Scheduling for AI

영남대학교
차세대 컴퓨터 시스템 연구실
석사과정생 이원호

LAS: Locality-Aware Scheduling for GEMM-Accelerated Convolutions in GPUs

IEEE Transactions on Parallel and Distributed Systems, 2023

차세대 컴퓨터 시스템 연구실

이원호

배경

- 인공지능에서 Conv 연산은 신경망의 실행 시간 중 90% 이상을 차지
- GPU에서 스레드 레벨 병렬화를 활용하기 위해 Conv 연산을 변형시켜 활용
  - Nested convolution loop → Large matrix multiplication 으로 lowering 진행
- GEMM 연산을 통해 많은 스레드 블록들이 동시에 진행할 수 있도록 함

Conv 연산 적용방식

- Direct convolution
  - Nested loop 구조를 적용

- Explicit GEMM
  - GPU의 전역 메모리에 공유 데이터를 할당
  - Loop nest를 그대로 활용할 때보다 8~12.5배 많은 메모리 사용
  - 높은 스레드 레벨 병렬화 수준

- **Implicit GEMM**
  - 각 처리 유닛이 필요한 만큼 공유 메모리에 할당
  - Loop nest 만큼 메모리 사용
  - GPU는 공유 메모리와 L1 캐시를 단일 칩에서 활용 (공유 메모리 증가 → L1 캐시 감소)

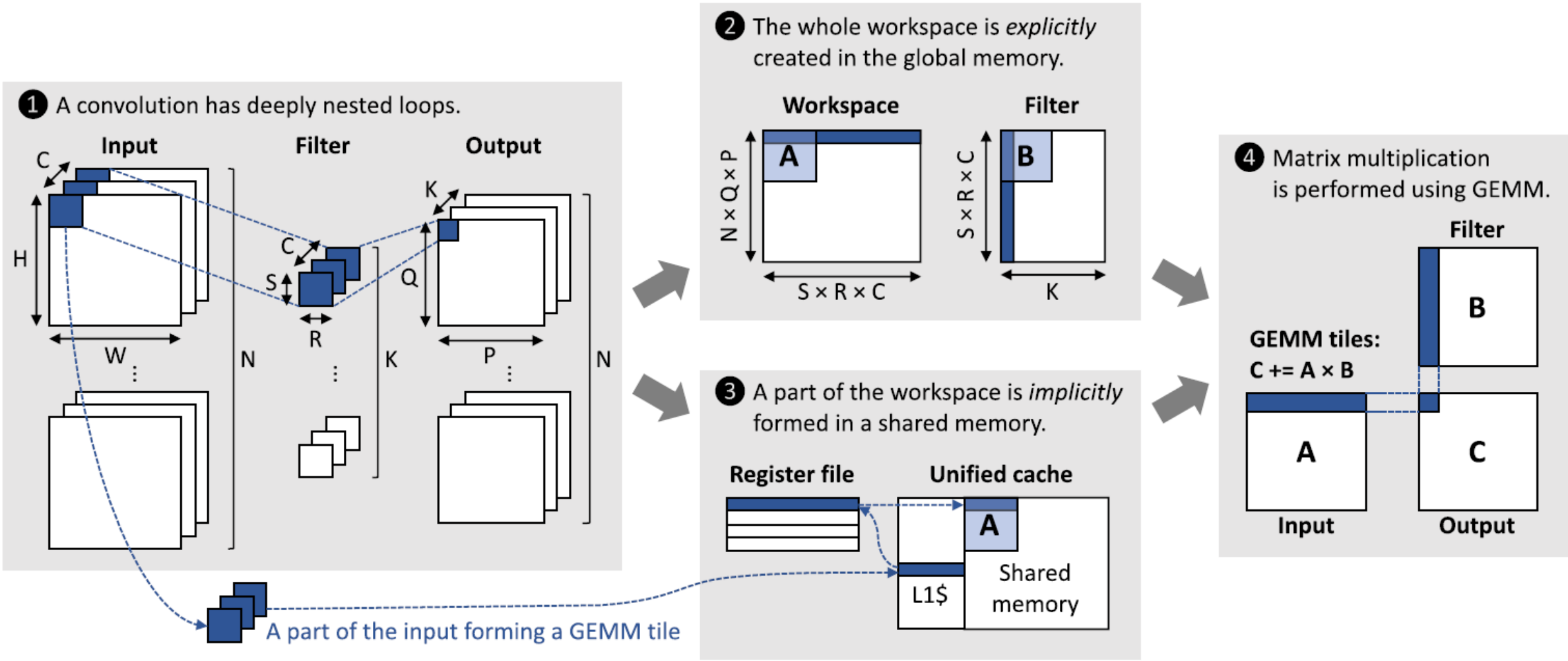


Fig. 1. Explicit GEMM follows the steps of ❶, ❷, and ❹ to unroll the deeply nested loops of a convolution into large matrix multiplication ($C+=A\times B$) by explicitly allocating workspace data in the global memory of a GPU. It significantly increases the memory capacity and bandwidth usage, but improved thread-level parallelism subdues the penalty and returns a performance enhancement. In contrast, implicit GEMM takes the procedure of ❶, ❸, and ❹ to allocate only the necessary parts of the workspace for handling scheduled thread blocks in each SM. Input data are loaded to a register file through on-chip caches and unrolled into a shared memory to create the necessary GEMM tiles during runtime executions.

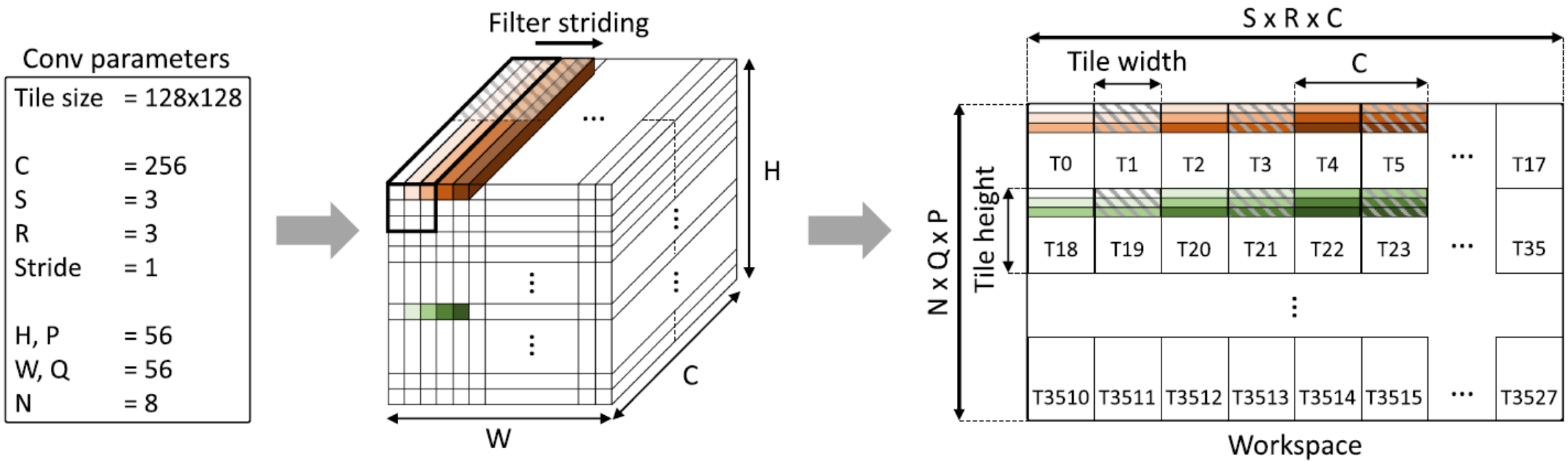
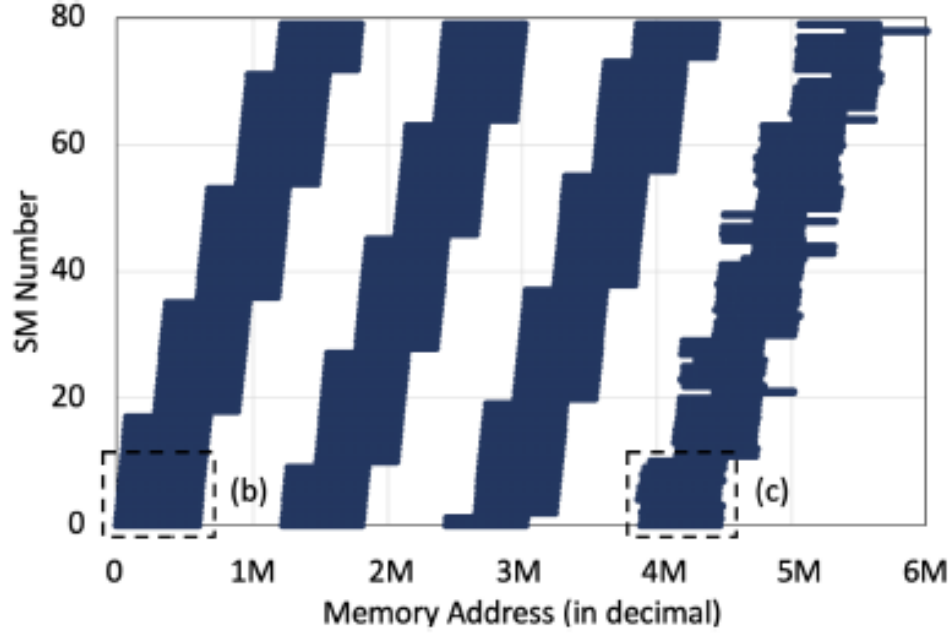


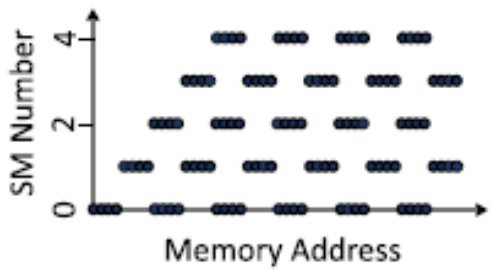
Fig. 2. To process the matrix multiplication of a large workspace in a GPU, it is sliced into smaller GEMM tiles, each mapped to a different CTA. The figure shows an example of the C5 layer of R-CNN [6], which creates a 25088×2304 workspace for the given convolution parameters. The workspace is partitioned into 3528 GEMM tiles, each sized as 128×128. Notably, the striding movements of filters cause input data to replicate in the workspace diagonally from the upper right to the lower left. In the figure, vectors of the same color and stripe have exactly the same data. For instance, the second and third rows of T0 are the same as the first and second rows of T2.

기존 방식

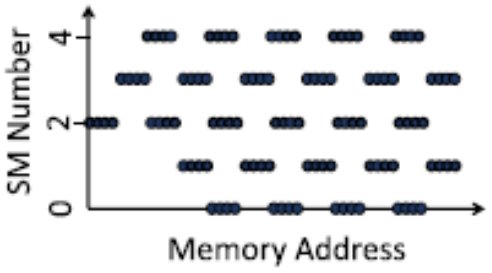
- 기존의 스케줄링 방식은 데이터의 지역성을 고려하지 않음
- 여러 유닛에서 같은 메모리 주소에 동시 접근하게 됨



(a) Memory accesses of input data across SMs



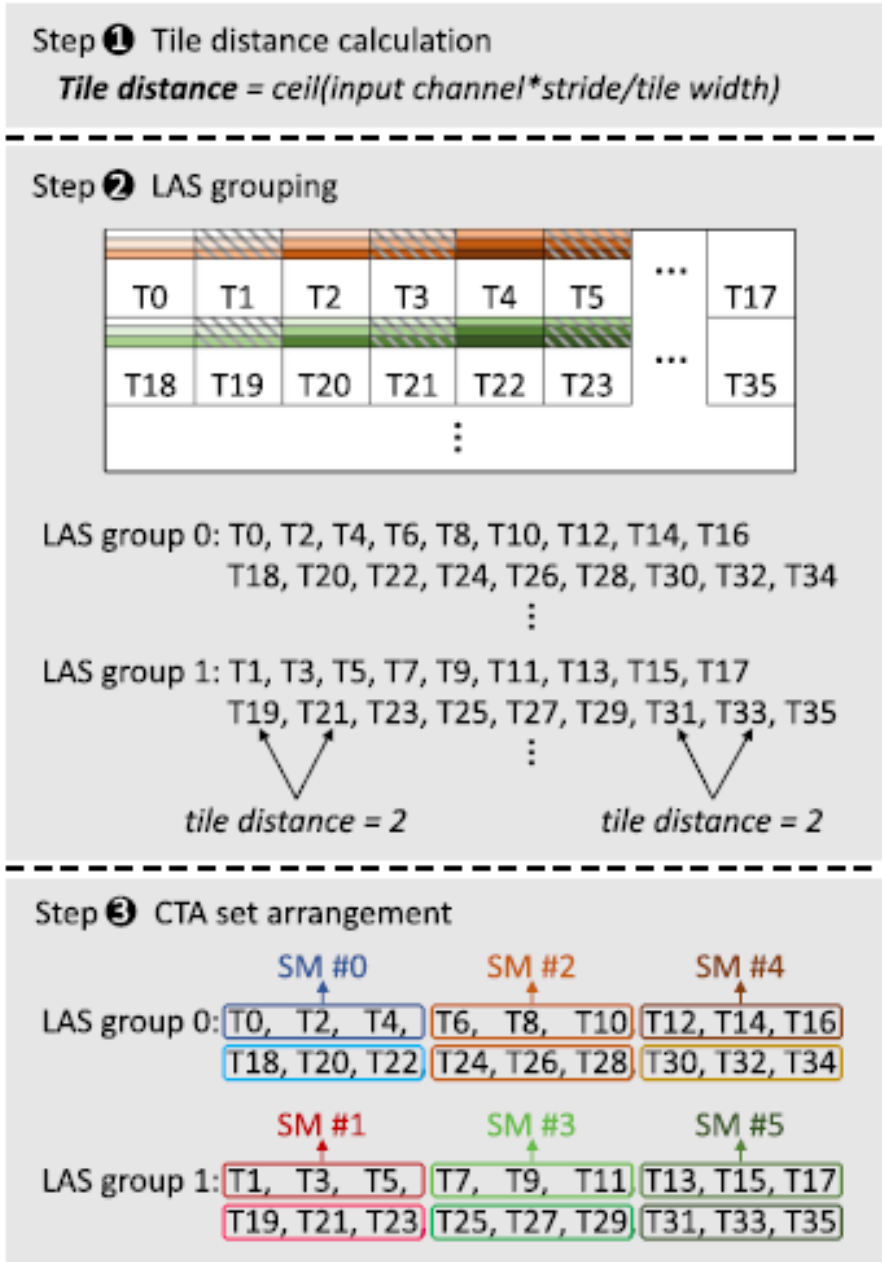
(b) Round robin scheduling



(c) Non-deterministic executions

제안 방식

- 스레드간 거리를 통해 데이터 유사도를 검증함
- 유사도를 기반으로 스레드를 그룹으로 그룹화
- 그룹 단위로 처리 유닛이 수용할 수 있는 최대량만큼 배치



제안 방식

- 처리가 완료된 스레드가 생겨 처리 유닛의 일부가 유휴 상태가 될 경우

- 같은 그룹내의 다른 블록을 가져와 해당 처리 유닛에 할당

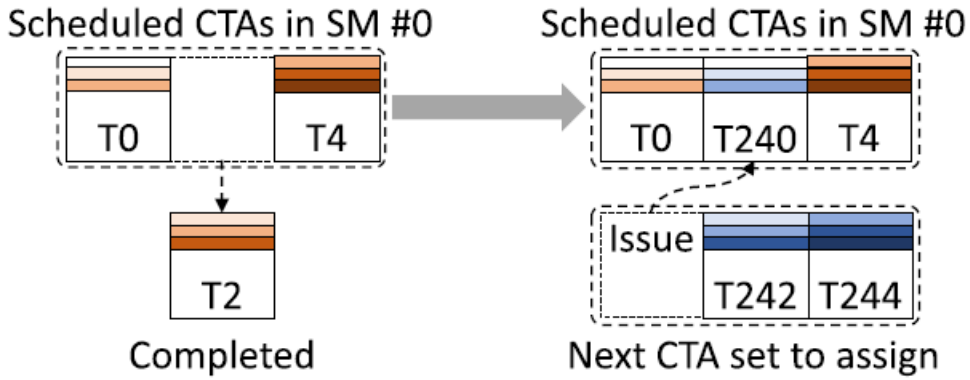
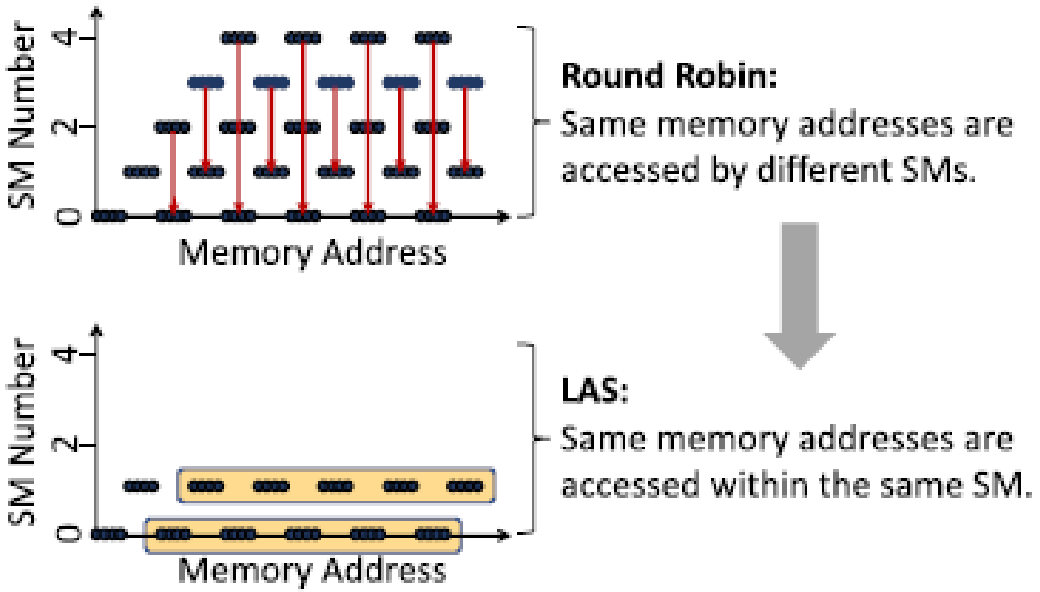
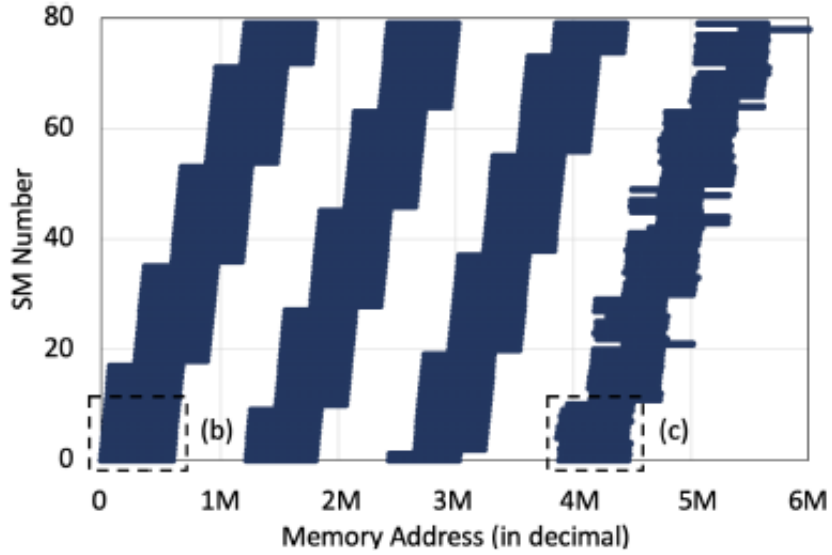


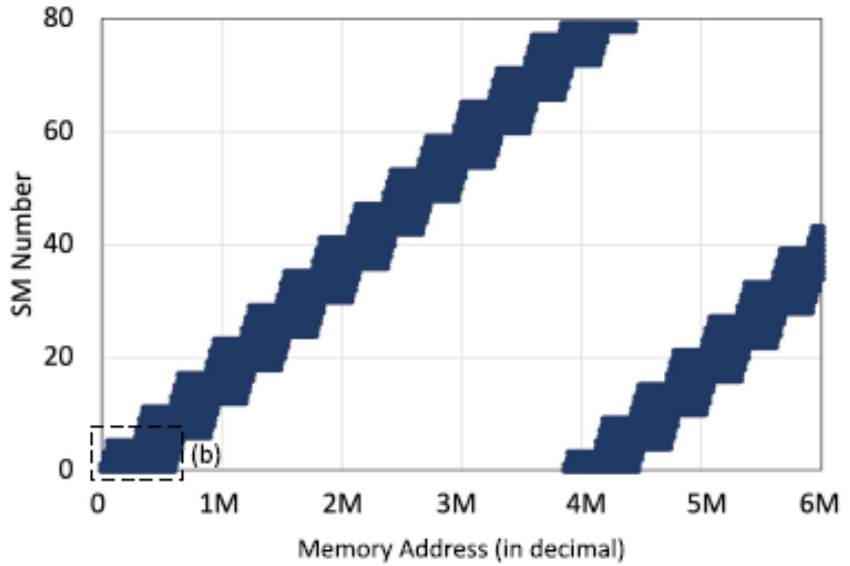
Fig. 5. LAS assigns CTAs to an SM at the set granularity. When a CTA in a set retires from an SM, a new CTA set is assigned to the SM. The first CTA of the new set (i.e., T240) takes the place of the retiring one (i.e., T2), and other CTAs in the set (i.e., T242, T244) will be scheduled later as soon as the SM becomes free. Thus, LAS ensures that a whole set of CTAs sharing many duplicate values are executed in the same SM.

제안 방식



(b) Changes of memory accesses from round robin to LAS





Orca: A Distributed Serving System for Transformer-Based Generative Models.

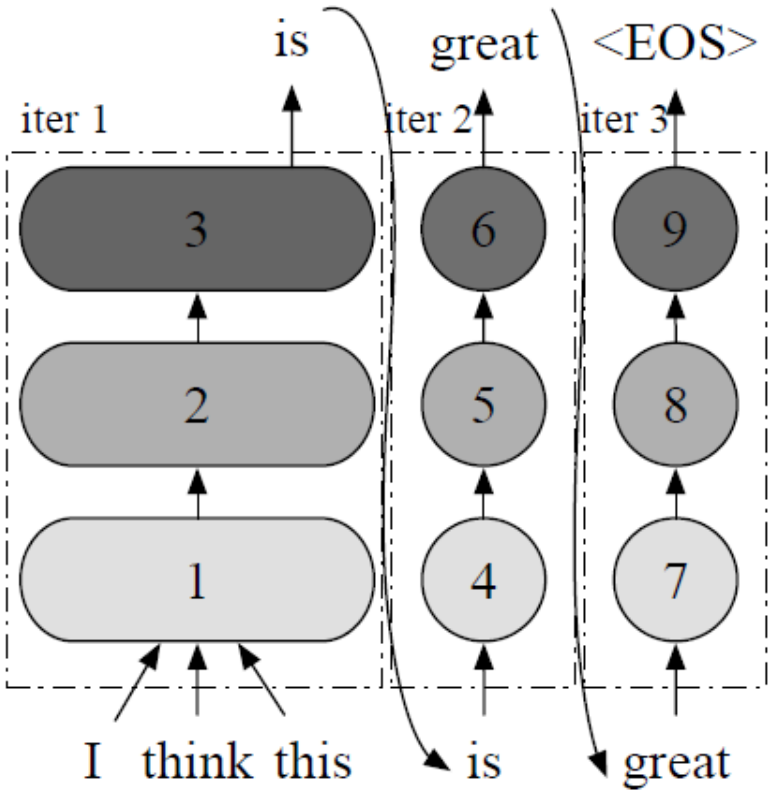
16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)

차세대 컴퓨터 시스템 연구실

이원호

배경

- 트랜스포머 기반의 생성 모델은 autoregressive한 방식으로 토큰을 생성



(a) A computation graph representing an inference procedure using a GPT model. The graph does not depict layers other than Transformer layers (e.g., embedding) for simplicity.

문제 제기

- 기존 트랜스포머를 활용한 생성 모델들은 각 반복마다 하나의 토큰을 반환하는 방식

- 서비스 수준에서 추론 단계는 요청에 대해 일괄 처리를 진행하기 때문에 여러 문제 발생
  - 같은 일괄 처리 안의 요청들은 가변적인 입력 길이와 출력 길이로 인해 처리가 끝났음에도 반환되지 않음
  - 가장 긴 요청에 대한 처리로 인해 대기가 길어짐

문제 제기

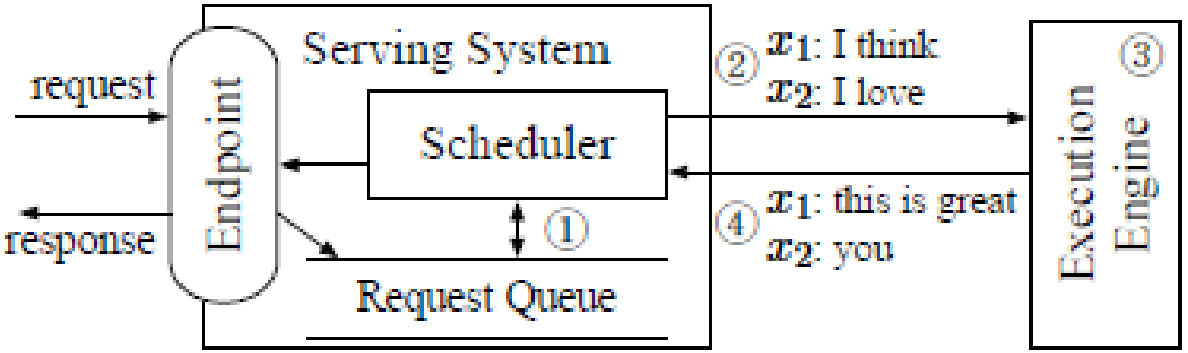


Figure 2: Overall workflow of serving a generative language model with existing systems.

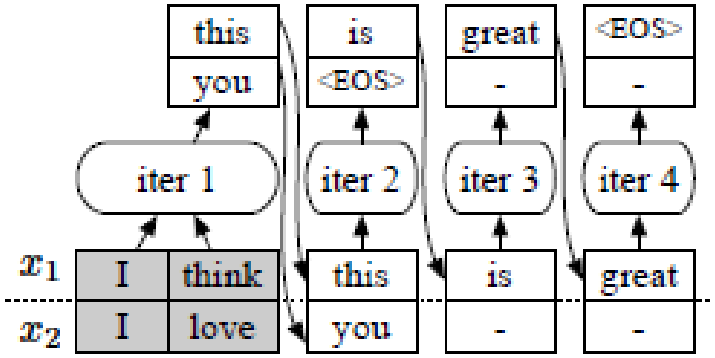


Figure 3: An illustration for a case where the requests have the same input length but some requests finish earlier than others. Shaded tokens represent input tokens. "-" denotes inputs and outputs of extra computation imposed by the scheduling.

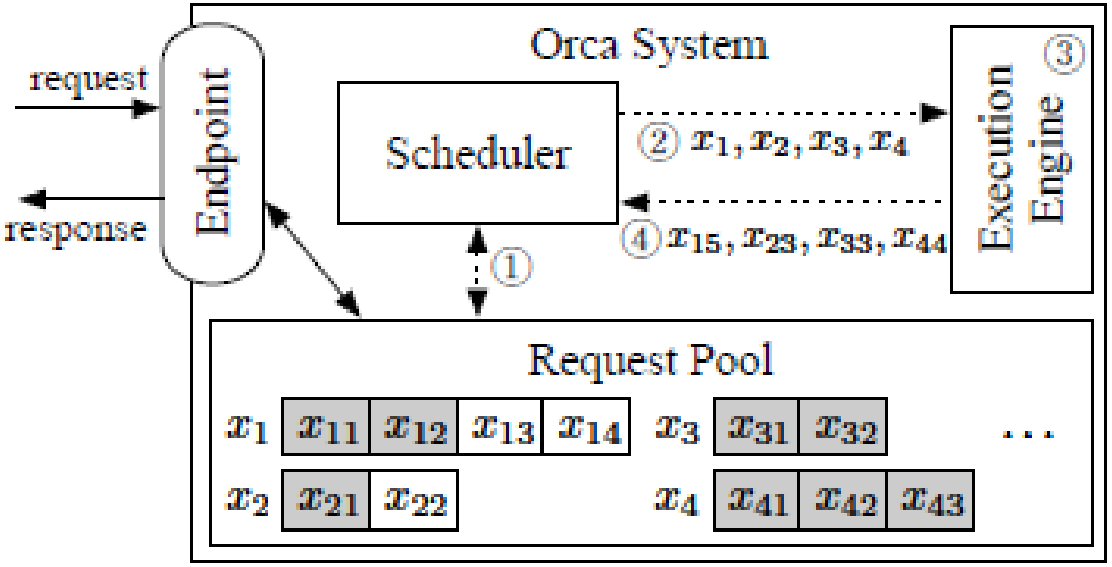
제안 사항

- Iteration-level scheduling
  - 요청 단위의 스케줄링이 아닌 반복 단위의 스케줄링

- Selective batching
  - 선택된 연산 집합에 대해서만 배치 처리 진행
  - 가변 길이의 입력에 대한 요청을 배치 처리 가능

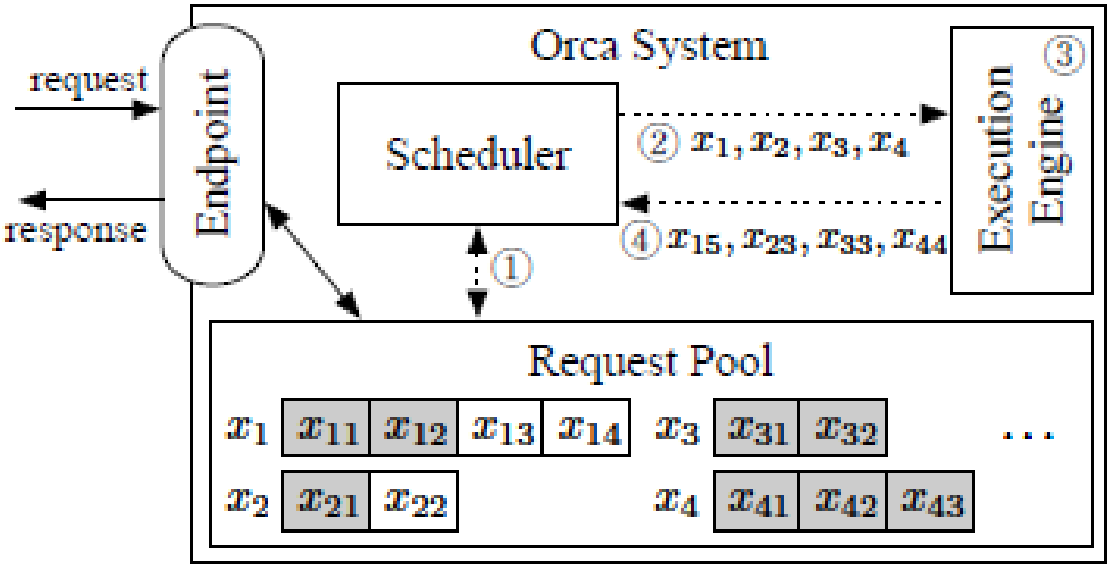
Iteration-level scheduling

- Iteration-level scheduling
  - 요청 단위의 스케줄링이 아닌 반복 단위의 스케줄링
  1. 다음 실행될 요청을 선택
  2. 선택된 요청에 대해 한 번의 반복 수준의 실행
  3. 스케줄러가 요청을 결과가 해당 요청에 대한 끝인지 추적



Selective batching

- Selective batching
  - Iteration-level scheduling 을 진행할 경우 해당 요청의 진행 상태와 입력 길이에 따라 배치가 불가능한 경우가 발생
    - 최초 요청시 입력의 길이가 다른 경우 (x3, x4)
    - 요청 진행 중 출력 인덱스가 다른 경우 (x1, x2)
    - 두 요청의 상태가 다른 경우 (x1, x3)



Selective batching

- Selective batching
  - 입력의 상태를 확인하고 그에 맞는 어텐션 K/V 상태를 확인
  - 배치를 통해 요청들을 진행할 수 있도록 함

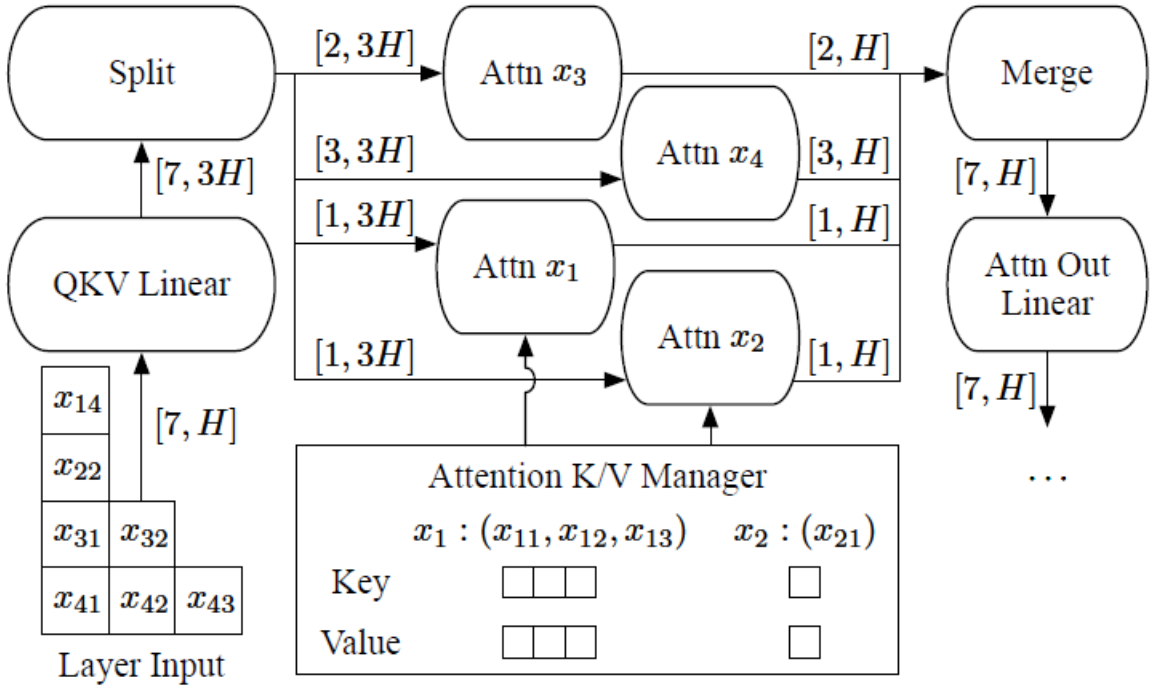
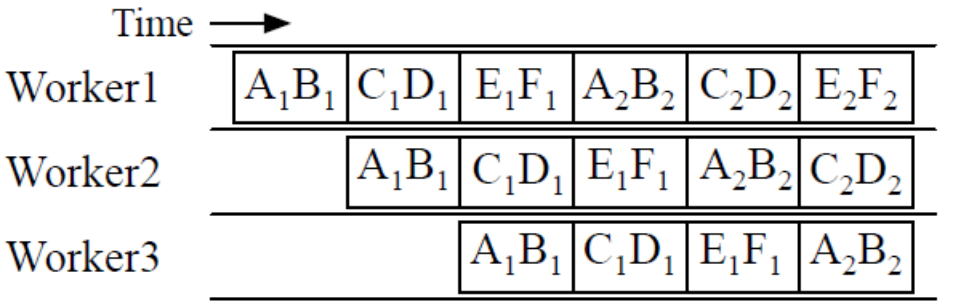
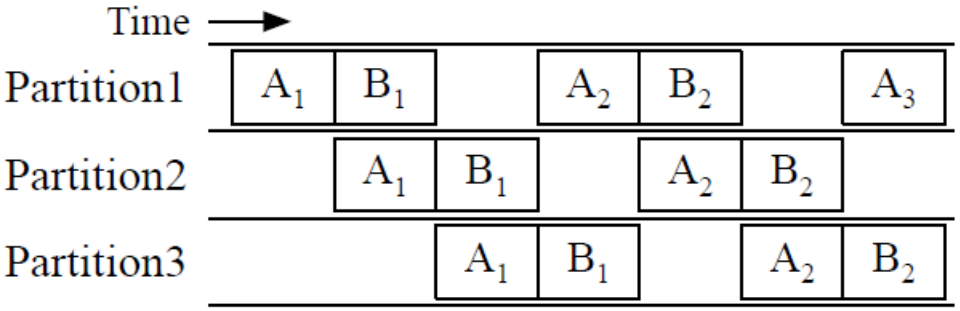


Figure 5: An illustration of ORCA execution engine running a Transformer layer on a batch of requests with selective batching. We only depict the QKV Linear, Attention, and Attention Out Linear operations for simplicity.

Appliance



(a) ORCA execution pipeline.



(b) FasterTransformer execution pipeline.

Figure 8: Comparison of the use of pipeline parallelism in ORCA and FasterTransformer where X_i is the i-th iteration of request X.

TurboTransformers: an efficient GPU serving system for transformer models

Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021

차세대 컴퓨터 시스템 연구실

이원호

문제 제기

- 온라인 서비스를 위해 트랜스포머를 GPU가 장착된 데이터 센터에서 배포하기 어려움
  - 트랜스포머 구조는 많은 연산량을 요구하기 때문에 latency와 throughput을 만족하기 어려움
  - NLP 작업들은 가변 길이의 입력 문장을 가짐
- 다차원 입력에 대해 처리하기 위해 효율적인 메모리 관리와 최적화가 필요함

제안 사항

- Computational Optimization
  - Kernel Fusion
  - GPU-based Batch Reduction
- Memory Manager
  - Sequence-length-aware allocator
- Batch Scheduling with DP

Computational Optimization

- Kernel Fusion
  - 두 가지 종류의 커널 연산
    - Element-wise operation
    - Reduction operation
  - 메모리 접근 횟수의 감소
  - 캐시 지역성 향상
  - 커널 실행 오버헤드 감소

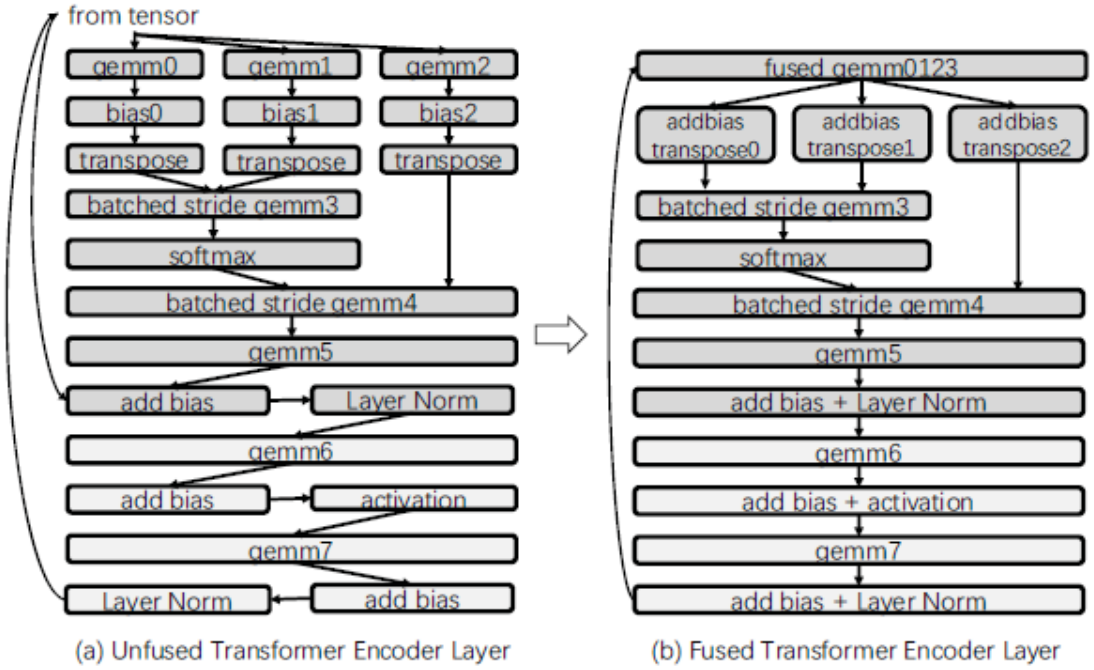


Figure 3. Kernel fusion of a transformer encoder. The part in darker color is a multi-head attention. The part in lighter color is a feed forward network.

Computational Optimization

- GPU-based Batch Reduction
  - LayerNorm과 Softmax 연산을 Batch Reduction 연산으로 볼 수 있음
  - 세가지 하드웨어 레벨 지원 가능

1. Batch 차원을 분할해 처리 유닛 레벨에서 병렬화
2. Wrap level parallelism을 통해 스레드간 효율성 증가
3. 명령어 레벨 병렬화를 통해 메모리 접근과 연산 명령을 겹침

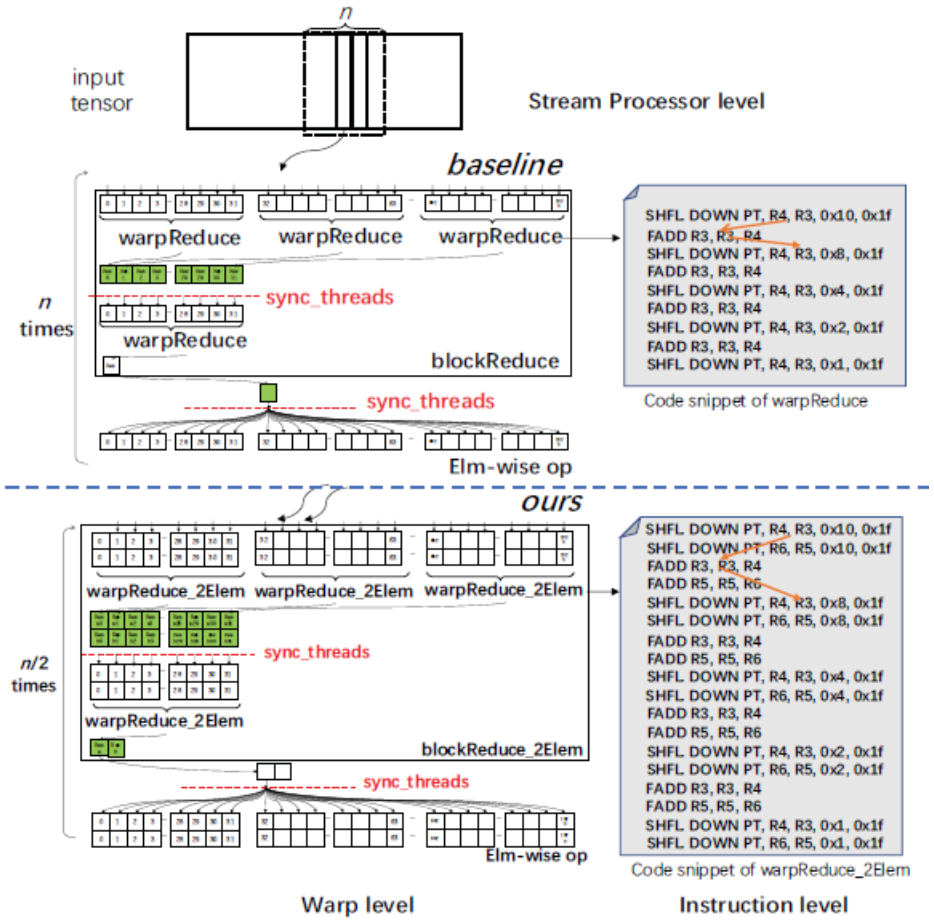


Figure 4. Parallel batch reduction optimizations on three hardware levels.

Memory Manager

- Sequence-length-aware allocator
  - 메모리 공간을 chunk 단위로 구성
  - 연산 그래프를 통해 중간 텐서의 생애주기를 사전 파악
  - 특정 chunk와 함께 각 텐서의 오프셋을 계산 (새로운 요청의 길이를 파악하기 전에)

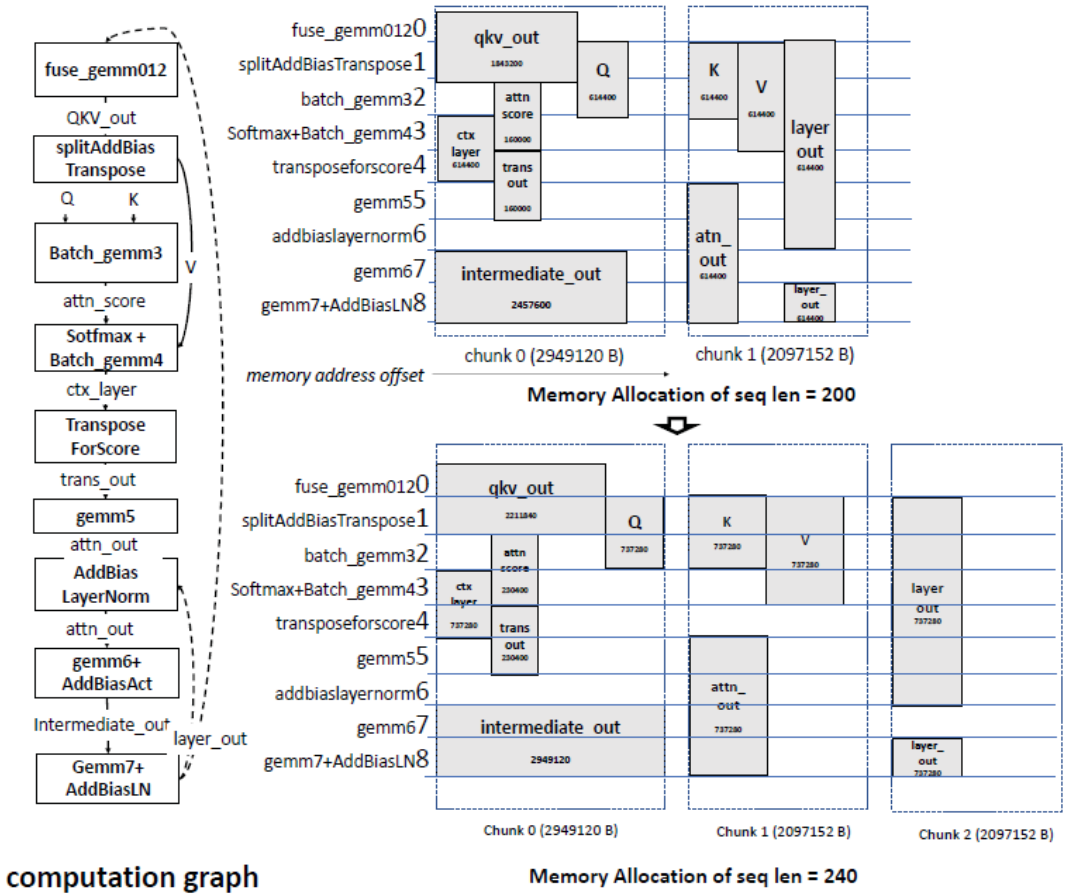


Figure 6. A memory allocation example uses our proposed variable-length-aware allocator.

Two Optimization in Serving

- Caching
- Batching

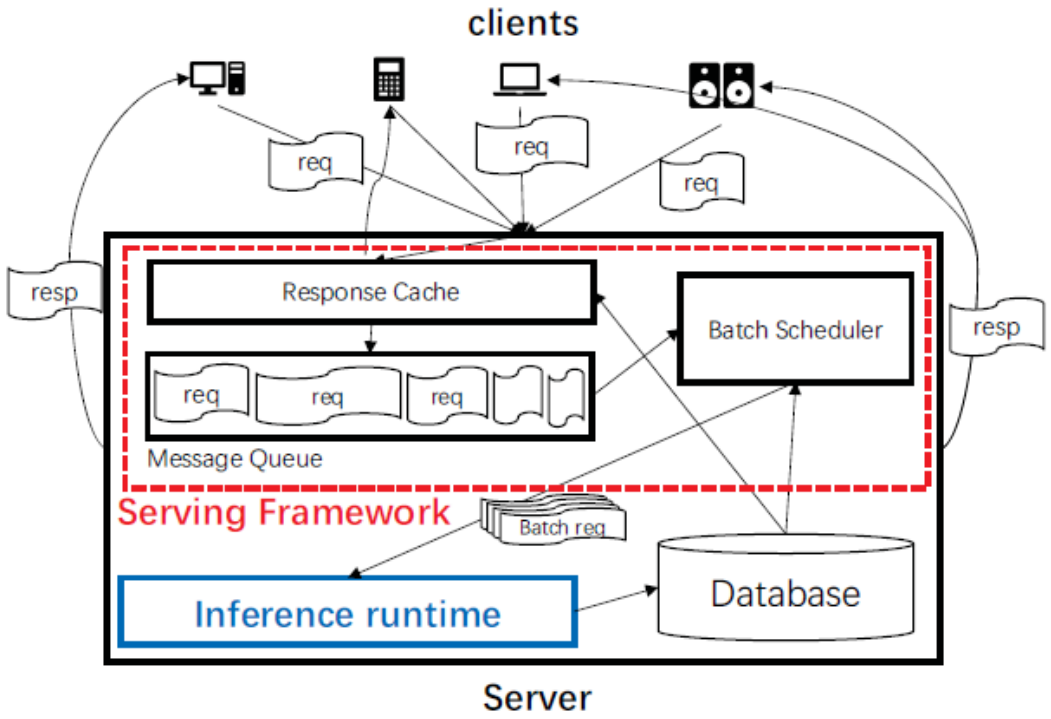


Figure 2. The serving system architecture adopted by Turbo-Transformers.

Batch scheduling with DP

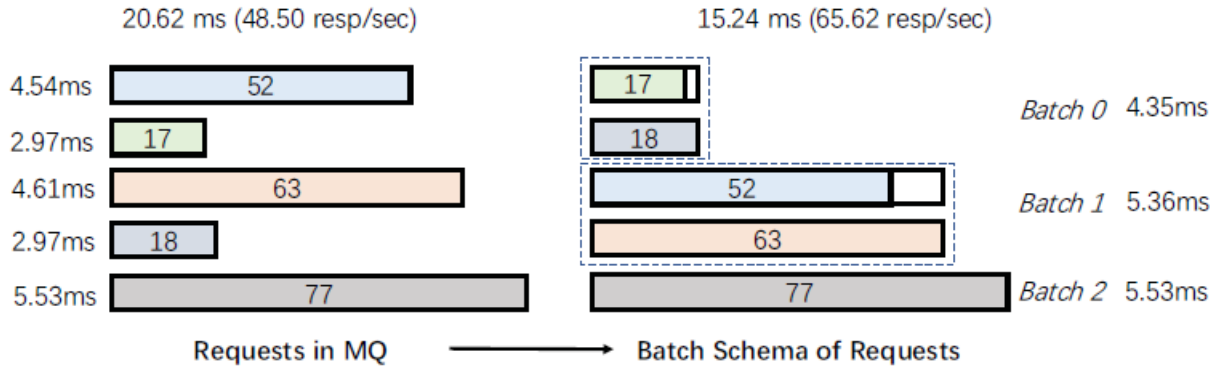


Figure 8. An example of batch scheduler for variable-length requests.

Algorithm 2: Batch Scheduler With DP

Input: *request_list*, *cached_cost*

sort *request_list* in increasing order with regards to sequence length;
$N \leftarrow$ Size(*request_list*);
Create *states*, *start_idx_lis* as lists of size $N + 1$;
$states[0] \leftarrow 0$; $i \leftarrow 1$;
while $i \leq N$ **do**
    $j \leftarrow i - 1$; $start\_idx = i - 1$;
    $cur\_length = request\_list[i - 1].length$;
    $min\_cost = cached\_cost[cur\_length][1] + states[j]$;
    while $j > 0$ **do**
        $tmp\_cost = states[j - 1] +$
          $cached\_cost[cur\_length][i - j + 1] * (i - j + 1)$;
        if $tmp\_cost < min\_cost$ **then**
            $min\_cost = tmp\_cost$; $start\_idx = j - 1$;
        end
        $j \leftarrow j - 1$;
    end
    $states[i] = min\_cost$; $start\_idx\_list[i] = start\_idx$;
    $i \leftarrow i + 1$;
end
$i = N$;
while $i > 0$ **do**
    $end\_idx \leftarrow i$; $start\_idx \leftarrow start\_idx\_list[i]$;
    pack $request\_list[start\_idx : end\_idx]$ into a batch;
    $i = start\_idx - 1$;
end

Reference

- LAS: Locality-Aware Scheduling for GEMM-Accelerated Convolutions in GPUs. IEEE Transactions on Parallel and Distributed Systems, 2023
- Orca: A Distributed Serving System for Transformer-Based Generative Models. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)
- TurboTransformers: an efficient GPU serving system for transformer models. Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021