Large Language Models on Qualcomm ® Adreno™ GPU

Siva Rama Krishna Reddy B
Senior Staff Engineer
Qualcomm India Private Limited

Agenda

- About Generative AI & Large Language Models
- Challenges on Edge devices
- Baseline reference
- Performance improvement
- Upcoming

About Generative AI & Large Language Models

- Discriminative AI
  - Focused to learn the boundaries
  - Classification, Object detection, Image quality enhancements.
- Generative AI
  - Focused to generate new data like the training data in a meaningful way.
  - Models learn the underlaying probability distribution of training data.
  - Capable of image synthesis, text generation, music and video generation.
  - Generative Adversarial Networks (GANs)
  - Variational Autoencoders (VAEs)
  - Transformers and diffusion models
- Large Language Models
  - Generative AI models trained with volumes of text data.
  - Capable of performing wide range of tasks
  - Answering, Article writing (code also) , language translators, chat bots and many more

Challenges on Edge device

- Exponentially increased params
  - 7B model has 12GB of parameters.
- DDR throughput impacts the overall performance.
  - Every token processing needs all the params and context data.
- Increased ALU utilization for prompt processing.
- Additional optimizations for new operations
  - Attention layers
  - Grouped SoftMax
  - Context or state maintenance across calls

Baseline Reference

About LLaMA from Meta

- LLaMa, A collection of Large Language Models from Meta.
- Collection of pre trained and fine-tuned models from GenAI, Meta.
- LLaMa (Feb 2023): https://arxiv.org/pdf/2302.13971.pdf
- LLaMa 2 (Jul 2023): *https://arxiv.org/pdf/2307.09288.pdf*
- Parameters ranging from 7B to 70B

- LLaMa-13B outperforms GPT-3 (175B) on most benchmarks
- LLaMa65B is competitive with the best models, Chinchilla-70B and PaLM-540B

Competitive Opensource LLM's

- MPT, Mosiac ML
  - https://www.mosaicml.com/blog/mpt-7b
  - 7B, 30B
- Falcon LLM, Abu Dhabi-UAE – The Technology Innovation Institute (TII)
  - https://falconllm.tii.ae
  - 1.3B, 7.5B, 40B, 180B
- Mistral 7B, Mistral Ai
  - https://docs.mistral.ai
  - 7B
- Pythia, Eleuther
  - https://www.eleuther.ai, https://github.com/EleutherAI/pythia
  - 70M to 12B
- Dolly, Databricks
  - https://github.com/databrickslabs/dolly
  - 3B, 6B, 7B, 12B

Competitive Private LLM's

- Palm and PaLM2:
  - Google
  - PaLM 2 is a state-of-the-art language model with improved multilingual, reasoning and coding capabilities.
  - 540B
  - https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html
- Chinchilla:
  - Company – DeepMind
  - 70B
- GPT-3
  - Company – OpenAI
- Claude2
  - Company – Anthropic
  - 100K context windows ~ 12x of GPT-4, ~ 24x of Llama2
  - Anthropic \ Claude 2

About MLC.ai

About MLC:

- Machine Learning Compilation. Designed to transform and optimize machine learning execution from "**Development Form**" to "**Deployment Form**"
- It minimizes integration and dependency, leverages hardware native acceleration and offers general optimizations
- This community offers
  - LLM: https://github.com/mlc-ai/mlc-llm
  - Web LLM: https://github.com/mlc-ai/web-llm
  - Web Stable Diffusion: https://github.com/mlc-ai/web-stable-diffusion

ML Development Gap

Ai Model landscape

NLP & Speech:
- Language Translators
- Speech to text
- LLM

Vision
- Primitive models
- Diffusion models
- Video generation

GAP

- Diversified
  - Hard wares
  - Environments
  - Target capabilities
- Need to tailor to achieve best results.
- Should be scalable for emerging workloads.

Deployment environment

Cloud:
- More processing power
- Large memory
- Perf is key here

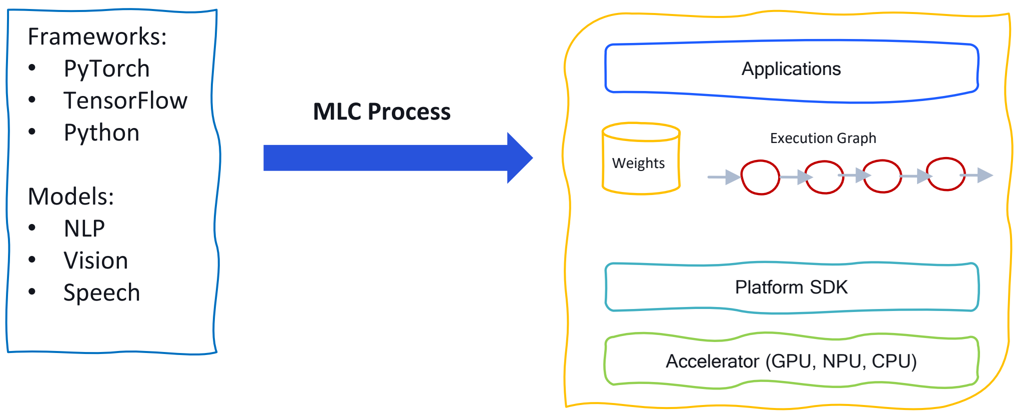
Mobile:
- Balanced processing power
- Limited memories
- Need balance between power and perf.

Edge
- Constrained environment
- Limited resources
- Perf/ watt is the key

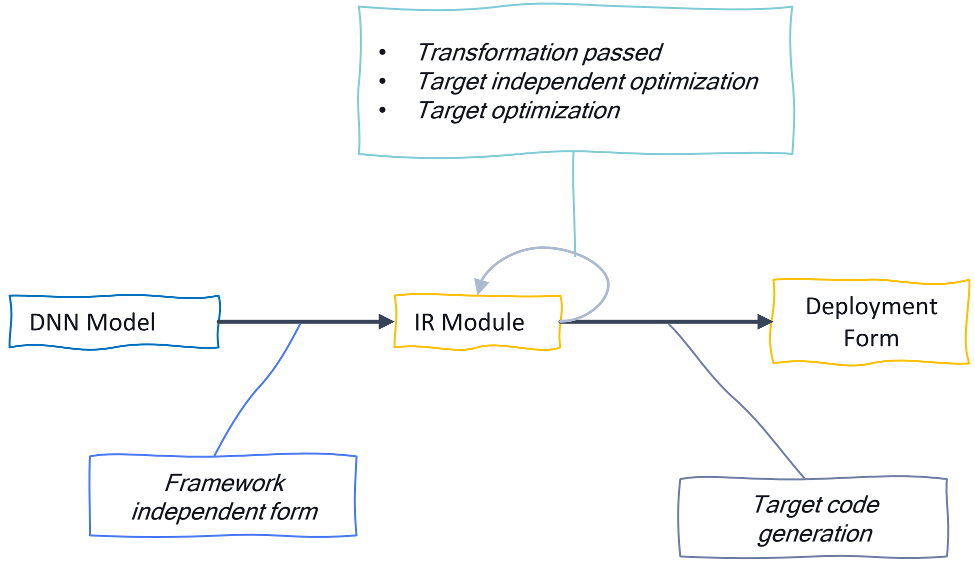
What is ML Compilation

Development Form

Android Deployment Form



MLC Workflow



Quantization with MLC

- LLaMa-7b has on disk parameters of nearly 12GB.
- MLC offer various quantization options to optimize the parameter storage.
- For LLaMA we use q4f16_0 quantization schema.
- We use group quantization here with group size of 32.

```
"q4f16_0": QuantizationScheme(
        name="q4f16_0",
        linear_weight=GroupQuantizationSpec(
            dtype="float16",
            mode="int4",
            sym=True,
            storage_nbit=32,
            group_size=32,
            transpose=True,
        ),
),
```

Quantization Schema for LLaMa

Param: 4096 x 12288 (FP16)
96 MB On disk

Group Quantization

Quantized Value: 512 x 12288 (UINT32)
24 MB On disk

Scale:128 x 12288 (FP16)
3 MB On Disk

96MB Parameter is represented as 27 MB (24 + 3) on disk
Quantization reduces over all params size from 12GB to nearly 3.5 GB

Param: 4096 x 12288 (FP16)

Grouped Param: 32 x 128 x 12288 (FP16)

Compute Scale per Group : 128 x 12288 (FP16)
Grouped Param: 32 x 128 x 12288 (FP16)

Scale : 128 x 12288 (FP16)
4bit conversion: 32 x 128 x 12288 (UINT32)

Scale : 128 x 12288 (FP16)
Group 4bit values:
8 x [4 x 128] x 12288 (UINT32)

Scale : 128 x 12288 (FP16)
Pack 8 x 4bit as unit32:
1 x 512 x 12288 (UINT32)

Scale : 128 x 12288 (FP16)
Quantized Param: 512 x 12288 (UINT32)

Quantization Algorithm

- MLC uses on the fly dequantization in lined into shader computation.
- Each group is represented by
    - One scale (FP16)
    - 32 4bit (4 UINT32) values.
- MLC's inner most loops treats each group as a single entity and performs computation to avoid redundant data access.

Runtime In lined dequantization

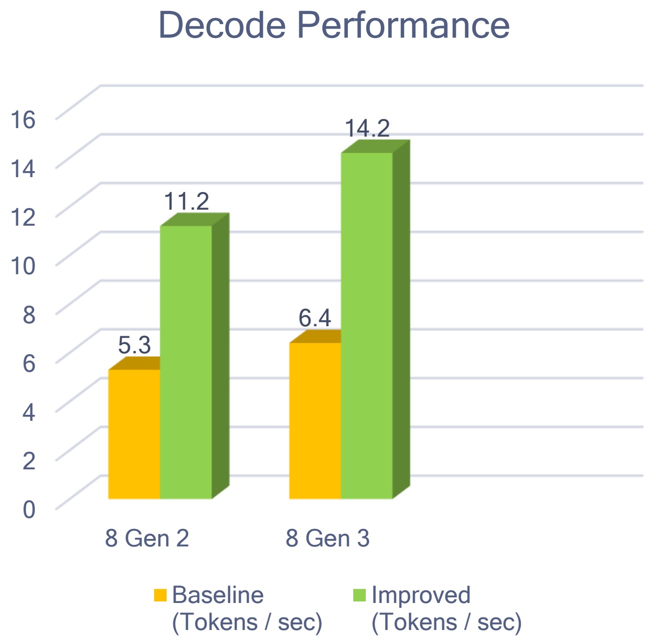
```
__kernel void fused_decode2_fused_NT_matmul2_silu_kernel(__global uint* restrict lv36, __global half* restric
t lv37, __global half* restrict lv45, __global half* restrict p_output0_intermediate, int n) {
  half8 var_NT_matmul_intermediate_pad_local[4];
  half8 lv37_local[1];
  uint8 lv36_local[1];
  half8 decode_local[1];
  half lv45_pad_local[4];
  for (int i0_i1_fused_1_2_init = 0; i0_i1_fused_1_2_init < 4; ++i0_i1_fused_1_2_init) {
    var_NT_matmul_intermediate_pad_local[i0_i1_fused_1_2_init] = ((half8)((half)0.000000e+00f, (half)0.000000
e+00f, (half)0.000000e+00f, (half)0.000000e+00f, (half)0.000000e+00f, (half)0.000000e+00f, (half)0.000000e+00
f, (half)0.000000e+00f));
  }
  for (int k_0 = 0; k_0 < 128; ++k_0) {
    lv37_local[0] = vload8(0, lv37 + (((k_0 * 11008) + ((convert_int(get_group_id(0))) * 128)) + ((convert_in
t(get_local_id(0))) * 8)));
    for (int k_1 = 0; k_1 < 4; ++k_1) {
      lv36_local[0] = vload8(0, lv36 + ((((k_0 * 44032) + (k_1 * 11008)) + ((convert_int(get_group_id(0))) *
128)) + ((convert_int(get_local_id(0))) * 8)));
      for (int k_2 = 0; k_2 < 8; ++k_2) {
        barrier(CLK_GLOBAL_MEM_FENCE);
        decode_local[0] = (((convert_half8(((lv36_local[0]  >>  ((uint8)(((convert_uint(k_2)) * (uint)4), ((c
onvert_uint(k_2)) * (uint)4), ((convert_uint(k_2)) * (uint)4), ((convert_uint(k_2)) * (uint)4), ((convert_uin
t(k_2)) * (uint)4), ((convert_uint(k_2)) * (uint)4), ((convert_uint(k_2)) * (uint)4), ((convert_uint(k_2)) *
(uint)4))))  &  ((uint8)((uint)15, (uint)15, (uint)15, (uint)15, (uint)15, (uint)15, (uint)15, (uint)15)))))
- ((half8)((half)7.000000e+00f, (half)7.000000e+00f, (half)7.000000e+00f, (half)7.000000e+00f, (half)7.000000
e+00f, (half)7.000000e+00f, (half)7.000000e+00f, (half)7.000000e+00f))) * lv37_local[0]);

        barrier(CLK_GLOBAL_MEM_FENCE);
```

- 8 x 4bit quantized values are received as uint32.
- Dequantization here works vectors of size 8
    - 8 * Shift and Mask
    - 8 * Convert from int4 to FP16
    - 8 * FP16 Add
- Resulting in **3 instructions** per each component
- On the fly Dequantization is not impacting performance.

Decode

LLAMA-2 / Decode	No Of Times Invoked	Snapdragon 8 Gen 2 Baseline (us)	Snapdragon 8 Gen 2 Improved (us)	Snapdragon 8 Gen 3 Baseline (us)	Snapdragon 8 Gen 3 Improved (us)
divide_kernel	1	13	13	8	8
full_kernel	1	9	9	5	5
fused_fused_decode10_matmul8_kernel	32	73444	30667	54633	26322
fused_fused_decode11_fused_matmul9_add1_kernel	32	39029	14503	28928	13228
fused_fused_decode1_take1_kernel	1	11	11	6	6
fused_fused_decode7_fused_matmul5_cast2_kernel	1	1701	1514	1355	1214
fused_fused_decode8_matmul6_kernel	32	40871	17123	30399	14790
fused_fused_decode9_fused_matmul7_add1_kernel	32	12143	5684	9248	5040
fused_NT_matmul1_divide2_maximum1_minimum1_cast3_kernel	32	6178	5028	3312	699
fused_softmax2_cast4_kernel	32	585	589	352	227
fused_split3_silu1_multiply1_kernel	32	375	366	191	193
NT_matmul3_kernel	32	2131	1849	1737	537
rms_norm1_kernel	65	1377	1384	813	809
rotary_embedding1_kernel	64	644	645	343	337
softmax_kernel	1	171	173	110	110
split2_kernel_1	32	319	319	159	160
split2_kernel_2	32	319	320	161	161
split2_kernel	32	319	319	164	156
transpose4_kernel	64	9049	0	8453	0
Total	550	188688	80516	140377	64002



Decode	8 Gen 2	8 Gen 3
Baseline (Tokens / sec)	5.3	6.4
Improved (Tokens / sec)	11.2	14.2
% Improvements	2.1 x	2.2 x

Note: These performance numbers are measured on Community baseline on 20th July 2023. Later versions on community may have improved numbers.

Adreno Optimizations : Decode

Baseline TVM schedules

- Under utilizing the concurrency for large vector to matric multiplications.
- We have split the dot product followed by reduction into parallel threads and reduced cooperatively. Rewrote the schedules for intensive kernels.
- Per work group local memory usage also reduced to enable more concurrent work groups.

Kernel	Initial Global	Initial Local	Modified Global	Modified Local	% Improvement
fused_decode5_fused_matmul7_add1_kernel	2048:01:01	64:01:01	1024:08:01	32:08:01	49.67
fused_decode5_matmul7_kernel	2048:01:01	64:01:01	1024:08:01	32:08:01	50.77
fused_decode6_fused_matmul9_multiply1_kernel	2752:01:01	64:01:01	2752:02:01	64:02:01	12.62
fused_decode6_fused_matmul9_silu1_kernel	2752:01:01	64:01:01	2752:02:01	64:02:01	7.79
fused_decode7_fused_matmul10_add1_kernel	2048:01:01	256:01:01	1024:04:01	128:04:01	52.74

These early optimizations resulted in 35% uplift at decode block level

Network architecture

- Identified the significant time taken by Transpose ops across a MatMul
- Found an opportunity to remove the Transpose and Alter MatMul schedules to work on original layout
- Implemented two Transformation passed that identify this pattern and replace with op and corresponding schedule.
- These passes improved both encode and decode.

```
  mod = mlc_llm.transform.FuseDecodeTranspose()(mod)  # pylint: disable=not-callable
+ mod = mlc_llm.transform.FuseTranspose1Matmul()(mod)
+ mod = mlc_llm.transform.FuseTranspose2Matmul()(mod)
  mod = mlc_llm.transform.FuseTransposeMatmul()(mod)  # pylint: disable=not-callable
  mod = relax.pipeline.get_pipeline()(mod)  # pylint: disable=no-value-for-parameter
  mod = mlc_llm.transform.FuseDecodeMatmulEwise(  # pylint: disable=not-callable
```

Network optimizations resulted in 10 % uplift at decode block level in 8 Gen 3 and 8 Gen 2 devices.

Other minor but critical Improvements:

- Used precompiled bin loading for clKernels. Improved the load time from 450ms to 70ms.
- Texture (1D) promotion for arguments in selective kernels
- Temperature NDArray initialization was redundant at each token.
- Model context initialization prompt was optimized to reduce initial warmup time.
- LLaMa-v2 support is enhanced before community enabled v2 support.
- Support for Baichuan model (Chinese LLaMa variant).
- Schedule optimizations enabled for Baichuan model too.

Target Optimizations

- Operations in decode is mostly 1D-2D (vector to matrix) multiplications.
- By nature, these are memory bound (Vector is cached and matrix is never reused).
- Performance of decode here is directly proportional to memory bandwidth.

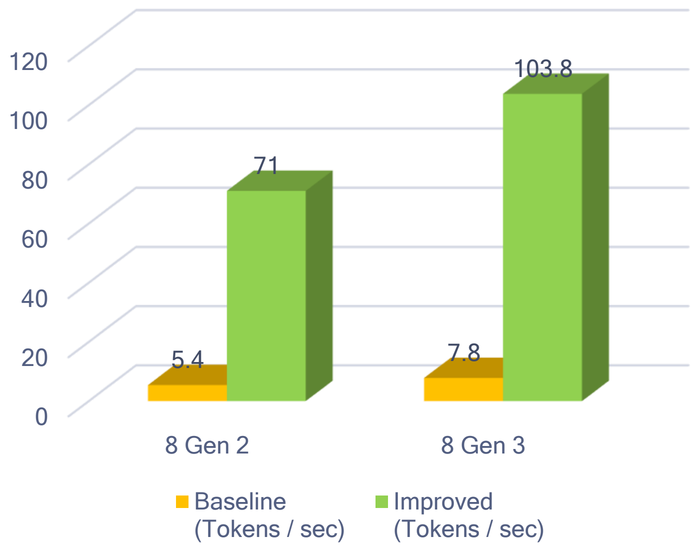
- In summary, we can tune the GPU frequency to get best efficiency (power-to-performance ratio).

Prefill / Encode

LLAMA-2 / Encode	No Of Times Invoked	Snapdragon 8 Gen 2 Baseline (us)	Snapdragon 8 Gen 2 Improved (us)	Snapdragon8 Gen 3 Baseline (us)	Snapdragon 8 Gen 3 Improved (us)
divide_kernel	1	13	13	8	8
extend_te_kernel	1	20	21	12	12
fused_fused_decode1_take_kernel	1	105	104	80	81
fused_fused_decode2_NT_matmul4_kernel	32	15550161	697488	10023110	461033
fused_fused_decode3_fused_NT_matmul5_add_kernel	32	691707	301002	459036	209973
fused_fused_decode4_NT_matmul6_kernel	32	27933885	1172984	18000559	735612
fused_fused_decode5_fused_NT_matmul7_add_kernel	32	1745141	537726	1150687	381043
fused_fused_decode7_fused_matmul5_cast2_kernel	1	1704	1517	1308	1302
fused_min_max_triu_te_broadcast_to_kernel	1	16	16	8	8
fused_NT_matmul_divide1_maximum_minimum_cast_kernel	32	344268	352917	222482	228584
fused_softmax1_cast1_kernel	32	16765	16733	11344	11394
fused_split1_silu_multiply_kernel	32	10242	9862	9427	8620
NT_matmul2_kernel	32	344371	334431	223535	216529
rms_norm_kernel	65	20569	20663	14519	14668
rotary_embedding_kernel	64	6787	6770	5063	5018
slice_kernel	1	11	10	5	5
softmax_kernel	1	171	172	115	108
split_kernel_1	32	3198	3201	2120	2018
split_kernel_2	32	3200	3200	2184	2008
split_kernel	96	3195	3203	6733	2048
transpose4_kernel	96	13094	0	10202	0
transpose7_kernel	32	3206	0	2266	0
Total	680	46691829	3462033	30144803	2280072

Encode	Gen 2	Gen 3
Baseline (Tokens / sec)	5.4	7.8
Improved (Tokens / sec)	71	103.8
% Improvements	13.14 x	13.3 x

Encode / Prefill Performance



Optimizations

- Prompt (encode) ops are mostly nD-2D (batch matrix to matrix) multiplications.
- By nature, these are ALU bound operations.
- Optimizations are driven by hand crafted OpenCL kernels to draw best performance possible.
- Integrated as BYOC (Bring Your Own Codegen) to let the optimized kernels to co work with TVM native kernels in over all solution.

Note: These performance numbers are measured on Community baseline on 20th July 2023. Later versions on community may have improved numbers.

Upcoming

Open source

- Today MLC supports wide range of LLM’s.
- Focused to improve the performance.
- Committed to upstream or share with communities.
- Optimized Mistral-7B, Qwen-7B, Gemma are on the way.

Thank you

Follow us on:

For more information, visit us at:

qualcomm.com & qualcomm.com/blog