Descriptions of different Optimizations used in Android

Note: This document is currently out-of-date and does not include the new kid on the block UBERTC. Based on my current experience UBERTC which is a combination of AOSP and LInaro but merged to the latest Toolchains components from gnu.org (similar to SaberMod) is the most stable and fastest toolchain out there. Make sure to stop by and check out https://github.com/UBERTC/ find uber-manifest and build your own to test out!

Linaro: See http://www.linaro.org/linux-on-arm/

There are several hundreds of developers working on Linaro. Linaro merges changes from Google and also from GNU.org to optimize toolchains which they use to optimize kernels and ROMs. They also make changes to AOSP code to allow compiling with either higher GCC version or for better optimization. Linaro is always months ahead of Google. Example: Google still uses 4.7.2 while Linaro uses 4.7.4. The differences from 4.7.2 and 4.7.4 are literally thousands and thousands of lines of code to make toolchains more optimized. This is why many developers choose the Linaro route. Benchmarks and overall user experience has shown that linaro does a better job of optimization than Google.

Fun Fact: Ubuntu 14.04 has Linaro 4.8 set as default GCC currently. (There is obviously more to linaro than many give it credit. I can't simply be a buzzword if even PCs use it.)

SaberMod: Sources see https://github.com/SaberMod

SaberMod is a smaller scale project manned by only a few developers with the goal to enhance android in every way possible. Emphasis is placed on keeping things closer to AOSP and not deviating as far away from stock feel as say Linaro. Developers here use latest GCC merging it on a weekly basis and continuously patch AOSP patches as Google makes them. SaberMod developers always build toolchains with the latest Ubuntu internal toolchains and use all of the GCC optimizations possible to make your ROM fast as ever! SaberMod also makes use of full system too and all of the -O3, Graphite, and other great GCC flags to make your ROM/Kernel as efficient as possible. They patch anything and everything android and their hardwork indeed pays off! Currently SaberMod benchmarks are even higher than Linaro. Don't believe me try something sabermod built for yourself! You won't regret it!

Regular Google AOSP Toolchains: see https://android.googlesource.com/toolchain/gcc/

Note: These toolchains are not used as buzzwords this is more for comparison sake

Currently Google toolchains are anywhere from 9 months to more than a year old. Don't believe me checkout the sources? Google also seemed to have built its last batch of toolchains on an older Ubuntu version with GCC 4.6 or maybe GCC 4.7 as the system default toolchain. Even a rebuild of their current source using the Latest Ubuntu 14.04 with out-of-box GCC 4.8 or experimental GCC 4.9 increases overall UI responsiveness. Benchmarks are also slightly improved. This is why it is important to learn to build your own toolchains because Google is always way behind and you get better performance out of your own toolchains. This is not a fallacy!!! Give these things a try for yourself and you will feel the difference.

Toolchain Optimization Flags: See https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

-O2 vs. -O3 vs -Ofast: Normally stock kernels come with -O2 as the default optimization level. Many folks find themselves asking questions like How does this differ from “-O3” or “-Ofast” Is this a placebo? Read this link https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html about GCC Optimization options and you’ll find out the differences between -O2, -O3, and -Ofast. You’ll notice several more optimizations are added from -O2 to -O3. Unfortunately, Google 4.7/4.8 toolchains don’t even support -O3 or -Ofast (At least the arm-eabi toolchains don’t, I haven’t even wasted my time with arm-androideabi). So they’re not only old but don’t even have the ability to optimize as much either.

Graphite: See here https://gcc.gnu.org/wiki/Graphite-4.8. Graphite is a framework for high-level memory optimizations using the polyhedral model.

From the developers about said pelohedral optimization: "To get a real, generic polyhedral optimizer for Graphite we have chosen the Pluto algorithm. Pluto is an polyhedral optimizer that uses a single cost function to optimize simultaneously for data locality, parallelism and tileability. It has shown good results on various kernels and Uday, the original author was employed to reimplement it in IBM XL. We added an implementation of this algorithm to isl. My recent patch set enables Graphite to use this new optimizer. Even though the patch is an early draft and definitely needs tuning to match the results of the original implementation, it is a great starting point for a real polyhedral optimizer in Graphite." This was over a year ago and it is now fully implemented.

If you want to learn more: http://gcc.gnu.org/wiki/Graphite-4.8 the first 3 of the 4 things they wanted to accomplish have been achieved.

Opticharging

Custom ROMs in the beginning had limited space to work with which is why Cyanogen introduced opticharging to the custom release tool in order to shrink apks to fit more apps on

system partitions of the original android devices. The opticharger script pulls apart apks near the end of the build and optimizes all pngs drawables found in them. Originally this script used optipng which is great and there is absolutely no quality loss involved in the png compressions but more recently I've begun using pngquant because it compresses pngs even smaller 30-70% with usually an average of about 50%. Pngquant does result in a slight loss of quality but nobody has noticed yet ;) (See more on pngquant here: http://pngquant.org/) (If you are worried about quality you can always use optipng which compresses without any quality loss themers usually use this option instead)

Currently, CyanogenMod has abandoned the use of the opticharger but many ROMs still use it such as SLIM, AOKP, LiquidSmooth, Dirty Unicorns, Carbon, Validus, and many others. Themers and app developers alike use these techniques as well to make their apps/themes run more smoothly. Making pngs 70% smaller actually makes the loading time 3 times faster for these pngs and also saves you RAM. I understand apks aren't all pngs but you'd be surprised how many junk pngs google has left behind since froyo that serve no purpose but are loaded into your ram with SystemUI anyways. Thankfully opticharging shrinks all of those undesirables by usually 50-70% and saves you several MB worth of space and speeds up your SystemUI. Since you can't unload the system UI this is well worth the effort! My motto always is every little bit counts :)

Many still argue that high end device don't need opticharging anymore because they are fast enough to handle these full-sized pngs. While this is true it still doesn't change the fact that opticharing IS still slightly faster and opticharged apks do use less ram. If you don't believe me try it for yourself!