CMSSW ON ARMV7-A ISA

DAVID ABDURACHMANOV

CERN, CMS && Vilnius University (VU)

AGENDA

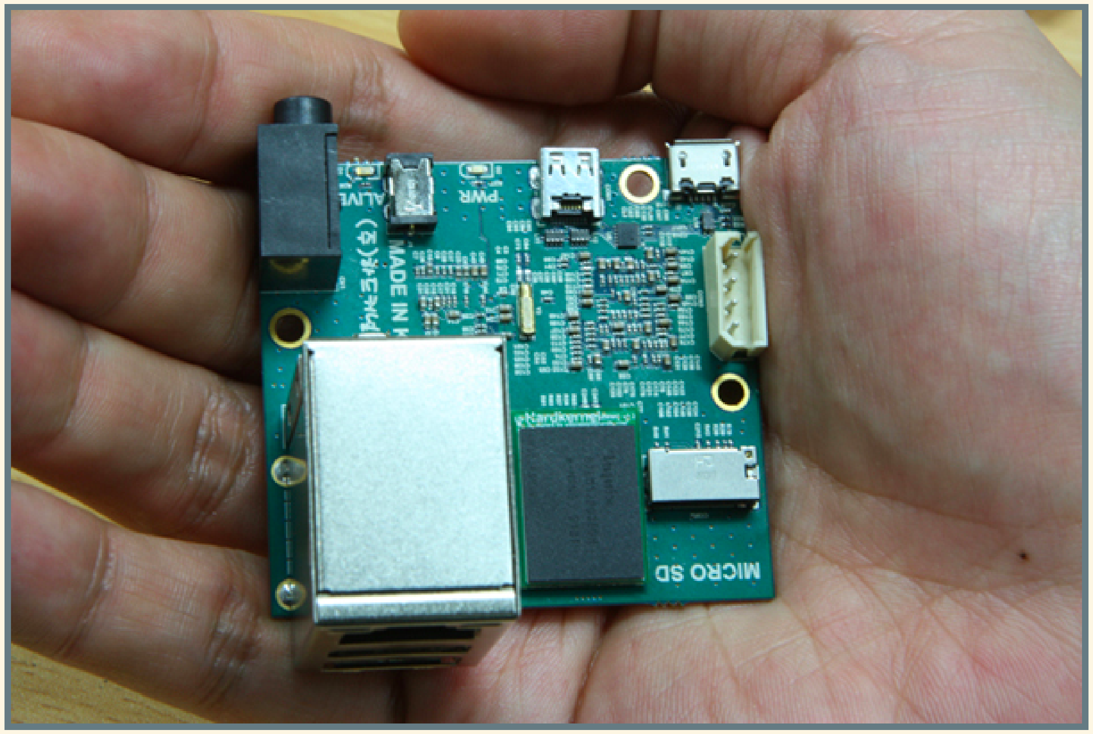
1. Introduction to hardware
2. Status of CMSSW port to ARMv7-A
3. ROOT linCintex port to ARMv7-A
4. Development issues and solutions
5. Cycle counter
6. VDT NEON vectorization
7. Future work

WHY?

- We want to have cross-compilation toolchain (**cmsBuild**) and ARMv7-A provides a radically different architecture compared to **i*86/x86_64** we are used to.
- Every migration to new OS or/and new compilers or compiler versions allows us to discover a number of hidden issues in our code base (**CMSSW**) and build tools.

ARMV7-A HARDWARE WE USE

- CMS ARM Cluster: **armcms{01,02}.cern.ch**
- Plus, an additional non-public machine is used for development.
- We are using **ODROID-U2**:
  - CPU is quad-core **Exynos4412 Prime** at 1704Mhz.
  - CPU is capable running at 2000Mhz (might yield undefined behavior at 2000Mhz).
  - Cores are **Cortex-A9 MPCore**.
  - 2048MB LP-DDR2 memory (512MB per core).
  - eMMC, uSD, USB 2.0, Ethernet RJ-45.


ALIVE
PWR
Hardkernel
MICRO SD

WHAT MARKET THINKS

- HP says:

  > *“89% less energy, 94% less space, and 63% less cost”*

- Server-grade ARM is available starting 2012 based on **Calxeda highbank** (current gen Cortex-A9 cores).
- **Boston Viridis**:
  - 2U contains 48 nodes (each has 4 quad-core SoC chips, 1GB per core). That's 192 cores per 2U enclosure.
  - It requires <300W (5W per node) per 2U enclosure.
  - That's 4032 cores, 4TB of RAM under 6.3kW per 42U rack.

CMSSW ON ARMV7-A

- Initial port was started on **QEMU** (incl. Linaro port).
- **QEMU** uses a single thread to emulate all virtual CPUs. Slow.
- Parallel execution of virtual cores possible, but not upstream. Alternative forks exists, .e.g, **COREMU**, but no stable support for **ARM**
- **QEMU** is buggy by experience. Bug report filed.
- Fedora 17 ARM was picked as Fedora -> RHEL -> SLC.
- Now using Fedora 18 ARM Remix (kernel 3.0.65+).
- Fedora 17/18 should be similar to RHEL 7.
- Official CMS architecture target: **fc18_arm7hl_gcc480**.
  - Fedora 18 ARM, ARMv7-A Hard Floats, GNU GCC 4.8.0

CMSSW IS BUILT IN STAGES #1

1. **build_rpm.sh** builds prerequisites for **Bootstrap Driver Kit**. Mainly builds our RPM for packaging.
2. Using **cmsBuild** build **bootstrap-driver**, **cms-common**, **lcg-dummy**, and **local-cern-siteconf** targets.
3. Then build **cmssw-tool-conf** target, which holds all CMSSW prerequisites.
4. Then build **cmssw** target.

CMSSW IS BUILT IN STAGES #2

1. **Stage1** is done. Takes 0:30 for compilation.
2. **Stage2** is done. Takes 3:30 for compilation.
   - External package not available for ARM: **oracle** (proprietary binaries only exist for **i*86** and **x86_64**).
3. **Stage3** is done. Takes about 12:00 for compilation.
   - Compilation time could be improved. CPU is not utilized efficiently.
4. We are **Stage4** currently working on compiling **CMSSW_6_2_X** on ARMv7-A. Takes 25:30 for compilation.
   - 6 out of 1116 packages are affected by missing external packages.

ROOT 5 ON ARMV7-A ISA

- Copy missing **iosenum** file

```
cp ./cint/iosenum/iosenum.linux3 ./cint/iosenum/iosenum.li
nuxarm3
```

- Configure with target **linuxarm**
- Patch **Cintex** to build on **armv7l** machine
- All the details **savannah.cern.ch/bugs/?100934**

PORTING CINTEX

- **Cintex** was written for **i*86** and **x86_64** targets
- **libCintex** includes compiled-in function calls (templates) via function pointer:

```
static void f0a() {
  typedef void (*f_t)(void*);
  ((f_t)FUNCPATTERN)((void*)DATAPATTERN);
}
```

- Where **FUNCPATTERN** and **DATAPATTERN**:

```
#define FUNCPATTERN 0xFAFAFAFAL
#define DATAPATTERN 0xDADADADAL
```

- Are compiled-in patterns later substituted with real addresses.

FOA() X86_64 VARIANT

On **x86_64** compiled-in patterns are easily noticeable. Easy to find offsets and modify in-memory addresses for **FUNCPATTERN** and **DATAPATTERN**.

```
0000000000000000 <_ZN4ROOT6CintexL3f0aEv>:
   0:	48 bf da da da da da 	mov    $0xdadadadadadadada,%rdi
   7:	da da da 
   a:	48 b8 fa fa fa fa fa 	mov    $0xfafafafafafafafa,%rax
  11:	fa fa fa 
  14:	ff e0                	jmpq   *%rax
  16:	66 2e 0f 1f 84 00 00 	nopw   %cs:0x0(%rax,%rax,1)
  1d:	00 00 00 
```

F0A() ARMV7L (ARM MODE) VARIANT

- Fixed-length (32-bit) instructions.
- Requires two **MOV*** instructions to copy 32-bit address.
- Compiled-in patterns are not noticeable, e.g., **e30f3afa**. Where is **0xfafa?**
- It's **MOVW** A2 encoding instruction where

```
imm32 = ZeroExtend(imm4:imm12, 32);
```

i.e., **0xfafa** is divided into two bit sequences inside the instruction.

```
00002364 <_ZN4ROOT6CintexL3f0aEv>:
 2364:       e92d4800        push    {fp, lr}
 2368:       e28db004        add     fp, sp, #4
 236c:       e30f3afa        movw    r3, #64250      ; 0xfafa
 2370:       e34f3afa        movt    r3, #64250      ; 0xfafa
 2374:       e30d0ada        movw    r0, #56026      ; 0xdada
 2378:       e34d0ada        movt    r0, #56026      ; 0xdada
 237c:       e12fff33        blx     r3
 2380:       e8bd8800        pop     {fp, pc}
```

FINDING COMPILED-IN PATTERNS

The following

```
if ( *(size_t*)b == DATAPATTERN ) fa_offset = o;
```

becomes

```
if ( ((*(size_t*)b) & 0x000F0FFFUL) == 0x000D0ADAUL &&
   ((*((size_t*)b + 1)) & 0x000F0FFFUL) == 0x000D0ADAUL )
     fa_offset = o;
```

Ignores **<cond>** and **<Rd>** fields in the instruction. Checks how top and low address halves should look in the instruction.

CHANGING COMPILED-IN PATTERNS

```
size_t addr16, addr16_mov;
// Lower part (MOVW)
// 32-bit aligned lower part
addr16 = (size_t)address & 0x0000FFFFUL;
// make imm4:imm12 bit mask
addr16_mov = (addr16 | ((addr16 << 4) & 0x000F0000UL)) & 0x000F0FFFUL;
// apply address correction
*(size_t*)destination = (*(size_t*)destination & 0xFFF0F000UL) | addr16
_mov;
// Top part (MOVT)
addr16 = ((size_t)address & 0xFFFF0000UL) >> 16;
addr16_mov = (addr16 | ((addr16 << 4) & 0x000F0000UL)) & 0x000F0FFFUL;
*((size_t*)destination + 1) = (*((size_t*)destination + 1) & 0xFFF0F000
UL) | addr16_mov;
```

A bit more complicated.

The snippet is only safe on ARMv7-A

ROOT SUMMARY

- The method used in **Cintex** is not safe for **i*86/x86_64** as it heavily relies on compiler optimizer picked instructions.
- Tested only on GNU GCC 4.8.0 in CMSSW.
- Should be safe in **ARM** mode, but not in **Thumb** mode.
- Depends on **MOV{W,T} A2** encoding instructions, but other encodings could be implemented also.
- As long as we compile ROOT and understand how **libCintex** works, we should be fine.

GENERAL ISSUES PORTING SOFTWARE

based experience

SIGNEDNESS IS DIFFERENT ON ARM/POWER

- On INTEL/AMD by default **char** and **bit-fields** are signed.
- On ARM/POWER their are unsigned.
- The following is not portable code:

```
int main(void) {
    char c = 255;
    if (c > 128) {
      return 0; /* unsigned */
    } else {
      return 1; /* signed */
    }
    }
```

- Warning on INTEL/AMD (singed):

```
warning: comparison is always false due to limited range of data type [-Wtyp
e-limits]
```

- Could use **-funsigned-char** / **-fsigned-char** and **-funsigned-bitfields** / **-fsigned-bitfields** to tell GCC what signedness you need.

SIGNEDNESS HITS GETCHAR-LIKE FUNCTIONS

- Mostly noticed in **CASTOR**.
- The following is incorrect usage:

```
char c;
while ((c = getchar()) != EOF) { /* magic with c */ }
```

- It should be like:

```
int i;
while ((i = getchar()) != EOF) { unsigned char c = i; /* magic with actual
char */ }
```

- **EOF** is -1.

ARM DO NOT SUPPORT -M32/-M64

Review your **Makefile**s that on **armv7l** (32-bit) target **-m32** compiler option would not be used. Otherwise compiler will fail with unrecognized option.

ERROR: INTEGER CONSTANT IS TOO LARGE FOR 'LONG' TYPE #1

/usr/include/bits/wordsize.h

```
#define __WORDSIZE      32
```

/usr/include/stdint.h

```
#if __WORDSIZE == 64
typedef unsigned long int       uint64_t;
#else
__extension__
typedef unsigned long long int  uint64_t;
#endif
```

Pre-processor

```
#define __SIZEOF_INT__ 4
#define __SIZEOF_POINTER__ 4
#define __SIZEOF_LONG__ 4
#define __SIZEOF_LONG_LONG__ 8
```

ERROR: INTEGER CONSTANT IS TOO LARGE FOR 'LONG' TYPE #2

VDT failed at:

```
const uint64_t mask=0x8000000000000000;
```

> *"To make an integer constant of type unsigned long long int, add the suffix `ULL' to the integer."*

Initial fix:

```
const uint64_t mask=0x8000000000000000ULL;
```

But *ull* suffix is **C++11** and compiler complains (-*pedantic*). Now we compile in **C++11**. That's required in CMSSW.

CHECK IF COMPILER SUPPORT SSE AND AVX (INTEL/AMD)

- **General rule:** ask compiler, do not check the current system for capabilities. Compiler knows your target.
- Some packages by default assume that **SSE** is available in the machine. **Not true** on ARM.
- Do not use */proc/cpuinfo* or *sysctl* to detect CPU capabilities.
- With **autoconf** check if compiler supports the flag.
- With **CMake** write **FindSSE.cmake**, which compiles a series of *conftest.c* to check compiler flags.

```
cc1: error: unrecognized command line option "-msse4"
```

CORRECTLY DETECT 32/64-BIT TARGETS

- **General rule:** test features, not platforms.
- Do not use **uname -m** to detect 32 or 64 bits based on machine.
- Kernel might still be **i*86**, yet the system is **x86_64**.
- Use **__SIZEOF_POINTER__** macro to check if target is 32 or 64 bit.

```
#define __SIZEOF_POINTER__ 8
```

VIRTUAL MEMORY EXHAUSTED

- 17 object files failed to compile:

```
virtual memory exhausted: Cannot allocate memory
```

- Translation Units (TU) requires high amount of memory to compile.
- Mostly happens with *Reflex* ROOT dictionaries.
- Resource limits:

```
-m: resident set size (kbytes)      unlimited
-v: address space (kbytes)          unlimited
```

- Most likely we hit *high memory* (user space) limit. According to */proc/meminfo*

```
HighTotal:       1342464 kB
```

- 1311MB is the limit for user space. Going above *malloc()* would return *NULL*.
- **Solution**: divide TUs into a smaller units.

INLINE ASSEMBLY AND INTRINSICS

Developers assume that their code will compile on specific machine with needed capabilities. **No or not sufficient enough guards are provided for such parts of the code.**

Do not assume and use pre-processor macros to protect all parts of the code.

NEON intrinsics are available in *arm_neon.h*. **But developers should depend on compiler auto-vectorization capabilities instead.**

CYCLE COUNTER

- INTEL provides Time Stamp Counter (TSC) register since Pentium-era.
- **RDTSC** is used to retrieve TSC value.
- **ARMv7-A** provides Performance Measurement Unit (PMU), which has **PMCCNTR** (32-bit cycle count register).
- It can increment once every processor clock cycle or once every 64 cycles.
- PMU is not visible in user mode, needs to enabled in privileged mode. Easiest way is to write a kernel module, which on initialization enables PMU for user mode.

VDT AND VECTORIZATION

- By default GCC 4.8.0 is configured with **vfpv3-d16** as FPU.
- **NEON** (SIMD unit) is also available: -mfpu=neon
- VDT 0.3.2 introduces **NEON** vectorization.
- For SP we have successful vectorization, e.g.:
  - **Fast_Expf** (72.66ns) - **Fast_Expfv** (26.72ns)
  - **Fast_Logf** (69.61ns) - **Fast_Logfv** (29.11ns)
- But fails in others, e.g.:
  - **Fast_Sinf** (54.76ns) - **Fast_Sinfv** (66.21ns)
  - **Fast_Cosf** (52.00ns) - **Fast_Cosfv** (63.11ns)
- DP does not vectorize. **NEON** does not support DP. DP will be supported in ARMv8 *AArch64* mode.

FUTURE WORK

- Fix current issues with **CMSSW_6_2_X** on ARMv7-A.
- Prepare official integration builds (IB).
- Clean up **CMSSW** and external tool recipes (RPM SPECs) to be cross-platform friendly.
- Prepare **cmsBuild** for cross-compilation, which currently is not supported. On ARM target we are lucky to build native **CMSSW** releases, but does not apply for Intel **MIC**.
- Prepare for **ARMv8** 64-bit architecture. **ARMv8** is coming to server/desktop market in 2014-2015 with **Cortex-A57** and **Cortex-A53**.

Q & A

Kudos to GiulioE, VincenzoI, DaniloP, ShahzadM, ThomasS, et al. for ideas and solving problems