A fork() in the Road: Cross-Architecture Considerations

Quentin Anthony

Zyphra

AMD Kernel Blog

Zyphra

Introduction: The State of SIMD Architectures

- NVIDIA still dominates, but AMD is catching up
  1. AMD MI300X is cheaper, has more VRAM quantity + bandwidth than H100
  2. These make MI300X great for inference (lower $/tok, less parallelism and/or higher batch size, faster decode)
  3. Navigating AMD software is hard, and forward-pass kernels are easier to write
  4. Because of #1-3, **many companies are training on NVIDIA and inferencing on AMD**
- GPU software is hard to navigate, and optimizations cost labor and time
  - Therefore, we seek to reuse as much of the existing algorithms (e.g. GEMM kernels, FlashAttention, etc) and their associated implementations (CUDA, Triton, CUTLASS, etc) as much as possible
  - What about new kernels? How can we write them with this bifurcated ecosystem in mind (do I write in CUDA then port to HIP? Do I just write in Triton? Etc)
- This talk seeks to give a practitioners' guide to optimizing kernels across architectures

Some Things Translate

- Let’s use FlashAttention (FA) as an example GPU op to run on H100 and MI300X
- Some things translate to any SIMD architecture
  - Algorithms and their structure, such as (for FA)
    - IO-aware chunking (all SIMD have registers/SRAM/HBM/etc)
- Some things don’t
  - Tuning
    - E.g. MI300X has less SRAM but more SMs, so different block sizes are amenable
    - Number of kernel pipeline stages
  - Kernel flow
    - Harder to define, but GPU kernels require a lot of different hardware units (CUDA/Tensor cores, memory stages and their buses, etc) and control flow instructions (synchronization, overlap, etc) to work together
    - Even if a kernel is ported and tuned, bottlenecks often exist due to implicit assumptions in the kernel’s flow
    - Needs kernel restructuring!

Kernel Translation

- NVIDIA Kernel written in:
  - Triton
  - CUDA

    Hipification
  - CUTLASS

- AMD Kernel written in:
  - Triton
    - No conversion needed
    - AMD compiler target needs work
    - Mid-level perf
  - HIP
    - Easy conversion
    - Requires tuning
    - Often requires restructuring
    - High perf
  - Composable Kernel (CK)
    - Manual conversion from ground up
    - Requires tuning + restructuring from ground up
    - High perf

Triton Example

- https://github.com/Dao-AILab/flash-attention/blob/main/flash_attn/flash_attn_triton.py

CUDA → HIP Example

- https://github.com/ROCm/rocm-blogs/blob/release/blogs/software-tools-optimization/hipify/README.md

Porting Kernels Summary

- Triton
  - Great for rapid prototyping on AMD and NVIDIA
  - You’re limited by the compiler
    - Getting to near-optimal on NVIDIA is more achievable, the AMD compiler target still has some holes
    - Convenience languages are a double-edged sword. Low-level optimization becomes doubly hard:
      - With CUDA/HIP or PTX/AMDGCN, you *just* need to understand hardware optimization
      - With Triton, you need to understand optimization *and* the triton compiler itself
      - Example: In the FA example, register pressure was high. I want to keep the Q tensor resident in registers. In CUDA/HIP and PTX/AMDGCN, I directly update the kernel. In Triton, I need to figure out how the compiler generates PTX/AMDGCN and update that logic to account for my kernel without breaking the compiler overall

Porting Kernels Summary

- CUDA/HIP
  - General-purpose GPU programming models. Great for max expressivity and therefore optimization.
  - Hipification lets you have something running day-1, but optimization takes more time
    - In the best-case for simpler kernels, you just manually tune, then play bottleneck whack-a-mole until optimal
    - In the worst-case, one needs to full rewrite around the target hardware’s implicit assumptions
- CUTLASS (NVIDIA) and Composable Kernel (AMD)
  - Optimized building blocks for common DL subroutines
  - Many kernel engineers are moving towards these convenience libraries. More flexibility and easier to optimize than Triton, as well as faster prototyping than CUDA/HIP
- Even with all of this, modern models need to be trained with diverse hardware in mind!
  - SIMD architectures are highly sensitive to kernel input sizing!

Introduction: Transformer Sizing

Andrej Karpathy
@karpathy

The most dramatic optimization to nanoGPT so far (~25% speedup) is to simply increase vocab size from 50257 to 50304 (nearest multiple of 64). This calculates added useless dimensions but goes down a different kernel path with much higher occupancy. Careful with your Powers of 2.

10:36 AM · Feb 3, 2023 · **1.2M** Views

Introduction: Transformer Sizing

- Consider (2^{10}) vs (2^{10}-1) neurons in a hidden layer
  - Same model accuracy
  - ~10-40% increase in layer throughput?!

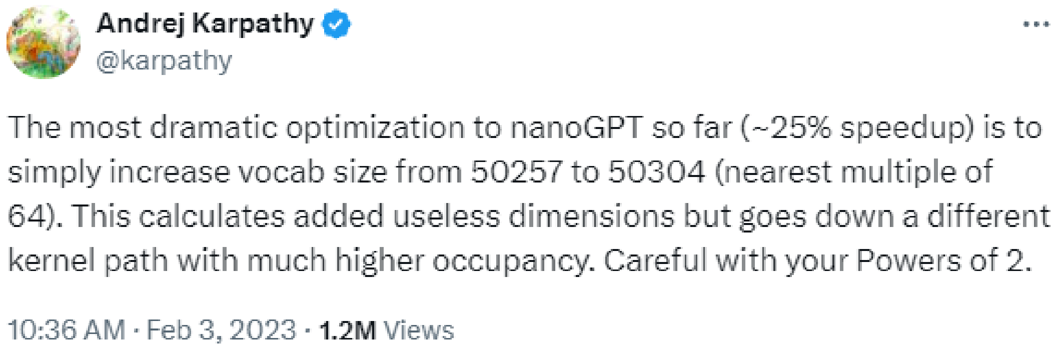
Andrej Karpathy
@karpathy

The most dramatic optimization to nanoGPT so far (~25% speedup) is to simply increase vocab size from 50257 to 50304 (nearest multiple of 64). This calculates added useless dimensions but goes down a different kernel path with much higher occupancy. Careful with your Powers of 2.

10:36 AM · Feb 3, 2023 · 1.2M Views

Introduction: Transformer Sizing

- Consider (2^{10}) vs (2^{10}-1) neurons in a hidden layer
  - Same model accuracy
  - ~10-40% increase in layer throughput?!


Andrej Karpathy
@karpathy

The most dramatic optimization to nanoGPT so far (~25% speedup) is to simply increase vocab size from 50257 to 50304 (nearest multiple of 64). This calculates added useless dimensions but goes down a different kernel path with much higher occupancy. Careful with your Powers of 2.

10:36 AM · Feb 3, 2023 · 1.2M Views

- Finding efficient sizes is especially important since model designers tend to copy structure
  - E.g. this 2.7B was defined in GPT-3, then used in OPT, GPT-Neo, Cerebras-GPT, RedPajama-INCITE, and Pythia
  - Small tweaks to this shape provides over 20% throughput improvement!

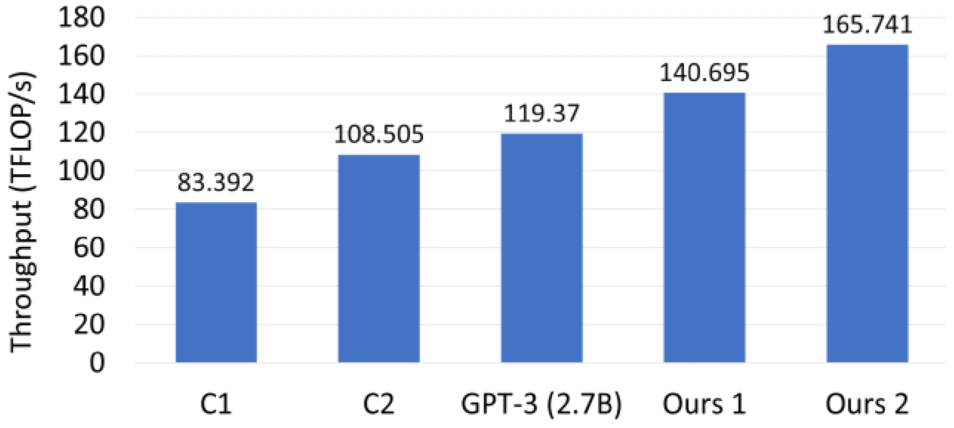


Fig. 1: Transformer single-layer throughput of various architectures for a 2.7 billion parameter model (C1 and C2 are defined by this paper as C1: $h = 2560, a = 64$, C2: $h = 2560, a = 40$).

Sizing

Sizing

a	Number of attention heads	s	Sequence length
b	Microbatch size	t	Tensor-parallel size
h	Hidden dimension size	v	Vocabulary size
L	Number of transformer layers		

TABLE I: Variable names.

- Treat transformer model as a sequence of GPU kernels
  - What kernels take up a single layer's latency?

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.

Sizing

a	Number of attention heads	s	Sequence length
b	Microbatch size	t	Tensor-parallel size
h	Hidden dimension size	v	Vocabulary size
L	Number of transformer layers		

TABLE I: Variable names.

- Treat transformer model as a sequence of GPU kernels
  - What kernels take up a single layer's latency?
- As model size grows, majority are GEMMs from attention and MLPs (4h → h and h → 4h)

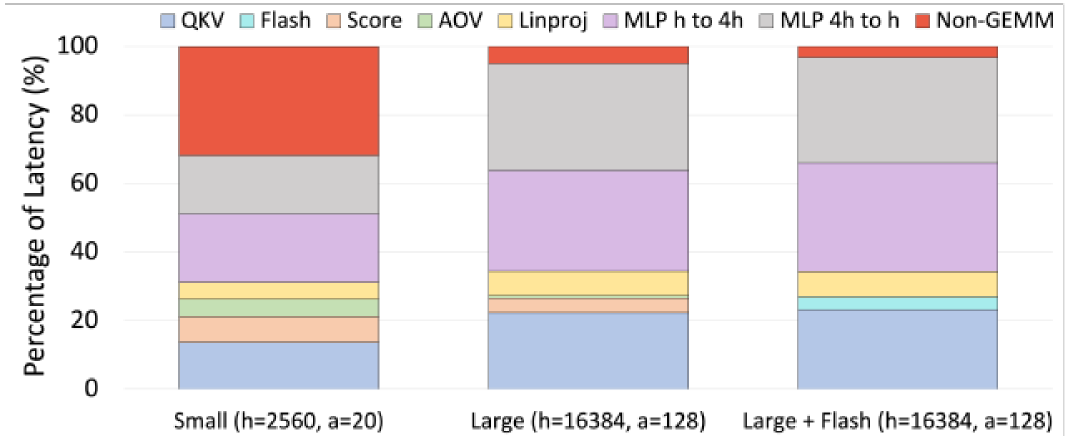


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.

Sizing

- **Good news**: MLP GEMM sizes depend on h and 4h, which is easy to make divisible by 64!
  - Just need a large enough hdim to saturate GPU cores. Classic roofline.

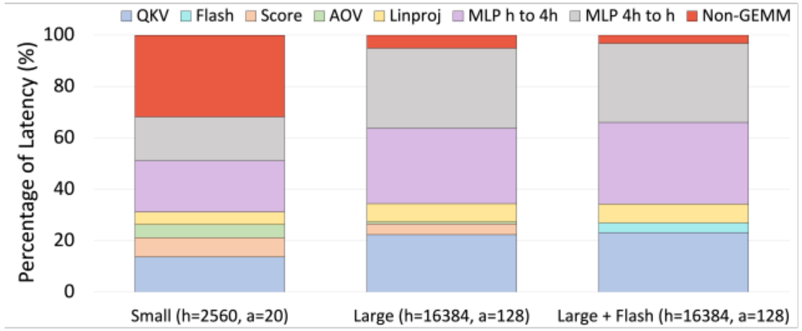


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.

Sizing

- **Good news**: MLP GEMM sizes depend on h and 4h, which is easy to make divisible by 64!
  - Just need a large enough hdim to saturate GPU cores. Classic roofline.
- The only MLP constraint is capacity. Make sure hdim is large enough to escape the memory-bound regime!

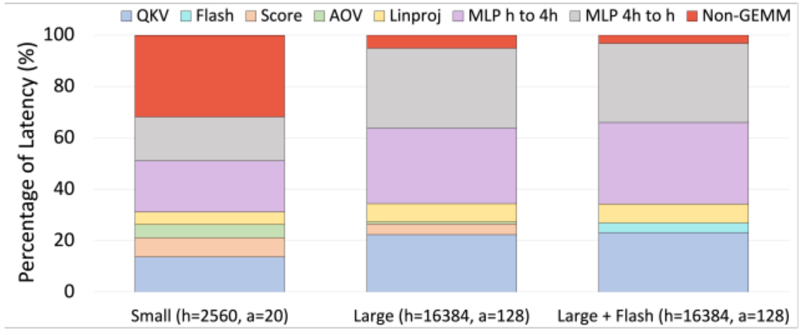
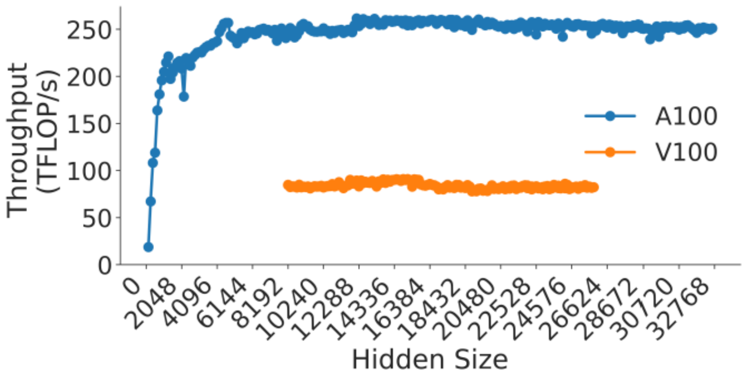


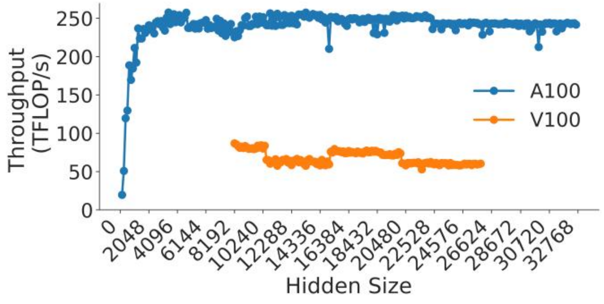
Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.



(a) MLP h to 4h Block



(b) MLP 4h to h Block

Fig. 10: Throughput (in teraFLOP/s) for multilayer perceptrons (MLP) for each transformer layer as a function of hidden dimension for $a = 128$.

Sizing: Attention

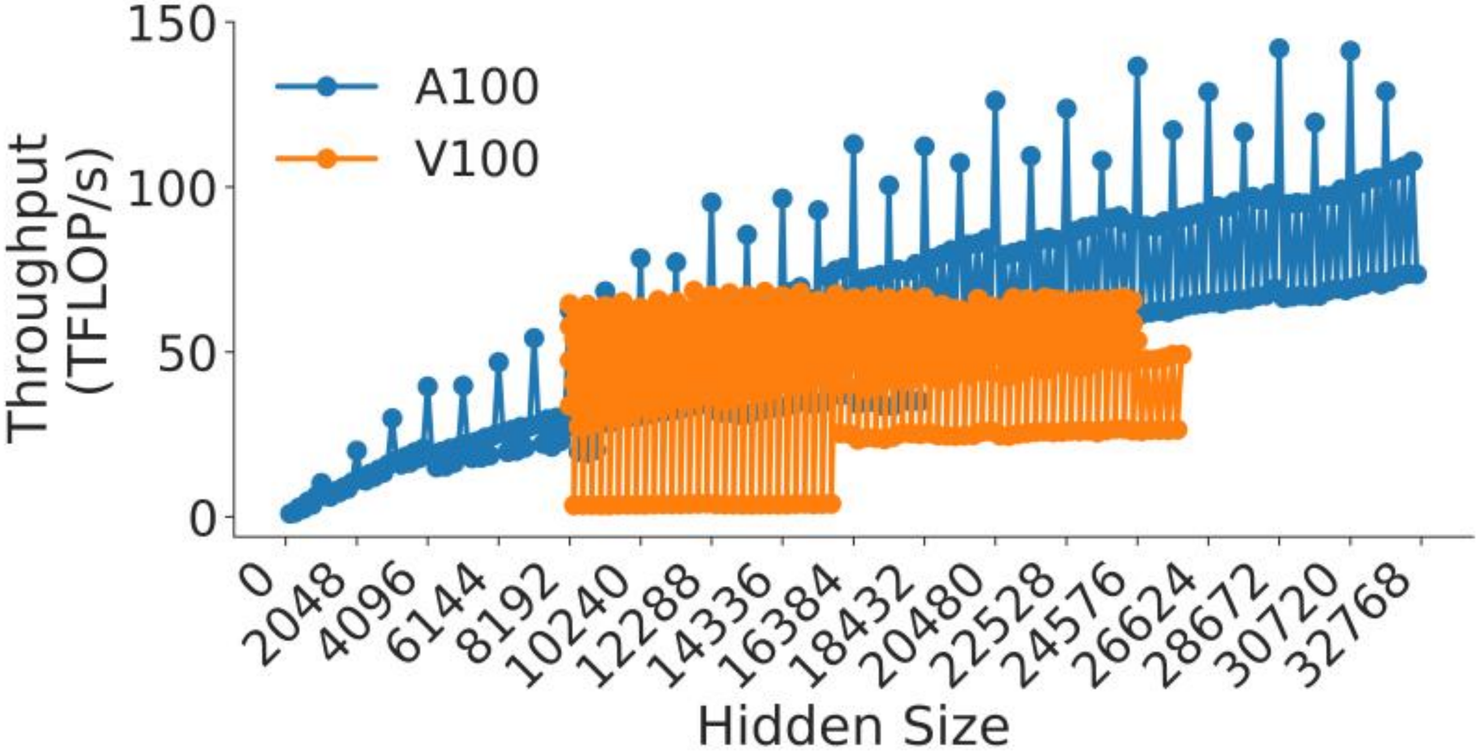


Fig. 17: Attention key-query score computation (KQ^T).

Sizing

- The two most sensitive kernels for the (non-flash!) attention calculation:
  - Attention over values (AOV)
  - Query-key-value transform (QKV)

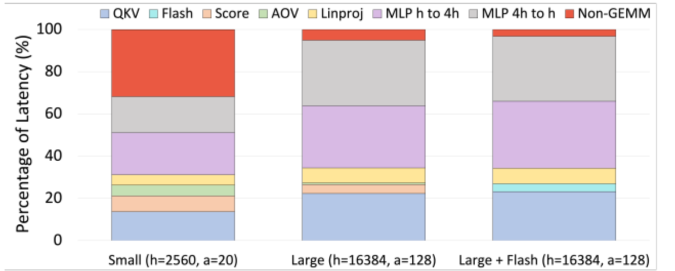


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.

Sizing

- The two most sensitive kernels for the (non-flash!) attention calculation:
  - Attention over values (AOV) - Query-key-value transform (QKV)
- These kernels are much more sensitive to model size
  - MLP gemms are (h,4h) whereas attention kernels get the dimension of h/a

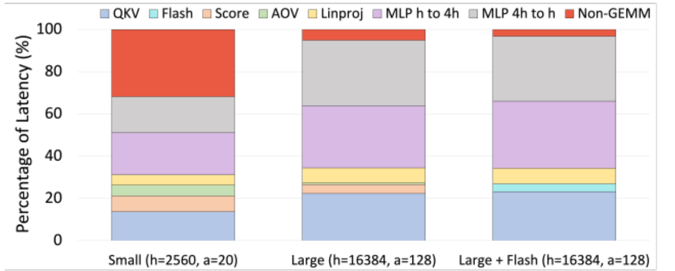
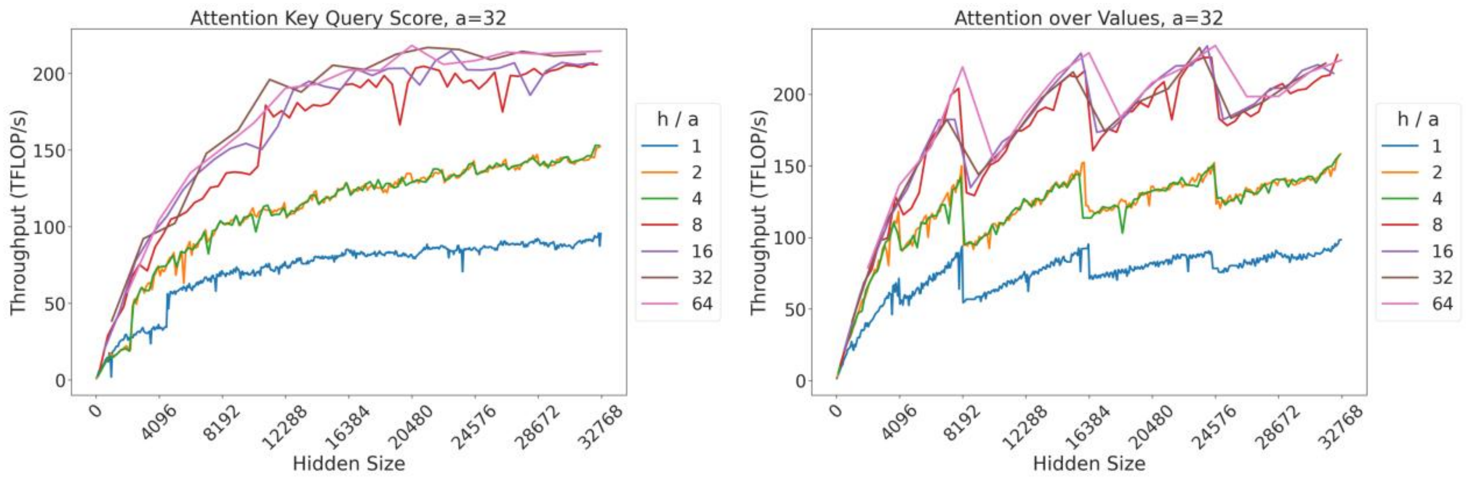


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.



(a) Attention key-query score GEMM throughput for 32 attention heads. (b) Attention over value GEMM throughput for 32 attention heads.

Fig. 7: Attention GEMM performance on A100 GPUs. Each plot is a single series (i.e. if we didn't split, there would be three regions with spikes), but split by the largest power of two that divides h/a to demonstrate that more powers of two leads to better performance up to $h/a = 64$.

Sizing

- The two most sensitive kernels for the (non-flash!) attention calculation:
  - Attention over values (AOV)
  - Query-key-value transform (QKV)
- These kernels are much more sensitive to model size
  - MLP gemms are (h,4h) whereas attention kernels get the dimension of h/a
- **Ensure h/a has as many factors of 2 as possible!**
  - Preferably h/a divisible by at least 64

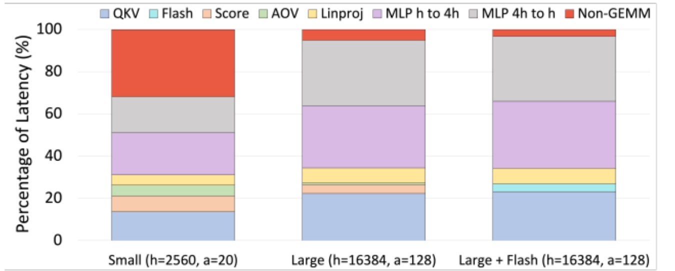
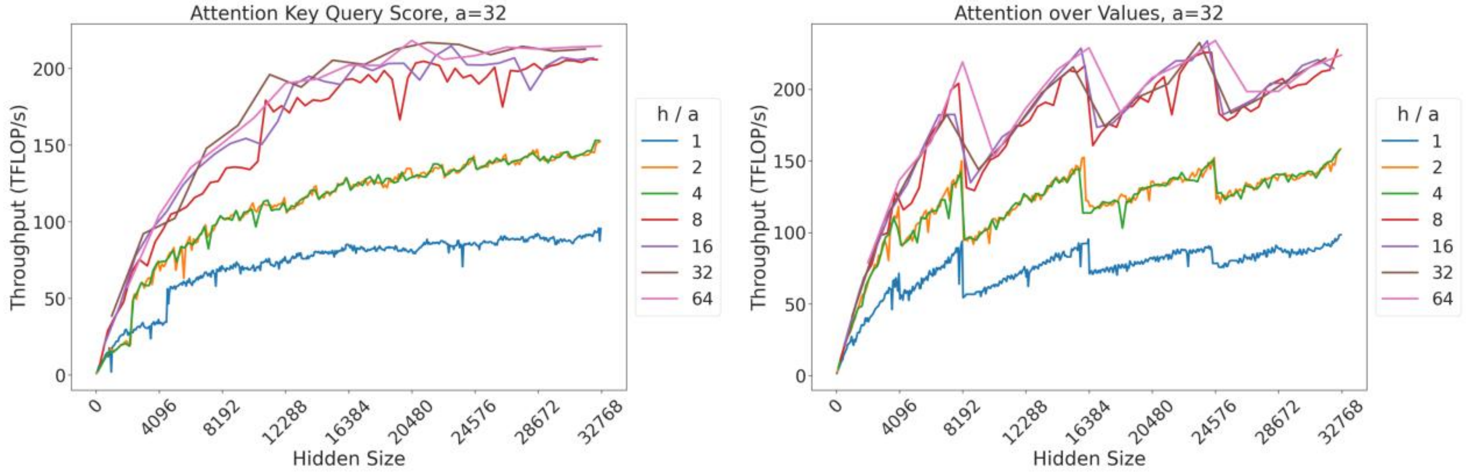


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.



(a) Attention key-query score GEMM throughput for 32 attention heads. (b) Attention over value GEMM throughput for 32 attention heads.

Fig. 7: Attention GEMM performance on A100 GPUs. Each plot is a single series (i.e. if we didn't split, there would be three regions with spikes), but split by the largest power of two that divides h/a to demonstrate that more powers of two leads to better performance up to $h/a = 64$.

Sizing

- The two most sensitive kernels for the (non-flash!) attention calculation:
  - Attention over values (AOV)
  - Query-key-value transform (QKV)
- These kernels are much more sensitive to model size
  - MLP gemms are (h,4h) whereas attention kernels get the dimension of h/a
- **Ensure h/a has as many factors of 2 as possible!**
  - Preferably h/a divisible by at least 64

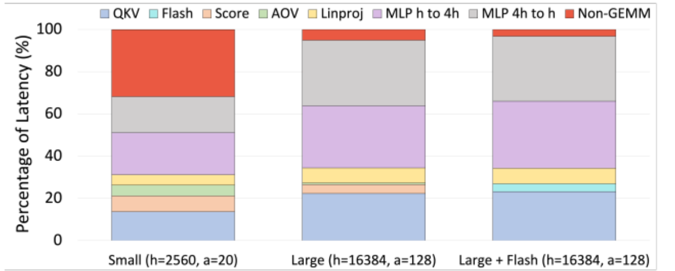
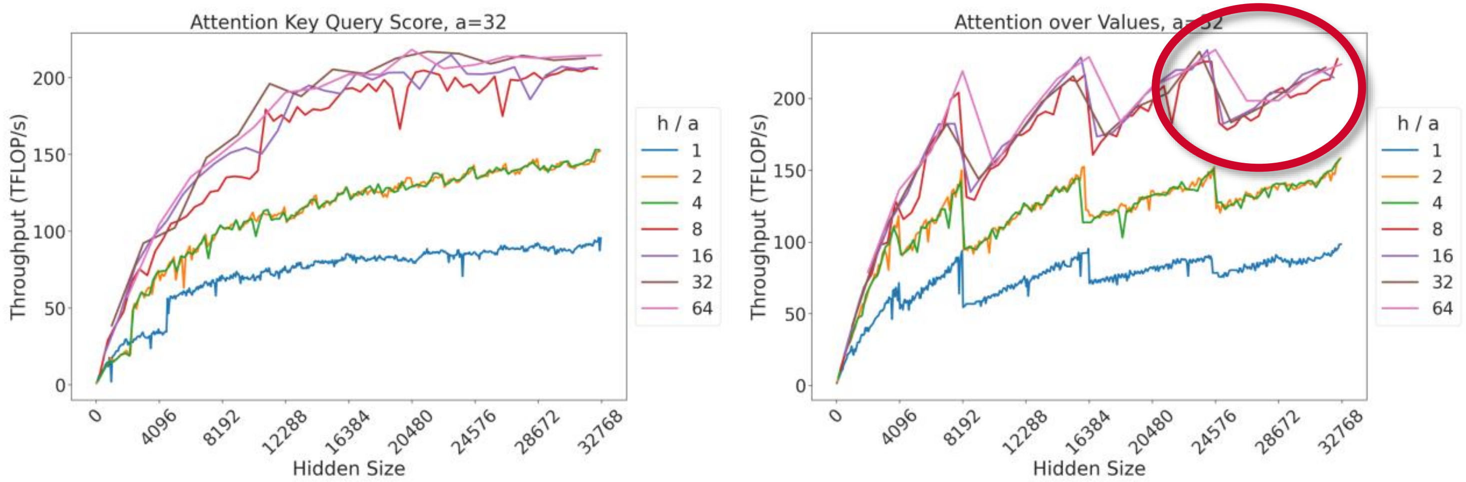


Fig. 11: The proportion of latency of each GEMM module in a medium sized transformer model.

What are these waves?

Module	GEMM Size	Figure
Input Embedding	—	—
Layer Norm 1	—	—
QKV Transform	$(b \cdot s, h) \times (h, \frac{3h}{t})$	16
Attention Score	$(\frac{b \cdot a}{t}, s, \frac{h}{a}) \times (\frac{b \cdot a}{t}, \frac{h}{a}, s)$	7a 8
Attn over Value	$(\frac{b \cdot a}{t}, s, s) \times (\frac{b \cdot a}{t}, s, \frac{h}{a})$	7b 9
Linear Projection	$(b \cdot s, \frac{h}{t}) \times (\frac{h}{t}, h)$	19
Layer Norm 2	—	—
MLP h to $4h$	$(b \cdot s, h) \times (h, \frac{4h}{t})$	10a
MLP $4h$ to h	$(b \cdot s, \frac{4h}{t}) \times (\frac{4h}{t}, h)$	10b
Linear Output	$(b \cdot s, v) \times (v, h)$	20

TABLE II: Summary of operators in the transformer layer considered in this paper, along with the size of the GEMMs used to execute these operators.

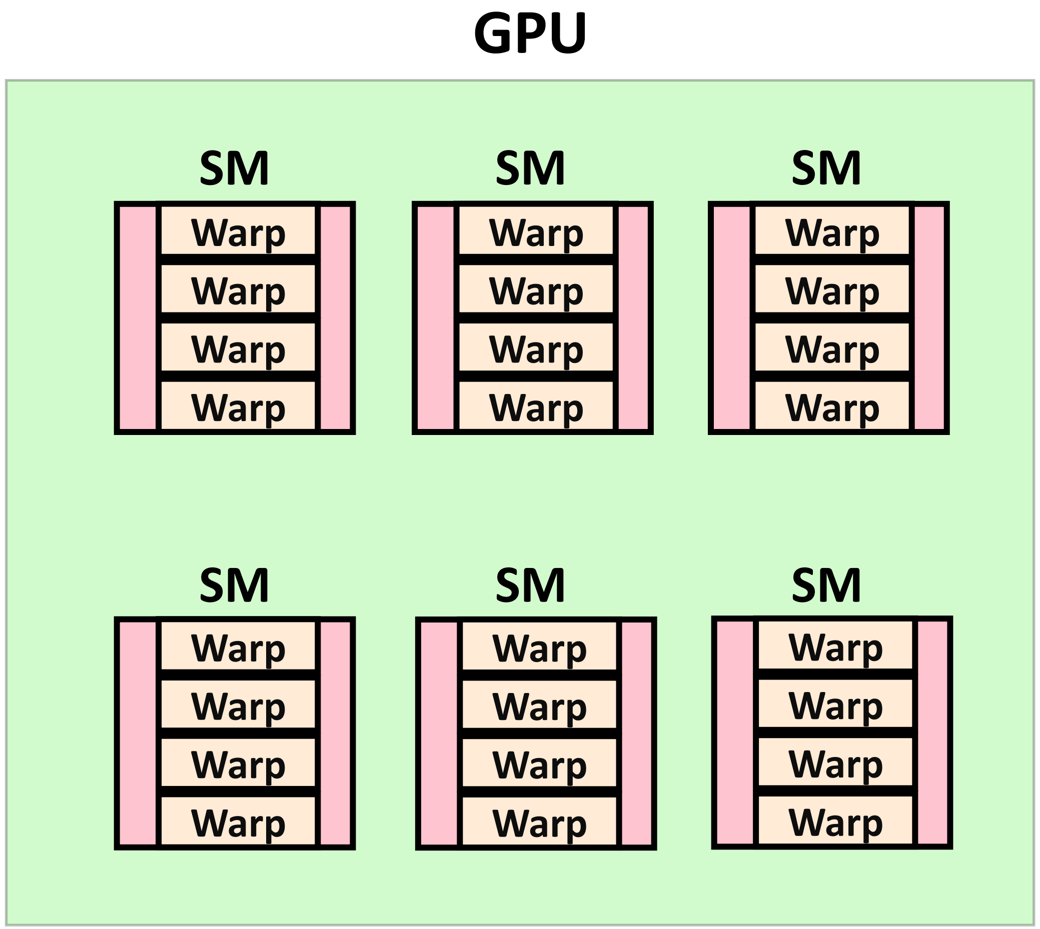


(a) Attention key-query score GEMM throughput for 32 attention heads. (b) Attention over value GEMM throughput for 32 attention heads.

Fig. 7: Attention GEMM performance on A100 GPUs. Each plot is a single series (i.e. if we didn't split, there would be three regions with spikes), but split by the largest power of two that divides h/a to demonstrate that more powers of two leads to better performance up to $h/a = 64$.

Background (alt. why are GPUs amazing for parallelism?)

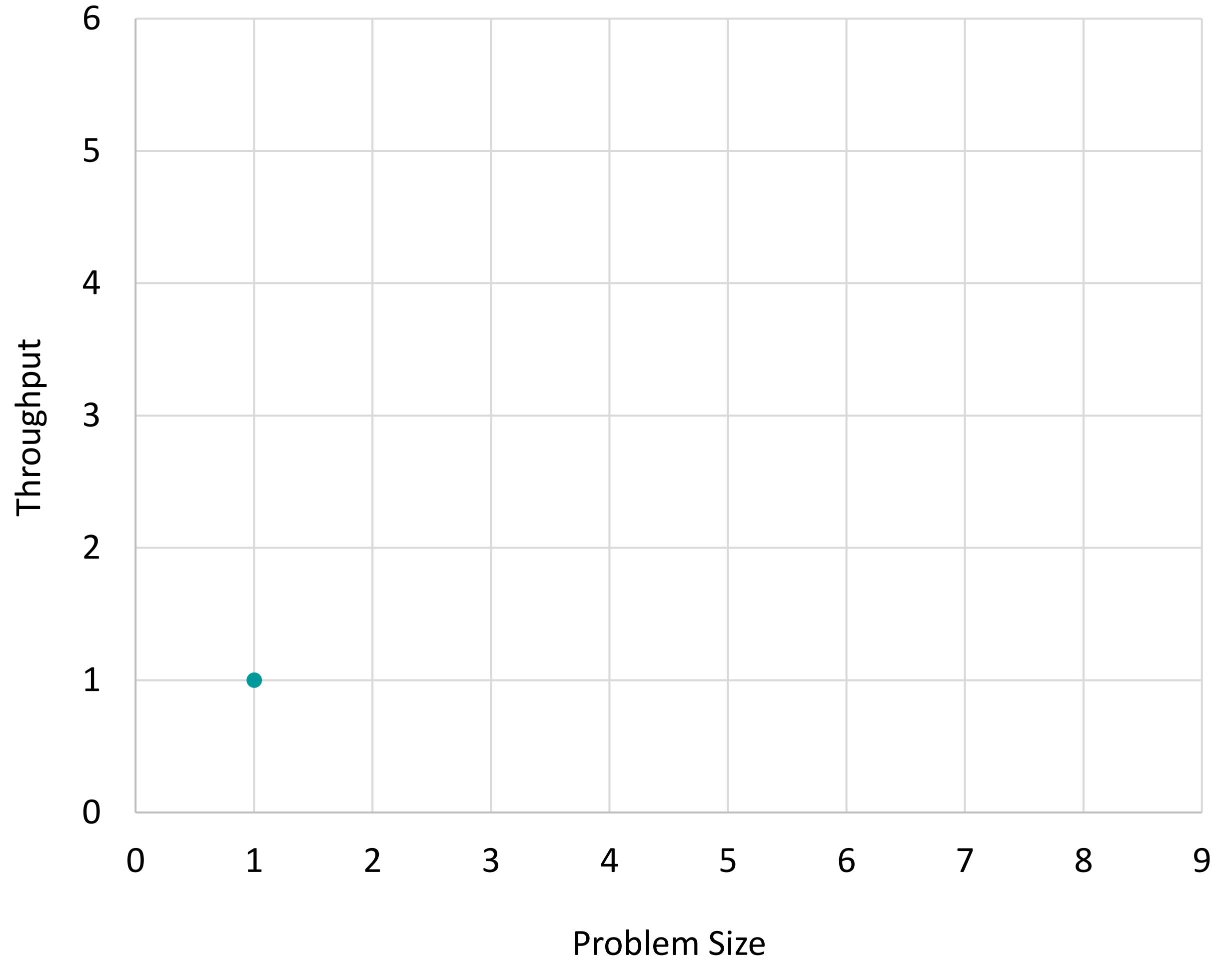
- GPUs are composed of streaming multiprocessors (SMs)
  - SMs contain *warps* (i.e. thread blocks). Warps contain *cores*.
  - SMs run in parallel
- Keep all those *cores* busy!
  - GPU Occupancy = $\text{Sum}_i(\text{SM}_i \text{ Occupancy}) / \#\text{SMs}$
  - Want GPU occupancy to be high!
    - Fill each SM with enough* warps to hide instruction latency
    - Fill aggregate SMs with enough warps to keep cores busy

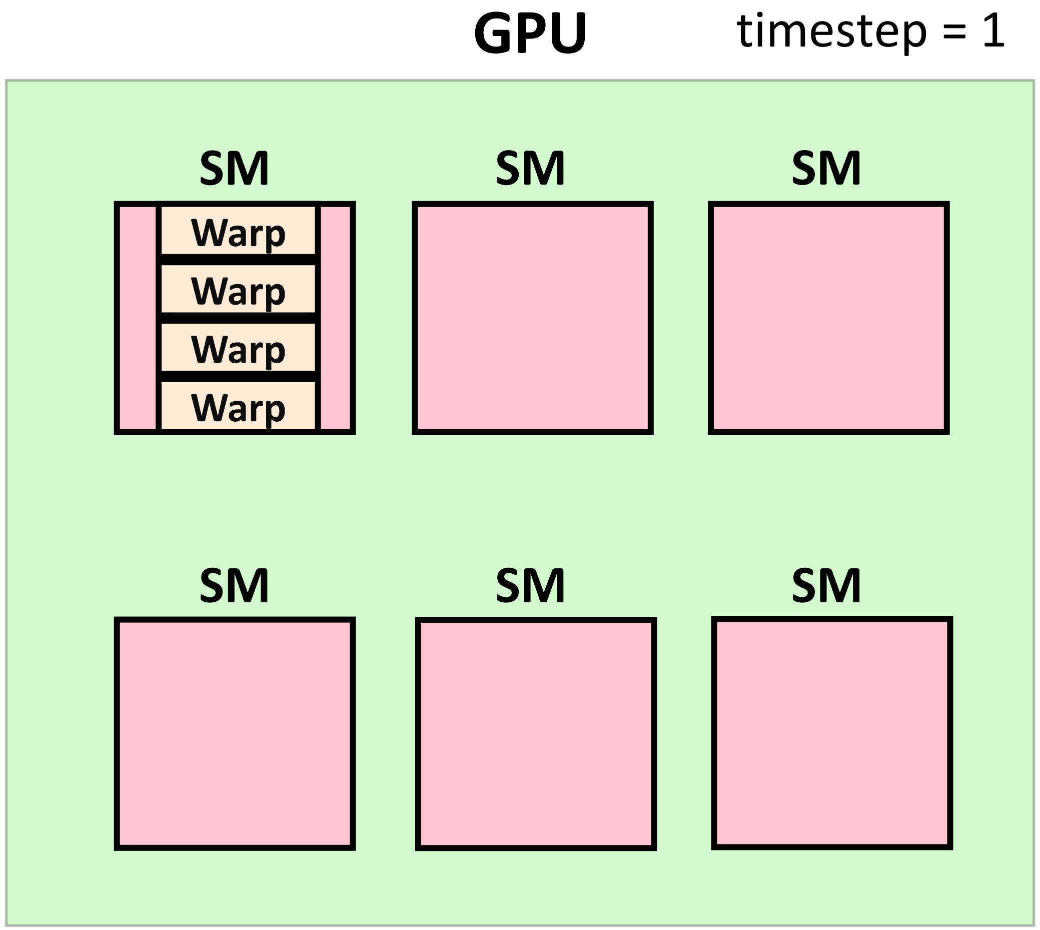


* Too low, SMs are idle or latency between instructions isn't hidden (really bad!). Too high, not enough cores per warp (kinda bad)

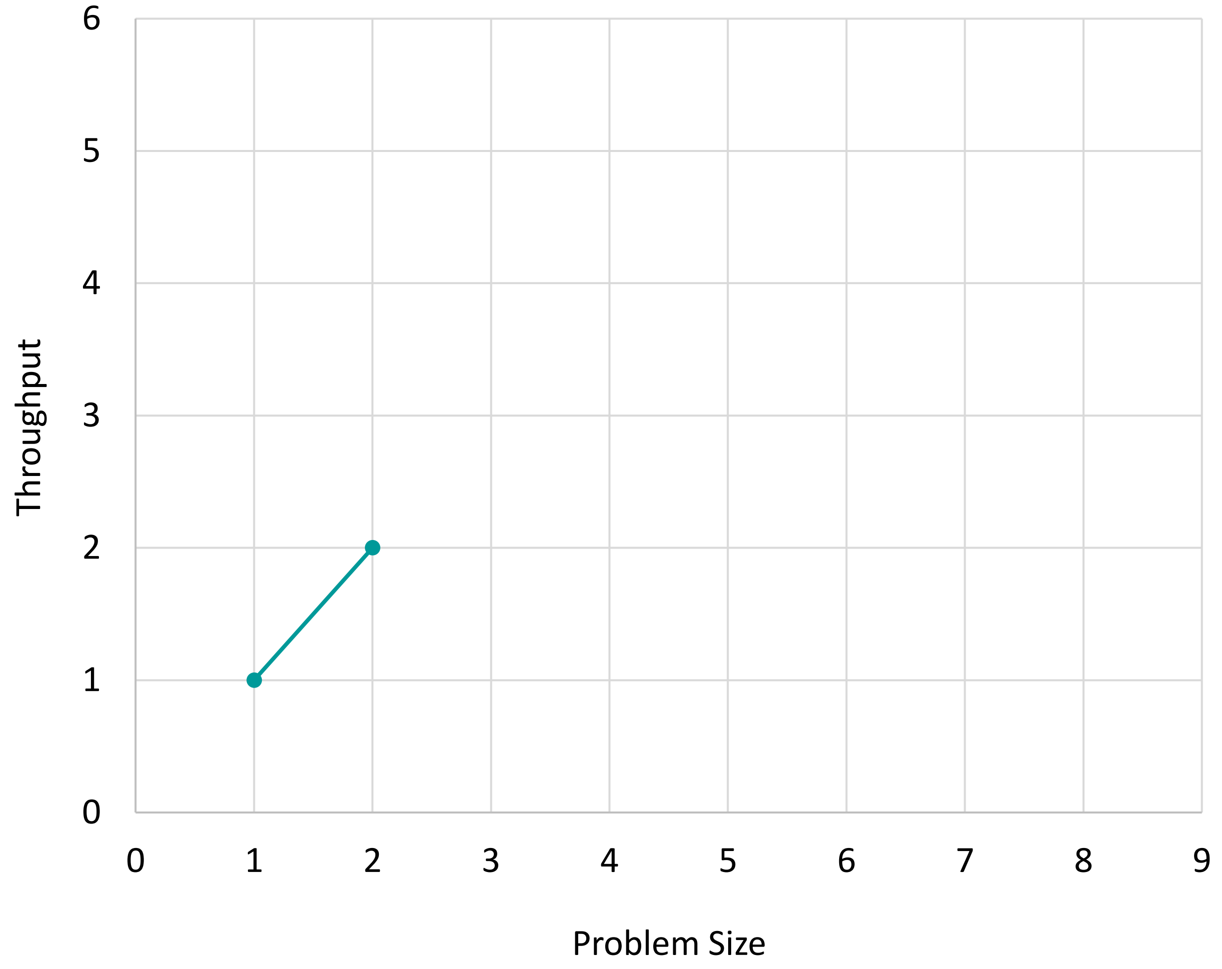
Background (Wave Quantization)

Total units of work to compute = 1

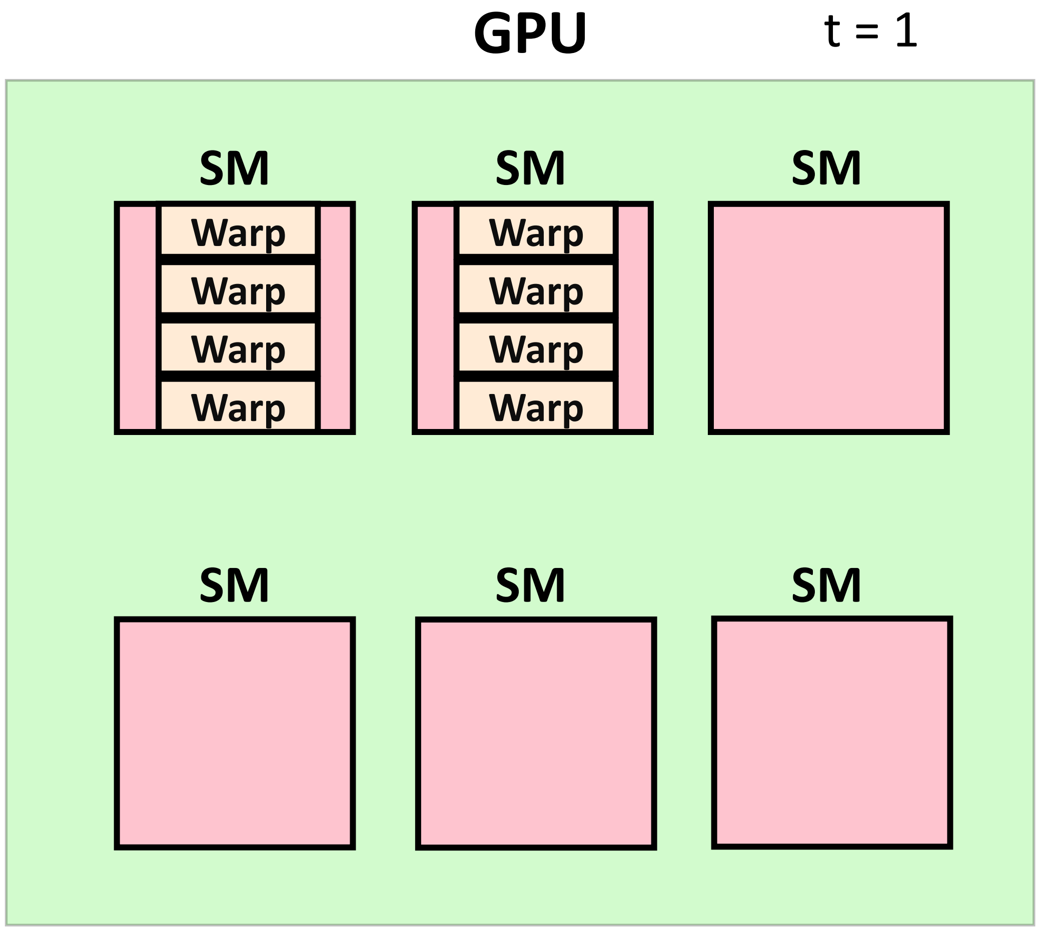




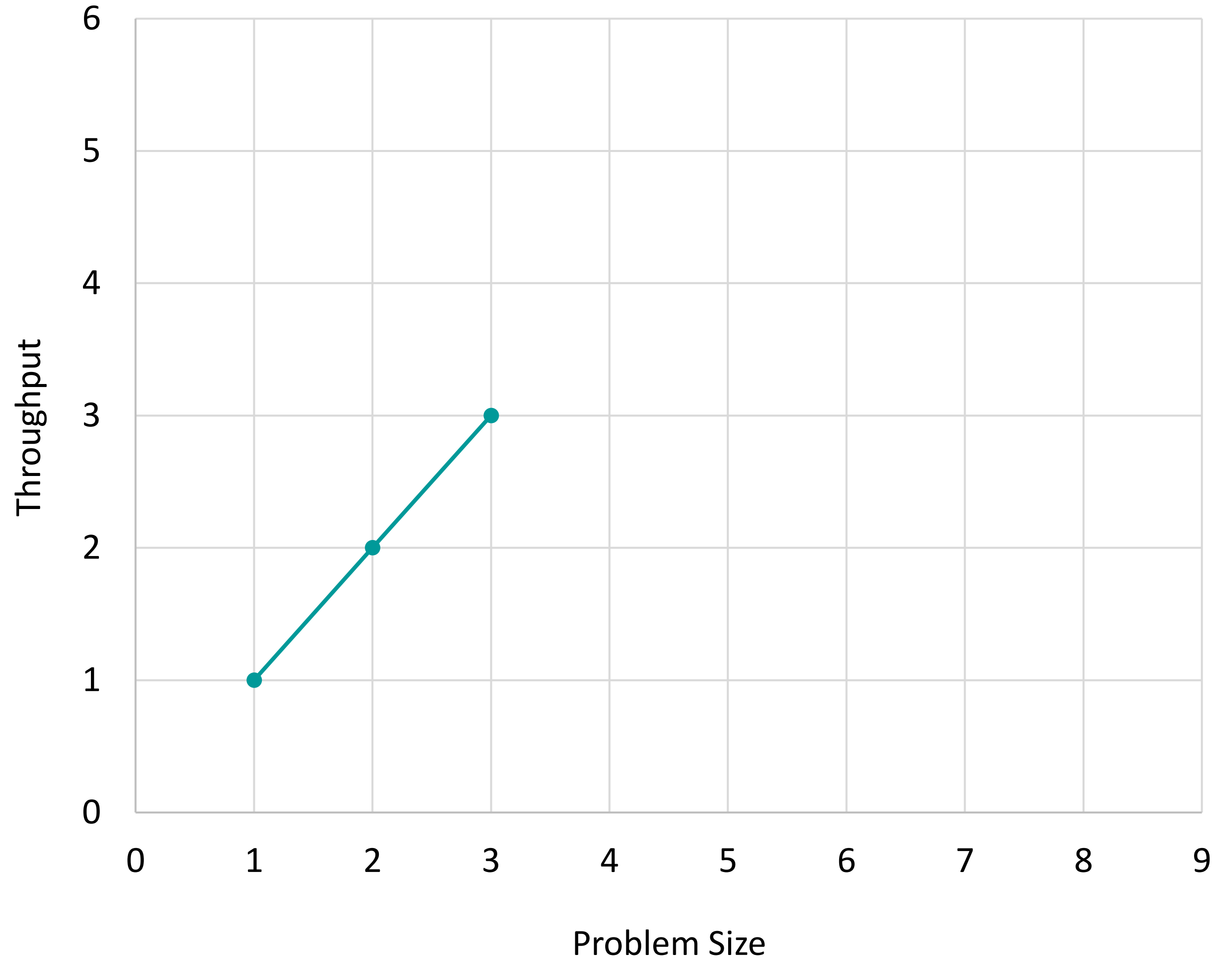
Background (Wave Quantization)



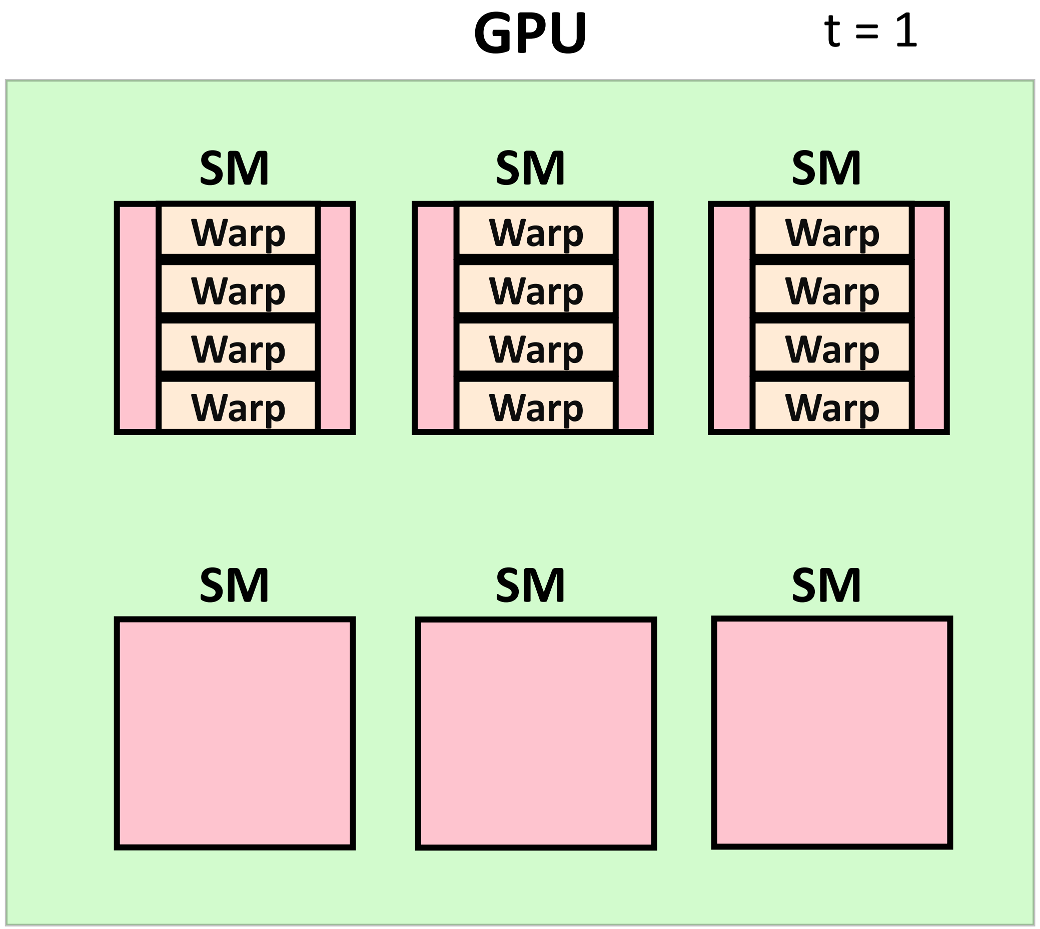
Total units to compute = 2



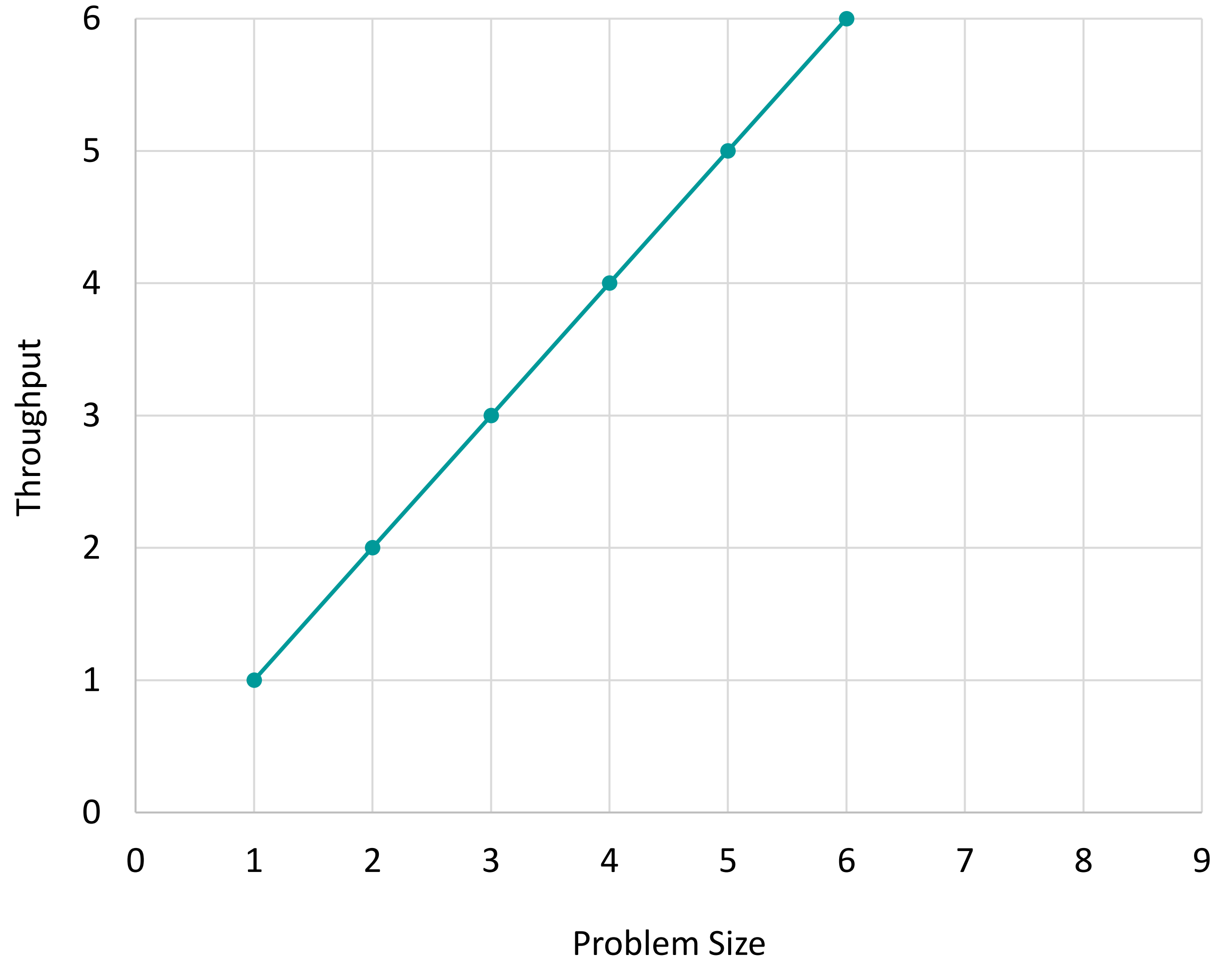
Background (Wave Quantization)



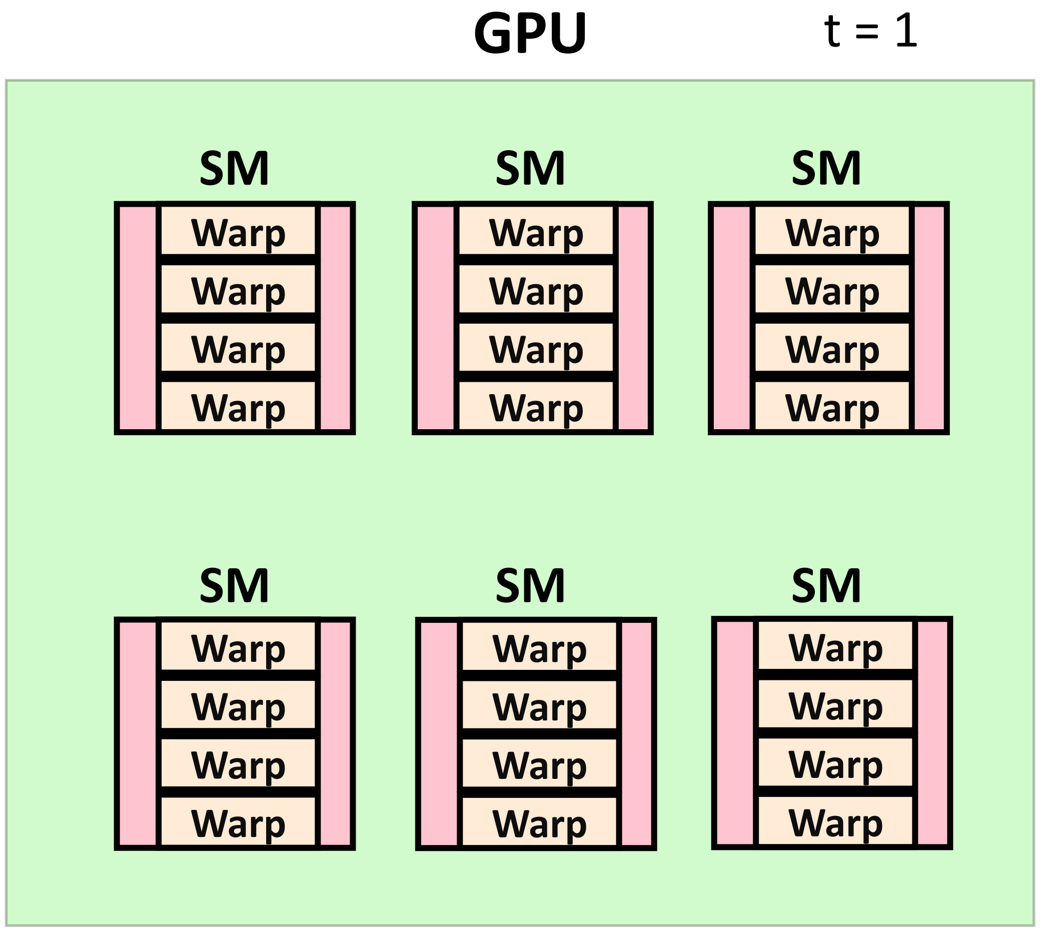
Total units to compute = 3



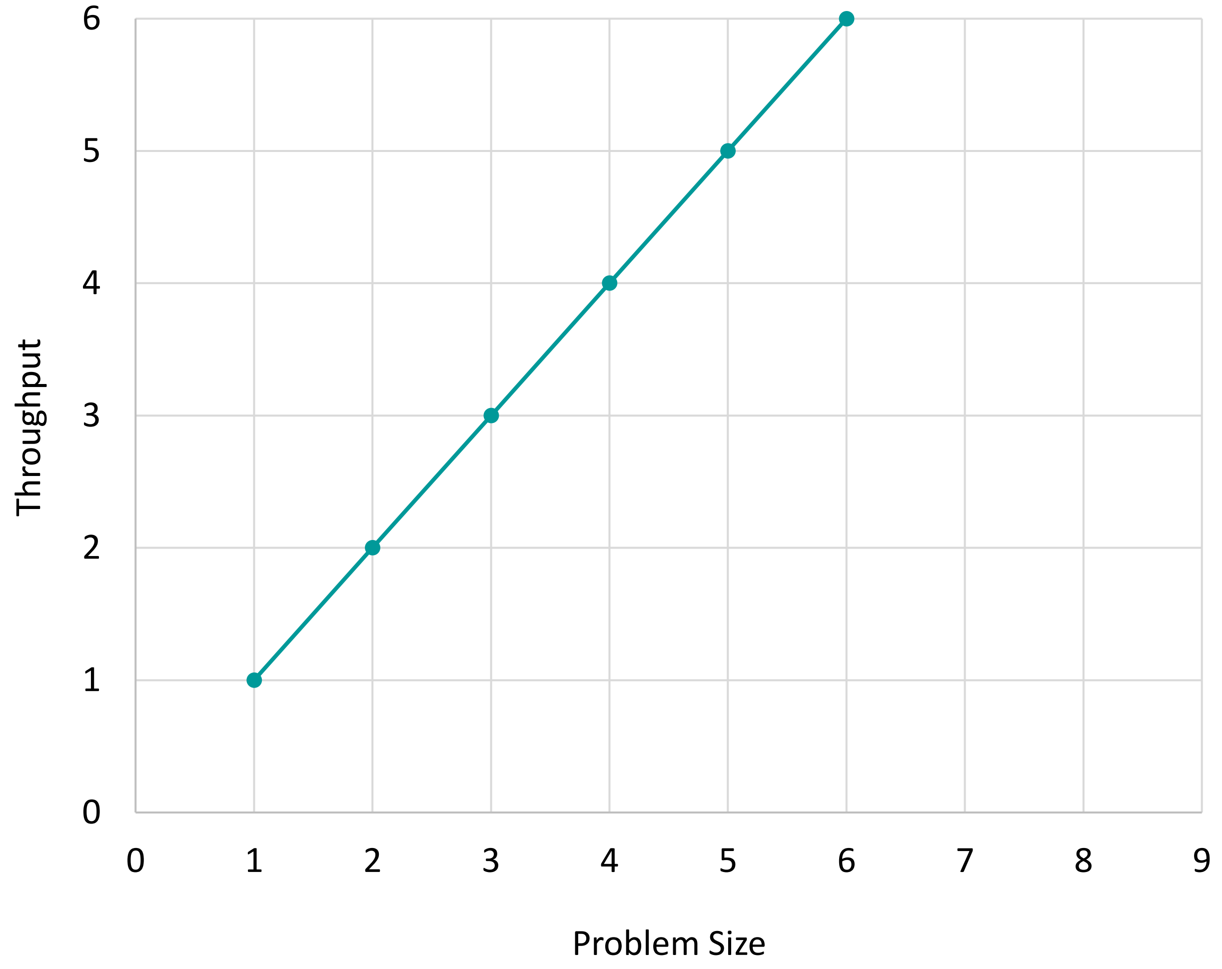
Background (Wave Quantization)

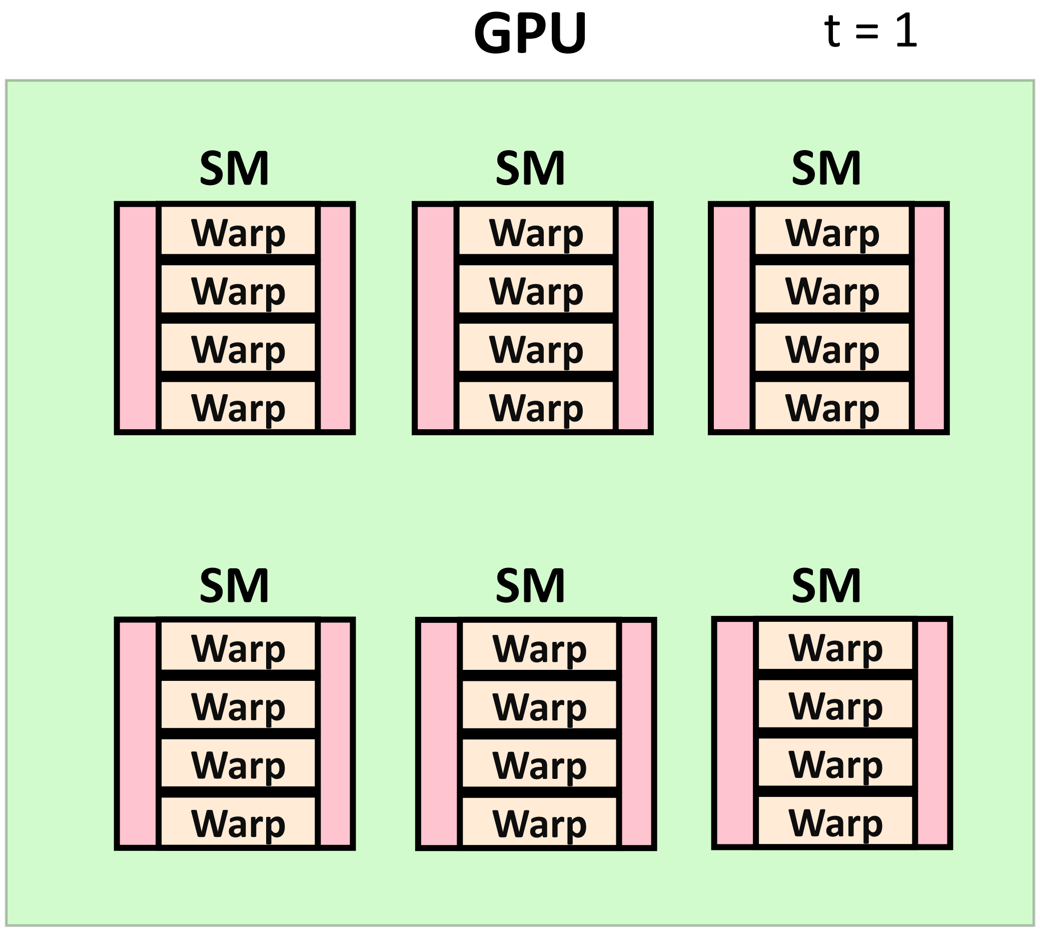


Total units to compute = 6

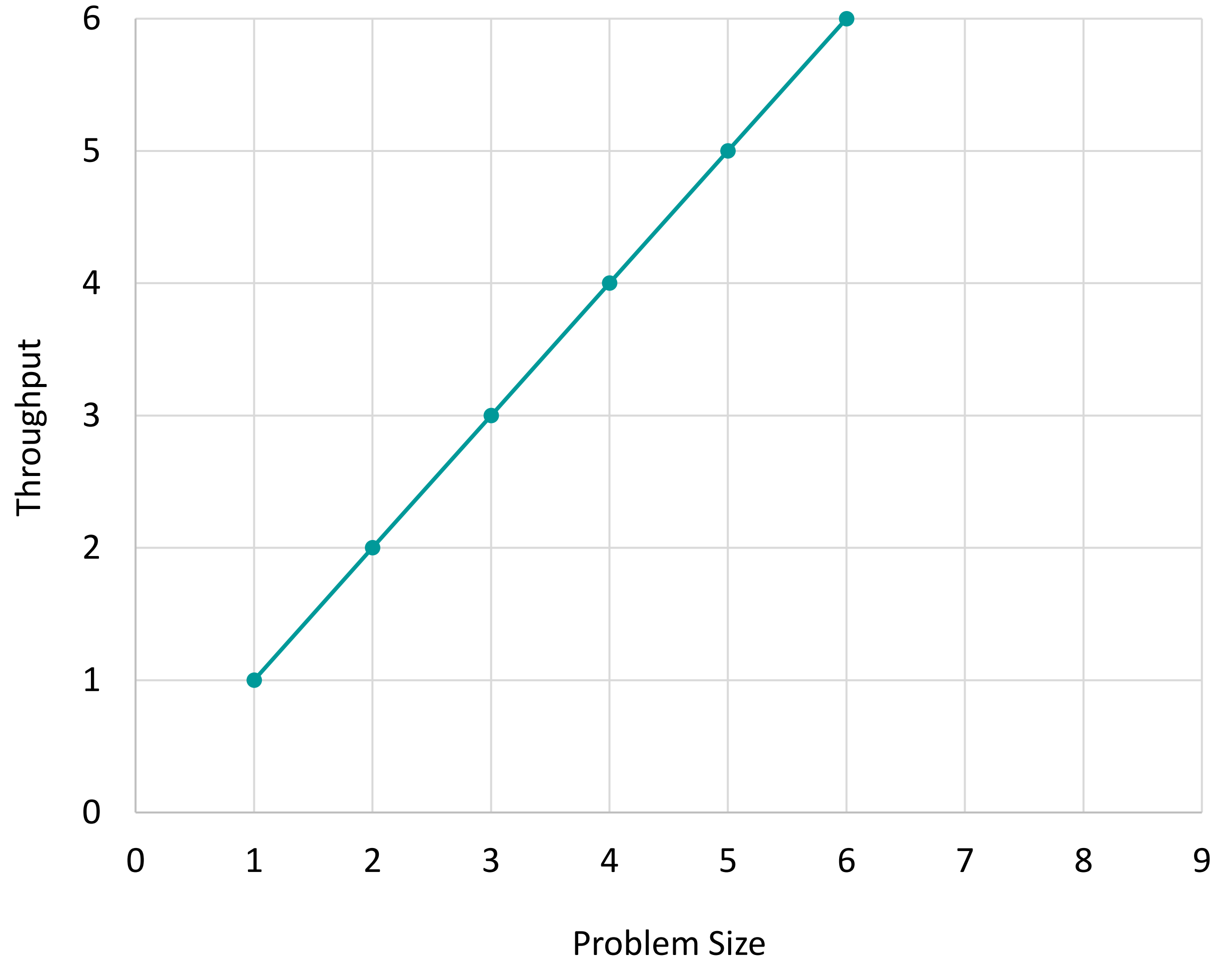


Background (Wave Quantization)

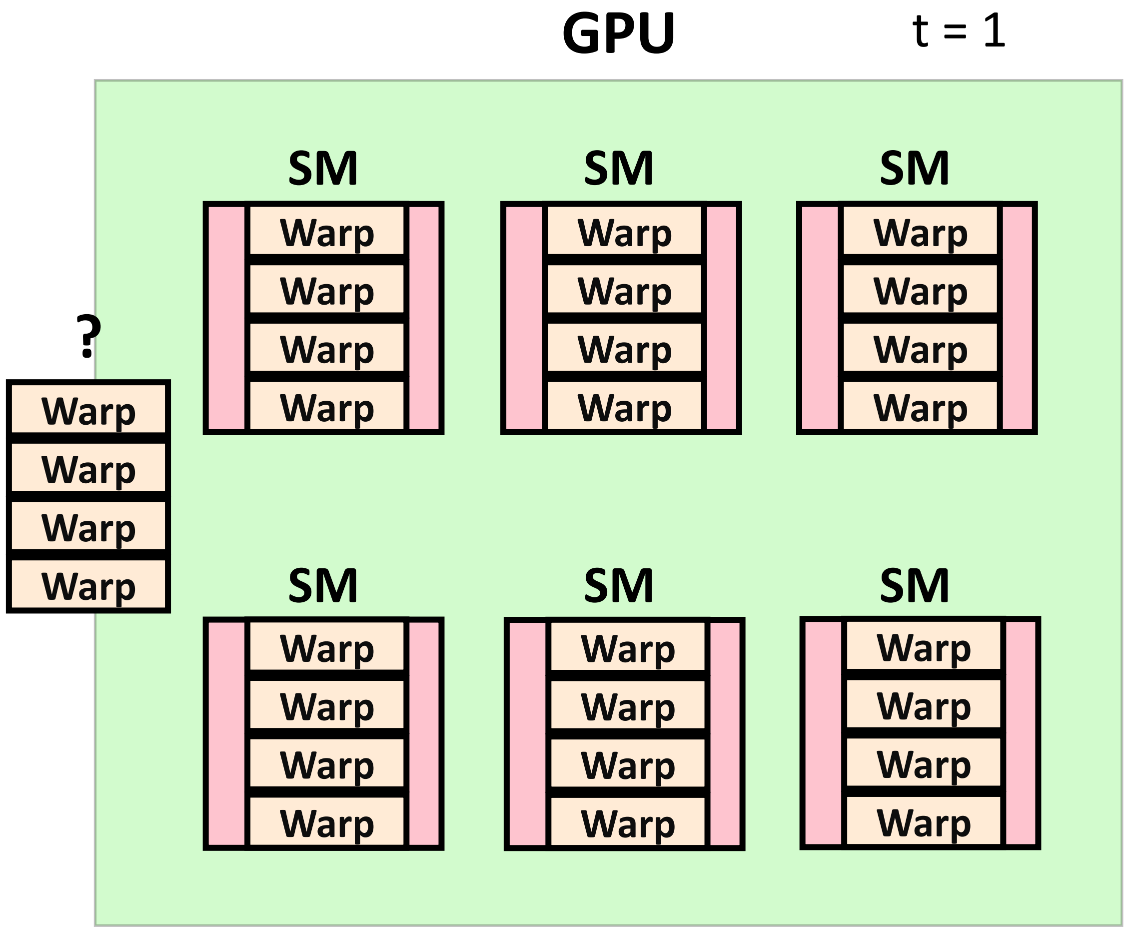




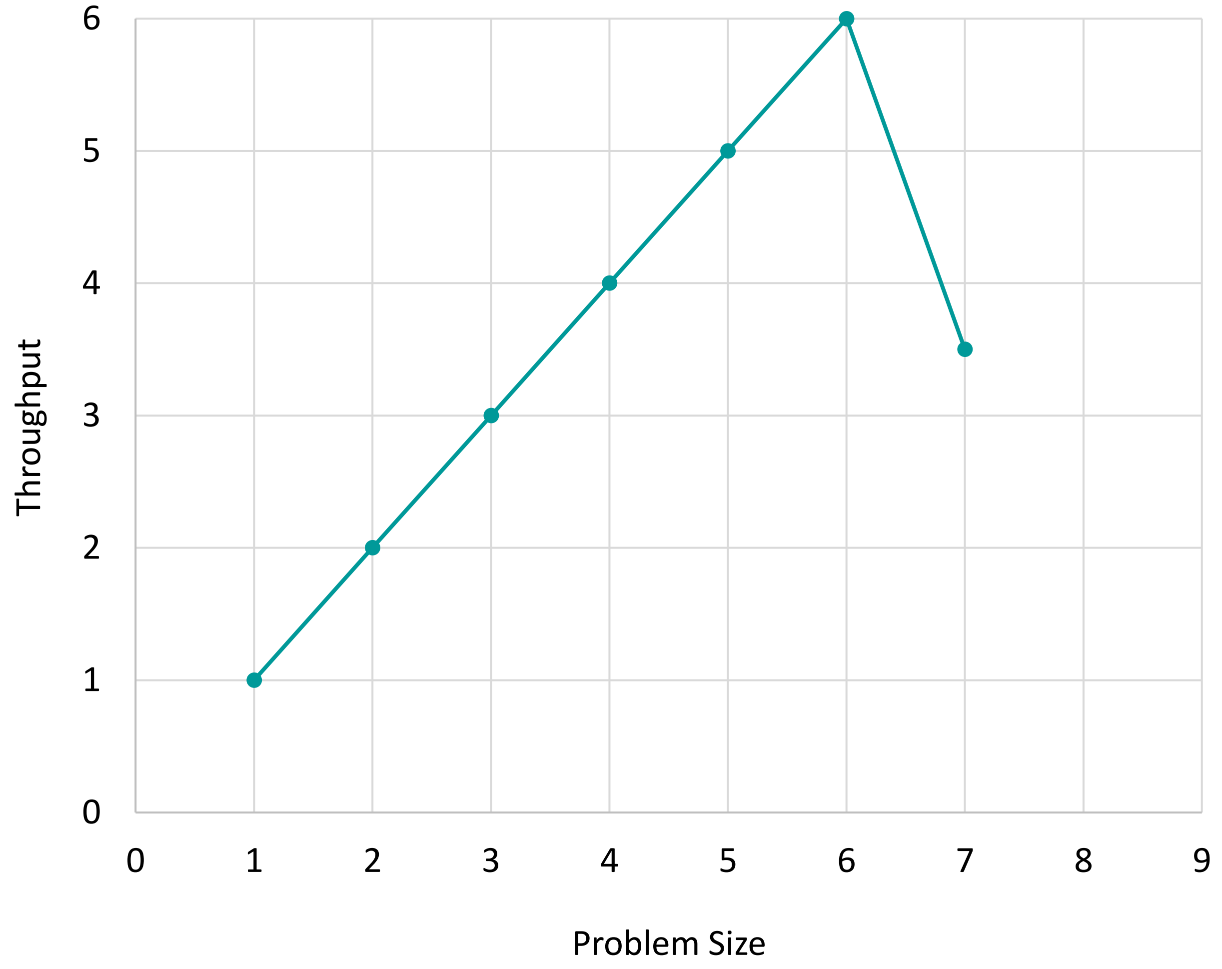
Background (Wave Quantization)



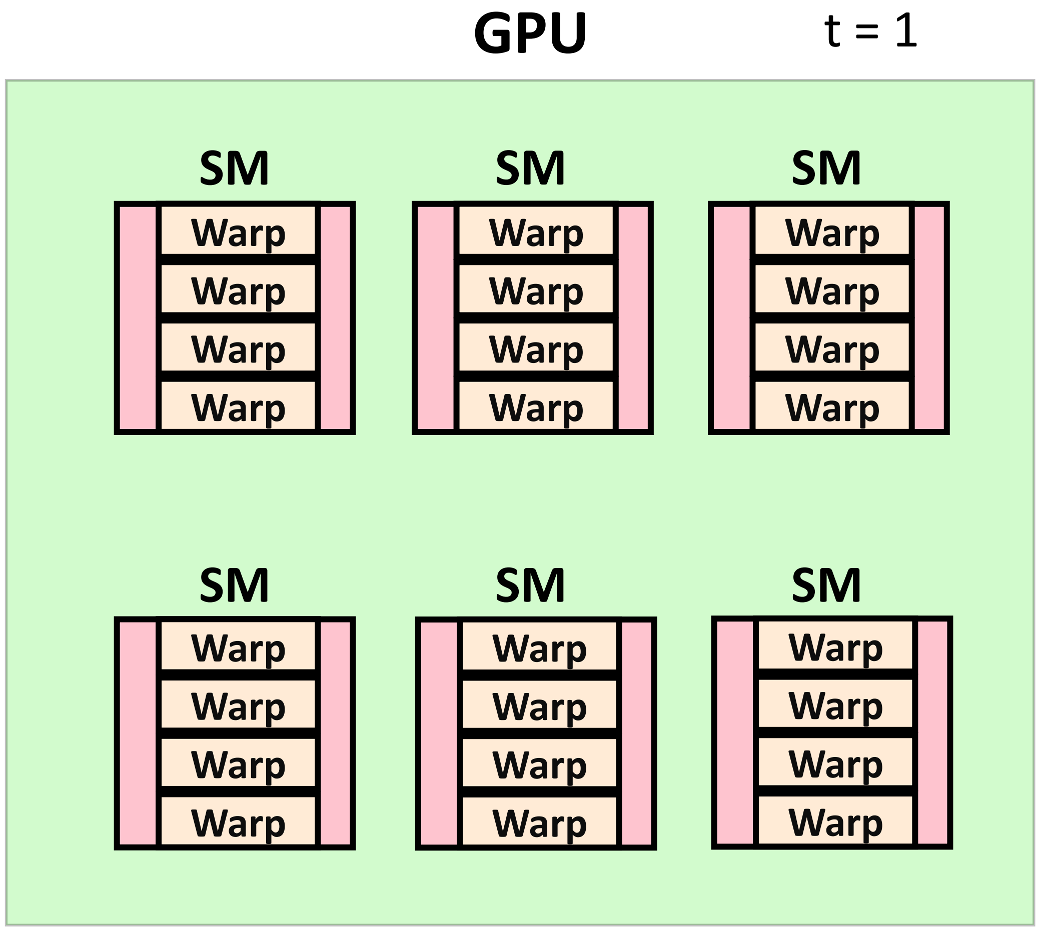
Total units to compute = 7?



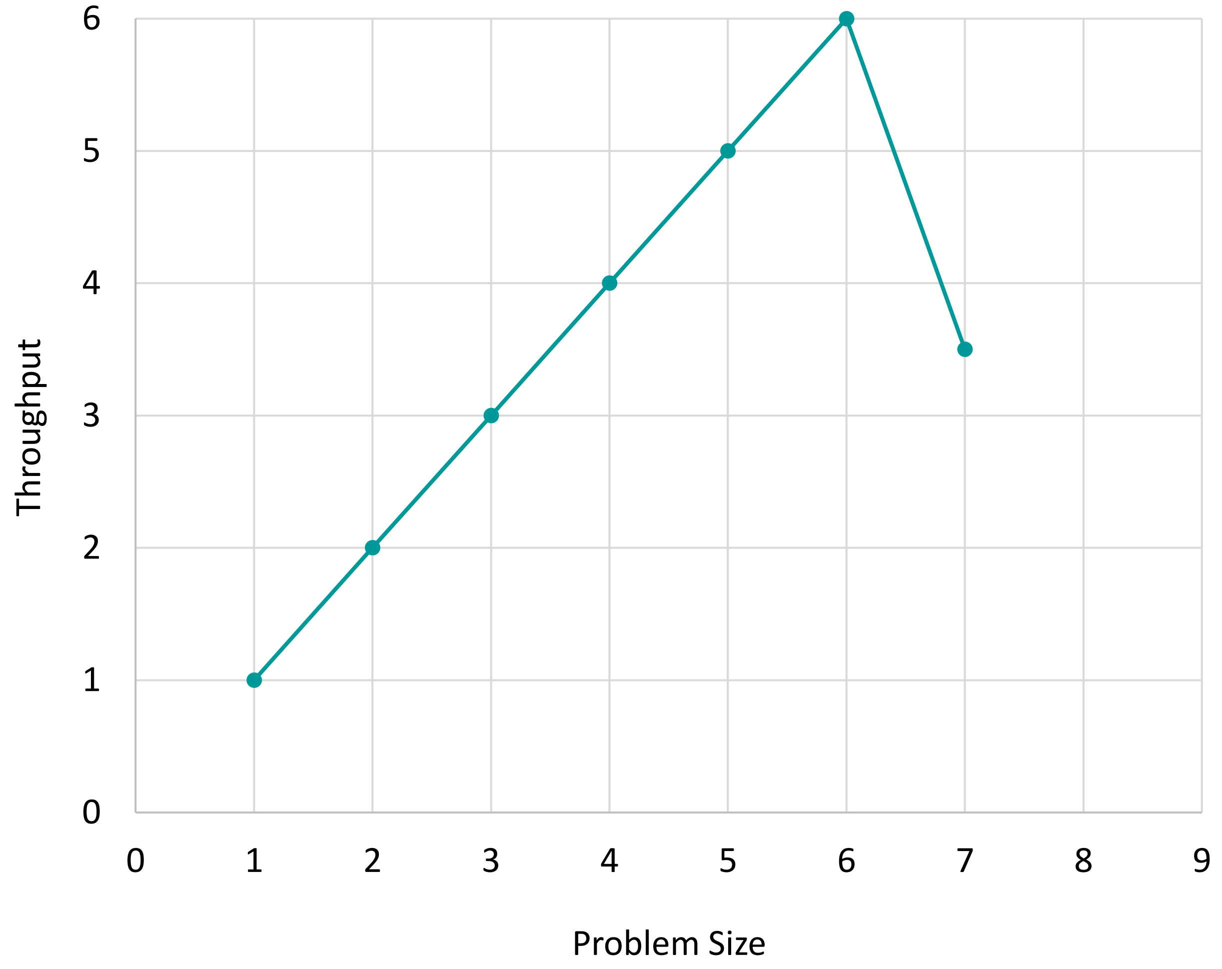
Background (Wave Quantization)



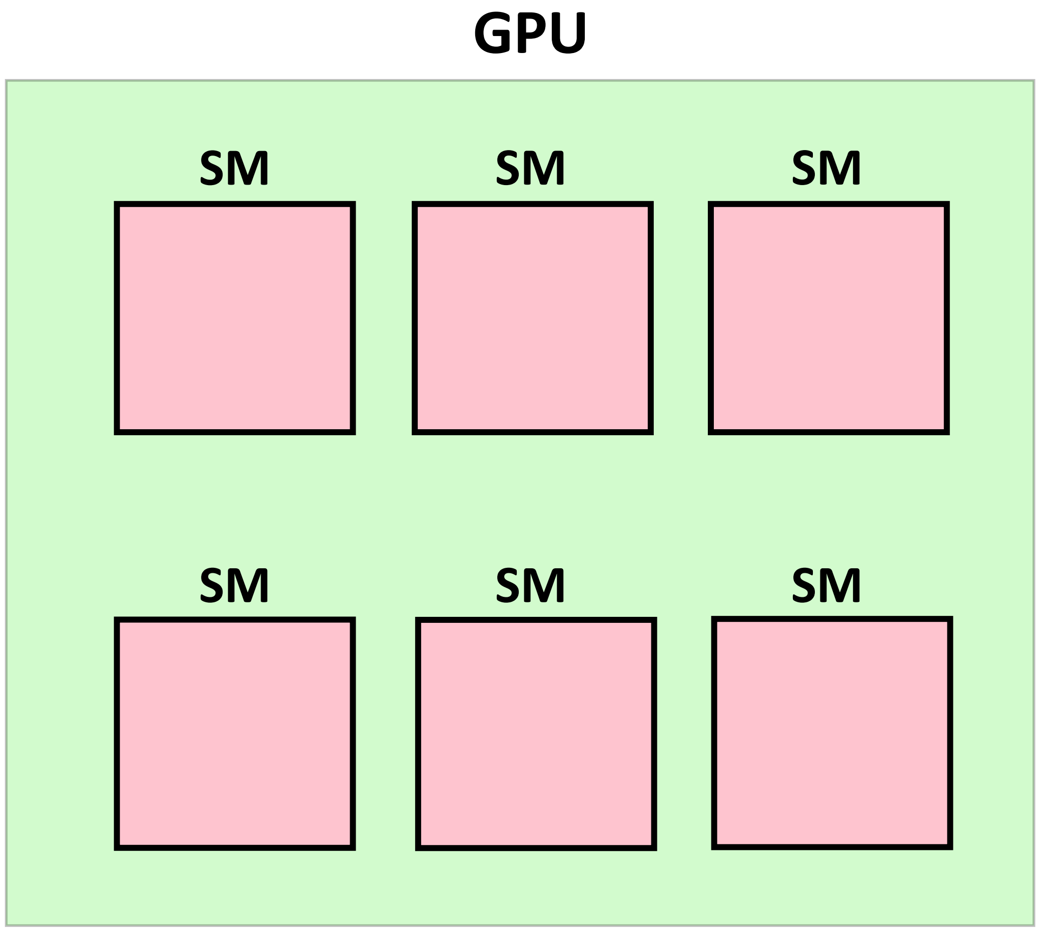
Total units to compute = 7



Background (Wave Quantization)

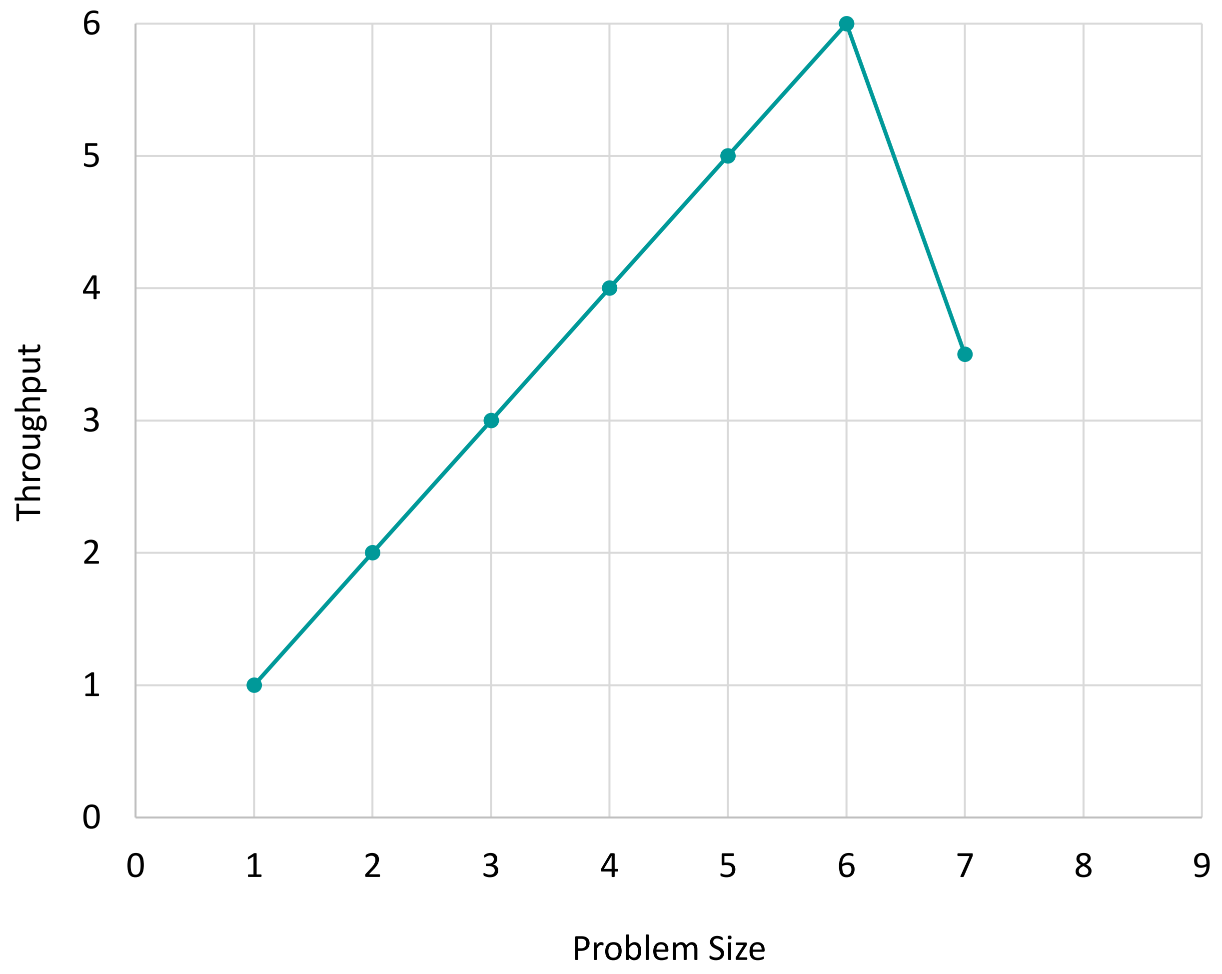


Total units to compute = 7

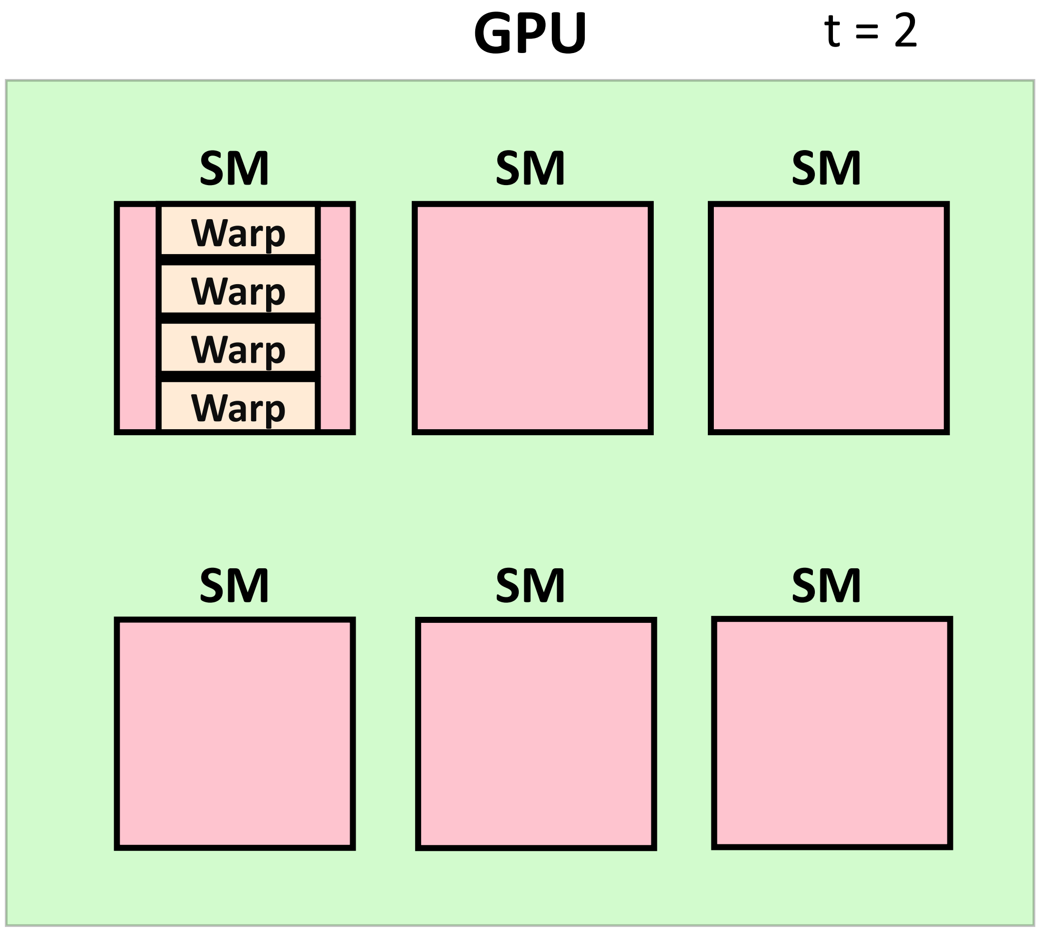


Background (Wave Quantization)

Throughput = (6 + 1) / 2 = 3.5

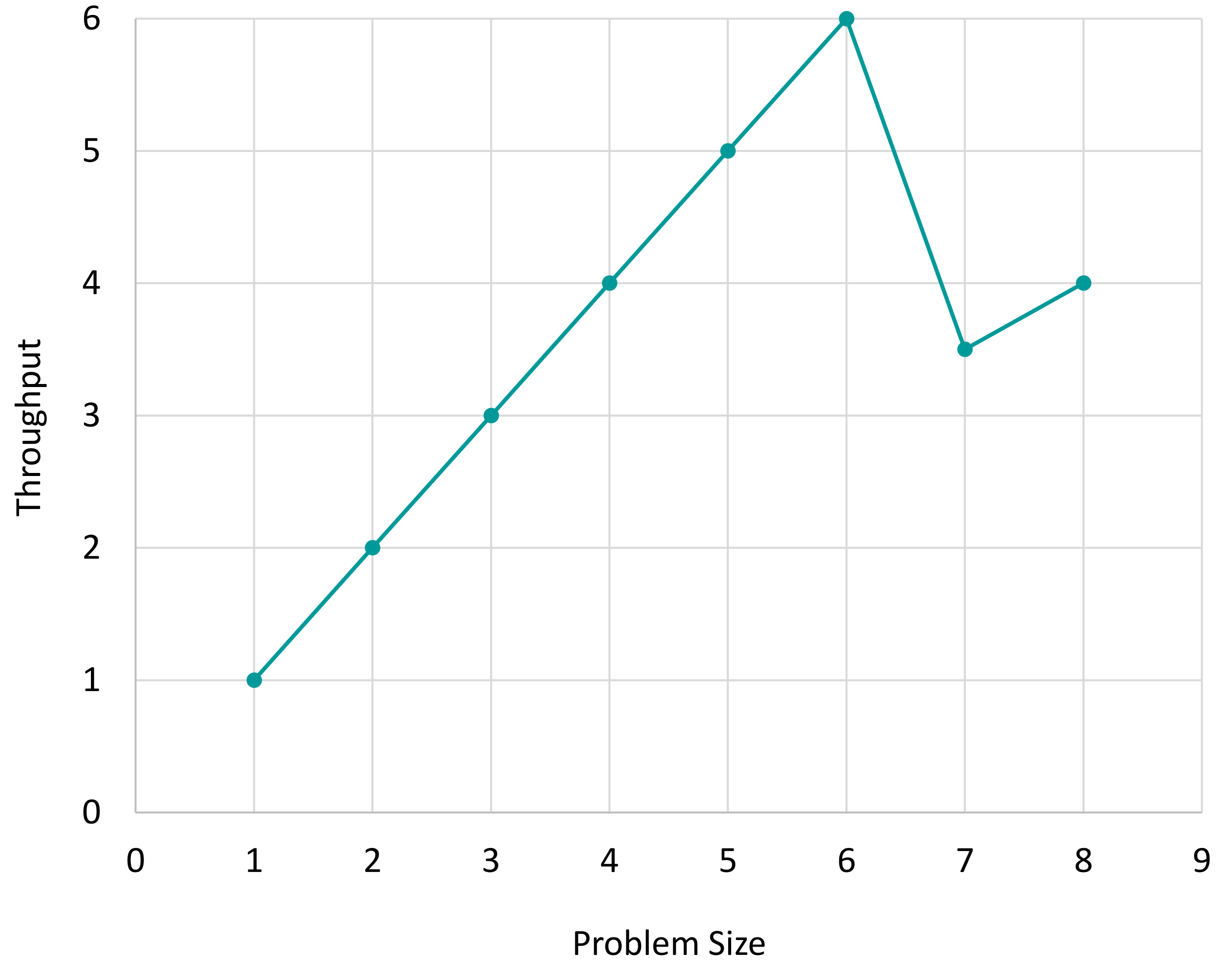


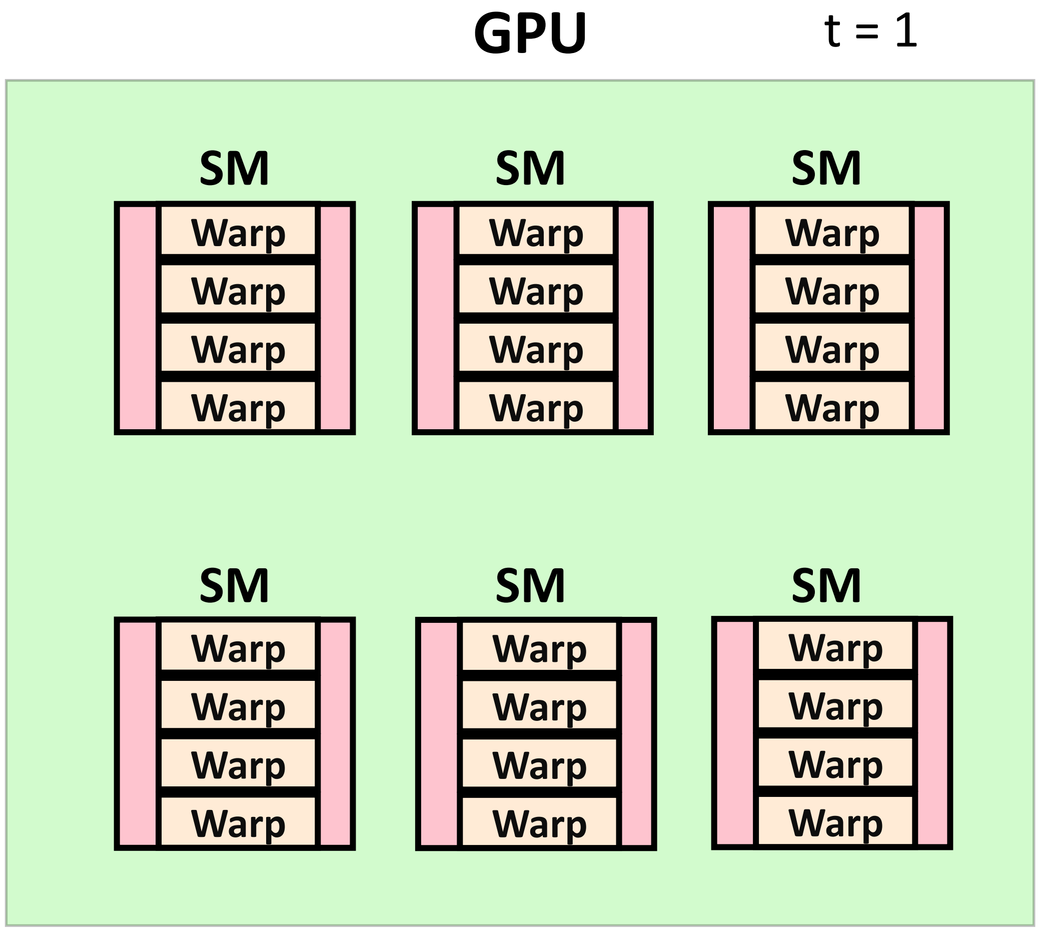
Total units to compute = 7



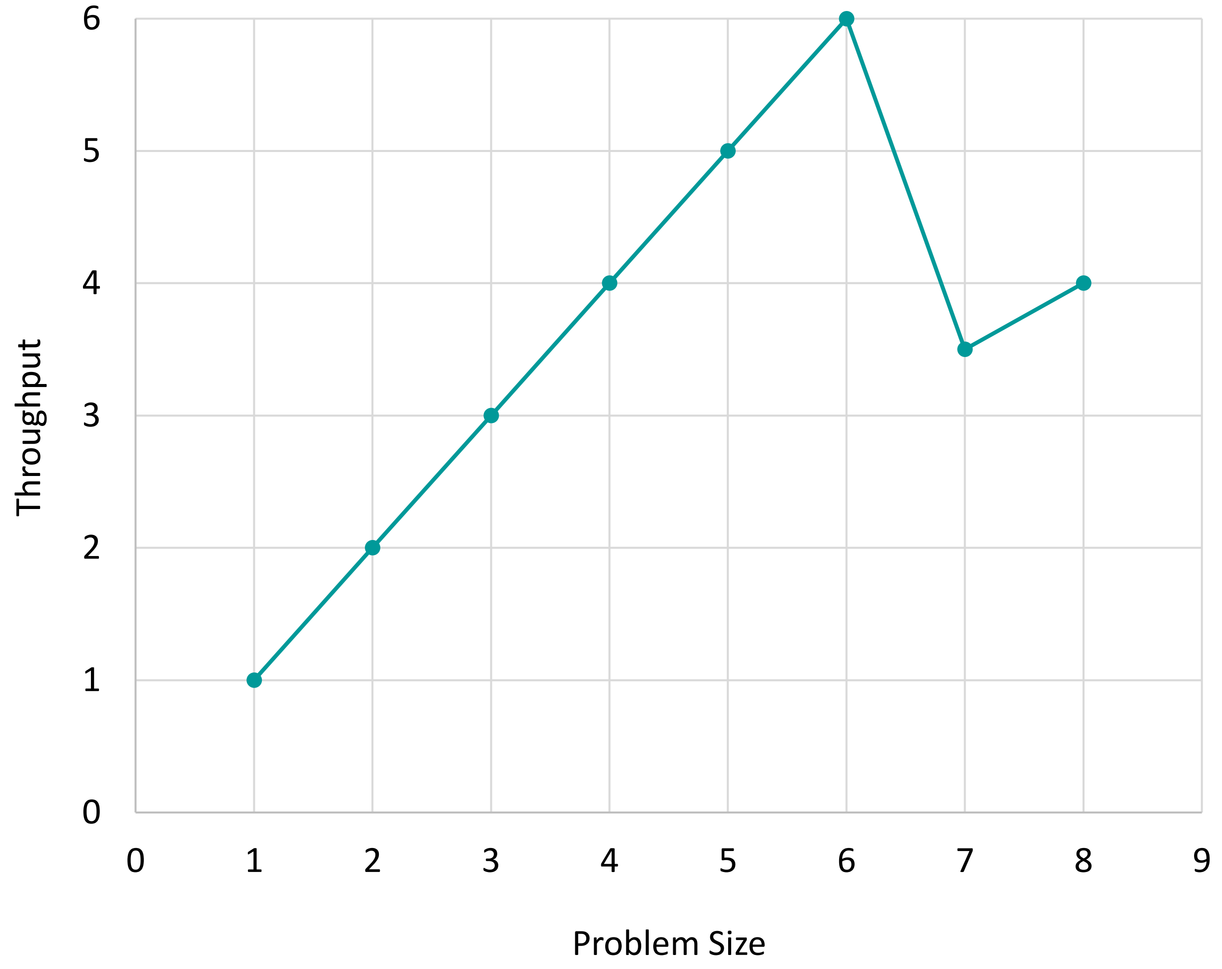
Background (Wave Quantization)

Total units to compute = 8

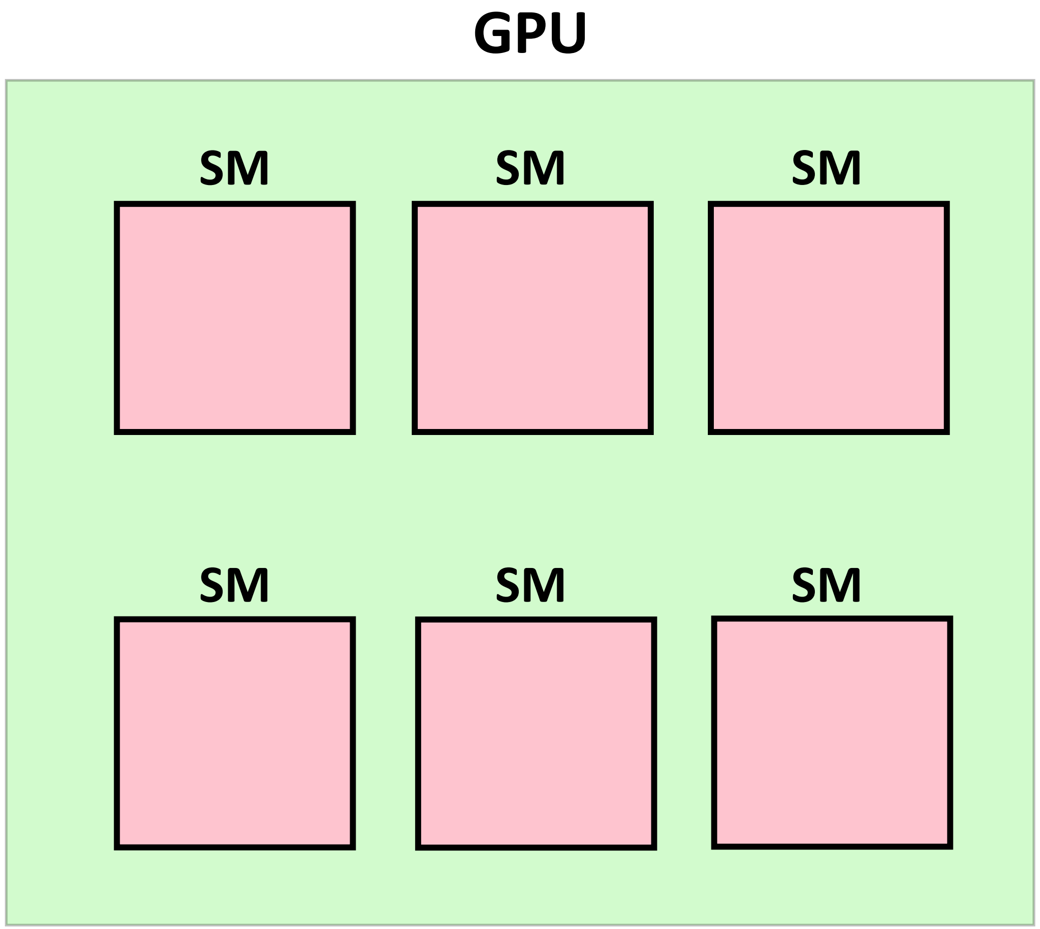




Background (Wave Quantization)

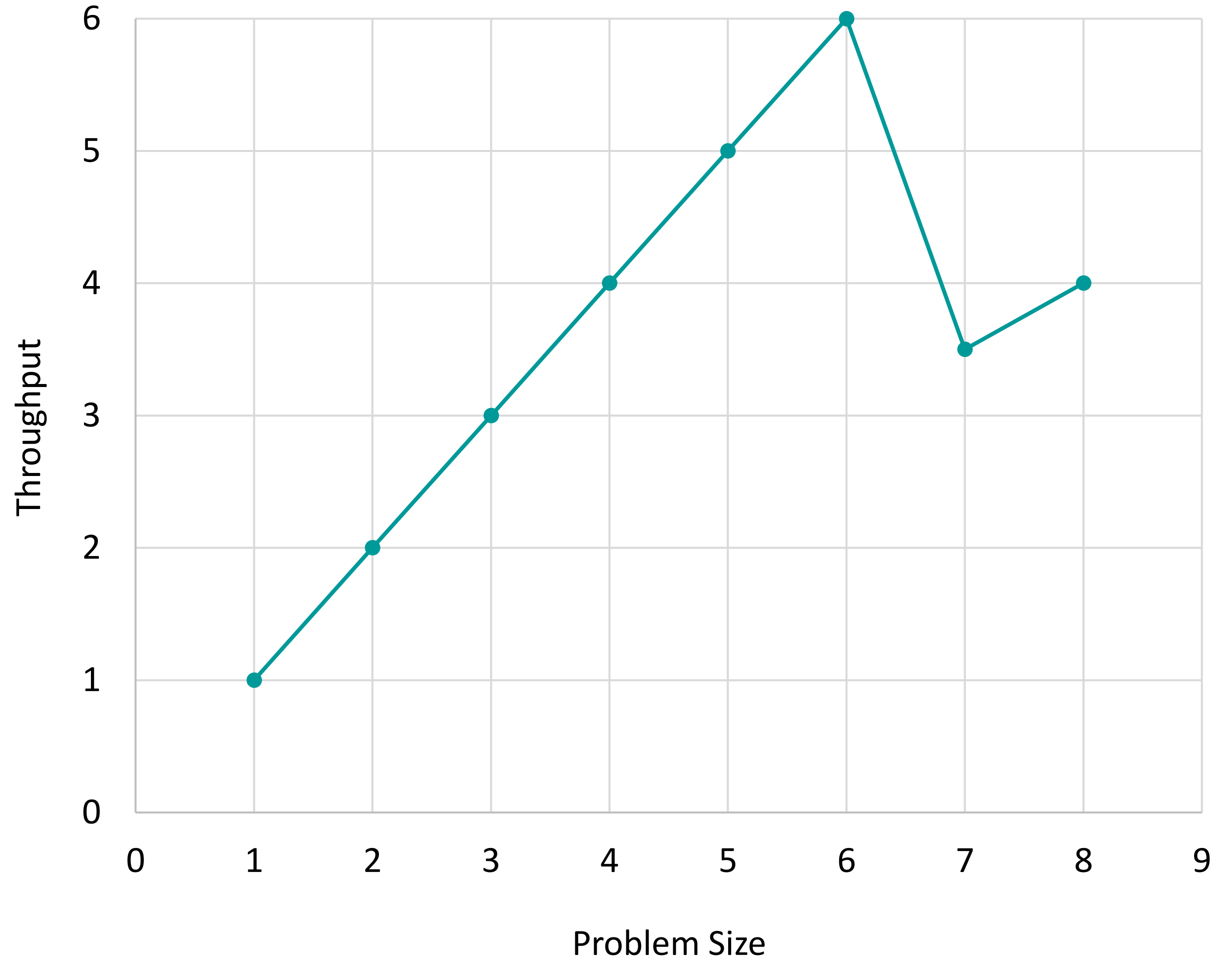


Total units to compute = 8

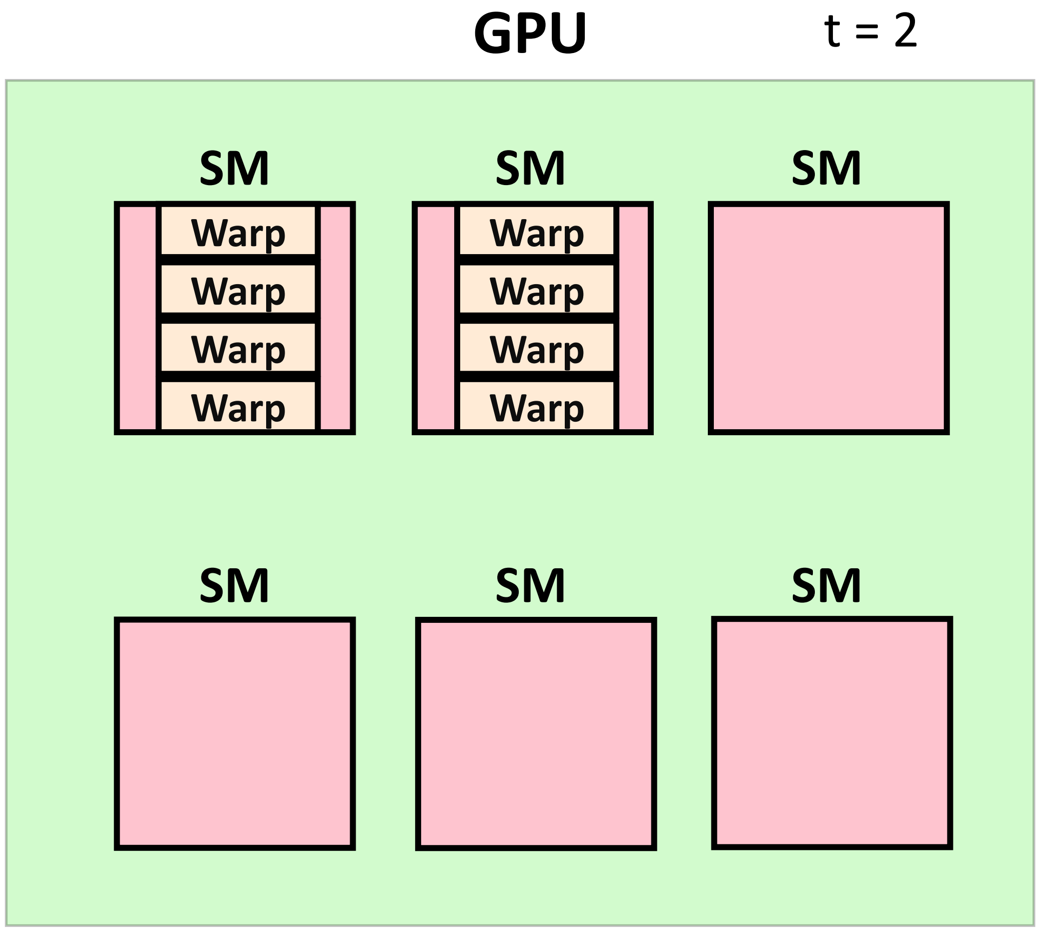


Background (Wave Quantization)

Throughput = (6 + 2) / 2 = 4



Total units to compute = 8



Sizing

a	Number of attention heads	s	Sequence length
b	Microbatch size	t	Tensor-parallel size
h	Hidden dimension size	v	Vocabulary size
L	Number of transformer layers		

TABLE I: Variable names.

- Boiled things down to the following:

Therefore to ensure the best performance from transformer models, ensure:

- The vocabulary size should be divisible by 64.
- The microbatch size b should be as large as possible [24].
- $b \cdot s$, $\frac{h}{a}$, and $\frac{h}{t}$ should be divisible by a power of two, though there is no further benefit to going beyond 64.
- $(b \cdot a)/t$ should be an integer.
- t should be as small as possible [25].

Importantly, the microbatch size b does not itself need to be divisible by a large power of 2 since the sequence length s is a large power of two.