Design Doc: Home exam 1

Codec63 Encoder on CUDA — IN5050

{liseej, rolfvh}@uio.no

2025-02-06

Codec63 is a variant of MJPEG designed to explore advanced video encoding techniques such as motion prediction and compensation (UiO, 2025), adapted for CUDA to enhance processing on Nvidia GPUs.

1 PCAM

To systematically optimize and parallelize tasks, we mainly apply the PCAM model (Foster 1995), which facilitates a structured analysis and decision-making in parallel computing environments.

1.1 Partitioning

Partitioning in the context of CUDA involves identifying computational tasks that can be executed in parallel while recognizing the data associated with each task. This step is crucial for optimizing performance by ensuring that each GPU thread has a meaningful workload. Here we detail all possible partitionable units within the Codec63 encoding process:

- **Macroblocks**: Basic units of video processing, each frame is divided into 8x8 blocks of pixels processed independently. Further split into either rows or columns and their underlying YUV samples.
- **Frames**: Classified into different types based on their role:
  - **I-frames**: Fully encoded frames that serve as reference points and do not rely on other frames.
  - **P-frames**: Predicted frames that use data from previous frames to encode only the differences.
- **Motion Vectors**, **Residuals**: Intermediary matrices used to describe the difference between two frames in time in a compressed manner.
- **Predicted Frame**, and **Reconstructed Frame**: Refers to different transformations from a given reference frame.
- **DCT (Discrete Cosine Transform)** and **Quantization**: Processes applied independently to each macroblock, facilitating compression.
- **Entropy Coding**: Compresses the quantized data to a smaller bitsize at the frame level. Applied separately to the DCT and Quantization since it's not used during Motion Compensation.

On top of the points listed above, which are crucial for understanding how the codec works, we also need to consider how *hot* each piece of data is, but we choose to omit it here since it's more relevant to mapping.

1.2 Communication

The communication phase in the PCAM process identifies the explicit data dependencies between the partitioned units listed above. This analysis ensures that the implementation considers any interactions without introducing any synchronization or consistency issues.

- **Frame Dependencies**:

  - **I-frames (Intra-coded frames)**: These are independent frames that do not rely on other frames for their data. They are self-contained.
  - **P-frames (Predictive frames)**: Depend on previous frames (either I-frames or preceding P-frames) for reference data to compute motion vectors and residuals.

- **Macroblock Processing**:
  - Each macroblock within a frame can be processed independently in terms of motion estimation but requires the reconstructed data from previous frames' macroblocks to compute motion vectors.
  - Residuals for each macroblock are dependent on the original data of the macroblock and the predicted data derived from motion compensation.

- **Motion Vectors and Residuals**:
  - Motion vectors for each macroblock rely on the search results from the motion estimation process which compares the current macroblock against corresponding blocks in the reference frame.
  - Residual calculation for each macroblock depends on the output from the motion compensation which uses the computed motion vectors.

- **Predicted Frames**:
  - The construction of a predicted frame is dependent on the availability of motion vectors calculated from the reference frames.
  - Each predicted frame serves as a basis for the next P-frame's motion estimation, linking each frame in a dependency chain.

- **Data encoding**:
  - The *Discrete Cosine Transform* (DCT) of a macroblock is independent in its calculation but requires the actual pixel data from the macroblock itself.
  - *Quantization* (of the DCT coefficients) depends on the results from the DCT process and follows directly.
  - While the *entropy coding* process can be started independently for different macroblocks, all the macroblocks we're compressing must have finished their quantization.

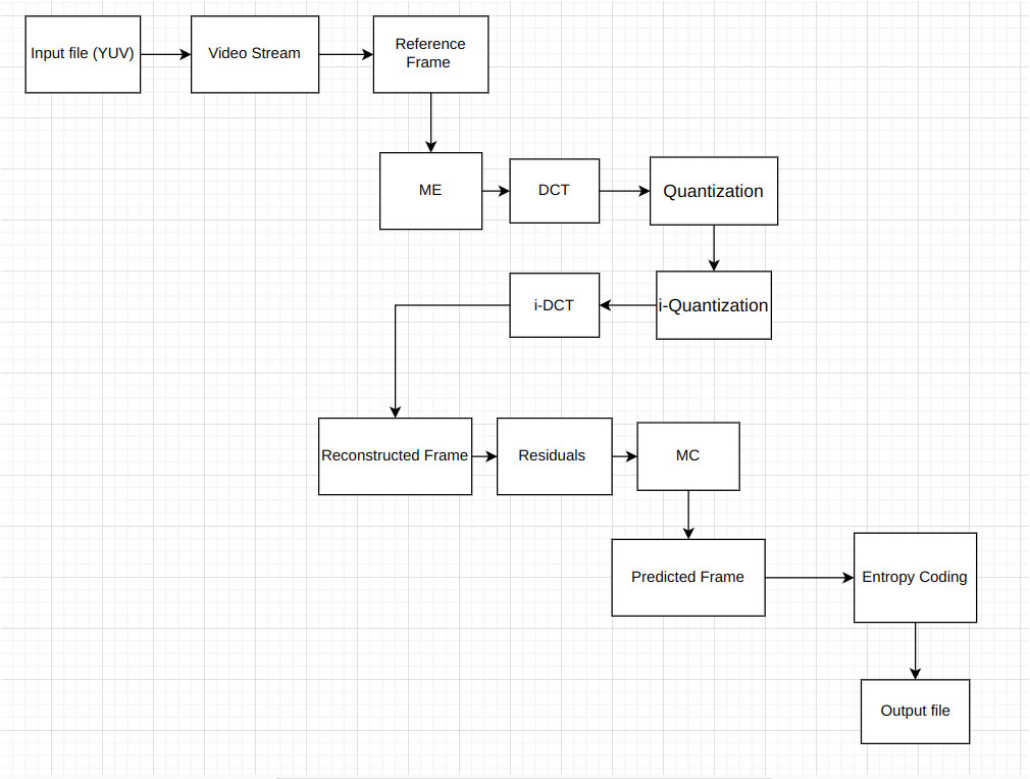

Figure 1: Data flow diagram showing the relationship between frames

1.3 Agglomeration

Given the identified dependencies among tasks in the Codec63 encoding process, we can strategically combine certain tasks to optimize resource utilization and enhance execution efficiency. The main benefit of grouping tasks together is to leverage the locality of data, improving the throughput, and lowering the latency between data transfers. Knowing what data can be grouped together also allows us to know what data can be run concurrently without synchronization.

- **DCT and Quantization**: Since the output of the Discrete Cosine Transform (DCT) is immediately used by the quantization step, these tasks can be agglomerated into a single operation. Additionally, the intermediate step here is not ever used anywhere else.
- **Macroblock Processing**: Entire macroblock processing, from motion estimation through to entropy coding, can be viewed as a pipeline of dependent tasks. Is there some opportunity for caching between frames through this pipeline?
- **Motion Estimation and Compensation**: Motion estimation and motion compensation are deeply interconnected, with compensation directly using the motion vectors calculated by estimation. Each macroblock calculation is independent to other macroblocks, allowing for calculating them concurrently.
- **I-frames and reference frames**: Since P-frames always relies on using its previous frame as a reference frame, and are reset on I-frames, this is a natural group of tasks requiring a sequential order.
- **Entropy Coding Synchronization**: While entropy coding can start as soon as a macroblock is quantized, synchronizing the start of entropy coding across multiple macroblocks may optimize the utilization of the encoder's resources. Would waiting until an entire batch of macroblocks is ready make better use of the GPU?

This focused approach to agglomeration, leveraging the natural dependencies within the Codec63 encoding tasks, aims to streamline operations and enhance overall performance by reducing unnecessary operations and focusing on task

1.4 Mapping

Mapping tasks onto CUDA cores involves tuning where and how to assign jobs to specific threads and blocks within the GPU architecture. This stage requires us to understand the memory architecture our GPU, where the goal is to eliminate unnecessary overhead, minimize bottlenecks and avoid synchronization waiting.

- **Memory Locality**:
  - *Global Memory Usage*: The global memory is the slowest, but is the only one the GPU can use to communicate with the host. Is there a chance we could write read-only image frames to the constant memory or texture memory?
  - *Shared Memory Utilization*: Shared memory exists per block (core). We should keep data which needs to be reused here. Which data, if any, would that be?
- **Parallelization of I-frame Processing**: Process I-frames (and their sequential P-frames) independently to maximize throughput, as they do not depend on other frames. This allows for continuous, uninterrupted processing of I-frames across different CUDA cores.
- **Concurrent Processing of Macroblocks**: Each macroblock's processing pipeline from motion estimation (ME) to motion compensation (MC) to quantization can be executed independently of others. We can schedule these pipelines across different blocks and streams to achieve high parallelism and reduce waiting times between stages.
- **Symmetric GPU Architecture**: Since our GPU is symmetric (no difference in instruction sets between cores), we don't need to consider which types of tasks should be assigned to which cores. That said, does running similar instructions across all cores play a role in how effective the hardware can prefetch instructions?

Effective mapping not only considers the capabilities and limitations of the GPU hardware but also strategically utilizes its features to optimize data processing and flow. Should we prioritize running macroblocks concurrently, or focus on running many I-frame sets simultaneously? A preemptive guess here is that we should avoid running more macroblocks concurrently than would fit in the shared memory.

2 Workflow

2.1 Testing and Validation

Efficiency and accuracy are critical to our project. We aim to ensure that the video output has no degradation compared to the original, while also working to enhance encoding efficiency. The efficiency measurement is the sum of encoding and decoding of a sample raw YUV video.

- **Video Quality Assurance**: Involves checking for fidelity, color accuracy, and frame integrity to ensure that the encoding process does not introduce noticeable artifacts or errors. Can we expect the output file to be byte-identical?
- **Performance Profiling**: Using profiling tools to measure the performance of different encoding strategies. For this, we will mainly employ Nsight Systems, which supports our lab's Nvidia Quadro card (unlike the better maintained Nsight Compute software), to monitor and analyze GPU resource usage, identify bottlenecks, and evaluate memory efficiency.

Initial profiling has shown that approximately 70% of the compute time is consumed by the `sad_block_8x8()` function. This indicates a significant area for potential optimization during motion estimation.

2.2 Implementation Roadmap

To effectively improve our codec's performance, the initial step involves transitioning parts of the code from the CPU to the GPU.

It seems like the natural starting point for code migration is the mentioned `sad_block_8x8()` function. From here, we can also look at the following optimizations:

- **Optimizing Data Batches**: Segmenting batches of I-frames and their associated P-frames. By processing these frames in intelligently grouped chunks, we may be able to reduce latency and thus increase throughput.
- **Hardware Utilization**: Fine-tuning the usage of GPU cores and memory resources. This includes adjusting the core counts involved in both of the steps above to enhance overall system performance.

References

Foster, Ian. 1995. "Designing and Building Parallel Programs." https://www.mcs.anl.gov/~itf/dbpp/.

in5050@ifi.uio.no. 2025. *Home Exam 1: Video Encoding on nVIDIA GPUs using CUDA*. University of Oslo. https://www.uio.no/studier/emner/matnat/ifi/IN5050/v25/in5050---home-exam-1.html.