Android Platform Optimizations

ELC-Europe
Prague, October 2011
Ruud Derwig
Helping Design the Chips Inside

Mobile Multimedia

Digital Home

Data Center & Networking

Computing & Peripherals

Medical

Automotive

Industrial

Military / Aerospace

Other
## Agenda

- **Market & value drivers**
- **What to optimize?**
- **How to optimize?**
- **Results & conclusion**
Android Markets

- Smartphones
- Tablets
- TV
- STB / multimedia

- Others / new
Android Markets

- Smartphones
- Tablets
- TV
- STB / multimedia
- Others / new
Android Markets

- Smartphones
- Tablets
- TV
- STB / multimedia
- Others / new
Key Value Drivers & System Architecture Choices

- Power consumption
  → optimize performance / mW

- Product cost
  → optimize performance / area
  → optimize development efficiency

- Hardware – Software trade-offs
  - Maximum flexibility & developer efficiency: “virtual everything”
    - PC model, multi-GHz SMP processor centric designs
  - Minimal power & optimal performance: “dedicated hardware”
    - dedicated, fixed function device
  - Sweetspot: “heterogeneous, HW accelerated multi-core”
    - Mix of CPU, DSP, and dedicated HW
Market & value drivers

What to optimize?

How to optimize?

Results & conclusion

© Synopsys 2011
Linux Kernel & Library Optimizations

• Important,
• … but not Android specific

• Optimization options
  – Optimize hotspots
    • compiler
    • handwritten assembler
  – CPU hardware optimizations
    • MMU
    • special instructions
Dalvik Virtual Machine

• “Java” * virtual machine
  – Register-based architecture (Java VMs are stack machines). Dalvik registers are typically stored in memory (on the stack, like local variables in C).
  – Own bytecode

• Three virtual machines
  – Portable: completely C-based, in fact one large switch{} statement with a case x: for every Dalvik opcode.
  – Fast (a.k.a. MTERP): assembly-coded handlers for every Dalvik opcode, which are aligned on 64 bytes addresses, so that the address of the handler can be easily calculated from the opcode, saving a lookup.
  – JIT: just-in-time compiler, initially starts as fast/mterp interpreter, but will identify ‘hot’ traces and pass these to the compiler thread.

*Dalvik is a clean-room implementation of Java for copyright reasons. The syntax is similar.*
Android Media Player Architecture

JAVA
- Media Player App
  - Media Player App Framework

Linux User Space
- Media Player Service
  - AudioFlinger
    - Other Audio Driver
    - Alsa Kernel Driver

Linux Kernel Space

Applications
- Home
- Contacts
- Phone
- Browser
- ...

Application Framework
- Activity Manager
- Window Manager
- Content Providers
- View System
- Package Manager
- Telephony Manager
- Resource Manager
- Location Manager
- Notification Manager

Libraries
- Surface Manager
- Media Framework
- SQLite
- Core Libraries
- OpenGL ES
- FreeType
- WebKit
- Dalvik Virtual Machine
- SQL
- SSL
- lib

Android Runtime
- Core Libraries
- Dalvik Virtual Machine

Linux Kernel
- Display Driver
- Camera Driver
- Flash Memory Driver
- Binder (IPC) Driver
- Keypad Driver
- WiFi Driver
- Audio Drivers
- Power Management
Android Media Player Architecture

- Google’s player of choice is the Stagefright
  - multi-format A/V player, newly developed for Android
  - Simple fixed graph – selects demuxer and decoder based on file extension
- Alternatives exist
  - Gstreamer based
  - proprietary / legacy

Media Player Service

StageFright Player
GStreamer Player

OpenCore Player
MIDI Player
Vorbis Player

Video Decoding
Video Rendering (➔ SurfaceFlinger)

Audio Decoding
Audio Rendering (➔ AudioFlinger)
Audio Optimization Option: off-load audio processing to DSP

GP-CPU
Audio DSP
Android Graphics - Architecture

• 2D
  – Canvas/Skia
  – OpenVG

• 3D
  – OpenGL-ES 1.x
  – OpenGL-ES 2

• Renderscript
  – Expose native GPU/SMP to (portable) applications
  – C99 -> LLVM intermediate bitcode -> machine code
Graphics Optimization Options

• Graphics drawing/rendering
  – Software/assembler optimization
    • Skia, PixelFlinger
  – Hardware acceleration
    • GPU (OpenGL-ES 2)
    • 2D accelerator (OpenVG compatible or other)
    • Memory architecture, caching
  – Renderscript

• Surface Composition
  – Scaling, colorspace conversion
    • Custom instructions
    • GPU
    • Dedicated hardware acceleration (bitblit)
Agenda

Market & value drivers

What to optimize?

How to optimize?

Results & conclusion
Optimized Designware ARC Android

- Full port of the Android Froyo/Gingerbread release to the ARC processor architecture and build environment
- Including NDK and SDK to support Android application building/porting
- Google/OHA Compatibility Test Suite tested
# Differences between VM Implementations

<table>
<thead>
<tr>
<th>Portable</th>
<th>MTerp</th>
<th>JIT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>switch (opcode) {</code>&lt;br&gt;<code>  case add: a = b + c;</code>&lt;br&gt;<code>   break;</code>&lt;br&gt;<code>  case sub: a = b - c;</code>&lt;br&gt;<code>   break;</code>&lt;br&gt;<code>  ...</code>&lt;br&gt;<code>}</code></td>
<td><code>ld r0, [b]</code>&lt;br&gt;<code>  ld r1, [c]</code>&lt;br&gt;<code>  add r0, r0, r1</code>&lt;br&gt;<code>  st r0, [a]</code>&lt;br&gt;<code>  ld r0, [next_opcode]</code>&lt;br&gt;<code>  asl r0, r0, 6</code>&lt;br&gt;<code>  add r0, r13, r0</code>&lt;br&gt;<code>  j [r0]</code></td>
<td><code>ld r0, [b]</code>&lt;br&gt;<code>  ld r1, [c]</code>&lt;br&gt;<code>  add r0, r0, r1</code>&lt;br&gt;<code>  st r0, [a]</code>&lt;br&gt;<code>  </code>OR<code>&lt;br&gt;</code>  add r20, r20, r21`</td>
</tr>
</tbody>
</table>

```
ld r0, [next_opcode]
<pipeline stall>
ld.as r1, [jump_table, r0]
<pipeline stall>
j [r1]
```
Register- and Stack-based VMs

*Example:* \( a = b + c \)

<table>
<thead>
<tr>
<th>Java</th>
<th>Dalvik</th>
<th>Dalvik for ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>iload b</td>
<td>add-int a, b, c</td>
<td>add-int a, b, c</td>
</tr>
<tr>
<td>iload c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istore a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld r0, [b]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld r0, [c]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r0, r0, r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>st r0, [a]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Registers are only saved/restored when changing stack frames or when moving to interpreter.
Audio Processing on DSP

- Audio decoding and Post-processing off-loaded to ARC Sound Processor
- Special host Audio Decoder implementation that takes care of off-loading with standard host decoder interfaces, so seamless integration
- Post-processing control through Renderer on host (special Renderer or Renderer plug-in component)
- MSF = Media Streaming Framework
  ARC DSP optimized, lightweight streaming framework
- MQX = Real-time Operating system
- RPC/IPC = Remote Procedure call / Inter Processor Communication
Android & Audio APIs

- Stagefright supports 2 types of interfaces
  - OpenMax-IL: for re-use of OMX components
  - Stagefright codec interface: for native Stagefright codecs

- AudioFlinger uses dedicated interfaces
  - standard implementation using "ALSA" exist
  - developments ongoing (?) to support OpenSL-ES Khronos standard (like OMX)

- SNPS API choice not yet made
  - OMX-IL pro: open standard
  - OMX-IL con: efficiency, complexity: standard by committee…
  - Stagefright pro: efficient integration with Stagefright
  - Stagefright con: not an open standard, no deep tunneling
Alternative: Gstreamer

- GStreamer Android Player
  - see e.g. ELC-E 2010 presentation

- “The goal of the project is to both allow hardware makers to standardize on GStreamer across their software platforms, but also to make the advanced functionality of GStreamer available on the Android platform, like video editing, DLNA Support and Video conferencing.”
GStreamer DSP Off-loading with “Deep Tunneling”

- Gstreamer-MSF integration makes heterogeneous multi-core SW development transparent to user

- Instantiation of Gstreamer element → instantiation of module on one of the ARC cores

- Creation of link → local connection or core-crossing connection between modules
Gstreamer Deep Tunneling

```c
static void connect_msf_outpin ( GstPad* pad )
{
    GstPad  *peerpad = gst_pad_get_peer ( pad );
    GstElement  *element = gst_pad_get_parent_element ( pad );
    GstElement  *peerelement = gst_pad_get_parent_element ( peerpad );
    GstAudioModule  *filter = GST_AUDIO_MODULE ( element );
    guint32  
    result;

    if ( !pad_is_deeptunnel ( pad ))
    {
        /* not a deep tunnel */
        /* create sink module */
        msf_api_sink_module_create ( filter->msf_coreid, "Sink module", output_fifo_buffer,
                                   sink_pv_data, sizeof ( sink_pv_data ), &sink_module_id ));
        msf_api_connect_pins ( filter->msf_moduleid, sink_module_id, 0, 0 ));
    }
    else
    {
        if ( pad_is_corecrossing ( pad ))
        {
            /* deep tunnel AND core-crossing */
            /* create sink module */
            msf_api_sink_module_create ( filter->msf_coreid, "Sink module", filter->msf_sharedfifo,
                                       sink_pv_data, sizeof ( sink_pv_data ), &sink_module_id ));
            msf_api_connect_pins ( filter->msf_moduleid, sink_module_id, 0, 0 ));
        }
        else
        {
            /* deep-tunnel AND no core-crossing */
            guint32  peer_module_id;

            /* get the module id of the peer MSF module */
            g_object_get ( G_OBJECT ( peerelement ), "msf_moduleid", &peer_module_id, NULL);
            msf_api_connect_pins ( filter->msf_moduleid, peer_module_id, 0, 0 ));
        }
    }
}
```
ARC HW Extensions

ARC EIA (Extension Interface Automation):
• supports user defined custom instructions
• accelerates typical Dalvik (Java VM) and pixelflinger (2D GUI) instruction sequences

Prefetcher:
• Eliminates pipeline stalls in high latency memory environments
• L2$ not required in this case
• Configurable depending on application

CPU area, excluding memories

© Synopsys 2011
Leveraging the ARC EIA Capabilities

Example: Colour Space Conversion

8 operations are required for a conversion from ABGR8888 to RGB565.

This can be combined into one single EIA instruction.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld.ab</td>
<td>r1, [r4, 0x4]</td>
</tr>
<tr>
<td>and</td>
<td>r2, r1, 0x8</td>
</tr>
<tr>
<td>asl</td>
<td>r2, r2, 8</td>
</tr>
<tr>
<td>and</td>
<td>r3, r1, 0xfc00</td>
</tr>
<tr>
<td>lsl</td>
<td>r11, r3, 5</td>
</tr>
<tr>
<td>or</td>
<td>r2, r2, r11</td>
</tr>
<tr>
<td>and</td>
<td>r3, r1, 0xf8000</td>
</tr>
<tr>
<td>lsl</td>
<td>r11, r3, 19</td>
</tr>
<tr>
<td>or</td>
<td>r2, r2, r11</td>
</tr>
<tr>
<td>stw.ab</td>
<td>r2, [r5, 0x2]</td>
</tr>
</tbody>
</table>

Instruction | Operands       |
-------------|----------------|
| ld.ab       | r1, [r4, 0x4]  |
| upk8        | r2, r1, r6     |
| stw.ab      | r2, [r5, 0x2]  |
Optimizing Dalvik VM

Android VM Performance (CaffeineMark Benchmark)
(measured on production silicon @ 600MHz)

<table>
<thead>
<tr>
<th>Date</th>
<th>CM</th>
<th>CM/MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct'10</td>
<td>535</td>
<td>0.89</td>
</tr>
<tr>
<td>Nov'10</td>
<td>1358</td>
<td>2.26</td>
</tr>
<tr>
<td>Feb'11</td>
<td>2400</td>
<td>4.00</td>
</tr>
<tr>
<td>July'11</td>
<td>2700</td>
<td>4.50</td>
</tr>
<tr>
<td>Aug'11</td>
<td>3071</td>
<td>5.12</td>
</tr>
<tr>
<td>latest</td>
<td>3296</td>
<td>5.49</td>
</tr>
</tbody>
</table>
Optimizing Dalvik VM

Dalvik JIT Optimization Results
Relative Performance Increase compared to Interpreter

~20%

x 4.9 … and going

Pixelflinger Optimization Results
Relative Performance Increase resulting from Pixelflinger JIT

x 11.3 … and going

Core Mark | Caffeine Mark | Without L2 cache
---|---|---
1,9 | 4,9 | /MHz
37 | 90 | /mW
14 | 35 | /MHz/mm²

measurements are done on 50MHz FPGA
results are without performance gains from hardware extensions
Optimizing Hardware
Custom Instructions & Prefetching

![Bar chart showing execution time (ARC cycles) vs. memory latency (cycles) for different code configurations with and without prefetching.]
Linux kernel + ARC HW optimizations

LMBench measured Latency improvements of Linux Optimized ARC750D upgrade versus current silicon solution
Conclusions

• There are more markets for Android than high-end smartphone
• There are more optimizations possible than relying on Moore’s law for GHz multi-cores

• Optimize performance / mW & performance / area
• Sweetspot: “heterogeneous, HW accelerated multi-core”
  – Mix of CPU, DSP, and dedicated HW
  – Highly optimized platform infrastructure SW hides heterogeneous complexities

• ‘Simple’ ARC processor with SW optimized Dalvik VM performs equal or better as others, thanks to careful SW optimizations, and the use of simple HW acceleration
  – Custom instructions tailored for specific tasks
  – Prefetcher iso. general purpose 2nd level cache
  – DSP more efficient in audio processing than CPU
Fast Forward to Predictable Success