Large Language Model Application Development Guide on Arc dGPU by BigDL-LLM Python*

User Guide

December 2023

Document Number: 795058-0.5

Contents

Figures

Tables

Revision History

Date	Revision	Description
December 2023	0.5	Initial release.

§

1.0 Introduction

Recently, Artificial Intelligence Generated Content (AIGC) algorithms have emerged to resolve the challenges of digital intelligence in the digital economy, especially for the high demand of large language model (LLMs). Users may prefer LLM applications on a local device for personal data security.

This document provides the solution for large language model application development on Arc dGPU by Gradio and Intel BigDL-LLM package. The CPU platform should be 12th Gen Intel® Core™ Processors or future with 16 GB system memory or higher.

Intel BigDL-LLM seamlessly scales data analytics and AI applications from laptop to cloud, with LLM: Low-bit (INT3/INT4/INT5/INT8) large language model library for Intel CPU/GPU and so on.

Gradio is the fastest way to demo machine learning models with a friendly web interface so that anyone can use it anywhere!

§

2.0 Setup Environment and Development Guide

This section provides environment setup and a large language model application development guide on Ubuntu 22.04 with Arc A770 gGPU. Application support models like:

- ChatGLM2-6B: Chinese-English chat large language model
- LLaMa2-13B: English chat large language model
- StarCoder-15.5B: English code generation large language model

The following components are required, and we have validated them in this document:

Requirement	Description
System	12th Gen Intel® Core™ Processors or future. (Validated: 12th Gen Intel® Core™ Processors (Alder Lake i7-12700) on Linux* Ubuntu* 22.04.3.)
Python	3.9.16.
Pytorch	Pytorch 2.0.1a0+cxx11.abi
Pytorch extension	intel-extension-for pytorch 2.0.110+xpu
Torchvision	torchvision 0.15.2a0+cxx11.abi
Gradio	3.41.1
mdtex2html	1.2.0
Accelerate	0.23.0
Sentencepiece	0.1.99
bigdl-llm	2.4.0b20230827
bigdl-core-xe	2.4.0b20230827
Ubuntu	22.04
Kernel	>5.15
GPU driver	647.21

Arc DGPU Environment & Software Setup Guide is available at 779788.

2.1 Install OneAPI 2023.2 on Ubuntu 22.04

Bigdl-llm depends on Intel Extension for Pytorch for running on Arc Graphics. Intel Extension for Pytorch relies on the OneAPI DPC++ Compiler and Math Kernel Library. OneAPI is required for Intel Extension for Pytorch execution.

Open Terminal and run the following command to download and install oneAPI 2023.2:

```
wget https://registrationcenter-
download.intel.com/akdlm/IRC_NAS/992857b9-624c-45de-9701-
f6445d845359/l_BaseKit_p_2023.2.0.49397.sh

sudo sh ./l_BaseKit_p_2023.2.0.49397.sh -a -c -proxy http://proxy.com
```

2.2 Install Python Dependencies

Install Miniconda or Anaconda by downloading the package from https://docs.conda.io/projects/miniconda/en/latest/miniconda-other-installer-links.html and create Python 3.9 virtual environment.

Or run the following command to download Miniconda and create a Python 3.9 virtual environment.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-py39_23.5.2-0-
Linux-x86_64.sh

chmod +x Miniconda3-py39_23.5.2-0-Linux-x86_64.sh

 ./Miniconda3-py39_23.5.2-0-Linux-x86_64.sh

conda create -n llm python=3.9
```

Install dependencies by running the following commands on the Terminal:

```
pip install bigdl-llm[xpu] -f https://developer.intel.com/ipex-whl-
stable-xpu

pip install bigdl-core-xe

pip install gradio mdtex2html

pip install accelerate sentencepiece

pip install torch==2.0.1a0 torchvision==0.15.2a0
intel_extension_for_pytorch==2.0.110+xpu -f
https://developer.intel.com/ipex-whl-stable-xpu-idp
```

2.3 Convert and Load Model by Python

For example, download ChatGLM2, LLaMa2 and StarCoder models from hugging face.

- ChatGLM2-6B：https://huggingface.co/THUDM/chatglm2-6b/tree/main
- LLaMa2-13B: https://huggingface.co/meta-llama/Llama-2-13b-chat-hf/tree/main
- StarCoder-15.5B: https://huggingface.co/bigcode/starcoder/tree/main

Models can be saved in Bigdl-llm transformer INT4 format to reduce model loading time. Arc GPU supports transformer INT4 optimization in Bigdl-llm and doesn't support native INT4 optimization.

Use Python code to save the models into transformer INT4 format:

```
from bigdl.llm.transformers import AutoModel
from transformers import AutoTokenizer
from bigdl.llm.transformers import AutoModelForCausalLM
model_name = "chatglm2-6b"
model_all_local_path = "./"
model_name_local = model_all_local_path + model_name

if model_name == "chatglm2-6b":
    tokenizer = AutoTokenizer.from_pretrained(model_name_local,
trust_remote_code=True)
    model = AutoModel.from_pretrained(model_name_local,
trust_remote_code=True, optimize_model=False, load_in_4bit=True)
    model.save_low_bit("./chatglm2-6b-int4/")
    tokenizer.save_pretrained("./chatglm2-6b-int4/")

elif model_name == "llama2-13b" or model_name == "StarCoder":
    tokenizer = AutoTokenizer.from_pretrained(model_name_local,
trust_remote_code=True)
    model = AutoModelForCausalLM.from_pretrained(model_name_local,
trust_remote_code=True, optimize_model=False, load_in_4bit=True)
    model.save_low_bit("./"+model_name+"-int4/")
tokenizer.save_pretrained("./"+model_name+"-int4/")
```

Load transformer INT4 format model by Python:

```
if model_name == "chatglm2-6b":
    print("******* loading chatglm2-6b")
    model_path = model_all_local_path + model_name+"-int4"
    model =  AutoModel.load_low_bit(model_path, trust_remote_code=True,
optimize_model=False)
    #model = model.half().to(device)
    model = model.to(device)
    tokenizer = AutoTokenizer.from_pretrained(model_path,
trust_remote_code=True)
elif model_name == "llama2-13b":
    print("******* loading llama2-13b")
    model_path = model_all_local_path + "llama-2-13b-chat-hf-int4"
    print(model_path)
    model = AutoModelForCausalLM.load_low_bit(model_path,
trust_remote_code=True, optimize_model=False)
    model = model.to(device)
    #model = model.half().to(device)
    tokenizer = AutoTokenizer.from_pretrained(model_path,
trust_remote_code=True)
elif model_name == "StarCoder":
    print("******* loading StarCoder")
    model_path = model_all_local_path + "starcoder-int4"
    model = AutoModelForCausalLM.load_low_bit(model_path,
trust_remote_code=True, optimize_model=False)
```

```
    model = model.to(device)
    #model = model.half().to(device)
    tokenizer = AutoTokenizer.from_pretrained(model_path,
trust_remote_code=True)
```

2.4 Large Language Model Benchmark on Arc dGPU

Test transformer INT4 LLM benchmark on Arc dGPU to get the best performance data.

Download benchmark_util.py from https://github.com/intel-analytics/BigDL/blob/05ffcda934d44bf6d3324fd65e0855087892d8ed/python/llm/dev/benchmark/benchmark_util.py, and run the following code:

```
import torch
import intel_extension_for_pytorch as ipex
from bigdl.llm.transformers import AutoModel
from transformers import AutoTokenizer
from benchmark_util import BenchmarkWrapper

model_path ='./chatglm2-6b-int4'
model =  AutoModel.load_low_bit(model_path, trust_remote_code=True,
optimize_model=False)
model = model.to('xpu')
model = BenchmarkWrapper(model, do_print=True)
tokenizer = AutoTokenizer.from_pretrained(model_path,
trust_remote_code=True)
prompt = " Once upon a time, there existed a little girl who liked to
have adventures. She wanted to go to places and meet new people, and
have fun"

with torch.inference_mode():
    # wamup two times as use ipex
    for i in range(2):
        input_ids = tokenizer.encode(prompt,
return_tensors="pt").to('xpu')
        output = model.generate(input_ids, do_sample=False,
max_new_tokens=32)
        output_str = tokenizer.decode(output[0],
skip_special_tokens=True)
    # collect performance data now
    for i in range(5):
        input_ids = tokenizer.encode(prompt,
return_tensors="pt").to('xpu')
        output = model.generate(input_ids, do_sample=False,
max_new_tokens=32)
        output_str = tokenizer.decode(output[0],
skip_special_tokens=True)
```

Then, get some outputs like:

```
=========First token cost 0.1896 s=========

=========Rest tokens cost average 0.0327 s (31 tokens in all)=========
```

This means that the first token takes 0.1896 seconds, and the rest tokens take an average of 0.0327 seconds.

For more information about benchmarking, please visit https://github.com/intel-analytics/BigDL/tree/05ffcda934d44bf6d3324fd65e0855087892d8ed/python/llm/dev/benchmark.

2.5 Large Language Model Stream Chat in Application

Use TextIteratorStreamer and Thread to get stream_chat function:

```
def stream_chat(model, tokenizer, prompt, input, max_new_tokens,
history=[], device="xpu"):
    input_ids = tokenizer([prompt], return_tensors='pt').to(device)

    streamer = TextIteratorStreamer(tokenizer,
                                    skip_prompt=True, # skip prompt in
the generated tokens
                                    skip_special_tokens=True)

    generate_kwargs = dict(
        input_ids,
        streamer=streamer,
        max_new_tokens=max_new_tokens
    )

    # to ensure non-blocking access to the generated text, the
generation process should be run in a separate thread
    from threading import Thread

    thread = Thread(target=model.generate, kwargs=generate_kwargs)
    thread.start()
    history = []

    output_str = ""
    for stream_output in streamer:
        output_str += stream_output
        yield output_str, history
```

For ChatGLM2, LLaMa2, and StarCoder model, the parameters are:

- Temperature: The higher the value, the more random the output; the adjustable range is 0~1.
- Top P: The higher its value, the greater the diversity of word choices, adjustable range 0~1.
- max_tokens: The maximum tokens for the output text, the adjustable range is 1~2048. The upper limit is determined by the model.

Using ChatGLM2-6B model as an example to achieve large language model stream output:

```
from bigdl.llm.transformers import AutoModel
from transformers import AutoTokenizer
import torch
model_name = "chatglm2-6b"
model_all_local_path = "./"
model_path = model_all_local_path + model_name
prompt = "What is AI?"
model =  AutoModel.load_low_bit(model_path, trust_remote_code=True,
optimize_model=False)
model = model.to(device)
tokenizer = AutoTokenizer.from_pretrained(model_path,
trust_remote_code=True)
model = model.eval()
timeFirst = 0
timeFirstRecord = False
torch.xpu.synchronize()
timeStart = time.time()
if model_name == "chatglm2-6b":
    template = "问：{prompt}\n\n 答："
    prompt = template.format(prompt=input)
    with torch.inference_mode():
        for response, history in stream_chat(model, tokenizer, prompt,
input, max_new_tokens=max_length):
            if timeFirstRecord == False:
                torch.xpu.synchronize()
                timeFirst = time.time() - timeStart
                timeFirstRecord = True
            yield chatbot, history,  "", ""

torch.xpu.synchronize()
timeCost = time.time() - timeStart

token_count_input = len(tokenizer.tokenize(prompt))
token_count_output = len(tokenizer.tokenize(response))
ms_first_token = timeFirst * 1000
ms_after_token = (timeCost - timeFirst) / (token_count_output - 1+1e-8)
* 1000
print("input: ", prompt)
print("output: ", parse_text(response))
print("token count input: ", token_count_input)
print("token count output: ", token_count_output)
print("time cost(s): ", timeCost)
print("First token latency(ms): ", ms_first_token)
print("After token latency(ms/token)", ms_after_token)
```

2.6 Develop Application by Gradio Web UI

Put transformer INT4 format models in chatglm2-6b-int4, llama-2-13b-chat-hf-int4 and starcoder-int4 folder.

Run `LLM_demo_v1.1_arc.py` script on Terminal to open LLM Application UI as shown in Figure 1.

```
git clone https://github.com/violet17/LLM_Arc_dGPU.git

source /opt/intel/oneapi/setvars.sh

conda activate llm

export http_proxy=

export https_proxy=

python LLM_demo_v1.1_arc.py
```

Figure 1. LLM Application UI

- Large language model application has the following files:

  `LLM_demo_v1.1_arc.py`

  `theme3.json`

  `chatglm2-6b-int4` (***Note***: model folder)

  `llama-2-13b-chat-hf-int4` (***Note***: model folder)

  `starcoder-int4` (***Note***: model folder)

§

3.0 Conclusion

This document helps users to enable large language model application development on Arc dGPU for Ubuntu 22.04 by Gradio and Intel BigDL with large language model such as ChatGLM2-6B, LLaMa2-13B, StarCoder-15.5B and so on.

§

4.0 Reference Documents

Note: Third-party links are provided as a reference only. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Table 1. Reference Documents

Document	Document Location
LLM Application on Arc dGPU	https://github.com/violet17/LLM_Arc_dGPU
LLM Application on Windows	https://github.com/KiwiHana/LLM_UI_Windows_CPU
BigDL	https://github.com/intel-analytics/BigDL
BigDL Tutorial	https://github.com/intel-analytics/bigdl-llm-tutorial/tree/main

§