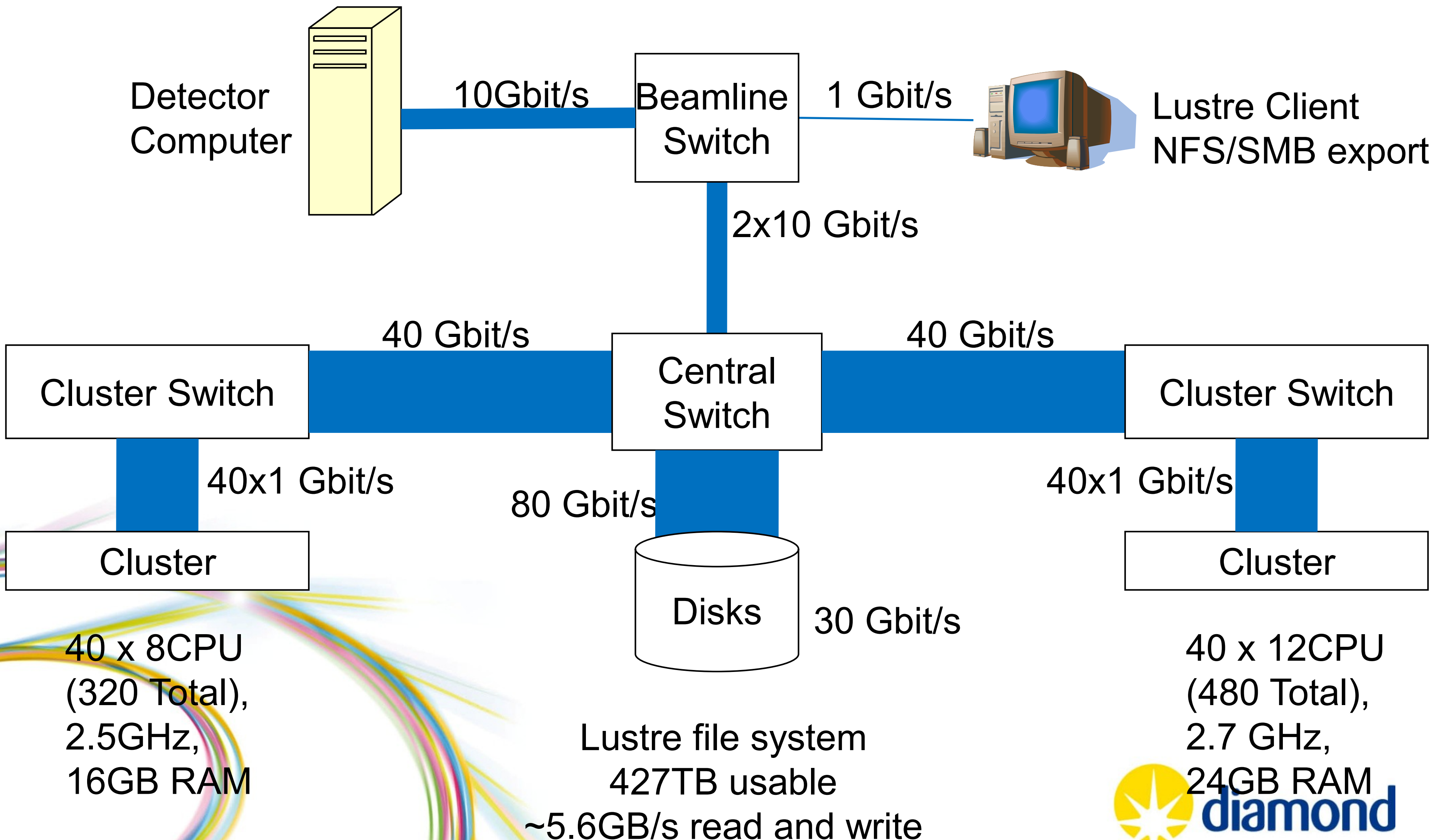
HDF5 Parallel Reading and Tomography Processing at DLS

Mark Basham
Scientific Software Team
Diamond Light Source Ltd.

Network Infrastructure

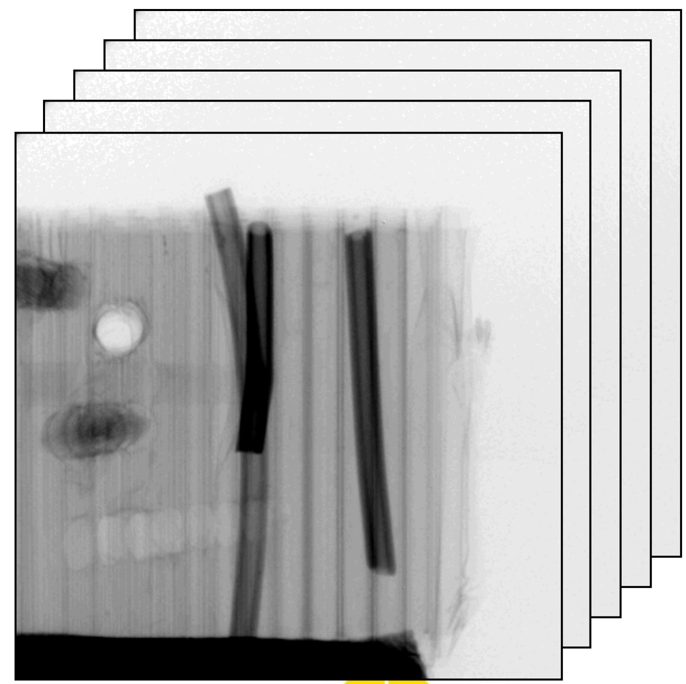


diamond

Tomography – Data Collection

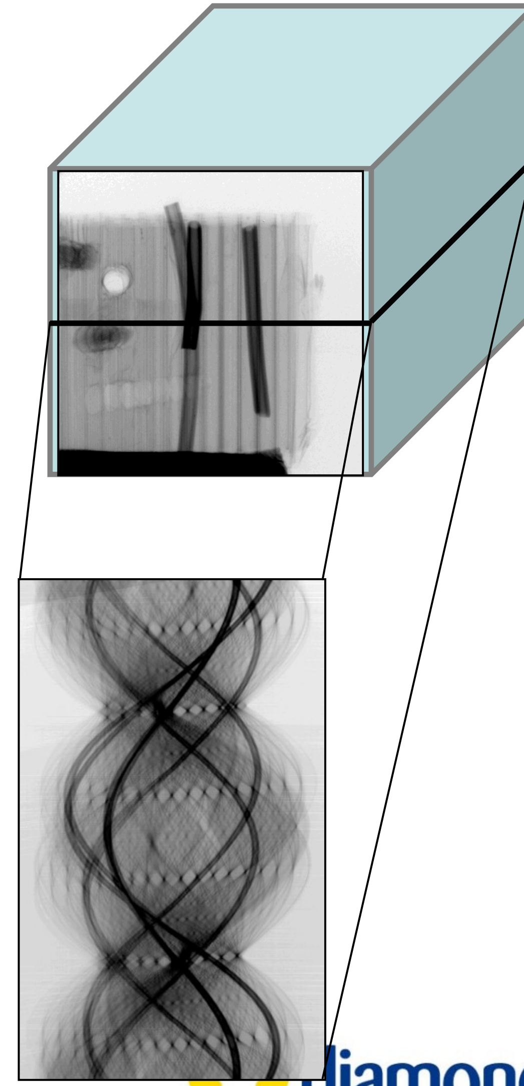
- Data is collected straight to HDF5 3D data arrays

- 4000x2600 pixel 16 bit greyscale image
- Stored as an [4000,2600,6000] array
- Totals around 120 GB of data
- Collection Time can be as fast as 30 minutes.



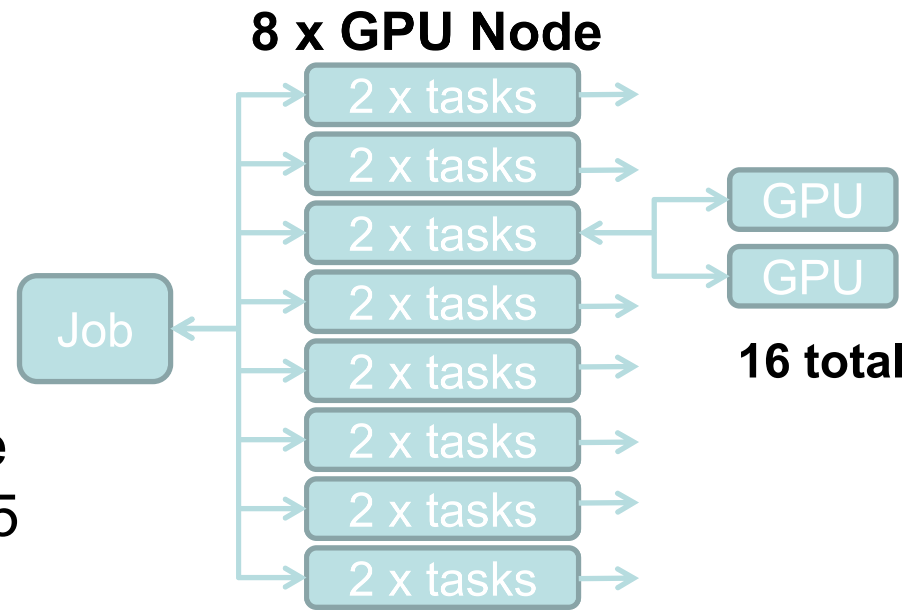
Tomography – Data Slicing

- The data is collected as [4000,2600,1] images
- The Camera software caches images though so that it can be written in [4000,16,8] chunks
- This is so that the data can be read out quickly as [4000,1,6000] sinograms, directly from the file.



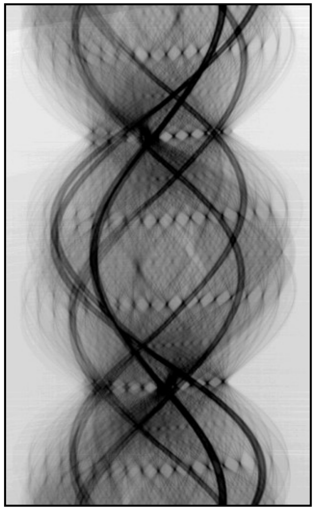
Tomography – Data Extraction

- Data is read out as a stack of sinograms using parallel hdf5 across 8 dedicated GPU cluster nodes, running 2 tasks per machine.
- The bandwidth from the Lustre file system using parallel HDF5 is 25MB/s per Job, meaning a total parallel read speed of about 400MB/s
- We are hoping to push this closer to line speed (50MB/s) with further optimisations.

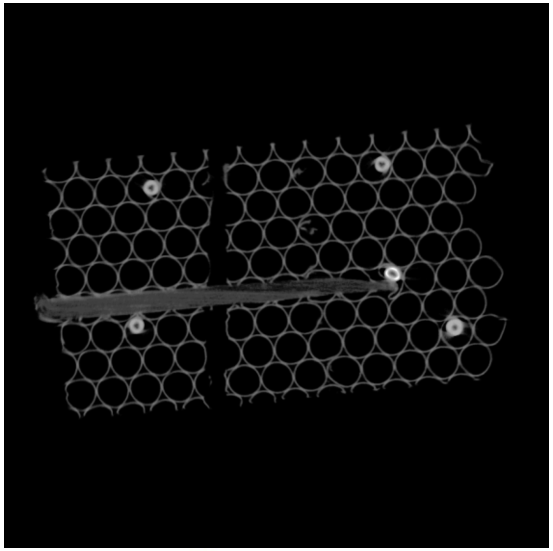


diamond

Tomography – Reconstruction



- Each of the 16 jobs then processes their set of sinograms using one of our GPUs
- Currently takes about 17 seconds per Sinogram, so this totals a reconstruction time of about 45 minutes
- We are currently looking to treble the number of GPU's in this older cluster, and buy a new GPU Cluster to decrease times, and deal with more tomography beamlines coming on line.

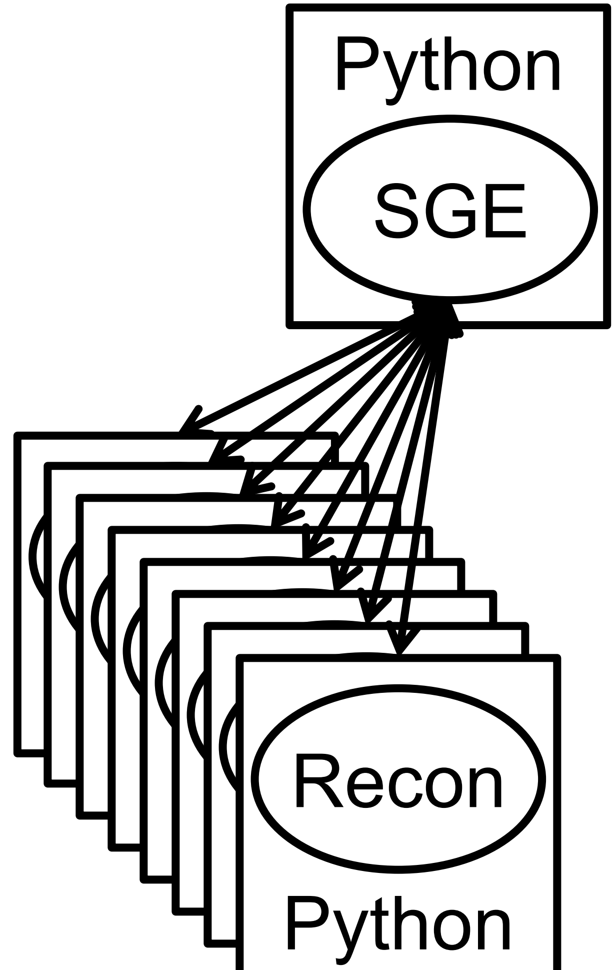


Tomography – Storage

- The results are then stored as a stack of TIFF images
- This will soon be instead to a single HDF5 volume
- This will mean that the user can leave Diamond with 2 files, one of the raw data, and one with the full reconstruction.

Tomography – Current Implementation Details

- Reconstruction routine is from Manchester University and converts a TIFF sinogram to a TIFF reconstructed slice.
- The HDF5 extraction code is written in python using numpy and h5py, and wrappers this program.
- There is a top level Python script which uses qsub to run the jobs in parallel on the cluster



Tomography – Current Implementation What we have Learned.

- H5py incredibly easy to deal with hdf5 and nexus files in python.
- There are some overheads in running 16 concurrent reads from the same file with h5py in our Lustre file system
- Be careful with caching
  - 1 x [4000,1,2000] – 18 seconds
  - 15 x [4000,1,2000] – 2 seconds each – 30 seconds
  - 1 x [4000,16,2000] – 16 seconds
- Cant write back easily at all.
  - H5py dose not lock as expected.

Next Steps – A generic HDF5 Reader with processing library

- Motivation
  - Scientists don't want to or know how to program properly with HDF5 or MPI or Both
  - A lot of reduction and processing steps are slice by slice processing
    - Tomography
    - NCD Data Reduction
    - MX Data Processing

Next Steps – A generic HDF5 Reader with processing library

- Implementation
  - Program written in c, using MPI and pHDF5
  - Connects to a user library which has a simple call-back style API, requiring a minimum of effort for someone to create and compile a processing library using any tools they wish
  - Most of the memory management is done outside of the library to try to keep down memory management issues

API – Get library info, initialise

- *int h5io_init(void* context, const h5io_global_param_t param, int *num_inputs, int *num_outputs);*
  - num_inputs = 3
  - num_outputs = 1
  - return 0

Parallel HDF5 Reading Infrastructure

- *int request_input(void* context, int input_index = 0, char **dataset_location);*
  - dataset_location => "/data/imagedata“
- *int set_input_slice(void* context, int input_index = 0, const h5io_dataspace_t *inputs, int **slicing_map);*
  - Input->shape = [1024,1024,5]
  - slicing_map => [0,0,1]
  - Return 0,1

Parallel HDF5 Reading Infrastructure

- *int request_input(void* context, int input_index = 1, char **dataset_location);*
  - dataset_location => "/ data/darkdata“
- *int set_input_slice(void* context, int input_index = 1, const h5io_dataspace_t *inputs, int **slicing_map);*
  - Input->shape = [1024,5]
  - slicing_map => [0,1]
  - Return 0,2

Parallel HDF5 Reading Infrastructure

- *int request_input(void* context, int input_index = 2, char **dataset_location);*
  - dataset_location => "/data/motorposition“
- *int set_input_slice(void* context, int input_index = 2, const h5io_dataspace_t *inputs, int **slicing_map);*
  - Input->shape = [25,30]
  - slicing_map => [0,0]
  - Return 0,3

Parallel HDF5 Reading Infrastructure

- *int request_output(void* context, int output_index = 0, char **dataset_location, h5io_dataspace_t *output);*
  - dataset_location => “output/data”
  - output
    - shape = [512,512,-1]
    - type = h5io_int32

Parallel HDF5 Reading Infrastructure

- *int process(void* context, int slice_number = 0, int num_inputs = 3, const h5io_dataspace_t *input_data, int num_outputs = 1, const h5io_dataspace_t *output_data);*
  - slice_number = 0
  - Input data
    - /data/imagedata = [1024,1024]
    - /data/darkdata = [1024]
    - /data/motorposition = [25,35]
  - Output data
    - /output/data = [512,512]

Parallel HDF5 Reading Infrastructure

- *int process(void* context, int slice_number = 1, int num_inputs = 3, const h5io_dataspace_t *input_data, int num_outputs = 1, const h5io_dataspace_t *output_data);*
  - slice_number = n
  - Input data
    - /data/imagedata = [1024,1024]
    - /data/darkdata = [1024]
    - /data/motorposition = [25,35]
  - Output data
    - /output/data = [512,512]

diamond

Parallel HDF5 Reading Infrastructure

- *int processing_complete(void* context);*
  - This is called once all the processing is complete, the user library should clean up any resources it is using here.
- *const char* get_error_code(void* context, int error_code);*
  - This function is called if any of the other library functions return an non zero value, it should be a case statement

Processing Libraries

- There are several example libraries planned for the pHDF5 reader tool:
  - Benchmarking library
  - Tiff Extractor
  - External Process wrapper
  - Direct interface to Manchester Tomography Program

Any Questions?