Notes on CHORD F→X packet format

kmsmith

October 31, 2023

Introduction and fiducial data layout. The channelized electric field is a logical 5-d `int4` array, with axes:

$$(\text{time}, \text{freq}, \text{pol}, \text{dish}, \text{ReIm}) \tag{1}$$

Our GPU kernels currently assume that when the F-engine sends data to the X-engine, it uses a "fiducial" data layout defined by ordering the axes in Eq. (1) from slowest to fastest varying. This is an efficient data layout for the X-engine, in the sense that we do not need transpose kernels, either on the CPU or GPU.

When thinking about new F-engine designs, we may want to consider more complicated data layouts (especially if the corner turn is done in a switch instead of a backplane). The purpose of this note is to document which data layouts are optimal for the X-engine, and the X-engine performance penalty if the layout is suboptimal. These considerations can be weighed against FPGA and networking considerations, in a larger analysis.

GPU-friendly layouts. Suppose we divide each UDP packet into 128-byte chunks (the cache line size on an nvidia GPU). Each such 128-byte chunk will contain a subarray of the electric field. Let $(N_t, N_f, N_p, N_d, N_r)$ be the shape of this subarray (with axes from Eq. (1)). Note that:

$$N_t N_f N_p N_d N_r = 256 \tag{2}$$

since chunks are 128 bytes, and array elements are 4 bits.

We say that a data layout is *GPU-friendly* if the following conditions are satisfied:

$$N_f = N_p = 1 \qquad N_t \leq 16 \qquad N_d \leq 128 \tag{3}$$

In particular, the fiducial data layout above is GPU-friendly, with $(N_t, N_f, N_p, N_d, N_r) = (1, 1, 1, 128, 2)$.

If a data layout is GPU-friendly, then it should be possible to transfer the data to the GPUs, and process the data with the four main GPU kernels[1], without needing an extra memory read-write cycle to transpose the data. This is not obvious, but follows from detailed analysis of the memory access patterns of the GPU kernels. I've omitted the details in this short note, but please ask me about it if you're curious!

So, you should feel free to change the $F \to X$ packet format to any GPU-friendly layout (in the sense defined by Eq. (3)) without worrying about performance implications in the X-engine.

GPU-unfriendly layouts. Next we consider the question: if a data layout is not GPU-friendly, then what are the performance implications? I think it's hard to give a completely general answer to this question, but here is some general discussion.

If a data layout is GPU-unfriendly, then it needs to be transposed into a GPU-friendly layout. The transpose could be done either on the CPU or GPU, with tradeoffs as follows:

- A GPU transpose seems preferable, since it should use around 3% of GPU resources, assuming a 128×A40 X-engine. For a GPU transpose to work, the only constraint on the data layout (I think!) is that each 64-byte cache line should contain only one frequency.[2] If the data layout satisfies this constraint, then the CPU would just need to "fragment" UDP packets by writing different cache lines

[1] The four main GPU kernels are: (1) visibility matrix computation, (2) baseband beamformer, (3) FRB beamformer, (4) upchannelization.

[2] Note that CPU cache lines are 64 bytes. Previously near Eqs. (2)–(3), we were considering 128-byte GPU cache lines.

to different parts of DRAM. Further transposing of data between cache lines could be done on the GPU. GPU transpose kernels are straightforward to write, and I'm confident that even a worst-case transpose will be GPU memory bandwidth limited.

- If 64-byte cache lines contain multiple frequencies, then we'd need to do a CPU transpose. This would be a performance disaster if it requires a full read-write DRAM cycle. However, a full DRAM cycle might be avoided with a clever kernel which does its I/O while the full UDP packet is in L1 or L2 cache. This is more likely to work if the number of frequencies per 64-byte cache line is not too large. I'm not 100% confident it would work, so it's something that we'd definitely want to demonstrate, for a specific proposed UDP packet format, before buying hardware!

- In either of the above scenarios (GPU or CPU transpose), we'd probably be "locked in" to CPU-based packet processing, where each packet gets read into CPU cache, parsed, and streamed out to "assembled" arrays in DRAM. This would preclude some interesting alternative packet processing schemes, such as RDMA or parsing/indexing packets on the GPU. These alternative schemes could have higher performance, and free up X-engine output bandwidth to the science backends.