CS6886 – ASSIGNMENT II

-Shreyas S (EP22B040)

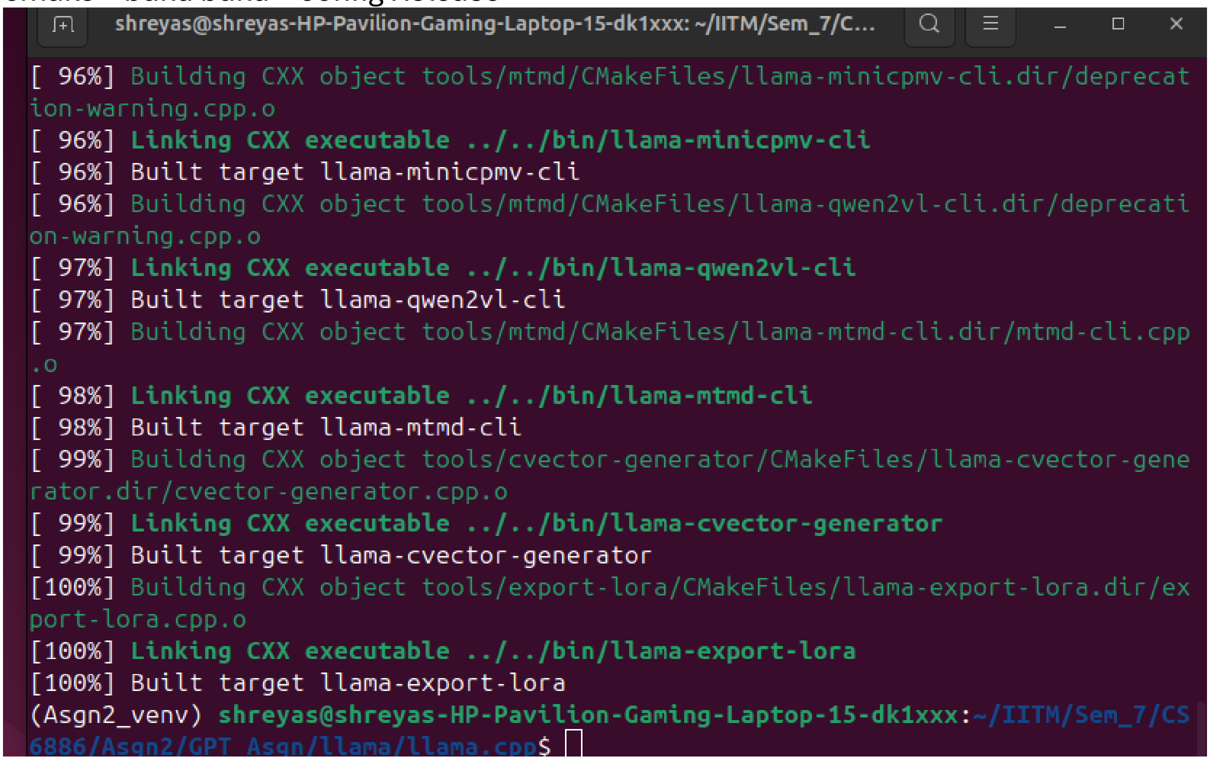
Task 1: Install llama.cpp from source

Steps to install & set-up llama:

Creating a conda environment and installing all the requirements within this conda environment

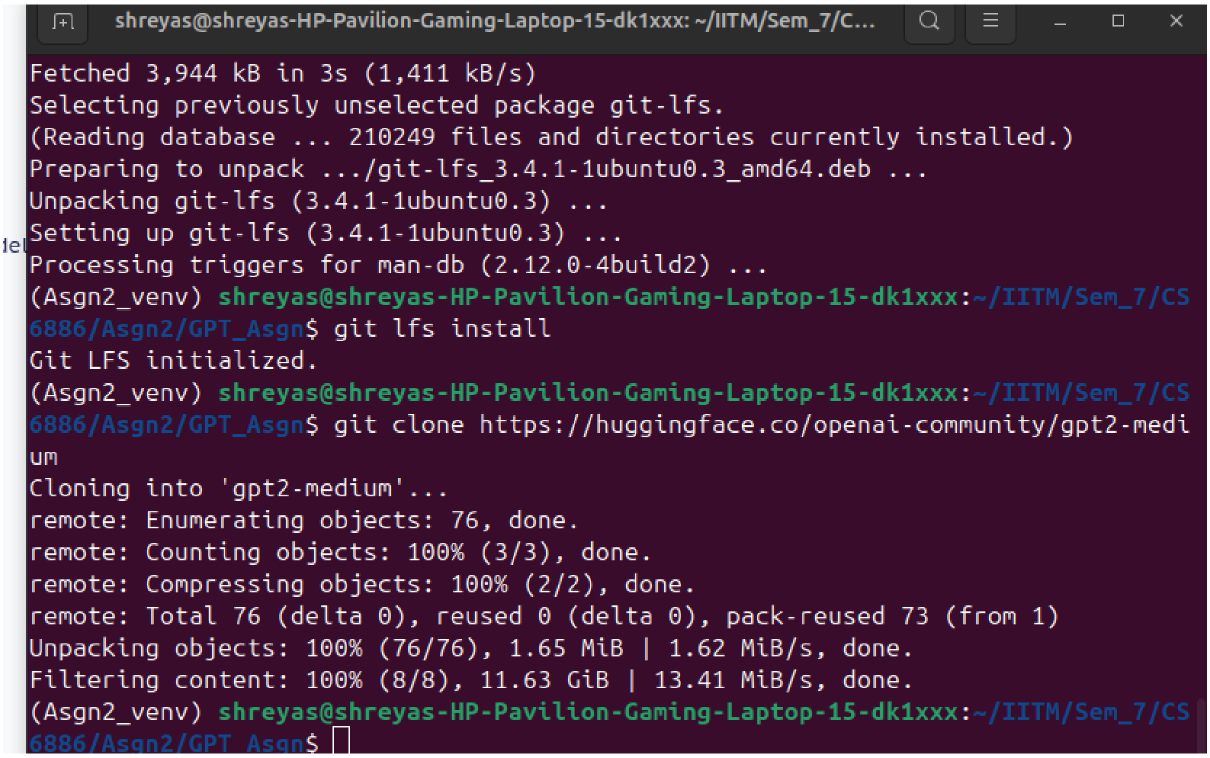
1. Clone llama from official git repo
2. Build llama.cpp using CMAKE
3. Commands:

git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
cmake -B build
cmake --build build --config Release



Task 2: Setting up the GPT model

Installed git-lfs to fully download the large model files of the Hugging Face GPT-2 model (11.63 GB)



```
shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx: ~/IITM/Sem_7/C...
Fetched 3,944 kB in 3s (1,411 kB/s)
Selecting previously unselected package git-lfs.
(Reading database ... 210249 files and directories currently installed.)
Preparing to unpack .../git-lfs_3.4.1-1ubuntu0.3_amd64.deb ...
Unpacking git-lfs (3.4.1-1ubuntu0.3) ...
Setting up git-lfs (3.4.1-1ubuntu0.3) ...
Processing triggers for man-db (2.12.0-4build2) ...
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS
6886/Asgn2/GPT_Asgn$ git lfs install
Git LFS initialized.
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS
6886/Asgn2/GPT_Asgn$ git clone https://huggingface.co/openai-community/gpt2-medi
um
Cloning into 'gpt2-medium'...
remote: Enumerating objects: 76, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 76 (delta 0), reused 0 (delta 0), pack-reused 73 (from 1)
Unpacking objects: 100% (76/76), 1.65 MiB | 1.62 MiB/s, done.
Filtering content: 100% (8/8), 11.63 GiB | 13.41 MiB/s, done.
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS
6886/Asgn2/GPT_Asgn$
```

Conversion to .gguf Model:

Conversion:

Model path:
/home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium
Conversion Script Path:
~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/convert_hf_to_gguf.py
Output File Path:
~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf

Sanity Benchmark:
Path to bench:
/home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin/llama-bench

Output:

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp$ ./build/bin/llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-

medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	CPU	4	tg256	14.90 ± 0.34

build: 05c0380f (6364)

```
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GP
T_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 16 -t 1
| model                          |       size |     params | backend    | threads |            test |                  t/s |
| ------------------------------ | ---------: | ---------: | ---------- | ------: | --------------: | -------------------: |
| gpt2 0.4B F16                  | 679.38 MiB |   354.82 M | CPU        |       1 |            tg16 |          0.54 ± 0.00 |

build: 05c0380f (6364)
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$
```

Task 3: Naive Execution

Built the *gpt2-medium* model with no forms of data-level, instruction-level or thread-level parallelism.

Result:

(Due to the absence of most of the optimizations, I have run the benchmark once for 16 tokens alone to verify the build before running it over 256 tokens.)

16 tokens:

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 16 -t 1

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	CPU	1	tg16	0.54 ± 0.00

build: 05c0380f (6364)

```
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GP
T_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 16 -t 1
| model                          |       size |     params | backend    | threads |            test |                  t/s |
| ------------------------------ | ---------: | ---------: | ---------- | ------: | --------------: | -------------------: |
| gpt2 0.4B F16                  | 679.38 MiB |   354.82 M | CPU        |       1 |            tg16 |          0.54 ± 0.00 |

build: 05c0380f (6364)
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$
```

256 tokens:

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	CPU	1	tg256	0.53 ± 0.00

build: 05c0380f (6364)

```
build: 05c0380f (6364)
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GP
T_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1
| model                          |       size |     params | backend    | threads |            test |                  t/s |
| ------------------------------ | ---------: | ---------: | ---------- | ------: | --------------: | -------------------: |
| gpt2 0.4B F16                  | 679.38 MiB |   354.82 M | CPU        |       1 |           tg256 |          0.53 ± 0.00 |

build: 05c0380f (6364)
```

Task 4: Default Execution

Building llama using the official build configuration given by default. This enables the SIMD/vectorized instructions (which is inferred while observing the counters for a later task).

Result:

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build_default/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	CPU	1	tg256	6.27 ± 0.05

build: 05c0380f (6364)

```
llama-diffusion-cli          llama-llava-cli       llama-q8dot          llama-tts                test-grammar-integration  test-regex-partial
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build_default/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/
Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1
| model                          |       size |     params | backend    | threads |            test |                  t/s |
| ------------------------------ | ---------: | ---------: | ---------- | ------: | --------------: | -------------------: |
| gpt2 0.4B F16                  | 679.38 MiB |   354.82 M | CPU        |       1 |           tg256 |          6.27 ± 0.05 |

build: 05c0380f (6364)
```

Task 5: "Near-Optimal Execution with mkl"

The mkl tools were installed from official webpage as linked. There were some errors while following the exact instructions given in the portal, however it was eventually fixed.

Result:

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	1	tg256	6.36 ± 0.12

build: 05c0380f (6364)

```
llama-cvector-generator      llama-imatrix         llama-perplexity     llama-tokenize           test-gguf                 test-quantize-stats
(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GP
T_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 0 -n 256 -t 1
| model                          |       size |     params | backend    | threads |            test |                  t/s |
| ------------------------------ | ---------: | ---------: | ---------- | ------: | --------------: | -------------------: |
        | gpt2 0.4B F16                  | 679.38 MiB |   354.82 M | BLAS       |       1 |           tg256 |          6.36 ± 0.12 |

build: 05c0380f (6364)
```

Observation:

- There is hardly any difference in run-time and average tokens/s reported for default and mkl cases.
- The expected significant boost-up was not observed.
- While probing further, when the benchmark was run with the flag -p 1, two processes ran. One of the two processes did show a significant speed up wrt tokens/s.
    - For the sake of consistency with the rest of the tasks, the flag was set to -p 0
- While enabling the verbose from onemkl toolkit, it was noticed that mkl was called initially but later the benchmark quite mkl and hence it finally ran the same setup as of default execution. Hence the results for all the mkl-based tasks are very similar to default-configuration tasks.

Results with -p 0 [mkl build]

(Asgn2_venv) shreyas@shreyas-HP-Pavilion-Gaming-Laptop-15-dk1xxx:~/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/llama/llama.cpp/build/bin$ ./llama-bench -m /home/shreyas/IITM/Sem_7/CS6886/Asgn2/GPT_Asgn/gpt2-medium.gguf/gpt2-medium-F16.gguf -p 1 -n 256 -t 1

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	1	pp1	15.79 ± 0.51
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	1	tg256	15.15 ± 0.04

build: 05c0380f (6364)

Task 6: Reporting performance counters

Floating Point Counters:

Counter	Description
fp_arith_inst_retired.128b_packed_double	Counts once for most SIMD 128-bit packed computational double precision floating-point instructions retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired
fp_arith_inst_retired.128b_packed_single	Counts once for most SIMD 128-bit packed computational single precision floating-point instruction retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired
fp_arith_inst_retired.256b_packed_double	Counts once for most SIMD 256-bit packed double computational precision floating-point instructions retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired.
fp_arith_inst_retired.256b_packed_single	Counts once for most SIMD 256-bit packed single computational precision floating-point instructions retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired.
fp_arith_inst_retired.scalar_single	Counts once for most SIMD scalar single computational precision floating-point instructions retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired.
fp_arith_inst_retired.scalar_double	Counts once for most SIMD scalar double computational precision floating-point instructions retired. Counts twice for DPP and FM(N)ADD/SUB instructions retired.

Memory Counters:

Counter	Description
uncore_imc/data_reads/	Counts memory controller data read requests from DRAM.
uncore_imc/data_writes/	Counts memory controller data write requests to DRAM.
uncore_imc/gt_requests/	Graphics/uncore requests serviced by the memory controller.
uncore_imc/io_requests/	I/O subsystem requests to the memory controller.

Note: While these uncore_imc/ instructions give an accurate picture of the memory access, while printing the perf stats for the benchmarks, these counters are declared as <not supported>. This is because these counters are present in the memory controller itself and the perf tool is not given access to the state of memory controller's counters.

Work-around:

Counter	Description
Cache-misses	Counts the number of cache misses in the L3 cache. Due to the hierarchy, all L3 cache misses are fetched from the DRAM itself. Therefore, this counter acts as an estimate to evaluate the memory access

- The exact number of bytes accessed is given by:

  Num. of bytes = Cache-misses × cache-line-size [64 bytes – in my computer]

Task 7: Performance Counters and Roofline Analysis

Obtaining the roofline for my PC:

Compute bound region:

The peak performance is obtained by the following estimate (based on specs from lscpu):

- CPU is intel i5-10300H (4 cores, 8 threads, base 2.5 GHz, turbo up to 4.5 GHz). This corresponds to (upto):
  - 8 single-precision operations per vector
  - FMA is available => 16 Flops per vector
  - Each core has 2 vectors => 32 Flops/cycle/vector
  - For peak turbo frequency of 4.5GHz:
    - $4.5 \times 10^9 \times 32 \sim 144$ GFlops/core
  - 4 cores available:
    - $4 \times 144 = 576$ GFlops peak

Memory bound region:

The ridge point is found by equating the operational intensity vs performance equation with the peak performance itself.

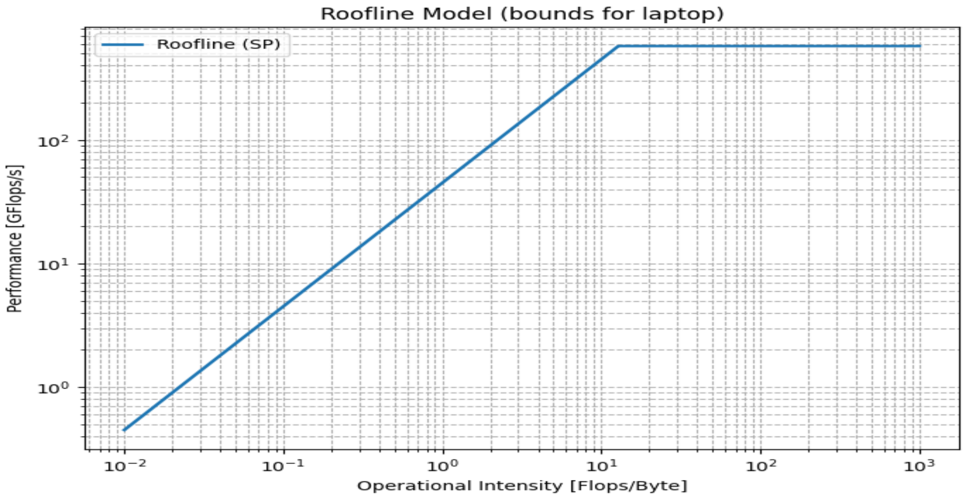
- Performance (Flops/s) = Bandwidth (Bytes/s) × Operational Intensity (Flops/byte)
- Bandwidth of my DDR4 (2933) RAM:
  - Transfer rate = 2933 MT/s (Mega-Transfers / s) [from online]
  - Bus width = 64 bits = 8 bytes
    - $2933 \times 10^6 \times 8 \sim 23.5$GB/s per channel
  - PC supports dual channel
    - Peak Bandwidth ~ 23.5 × 2 GB/s ~ 45 GB/s
  - Therefore, ridge-point:
    - Operational Intensity = performance[peak] / BW[peak]
    - **Operational Intensity[ridge] = 576 / 46 ~ 12.5 Flops/Byte**

Therefore any operational intensity,

<= 12.5 Flops/Byte is Memory-Bound

>= 12.5 Flops/Byte is Compute-Bound

Roofline for the PC:



Various performance counters collected for the 3 cases of Single-Thread Execution:

Execution	Build_Files	tokens/s	A	B	C	D	E	F	Cache_Misses	time_elapsed (s)
Naive	build_naive	0.51	463,594,208,427	461,988,872,999	585,425,071	1,932,729	1,933,319	1,931,605	23,706,699,353	2498.65575
Default_1	build_default	6.04 ± 0.01	683,393,173	139,464,356	1,369,767,901	172,168	117,125,741,086	171,593	16,954,671,743	212.7190231
mkl_1	build_mkl	6.00 ± 0.26	1,031,290,651	160,225,893	1,368,029,554	182,884	117,040,353,304	497,043	16,866,391,759	214.5796696

Where,

- A = FP_Single_Scalar
- B = FP_Double_Scalar
- C = FP_128b_Single_Packed
- D = FP_128b_Double_Packed
- E = FP_256b_Single_Packed
- F = FP_256b_Double_Packed

Derivation of Operational Intensity:

From the above defined notation,

A => 1 Floating point Operation

B => 2 Floating point Operations

C => (128/32) = 4 Floating point Operations

D => (128/64) = 2 Floating point Operations

E => (256/32) = 8 Floating point Operations

F => (256/64) = 4 Floating point Operations

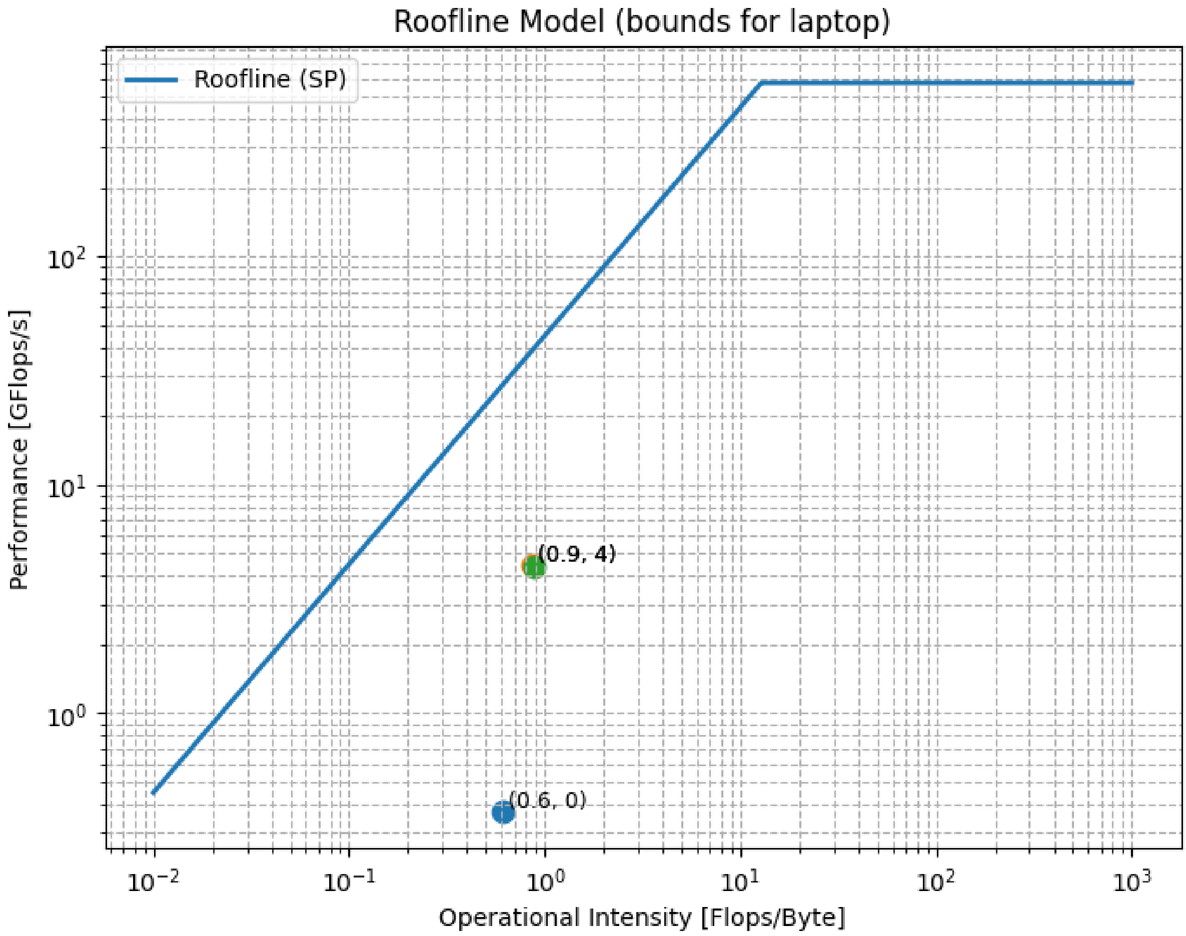
Therefore, for computing the average performance over run-time and average operational intensity:

- **Total Flops = (A + 2B + 4C + 2D + 8E + 4F)**
- Operational Intensity = Total Flops / Bytes Accessed
    - Bytes accessed (prev defined) = cache-misses * cache-line-size[64]
- Performance = Total Flops / Time Elapsed

Execution	time_elapsed (s)	FLOPs	GFlops	Bytes_Accessed	O.I. (Flops/byte)	Performance (GFlops/s)
Naive	2498.65575	927951840140	927.9518401	1517228758592	**0.6116097094**	**0.3713804274**
Default_1	212.7190231	943308888529	943.3088885	1085098991552	**0.8693297993**	**4.434529995**
mkl_1	214.5796696	942988815132	942.9888151	1079449072576	**0.8735834224**	**4.394586015**

- As expected, all three variants are deep into the memory-bound regions

Roofline Model along with the benchmarks:



Task 8: Thread Scaling

Benchmark Logs:

model	size	params	backend	threads	test	t/s
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	1	tg256	6.00 ± 0.26
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	2	tg256	9.82 ± 0.46
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	4	tg256	13.61 ± 0.10
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	8	tg256	7.32 ± 0.78
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	12	tg256	2.51 ± 0.10
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	16	tg256	1.86 ± 0.14
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	20	tg256	1.46 ± 0.22
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	24	tg256	1.21 ± 0.16
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	28	tg256	1.04 ± 0.05
gpt2 0.4B F16	679.38 MiB	354.82 M	BLAS	32	tg256	4.15 ± 0.21

Note: The performance, as measured by tokens/s, increases in the beginning upto 4 threads and starts declining again. Therefore, only diminishing results are obtained while not using specialized kernels for the hardware, inspite of the massively parallel nature of the computation. [mkl falls back to the default configuration]

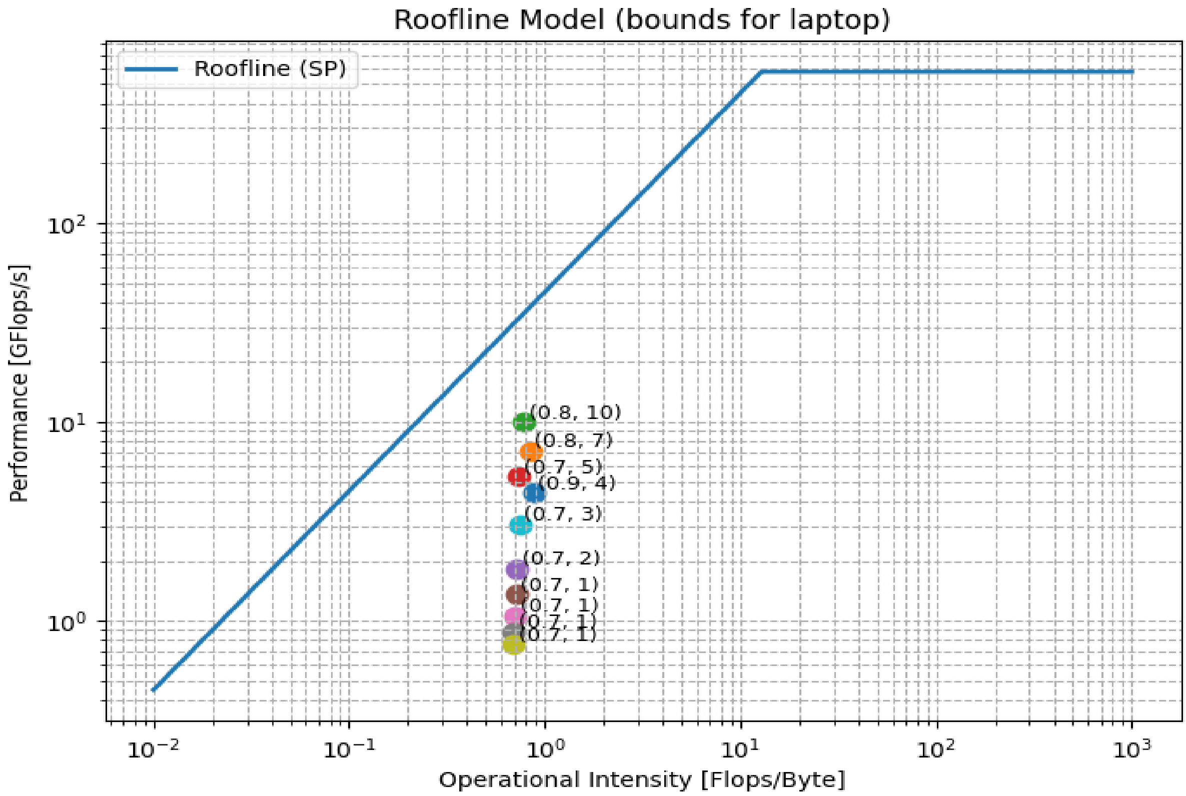
Various performance counters collected while thread scaling:

Execution	**Build_Files**	**tokens/s**	**A**	**B**	**C**	**D**	**E**	**F**	**Cache_Misses**	**time_elapsed (s)**
mkl_1	build_mkl	6.00 ± 0.26	1,031,290,651	160,225,893	1,368,029,554	182,884	117,040,353,304	497,043	16,866,391,759	214.5796696
mkl_2	build_mkl	9.82 ± 0.46	1,032,072,853	161,483,008	1,367,495,642	215,542	117,054,688,988	217,104	17,439,130,641	131.3303477
mkl_4	build_mkl	13.61 ± 0.10	1,033,913,223	162,652,469	1,368,048,096	293,956	117,023,542,591	314,897	18,770,087,553	94.86685788
mkl_8	build_mkl	7.32 ± 0.78	1,040,088,924	168,390,075	1,370,284,761	998,902	116,918,559,259	1,313,776	19,930,302,494	177.3067344
mkl_12	build_mkl	2.51 ± 0.10	1,092,044,751	169,622,660	1,369,943,108	764,307	116,793,723,715	993,078	20,313,564,802	512.2980299
mkl_16	build_mkl	1.86 ± 0.14	1,060,752,541	171,792,630	1,367,766,286	894,358	116,981,822,508	896,182	20,623,783,002	693.2560217
mkl_20	build_mkl	1.46 ± 0.22	1,164,313,986	175,508,667	1,370,023,485	1,051,752	117,238,474,254	1,366,061	20,906,382,451	892.9627395
mkl_24	build_mkl	1.21 ± 0.16	1,211,223,213	177,355,649	1,369,228,635	1,190,277	117,086,212,088	1,349,726	21,178,920,359	1075.289955
mkl_28	build_mkl	1.04 ± 0.05	1,203,402,444	180,889,418	1,368,625,135	1,288,615	116,985,135,400	1,291,162	21,308,735,780	1240.579994
mkl_32	build_mkl	4.15 ± 0.21	1,203,807,928	182,805,724	1,368,253,186	479,882	117,129,840,569	559,624	19,713,921,139	311.0446962

Operational Intensity while Thread Scaling:

Execution	time_elapsed (s)	FLOPs	GFLOPs	Bytes_Accessed	O. I. (Flops/byte)	Performance (GFlops/s)
mkl_1	214.5796696	942988815132	942.9888151	1079449072576	0.8735834224	4.394586015
mkl_2	131.3303477	943102349833	943.1023498	1116104361024	0.8449947718	7.181145609
mkl_4	94.86685788	942858946304	942.8589463	1201285603392	0.7848749237	9.938760146
mkl_8	177.3067344	942045345023	942.045345	1275539359616	0.7385466688	5.313082709
mkl_12	512.2980299	941096730489	941.0967305	1300068147328	0.7238826152	1.837010247
mkl_16	693.2560217	942563563823	942.5635638	1319922112128	0.7141054424	1.359618286
mkl_20	892.9627395	944735278373	944.7352784	1338008476864	0.7060757048	1.057978387
mkl_24	1075.289955	943562969564	943.5629696	1355450902976	0.6961247858	0.8774963118
mkl_28	1240.579994	942747617480	942.7476175	1363759089920	0.6912860376	0.7599248915
mkl_32	311.0446962	943901549208	943.9015492	1261690952896	0.7481242114	3.034617084

Roofline plot along with the thread scaling benchmarks



Discussion on Thread Scaling:

- Since the benchmarking is falling back to default execution, the benefits of thread scaling can be observed only until 4 threads.
- At a mere 8 threads itself, the performance drops again – this implies that the compiler is unable to parallelize the workload beyond 4 threads.
- The embarrassingly parallel operating potential is not really utilized inspite of the provision of separate hardware for the same.
- This could be improved by special kernels written to accelerate this particular task of functions like matrix multiplication involved in this inference benchmark.
- Even in the presence of such kernels, blindly increasing the number of parallel operations could backfire as seen here – the sequential overhead inherent in the feedforward networks should be accounted for. This also aligns well with the philosophy emphasized in the end of the assignment's manual aswell.

Additionally,
All the points continue to remain in the memory bound region!

This calls for using specialized inference accelerators as the general-purpose cores are NOT optimized for the memory bound cases.