HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism

Jay H. Park,
Gyeongchan Yun, Chang M. Yi, Nguyen T. Nguyen, Seungmin Lee,
Jaesik Choi†, Sam H. Noh, and Young-ri Choi

†
Motivation

- DNN (Deep Neural Network) models continue to grow

- Need more powerful GPUs for training!
Motivation

- Short release cycle of new GPU architectures

Use of heterogeneous GPUs is inevitable!
What to do with *whimpy* GPUs?
DNN Training

Minibatch $i$ (Training Data)

- Forward Pass $i$
- Backward Pass $i$
- Weight Parameter $w$
- $w_{i+1} = w_i - \eta \cdot u_i$

Cat?
Loss
Parallelizing DNN Training

- **Data parallelism (DP)**
  - Worker 1
  - Worker n
  - Weights synchronized through **PS** or **AllReduce**
  - GPU memory limitation

- **Model parallelism (MP)**
  - **Parameter Server (PS)**
  - Low GPU utilization
Attempts to improve MP utilization
• Pipelined model parallelism (PMP)

- PipeDream [SOSP’19]
- GPipe [NIPS’19]

PMP Worker

- Designed for homogeneous GPUs
- Designed for a single PMP worker
HetPipe in a Nutshell

Support Heterogeneous GPUs

Integrates PMP + DP

VW: A group of multiple GPUs

Virtual Worker (VW) 1

VW n

(Wave Synchronous Parallel)

Parameter Server
Challenges in integration PMP+DP in Heterogeneous GPUs

- What weight version should be used by each VW to synchronize with other VWs?

- How do we reduce virtual worker stragglers when we consider DP?

Many more in the paper
HetPipe Contributions

Enable Large DNN Training on Heterogeneous GPUs
Aggregate heterogeneous resources
Reduce the straggler problem

Integrates PMP + DP
Novel parameter synchronization model
WSP (Wave Synchronous Parallel)

Proof of WSP Convergence
HetPipe Workflow

Cluster Configuration

Resource Allocator
Assign $k$ GPUs to each virtual worker

DNN Model

Model Partitioner
Divide model into $k$ partitions

VW 1

VW $n$

PS

Time

P1
P2
P3
P4

P1'
P2'
P3'
P4'

VW 1

VW $n$

P4 R
P3 R
P2 G
P1 G

P4' V
P3' V
P2' Q
P1' Q

Assign $k$ GPUs to each virtual worker.

Divide model into $k$ partitions.
HetPipe Workflow

Cluster Configuration

Resource Allocator
Assign $k$ GPUs to each virtual worker

DNN Model

Model Partitioner
Divide model into $k$ partitions

Divide model into $k$ partitions

VW 1

P4

P3

P2

P1

VW 1

P4'

P3'

P2'

P1'

VW n

P4

P3

P2

P1

VW n

P4'

P3'

P2'

P1'

Global

Local

Staleness

PS

Local

Global
· Motivation & Background
· HetPipe in a Nutshell

· Our System: HetPipe
  · Pipelined Model Parallelism Within a VW
  · Data Parallelism with Multiple VWs

· Evaluation
· Conclusion
### Pipelined Model Parallelism Within a VW

#### Execution of a virtual worker

<table>
<thead>
<tr>
<th>GPU4</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>6</th>
<th>6</th>
<th>7</th>
<th>7</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>GPU2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>GPU1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

$N_m$ minibatches processed concurrently in pipeline manner

$W_{\text{local}}$ is a consistent version of weights within a VW
### Weight management procedure

**Initial weight version \( w_0 \)**

\[ w_{local} = w_0 = w_1 = w_2 = w_3 = w_4 \]

**Update \( u_1 \)**

\[ w_5 \leftarrow w_{local} \]

\[ w_{local} \leftarrow w_{local} + u_1 \]
### Local staleness ($S_{local}$): maximum missing updates

<table>
<thead>
<tr>
<th>Time</th>
<th>GPU1</th>
<th>GPU2</th>
<th>GPU3</th>
<th>GPU4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

$S_{local} = 3$  

$w_5 \leftarrow w_{local}$

$w_5$ missing updates of minibatches 2 to 4

$w_{local} \leftarrow w_{local} + u_1$
Pipelined Model Parallelism Within a VW

- **Local staleness ($S_{local}$): maximum missing updates**

![Diagram showing pipeline model parallelism between four GPUs with forward and backward passes.]

- $S_{local} = 3$
- $w_6$ missing updates of minibatches 3 to 5
- $w_{local} \leftarrow w_{local} + u_2$
- $w_{0} + u_1$
Motivation & Background

HetPipe in a Nutshell

Our System: HetPipe
- Pipelined Model Parallelism Within a VW
- Data Parallelism with Multiple VWs

Evaluation

Conclusion
Data Parallelism with Multiple VWs

**Wave**: Sequence of concurrently executing $N_m$ minibatches

Progress of minibatch execution

Parameter Server: $w_{global}$
Data Parallelism with Multiple VWs

- **Push** occurs every clock

```
0 1
  0  1  2  3  4  5  6  7  8
  |   |   |   |   |   |
  ↑   ↑   ↑   ↑   ↑   ↑
VW 1 VW 2 VW 3 VW 4 VW 5
```

```
VW 1
VW 2
VW 3
VW 4
VW 5
```

Block minibatch 8

**Push** aggregated updates of wave0 ($\tilde{u}$)

\[
\tilde{u} = u_1 + u_2 + u_3 + u_4
\]

\[
w_{global} \leftarrow w_{global} + \tilde{u}
\]

Parameter Server: $w_{global}$
Pull occurs intermittently - Depending on user defined clock distance $D$

- If $D = 0$ pull occurs every clock

VW1 waits before pull until VW2 pushes

Parameter Server: $w_{global}$
Data Parallelism with Multiple VWs

- **Pull** occurs intermittently - Depending on user defined clock distance $D$

If $D = 0$

VW1 waits before pull until VW2 pushes

VW2 Push aggregated updates ($\tilde{u}$)

$$w_{global} \leftarrow w_{global} + \tilde{u}$$
Pull occurs intermittently - Depending on user defined *clock distance* $D$

If $D = 0$

**Pull** occurs after all VWs have been pushed

$w_{local} \leftarrow w_{global}$
Data Parallelism with Multiple VWs

- **Pull** occurs intermittently - Depending on user defined *clock distance D*

![Diagram](image)

- **Push** & Pull

Parameter Server: $w_{global}$

If $D = 0$

$w_8 = w_0 + (u_1 + u_2 + u_3 + u_4)_{vw1,vw2}$
Data Parallelism with Multiple VWs

- **Local staleness** \( S_{local} \) and **global staleness** \( S_{global} \) with WSP

\[
W_{global} = w_0 + (u_1 + u_2 + u_3 + u_4)_{vw1,vw2}
\]

\[
S_{global} = (u_5 + u_6 + u_7)_{vw1}
\]

\[
S_{local} = (u_8 + u_9 + u_{10})_{vw1}
\]
Local staleness ($S_{local}$) and global staleness ($S_{global}$) with WSP

Minibatch 12 has to wait
Data Parallelism with Multiple VWs

- **Example of clock distance threshold** $D$

  If $D = 1$

  Can start minibatch 8 without *pull*

**Push & Pull**

Parameter Server: $w_{global}$
Example of clock distance threshold $D$

If $D = 1$

Minibatch 12 has to wait

$w_{11} = w_0 + (u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_7)_{vw1}$

$S_{global} \rightarrow S_{local}$

$(u_8 + u_9 + u_{10})_{vw1}$

$(u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_7)_{vw2}$

$w_{global} = w_0$
Outline

- Motivation & Background
- HetPipe in a Nutshell
- Our System: HetPipe

Evaluation
  - Setup
  - Resource Allocation for Virtual Workers
  - Results

Conclusion
Evaluation Setup

- **Cluster setup - 4 heterogeneous GPU nodes**

  ![Diagram showing 4 GPU nodes connected via InfiniBand (56 Gbps)]

- **Two DNN models**

<table>
<thead>
<tr>
<th></th>
<th>ResNet-152</th>
<th>VGG-19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset, minibatch size</td>
<td>ImageNet, 32</td>
<td></td>
</tr>
<tr>
<td>Model parameter size</td>
<td>230 MB</td>
<td>548 MB</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Large activation output</td>
<td>Large parameter size</td>
</tr>
</tbody>
</table>
Resource Allocation for Virtual Workers: NP, ED, HD

- NP (Node Partition)

- Minimum communication overhead within VW
- Performance of each virtual worker varies
- Straggler may degrade performance with DP
ED (Equal Distribution)

- Performance will be the same across the VWs
- Mitigates the straggler problem
- High communication overhead within each VW
**HD (Hybrid Distribution)**

- Mitigates the straggler problem
- Reduces communication overhead within each VW
- Round-robin policy (default)
  - Can be used in all three policies: NP, ED, and HD

Example: ED

Parameters of each layer:
Local placement policy

- ED-local

Parameters of each layer:

- Significantly reduces communication overhead
- Parameter communication occurs
Baseline Horovod

- State-of-the-art DP using AllReduce

- For ResNet-152, the whole model is too large to be loaded into a single type GPU (batch size = 32)

- ED: reduces the straggler problem
- ED-local: significantly reduces communication overhead

Compare Throughput with Horovod
Performance Improvement of Adding Whimpy GPUs

- With additional GPUs, HetPipe achieves up to 2.3X speed up
- Additional whimpy systems allow for faster training
Convergence Results

- **ResNet-152**

  ![Graph showing convergence results](image)

  - **Target accuracy: 74%**
  - **Up to 39% faster**
  - **HetPipe reduces straggler problem in heterogeneous environment**
  - **Adding four more whimpy G GPUs, performance improves even more**

  - **7% faster**

- **Horovod 12GPUs**
- **HetPipe 12GPUs**
- **HetPipe 16GPUs**
**VGG-19**

- **Convergence Results**

  - Target accuracy: 67%
  - HetPipe (D=0) is 29% faster than Horovod
  - Up to 49% faster
  - Higher global staleness (i.e., 32) can degrade convergence performance
  - 4.7% slower

---

**Graph Details**

- **Axes:**
  - X-axis: Time (hours)
  - Y-axis: Top-1 Accuracy

- **Legend:**
  - Horovod
  - HetPipe D=0
  - HetPipe D=4
  - HetPipe D=32
- Provide convergence proof of WSP
- Partitioning algorithm
- Performance of a single virtual worker
- Comparison to PipeDream
HetPipe makes it possible to efficiently train large DNN models with heterogeneous GPUs

- Integrate pipelined model parallelism with data parallelism
- Propose a novel parameter synchronization model: WSP
- DNN models converge up to 49% faster with HetPipe
Thank you!