Optimizing the Embedded Platform using OpenCV

February 17, 2012
Matt Weber
(mlweber1@rockwellcollins.com)
• Goals
• Project Approach & Results
• Future Ideas
• References
Goals

• To quantify the effects of the many optimizations available and see what effect, if any power management has

• Most Important Requirements (MIRs)
  – Minimal startup and low latency processing time
  – On-demand Power Management

• Background
  – Utilized a OMAP3 processor for image processing
  – Linux 2.6.39.4 Kernel with OMAP PM patches
  – Buildroot w/ Crosstool-ng toolchain
Project Approach

• Cost/Benefit
  – Compiler → Co-Processor → Power Management → Specialized Cores
  – Supporting software (which kernel, packages, vendor libraries, etc)

• Define benchmarking tool

• Gather metrics for optimization methods applied to
  – Platform (Kernel/rootfs)
  – Application
  – With power management active
Project Approach: Compiler/Toolchain

• Gotchas
  – Are Binary compatibility & architecture (armv5, v6, v7a....) masking a problem?
  – Are your Platform & App using the same toolchain?
  – Are features like VFP (Vector Floating Point) & Advanced SIMD extension (aka NEON) enabled?

• Building your own has some additional benefits
  – Source control & ability to recreate/fix issues
  – Geared towards your CPU arch & hardware FPU
  – Could tailor kernel headers to get a newer feature
  – Possibly incorporate the latest Linaro GCC

Know your toolchain!
Project Approach: Benchmarking Tool

• OpenCv 2.1
  – cvMatchTemplate() algorithm as the test case
    
  cvMatchTemplate( img, tpl, res, CV_TM_CCORR_NORMED );
  – Lots of matrix math
  – Each of the time measurements were just for the algorithm execution and not the image load time
  – 5.5MB image is searched for the image of a small boat
Project Approach: Metrics Test #1

• **Test:** Compiler Optimization

• **Description:** Kernel and Rootfs are built with same flags below and executing off an SDCard.

• **Flags:**

  \[ CFLAGS += \text{-pipe \text{-O3}} \]

• **Result:** \[ \sim 19.35 \text{sec} @800\text{Mhz} \]
Project Approach: Metrics Test #2

- **Test:** Compiler Optimization & use of hardware co-processors
- **Description:** Kernel and Rootfs are built with same flags below and executing off an SDCard.
- **Flags:**
  
  ```
  CFLAGS += -pipe -O3 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp
  ```
- **Result:** \(\sim 4.91\text{sec} \at 800\text{Mhz}\)
  
  \(\sim 75\% \text{ increase in performance}\)
• **Test:** Compiler Optimization & Power Management

• **Description:** Kernel and Rootfs are built with same flags below. Power management is enabled to idle and frequency scale the CPU on-demand between 300 and 800Mhz. It uses the default scaling trigger threshold for the 2.6.39.4 kernel.
  
  (Note: Purely ARM core instructions.)

• **Flags:**
  
  -pipe -O3

• **Result:** \(~19.39\text{sec} \ @300-800\text{Mhz}\)

  \(~40\text{msec} (2\%)\) increase in processing time w/ PM

• **Comment:** Solely ARM instructions cause the scheduler to have more demand for a higher clock speed earlier, so it results in a small increase in the additional processing time required.
Project Approach: Metrics Test #4

• **Test:** Compiler Optimization, co-processors and Power Management

• **Description:** Kernel and Rootfs are built with same flags below. Power management is enabled to idle and frequency scale the CPU on-demand between 300 and 800Mhz. It uses the default scaling trigger threshold for the 2.6.39.4 kernel. (Note: ARM core and Neon instructions.)

• **Flags:**
  -pipe -O3 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp

• **Result:** \(~5.12\)sec @300-800Mhz
  \(~210\)msec (4%) increase in processing time w/ PM

• **Comment:** Less time spent executing ARM instructions, since the Neon core is offloading some of the processing, causes more execution at 300Mhz and a slight increase in processing time.
Project Approach: Future Tests

- Finish testing with DSP and TI Codec Engine
  - Initial tests with CMEM, LPM, DSPLINK, TI Codec Engine are working
  - Issues were found with the C6Accel used in SoC OpenCV DSP work (newer TI libraries, kernel and compiler issues.....)
  - TI measurements with Integra SOC (floating point DSP) show a 86% speed up for the match template algorithm
Project Approach: Performance Metric Summary

<table>
<thead>
<tr>
<th>Test</th>
<th>Result (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 -O3</td>
<td>19.35</td>
</tr>
<tr>
<td>#2 -O3 &amp; Neon</td>
<td>4.91</td>
</tr>
<tr>
<td>#3 -O3 w/ PM</td>
<td>19.39</td>
</tr>
<tr>
<td>#4 -O3 &amp; Neon w/PM</td>
<td>5.12</td>
</tr>
<tr>
<td>#5 -O3 &amp; Neon w/PM &amp; DSP</td>
<td>Est. ~3.07</td>
</tr>
</tbody>
</table>

The key to the next step is controlling offloading overhead
Project Approach: Power Management Test

- Tools → bench power-supply and data logging multimeter
- Startup board (power-supply is set to a 1A limit at 5V)
- First test is on-demand
  ```
  [root@buildroot ~]# echo "800000" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  [root@buildroot ~]# echo "ondemand" >/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  cpufreq-omap: transition: 800000 --> 300000
  [root@buildroot ~]# ./opencv_templatematch
  WORKING>>>
  cpufreq-omap: transition: 300000 --> 800000
  5.120000 seconds of processing
  t1: 320000   t2: 5600000
  Clockspersec: 1000000
  cpufreq-omap: transition: 800000 --> 300000
  [root@buildroot ~]#
  ```
- Second test is userspace set frequency
  ```
  [root@buildroot ~]# echo "userspace" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  [root@buildroot ~]# echo "800000" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
  cpufreq-omap: transition: 300000 --> 800000
  [root@buildroot ~]# ./opencv_templatematch
  WORKING>>>
  4.910000 seconds of processing
  t1: 110000   t2: 5020000
  Clockspersec: 1000000
  [root@buildroot ~]#
  ```
• Note: the DSP adds an additional \(\sim 375\) mW, shown in yellow & prevents the ARM from scaling up to 800Mhz. The chart shows only an estimate of DSP power draw\(^5\) and an approximate timeline from TI whitepaper findings.

• If an OMAP GPU options was added, the approx power draw would increase by \(\sim 93\) mW. We're not sure yet how much overhead this would cause on the ARM...
Future Ideas

• Investigate the new issues of Power Management in a multi-core world
  - How could load statistics be maintained for dynamic power control across cores?
  - Maybe add hooks into existing CPUFreq framework for on-demand based on anticipated completion from other cores? What if Linux on the primary CPU(s) suspended while the offloaded task is being processed?
Future Ideas

• GsoC project: OpenCV DSP Acceleration (2010)
  - Investigate OpenCV code issues (lots of floating point and STL)
  - Gather power, timing and latency/IPC overhead numbers using the TI Codec Engine approach
  - Possibly implement custom DSP approach based on results

• GPU
  - Investigate (future) SGX Graphics SDK with OpenCL support
  - Currently the only published vendor supporting OpenCL is ZiiLABS (ZMS SOC) and TI (OMAP5)
Project Information

- **Hardware**
  - BeagleboardXM
  - (optional) LI-5M03 camera

- **Repository & Wiki**
  - includes xloader, uboot, sdcard scripts, kernel & rootfs, test sequences
    - git://github.com/matthew-l-weber/buildroot.git
    - https://github.com/matthew-l-weber/buildroot/wiki

- **Buildroot Overview**
Credits/References