Proposal to Implement a DSP-based Co-Processor to Calculate Bunker Cache Addressing Parameters

Project Significance and Background

In data IO intensive workloads such as image processing and CNN, it is being increasingly pivotal to increase data throughput and enhance IO delay by increasing the cache locality. However, physical cache's performance is limited by its real-life physical constraints, and researchers have been exploring non-conventional methods to increasing cache locality. One of the methods to achieve better special locality is by Approximate Cache Access methods, exemplified by the "Bunker Cache" method in the course reading materials[1]. The main takeaway of the Bunker Cache paper was that in certain IO workloads with very high data redundancy, such as image processing, the cache controller can exploit the redundancy nature of the image data and map the vertical 'lines' of memory into the same cache address. In that case, whenever the processor wants to access the data 'line' of a cached data adjacent in 2D space, the cache access would become a hit and returns the same data that have just accessed before. This trade-off among data integrity and access speed / throughput have proven to be well-worth under certain workloads.

However, one of the drawbacks of the original Bunker Cache implementation was that the parameters of the cache address mapping functions was programmer or user defined. This causes certain amount of overhead on the programmer or compiler depending on the implementation details. I propose to implement a DSP based co-processor that work together with the Bunker Cache controller to automatically calculate the parameters with each IO on image data from disk to memory, at the same time seamlessly and synchronously updates the controller parameter registers to hide the implementation details of the Bunker Cache from the programmer.

For discussion purposes I make the following assumptions:

1. Assume all data that requires to be load by bunker cache are images.
2. Each image's data is stored in disk space linearly, consecutively, by row-order in a complete chunk, no data segmentation.
3. Each load of image data from disk to memory performs linearly and atomically. During the data IO on the image data, the data bus only serves to load data from that very image file.
4. All images are grey-scale (for the sake of proof-of-concept and simplicity)
5. Images with same dimension are accessed together. In other words, when accessing data from disk to memory, images with the same dimension are prioritized in IO scheduling. (this assumption make sense in real-life as many image-related workloads often have same dimensions)
6. Assume image have a reasonable minimum and maximum image size possible.
7. Assume image have a reasonable aspect ratio.

The proposed implementation of the co-processor is rather straightforward: With each IO operation from disk, the data bus feeds the data IO stream to both main memory as well as the Bunker Cache co-processor, the co-processor calculates a 1-D FFT upon the input data-stream in order to find some of the major discrete frequency. One of the major components of the frequency field is highly possibly to be the *stride* (or the width) of the image data. Therefore, the bunker cache controller can calculate the data mapping based on the calculated *stride* value.

To demonstrate that this method would yield expected result, I cherry-picked some image files and performs a 1D FFT on flattened, row-order on their data.

Fig 1, the cheery picked owl image

The owl image has a dimension of 510 * 340, after converting it to greyscale and flatten in column-major order, we can take a DFT of the image:

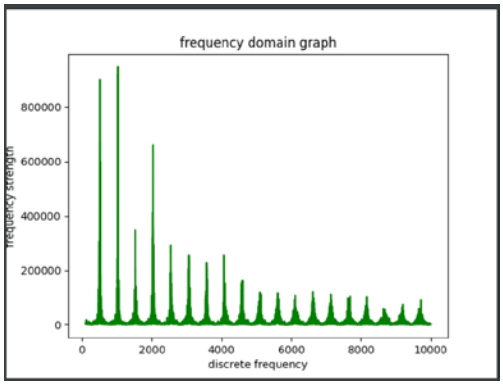

Fig 2, DFT of flattened image

If we zoom in the frequency near 510 (which is also the *stride* parameter)

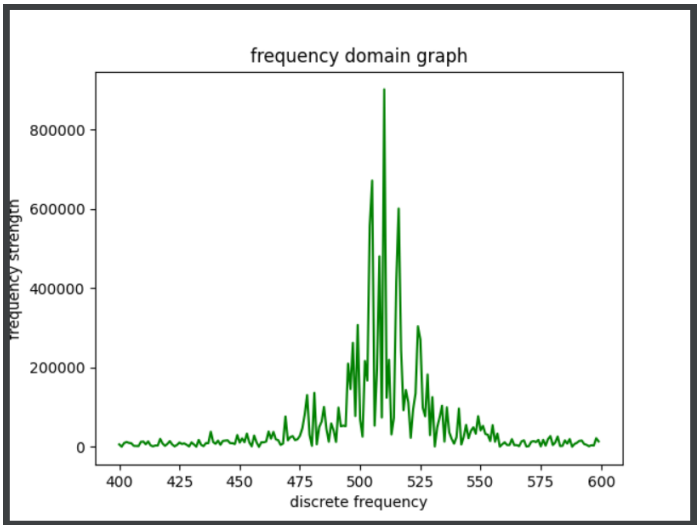

Fig 3, DFT near Fd around 500Hz

We can see a spike near 510, we can extract this peak index to update the bunker cache parameters.

Objectives

1. Implement a simplified, proof-of-concept version of the Bunker Cache on simulation level
2. Implement the DSP co-processor running FFT on IO data stream while updating the Bunker Cache parameters
3. Implement the verified version of the Co-processor on FPGA (hopefully)
4. Because the calculated parameter may have a slight difference than the user-defined one, run benchmark comparing the image process quality with pre-defined bunker cache parameters and automatic generated bunker cache parameters.

Reference:

[1] J. S. Miguel, J. Albericio, N. E. Jerger and A. Jaleel, "The Bunker Cache for spatio-value approximation," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 2016, pp. 1-12, doi: 10.1109/MICRO.2016.7783746.