Assignment 02: Mean Shift Tracking
IN4342 Embedded System Lab

Purpose
Implement in an efficient way, the Mean Shift Tracking application on a multi-core heterogeneous computing platform, namely the Beagle board [1].

Description
One common scenario for developing an embedded system application is having a team of engineers developing the application on a desktop computer, and, once completed, give it to a team of system engineers to implement it on the final system. In this assignment, you play the role of the system engineers that received the task of implementing such application on the OMAP 3530 system [2]. You will have to implement this application on the target platform and ensure that it executes according to the requirements for a set of data. You are free to modify the application as long as the functionality remains the same and you do not change the edge detection algorithm to a different algorithm. Reorganizing computations/loops/etc is permitted though. If you are in doubt please ask.

Application

Object tracking [4] is an important topic in the field of the computer vision and pattern recognition. The mean-shift algorithm [5,6,7] is an efficient approach to tracking objects whose appearance is defined by histograms. The Mean Shift algorithm is a robust, non-parametric technique that climbs the gradient of a probability distribution to find the peak of the distribution. Firstly, calculate the value of the mean shift of the current point, and move that point to a new position according to the value of the mean shift, then set this as the new starting point, and continue moving it until certain condition is met.

You are given a complete implementation of this algorithm, which utilizes the OpenCV library [8] mainly for extra tasks like video reading/writing etc. Video frames are read one by one and the object of interest (represented a rectangle in main.cpp) is tracked in these video frames to generate an output video, which is the result of tracking. The accompanying Makefiles contains the necessary compilation, execution, profiling and debugging rules for PC and ARM.

Compilation for PC:
To compile the application for PC, pc.mk file can be used, for example:

make -f pc.mk all

Similarly, other rules can also be specified instead of **all.**

Cross Compilation for ARM:
To cross-compile the application for Arm, pc.arm file can be used, for example:

make -f arm.mk all

Similarly, other rules can also be specified instead of **all.**

Helper example:
To help you with the development of the Beagle Board version, we provide an example application that performs a sum of a vector. It uses the POOL and NOTIFY components of DSPLINK API. The functionality is simple as given below:

- The address of the pool and the size of the vector are sent to the DSP using NOTIFY component

- The vector is initialized on ARM.
- The cache is invalidated to ensure that the DSP reads the same data as written by ARM.
- The DSP is notified to start its execution.
- At the end of execution, the DSP will notify ARM with the results.

The application takes one command line parameter, the vector size (which has to be less than the memory available for the pool. See section regarding OMAP3530 in this document). An example run:

./pool_notify ./pool_notify.out 4096

Note that the size has to be a multiple of 128 (the allocation pool size has to be multiple of 128).
To execute the sum of all elements of the vector on the ARM processor, remove the "-DDSP" from the CFLAGS (gpp/Makefile).

Minimum Requirement:

The minimum requirement for this assignment is to have an ARM+DSP+NEON optimized implementation which runs at least 4x faster than the baseline version (the start and stop timers are marked in the main.cpp). Baseline version is the solution running on ARM alone with -O3 optimization level. To be more specific, we ask you to optimize the tracking part of the application by at least 16x. You are allowed to use the compiler optimization flags and other facilities available in the architecture without changing the algorithm and other initialization options. The speedup should be calculated as follows:

Speedup = (Execution Time Baseline) / (Execution Time Optimized)

Suggestions

In order to better understand the application we advise you to profile it. You can do this by using the gprof and MCProf tools. To enable a profile build you have to add the proper compiler and linker flags in the makefile, compile the application and run it as also shown in the makefile. You can find additional information on the web [3]. There are multiple possibilities of accelerating the application. We give some hints below:

- The DSP is not well suited for floating point operations. So, study the function to identify if floating point is needed, given the size of the types used (fixed point is much faster than floating point on DSP).
- There are independent computations that can be performed in parallel by the ARM and DSP.
- Load balancing and resource utilization are also important as well.
- The DSP compiler generated code can be influenced by the command line parameters of the compiler. Check what options are available and if/how they influence the generated code.

Additional information related to OMAP 3530

- Since it is possible for the DSP's instruction cache to remain active even after the completion of an application, and thus ignore the next application, it is recommended that you "flush" the cache by power cycling the DSP. This can be done by using the power management module. To use it, you have to issue (once) the command:

  insmod /home/root/lpm_omap3530.ko

  To power cycle the DSP use the commands:

  `/home/root/lpmOFF.xv5T`
  `/home/root/lpmON.xv5T`

  This can also be achieved by using `/home/root/powercycle.sh` script on beagleboard.

- The total memory available for a memory pool is 851968 bytes. Note that the DSP is not as well protected as the ARM to illegal writes, meaning that a write to a memory outside the allocated area can have unexpected results, such as breaking the signaling mechanism to the ARM processor.
- The stack size of the DSP processor is 4096 bytes. A stack overflow might have the same unexpected results as described above. Both these sizes can be changed by recompiling the DSPLINK driver, but for the purpose of the assignment, that is not needed.

References

1 http://beagleboard.org/
2 http://www.ti.com/product/omap3530
3 http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html
4 A. Yilmaz , O. Javed and M. Shah, "Object Tracking: A Survey," ACM Computing Surveys. 38, No. 4, pp.13-40(2006)
5 D. Comaniciu, V. Ramesh, and P. Meer, “Kernel-based object tracking,” IEEE Trans. Pattern Anal. Mach. Intell. , vol. 25, no. 5, pp. 564–575, Apr. 2003.
6 Ido Leichter , Michael Lindenbaum, Ehud Rivlin. "Mean Shift tracking with multiple reference color histograms," Computer Vision and Image Understanding. 114, pp. 400-408(2010).
7 Allen John G, Xu Richard Y D, J in Jesse S. "Object tracking using Camshift algorithm and multiple quantized feature spaces". Pan-Sydney Area Workshop on Visual Information Processing. Sydney, Australia, pp.1-5(2003).
8 http://opencv.org/