UNIVERSITÀ DI PAVIA

Introduce Parallelism into Forward Propagation of Convolutional Layer with MPI

Naoya Kumakura
Department of Computer Engineering
Università di Pavia, Italy
Email: naoya.kumakura01@universitadipavia.it
Date: 2025/6/09
Github:
https://github.com/naoya526/aca-project-naoya

Abstract

CNN(Convolutional Neural Network) is one of the most dominant way for Solving Computer Vision Task Nowadays, but it requires huge amount of computational power. In this report, I implemented Batch parallelism in forward propagation of Convolutional Layer with MPI and made analysis of Memory Usage, Execution Time, Comparison of Fat/Light Cluster.

1 Analysis of the serial algorithm

In this section, the shape of the algorithm are defined as follows: **Convolutional2D**

- input: [batch size, in channels, height, width]
- kernel: [out channels, in channels, kernel size]
- output: [batch size, out channels, height-kernel size+1, width-kernel size+1]

Since this algorithm is simple calculation of each scalar of the kernel and corresponding scalar of the input, it is highly possible to parallelize.

Let x be the input tensor of shape $(B, C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$, w be the kernel weights of shape $(C_{\text{out}}, C_{\text{in}}, K)$, and y be the output tensor of shape $(B, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$, where $H_{\text{out}} = H_{\text{in}} - K + 1$ and $W_{\text{out}} = W_{\text{in}} - K + 1$.

Then the forward pass of a convolutional layer is given by:

$$y_{b,c_{\text{out}},h,w} = \sum_{c_{\text{in}}=0}^{C_{\text{in}}-1} \sum_{k_h=0}^{K-1} \sum_{k_w=0}^{K-1} x_{b,c_{\text{in}},h+k_h,w+k_w} \cdot w_{c_{\text{out}},c_{\text{in}},k_h,k_w} \tag{1}$$

for each batch index b, output channel c_{out}, and output spatial location (h, w). For the explanation, this is the example of without padding and stride 1.

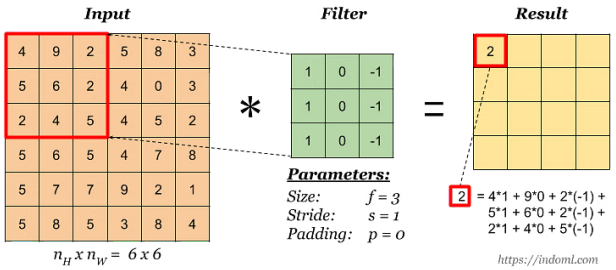

Figure1 CNN Structure

* indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/.

The primary objective of this study is to evaluate the performance of parallel processing techniques on a CPU-based cluster using OpenMPI. In particular, we focus on assessing the scalability and efficiency of data-parallel execution by distributing computation across multiple processes. Rather than utilizing GPU acceleration (e.g., via CUDA), this work emphasizes **message-passing-based parallelization**, with the goal of analyzing the effectiveness of MPI in handling computationally intensive tasks such as convolutional neural network (CNN) inference. Accordingly, the experiments are conducted exclusively on a CPU cluster environment without GPU support.

2 A-priori study of available parallelism

2.1 Amdahl's Law – Strong Scalability

Amdahl's Law expresses the theoretical speedup limit of a program using parallel processing. If a fraction p of a program is parallelizable and the execution runs on N processors, the speedup $S(N)$ is given by:

$$S(N) = \frac{1}{(1-p) + \frac{p}{N}} \tag{2}$$

This model assumes a fixed problem size, making it suitable for strong scalability analysis. It indicates that the serial fraction $(1-p)$ bounds the maximum speedup, regardless of the number of processors.

The CNN forward propagation is divided into four steps:

1. data loading
2. data transfer
3. data processing (parallelizable)
4. result aggregation.

Approximately 90% of the workload is parallelizable ($p = 0.9$). Thus, the maximum theoretical speedup is

$$S(N) = \frac{1}{0.1 + \frac{0.9}{N}} \tag{3}$$

which asymptotically approaches 10 as $N \to \infty$. The maximum number of processor which was used in this report was 8. Hence, Estimated $S(8) \approx 4.7$. In practice, (in ordinary case) communication overhead and load imbalance reduce the actual speedup, making it lower than this theoretical upper bound.

2.2 Gustafson's Law – Weak Scalability

Gustafson's Law addresses the limitations of Amdahl's model by considering scenarios where the problem size scales with the number of processors. Under this assumption, the speedup $S_G(N)$ is expressed as:

$$S_G(N) = N - (1 - p)(N - 1) \tag{4}$$

Here, p represents the fraction of the workload that can be parallelized. As N increases, $S_G(N)$ grows linearly, provided p is close to 1. This model is more appropriate for weak scalability analysis, where the goal is to maintain constant execution time while increasing workload with processor count. Based on the analysis which was on 2.1,

$$S_G(N) = N - (1 - 0.9)(N - 1) \tag{5}$$

$S_G(8) \approx 7.3$

3 MPI parallel implementation in C

Listing 1 Implementation of MPI parallelism

```
MPI_Barrier(MPI_COMM_WORLD); // all process are synchronized
    start_time = time(NULL);
    double start = MPI_Wtime();

    // Broadcast Kernel
    MPI_Bcast(kernel, OC * IC * K * K, MPI_FLOAT, 0, MPI_COMM_WORLD);

    // Scattering input
    MPI_Scatter(input_full, B_local * IC * H * W, MPI_FLOAT,
                input_local, B_local * IC * H * W, MPI_FLOAT,
                0, MPI_COMM_WORLD);

    // Convolution
    conv2d_forward(input_local, kernel, output_local, B_local, IC, OC, H, W, K);

    // Gather the output
    MPI_Gather(output_local, B_local * OC * out_H * out_W, MPI_FLOAT,
               output_full, B_local * OC * out_H * out_W, MPI_FLOAT,
               0, MPI_COMM_WORLD);
```

4 Testing and debugging

First, I executed on Local Machine. Table 1 shows the enviroment which the application is executed.

Table1 Environment and Configurations

Programming language	C(for MPI), Python(for loading image), Terraform(For deploy VM on GCP)
stdlib.h	`mpi.h`, `time.h`, `csv`, `string`, `torch`,
Evaluation metrics	Execution Time
Environment	Local machine (Ubuntu 22.04.5 LTS), GCP Virtual Machine(See "Cloud Deployment" for Detail)

stdlib.h is used for adminstion of memory and producing random Input. This enable various input. OpenMPI(mpi.h) is the fundamental application for this report. time.h is used for checking if the function of measuring executed time in OpenMPI.

4.1 Dataset

4.1.1 Configuration of Dataset

This report is to make the analysis of CNN parallelism implementation in **practical use case, not like toy case**. Hence, I decided to use **1024 pixel** color image as input:

$$input = (B, C_{\text{in}}, H_{\text{in}}, W_{\text{in}}) = (B, 3, 1024, 1024)$$

Kernel parameters are defined as follows:

$$kernel = (C_{\text{out}}, C_{\text{in}}, K) = (3, 3, 1)$$

Output:

$$output = (B, C_{\text{out}}, H_{\text{in}}, W_{\text{in}}) = (B, 3, 1022, 1022)$$

Figure2 Input(Left) and Output(Right)

4.1.2 Problem in Data Division

The parallel implementation distributes the batch of data across N processors. However, when B/N is not an integer, the data cannot be evenly divided, leading to load imbalance and varying execution times across processors. To avoid this and enable accurate performance analysis, the batch size B is chosen so that B/N is an integer. This ensures equal workload distribution and validates the correctness of the implementation.

4.2 Pre-experiment on Local

Table 2 shows the local execution with 4 thread on local. Master node has high memory usage, which is almost approximated to 4 times slave node's memory usage(71MB × 4=284)

Table2 MPI Execution Summary

Rank	Batch Range (start:end)	Local Batch Size	Memory Usage (MB)
0	0:1	1	312.52
1	1:2	1	71.14
2	2:3	1	71.50
3	3:4	1	71.10
Output Shape: (20, 3, 1022, 1022)			
Max Execution Time (across ranks): 0.506575 sec			
Total Memory Usage: 526.25 MB			

4.3 Analysis with mpiP

mpiP[*1] is one of the useful tool to analyze MPI parallelism implementation. the version is 3.5.0(latest). 3 shows Execution Time in Each Rank, MPI Time(the time MPI job executed), and MPI% which means how ratio MPI parallel job are included. Rank 0 is highly assign for non parallel job such as loading data, gathering data, and so on. All analysis result is on my Github pages.

4.4 Cloud Deployment

For the cloud deployment, Terraform was used. Terraform is used for making IaC (Infrastructure as Code), allowing reproducible and automated provisioning of cloud resources. In this project, it was used to configure virtual machines, networking, and storage on GCP. The MPI-based application was deployed to VM instances. SSH key pairs and security groups were also managed through Terraform to ensure secure and scalable deployment.

[*1] https://github.com/LLNL/mpiP

Rank	Execution Time	MPITime	MPI%
0	0.950	0.220	23.17
1	0.921	0.919	99.79
2	0.921	0.919	99.80
3	0.921	0.919	99.80

Table3 Task Data

5 Performance and scalibility analysis

In this section, parallelized algorithm are examined with Each two cluster(**Fat Cluster**/**Light Cluster**). Index are shown below:

- Execution Time, Speedup
- Total/Per-Rank Memory Usage

- **Fat Cluster** - n2-highcpu (16GB memory, 16core) × 1
- **Light Cluster** - e2-medium (4GB memory, 1core) × 8

Compared to another high-power Virtual Machines(such as), n2-highcpu has more CPU, less memory size. for using a lot of CPU such as MPI task, it is sutable. e2-medium has 1 core, which is sutable for making Light Cluster.

5.1 Fat Cluster

5.1.1 Strong Scalability of Fat Cluster

In Figure 3, the observed speedup exceeds the ideal speedup (depended on Amdahl's law, p = 0.9), which is theoretically impossible. This suggests inefficiencies in the single-threaded execution. If the single-process computation were properly optimized, the resulting speedup would likely remain within the bounds of the ideal speedup. Table 5 shows high memory usage on the master node, which leads to delays and becomes a bottleneck in the computation. To mitigate this bottleneck, one possible optimization is to save the results locally and later merging them(Distributed Data Sav-

ing). Another possible way is gathering step by step, with making tree structure(Gather with tree structure model).

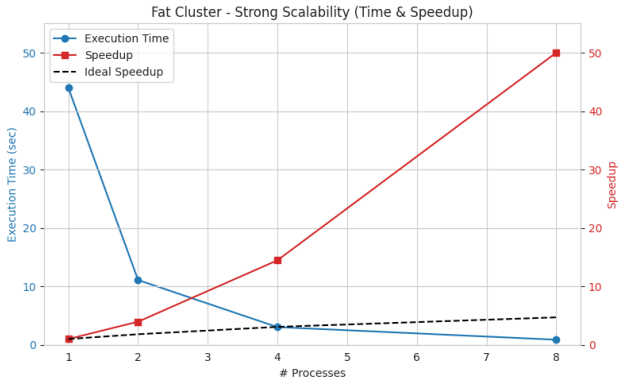

Figure3 Execution Time and Speedup in FatCluster with Strong Scalability

Table4 Fat Cluster Strong Scalability

# Processes	Execution Time (max)	Total Memory Usage (MB)	Output Shape
1	44.02 sec	1034.58	(80, 3, 1022, 1022)
2	11.10 sec	1107.22	(80, 3, 1022, 1022)
4	3.04 sec	1253.28	(80, 3, 1022, 1022)
8	0.88 sec	1541.04	(80, 3, 1022, 1022)
16	-killed process-	-	-

Table5 Per-Rank Memory Usage

# Proc	Rank 0	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Total Mem (MB)	Output Shape
1	1034.58	–	–	–	–	–	–	–	1034.58	(80, 3, 1022, 1022)
2	1034.45	72.77	–	–	–	–	–	–	1107.22	(80, 3, 1022, 1022)
4	1034.35	72.91	73.28	72.74	–	–	–	–	1253.28	(80, 3, 1022, 1022)
8	1034.45	73.06	72.88	72.95	70.81	70.89	72.99	73.01	1541.04	(80, 3, 1022, 1022)

5.1.2 Weak Scalability of Fat Cluster

In Figure 4, Execution time, Speedup and Ideal Speedup (based on Gustafson's law, p = 0.9).Theoretical/actual Speedup shows quite similar move. this shows my assumption was almost correct. The greater the problem becomes, The smaller the part of problem which is impossible to parallelize and each processor's process is not change. These effect enables the Cluster to scale in high efficiency. (Speedup is total speed up, and this does not mean that each processor can calculate efficiently.) As pointed out in Strong Scalability, master node has high memory usage. This could hinder the scalability of further calculation with larger dataset.

5.1.3 analysis

The 16-process configuration could not be measured due to insufficient memory resources. Given that each MPI rank consumes approximately 1 GB of memory at peak, the total memory required exceeded the available 14.4 GB RAM of the n2-highcpu-16 virtual machine, resulting in failure to execute the full 16-process run. This suggests that memory usage is a limiting factor in the current implementation, and optimizing memory consumption per process would be necessary to scale beyond 8 processes. This master node memory shortages comes from the "MPI_Gather" operation, which gather calculation results from all other nodes.

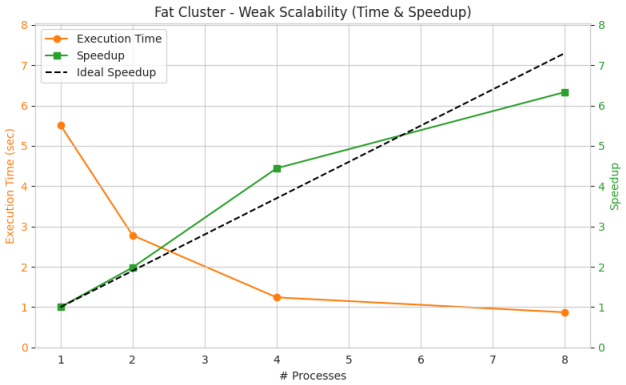

Figure4 Execution Time and Speedup in FatCluster with Weak Scalability

Table6 Fat Cluster Weak Scalability

# Processes	Execution Time (max)	Total Memory Usage (MB)	Output Shape
1	5.51 sec	194.39	(10, 3, 1022, 1022)
2	2.78 sec	387.45	(20, 3, 1022, 1022)
4	1.24 sec	775.20	(40, 3, 1022, 1022)
8	0.87 sec	1542.19	(80, 3, 1022, 1022)
16	-killed process-	-	-

Table7 Per-Rank Memory Usage (Weak Scaling)

# Proc	Rank 0	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Total Mem (MB)	Output Shape
1	194.39	–	–	–	–	–	–	–	194.39	(10, 3, 1022, 1022)
2	314.57	72.88	–	–	–	–	–	–	387.45	(20, 3, 1022, 1022)
4	556.62	72.77	72.84	72.96	–	–	–	–	775.20	(40, 3, 1022, 1022)
8	1034.52	72.79	72.71	72.90	70.70	72.70	72.93	72.94	1542.19	(80, 3, 1022, 1022)

5.2 Light Cluster

This section points out the influence of weak Master Node. Light Cluster(e2-medium) has small memory(4GB,1core). Hence, in the Section 5.2.1, it was impossible to execute with Batch Size $B = 80$, so alternatively it was executed with $B = 40$.

5.2.1 Strong Scalability of Light Cluster

Figure 5 shows the results of Execution Time, Speedup and Ideal Speedup(based on Ahamdal's law, p = 0.9) Speedup exceed Ideal Speedup slightly. Executing Parallel Algorithm with one thread could cause slight ineffeciency, leading to this delay.

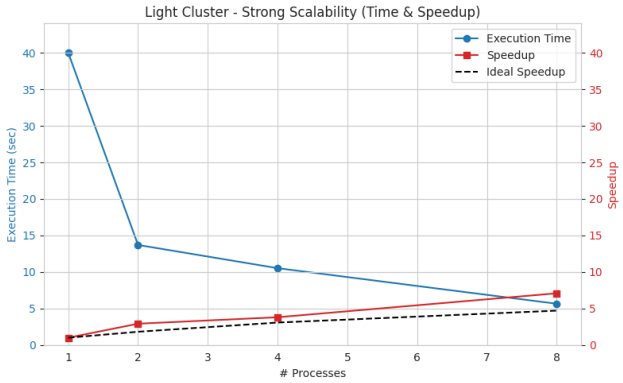

Figure5 Execution Time and Speedup in Light Cluster with Strong Scalability

Table8 Light Cluster Strong Scalability

# Processes	Execution Time (max)	Total Memory Usage (MB)	Output Shape
1	40.129868 sec	1926 MB	(40, 3, 1022, 1022)
2	13.703938 sec	1449 + 491 MB	(40, 3, 1022, 1022)
4	10.524437 sec	1187 + 3 × 229 MB	(40, 3, 1022, 1022)
8	5.648217 sec	1034 + 7 × 76 MB	(40, 3, 1022, 1022)

Table9 Per-Rank Memory Usage

# Proc	Rank 0	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Total Mem (MB)	Output Shape
1	1926	–	–	–	–	–	–	–	1926	(40, 3, 1022, 1022)
2	1449	491	–	–	–	–	–	–	1940	(40, 3, 1022, 1022)
4	1187	229	229	229	–	–	–	–	1874	(40, 3, 1022, 1022)
8	1034	76	76	76	76	76	76	76	1590	(40, 3, 1022, 1022)

5.2.2 Weak Scalability of Light Cluster

In figure 6,Execution time, Speedup and Ideal Speedup (based on Gustafson's law, p = 0.9). Ideal Speedup is Upperbound of Actual Speedup. This results shows overhead of having communication with numbers of processors. To some extent, Weak Scalability was dominant, but the greater the number of processor become, the heavier Overhead become. In light Cluster, e2-medium(4GB memory) is used and half of the memory was occupied(2150 MB). Gathering the infomation locally will reduce overhead will be one of the option for reducing this Overhead.

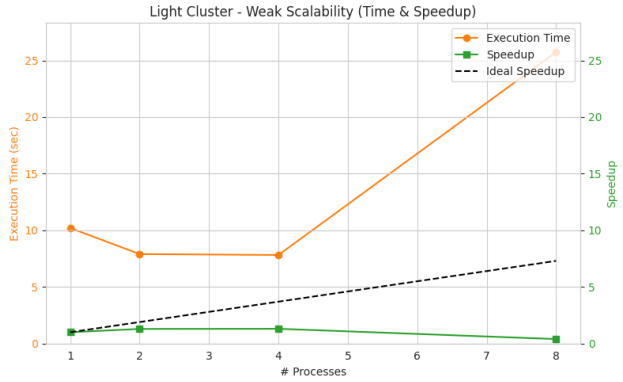

Figure6 Execution Time and Speedup in Light Cluster with Strong Scalability

Table10 Light Cluster Weak Scalability

# Processes	Execution Time (max)	Total Memory Usage (MB)	Output Shape
1	10.195611 sec	490 MB	(10, 3, 1022, 1022)
2	7.904873 sec	730 + 251 MB	(20, 3, 1022, 1022)
4	7.822986 sec	1187 + 3 × 229 MB	(40, 3, 1022, 1022)
8	25.733909 sec	2150 + 7 × 233 MB	(80, 3, 1022, 1022)

Table11 Per-Rank Memory Usage

# Proc	Rank 0	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Total Mem (MB)	Output Shape
1	490	–	–	–	–	–	–	–	490	(10, 3, 1022, 1022)
2	730	251	–	–	–	–	–	–	981	(20, 3, 1022, 1022)
4	1187	229	229	229	–	–	–	–	1874	(40, 3, 1022, 1022)
8	2150	233	233	234	233	233	233	231	3780	(80, 3, 1022, 1022)

6 Conclusion and Vision

From analysis of Fat Cluster, the Speedup exceed the theoretical Upperbound based on Amdahl's law. The reason could be the efficeiency in 1 thread Execution. From analysis of Light Cluster, shortage of Memory becomes bottleneck of the calculation, and it leads to stop VM working or make the execution time longer.

In this report, I implemented Batch parallelism, but other forms of parallelism remain to be explored(for example, Channels parallelism, Pixel(Height/Width) Par-allelism and even Kernel Parallelism). A comparison between GPU calculation and this extreme MPI parallelism implementation would be valuable.

7 Contribution

This project was independently conducted by Naoya Kumakura, with all implementation and reporting completed by the author. I would like to express my gratitude to everyone who read this report, my fellow students, and the supervising professor.