Lab 4 Report

Tejas Nitin Joshi

ECEN 5653 Real Time Digital Media

1. **Clearly identify who you will partner with on the Extended Lab or indicate if you plan to work alone. If you have a partner, you must each turn in an individual copy of this lab write-up, but your results for the algorithm prototyping may be a joint effort.**

   I shall be working alone for the Extended lab.

2. **[5 pts] Read Video Fingerprinting and Encryption Principles for Digital Rights Management. Please summarize the paper's main points (at least 3 or more) in a short paragraph.**

   For multicast servers the DRM has become an important application as the number of users have increased multifold ans so has the piracy of the content. Which brings into play the DRM for VoD providers. This usually involves a tradeoff on the provider and the consumer. Increased overheads and for enctyption and security has to be accomodated by the provider and consumer all the while maintaining the real-time requirements. This brings in the use of digital fingerprinting techniques.

   The transmitter utilizes greater bandwidth for a multiprogram multicast, as each stream is encrypted with a single key and and different fingerprint. For the reciever the fingerprinting can be affected by jitters. Here JDF is a better option.

   The paper discusses different partail encryption techniques which uses certain frames (Essental [EF] and Non-Essential[NEF] distingusihing) to perform selective /partial encription techniques which involves performing DCT and quantization along with scrambling the frame with a zig-zag run over the frame i.e shuffling. In contradiction with this technique the bitrate increases and this is overcome using VEA for MPEG. Another technique used here is the Hierarchial Based Encryption which utilizes wavelet compressions and linklists.

   JFD encryption is the same from the source using a common key for the group, whereas the receivers, are given slightly different keys, the convolution of the source and the receiver keys makes the end-decrypted content slight different in terms of the LSB content of the output,d(rendering fingerprinting). It uses the concept of chameleon ciphers and experiments of JFD on frames led to conclusions that, the method is still /susceptible to Key Collision Attack.

3.[5 pts] **Read through the basics on the** CUDA Developer Zone**,** The Khronos Group OpenCL web pages **and the** AMD StreamProc SDK **web pages.**
Describe how you could write a CUDA or OpenCL kernel for the most compute intensive portions of your frame processing to provide GPU-assist speed-up on NVIDIA GPUs using ECES system, Amazon, or your own equipment.
If you don't plan to use OpenCL or CUDA to accelerate your frame processing, how do you plan to provide acceleration?

Let us take a simple example of vector addition we have two arrays A and B which contain vectos and their sum is to be stored in C.

On a normal C program the program would use a for loop

```
for(i=0;i<N;i++)
        C[i]=A[i]+B[i]
```

This is the most repeteive part of the program and would be performed N times resulting in 3 memory operations and a addition operation for every vector pair.

When using a GPGPU for such calculation , the GPGPU has multiple cores and would be able to perform these calculations in parallel.

Hence the kernel module (in this case the for loop) would be created so that it can be given a id and individual cores could perform the vector addition at the same time.

Example kernel for the vector add:

```
        __global__ void VecAdd(const float* A, const float* B, float* C, int N)
{
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   if (i < N)
      C[i] = A[i] + B[i];
}
```

Here a thread Id is assigned to each core and this accelerates the preocessing.

4. **[10 pts] Linux Trace and Profile Tools Exploration Linux has some interesting profiling and trace tools that can be used in the Extended Lab to take you beyond simple syslog and timestamps. Please read through Systemtap,Linux ftrace, Kernel Shark and sysprof. Experiment with each if you'd like and please describe a major advantage and disadvantage of each trace/profile tool and provide a summary of your plans for profling and tracing in the extended lab.**

   System Map Summary:

   Tool that is used to write simple scripts that can perform deep analaysis of activities of a Linux System.

   The systemtap script names events and provides handlers for each. Whenever a event occurs the Linux kernel runs the handler like a quick subroutine and then resumes.

   Event can be classified as enetering function, exiting function, timer expiration, or starting or stopping the systemtap .

   Handler scrip usually involves extraction of data from the event context, storing them into internal variables or printing results.

   All the systemtap processes are driven from a sigle command line program stap(system script translator/driver).

   This program accepts probing instructions and converts and compiles it into C code and loads the kernerl modeule to perform the trac/probe function.

   PROBES:

   different kinds of probes are used for the systemtap script that calls different handlers when a certain event occurs.

   A complete listing of all the probes can be found in the stapprobes man page along with classification of probe families.

   PRINT:

   as set of print functions are also used which can print thrad ids, Process ids , user ids etc for identification of nested events and also concurrent events.

   Ftrace(Function Tracer)

   is included within the 2.6.27 kernel. The ftrace framework is not scripting intensive like Systemtap. It is much simpler. A set of virtual files in a debugfs directory can be ised to enable specific tracers on the kernel.

   The function "tracer" simply outputs each function called in the kernel as it happens.

   Other tracers are for events like wakeup latency, events enabling and disabling interrupts and preemption , task switches etc.

   Ftrace also enables developers to design and add their own custom tracers.

KernelShark is a front end reader for trace-cmd output.

The trace-cmd command interacts with the ftrace tracer,

The Kernel Shark has a GUI various ways for representing events in the form of graphs , lists, Task and CPU plots.

It can perform filtering of events and tasks.

sysprof is a profiling application, which lets user see what functions are taking up most of the CPU time so you can concetrate your optimization efforts on making the pieces of code run faster.

With the sysprof profiling for machine, including a multithreaded and multiprocessed application that are running on your machine.

The sysprof window is divided into three main sections: Functions, Callers, and Descendants lists. The Functions list is your starting point. By default you will see a list of the executed functions, starting with the one that took the largest total time to execute. Items in the Functions list include executable processes, shared libraries, and the kernel itself.

5.[20 pts] Build, run, and optimize the multi-core thread grid C code for sharpen (found here) for a dual-core or better machine (consider using an Amazon scalable instance to test or eces if you don't have a multi-core machine yourself). Now, profile using sysprof (yum install sysprof) and provide an overview of how much CPU your machine is using, how much is being spent on the 12 megapixel sharpen transformation, and how effectively you are loading all CPU cores.

Below is a expanded screen of “sharpen_grid” stats from sys prof:

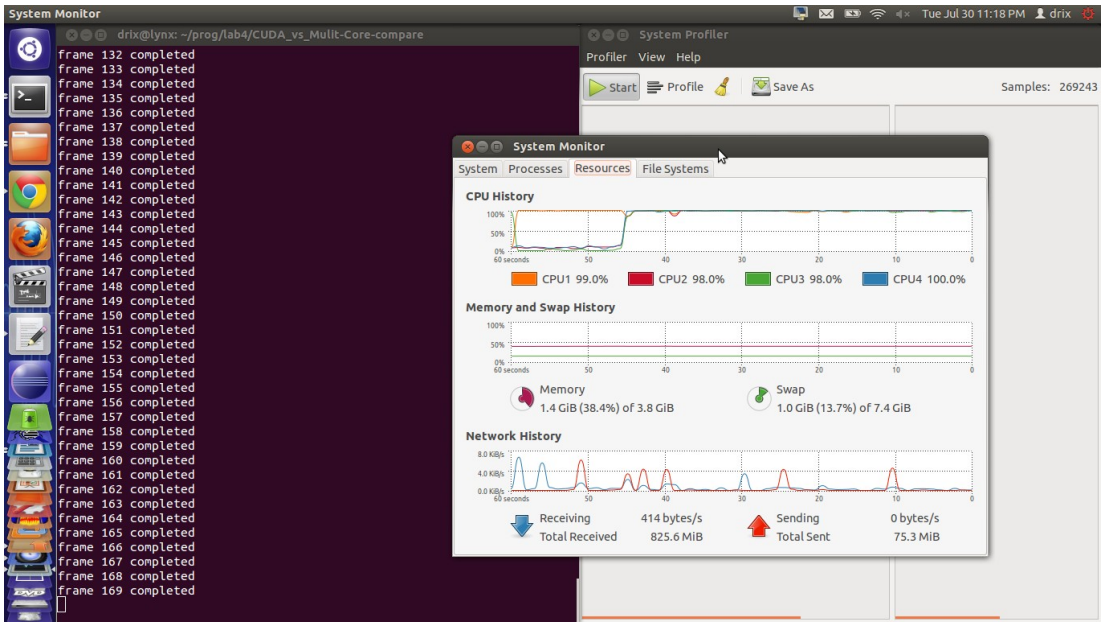
[sharpen_grid]	0.00%	76.61%
sharpen_thread	51.10%	51.20%
In file /lib/x86_64-linux-gnu/libpthread-2.15.so	0.05%	21.20%
__read_nocancel	0.03%	3.76%
No map [sharpen_grid]	0.00%	0.19%
- - kernel - -	0.00%	0.09%
main	0.03%	0.04%
__pthread_disable_asynccancel	0.02%	0.02%
start_thread	0.00%	0.02%
madvise	0.00%	0.02%
__pthread_enable_asynccancel	0.02%	0.02%
__read	0.01%	0.01%
write	0.01%	0.01%

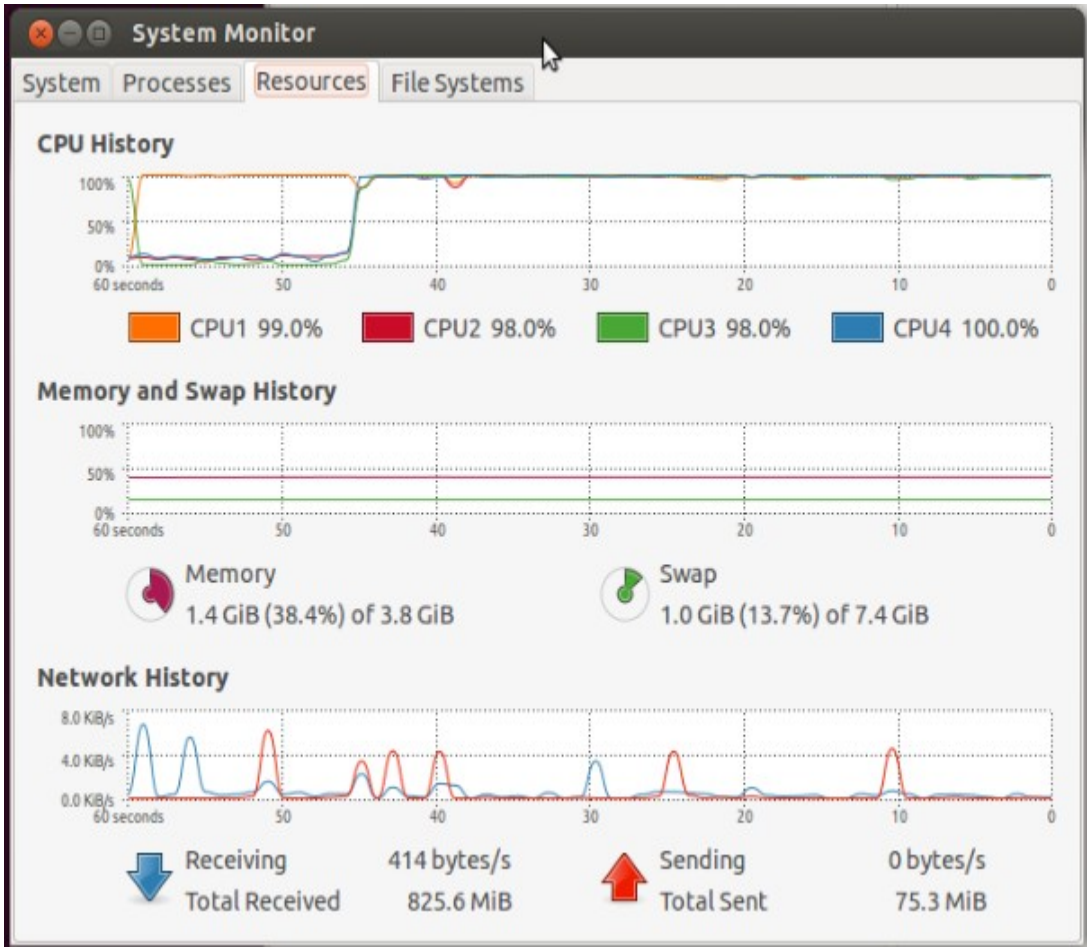
The sharpen program is utilizing **76.61 %** of the total is being utilized by sharpen_grid hence the remainder **23.39%** is being utilized by the system.

[Everything]	0.00 %	100.00 %
[sharpen_grid]	0.00 %	76.61 %
[sysprof]	0.00 %	7.09 %
[gnome-system-monitor]	0.00 %	4.23 %
[/usr/bin/X]	0.00 %	3.70 %
[compiz]	0.00 %	1.96 %
[kswapd0]	0.00 %	1.55 %
[/usr/lib/vmware/bin/vmware-vmx]	0.00 %	1.13 %
[nautilus]	0.00 %	0.76 %
[VC manager]	0.00 %	0.71 %
[/usr/bin/python]	0.00 %	0.53 %
[WorkerPool/6646]	0.00 %	0.29 %
[/opt/google/chrome/chrome]	0.00 %	0.26 %
[/opt/google/chrome/chrome --type=ppapi --channel=11723.283.685420180 --lang=en-US --enable-crash-r...	0.00 %	0.22 %
[/opt/google/chrome/chrome --type=renderer --lang=en-US --force-fieldtrials=AsyncDns/AsyncDnsB/Auto...	0.00 %	0.21 %
[pool]	0.00 %	0.11 %
[gnome-terminal]	0.00 %	0.09 %
[kworker/u:0]	0.00 %	0.04 %
[/usr/lib/indicator-appmenu/hud-service]	0.00 %	0.04 %
[/usr/lib/unity/unity-panel-service]	0.00 %	0.03 %
[kthreadd]	0.00 %	0.03 %
[//bin/dbus-daemon]	0.00 %	0.03 %
[/usr/lib/libreoffice/program/soffice.bin]	0.00 %	0.03 %
[/usr/lib/vmware/bin/thnuclnt]	0.00 %	0.02 %
[/usr/lib/gnome-settings-daemon/gnome-settings-daemon]	0.00 %	0.02 %
[/usr/lib/vmware/bin/vmware-hostd]	0.00 %	0.02 %
[/usr/lib/bamf/bamfdaemon]	0.00 %	0.02 %
[kworker/u:1]	0.00 %	0.01 %

Loading:

All four cores on the i5 processor are loaded to 100% when the convolution process runs. But while reading and writing the image file , one of the cores is at 100% while others are being utilized at 8% to 60% fluctuating rates .





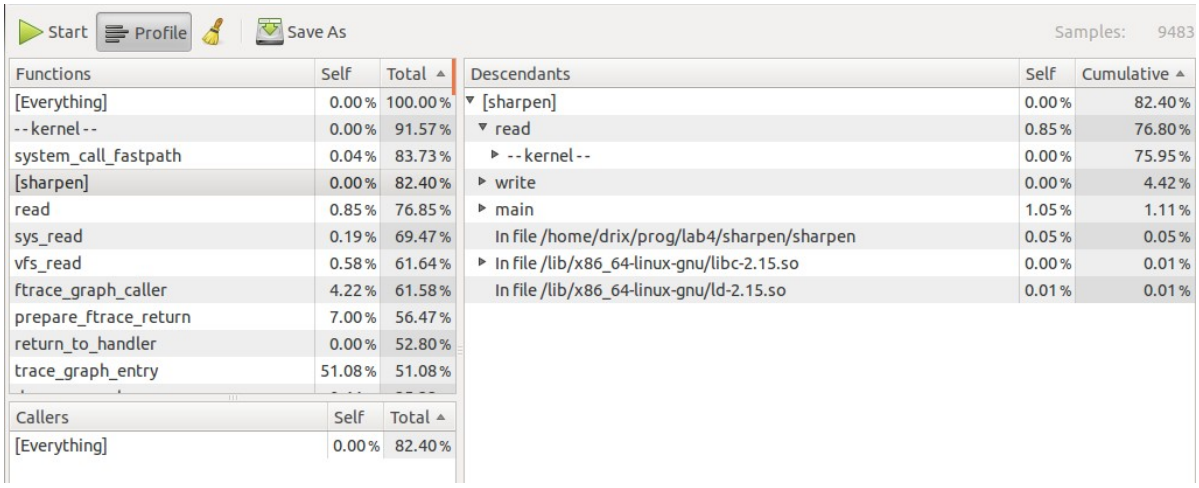
6. [20 pts] Build, run, and optimize the CUDA kernel for sharpen (found here) using your own CUDA enabled GPU or eces CUDA enabled nodes: mercedes, ecee-gpu4, ecee-gpu5 by logging into eces-shell and then ssh to any of the CUDA nodes). Note that the disk on the CUDA nodes should be considered "scratch", so keep all important code and data on the eces filesystem. Use the parameters you find from ad device query (C/bin/linux/release - ./deviceQuery in CUDA SDK) to optimize your thread grid. The CUDA Occupancy Calculator may be useful as you size your thread-grid blocks in the kernel for optimum performance. Measure the time it takes to transform the 12 megapixel image and using your timing measurements, show that your thread-grid sizing for the blocks in your kernel are optimal.

I was able to compile the cuda_sharpen.cu and sucessfully logged onto the ecee-gpu4 node but when executing the binary the ssh would hang everytime: (screenshot below)

```
tejo9148@ecee-gpu4:~/win7folders/Documents/RTDM/cuda_sharpen/CUDA_vs_Mul
it-Core-compare$ ls
bin                          cuda_sharpen        obj
Cactus-12mpixel.ppm          cuda_sharpen.c      Prime-Ramp-Down.png
Cactus-12mpixel-sharpen.ppm  cuda_sharpen.cu     Prime-Ramped.png
cmake_install.cmake          erast.c             sharpened.ppm
CMakelists.txt               erast_gpu.cu        sharpen_grid
CMakelists.txt~              Makefile            sharpen_grid.c
CMakeLists.txt               Makefile~           sharpen_grid.o
common.mk                    Makefile_erast.txt
CUDA_Occupancy_calculator.xls Makefile.txt
tejo9148@ecee-gpu4:~/win7folders/Documents/RTDM/cuda_sharpen/CUDA_vs_Mul
it-Core-compare$ ./cuda_sharpen Cactus-12mpixel.ppm
```

7.[20 pts] Using (yum install sysprof) and simple time-stamps profile your frame processing proposed code for the Extended lab running on just one frame transformation and identify clearly where the hot spots are in the code (where most time is spent) by function or basic code block and chart the top 9 locations along with a category for everything else as the 10th. If you don't have an algorithm done in time to do this, use sharpen.c on a 1080p or 720p and provide the same analysis for that well known frame processing algorithm.

Below show is the output of the Decendents window on sysprof



It is seen that the read and write-back of the image occupies most of the cpu .

8.[20 pts] Now run the same code over 30 frames and determine the IO time compared to CPU time for the entire run of 30 frames. Identify whether IO or CPU is the major bottleneck for throughput and quantify each on a per frame bases in milliseconds of latency. Describe how you can read ahead to get frames into memory before they are processed so that while there will be a start-up latency, once your pipeline is running, you won't be blocked by IO - how much memory would this require? What sort of bandwidth are you getting from your disk storage subsystem as shown with systat? Again, if your algorithm is not ready, use sharpen.c instead in the mean time.

Output for single HD frame of 1600x900 pixels

```
drix@lynx:~/prog/lab4/sharpen$ ./sharpen lion_hd_wallpaper.ppm
Cycle Count=18446744072977558546
Based on usleep accuracy, CPU clk rate = 18446744072977558546 clks/sec,
18446744072977.6 Mhz
tv_sec=1375273495, tv_usec=642783, sec=1375273495.642783
tv_sec=1375273503, tv_usec=84246, sec=1375273503.084246
stopTOD=1375273503.084246, startTOD=1375273495.642783, dt=7.441463
tv_sec=1375273503, tv_usec=84394, sec=1375273503.084394
tv_sec=1375273503, tv_usec=181128, sec=1375273503.181128
stopTOD=1375273503.181128, startTOD=1375273503.084394, dt=0.096734
Convolution time in cycles=241492975, rate=18446744072977, about 0 milli
secs
tv_sec=1375273503, tv_usec=181320, sec=1375273503.181320
tv_sec=1375273542, tv_usec=993170, sec=1375273542.993170
stopTOD=1375273542.993170, startTOD=1375273503.181320, dt=39.811850

Total read time: 7441.462994 millisec
Total write time: 39811.850071 millisec

Total IO time: 47253.313065 millisec
Total convolution time: 96.734047 millisec
```

seel below (output of 30 frames)

Output of 30 frames:

```
drix@lynx:~/prog/lab4/sharpen$ ./sharpen lion_hd_wallpaper.ppm
Cycle Count=18446744069614443438
Based on usleep accuracy, CPU clk rate = 18446744069614443438 clks/sec,
18446744069614.4 Mhz
tv_sec=1375273810, tv_usec=375081, sec=1375273810.375081
tv_sec=1375273817, tv_usec=661688, sec=1375273817.661688
stopTOD=1375273817.661688, startTOD=1375273810.375081, dt=7.286607
tv_sec=1375273817, tv_usec=661812, sec=1375273817.661812
tv_sec=1375273817, tv_usec=751914, sec=1375273817.751914
stopTOD=1375273817.751914, startTOD=1375273817.661812, dt=0.090102
Convolution time in cycles=224916882, rate=18446744069614, about 0 milli
secs
tv_sec=1375273817, tv_usec=752118, sec=1375273817.752118
tv_sec=1375273856, tv_usec=893392, sec=1375273856.893392
stopTOD=1375273856.893392, startTOD=1375273817.752118, dt=39.141274

Total read time: 218598.210812 millisec
Total write time: 1174238.219261 millisec

Total IO time: 1392836.430073 millisec
Total convolution time: 2703.058720 millisec
drix@lynx:~/prog/lab4/sharpen$
```

As seen from the timing statistics *I/O is very expensive* and proves as a bottleneck.

The disk usage statistic as generaed by the *sar – b command* , part of the sysstat library

```
drix@lynx:~/prog/lab4/sharpen$ sar -b -o datafile 1 20
Linux 3.5.0-37-generic (lynx)   07/31/2013      _x86_64_        (4 CPU

07:11:56 AM       tps      rtps      wtps   bread/s   bwrtn/s
07:11:57 AM      0.00      0.00      0.00      0.00      0.00
07:11:58 AM      0.00      0.00      0.00      0.00      0.00
07:11:59 AM      0.00      0.00      0.00      0.00      0.00
07:12:00 AM     12.00      0.00     12.00      0.00     96.00
07:12:01 AM      6.00      0.00      6.00      0.00     88.00
07:12:02 AM      1.00      0.00      1.00      0.00      8.00
07:12:03 AM      0.00      0.00      0.00      0.00      0.00
07:12:04 AM      0.00      0.00      0.00      0.00      0.00
07:12:05 AM      4.00      0.00      4.00      0.00    288.00
07:12:06 AM      0.00      0.00      0.00      0.00      0.00
07:12:07 AM      7.00      0.00      7.00      0.00    120.00
07:12:08 AM      0.00      0.00      0.00      0.00      0.00
07:12:09 AM      0.00      0.00      0.00      0.00      0.00
07:12:10 AM      4.00      0.00      4.00      0.00    104.00
07:12:11 AM      0.00      0.00      0.00      0.00      0.00
07:12:12 AM      4.00      0.00      4.00      0.00     48.00
07:12:13 AM      0.00      0.00      0.00      0.00      0.00
07:12:14 AM      0.00      0.00      0.00      0.00      0.00
07:12:15 AM      0.00      0.00      0.00      0.00      0.00
07:12:16 AM      0.00      0.00      0.00      0.00      0.00
Average:         1.90      0.00      1.90      0.00     37.60
drix@lynx:~/prog/lab4/sharpen$
```

The maximum bandwidth reached is **288 bits/second**. I believe read is not active as i used the execution binary(sharpen) toomany times and the data is already in the main memory.