Teledyne DALSA • 880 Rue McCaffrey • St-Laurent, Québec, H4T 2C7 • Canada
https://www.teledynedalsa.com/

SAP-AN0012: High Frame Rate / Very Large Image Acquisition

Guide to Acquiring Very Large Images or at High Frame Rate

Applies to all Teledyne DALSA cameras and frame grabbers

Overview

As camera resolutions and frame rates increase managing very large image acquisition is critical to obtaining optimal system performance. This application note discusses some of the hardware and software aspects involved in very large image acquisition.

Prerequisites

- Sapera LT 8.60 or later recommended

Sapera LT 8.60
Sapera LT is the image acquisition and control software development kit (SDK) for Teledyne DALSA cameras, which includes CamExpert, a utility providing user-friendly access to camera features for configuration and setup. Sapera LT SDK is available for download from the Teledyne DALSA website:

http://teledynedalsa.com/imaging/support/downloads/sdks/

Administrator rights are required to install Sapera LT.

Hardware Considerations

RAM

The maximum size and number of buffers that can be allocated is limited by the available system RAM; the greater amount of RAM, the more memory is available for buffer allocation.

Configuring Contiguous Memory

The Sapera Configuration utility allows users to specify the total amount of contiguous memory to be reserved for allocating buffers and messages. This RAM memory is used by frame grabbers to allocate DMA tables. In general, contiguous memory is used by legacy applications with older frame grabbers; most applications should use scatter-gather type memory. However, a certain amount of contiguous memory is required for Sapera LT buffer descriptors and 1MB for every 3000 buffers should be allocated.

For more information refer to the Getting Started Manual for Frame Grabbers included with the Sapera LT installation.

Harddrives

With very large images and / or high frame rates, even with large amounts of RAM, the allocated Sapera buffers may quickly become filled; buffers are then overwritten in circular fashion. The frame rate may not allow sufficient time for these buffers to be processed. In these cases, it may be necessary to transfer these buffers to harddrives on the host to avoid data loss.

For example, PCI Express 3.0 and 4.0 SSD hardrives with M.2 interfaces can provide high transfer speeds that allow successful copying from RAM of enough buffers to avoid data loss during acquisition. Images can then be processed on the host system. When evaluating harddrives verify that the peak and sustained transfer speeds are adequate for the application.

Multiple hardrives in a RAID 0 configuration can also be used to increase transfer performance.

The Data Transfer Software Example demonstrates how to perform this image buffer transfer operation.

Multiple CPU Systems

For motherboards with multiple CPUs that implement NUMA (Non-Uniform Memory Access), Teledyne DALSA recommends disabling NUMA support to avoid potential negative performance effects.

In a NUMA system, physical CPUs are arranged in smaller systems called nodes. Each node has its own processors and memory, and is connected to the larger system through a cache-coherent interconnect bus. NUMA architecture is non-uniform because each processor is close to some parts of memory and farther from other parts of memory. The processor quickly gains access to the memory it is close to, while it can take longer to gain access to memory that is farther away.

For Sapera applications, this can cause increased latency depending on the specific CPU running certain threads and the physical location of buffer memory assigned by the OS at runtime. When disabled, performance is stable regardless of on which CPU framegrabber driver or Sapera LT processes run.

NUMA can be disabled in the motherboard's BIOS settings; refer to the motherboard documentation for more information.

For users that want to optimize performance using NUMA enabled hardware, the Windows API includes functions for NUMA management (for example, **GetNumaHighestNodeNumber**); refer to the Windows API documention for more information. Users must manage memory allocation and processing such that all operations are local to the node.

Data Transfer Software Example

The following C code snippet demonstrates how to transfer image buffers from RAM to a harddrive within an acquisition callback function. The example transfers 1000 buffers to the harddrive within a single file to reduce file IO overhead, while displaying images intermittently.

```
FILE *fp;
int file_count = 1000;
int buffer_count = 0;
int missed_count = 0;

void CGrabDemoDlg::XferCallback(SapXferCallbackInfo *pInfo)
{
   CGrabDemoDlg *pDlg= (CGrabDemoDlg *) pInfo->GetContext();

   // If grabbing in trash buffer, do not display the image, update the
   // appropriate number of frames on the status bar instead
   if (pInfo->IsTrash())
   {
      CString str;
      str.Format(_T("Frames acquired in trash buffer: %d"), pInfo->GetEventCount());
         missed_count = missed_count + pInfo->GetEventCount();
      pDlg->m_statusWnd.SetWindowText(str);
   }

   // Refresh view
   else
   {
          // Process current buffer
          void* bufferData;

          // Get the buffer data address
          pDlg->m_Buffers->GetAddress(&bufferData);
          int bufferWidth = pDlg->m_Buffers->GetWidth();
          int bufferHeight = pDlg->m_Buffers->GetHeight();
          int bufferBytesPerPixel = pDlg->m_Buffers->GetBytesPerPixel();

          // Get the buffer pitch in bytes
          int pitch = pDlg->m_Buffers->GetPitch();

          if (buffer_count == 1000) {
                 fclose(fp);
                 buffer_count = 0;
                 file_count++;
                                  }

          if (buffer_count == 0){
          char buffer[32]; // The filename buffer.

          // Put "file" then k then ".raw" in to filename.
          // The filename can include the complete path to required harddrive folder.
          snprintf(buffer, sizeof(char) * 32, "file%i.raw", file_count);
          fp = fopen(buffer, "wb");

          CString str;
          str.Format(_T("writing buffer: %d  missed count = %d"),
file_count,missed_count);
          pDlg->m_statusWnd.SetWindowText(str);
                           }
```

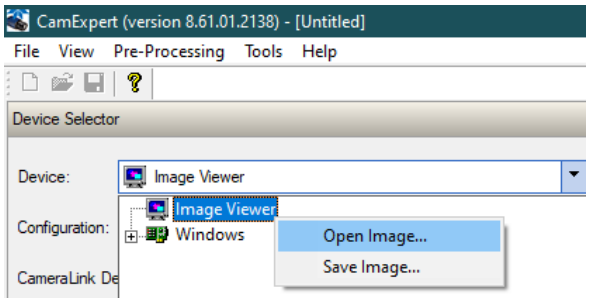
```
            buffer_count++;
            fwrite(bufferData, 1, (bufferWidth*bufferHeight*bufferBytesPerPixel), fp);

            // Display images intermittently.
           if(buffer_count/100 == 0)
                        pDlg->m_View->Show();
}
```

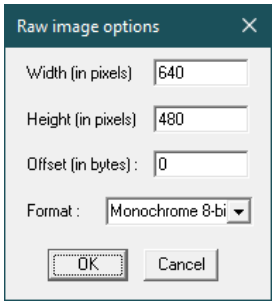
Viewing Images in CamExpert

Sapera's CamExpert application includes an Image Viewer to load and display images.

Images can be loaded using the File menu **Open Image...** command or by right-clicking on the Image Viewer icon in the Devices panel.



For *.raw* files containing multiple images, specify the image width and height; the offset in bytes allows for selecting the exact image within the file (*w* x *h* x *image index*)).



Creating a .mp4 file from Multiple Images

The following Python code demonstrates how to create a *.mp4* video file from a *.raw* file containing multiple images. The sample code extracts 1000 images (1080x2048) into a single *.mp4* file.

```
import time
import numpy as np
import glob
import cv2
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, 20.0, (2048,1080),0)
start_time = time.time()
for filename in glob.glob('*.raw'):
with open(filename, 'rb') as fd: # open in read-only mode
print (filename )
for x in range(0, 999):
rows = 1080
cols = 2048
fd.seek(rows*cols*x)
f = np.fromfile(fd, dtype=np.uint8,count=rows*cols)
if len(f) ==rows*cols:
im = f.reshape((rows, cols)) #notice row, column format
b = cv2.resize(im,(1024,1024),fx=0,fy=0, interpolation = cv2.INTER_AREA)
out.write(b)

fd.close()
out.release()
end_time = time.time()
print("Elapsed time was %g seconds" % (end_time - start_time))
```