DVABatch: Diversity-aware Multi-Entry Multi-Exit Batching for Efficient Processing of DNN Services on GPUs

Weihao Cui¹, Han Zhao¹, Quan Chen¹, Hao Wei¹, Zirui Li¹, Deze Zeng², Chao Li¹, and Minyi Guo¹

¹Shanghai Jiao Tong University, Emerging Parallel Computing Center
²China University of Geosciences
Deploying DNN Services On GPUs
Deploying DNN Services On GPUs

- DNN techniques are powering cloud services.
Deploying DNN Services On GPUs

• DNN techniques are powering cloud services.

• Dedicated accelerators like GPUs speed up DNN inferences.
Deploying DNN Services On GPUs

- DNN techniques are powering cloud services.
- Dedicated accelerators like GPUs speed up DNN inferences.
- Important to process DNN-based services efficiently on GPUs.
Batching Improves Efficiency
Batching Improves Efficiency

• A simple example: batched image classification.
Batching Improves Efficiency

• A simple example: batched image classification.

A time window to batch queries
Batching Improves Efficiency

- A simple example: batched image classification.

A time window to batch queries

DNN models on GPUs

Processed at once with high parallelism

• Man
• Dog
• Cat
• Car

A simple example: batched image classification.
Batching Improves Efficiency

- A simple example: batched image classification.
Batching Improves Efficiency

- A simple example: batched image classification.

Serving diversities diminish the efficiency of the single-entry single-exit scheme.
Inefficiency due to Diversities
Inefficiency due to Diversities

- Input diversity: DNN model accept inputs with different shapes.
Inefficiency due to Diversities

- Input diversity: DNN model accept inputs with different shapes.
Inefficiency due to Diversities

- Input diversity: DNN model accept inputs with different shapes.
Inefficiency due to Diversities

• Input diversity: DNN model accept inputs with different shapes.

• Extra padding for input diversity results in wasted computation.
Inefficiency due to Diversities

- Input diversity: DNN model accept inputs with different shapes.

- Extra padding for input diversity results in wasted computation.
Inefficiency due to Diversities

- Operator diversity: operators require different BS to fully utilize the GPU.

![Graph showing latency vs. batch size with a note: where batching takes effect!!!]
Inefficiency due to Diversities

- Operator diversity: operators require different BS to fully utilize the GPU.

![Diagram showing latency vs batch size]

Prefer $bs=1$
Inefficiency due to Diversities

- Operator diversity: operators require different BS to fully utilize the GPU.

- Longer latency without throughput improvement
Inefficiency due to Diversities

- Operator diversity: operators require different BS to fully utilize the GPU.

![Graph showing latency vs. batch size]

- Longer latency without throughput improvement

```
1  2  3  4

1  A  B  C  D
2  A  B  C  D
3  A  B  C  D
4  A  B  C  D
```

**Prefer bs = 1**

**IDEAL**

```
1  A  B  C  D
2  3/4T  B  C  D
3  3/4T  3/4T  B  C  D
4  3/4T  3/4T  3/4T  B  C  D
```
Inefficiency due to Diversities

- Load diversity: insufficient batch due to load variation.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

4T

Idle hardware resource

A B C D

A B C D

A B C D

A B C D
Inefficiency due to Diversities

• Load diversity: insufficient batch due to load variation.

- Hardware is idle while there are queries for processing
Inefficiency due to Diversities

- Load diversity: insufficient batch due to load variation.

- Hardware is idle while there are queries for processing.
Weakness of Single-entry Single-exit

- Assume that batch always have positive effects.

- An ongoing batch cannot be interrupted for adjustment.

- Cannot be configured to support various diversities.
Weakness of Single-entry Single-exit

- Assume that batch always have positive effects.
- An ongoing batch cannot be interrupted for adjustment.
- Cannot be configured to support various diversities.

A multi-entry multi-exit batch scheme?
Methodology
The Goal of DVABatch

- A runtime batch scheduling system integrated into existing frameworks
- A multi-entry multi-exit scheme for adjusting the batch on the fly.
- A holistic solution for different serving diversities.
Overview of DVABatch

- Holistic multi-entry and multi-exit scheme
Overview of DVABatch

- Holistic multi-entry and multi-exit scheme
- **Slice** DNN model into multiple stages (executors connected by queue).
Overview of DVABatch

- Holistic multi-entry and multi-exit scheme
- **Slice** DNN model into multiple stages (executors connected by queue).
- Three **meta-operations** for adjusting the batching.
Overview of DVABatch

• Workflow for a new DNN services
Overview of DVABatch

• Workflow for a new DNN services

 identify the serving diversities through and slice model.
Overview of DVABatch

- Workflow for a new DNN services
  1. Identify the serving diversities through and slice model.
  2. Load models and customize the scheduler.
Overview of DVABatch

- Workflow for a new DNN services
  1. Identify the serving diversities through and slice model.
  2. Load models and customize the scheduler.
  3. Exploit meta operations for efficient processing.
Holistic Multi-entry Multi-exit Scheme

• Key abstraction: meta operations to adjust an on-going batch
Holistic Multi-entry Multi-exit Scheme

• Key abstraction: meta operations to adjust an on-going batch
Holistic Multi-entry Multi-exit Scheme

- Key abstraction: meta operations to adjust an on-going batch

New

The new incoming queries are organized into a new batch.
Holistic Multi-entry Multi-exit Scheme

- Key abstraction: meta operations to adjust an on-going batch

- New
  - The new incoming queries are organized into a new batch.

- Stretch
  - An ongoing batch is stretched with new incoming queries.
Holistic Multi-entry Multi-exit Scheme

• Key abstraction: meta operations to adjust an on-going batch

The new incoming queries are organized into a new batch.

An ongoing batch is stretched with new incoming queries.

An ongoing batch is split into several batches to be processed separately.
Management of stage executors

- Process with multiple buffer pairs
Management of stage executors

- Process with multiple buffer pairs
Management of stage executors

- Process with multiple buffer pairs

![Diagram showing management of stage executors]

- Input buffers: 1, 2, 3
- Output buffers: 1, 2, 3
- Invalid buffer: 1
- Inreading buffer: 2
- Inwriting buffer: 3
- Available buffer: 4

Avoid Hazard

Process with legitimate buffer pair → Active
Extra legitimate buffer pair → Legitimate buffer pair
No legitimate buffer pair → Inactive

Legitimize buffer pair for \( \text{executor}_{i-1} \) when completing
Management of stage executors

- Process with multiple buffer pairs

**Work well for the single-entry single-exit pipeline**
Management of stage executors

• Validity problem for the multi-entry multi-exit pipeline.
Management of stage executors

- Validity problem for the multi-entry multi-exit pipeline.
- Scrambled access order due to Stretch and Split

![Diagram showing executors and their states with labels: Working with bp₃, Active with bp₂.]

```plaintext
Working with bp₃  Active with bp₂
executorᵢ  executorᵢ₊₁
```

```plaintext
1 2 3
... 1 2 3
```
Management of stage executors

- Validity problem for the multi-entry multi-exit pipeline.
- Scrambled access order due to Stretch and Split
- Work in parallel even with one buffer pair
Management of stage executors

- Manage stage executors with state transition rules.
  - Process with multiple buffer pairs without hazard.
  - Ensure correct access order of buffer pairs.
  - Support switching between serial and parallel work manner.

![Diagram showing state transitions and buffer pairs.]

- **Input buffers**
  - `executor_i`
  - Invalid buffer: 1
  - Inreading buffer: 2
  - Inwriting buffer: 3
  - Available buffer: 4

- **Output buffers**
  - `bp_j`

- **States**
  - **Checking**
  - **Active**
  - **Working**
  - **Inactive**

- **Transitions**
  - Legitimize `bp_j` for executor `i-1`
  - `bp_j` is not legitimate
  - Process with legitimate `bp_j`
  - Extra legitimate buffer pair
  - No legitimate buffer pair
Management of stage executors

- Manage stage executors with state transition rules.
- Process with multiple buffer pairs without hazard.
- Ensure correct access order of buffer pairs.
- Support switching between serial and parallel work manner.

- **Input buffers**
  - executor₁
- **Output buffers**
  - 1
  - 2
  - 3

**Buffer Pairs**
- **Invalid buffer:**
  - Inreading buffer:
  - Inwriting buffer:
  - Available buffer:
- **Available buffer:**
  - 1
  - 2
  - 3
  - 4

**State Transition**
- Guarantee using the right $bp$
- Legitimize $bp_j$ for executor$_{i-1}$
Management of stage executors

- Manage stage executors with state transition rules.
- Process with multiple buffer pairs without hazard.
- Ensure correct access order of buffer pairs.
- Support switching between serial and parallel work manner.
//stage executors run within a while loop
void Run():
    Batch& inBatch = BatchQueue.Get();
    CheckBuffer(inBatch);//Check buffer pair
    Execute(inBatch);
    Schedule(inBatch, outBatches);
    for (auto& batch : outBatches):
        nextBatchQueue.Push(batch)
        getBuffer(); //get legitimate buffer pair
    updatePrExecutor();//update preceding executor
    //call Schedule to perform meta operations
    void Schedule(Batch& inBatch, vector<Batch>&
    outBatches):
        BatchTable.update(inBatch);
        if userDefined1:
            outBatches = BatchTable.New(inBatch);
        else if userDefined2:
            outBatches = BatchTable.Stretch(inBatch);
        else if userDefined3:
            outBatches = BatchTable.Split(inBatch);
        else:
            outBatches.Copy(inBatch);
Interface for Creating Policies

- Executors continue to execute the accepted batches and push new batches into the next batch queue.

```cpp
// stage executors run within a while loop
void Run()
{
  Batch& inBatch = BatchQueue.Get();
  CheckBuffer(inBatch); // Check buffer pair
  Execute(inBatch);
  Schedule(inBatch, outBatches);
  for (auto& batch : outBatches):
    nextBatchQueue.Push(batch);
  getBuffer(); // get legitimate buffer pair
  updatePrExecutor(); // update preceding executor

  // call Schedule to perform meta operations
  void Schedule(Batch& inBatch, vector<Batch>& outBatches):
    BatchTable.update(inBatch);
    if userDefined1:
      outBatches = BatchTable.New(inBatch);
    else if userDefined2:
      outBatches = BatchTable.Stretch(inBatch);
    else if userDefined3:
      outBatches = BatchTable.Split(inBatch);
    else:
      outBatches.Copy(inBatch);
}
Interface for Creating Policies

- Executors continue to execute the accepted batches and push new batches into the next batch queue.

- Users can define conditions to adjust the on-going batch with three meta operations.

```c++
// stage executors run within a while loop
void Run()
{
    Batch& inBatch = BatchQueue.Get(); // Check buffer pair
    CheckBuffer(inBatch);
    Execute(inBatch);
    Schedule(inBatch, outBatches);
    for (auto& batch : outBatches):
        nextBatchQueue.Push(batch);
    getBuffer(); // get legitimate buffer pair
    updatePrExecutor(); // update preceding executor

    // call Schedule to perform meta operations
    void Schedule(Batch& inBatch, vector<Batch>& outBatches):
        BatchTable.update(inBatch);
        if userDefined1:
            outBatches = BatchTable.New(inBatch);
        else if userDefined2:
            outBatches = BatchTable.Stretch(inBatch);
        else if userDefined3:
            outBatches = BatchTable.Split(inBatch);
        else:
            outBatches.CopyTo(inBatch);
```
Policies for Three Diversities

• Input Diversity: split the accepted batch into small batches with similar sequence length and run them in parallel.

• Operator Diversity: split the accepted batch at specific stage and run them in serial.

• Load Diversity: stretch the insufficient batch with newly arrived queries.
Evaluation

03
Experiment Setup

• Benchmarks:
  • BertBase, BertLarge: input and load diversity.
  • Unet, LinkNet: operator and load diversity
  • VGG19, ResNet152: load diversity

• Load Generating:
  • Arrival pattern: Poisson distribution
  • Input pattern: GLUE dataset

<table>
<thead>
<tr>
<th>Hardware</th>
<th>CPU: Intel Xeon E5-2620, GPU: Nvidia Titan RTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS &amp; Driver</td>
<td>Ubuntu: 18.04.6 (kernel 4.15.0); GPU Driver: 470.57</td>
</tr>
<tr>
<td>Software</td>
<td>CUDA: 11.4; TensorRT: 8.03; Triton 21.10</td>
</tr>
<tr>
<td>Benchmarks</td>
<td>Unet [56]; LinkNet [16]; BertBase; BertLarge [27]; VGG19 [57]; ResNet152 [37]</td>
</tr>
<tr>
<td>Dataset</td>
<td>GLUE [59]</td>
</tr>
</tbody>
</table>
End to End Results

• Baselines:
  • ZeroBatch: batching with no extra time window.
  • DelayBatch: batching with a hand-tuned time window.

• Compared with ZeroBatch: 16.1%/39.0%/57.7% average latency reduction under high, medium, and low load.
• Compared DelayBatch: 35.4%/47.3%/48.5% average latency reduction under high, medium, and low load.
Robustness at Stepping Load

- All the benchmarks have lower latency with DVABatch than with DelayBatch in all cases.
- DVABatch increases 46.81% peak throughput for BertBase, $1.37 \times$ peak throughput for BertLarge.
Contributions

• We propose a multi-entry multi-exit batching scheme for efficient DNN service processing on GPUs.

• We provide a general scheduling mechanism that leverages meta operations, and state transition diagram to create policies for different serving diversities.

• We reduce 46.4% average latency and achieve up to 2.12× throughput improvement for the involved serving diversities.
Implications

• Omnipresent Diversity.
  • Existing DNNs may have a dynamic architecture in depth, width, and routing. As more dynamic attributes emerge, diversity spreads across new DNNs.
  • DVABatch is flexible to tackle diversities mentioned above, like layer-skip, early-exiting models.

• Intra-model Scheduling.
  • As DNNs grow larger and show more diversity, the execution of DNNs cannot be treated as a single function call.
  • Intra-model scheduling is a trend for future DNN inference of large models.
Thanks! Q&A

Weihao Cui: weihao@sjtu.edu.cn