Accelerate Android and Linux Applications and Effect Optimization

Ken Chen
Staff Applications Engineer
Problem: THIS IS not a desktop

- Mobile apps require special design considerations that aren’t always clear and tools to solve increasingly complex systems are limited
  - Animations and games drop frames
  - Networking, display, real time audio and video processing eat battery
  - App won’t fit in memory constraints
Analysis

- Fortunately Google, ARM® and many others are developing analysis tools and solutions to these problems

- Is my app … ?
  - CPU/GPGPU bound
  - I/O or memory constrained
  - Power efficient

- What can I do to fix it?
  (short of buying everyone who runs my app a dual Cortex®-A15 & Mali™-T604 or Octo phone)
Analysis of Java SDK android apps

- Static analysis with SDK Lint tool
- Dynamic analysis with DDMS
  - Allocation/heap
  - Process and thread utilization
  - Traceview (method)
  - Network
- Hierarchy Viewer
- Systrace
But ask yourself these questions

- Is this performance bottleneck parallelizable?
- Is this Java or Native? Would it be better the other way around?
- Has this been done before? Don’t reinvent the wheel.
- Am I being smart with resources?
- What version of Android should I target?
Starting EASY
Static analysis: LINT

Avoid object allocations during draw/layout operations (preallocate and reuse instead)

This TableLayout should use android:layout_height="wrap_content"

SharedPreferences.edit() without a corresponding commit() or apply()

This tag and its children can be replaced by one <TextView/>

Avoid hardcoding the debug mode; leaving it out allows debug activities to

Avoid hardcoding the debug mode; leaving it out allows debug activities to

Avoid hardcoding the debug mode; leaving it out allows debug activities to

Avoid hardcoding the debug mode; leaving it out allows debug activities to

Replace "..." with ellipsis character (...) &nb#8230;? (32 items)

The image icon.png varies significantly in its density-independent

I18N] Hardcoded string "Push me", should use @string resource

This Handler class should be static or leaks might occur (name.b

Use new SparseIntArray(...) instead for better performance

Avoid object allocations during draw/layout operations (preallocate and reuse instead)

Issue: Looks for memory allocations within drawing code

Id: DrawAllocation

You should avoid allocating objects during a drawing or layout operation. These are called frequently, so a smooth UI can be interrupted by garbage collection pauses caused by the object allocations.

The way this is generally handled is to allocate the needed objects up front and to reuse them for each drawing operation.

Some methods allocate memory on your behalf (such as Bitmap.create), and these should be handled in the same way.
## Static analysis: LINT

<table>
<thead>
<tr>
<th>Description</th>
<th>Category</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obsolete ProGuard file; use -keepclass(es)withmembers instead of -k</td>
<td>Correctness</td>
<td>Use android.util.FloatMath#cos() instead of java.lang.Math#cos to avoid argument float to double conversion</td>
</tr>
<tr>
<td>&quot;backup_api_key&quot; is not translated in de, zh-CN, zh-TW</td>
<td>Correctness:Message</td>
<td>SmallK</td>
</tr>
<tr>
<td>Call requires API level 11 (current min is 3): android.app.Activity#</td>
<td>Correctness:Message</td>
<td>SmallK</td>
</tr>
<tr>
<td>Avoid object allocations during draw/layout operations (preallocate)</td>
<td>Performance</td>
<td>Issue: Suggests replacing choose java.lang.Math calls with GameActivity</td>
</tr>
<tr>
<td>This TableLayout should use android:layout_height=&quot;wrap_content&quot;</td>
<td>Correctness</td>
<td>Android</td>
</tr>
<tr>
<td>SharedPreferences.edit() without a corresponding commit() or apply</td>
<td>Correctness</td>
<td>Android</td>
</tr>
<tr>
<td>This tag and its children can be replaced by one &lt;TextView/&gt; and a</td>
<td>Performance</td>
<td>Android</td>
</tr>
<tr>
<td>Avoid hardcasing the debug mode; leaving it out allows debug and</td>
<td>Security</td>
<td>Android</td>
</tr>
<tr>
<td>Avoid hardcoding the debug mode; leaving it out allows debug and</td>
<td>Security</td>
<td>Android</td>
</tr>
<tr>
<td>Avoid hardcoding the debug mode; leaving it out allows debug and</td>
<td>Security</td>
<td>Android</td>
</tr>
<tr>
<td>Replace &quot;...&quot; with ellipsis character (... , &amp;h2300;) ? (32 Items)</td>
<td>Usability:Typography</td>
<td>strings</td>
</tr>
<tr>
<td>The image icon.png varies significantly in its density-independent (65 Items)</td>
<td>Usability:Icons</td>
<td>strings</td>
</tr>
<tr>
<td>[18N] Hardcased string &quot;Push me&quot;, should use @string resource</td>
<td>Internationalization</td>
<td>main.xml</td>
</tr>
<tr>
<td>This Handler class should be static or leaks might occur (name.boyle)</td>
<td>Performance</td>
<td>SGTPU</td>
</tr>
<tr>
<td>Use new SparseInt(&lt;string&gt;...) instead for better performance</td>
<td>Performance</td>
<td>SoundCloud</td>
</tr>
<tr>
<td>Missing the following drawables in drawable-xhdpi: gltron_bitmap.png</td>
<td>Performance</td>
<td>Hello World</td>
</tr>
</tbody>
</table>

On modern hardware, "double" is just as fast as "float" though of course it takes more memory. However, if you are using floats and you need to compute the sine, cosine or square root, then it is better to use the android.util.FloatMath class instead of java.lang.Math since you can call methods written to operate on floats, so you avoid conversions back and forth to double.

http://developer.android.com/guide/practices/design/performance.html#avoidfloat
Beyond static analysis
Dalvik Debug Monitor Server (DDMS)

- DDMS Thread analysis (like “top” but better)

<table>
<thead>
<tr>
<th>ID</th>
<th>Tid</th>
<th>Status</th>
<th>utime</th>
<th>stime</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15135</td>
<td>native</td>
<td>8</td>
<td>7</td>
<td>main</td>
</tr>
<tr>
<td>2</td>
<td>15139</td>
<td>vmwait</td>
<td>898</td>
<td>107</td>
<td>GC</td>
</tr>
<tr>
<td>3</td>
<td>15140</td>
<td>vmwait</td>
<td>0</td>
<td>0</td>
<td>Signal Catcher</td>
</tr>
<tr>
<td>4</td>
<td>15141</td>
<td>running</td>
<td>16</td>
<td>35</td>
<td>JDWP</td>
</tr>
<tr>
<td>5</td>
<td>15142</td>
<td>vmwait</td>
<td>40</td>
<td>30</td>
<td>Compiler</td>
</tr>
<tr>
<td>6</td>
<td>15148</td>
<td>wait</td>
<td>1</td>
<td>1</td>
<td>ReferenceQueueDaemon</td>
</tr>
<tr>
<td>7</td>
<td>15149</td>
<td>wait</td>
<td>11</td>
<td>3</td>
<td>FinalizerDaemon</td>
</tr>
<tr>
<td>8</td>
<td>15150</td>
<td>wait</td>
<td>0</td>
<td>0</td>
<td>FinalizerWatchdogDaemon</td>
</tr>
<tr>
<td>9</td>
<td>15151</td>
<td>native</td>
<td>0</td>
<td>0</td>
<td>Binder_1</td>
</tr>
<tr>
<td>10</td>
<td>15152</td>
<td>native</td>
<td>0</td>
<td>0</td>
<td>Binder_2</td>
</tr>
<tr>
<td>11</td>
<td>15153</td>
<td>wait</td>
<td>17791</td>
<td>1150</td>
<td>GLThread 1989</td>
</tr>
<tr>
<td>12</td>
<td>15159</td>
<td>native</td>
<td>0</td>
<td>0</td>
<td>SoundPool</td>
</tr>
</tbody>
</table>
DDMS: Traceview
How much CPU time is each method consuming?

- Traceview (start method profiling button)
Allocations and HEAP
are you allocating in a high frequency method?

<table>
<thead>
<tr>
<th>Alloc Order</th>
<th>Allocation Size</th>
<th>Allocated Class</th>
<th>Thread</th>
<th>Allocation Class</th>
<th>Allocated in</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>28</td>
<td>short[]</td>
<td>11</td>
<td>com.gltTron.Game.Vector</td>
<td>storeIndices</td>
</tr>
<tr>
<td>510</td>
<td>28</td>
<td>short[]</td>
<td>11</td>
<td>com.gltTron.Game.Vector</td>
<td>storeIndices</td>
</tr>
<tr>
<td>508</td>
<td>28</td>
<td>float[]</td>
<td>11</td>
<td>com.gltTron.Game.Vector</td>
<td>&lt;init&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>File</th>
<th>Line</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.gltTron.Game.Vector.Trail</td>
<td>&lt;init&gt;</td>
<td>TrailMesh.java</td>
<td>84</td>
<td>false</td>
</tr>
<tr>
<td>com.gltTron.Game.Play</td>
<td>drawTrails</td>
<td>Player.java</td>
<td>393</td>
<td>false</td>
</tr>
<tr>
<td>com.gltTron.Game.GLTron</td>
<td>RenderGame</td>
<td>GLTronGame.java</td>
<td>579</td>
<td>false</td>
</tr>
<tr>
<td>com.gltTron.Game.GLTron</td>
<td>RunGame</td>
<td>GLTronGame.java</td>
<td>428</td>
<td>false</td>
</tr>
<tr>
<td>com.gltTron.OpenGLRenderer</td>
<td>onDrawFrame</td>
<td>OpenGLRenderer.java</td>
<td>104</td>
<td>false</td>
</tr>
</tbody>
</table>
HEAP:
Is your app running out of memory?

Heap updates will happen after every GC for this client

<table>
<thead>
<tr>
<th>ID</th>
<th>Heap Size</th>
<th>Allocated</th>
<th>Free</th>
<th>% Used</th>
<th># Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21.379 MB</td>
<td>3.594 MB</td>
<td>17.785 MB</td>
<td>16.81%</td>
<td>39,583</td>
</tr>
</tbody>
</table>

Display: Stats

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Total Size</th>
<th>Smallest</th>
<th>Largest</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>4,360</td>
<td>17.780 MB</td>
<td>16 B</td>
<td>3.876 MB</td>
<td>192 B</td>
</tr>
<tr>
<td>data object</td>
<td>23,431</td>
<td>691.312 KB</td>
<td>16 B</td>
<td>584 B</td>
<td>32 B</td>
</tr>
<tr>
<td>class object</td>
<td>2,631</td>
<td>777.367 KB</td>
<td>168 B</td>
<td>39.117 KB</td>
<td>168 B</td>
</tr>
<tr>
<td>1-byte array (byte[], boolean[])</td>
<td>205</td>
<td>1.228 MB</td>
<td>24 B</td>
<td>1.000 MB</td>
<td>40 B</td>
</tr>
<tr>
<td>2-byte array (short[], char[])</td>
<td>8,936</td>
<td>548.125 KB</td>
<td>24 B</td>
<td>28.023 KB</td>
<td>48 B</td>
</tr>
<tr>
<td>4-byte array (object[], int[], float[])</td>
<td>4,359</td>
<td>403.461 KB</td>
<td>24 B</td>
<td>40.453 KB</td>
<td>40 B</td>
</tr>
<tr>
<td>8-byte array (long[], double[])</td>
<td>21</td>
<td>2.523 KB</td>
<td>24 B</td>
<td>248 B</td>
<td>136 B</td>
</tr>
<tr>
<td>non-Java object</td>
<td>124</td>
<td>5.289 KB</td>
<td>16 B</td>
<td>464 B</td>
<td>32 B</td>
</tr>
</tbody>
</table>
Network statistics

- Save battery, look for short spikes that can be delayed
- TrafficStats API allows you to tag individual sockets
Adb shell DUMPsys

- With dumpsys you can check:
  - Event Hub State
  - Input Reader State
  - Input Dispatcher State
  - any number of other systems e.g. dumpsys gfxinfo
Dumpsys gfxinfo

- Drop dumpsys data columns in to a spreadsheet and visualize...
e.g. Will my animation drop frames?
Systrace

- I’ve done all I can do to analyze inside my app but still can’t find the bottleneck.

Systrace to the rescue!

- Systrace.py will generate a 5 second system level snapshot

```
android@android ~/android-sdk-linux/tools/systrace $ ./systrace.py
capturing trace... done
downloading trace... done
wrote file:///home/android/android-sdk-linux/tools/systrace/trace.html

android@android ~/android-sdk-linux/tools/systrace $ xdg-open trace.html
```
Systrace

html5 page of info: Navigate with ‘w’a’s’d’
Analyzing native C/C++ (NDK)

- But I didn’t use the Java SDK to write my app! How do I analyze my already wicked fast native (or iOS app objective-C port) code?
- What about the Linux kernel part of system analysis?
- Notes of caution
  - Applications that use NDK well will be faster and slicker
  - Ones that don’t will be cursed by unhappy users
  - If you build .so libraries for ARM, only ARM devices will be able to run your apps
  - Fortunately not many Android Platforms that aren’t ARM
  - Good use of the NDK will narrow the difference between high and low end devices
  - Moving inefficient code to Native doesn’t magically make it better code
ARM DS-5™ Community Edition
Free android Native analyzer and debugger

DS-5 Eclipse

- DS-5 Debugger [C/C++]
- Android Debugger ADT Plugin [Java]
- adb tool

HOST

USB / Ethernet

ADB daemon

Android / Linux Kernel

gator.ko

JTAG

VM Process

- Dalvik VM Application
- Java Debug Support
- Dalvik VM
- Native Libraries

gatord

Android TARGET
DS-5 CE for Android App Developers

- Friendly, Reliable App Debugger
  - Powerful graphical user interface
  - ADB integration for native debug
  - Java* and native debug in the same IDE

- System-wide Performance Analyzer
  - In-depth system performance statistics
  - Process to function level profiling (native)

- Integrated, validated solution
  - Comprehensive documentation
  - Support via ARM forums
  - Delivered as Eclipse plug-in on arm.com
  - Free of charge

* Java debug for Android requires SDK and ADT
Streamline: The Big Picture

- Find hotspots, system glitches, critical conditions at a glance

Select from 40+ CPU counters, OS level and custom metrics

Select one or more processes to visualize their instant load on CPU

Accumulate counters, measure time and find instant hotspots

Combined task switch trace and sampled profile for all threads
Mali GPU Graphics Analysis

- CPU, and GPU fragment and vertex processing activity
- OpenGL® ES API events
- Frame buffer filmstrip
- Hardware and Software counters
- Visualize application activity per processor or processor activity per application
Drilldown Software Profiling

Quickly identify instant hotspots

Filter timeline data to generate focused software profile reports

Click on the function name to go to source code level profile
Bandwidth Vertex Fragment CPU

- CPU doing too much and stalling GPU?
- Vertex shader operating on too many vertices stalls fragment shader and CPU?
- Fragment shader trying to do a lot of fancy effects stalling CPU and vertex shader?
Fragment Bound

- **Overdraw**
  - This is when you draw to each pixel on the screen more than once
  - Drawing your objects front to back instead of back to front reduces overdraw
  - Also limiting the amount of transparency in the scene can help

- **Resolution too high or too many effects or cycles in shader**
  - Every effect that you add to your scene. Every light that you add will add to the number of cycles your shader will take
  - If you decide to run your app at native resolution be careful

Nexus 10 Native Resolution = 2560 x 1600 = 4,096,000 pixels

Quad Core GPU 533Mhz = 520 Cycles per pixel Approx.

Targeting 30 FPS = 17 Cycles in your shader
Fragment Bound Streamline

- Involves just 1 counter and the frequency of the GPU
  - Job Slot 0 Active

Fragment Percentage = \( \frac{\text{Job Slot 0 active}}{\text{Frequency}} \times 100 \)

Fragment Percentage = 84%

Overdraw = \( \frac{\text{Fragment Threads Started} \times \text{Number of Cores}}{\text{Resolution} \times \text{FPS}} \)

Overdraw = 3.9
Vertex Bound

- Too many vertices in geometry
  - Get your artist to remove unnecessary vertices
    - A lot of artists still generate content for high end desktop content
    - Impose some budgeting and limits
  - Use LOD Switching
    - Only objects that take up a lot of screen space need to be in high detail
    - Objects that are further away don’t need the same level of detail
  - Use culling

- Too many cycles in the vertex shader
  - You only have a limited amount of cycles to do your vertex shading
  - The amount of cycles you can afford to spend on vertex shading is directly dependent on the number of vertices
Vertex Bound Streamline

- Involves just 1 counter and the frequency of the GPU
  - Job Slot 1 Active

Vertex Percentage = \( \left( \frac{\text{Job Slot 1 active}}{\text{Frequency}} \right) \times 100 \)

Vertex Percentage = 13%

Load Store CPI = Full Pipeline issues / Load Store Instruction Words Completed

Load Store CPI = 2.02
Bandwidth Bound

- When creating embedded graphics applications, Bandwidth is a scarce resource
  - A typical embedded device can handle \( \approx 5.0 \) Gigabytes a second of bandwidth
  - A typical desktop GPU can do in excess of 100 Gigabytes a second
- One way to reduce bandwidth is to use texture compression
  - The main popular format is ETC Texture Compression
  - This can help reduce your 32 bits per pixel texture into a 4 bits per pixel texture
  - Mali Texture Compression Tool can help convert your textures for you
- Another way to reduce bandwidth is to use 16 bit textures instead of 32
  - You won’t often notice the difference
Bandwidth Bound Streamline

- Involves just 2 Streamline Counters
  - External Bus Read Beats
  - External Bus Write Beats

\[ \text{Bandwidth in Bytes} = (\text{External Bus Read Beats} + \text{External Bus Write Beats}) \times \text{Bus Width} \]

\[ \text{Bandwidth} = 967 \text{ MB/S} \]

Texture Pipeline CPI = Threads in Loop 2 / Texturing Pipeline instruction words completed

Texture Pipeline CPI = 1.55
## CPU Bound

- Sometimes a slow frame rate can actually be a CPU issue and not a GPU one
  - In this case optimizing your graphics won’t achieve anything
- Most mobile devices have more than one core these days Are you threading your application as much as possible?
- Mali GPU is a deferred architecture
  - Reduce the amount of draw calls you make
  - Try to combine your draw calls together
- Offload some of the work to the GPU
  - Even easier with Mali-T604 supporting OpenCL Full Profile
CPU Bound Streamline

- Easy just look at the CPU Activity
  - Remember to look at all the cores.

Some of the area is greyed out due to Streamline’s ability to present per App CPU activity.
Mali Offline Shader Compiler

- Command-line interface: Easy integration into regression build and test systems
- Offline compilation of GLSL ES vertex & fragment shaders to Mali GPU binary
- Detailed output of shader performance
- Available on malideveloper.arm.com

```
C:\Program Files (x86)\ARM\Mali Developer Tools\Mali Offline Shader Compiler v4.0\bin\malisc.exe -v --frag --core=Mali-T600 "C:\Documents\Presentations\Own\gc\Example_FresnelFp.glsles.OLD"
0 error(s), 0 warning(s)
2 work registers used, 1 uniform registers used
Pipelines: A / L / T / Overall
Number of instruction words emitted: 10 + 3 + 3 = 16
Number of cycles for shortest code path: 4.5 / 3 / 3 = 4.5 (A bound)
Number of cycles for longest code path: 4.5 / 3 / 3 = 4.5 (A bound)
Note: The cycle counts do not include possible stalls due to cache misses.
```
Example Output of Compiler

```c
// Fragment program for distorting a texture using a 3D noise texture
void main()
{
    // Do the tex projection manually so we can distort _after_
    vec2 final = projectionCoord.xy / projectionCoord.w;

    // Noise
    vec3 noiseNormal = (texture2D(noiseMap, (noiseCoord.xy / 5.0).rgb - 0.5).rgb * noiseScale;
    final += noiseNormal.xz;

    // Fresnel
    normal = normalize(normal + noiseNormal.xz);
    float fresnel = fresnelBias + fresnelScale * pow(1.0 + dot(eyeDir, oNormal), fresnelPower);

    // Reflection / refraction
    vec4 reflectionColour = texture2D(reflectMap, final);
    vec4 refractionColour = texture2D(refractMap, final) + tintColour;

    // Final colour
    gl_FragColor = mix(refractionColour, reflectionColour, fresnel);
}
```

C:\Program Files (x86)\ARM\Mali Developer Tools\Mali Offline Shader Compiler v4.0\bin>malisc.exe -v --frag --core=Mali-T600 "C:\Documents\Presentations\Own\get\Example_FresnelFp.glsles.New"
0 error(s), 0 warning(s)

2 work registers used, 2 uniform registers used

Pipelines: A / L / T / Overall
Number of instruction words emitted: 7 + 3 + 3 = 13
Number of cycles for shortest code path: 3 / 4 / 3 = 4 (L bound)
Number of cycles for longest code path: 3 / 4 / 3 = 4 (L bound)
Note: The cycle counts do not include possible stalls due to cache misses.
Vertex Buffer Objects

- Using Vertex Buffer Objects (VBO’s) can save you a lot of time in overhead
- Every frame in your application all of your vertices and colour information will get sent to the GPU
- A lot of the time these won’t change. So there is no need to keep sending them
- Would be a much better idea to cache the data in graphics memory
- This is where VBO’s can be useful

```c
glGenBuffers(1, VertexVBOID);
glBindBuffer(GL_ARRAY_BUFFER, VertexVBOID);
glBufferData(GL_ARRAY_BUFFER, (sizeof(GLFloat)*3)* numVert, &pvertex[0], GL_STATIC_DRAW);
```

```c
Must pass an offset here instead of a pointer
```

```c
glVertexAttribPointer(vertexID,3, GL_FLOAT, false, 0, 0)
```
Batching

- Try to combine as many of your drawcalls together as possible
- If objects use different textures try to combine the textures together in a texture atlas
  - This can be done automatically but often best done by artists
  - Update your texture coordinates accordingly

```cpp
glBindTexture(<texture1>);
GlDrawElements(<someVertices>);
glBindTexture(<texture2>);
GlDrawElements(<someVertices2>);
glBindTexture(<texture3>);
GlDrawElements(<someVertices3>);
glBindTexture(<texture4>);
Etc....
```
Enabling Energy-Aware Coding

- ARM Energy Probe
  - Lightweight power measurement for software developers
  - Correlates power consumption with software execution in Streamline
  - Monitor up to three voltage rails simultaneously
  - Helps developers to make informed decisions at all layers of the software stack

Applications

Libraries

Kernel

Effective peripheral management, energy-efficient parallel code

Optimized energy hotspots (e.g. codecs)

Improved power management schemes
The Power of Having It All in One Place

- How effective are you managing your energy budget?

Monitor instant voltage, current and power per channel

How long it takes the power manager to respond to changes in CPU load?
Application Resource Optimizer (ARO)

- Free / Open Source Network-centric diagnostic tool
  - (yes, it is by AT&T but you don’t need an AT&T device)
  - Requires root for pcap/data collection
  - APK on device, java app for captured data analysis

Test Your Application  Transfer Trace Files  Process Trace
How Can ARO Make Apps Faster?

- The fixes identified by ARO will tune your application to higher performance and speed
  - App-specific Analysis
  - Highlight Key Areas to Improve
  - Increase Network Availability
  - Improve Battery Life
  - Get Faster Response Times
- Simple, common sense development best practices in network environments
  - Reducing connection times
  - Caching files
  - Eliminating errors
- Cross Platform and Network Agnostic
Analysis overload: Fixing the problems…

- **My leading questions:**
  - Am I being smart with resources?
  - Is this performance bottleneck parallelizable?
  - Is this Java or Native? Would it be better the other way around?
  - Has this been done before? Don’t reinvent the wheel.
  - What version of Android should I target?
Networking Resources

- Close Connections
  - >80% of applications do NOT close connections when they are finished
  - 38% more power on LTE (18% more power on 3G)

- Cache Your Data
  - 17% of all mobile traffic is duplicate download of the same unaltered HTTP content (1)
  - “It’s just a 6 KB logo” -- 6 KB * 3 DL/session * 10,000 users/day = 3.4GB/month
  - Reading from local cache is 75-99% faster than downloading from the web
  - Even if caching IS supported – it is OFF by default

- Manage Every Connection
  - Group your connections
    - Save battery, speed up applications

Closing Connections: CODE

- MultiRes Sample app from Android SDK

```java
HttpURLConnection getimagecloseconn = (HttpURLConnection) urln.openConnection();
getcodecloseconn.setRequestProperty("connection", "close");

getcodecloseconn.connect();
String cachecontrol = getimagecloseconn.getHeaderField("Cache-Control");
InputStream isclose = getimagecloseconn.getInputStream();
    bitmap = BitmapFactory.decodeStream(isclose);

getcodecloseconn.disconnect();
```
Caching Methods (How do I do it?)

**ETags**

HTTP/1.1 304 Not Modified
Date: Mon, 29 Oct 2012 20:53:03 GMT
Server: Apache
Connection: close
ETag: "2d4f-4cd3729619900"
Expires: Tue, 29 Oct 2013 20:53:03 GMT
Cache-Control: max-age=0

**Cache Control Headers**

HTTP/1.1 200 OK
Date: Mon, 29 Oct 2012 20:51:38 GMT
Server: Apache
Last-Modified: Mon, 28 Jun 2004 00:03:33 GMT
Accept-Ranges: bytes
Content-Length: 27007
Cache-Control: max-age=1437313
Connection: close
Content-Type: image/jpeg

- Each file has a Unique Tag
- Revalidated on server for each request
  - High Performance Web Sites: Rule 1 – Make Fewer HTTP Requests (1)
  - Adding a connection drains battery, adds 500-3,000 ms latency
- Important to carefully assign Max-Age times
- App will not check file on server until Max-Age is reached
  - Retrieval is strictly file processing time

(1) http://developer.yahoo.com/blogs/ydn/posts/2007/04/rule_1_make_few/
Caching: Worth the Effort?

Android 4.0:

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //establish a cache
    try {
        File httpCacheDir = new File(getCacheDir(), "http");
        long httpCacheSize = 10 * 1024 * 1024; // 10 MiB
        HttpResponseCache.install(httpCacheDir, httpCacheSize);
    } catch (IOException e) {
        Log.i(TAG, "HTTP response cache installation failed:" + e);
    }
}
```

Don’t leave older devices in the cold: Consider adding reflection for older versions of Android
Grouping Connections

1. Download an image every 60s
2. Download an Ad every 60s
3. Send Analytics to a Server every 60s

Ungrouped: 38J of energy used!!

Grouped: 16J of energy used!! 58% savings!
Other best network practices

- Remove redirects to files, they add ~2-3 seconds per request
- Pre-fetching files that are used often
- Thread file downloads instead of serial download
- No 4xx 5xx HTTP response error codes should occur
- Be careful with periodic connections
  - Regular 3 minute polls for updates could remain connected for 1.2 hours of the day consuming around 20% of your battery.

Ad download every 30s
Going Native (NDK)


• Native Development Kit is used for writing native C/C++ code and calling your it from within an Android App through Java Native Interface (JNI).
# Native Development Kit (NDK) for ARM

- NDK is a comprehensive tool kit to enable application developers to write directly for the ARM processor.

### ARM Improvements

<table>
<thead>
<tr>
<th>Improved performance and code density with GCC 4.4.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Optimizations for Cortex-A9</td>
</tr>
<tr>
<td>- Support for VFPv3</td>
</tr>
</tbody>
</table>

### General highlights

<table>
<thead>
<tr>
<th>New NativeActivity feature eliminates need to write Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Addition of default C++ STL</td>
</tr>
<tr>
<td>- New API’s</td>
</tr>
<tr>
<td>- Input subsystem, sensor data</td>
</tr>
<tr>
<td>- Windows, surface subsystem</td>
</tr>
<tr>
<td>- OpenGL ES Audio API</td>
</tr>
<tr>
<td>- Access to APK graphics assets</td>
</tr>
<tr>
<td>- EGL library to create and manage OpenGL ES textures and services</td>
</tr>
</tbody>
</table>

Android™ applications can be written in Java, native ARM code, or a combination of the two.

**NEON supported since the r5 release**
Benchmark Results
specific media intensive test case

Time in seconds

Just Native

For more info on NDK, see my webinar at: http://goo.gl/GTwPH
SMP and parallelization

- Nearly every Android and mobile device on the market today is multicore and the trend will continue – Design multi-threaded apps
- Davlik Java threads and IPC
  - AsyncTask is often the simplest way to quickly push a task onto a background worker thread with little IPC complexity
- Bionic C library implements a version of the Pthreads API
  - most of the pthread_* and sem_* functions are implemented but no SysV IPC
  - if it is declared in pthread.h or semaphore.h, it will mostly work as expected
SMP and parallelization

- **GPU Compute**: Rendscript Compute offers a high performance computation API at the native level
  - Write in C (C99 standard)
  - Run operations with automatic parallelization across all available processor cores
  - Platform independent

- Simpler to use than you might expect.
  - Your Rendscript code resides in .rs and .rsh files in the `<project_root>/src/` directory
  - Call `forEach_root()` with your renderscript function, input and output allocations. See: [developer.android.com/guide/topics/renderscript](http://developer.android.com/guide/topics/renderscript)
SMP and parallelization

- OpenGL ES 2.0 enables full programmable 3D graphics for programmable embedded GPUs
  - Royalty-free, cross-platform API
  - 2D and 3D graphics
  - Supported in both Android’s framework API and the NDK

malideveloper.arm.com

- OpenGL ES SDK and Sample Code
- Shader libraries and complier
- Texture compression and ASTC Codec
- Asset compiler and conditioning tools
- OpenGL ES 2.0 and 3.0 emulators

- Full profile OpenCL is an option with some GPUs and possible to use in Linux but not supported by Google in Android. Developer beware.
Write java for mobile/embedded/battery

- new
  - Don’t call this. Ever.
  - At least not in CPU bound/frequent activities
  - Try to use static variables or only allocate upfront or at natural pauses in activity
  - Avoid triggering Garbage Collection (use DDMS)
  - watch Java IO 2009: [http://goo.gl/7xCMg](http://goo.gl/7xCMg)

- In JellyBean use new features for graphics like
  - android.view.Choreographer for v-sync pulses
  - myView.postInvalidateOnAnimation()
  - don't draw stuff that won't be displayed
c.quickReject(items…), Canvas.EdgeType.BW
SIMD: NEON

- General purpose SIMD processing useful for many applications
- Supports widest range multimedia codecs used for internet applications
  - Many soft codec standards: MPEG-4, H.264, On2 VP6/7/8, Real, AVS, …
  - Supports all internet and digital home standards in software
- Fewer cycles needed
  - NEON will give 1.6x-2.5x performance on complex video codecs
  - Individual simple DSP algorithms can show larger performance boost (4x-8x)
  - Processor can sleep sooner => overall dynamic power saving
- Straightforward to program
  - Clean orthogonal vector architecture
    - Applicable to a wide range of data intensive computation.
    - Not just for codecs – applicable to 2D/3D graphics and other processing
    - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
  - Off-the-shelf Tools, OS, commercial & open source ecosystem support
Don’t Reinvent the wheel!
NEON in Open Source Today

- **Google WebM** – 11,000 lines NEON assembler!
- **Bluez** – official Linux Bluetooth protocol stack
- **Pixman** (part of cairo 2D graphics library)
- **ffmpeg (libav)** – libavcodec
  - LGPL media player used in many Linux distros and products
  - Extensive NEON optimizations
- **x264** – Google Summer Of Code 2009
  - GPL H.264 encoder – e.g. for video conferencing
- **Android** – NEON optimizations
  - **Skia** library, S32A_D565_Opaque 5x faster using NEON
  - Available in Google Skia tree from 03-Aug-2009
- **LLVM** – code generation backend used by Android RenderScript
- **Eigen2** – C++ vector math / linear algebra template library
- **TheorARM** – libtheora NEON version (optimized by Google)
- **libjpeg / libjpeg-turbo** – optimized JPEG decode
- **libpng** – optimized PNG decode
- **FFTW** – NEON enabled FFT library
- **Liboil / liborc** – runtime compiler for SIMD processing
- **webkit** – used by Chrome Browser
How to use NEON

Opensource libraries, e.g. OpenMAX, libav, libjpeg, Android Skia, etc.
- Freely available Open Source optimizations

Vectorizing Compilers
- Exploits NEON SIMD automatically with existing source code
- Status: Released (in DS-5 armcc, CodeSourcery, Linaro gcc and now LLVM)

C Intrinsics
- C function call interface to NEON operations
- Supports all data types and operations supported by NEON
- Status: Released (in DS-5 and gcc)

Assembler
- For those who really want to optimize at the lowest level
- Status: Released (in DS-5 and gcc/gas)

Commercial vendors
- Optimized and supported off-the-shelf packages
What is Project Ne10?

- NE10 is designed to provide a set of common, useful functions which
  - have been optimised for ARMv7 and NEON
  - provide consistent well tested behaviour
  - and that can be easily incorporated into applications
  - is targeted at Android and Linux to maximize app performance

- Features
  - Usable from C/C++ and Java/JNI
  - The library is modular; functionality that is *not* required within an App can be discarded
  - Functions similar to the Accelerate Framework provided by iOS
Why use Project Ne10?

- It is Free
  - No commercial complications- ‘build and ship’ Apache 2.0 License
  - No liability offered from ARM, no money paid to ARM
  - well-tested behavior with example code

- Use of the Ne10 library should be a joy, not a chore
  - Out-of-box and user experience is critical to success
  - Build and go, accessible documentation, clear code
  - Code promotes the best of the ARM Architecture- build on it
  - Lets you get the most out of ARMv7/NEON without arduous coding
  - Supported by ARM, community contributions welcome
Ne10Droid – The App in action

- NE10Droid is a benchmarking Android App that uses NE10.
- Routines are written using VFP in C, VFP in Assembly and NEON.

- Example routines:
  - `arm_result_t normalize_vec2f(arm_vec2f_t * dst, arm_vec2f_t * src, unsigned int count);`
  - `arm_result_t normalize_vec3f(arm_vec3f_t * dst, arm_vec3f_t * src, unsigned int count);`
  - `arm_result_t normalize_vec4f(arm_vec4f_t * dst, arm_vec4f_t * src, unsigned int count);`
Conclusions

- For the Simple, Quick-to-Market option, stick with Dalvik but consider JellyBean’s tools and NDK options
  - Can always optimize in version 1.1
- Be smart about your network resources
- Why write code you don’t have to?
  - Look for highly optimized code with a compatible license
  - Can be beneficial without being a perfect fit
- Ideal candidates for Threads, NEON and GPU Compute: audio, image, video and game code
- There is a lot of extra performance there if you really need it
  - Particularly if you can use ARMv6, v7 extensions (NEON)
  - Learning new stuff is fun, so experiment
How Do I Group Connections?

```java
if (Tel.getDataActivity() > 0) {
    if (Tel.getDataActivity() < 4) {
        // ok, we are passed the minimum time to check
        // and we found network activity-
        // download the image here using image getter
        imagegetter(counter, numberOfimages);
        // and show the ad
        AdRequest adRequest = new AdRequest();
        adRequest.addTestDevice(AdRequest.TEST_EMULATOR);
        adView.loadAd(adRequest);
        // Initiate a generic request to load it with an ad
        adView.loadAd(new AdRequest());
    }
}
```
Grouping Connections for Speed

- Threading file downloads vs. serial download

![Diagram showing comparison between threading and serial downloads.](image-url)
Closing Connections: Example

38% more power on LTE! (18% more power on 3G)
What’s in Ne10 today?

- Absolute value, multiply and accumulate and other arithmetic operations of floating point arrays with scalar and constant values along with utility functions.
- DSP FIR/IIR, CFFT/CIFFT and RFFT/RIFFT functions
- SIMD Component-wise Arithmetic on Two Vectors
- Normalize up to 4 dimensional vectors of the input array and store them in the corresponding elements of the output array.

\[
\hat{a} = \frac{a}{\|a\|} = \frac{a_1}{\|a\|}e_1 + \frac{a_2}{\|a\|}e_2 + \frac{a_3}{\|a\|}e_3
\]

(*normalize_vec3f)(arm_vec3f_t * dst, arm_vec3f_t * src, unsigned int count)

- Matrix-Constant Arithmetic e.g. add, sub, mult, div, invert, transform, identity, Matrix-Vector, Vector-Vector and Matrix-Matrix Algebra up to 4 dimensions provided

\[
A^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}^{-1} = \frac{1}{\det(A)} \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & K \end{bmatrix}^T = \frac{1}{\det(A)} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & K \end{bmatrix}
\]

(*invert_mat3x3f)(arm_mat3x3f_t * dst, arm_mat3x3f_t * src, unsigned int count)
What is NEON?

- NEON is a wide SIMD data processing architecture
  - Extension of the ARM® instruction set
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
- NEON Instructions perform “Packed SIMD” processing
  - Registers are considered as vectors of elements of the same data type
  - Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single prec. float
  - Instructions perform the same operation in all lanes
Using NDK with a Web UI

- Prefer Web-style UI to the normal Android Widgets?
- Or just familiar with HTML and don’t want to change?
  - Good news, you can still take advantage of the NDK
- Put a little extra zip in your Javascript
  - The pages don’t have to be part of the assets
- Embed your web pages into an Android App Shell
  - You can extend the Javascript libraries with calls to Java
  - And those calls to Java can call NDK JNI Functions

```java
WebView web = (WebView)findViewById(R.id.webview);
ProxyBridge jscriptBridge = new ProxyBridge();
web.addJavascriptInterface(jscriptBridge, "pBridge");

WebSettings settings = web.getSettings();
settings.setJavaScriptEnabled(true);
web.loadUrl("file:///android_asset/index.html");
```
Runtime Choice of Binaries

- Ideally want to ship one binary for all Android Devices
  - This can be done with a little thought
- Build a shared library for each HW variant you want
  - Remember to keep a Java implementation as fallback
- Use a Interface or Abstract Class pattern select the right code at runtime
- Use either Board (which gives device information) or a manual scan of ‘/proc/cpuinfo’ to select which code to use
- Will bloat your App
- Use Thumb or Thumb2 instead ARM where you can
  - Native Thumb is still much faster than Dalvik
  - Thumb-2 is similar to ARM, supports NEON and V6 SIMD
- Weigh each choice of Native method carefully
THANK YOU