VECTOR PROCESSING ENHANCEMENTS FOR REAL-TIME IMAGE ANALYSIS*

S. Shoaf, Advanced Photon Source,
Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, U.S.A.

Abstract

A real-time image analysis system was developed for beam imaging diagnostics. An Apple Power Mac G5 with an Active Silicon LFG frame grabber was used to capture video images that were processed and analyzed. Software routines were created to utilize vector-processing hardware to reduce the time to process images as compared to conventional methods. These improvements allow for more advanced image processing diagnostics to be performed in real time.

INTRODUCTION

Conventional methods for image processing often use loops in software to operate on one pixel at a time. This serial-based approach to pixel manipulation is very simple and easy to implement. The downside of such an approach is the amount of processing time required to work on each individual pixel. As the number of pixels in the image increases, so does the processing time that is required to analyze the data. This paper provides one possible solution to address the large processing times that occur when performing image analysis in real time.

SYSTEM OVERVIEW

An analog-based video system was developed to perform real-time analysis of x-ray beam images. Analog video signals from cameras are sent to a video multiplexer that routes the signal to the input of the analog video frame grabber. The analog frame grabber captures VGA-sized (640x480 pixel) video at 30 frames per second (fps). The images are then processed and analyzed in real time between image acquisitions. The operations involved in the capturing, processing, analysis, and image display must be performed quickly, as images are acquired every 33 ms. Because of computer system overhead, the entire 33 ms of time between image acquisitions is not completely available to a developer. This adds additional constraints to try and maintain real-time performance.

HARDWARE

The host computer was an Apple Power Mac G5 with dual 2.7-GHz processors, 1 GB of RAM, running Mac OS 10.4. The frame grabber was an Active Silicon LFG series PCI (AS-LFG4-PCI64) board. External TTL level trigger signals were connected to the frame grabber board for interrupt-driven response. Analog (RS-170) video cameras were used to feed an input signal to the frame grabber.

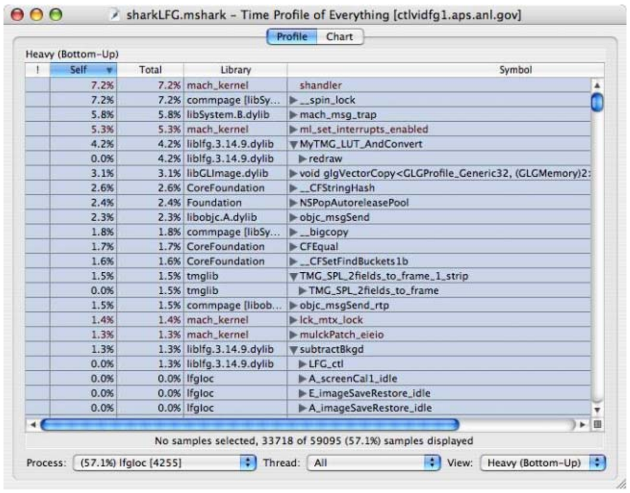
*Work supported by U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.

SOFTWARE

The host computer runs Mac OS 10.4 and EPICS R3.14.8 natively for the control system. Custom EPICS device support was written to control the frame grabber hardware and respond to the interrupt signals. A custom program written in C was developed to perform the image processing and analysis operations, and to display the resultant image all in real time. In order to improve on software efficiency, several other tools were used during the software development.

DEVELOPMENT TOOLS

Apple Inc. provides developer tools with its Mac OS X operating system. The XCode development suite and CHUD toolkit were used on this project. One very helpful tool available to Mac developers is Shark. Shark is used to analyze how much Central Processing Unit (CPU) time the different components of a program are using. From this information, a developer can see where they would gain the most performance improvement by optimizing code. When running Shark for the first time on this project, the analysis results showed that the most CPU cycles were being used by the image pixel manipulation routines and the graphics image display routine. A sample output from Shark is presented in Figure 1. Apple also provides the OpenGL Profiler tool that analyzes the performance of the video graphics card and reports how the Graphics Processing Unit (GPU) cycles are being used. A sample output from the OpenGL Profiler is shown in Figure 2. These tools provided the added capability to pinpoint the software routines that needed the most optimization in order to reduce the overall processing time.



sharkLFG.mshark – Time Profile of Everything [ctlvidfg1.aps.anl.gov]

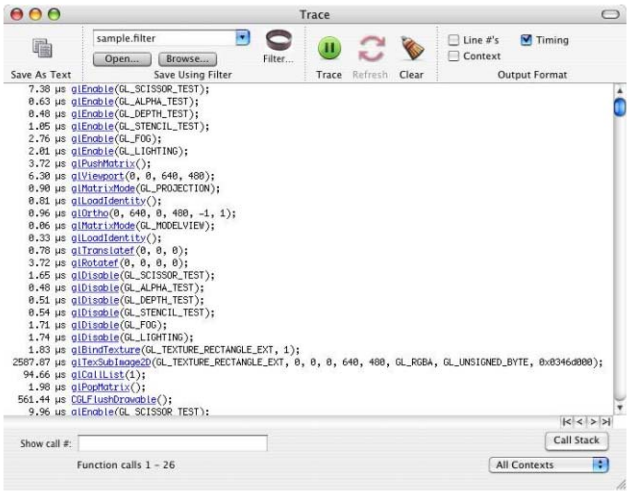
Heavy (Bottom-Up)

!	Self	Total	Library	Symbol
	7.2%	7.2%	mach_kernel	shandler
	7.2%	7.2%	commpage [libSy...	▶ __spin_lock
	5.8%	5.8%	libSystem.B.dylib	▶ mach_msg_trap
	5.3%	5.3%	mach_kernel	▶ ml_set_interrupts_enabled
	4.2%	4.2%	liblfg.3.14.9.dylib	▼ MyTMG_LUT_AndConvert
	0.0%	4.2%	liblfg.3.14.9.dylib	▶ redraw
	3.1%	3.1%	libGLImage.dylib	▶ void glgVectorCopy<GLGProfile_Generic32, (GLGMemory)2:
	2.6%	2.6%	CoreFoundation	▶ __CFStringHash
	2.4%	2.4%	Foundation	▶ NSPopAutoreleasePool
	2.3%	2.3%	libobjc.A.dylib	▶ objc_msgSend
	1.8%	1.8%	commpage [libSy...	▶ __bigcopy
	1.7%	1.7%	CoreFoundation	▶ CFEqual
	1.6%	1.6%	CoreFoundation	▶ __CFSetFindBuckets1b
	1.5%	1.5%	tmglib	▼ TMG_SPL_2fields_to_frame_1_strip
	0.0%	1.5%	tmglib	▶ TMG_SPL_2fields_to_frame
	1.5%	1.5%	commpage [libob...	▶ objc_msgSend_rtp
	1.4%	1.4%	mach_kernel	▶ lck_mtx_lock
	1.3%	1.3%	mach_kernel	▶ mulckPatch_eieio
	1.3%	1.3%	liblfg.3.14.9.dylib	▼ subtractBkgd
	0.0%	1.3%	liblfg.3.14.9.dylib	▶ LFG_ctl
	0.0%	0.0%	lfgloc	▶ A_screenCal1_idle
	0.0%	0.0%	lfgloc	▶ E_imageSaveRestore_idle
	0.0%	0.0%	lfgloc	▶ A_imageSaveRestore_idle

No samples selected, 33718 of 59095 (57.1%) samples displayed

Process: (57.1%) lfgloc [4255] Thread: All View: Heavy (Bottom-Up)

Figure 1: Sample output from Shark developer tool.



```
   7.38 µs glEnable(GL_SCISSOR_TEST);
   0.63 µs glEnable(GL_ALPHA_TEST);
   0.48 µs glEnable(GL_DEPTH_TEST);
   1.05 µs glEnable(GL_STENCIL_TEST);
   2.76 µs glEnable(GL_FOG);
   2.01 µs glEnable(GL_LIGHTING);
   3.72 µs glPushMatrix();
   6.30 µs glViewport(0, 0, 640, 480);
   0.90 µs glMatrixMode(GL_PROJECTION);
   0.81 µs glLoadIdentity();
   0.96 µs glOrtho(0, 640, 0, 480, -1, 1);
   0.06 µs glMatrixMode(GL_MODELVIEW);
   0.33 µs glLoadIdentity();
   0.78 µs glTranslatef(0, 0, 0);
   3.72 µs glRotatef(0, 0, 0, 0);
   1.65 µs glDisable(GL_SCISSOR_TEST);
   0.48 µs glDisable(GL_ALPHA_TEST);
   0.51 µs glDisable(GL_DEPTH_TEST);
   0.54 µs glDisable(GL_STENCIL_TEST);
   1.71 µs glDisable(GL_FOG);
   1.74 µs glDisable(GL_LIGHTING);
   1.83 µs glBindTexture(GL_TEXTURE_RECTANGLE_EXT, 1);
2587.87 µs glTexSubImage2D(GL_TEXTURE_RECTANGLE_EXT, 0, 0, 0, 640, 480, GL_RGBA, GL_UNSIGNED_BYTE, 0x0346d000);
  94.66 µs glCallList(1);
   1.98 µs glPopMatrix();
 561.44 µs CGLFlushDrawable();
   9.96 µs glEnable(GL_SCISSOR_TEST);
```

Show call #:

Function calls 1 - 26

Figure 2: Sample output from OpenGL Profiler tool.

Once the sections of the code that were using the most CPU cycles during execution were identified by the use of the aforementioned tools, a technique had to be developed and/or used to reduce the processing time. The potential speed increases that could be achieved by taking advantage of the large bandwidth capabilities that vector processing offers made it a clear choice for this project.

VECTOR PROCESSING

The G5 processor is used in the Apple Power Mac computers. There are system optimizations in the G5 platform available to a programmer that can enhance performance [1]. The G5 processor has a special processing unit called the Velocity Engine, which is built into the PowerPC architecture. The Velocity Engine is a 128-bit vector execution unit that operates on large amounts of data in parallel. By writing software that utilizes the AltiVec instruction set in the Velocity Engine, one can lower processing time for high-bandwidth applications. Speed enhancements of up to 30X on operations are possible using AltiVec code [2].

One of the typical image operations implements a background noise level subtraction. The background image subtraction function was rewritten to take advantage of the AltiVec vector processing support in the Power PC chip. Figure 3 shows a representation of a single vector subtraction operation. Using vectors, 16 8-bit pixels of the acquired image are loaded into a vector. The same corresponding pixels from a previously captured background noise level image are loaded into a different vector. The 16 background pixels are subtracted in parallel from the 16 acquired image pixels and stored in another vector. The results of the subtraction operation are stored in memory. This process is repeated until all of the pixels in the originally acquired image have had the corresponding pixels from the background image subtracted as well. The vector subtraction implementation of the background noise level resulted in a four-fold decrease from the original time it took to subtract an image. Using a simple for-next loop, it took ~4 ms to perform a background subtraction on the image. The new vector-based background subtraction completes in less than 1 ms. The vector operation did not result in a speed increase of 16X even though there were 16 pixels being subtracted simultaneously. Although there are many advantages to using vectors, their overhead costs still limit the overall performance increase.

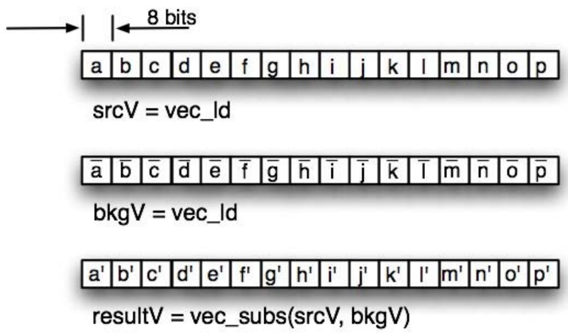

8 bits
a b c d e f g h i j k l m n o p
srcV = vec_ld
ā b̄ c̄ d̄ ē f̄ ḡ h̄ ī j̄ k̄ l̄ m̄ n̄ ō p̄
bkgV = vec_ld
a' b' c' d' e' f' g' h' i' j' k' l' m' n' o' p'
resultV = vec_subs(srcV, bkgV)

Figure 3: Representation of a single vector operation for the background noise-level subtraction operation.

One of the desired operations was to average up to 255 images in a moving boxcar algorithm. The simple for-next loop would take approximately 4 ms to add an image into the buffer and another 4 ms to subtract an image that needs to be removed from the boxcar once the loop reached the number of images that should be included. That was an additional 8 ms of processing time that would have been added to the total time, not including the time to do the actual averaging calculation. The total time had to be well less than 33 ms, otherwise the system could not acquire and analyze the next frame in real time. There was other overhead in the system that typically interrupted processing as well, while important system interrupts were handled. All of this implied that the image operations should be performed in the least amount of time possible. A vector-processing algorithm was developed to perform this operation. It is not shown in the text of this paper because of size constraints. The new vector-processing routine for this operation takes less than 3 ms to complete its task.

Another function taking a considerable amount of time was the required conversion of an 8-bit grayscale image to a 24-bit color image for display on the monitor. A vector-based image conversion routine to convert the 8-bit grayscale image to a 24-bit RGB color image was written and reduced processing time from 15 ms down to 10 ms. Figure 4 shows how the 8-bit pixels in the vector were extended to create a 24-bit RGBX format in memory. There are four iterative operations that take place for each of the 16 8-bit pixels from the original image to extend it to 24 bits. This new routine was a marked improvement in processing time, but further improvements were still required. By utilizing the OpenGL Profiler tool from Apple, improvements in the OpenGL display routine were then developed to reduce the overall display time down to less than 5 ms.

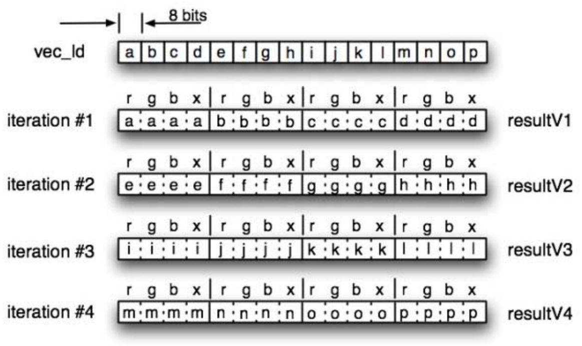

8 bits
vec_ld
a b c d e f g h i j k l m n o p
r g b x
iteration #1
a a a a b b b b c c c c d d d d
resultV1
iteration #2
e e e e f f f f g g g g h h h h
resultV2
iteration #3
i i i i j j j j k k k k l l l l
resultV3
iteration #4
m m m m n n n n o o o o p p p p
resultV4

Figure 4: Representation of the vector operation to extend an 8-bit grayscale image into a 24-bit color image.

CONCLUSION

Conventional methods for image processing in a real-time image analysis system can limit the complexity and/or quantity of analysis being performed. By taking advantage of vector processing, one can greatly decrease the processing time needed to perform image analysis operations. This then allows for more complex image processing and/or analysis techniques to be implemented in a real-time beam image diagnostic tool. The work done in this paper was performed on a PowerPC-based Apple computer. In the future, there are plans to convert the AltiVec vector instructions to SSE vector instructions to support Intel-based Apple computers [3]. SSE instructions are the Intel 128-bit vector extensions similar to AltiVec. Work is also planned to implement the techniques in this paper to a 16-bit Camera Link® digital-based image acquisition system.

ACKNOWLEDGMENTS

I would like to thank my colleague Dr. William Eric Norum of the APS Controls Group for all of his programming insights and his many contributions. I would also like to thank the technical support staff from Active Silicon, Inc. for their assistance with the frame grabber hardware. One last thank you goes to the Mac support group at the APS, who contributed many hours to configure our systems so they worked like any other Unix system used onsite.

REFERENCES

[1] Apple Inc, “Optimizing for the Power Mac G5,” http://developer.apple.com/performance/g5optimization.html
[2] Apple Inc., “Velocity Engine,” http://developer.apple.com/hardwaredrivers/ve/index.html
[3] Apple Inc., “SSE Performance Programming,” http://developer.apple.com/hardwaredrivers/ve/sse.html