Portability and Optimizations

Lecture 11

Android Native Development Kit

13 May 2014

Portability

Optimization

Bibliography

Keywords

Outline

Portability

Optimization

Bibliography

Keywords

Portability - Different Operating Systems

- Portability on different operating systems
- Each operating system has its own System API and Libraries
- Possible solutions:
  - Standard Libraries (C, C++, etc.)
    - Implement a certain standard
    - Standard C library implemented after ANSI C standard
    - Interaction with the OS
  - Wrappers around API (system calls, library calls)
    - Identify OS and make appropriate calls
  - Use compiler macros
    - Identify the OS
    - `__ANDROID__`, `__linux__`, `_WIN32`, `__MACH__`, etc.

Portability - Different Architectures

- Portability on different hardware platforms
- Each architecture has a certain ABI
- ABI:
  - Describes data type size, align, calling convention, dealing with system calls, binary format of object files, etc.
  - Also depends on OS
- Solution: compile for a certain ABI with the appropriate toolchain
- Cross-compilation:
  - Compile for a different architecture than the one we are on
  - Select the appropriate toolchain for the target architecture and OS
  - Toolchains with correct headers, libraries and ABI

- Can generate standalone toolchain for an Android version and ABI
- Easily integrate with build system for other platforms
- `$NDK/build/tools/make-standalone-toolchain.sh --platform=android-<API_VERSION> --arch=<ARCHITECTURE> --install-dir=<DIRECTORY>`
- `ARCHITECTURE` can be x86, ARM (default) or MIPS
- Contains C++ STL library with exceptions and RTTI

Outline

Portability

Optimization

Bibliography

Keywords

Compiler

- Identify performance problems using profiling
- Compilers can generate optimized code
- `APP_OPTIM` - differentiate between debug and release versions
  - Defined in `Application.mk`
- For release versions it uses `-O2` and defines `NDEBUG`
  - `NDEBUG` disables assertions and can be used to remove debugging code
- The compiler may perform (implicit) vectorization => increase performance
- Might not do vectorization when appropriate, sometimes it's necessary to optimize by hand (but check your algorithm first)

Libraries

- Libraries can provide highly optimized functions
- Some are architecture dependent
- Math:
  - Eigen
    - C++ template library for linear algebra
  - ATLAS
    - Linear algebra routines
    - C, Fortran
- Image and signal processing:
  - Intel Integrated Performance Primitives
    - Multimedia processing, data processing, and communications applications
  - OpenCV
    - Computational efficiency, real-time applications
    - C++, C, Python and Java
- Threading:
  - Intel Threading Building Blocks
    - C and C++ library for creating high performance, scalable parallel applications

Low level optimizations

- Optimized algorithm, compiler does not optimize properly, no optimized libraries are available => low level optimizations
  - Use (explicit) vectorization
  - Intrinsic compiler functions or assembly
- Not all CPUs have the same capabilities
- At compile time:
  - Build different versions of libraries for each architecture
  - In Makefile depending on the ABI
- At runtime:
  - Execute a certain piece of code only on some architectures
  - Choose specific optimizations based on CPU features at runtime

CPU Features Library

- cpufeatures library on Android
- Identifies processor type and attributes
- Make optimizations at runtime according to the procesor
- Main functions:
  - android_getCpuFamily
    - ANDROID_CPU_FAMILY_ARM, ANDROID_CPU_FAMILY_X86, etc.
  - android_getCpuFeatures
    - Returns a set of bits, each representing an attribute
    - Floating point, NEON, instruction set, etc.
  - android_getCpuCount
    - Number of cores

ABI: x86

- Android NDK supports 4 ABIs: `x86`, `armeabi`, `armeabi-v7a`, `mips`
- `x86` supports the instruction set called 'x86' or 'IA-32'
- Includes:
  - Pentium Pro instruction set
  - MMX, SSE, SSE2 and SSE3 instruction set extensions
- Code optimized for Atom CPU
- Follows standard Linux x86 32-bit calling convention

ABI: armeabi

- Supports at least ARMv5TE instruction set
- Follows little-endian ARM GNU/Linux ABI
  - Least semnificative byte at the smallest address
- No support for hardware-assisted floating point computations
  - FP operations through software functions in libgcc.a static library
- Does not support NEON
- Supports Thumb-1
  - Instruction set
  - Compact 16-bit encoding for a subset of ARM instruction set
  - Used when you have a small amount of memory
  - Android generates Thumb code default

- Extends `armeabi` to include instruction set extensions
- Supports at least ARMv7A instruction set
- Follows little-endian ARM GNU/Linux ABI
- Supports VFPv3-D16
  - 16 dedicated 64-bit floating point registers provided by the CPU
- Supports Thumb-2
  - Extends Thumb with instructions on 32 bits
  - Cover more operations

ABI: armeabi-v7a

- Supports NEON
  - 128-bit SIMD architecture extension for the ARM Cortex^{TM}-A
  - Accelerate multimedia and signal processing: video encode/decode, 2D/3D graphics, image/sound processing
  - Set `LOCAL_ARM_NEON` to true in Android.mk
    - All sources are compiled with NEON support
    - Use NEON GCC intrinsics in C/C++ code or NEON instructions in Assembly code
  - Add `.neon` suffix to sources in `LOCAL_SRC_FILES`
    - Compile only those files with NEON support
    - `LOCAL_SRC_FILES := foo.c.neon bar.c`

NEON

```
# define a static library containing our NEON code
ifeq ($(TARGET_ARCH_ABI),armeabi-v7a)
        include $(CLEAR_VARS)
        LOCAL_MODULE     := neon-example
        LOCAL_SRC_FILES  := neon-example.c
        LOCAL_ARM_NEON   := true
        include $(BUILD_STATIC_LIBRARY)
endif # TARGET_ARCH_ABI == armeabi-v7a
```

```
#include <cpu-features.h>
[..]

if (android_getCpuFamily() == ANDROID_CPU_FAMILY_ARM &&
        (android_getCpuFeatures() &
        ANDROID_CPU_ARM_FEATURE_NEON) != 0){
        // use NEON-optimized routines
        [..]
}
else{
        // use non-NEON fallback routines instead
        [..]
}
```

Outline

Portability

Optimization

Bibliography

Keywords

- $NDK/docs/STANDALONE-TOOLCHAIN.html
- $NDK/docs/CPU-FEATURES.html
- $NDK/docs/CPU-ARCH-ABIS.html
- $NDK/docs/CPU-ARM-NEON.html
- $NDK/docs/ANDROID-MK.html
- $NDK/docs/APPLICATION-MK.html
- `https://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html`

Portability

Optimization

Bibliography

Keywords

- Portability
- Standard Libraries
- Wrappers
- ABI
- Toolchain
- Cross-compilation
- Profiling
- Optimization
- Vectorization
- Optimized libraries
- CPU features
- MMX, SSE
- NEON
- Little-endian
- Thumb
- VFP