Multicore & GPUG Programming, Spring 2018 Stephen Longofono

Summary of Message Passing

Sending Image Dimensions

Since I opted not to define the Packed3DArray type to OpenMPI, I needed to send the dimensions of the images to each node, using a tag to differentiate it from a data message. I didn't necessarily care to check that it got there, nor did I need that data anymore, so I used a simple call to MPI_Isend(), using a tag I specified for dimensions messages.

Sending Image Data

For all but the smallest images, holding image data in a buffer on the stack results in a stack overflow. I wrote all my code under the assumption that Packed3DArray was an opaque type, so when unpacking I put everything on the heap again. Due to the use of const in the internal representation of the Packed3DArray, it was prohibitively difficult to adapt my code to use this internal representation. I processed images for transit in a loop, so I used a blocking send to ensure that the data had begun to arrive at the receiver before freeing the buffer and moving on. With this approach, I was able to safely free the buffer immediately after the call to MPI_Send() returned.

Receiving Image dimensions

Each node needs to know its dimensions in order to declare an appropriately sized array to catch image data. There is nothing to until that information arrives. Thus I used a blocking receive call.

Receiving Image Data

Using an array of the appropriate size, I used a blocking call with the data message tag to catch image data. This needed to be blocking since there is no work to be done until the image data arrives.

Distributing Histograms

After each node (root included) had its histogram, I used a blocking, all-to-all gather to distribute the histograms. The normalized nature of the histograms made the dimensions of the histograms array well-defined, so there was no reason to communicate dimensions. Since I intended to use the data immediately thereafter, I used a blocking call. I chose not to use non-blocking calls and process them as they arrived, because it would have added unnecessary complexity to the project. However, in time-sensitive applications, it would be worthwhile to take the extra steps to process them as they arrive with MPI_Iallgather(). I chose not to, but I understand how to do so if the application warranted the extra work.

Returning Results

On both ends, I used a blocking call to MPI_Gather() to return results to rank 0. On the child nodes side, the call needs to be blocking to ensure that the data has begun transfer before the node's memory is freed. There is nothing else to do before the node completes its execution, so the data would be freed too soon without blocking. On the root side, the call is blocking since I need to report it, and there is no real useful work to be done in the interim. Again, I could have used an immediate call with a bank of request objects to process results as they came in, but there was not any time constraint and it was not worth the additional complexity for this project.