Optimizing and Running LLaMA2 on Intel® CPU

White Paper

October 2023

Intel Confidential

Authors:

Xiang Yang, Lim
Chen Han, Lim
Chun Sheong, Foong

Document number: 791610-1.0

Contents

Tables

Figures

Revision History

Date	Revision	Description
October 2023	1.0	Initial release

§

1.0 Introduction

Large Language Models (LLMs) are deep learning algorithms that have gained significant attention in recent years due to their impressive performance in natural language processing (NLP) tasks. However, deploying LLM applications in production has a few challenges ranging from hardware-specific limitations, software toolkits to support LLMs, and software optimization on specific hardware platforms. In this whitepaper, we demonstrate how you can perform hardware platform-specific optimization to improve the inference speed of your LLaMA2 LLM model on the llama.cpp (an open-source LLaMA model inference software) running on the Intel® CPU Platform.

Table 1. Acronyms

Term	Description
LLM	Large Language Model
NLP	Natural Language Processing
CPU	Central Processing Units

§

2.0 *Large Language Model on Intel® CPU*

2.1 Introduction

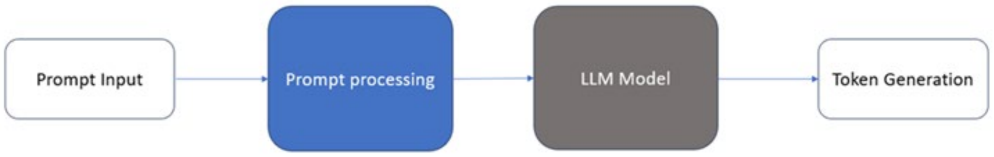
Deploying an LLM is usually bounded by hardware limitations as LLM models usually are computationally expensive and Random Access Memory (RAM) hungry. Taking an example of the recent LLaMA2 LLM model released by Meta Inc., the model size scales from 7 billion to 70 billion parameters. The RAM usage of LLaMA2 deployment can be seen in Table 2.

Table 2. The RAM Usage of the LLaMA2 Model with Different Model Sizes

Models	Model Size (Parameters)	RAM Requirements
LLaMA2	7B	~14GB
	13B	~26GB
	70B	~140GB

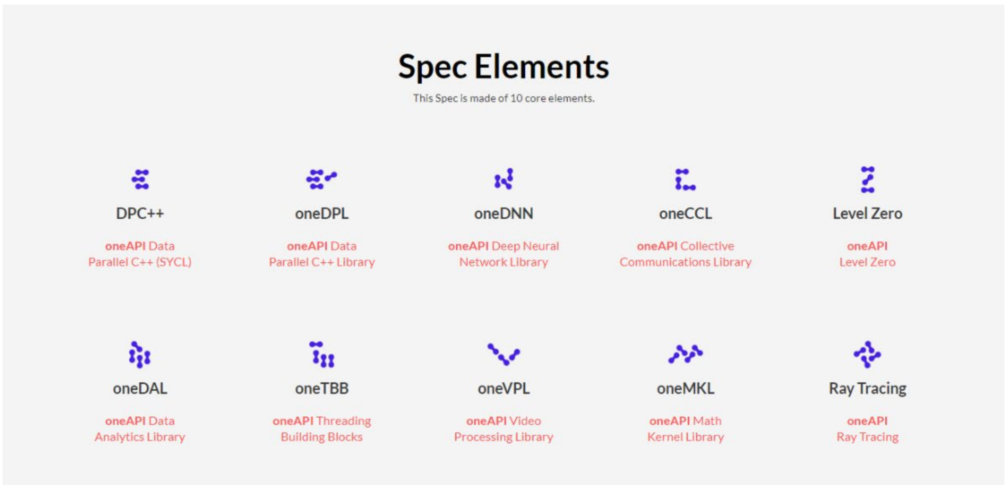
LLM inference pipelines can be broken down into several parts which are prompt input, prompt processing, model inference, and token generation as shown in Figure 1. Prompt input plays an important role in the token generation by the LLM model as it can engineer the model to generate a more concise and desired output. However, this also means that a comprehensive and long context is usually required to be fed into the model. A large chunk of prompt input will slow down the tokenization process and increase the overall inference time.

Figure 1. Overall LLM Inference Pipeline



For years, Intel has been working on software libraries that are heavily optimized on Intel® hardware. By enabling and adopting Intel software libraries, a great performance boost can be seen during software runtime. Intel® oneAPI is a specification that is open and standard-based, supporting multiple Intel® hardware architecture types, for example, CPU, GPU, and FPGA. This specification has both direct programming and API-based programming paradigms. Examples of the software libraries that are available in Intel® oneAPI toolkit can be found in Figure 2.

Figure 2. Software Libraries in Intel® oneAPI


Spec Elements
This Spec is made of 10 core elements.
DPC++
oneAPI Data Parallel C++ (SYCL)
oneDPL
oneAPI Data Parallel C++ Library
oneDNN
oneAPI Deep Neural Network Library
oneCCL
oneAPI Collective Communications Library
Level Zero
oneAPI Level Zero
oneDAL
oneAPI Data Analytics Library
oneTBB
oneAPI Threading Building Blocks
oneVPL
oneAPI Video Processing Library
oneMKL
oneAPI Math Kernel Library
Ray Tracing
oneAPI Ray Tracing

Besides, using an optimized CPU instructions set can accelerate the performance of the software. Intel AVX, AVX2, AVX_VNNI, AVX-512, and AVX-512_VNNI are some of the expansions of the Intel x86 instruction set that can help boost the performance of running deep learning applications on the CPU. Intel® AVX instructions set can help to improve certain integer and floating-point operations to allow the software to achieve a higher throughput.

In this whitepaper, we will show how to use the Intel® oneAPI toolkit to optimize and enhance the performance of the software. This software library provides an open, stable, and standard API for users to leverage common, parallelizable, mathematical operations in tensor operations for LLM computations. Intel® oneAPI DPC++/C++ compiler included in the Intel® oneAPI toolkit can perform additional optimizations for Intel® microprocessors. It uses the latest standards including C++ 20, SYCL, and OpenMP 5.0 and 5.1 for GPU offloading. Besides, we will also enable the Intel® AVX_VNNI CPU instruction set to further optimize the software execution. By enabling these software optimizations, it is expected to see a performance boost in the prompt processing for LLMs.

3.0 Hardware and Software Setup

3.1 Hardware Specifications

The hardware used in the whitepaper are shown in Table 3.

Table 3. Hardware Specifications

Hardware	Details
CPU	13th Gen Intel® Core™ i9-13900K
RAM	32GB RAM (Speed: 4800 MT/s)

3.2 Software Specifications

The software used in the whitepaper are shown in Table 4.

Table 4. Software Specifications

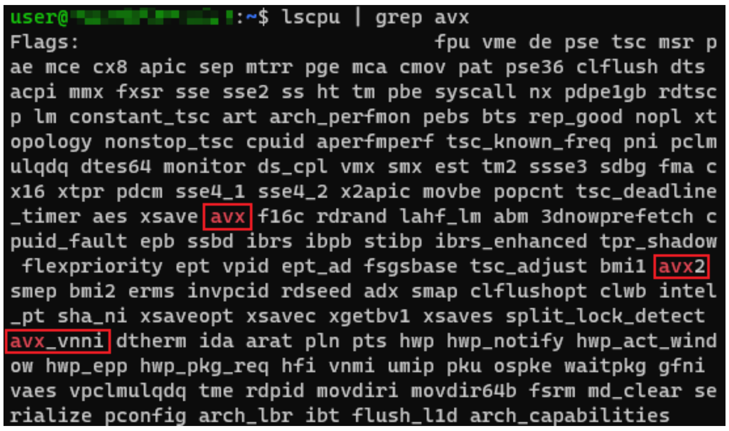
Software	Details
OS	Ubuntu 22.04.3 LTS
Kernel	6.2.0-26-generic
Intel® oneAPI	2023.2.0

3.3 Verification for CPU Instructions Sets

Run the following command to verify if the required CPU instruction sets are available as shown in Figure 3.

```
lscpu | grep avx
lscpu | grep avx2
lscpu | grep avx_vnni
```

Figure 3. Verification of CPU Instruction Sets are Available



```
user@[redacted]:~$ lscpu | grep avx
Flags:                            fpu vme de pse tsc msr p
ae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtsc
p lm constant_tsc art arch_perfmon pebs bts rep_good nopl xt
opology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclm
ulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma c
x16 xtpr pdcm sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline
_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch c
puid_fault epb ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow
 flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid rdseed adx smap clflushopt clwb intel
_pt sha_ni xsaveopt xsavec xgetbv1 xsaves split_lock_detect
avx_vnni dtherm ida arat pln pts hwp hwp_notify hwp_act_wind
ow hwp_epp hwp_pkg_req hfi vnmi umip pku ospke waitpkg gfni
vaes vpclmulqdq tme rdpid movdiri movdir64b fsrm md_clear se
rialize pconfig arch_lbr ibt flush_l1d arch_capabilities
```

3.4 Installation of Software Dependencies

1. Update the Ubuntu APT repository.

```
sudo apt update
```

2. Update the software packages.

```
sudo apt upgrade
```

3. Install the required software dependencies.

```
sudo apt install git cmake build-essential pkg-config python3-pip
python3-venv
```

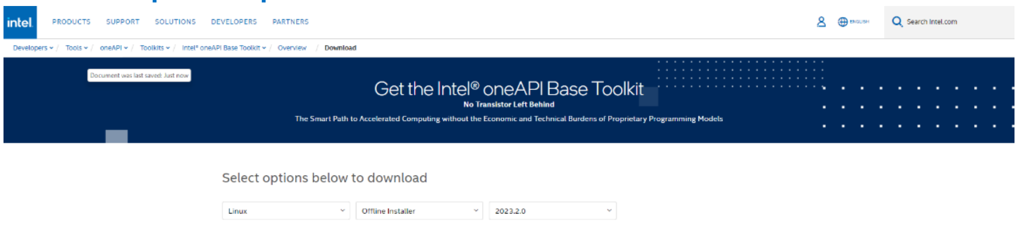
4. Create a workspace to store all the software changes.

```
mkdir -p ~/workspace
```

3.5 Installation of Intel® oneAPI Toolkit

1. Open a browser and browse the download link: https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit-download.html
2. Select the options below to start downloading the Intel® oneAPI Toolkit. In this whitepaper, we choose the following options.
   - Operating system: Linux
   - Distributions: Offline Installer
   - Version: 2023.2.0

Figure 4. Select the Specific Options to Download


intel
PRODUCTS
SUPPORT
SOLUTIONS
DEVELOPERS
PARTNERS
Search Intel.com
Developers / Tools / oneAPI / Toolkits / Intel® oneAPI Base Toolkit / Overview / Download
Document was last saved: Just now

Get the Intel® oneAPI Base Toolkit

No Transistor Left Behind

The Smart Path to Accelerated Computing without the Economic and Technical Burdens of Proprietary Programming Models

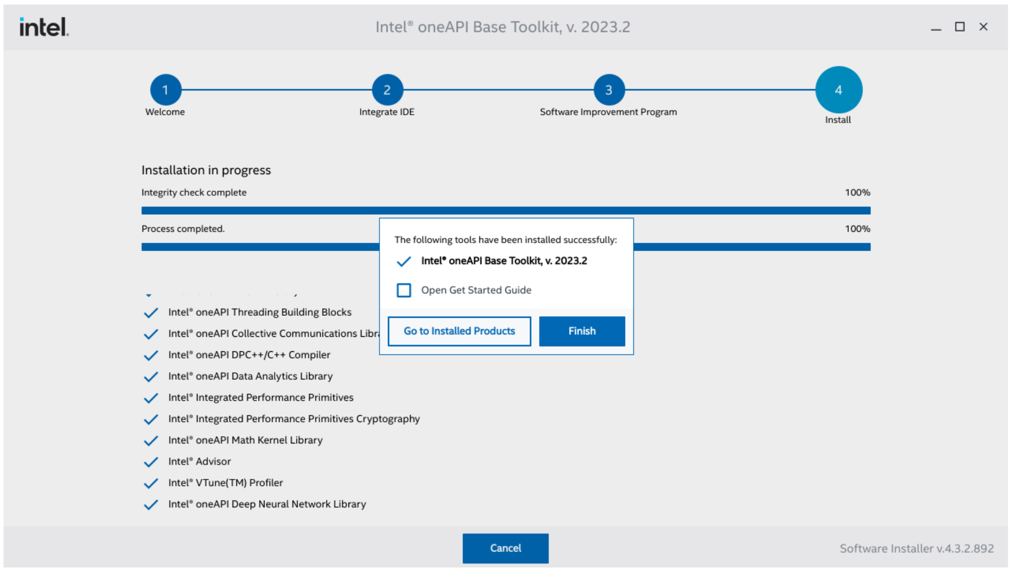
Select options below to download

Linux | Offline Installer | 2023.2.0

3. Go to the Command Line Download section and copy the command line for installations under Version 2023.2.0.

4. On the target system, run the copied command in the terminal to install the Intel® oneAPI base kit.

Figure 5. Intel® oneAPI Toolkit Installation


intel
Intel® oneAPI Base Toolkit, v. 2023.2
1 Welcome
2 Integrate IDE
3 Software Improvement Program
4 Install
Go to Installed Products
Finish
Cancel
Software Installer v.4.3.2.892

Installation in progress

Integrity check complete 100%

Process completed. 100%

The following tools have been installed successfully:

✓ Intel® oneAPI Base Toolkit, v. 2023.2

☐ Open Get Started Guide

✓ Intel® oneAPI Threading Building Blocks
✓ Intel® oneAPI Collective Communications Libra
✓ Intel® oneAPI DPC++/C++ Compiler
✓ Intel® oneAPI Data Analytics Library
✓ Intel® Integrated Performance Primitives
✓ Intel® Integrated Performance Primitives Cryptography
✓ Intel® oneAPI Math Kernel Library
✓ Intel® Advisor
✓ Intel® VTune(TM) Profiler
✓ Intel® oneAPI Deep Neural Network Library

3.6 Installation of llama.cpp

Llama.cpp is an open-source software project that can run the LLaMA model using 4-bit integer quantization. Intel® hardware can be built with some specific optimization tags to allow a faster prompt processing speed. The following steps shows how to build the llama.cpp software with Intel® software optimizations.

1. Clone llama.cpp from GitHub with tag b1309 in the workspace folder

```
cd ~/workspace
git clone https://github.com/ggerganov/llama.cpp.git -b b1309
cd llama.cpp
```

2. If the platform has AVX_VNNI instruction set support (verify by following the command in section Verification for CPU Instructions Sets), several modifications to the source can be done to enable this instruction set for faster speed in token generations.

3. In llama.cpp/ggml.c, add the following lines after line 21480.

```
int ggml_cpu_has_avx_vnni(void) {
#if defined(__AVXVNNI__)
    return 1;
#else
    return 0;
#endif
}
```

4. In llama.cpp/ggml.h, add the following line after line 2050.

```
GGML_API int ggml_cpu_has_avx_vnni   (void);
```

5. In llama.cpp/llama.cpp, add the following lines after line 7615.

```
s += "AVX_VNNI = "    + std::to_string(ggml_cpu_has_avx_vnni())
+ " | ";
```

6. In llama.cpp/common/common.cpp, add the following lines after line 1207.

```
fprintf(stream, "cpu_has_avx_vnni: %s\n", ggml_cpu_has_avx_vnni()
? "true" : "false");
```

7. In llama.cpp/spm-headers/ggml.h, add the following lines after 2050.

```
GGML_API int ggml_cpu_has_avx_vnni   (void);
```

8. After the modifications are done, run the command below to compile the llama.cpp software with Intel® oneAPI DPC++/C++ compiler.

```
cd ~/workspace/llama.cpp
rm -rf build && mkdir build
cd build
source /opt/intel/oneapi/setvars.sh
cmake .. -DLLAMA_BLAS=ON -DLLAMA_BLAS_VENDOR=Intel10_64lp -
DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx -DLLAMA_NATIVE=ON
cmake --build . --config Release
```

3.7 Prepare and Quantize the LLaMA2 Model

1. Get the LLaMA2 model weights and place them in the models folder.

2. Create a Python3 virtual environment.

```
cd ~/workspace/llama.cpp
python3 -m venv llama2
```

3. Source the Python3 environment and install the dependencies.

```
source llama2/bin/activate
python3 -m pip install -r requirements.txt
```

4. Download the LLaMA2 model weight from the official website and put it into the models/7B directory. Figure 6 shows the required files in the models/7B directory.

```
mkdir -p models/7B
```

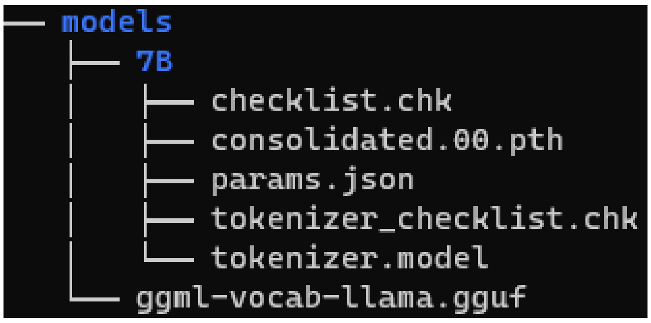
5. Convert the LLaMA2 7B model to FP16 format using the command below.

```
python3 convert.py models/7B/
```

6. Quantize the model to 4 bits using the command below.

```
./build/bin/quantize ./models/7B/ggml-model-f16.gguf
./models/7B/ggml-model-q4_K_S.gguf q4_K_S
```

Figure 6. LLaMA2 Model Weight Files



```
── models
   ├── 7B
   │   ├── checklist.chk
   │   ├── consolidated.00.pth
   │   ├── params.json
   │   ├── tokenizer_checklist.chk
   │   └── tokenizer.model
   └── ggml-vocab-llama.gguf
```

4.0 Software

4.1 Running the Program

1. Verify the number of threads available in the system using command below.

```
nproc --all
```

2. Follow the command below to run the LLaMA2 program. Replace the *number of threads* to the output in step 1.

```
cd ~/workspace/llama.cpp
./build/bin/main -m ./models/7B/ggml-model-q4_K_S.gguf -t
<number-of-threads> -b 512 -p "What is Intel OpenVINO"
```

3. Verified if the software is compiled with Intel® oneAPI and AVX_VNNI is available. You should see the following result in the terminal.

```
main: built with Intel(R) oneAPI DPC++/C++ Compiler 2023.2.0
(2023.2.0.20230622) for x86_64-unknown-linux-gnu
…
system_info: n_threads = 32 / 32 | AVX = 1 | AVX2 = 1 | AVX512 =
0 | AVX512_VBMI = 0 | AVX512_VNNI = 0 | AVX_VNNI = 1 | FMA = 1 |
NEON = 0 | ARM_FMA = 0 | F16C = 1 | FP16_VA = 0 | WASM_SIMD = 0 |
BLAS = 1 | SSE3 = 1 | SSSE3 = 1 | VSX = 0 |
sampling: repeat_last_n = 64, repeat_penalty = 1.100000,
presence_penalty = 0.000000, frequency_penalty = 0.000000, top_k
= 40, tfs_z = 1.000000, top_p = 0.950000, typical_p = 1.000000,
temp = 0.800000, mirostat = 0, mirostat_lr = 0.100000,
mirostat_ent = 5.000000
generate: n_ctx = 512, n_batch = 512, n_predict = -1, n_keep = 0
```

§

5.0 Conclusion

The whitepaper provided the steps to optimize and running LLaMA2 model using Intel® oneAPI toolkit and AVX_VNNI CPU instruction set.

§