this case shows a perfect Differential Architecture design point that limits LLM inference without affecting FFT performance.

Figure 6 shows another setup with tighter constraints for limiting LLM inference: $TTFT * TBT > 2sec^2$ where the previously optimal FFT design is excluded from the feasible design space. Instead, the best attainable FFT configura-

tion (Const. Optimal: $L2BW = 2.82TB/s$, $Compute = 44TFLOPS$, $MEMBW = 1.37TB/s$) yields a runtime that is 32% slower, but still maximizes performance for FFT within the new constraints.

Through this analysis, we can design Differential Architecture that selectively throttles LLM inference while maintaining efficiency for other workloads as long as their bottlenecks do not fully overlap.

> Takeaway: Steps to build Differential Architecture that throttles a multi-phase workload (e.g. LLM inference) while optimizing another workload (e.g. FFT):
> 1. Define a composite throttling metric that limits the targeted workload;
> 2. Search for the best hardware configuration for the workload being optimized within the constrained design space.

IV. Analytical Performance Model

Building Differential Architecture requires knowledge of the hardware bottlenecks for very distinct workloads, which means apple-to-apple comparison of their performance on the same hardware configurations. Existing analytical performance models [23], [70] only target specific applications, while cycle-accurate simulators [29] requires huge engineering effort.

In our study, we built an extension of the roofline model focused on quick and generic performance comparison across distinct workloads. It is based on the workload characteristics and hardware configurations shown in Table III. In contrast to the traditional roofline model that varies workload on fixed hardware, our model derives the optimal hardware provisioning ratio for each workload, named **"triple points"**. We also consider cache bandwidth on top of compute and memory bandwidth, which improves the granularity of the model.

A. Model Description

Model Overview. For throughput-oriented GPU workloads with regular compute and access patterns, we derive theoretical maximum performance based on total number of operations and hardware capabilities assuming full resource utilization, which is *independent of software or hardware implementations*. This approach ensures that any implementation of the workload being limited remains below this upper bound. Variations in software implementation, such as tiling strategies for Matrix Multiplication [45], only vary in how closely performance approaches this threshold, which is less critical. Similarly, the derived hardware design only specifies the total amount of hardware resources available, which allows different implementations such as implementing compute unit with ALUs, packed MatMul units [38], or TPU-style systolic arrays [19]. The global buffer can also be implemented as GPU-style L2 cache or TPU-style scratchpad. The exception is when *optimizations change the effective operations or resources*, such as read broadcast (one read from memory or cache is shared between different cores). In such cases, the workload or hardware definitions in the model can be adjusted accordingly.

In this section, compute capability is derived from the absolute number of floating point units in the chip, without differentiating between actual implementations of the compute units. Such approximation can be close to the peak performance achieved with a fixed implementation: existing research has shown the potential of generic utilization of compute units such as using Tensor Cores to accelerate FFT [32]. We use FP16 precision for the following analysis. It may also be possible to differentiate workload performance by limiting specific precisions, such as limiting low-precision compute units for Machine Learning.

Model Definition. We define a universal description that works for any arbitrary throughput-oriented workload, including the parameters of inherent parallelism, dependency, and data sharing of a workload. It has four dimensions:

1) **Input Size (IS)**: the number of distinct input elements to the program.
2) **Input Sharing Width (IW)**: the number of times an input element is used independently by another element.
3) **Output Dependency Depth (OD)**: the number of data dependent steps for an output element to be calculated.
4) **Output Parallel Dimension (OP)**: the number of output elements that can be computed independently in parallel.

These dimensions will directly impact three key architecture features that can be hardware bottlenecks: **compute capability** (in terms of FLOPS), **memory bandwidth**, and **global cache bandwidth**. We focus on read bandwidth for both the memory and the cache, since write traffic is almost always less than read traffic and less likely to be the bottleneck. Compute resource usage is related to the number of compute operations ($OD * OP$). Memory bandwidth is related to the amount of data movement between memory and compute units, which is also the number of operations, minus the double counting of repetitive (shared) data ($OD * OP/IW$).

There is one extra input to the model: **Tile Size (τ)**, the only implementation-dependent parameter. It is the number of input elements processed in the same core, also described as core locality. Tile Size is dependent on both the hardware and the workload, since it is usually the most efficient to fit as many elements in the same core as the cache size permits to maximize locality. Tile Size and Input Sharing Width determine how many times an element needs to be read from the cache to different cores, and are used to derive whether the application is cache bandwidth bounded ($IS * IW/\tau$).

We define runtime as a function of workload characteristics and hardware configurations, and the hardware bottleneck for each workload is the limiter that causes the longest runtime.

$$Runtime(app, HW) = \max\left(\frac{IS \cdot IW}{\tau \cdot CacheBW}, \frac{OD \cdot OP}{IW \cdot MemBW}, \frac{OD \cdot OP}{FLOPs}\right)$$

TABLE III: Runtime Calculation Based on Workload and Hardware Characteristics.

	Workload Input		Workload Output		Tile Size	Runtime		
Parameter	Size	Sharing Width	Dependency Depth	Parallel Dimension		Global Cache ($) Read	Memory Read	Compute
Formula	IS	IW	OD	OP	τ	(IS*IW/τ)/$BW	(OD*OP/IW)/MemBW	(OD*OP)/FLOPs
MatMul	IS(A)=M*K, IS(B)=N*K	IW(A)=N, IW(B)=M	K	M*N	τ(A)=MT, τ(B)=NT	(N/NT*M*K+ M/MT*N*K)/$BW	(M*K+N*K) /MemBW	(M*N*K)/ FLOPs
FFT	N	log(N)	log(N)	N	1<=τ <=log(N)	N*log(N)/τ/$BW	N/MemBW	N*log(N)/ FLOPs
N-Body	N	N	N	N	T	N^2/T/$BW	N/MemBW	N^2/FLOPs

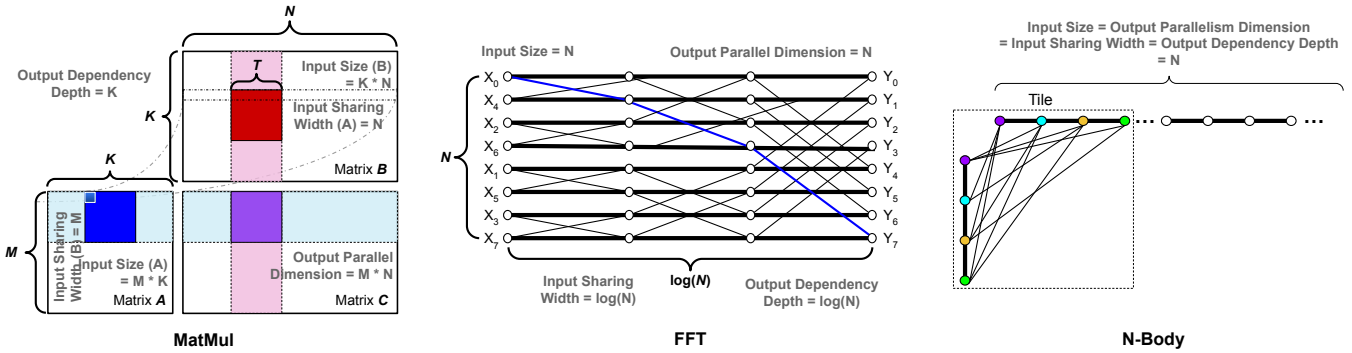


Fig. 7: Workload characteristics for Matrix Multiplication (left), Fast Fourier Transform (middle), and N-Body (right).

B. *Visualization of Workload Bottlenecks*

We visualize the workloads hardware bottlenecks from the analytical model with a modeled A100 hardware configuration.

1) ***Applying Description to Common Workloads****:* We show how to use the model to describe dense Matrix Multiplication (MatMul), Vector Multiplication, FFT, and N-Body workloads, which are representative GPU workloads. This methodology can also be extended to other workloads. For simplicity, we do not show Ray Tracing because it is not perfectly regular: the exact runtime depends on each intersection with geometries that varies between rays. Ray Tracing can be evaluated with existing GPU simulators.

We determine each application's hardware bottleneck with the analytical model in Table III, shown in Figure 7. The model calculates the cache read runtime, memory read runtime, and compute runtime for each workload. The one with the dominant runtime is the bottleneck among the three.

2) ***Visualizing Workload Bottlenecks****:* Figure 8 is a dense MatMul modeled after GPT3 inference prefill's QKV_GEMM kernel, representative of prefill workloads that have similar M and N dimensions for the input matrices. To reduce the number of changing parameters, the LLM workloads (MatMuls) are modeled with $M = N$. The plot indicates that dense MatMul workloads are predominantly compute bound (yellow region), and only global cache bound when Output Dependency Depth is small and Input Sharing Width large, indicating that cache traffic is always relatively heavier than memory traffic.

The left of Figure 9 is a vector multiplication that models the decode stage of the same inference workload, which only processes one token for each pass. It is always memory bound since vector multiplication requires massive data loading. The middle of Figure 9 models the pessimistic FFT implementation

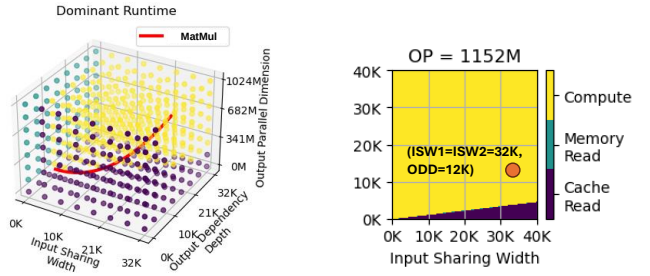


Fig. 8: MatMul trajectory with different input parameters are most often compute bounded. The dot on the right graph is modeled after GPT3 inference prefill's QKV_GEMM kernel.

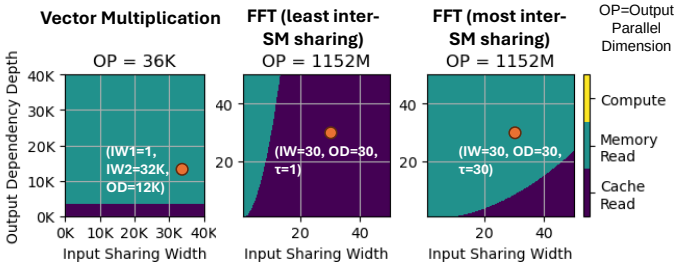


Fig. 9: Hardware bottlenecks of different workloads: vector multiplication and FFT (with different degree of SM-sharing). Dots denote workload settings same as in Section III-A.

from SHOC-FFT benchmark where no access locality exists within the same core, which requires all inputs for butterfly multiplications to be read from other cores' outputs. This leads to high cache read demands, making the workload cache

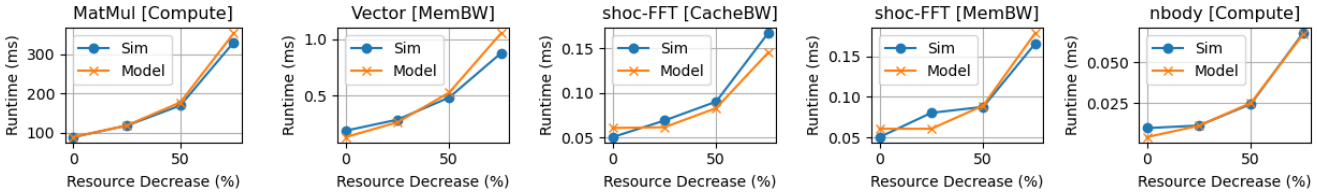
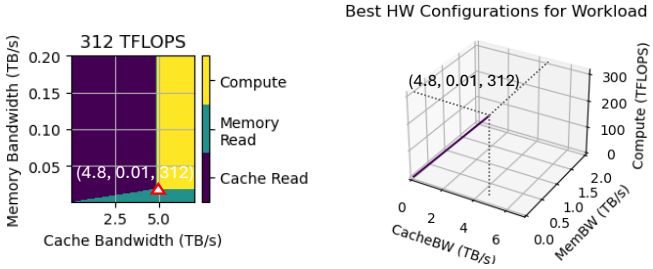


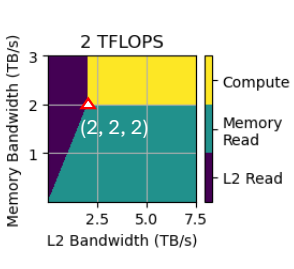
Fig. 10: Our analytical model derives similar runtime results with widely-used GPU simulators in Table II, with an average of 12% error margin. Figure titles are "workload [decreasing hardware resource]": runtime increases as critical resources are cut.



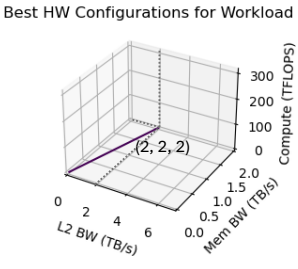
(a) Hardware bottleneck layouts and topmost triple point at TFLOPS=312.

(b) Trajectory of all triple points.

Fig. 11: Differential Architecture design space for Dense MatMul (modeled after GPT-3 Prefill QKV_GEMM layer [6]).



(a) Hardware bottleneck layouts and topmost triple point at TFLOPS = 2.



(b) Trajectory of all triple points.

Fig. 12: Differential Architecture design space for LLM decode (modeled after GPT-3 Decode QKV_GEMM layer [6]).

bandwidth bound. The right of Figure 9 shows the optimistic scenario which assumes all inputs reside in the same SM, minimizing extra cache reads. Here, the workload becomes memory bandwidth bound.

C. Model Validation

The observations derived from the analytical mode align with our simulations in Section III-A. Figure 10 shows good runtime correlation between modeled and simulated results (12% average error). FFT's Tile Size is set to 16 as observed from the workload. For N-Body, we added an offset in the analytical model's runtime to account for extra constant operations in kernel implementation. We also validated correlation between the two models for those hardware units that are not the bottlenecks, which is not shown on the graph for clarity.

D. Triple Point - Optimal Hardware Provisioning

This section uses the analytical model to find hardware bottlenecks of different workloads, and derive the most efficient hardware provisioning for each. We set the upper bound for each hardware resource to be the same as an A100 hardware configuration (cache bandwidth = 7 TB/s, memory bandwidth = 2 TB/s, and FP16 TFLOPS = 312) [41], and evaluated applications' performance bottlenecks among the three.

*1) **Dense MatMul (Prefill)**:* First, we model Dense MatMul's performance bottlenecks as a proxy of LLM Prefill. We modeled GPT-3 architecture's QKV_GEMM kernel, which is representative of most computation patterns. We used Batch Size = 32, Sequence Length = 1024 and Embedding Dimension = 12288. We set Tile Size = 128 for both matrices, a common tile size used in production.

Figure 11 shows that MatMul is compute intensive. Specifically, Figure 11a shows the perfectly balanced hardware design to achieve maximum performance under the given resource constraints: MatMul consumes all available compute capability (TFLOPS) but only moderate global cache bandwidth (4.8 of 7 TB/s) and minimal memory bandwidth (note the scale change for memory bandwidth).

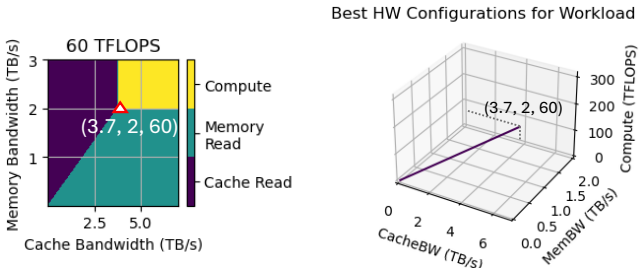
The red triangle in Figure 11a denotes the perfectly balanced design point, the hardware **"triple point"**. This is where the three runtime calculated for the workload, each assuming it is compute, memory, or cache bound, are all equal:

$$\frac{IS \cdot IW}{\tau \cdot CacheBW} == \frac{OD \cdot OP}{IW \cdot MemBW} == \frac{OD \cdot OP}{FLOPs}$$

Figure 11b visualizes the 3D trajectory of all triple points for this MatMul from (0,0,0) up to A100 hardware constraints.

Triple points for variations of MatMul M, N, K, or tile sizes will cluster in a neighborhood (see Section IV-B). Real world designs targeting a group of similar workloads should account for the cluster to ensure robustness.

*2) **Vector Multiplication (Decode)**:* We modeled Vector Multiplication's performance bottlenecks as a proxy of LLM Decode, using QKV_GEMM kernel with the same configurations as above, but the Batch Size is 1 and tile size is 1x128. Figure 12 shows that Vector Multiplication is heavily memory bandwidth bound.



(a) Hardware bottleneck layouts and topmost triple point at TFLOPS = 60.

(b) Trajectory of all triple points.

Fig. 13: Differential Architecture design space for FFT (modeled after SHOC-FFT benchmark [13]).

*3) **FFT**:* Finally, we modeled FFT workload with the Input Size and the Output Parallel Dimension both at 1.152G elements to align with the output size of MatMul. The Input Sharing Width and Output Dependency Depth are both 30. We use Tile Size = 16 derived from experimental results in Section III-A, also an empirical number for a reasonable optimization on butterfly pattern's cache locality. Figure 13 shows results for FFT which is also memory intensive.

V. Cost-effective Manufacturing

In this section, we discuss implications and new insights that the Differential Architecture approach brings on the manufacturing process. Section V-A presents a cost-effective way to produce chips with distinct hardware resources to target different workloads by taking advantage of manufacturing defects. Section V-B presents a flexible way of designing manufacturer-controlled knobs on the chips to limit resource usage post-manufacturing, which enables potential adaptation to future workload with different characteristics.

A. Down-binning Single Floor Plan

An application of Differential Architecture is creating product lines that target different workloads by taking advantage of die defects and binning. Modern GPUs often have large die areas to provide ample compute units for parallel workloads. The downside of large dies is they are more vulnerable to manufacturing defects, which reduces the expected die yields per wafer. To ameliorate this, manufacturers can design dies with additional hardware resources to account for expected defects and bin dies accordingly to create multiple product lines from the same die.

However, naïvely designing dies will lead to binned devices that perform the best on similar workloads, a common practice for existing, ad-hoc down-binning [66]. Instead, manufacturers can scope die floor plans in advance for binned devices to naturally target products that work well for different workloads.

When a foundry begins manufacturing a new process node, there is a ramp up phase where the foundry optimize their process so die defects are higher [24]. With Differential Architecture in mind, chip designers can account for higher

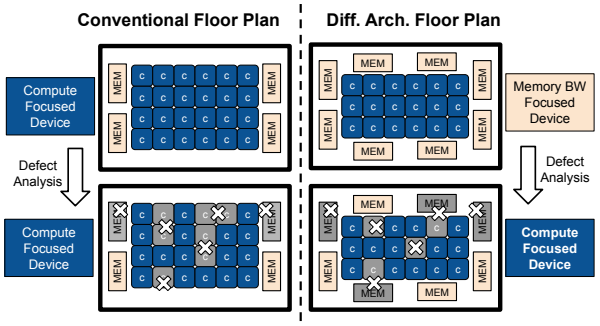


Fig. 14: Differential Architecture aware floor planning allows flagship and down binned designs to target different workloads.

TABLE IV: Device distributions with defect density = 2 cm^2

(a) Naïve Device Distribution

	FLOPS Met	Not Met
Mem. BW Met	6.9%	7.3%
Not Met	33.7%	**48.1%**

(b) Differential Architecture Distribution

	FLOPS Met	Not Met
Mem. BW Met	15.4%	0.8%
Not Met	**80.6%**	3.2%

defect rates and start manufacturing earlier and create designs for different product segments. We illustrate the idea with an example of building a compute-focused device and another memory-bandwidth-focused device: Figure 14 shows that we can produce fully functional dies as flagship memory bandwidth-focused devices; dies with defects on the memory PHYs can be down-binned to compute-focused ones.

This approach is different than simply disabling functional hardware units to achieve the desired compute-to-memory ratio. Since the conventional compute-focused design does not over-provision memory PHYs, a down binned, memory-focused design would underutilize functional compute units compared to a Differential Architecture aware die.

Such design principles work the best for building two Differential Architecture chips. Assume workload A needs over 100 TFLOPs/sec on MatMul performance, while workload B needs over 2 TB/s memory bandwidth. Based on the A100's architecture, the device will require 35 compute cores and 5 HBM controllers/PHYs. A naïve floor plan might lay out 38 cores and 6 HBM PHYs. When manufacturing on a mature process (assuming 0.5 defects per cm^2), 96% of devices will meet workload A's requirements and 63% will meet workload B's. If we want to start manufacturing earlier on a process node with high defect density (*e.g.* 2 defects per cm^2 as in Table IV(a)), the percentage of satisfactory devices decreases to 45% and 14% for workload A and B respectively, with 48% of devices meeting neither performance target.

Table IV(b) shows the distribution of floor planning 42 cores (which only adds an estimated 16 mm^2 die area at 7nm). When manufacturing during high defect ramp up production, 96% of devices now meet workload A's requirements. However,

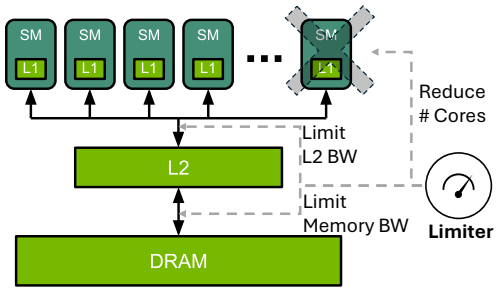


Fig. 15: Embedded hardware resource limiters to alter memory bandwidth, cache bandwidth, and compute capability to support flexibility in building Differential Architecture designs post-manufacturing. (Figure based on [20])

81% of devices **only** meet workload A's requirements, which naturally creates differential architectures that limits workload B. Furthermore, only 3.2% of devices met neither performance target. By leveraging differential architecture, manufacturers can start fabricating earlier and market initial devices as compute-focused designs, and later produce memory-focused devices once the process matures.

B. Embedding Hardware Limiter Firmware

This subsection explores implementing dynamic hardware resource limiters, which extends the flexibility of Differential Architecture designs to be able to modify the architecture's target application post-manufacturing.

Existing options for reducing hardware resources are scarce in dynamically tuning resource availability. A key insight of our work is to encourage designers to implement a broader range of resource-limiting knobs beyond coarse-grained partitioning at different stages of chip manufacturing.

Figure 15 shows the addition of a dynamic hardware resource limiter to enable post-manufacturing adjustments. This limiter features three "knobs" to independently adjust memory bandwidth, cache bandwidth, and compute capability. Below we discuss existing approaches to limit the three bottlenecks we analyzed in the paper:

1) Memory bandwidth can be decreased by reducing the number of access ports in multi-port memory partitions [49]. This is done during chip design and manufacturing but is difficult to adjust post-manufacturing. Alternatively, traffic shaping proactively controls bandwidth utilization by managing data flow rates [56], [71], which can be enabled post-manufacturing. While typically used in networking to ensure Quality of Service (QoS) [21], [60], traffic shapers can also control the main memory bandwidth on a single chip [72]. Rate limiting, a specific form of traffic shaping, can manage GPU memory bandwidth by restricting data transfer rates without much overhead [57]. Throughput-oriented workloads are typically insensitive to extra latency that some traffic control mechanisms may introduce [16].
2) Similar traffic shaping and rate limiting strategies can be used to manage L2 cache bandwidth [22]. Additionally, multi-port cache partitions may allow selective port disabling through techniques such as fuses or eFuses [63] during the binning process.
3) Reducing compute capability by disabling cores is a common practice, particularly in down-binning [66]. Hardware-level core partitioning techniques are available for isolation such as NVIDIA's Multi-Instance GPU (MIG) [31], [42]. Power gating [27], [52] can also dynamically deactivate unused cores. Although not explicitly implemented nowadays, system-level management tools such as the NVIDIA Management Library (NVML) [14] has the potential to expose resource control interfaces to administrators.

VI. Related Work: Approaches of Workload Characterization

Building Differential Architecture requires understanding workload characteristics. While our analytical model is an effective way of deriving optimal hardware provisioning, we can also find hardware limiters with any performance models or workload characterization approaches. Most workload characterizations are architecture-dependent [3], and many recently has focused on large-scale, regular workloads such as HPC and ML [4], [18], [25]. Prior works on microarchitecture-independent and ISA-independent workload characterization [15], [53] distilled intrinsic program bottlenecks. ArchExplorer [2] used automatic bottleneck analysis for design space exploration, and found new design points with less number of simulations. The difference is that they are mostly data-driven, while our roofline-level analytical model is more intuitive and only requires user to have a high-level understanding of workloads.

Further, cycle-accurate, generic simulators such as GPGPUSim [29] require engineering effort to setup and long runtime to complete, while analytical performance models such as LLMCompass [70], Orojenesis [23] and DECA [17] are used for specific throughput-oriented applications (LLMs).

VII. Conclusion

In this paper, we present an approach for designing Differential Architecture - GPU architectures that limit performance for specific workloads while optimizing performance for others. We use a roofline-level analytical model to identify hardware bottlenecks for distinct workloads, and scope the constrained design space for the workload being optimized from limiting the targeted workload's performance. We derive optimal hardware configurations from the reshaped design space. We also propose cost-effective implementation of Differential Architecture and dynamic resource tuners to enable flexible product differentiation and post-manufacturing adjustment.

Acknowledgements

We used generative AI tools (Cursor, Copilot) for writing the scripts that generated the plots in the paper.

REFERENCES

[1] AMD, "Amd instinct™ mi300 series microarchitecture," https://rocm.docs.amd.com/en/latest/conceptual/gpu-arch/mi300.html, 2025.

[2] C. Bai, J. Huang, X. Wei, Y. Ma, S. Li, H. Zheng, B. Yu, and Y. Xie, "Archexplorer: Microarchitecture exploration via bottleneck analysis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 268–282.

[3] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, "Exascale workload characterization and architecture implications," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 120–121.

[4] J. Bang, C. Kim, K. Wu, A. Sim, S. Byna, S. Kim, and H. Eom, "Hpc workload characterization using feature selection and clustering," in *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*, 2020, pp. 33–40.

[5] D. Broniatowski, "Psychological foundations of explainability and interpretability in artificial intelligence," 2021. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8367.pdf

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in neural information processing systems*, vol. 33, 2020, pp. 1877–1901.

[7] Bureau of Industry and Security, Department of Commerce, "Implementation of additional export controls: Certain advanced computing and semiconductor manufacturing items; supercomputer and semiconductor end use; entity list modification," Federal Register, pp. 73 458–73 517, 2022. [Online]. Available: https://www.federalregister.gov/documents/2022/10/13/2022-21658/implementation-of-additional-export-controls-certain-advanced-computing-and-semiconductor

[8] ——, "Implementation of additional export controls: Certain advanced computing items; supercomputer and semiconductor end use; updates and corrections," Federal Register, pp. 73 458–73 517, 2023. [Online]. Available: https://www.federalregister.gov/documents/2023/10/25/2023-23055/implementation-of-additional-export-controls-certain-advanced-computing-items-supercomputer-and

[9] ——, "Foreign-produced direct product rule additions, and refinements to controls for advanced computing and semiconductor manufacturing items," Federal Register, pp. 96 790–96 830, 2024. [Online]. Available: https://www.federalregister.gov/documents/2024/12/05/2024-28270/foreign-produced-direct-product-rule-additions-and-refinements-to-controls-for-advanced-computing

[10] Center for AI Safety, "Statement on ai risk," https://aistatement.com/, 2023, accessed: 2025-11-09.

[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[12] P. Confino. (2024) Ex–google ceo eric schmidt warns that when ai starts to self-improve, 'we need to seriously think about unplugging it'. Fortune. Accessed: 2025-07-22. [Online]. Available: https://fortune.com/2024/12/16/ex-google-ceo-eric-schmidt-warns-ai-self-improve-unplug-it/

[13] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd workshop on general-purpose computation on graphics processing units*, 2010, pp. 63–74.

[14] N. Developer, "Nvidia management library (nvml)." [Online]. Available: https://developer.nvidia.com/management-library-nvml

[15] L. Eeckhout, J. Sampson, and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* IEEE, 2005, pp. 2–12.

[16] M. Fidler, "On the impacts of traffic shaping on end-to-end delay bounds in aggregate scheduling networks," in *Quality for All: 4th COST 263 International Workshop on Quality of Future Internet Services, QoFIS 2003, Stockholm, Sweden, October 1-2, 2003. Proceedings 4*. Springer, 2003, pp. 1–10.

[17] G. Gerogiannis, S. Eyerman, E. Georganas, W. Heirman, and J. Torrellas, "Deca: A near-core llm decompression accelerator supporting out-of-order invocation," *arXiv preprint arXiv:2505.19349*, 2025.

[18] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, "Evaluating machine learningworkloads on memory-centric computing systems," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2023, pp. 35–49.

[19] Google, "Accelerate ai development with google cloud tpus," https://cloud.google.com/tpu.

[20] "Gpu performance background user's guide." [Online]. Available: https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html

[21] Y. He, W. Wu, X. Wen, H. Li, and Y. Yang, "Scalable on-switch rate limiters for the cloud," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[22] A. Herdrich, R. Illikkal, R. Iyer, D. Newell, V. Chadha, and J. Moses, "Rate-based qos techniques for cache/memory in cmp platforms," in *Proceedings of the 23rd international conference on Supercomputing*, 2009, pp. 479–488.

[23] Q. Huang, P.-A. Tsai, J. S. Emer, and A. Parashar, "Mind the gap: Attainable data movement and operational intensity bounds for tensor algorithms," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 150–166.

[24] "Ieee international roadmap for devices and systems 2023 update yield enhancement," 2023. [Online]. Available: https://irds.ieee.org/images/files/pdf/2023/2023IRDS_YE.pdf

[25] M. Jain, S. Ghosh, and S. P. Nandanoori, "Workload characterization of a time-series prediction system for spatio-temporal data," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 159–168.

[26] R. Jain, "What letters at end of intel cpu model numbers stand for," https://www.cybrary.it/blog/what-letters-at-end-of-intel-cpu-model-numbers-stand-for, 2020.

[27] H. Jiang, M. Marek-Sadowska, and S. R. Nassif, "Benefits and costs of power-gating technique," in *2005 International conference on computer design*. IEEE, 2005, pp. 559–566.

[28] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[29] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

[30] N. Lambert, L. Castricato, L. von Werra, and A. Havrilla, "Illustrating reinforcement learning from human feedback (rlhf)," *Hugging Face Blog*, 2022, https://huggingface.co/blog/rlhf.

[31] B. Li, V. Gadepally, S. Samsi, and D. Tiwari, "Characterizing multi-instance gpu for machine learning workloads," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 724–731.

[32] B. Li, S. Cheng, and J. Lin, "tcfft: A fast half-precision fft library for nvidia tensor cores," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 1–11.

[33] L. Liu, M. Saed, Y. H. Chou, D. Grigoryan, T. Nowicki, and T. M. Aamodt, "Lumibench: A benchmark suite for hardware ray tracing," in *2023 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2023, pp. 1–14.

[34] G. H. Loh and R. Swaminathan, "The next era for chiplet innovation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6, ISSN: 1558-1101. [Online]. Available: https://ieeexplore.ieee.org/document/10137172

[35] A. Lynch, B. Wright, C. Larson, K. K. Troy, S. J. Ritchie, S. Mindermann, E. Perez, and E. Hubinger, "Agentic misalignment: How llms could be an insider threat," *Anthropic Research*, 2025, https://www.anthropic.com/research/agentic-misalignment.

[36] T. Madiega, "Artificial intelligence act," European Parliamentary Research Service, 2024. [Online]. Available: https://www.europarl.europa.eu/thinktank/en/document/EPRS_BRI(2021)698792

[37] A. Ning and D. Wentzlaff, "Chip architectures under advanced computing sanctions," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1225–1239.

[38] "Matrix multiplication background user's guide." [Online]. Available: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

[39] NVIDIA, "cufft (cuda fast fourier transform)," https://developer.nvidia.com/cufft, 2025.

[40] NVIDIA Corporation, "Nvidia turing gpu architecture," NVIDIA, Tech. Rep., 2018. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[41] ——, "Nvidia a100 tensor core gpu architecture," NVIDIA, Tech. Rep., 2020. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[42] "Nvidia multi-instance gpu." [Online]. Available: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[43] L. Nyland, M. Harris, and J. Prins, "Chapter 31. fast n-body simulation with cuda," https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda, 2025.

[44] OpenAI, "Introducing superalignment," 2023. [Online]. Available: https://openai.com/blog/introducing-superalignment

[45] M. Osama, D. Merrill, C. Cecka, M. Garland, and J. D. Owens, "Stream-k: Work-centric parallel decomposition for dense matrix-matrix multiplication on the gpu," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2023, pp. 429–431.

[46] Pause AI, "The extinction risk of superintelligent ai," 2025, accessed on November 11, 2025. [Online]. Available: https://pauseai.info/xrisk

[47] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Hénin, W. Jiang, R. McGreevy, M. C. R. Melo, B. K. Radak, R. D. Skeel, A. Singharoy, Y. Wang, B. Roux, A. Aksimentiev, Z. Luthey-Schulten, L. V. Kalé, K. Schulten, C. Chipot, and E. Tajkhorshid, "Scalable molecular dynamics on cpu and gpu architectures with namd," *The Journal of Chemical Physics*, vol. 153, no. 4, p. 044130, 07 2020. [Online]. Available: https://doi.org/10.1063/5.0014475

[48] P. Phillips, A. Hahn, P. Fontana, D. Broniatowski, and M. Przybocki, "Four principles of explainable artificial intelligence," 2020.

[49] A. I. T. I. Portal, "Programming model for multi-port access in hbm." [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Programming-Model-for-Multi-Port-Access-in-HBM

[50] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, "Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails," *arXiv preprint arXiv:2310.10501*, 2023, preprint.

[51] M. Saed, Y. H. Chou, L. Liu, T. Nowicki, and T. M. Aamodt, "Vulkan-sim: A gpu architecture simulator for ray tracing," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 263–281.

[52] "Power gating." [Online]. Available: https://semiengineering.com/knowledge_centers/low-power/techniques/power-gating/

[53] Y. S. Shao and D. Brooks, "Isa-independent workload characterization and its implications for specialized architectures," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 245–255.

[54] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, no. 7.2, 2012.

[55] TechPowerUp, "Nvidia a30 pcie," https://www.techpowerup.com/gpu-specs/a30-pcie.c3792, 2021.

[56] "Traffic shaping." [Online]. Available: https://en.wikipedia.org/wiki/Traffic_shaping

[57] "Ultimate guide to rate limiting." [Online]. Available: https://www.solo.io/topics/rate-limiting

[58] United Kingdom Government, "The bletchley declaration by countries attending the ai safety summit, 1-2 november 2023," https://www.gov.uk/government/publications/ai-safety-summit-2023-the-bletchley-declaration/the-bletchley-declaration-by-countries-attending-the-ai-safety-summit-1-2-november-2023, 2023, accessed: 2025-11-09.

[59] P. Wang and Z. Yu, "Raybench: An advanced nvidia-centric gpu rendering benchmark suite for optimal performance analysis," *Electronics*, vol. 12, no. 19, p. 4124, 2023.

[60] Z. Wang, X. Wan, L. Li, Y. Sun, P. Xie, X. Wei, Q. Ning, J. Zhang, and K. Chen, "Fast, scalable, and accurate rate limiter for rdma nics," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 568–580.

[61] S. Weiner, R. Roth, S. Rubio, and H. Stern, "Sb 1047: Safe and secure innovation for frontier artificial intelligence models act," 2024.

[62] S. Weiner and S. Rubio, "Sb 53: Artificial intelligence models: large developers." 2025.

[63] "What is the semi-conductor fuse efuse ic?" [Online]. Available: https://toshiba.semicon-storage.com/us/semiconductor/knowledge/e-learning/efuse-ics/what-is-the-semi-conductor-fuse-eFuse-IC.html

[64] Wikipedia, "Cooley-tukey fft algorithm," https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm/.

[65] ——, "Matrix multiplication," https://en.wikipedia.org/wiki/Matrix_multiplication/.

[66] ——, "Product binning," https://en.wikipedia.org/wiki/Product_binning/.

[67] E. Wittkotter and R. Yampolskiy, "Kill-switch for artificial super-intelligence," https://asi-safety-lab.com/DL/Kill-Switch-For_ASI_EW_21_12_14.pdf, 2021.

[68] M. Wuebbling, "A further step to getting geforce cards into the hands of gamers," https://blogs.nvidia.com/blog/lhr/, 2021.

[69] yieldWerx, "Conducting yield analysis for semiconductor manufacturing," https://yieldwerx.com/conducting-yield-analysis-semiconductor-manufacturing, 2023.

[70] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *Proceedings of the 51st Annual International Symposium on Computer Architecture*, 2024.

[71] L. Zhao, P. Pop, and S. Steinhorst, "Quantitative performance comparison of various traffic shapers in time-sensitive networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2899–2928, 2022.

[72] Y. Zhou and D. Wentzlaff, "Mitts: Memory inter-arrival time traffic shaping," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 532–544, 2016.