An idea to dynamically allocate memory for multi-LLMs in vLLM

Chenye Wang

Jan 14, 2025

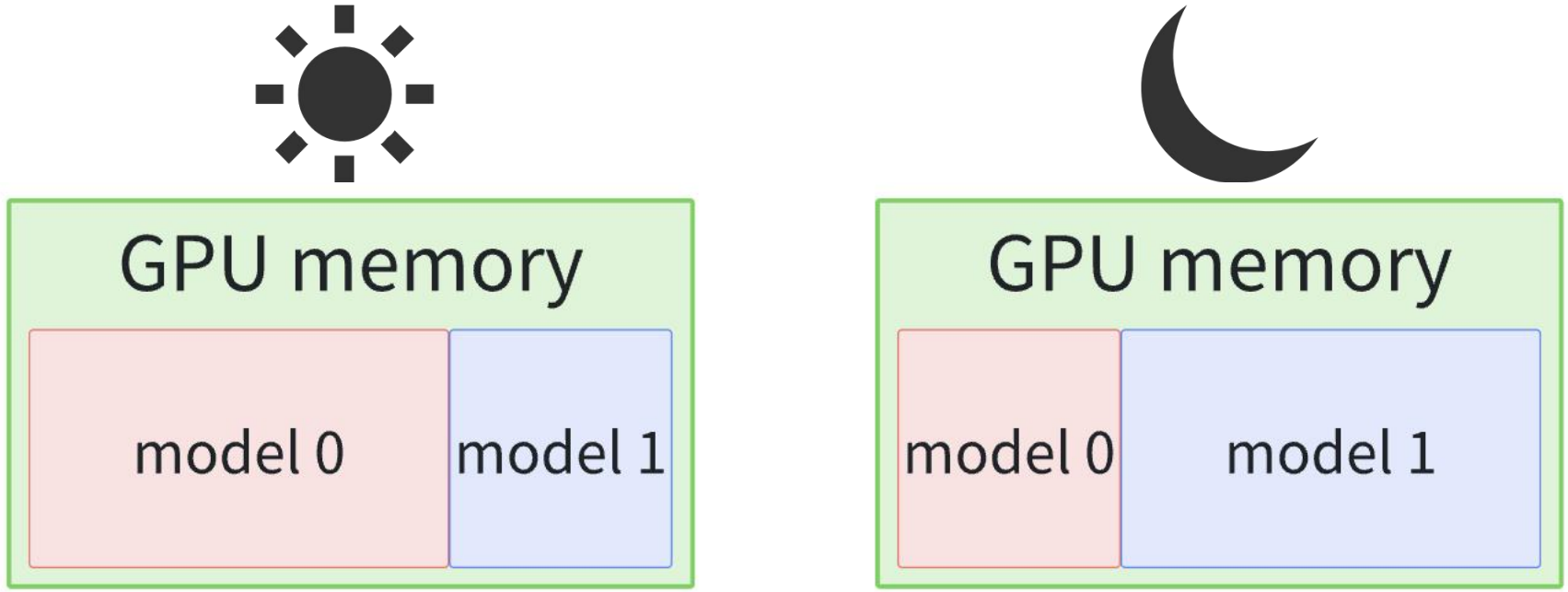
vLLM memory usage problem

- A model occupies a fixed amount of memory allocated to it throughout its inference, regardless of whether it needs less or more memory

```
# vllm/engine/llm_engine.py
class LLMEngine:
    def _initialize_kv_caches(self) -> None:
        num_gpu_blocks, num_cpu_blocks = (
            self.model_executor.determine_num_available_blocks())
        ...
        self.model_executor.initialize_cache(num_gpu_blocks, num_cpu_blocks)
```

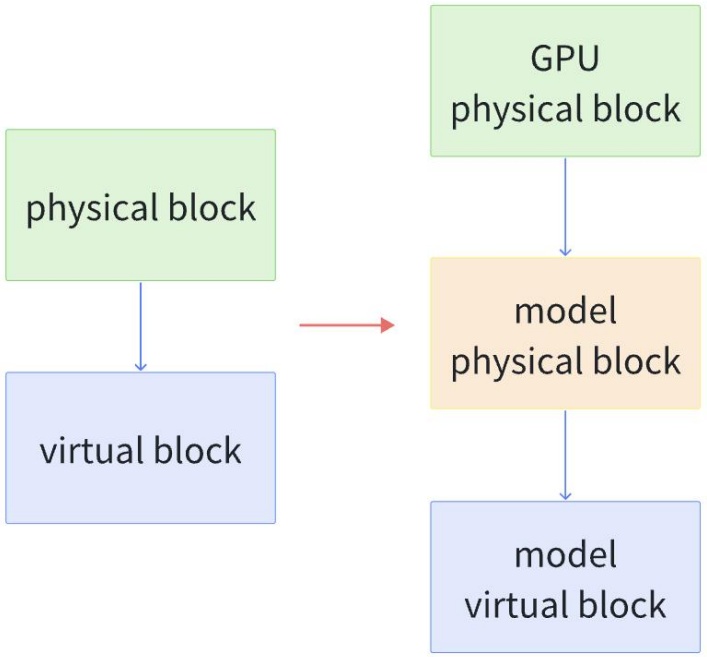
Challenges when multi-LLMs share GPU memory

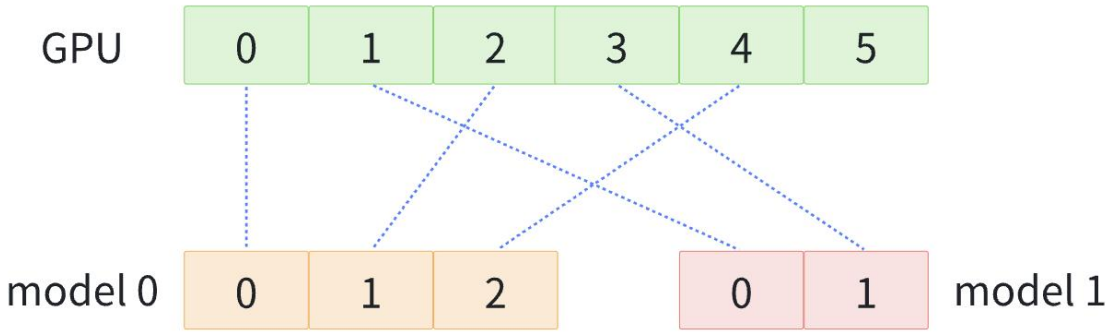
- Various models exhibit distinct patterns of GPU memory consumption across different time intervals.



How to dynamically allocate memory?

- Inspired by Expandable Segment, the entire GPU memory is treated as a block pool, adding a layer of mapping from the entire memory to the model memory.

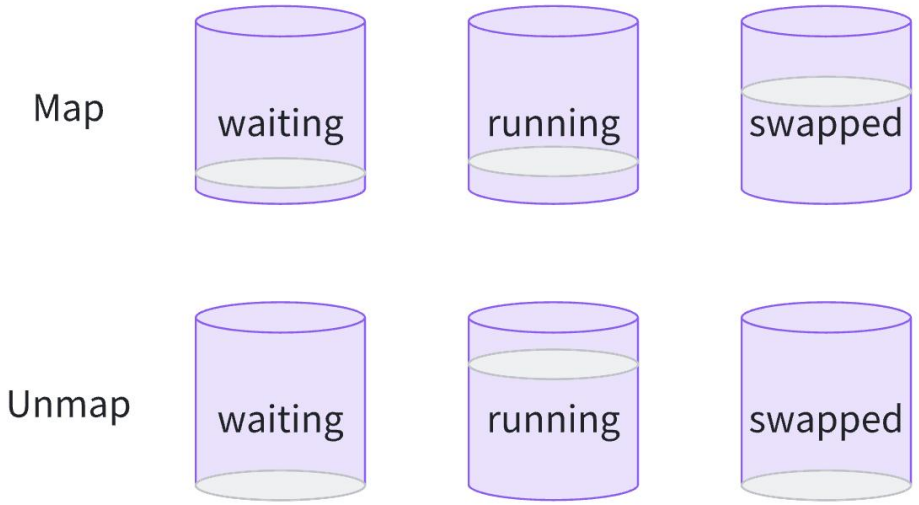




[1] https://developer.nvidia.com/blog/introducing-low-level-gpu-virtual-memory-management/

When to “malloc/free”?

- API
  - Reserve/Free: virtual
  - Map/Unmap: physical



- At each step
  - When the number of sequences in swapped queue exceeds a certain percentage in running queue, apply a new physical block.
  - When there is no sequence in either waiting or swapped queue, try to free a physical block.

Potential problems

- Overhead of calling API
- Boundary conditions
- Models need to have the same tensor shape