# Interface for Model Parallelism

**Author**
Dr. Martin Kühn

May 14, 2020

**Abstract**

# Contents

# Chapter 1

# Definition

## 1.1 Input parameter

$r$ is the rank of the current process.

$M = r_0, r_1, \ldots$ is the full machine consisting of all the ranks. The list is ordered. $|M|$ is the number of ranks.

$D$ is the decompostion of a tensor over the ranks of the machine as follows $\mathcal{N} \times \mathcal{C} \times \mathcal{H} \times \mathcal{W}$.

# Chapter 2

# Tarantella

Tarantella should provide a function that returns a pointer to a buffer in a GPI segment.

get_segment_buffer(size)

# Chapter 3

# Convolutionial layer

The layer must have global parameters describing the global parameters. These are $N, C, F, H, W$. $N$ is the batch size, $C$ the number of input channels, $F$ the number of output channels or filters, $H$ the height and $W$ the width of the images.

Additionally for a distributed convolutional layer we need to know the following.

The own rank $r$ and the machine $M$.

The distribution of the input tensor over the dimensions $\bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}}$. Here $\bar{\mathcal{N}}$ is not the number of samples but the number of instances this dimenstion is distributed over and so on. So $\bar{\mathcal{N}} = 1$ means no distribution.

The distribution of the output tensor over the dimensions $\hat{\mathcal{N}} \times \hat{\mathcal{C}} \times \hat{\mathcal{H}} \times \hat{\mathcal{W}}$.

Maybe we want to give strides for distribution of the dimensions over the ranks. $\tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}}$ or $\breve{\mathcal{N}} \times \breve{\mathcal{C}} \times \breve{\mathcal{P}}$. Here stands $\tilde{\mathcal{P}}$ for $\tilde{\mathcal{H}} \times \tilde{\mathcal{W}}$.

Further we allow

$$\begin{aligned}
\bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{P}} && (3.1)\\
\bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{C}} \times \tilde{\mathcal{P}} && (3.2)\\
\bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}} \leq |M| && (3.3)\\
\hat{\mathcal{H}} \times \hat{\mathcal{W}} &\leq \breve{\mathcal{P}} && (3.4)\\
\hat{\mathcal{C}} \times \hat{\mathcal{H}} \times \hat{\mathcal{W}} &\leq \breve{\mathcal{C}} \times \breve{\mathcal{P}} && (3.5)\\
\hat{\mathcal{N}} \times \hat{\mathcal{C}} \times \hat{\mathcal{H}} \times \hat{\mathcal{W}} &\leq \breve{\mathcal{N}} \times \breve{\mathcal{C}} \times \breve{\mathcal{P}} \leq |M| && (3.6)
\end{aligned}$$

Later on we would need a parameter that defines distribution of the weights. Currently we are distributing the images only, so the weights are copied and synchronized always.

Further we would need resources like GPI segments and allreduces.

In summary we have

$(N, C, F, H, W)$
$r$
$M$
$\bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}}$
$\tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}}$
$\hat{\mathcal{N}} \times \hat{\mathcal{C}} \times \hat{\mathcal{H}} \times \hat{\mathcal{W}}$
$\breve{\mathcal{N}} \times \breve{\mathcal{C}} \times \breve{\mathcal{P}}$
(distribution pattern weights)
GPI segmnts allreduce double buffer

## 3.1 Callback functions

The ranks to synchronize the weights is returned by a callback function.

get_ranks_synch_weights().

The size of the gradients is returned.

get_sizeof_gradients()

Tarantella has to create an allreduce (double buffer) for this layer if the return of get_ranks_synch_weights() is not empty. The size is given by get_sizeof_gradients().

get_sizeof_input_tensor() get_sizeof_output_tensor()

## 3.2 Implementation

Tarantella creates an allreduce of size get_sizeof_gradients() on the ranks given by get_ranks_synch_weights(). Tarantella delivers pointers within GPI segments to store the local gradients in.

The implementation puts the local gradients directly into the segments.

The Tarantella function get_segment_buffer(size) is used to allocate a buffer in a GPI segment. On the first call they exchange a pointer to the exchange buffers with all their neighbors.

### 3.2.1 Forward pass

- start collecting all the input data needed
- calculate convolution with local data
- calculate convolution with remote data
- fuse both data

### 3.2.2 Backward pass

- start collecting all the input data needed
- calculate gradient weights with local data
- calculate error signal with local data

- calculate gradient weights with remote data
- calculate error signal with remote data
- fuse data

# Chapter 4

# Instance Normalisation

The layer must have global parameters describing the global parameters. These are $N, C, H, W$. $N$ is the batch size, $C$ the number of input channels, $H$ the height and $W$ the width of the images.

Additionally for a distributed convolutional layer we need to know the following.

The own rank $r$ and the machine $M$.

The distribution of the input tensor over the dimensions $\bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}}$. The dimension of the input tensor is the dimension of the output tensor.

Maybe we want to give strides for distribution of the dimensions over the ranks. $\tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}}$. Here stands $\tilde{\mathcal{P}}$ for $\tilde{\mathcal{H}} \times \tilde{\mathcal{W}}$.

Further we allow

$$\begin{aligned} \bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{P}} && (4.1) \\ \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{C}} \times \tilde{\mathcal{P}} && (4.2) \\ \bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}} &\leq \tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}} \leq |M| && (4.3) \end{aligned}$$

Further we would need resources like allreduces.

In summary we have

$(N, C, H, W)$

$r$

$M$

$\bar{\mathcal{N}} \times \bar{\mathcal{C}} \times \bar{\mathcal{H}} \times \bar{\mathcal{W}}$

$\tilde{\mathcal{N}} \times \tilde{\mathcal{C}} \times \tilde{\mathcal{P}}$

allreduce double buffer weights allreduce double buffer norm

The allreduce double buffer norm is not part of the gradients update but is used inside the layer to reduce the norm. Latency is critical here.

## 4.1 Callback functions

The ranks to synchronize the weights is returned by a callback function.

get_ranks_synch_weights().
The size of the gradients is returned.
get_sizeof_gradients().
The ranks to synchronize the norm is returned by a callback function
get_ranks_synch_norm().
The size of the norm is returned by
get_sizeof_norm().

Tarantella has to create two allreduces (double buffer) for this layer if the return of get_ranks_synch_weights() andget_ranks_synch_norm() are not empty. The size is given by get_sizeof_gradients() and get_ranks_synch_norm().

get_sizeof_tensor()

## 4.2 Implementation

### 4.2.1 Forward pass

- compute local norm
- allreduce norm
- perform scaling

### 4.2.2 Backward pass

Not sure if it works like this:

- calculate local gradient of norm
- calculate local error signal of norm
- allreduce local data
- perform backward gradient of norm.
- perform backward gradient of error signal

# Chapter 5

# Concat