SYNOPSYS®
Accelerating Innovation

ARC Linux

“From a tumbling Toddler to a graduating Teen”

ELC-Europe
Barcelona, November 2012
Vineet Gupta

Agenda

- Introduction
  - ARC Architecture
  - ARCompact ISA
  - Linux Evolution
- Early challenges
- Optimizations
- Community

ARC Architecture

- 32bit RISC load/store Architecture with deep register file (32)
- Cores: ARC600 (MMU-less), ARC700 (MMU)
- MMU: Software Managed TLB, Address Space ID (ASID)
- Caches: L1: Non-snooping, VIPT
- Two interrupt priorities provided by “in-core” IRQ Controller
- Configurability: at the click of a button - literally !
  - **Custom Instructions to extend the ISA**
  - Configurable Cache / MMU Geometry, Page Size
  - Gates vs. Performance (e.g. Small or Fast Multiplier)
- Highly efficient with regards to performance/area and power/area

ARCompact ISA

- ISA allow free intermixing of short (2 byte) and long (4 byte) instructions and each of these in turn can take a 4 byte LIMM (Long immediate).
- Unaligned Data access not supported.
- Most instructions can be conditionally executed (e.g. **ADD.ne**)
- Dedicated Call return register (**BLINK**)
- LP instructions for Hardware Loops (a.k.a. **Zero Overhead Loops**)
  - LP_START, LP_END, LP_COUNT registers
  - Avoids need for software to decrement-counter and compare-and-branch every loop iteration

Toddler Days: Tumbling and Trying

- **2.6.19 ('08)**
  - Arbitrary tasks segfaulted when system stressed
  - Gdb Breakpoints (user-code) worked semi-randomly
  - Support for MMU v2 - First tapeout of ARC700 for Linux Host
  - Cache “current” task pointer in register optimization
  - OProfile support
  - Kernel Stack Tracing (Dwarf2 unwinder based)
  - Low level Event logging (poor man's LTT)
- **2.6.26 ('09)**
  - High Resolution Timers / Tickless Idle / RT Signals
  - File system corrupted due to Cache flush loop overflow
  - Kernel panic on customer board with A/v drivers as modules

Teen: Growing and Maturing

- **2.6.30 ('10)**
  - OProfile “opcontrol” shell script randomly terminated
  - Serious Optimizations
  - Strace Port
  - QT 4.5.2 Port
  - Support for Extension Instruction to do Endian Swap
  - LTP Open Posix Supported
- **2.6.35 ('11)**
  - Kprobes Support
  - Support for ARC700 4.10 (770 core)
    - MMU v3 / SASID / LLOCK-SCOND instructions

Agenda

- Introduction
- **Early challenges**
- Optimizations
- Community

Filesystem Corruption with DMA

- When IDE Disk Driver switched to DMA, the on-disk ext2 filesystem started showing corruption
- ARC Caches are non-snooping / non-coherent
  - DMA From-Device requires Invalidating the D$ lines
  - DMA To-Device requires Flushing the D$ lines
- Off-by-one error in the D$ line invalidate loop
  - Cache flush callers expect inclusive @start but exclusive @end

```
void inv_dcache_range(unsigned long start, unsigned long end)
{
 ...
    /* Throw away the Dcache lines */
-   while (end >= start) {
+    while (end > start) {
        write_new_aux_reg(ARC_REG_DC_IVDL, start);
        start = start + ARC_DCACHE_LINE_LEN;
    }
```

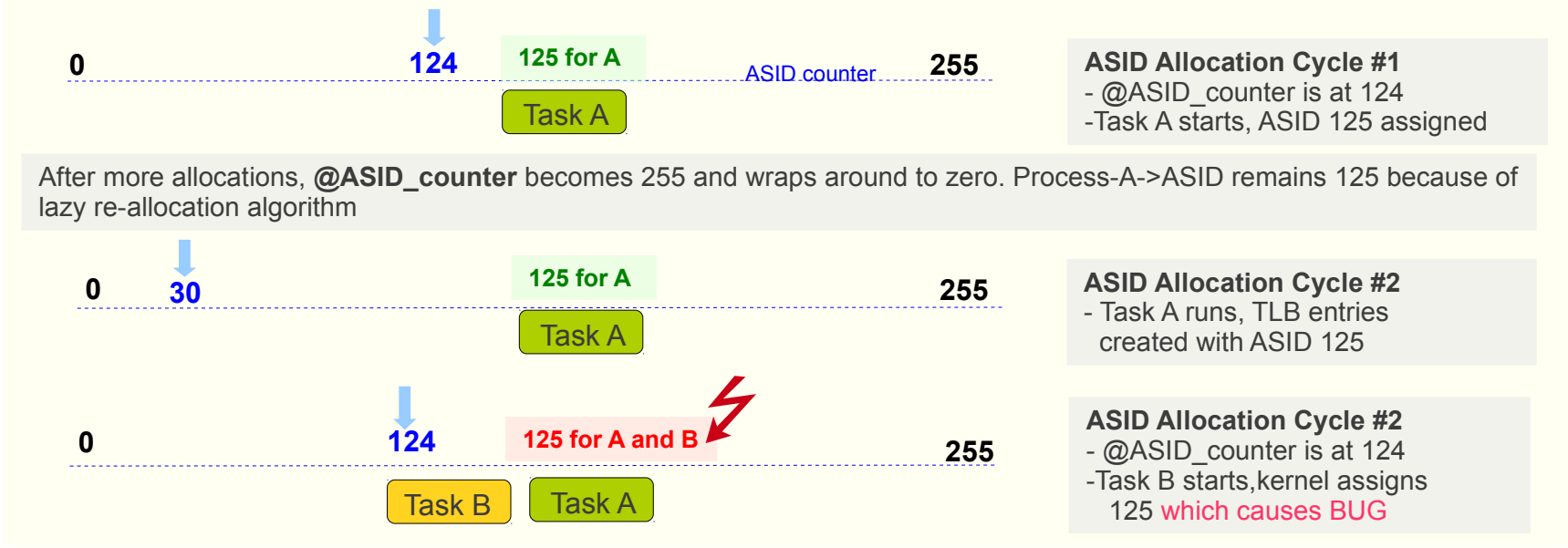
Task Randomly Segfaulting ...

- With 3 telnet sessions, each with a shell script infinite looping on ls, sometimes a telnet task segfaulted
- Low Level Event Logger implemented to trace critical Machine Events [IRQs, TLB Miss Exceptions, syscalls] as well as key kernel events: e.g. do_signal, __switch_to( )
- Analyzed the events backwards - from do_signal( ) and in error case, it was taking 1 fewer D-TLB exception right before hitting the signal
- Background
  - MMU exclusively deals with TLB entries (Page Tables are purely for kernel, to help manage the TLB entries)
  - Page faults are called TLB Miss Exceptions on ARC (separate for Instruction-Fetch and Data load/store)
- So task was somehow using an existing D-TLB entry and some data in that page mapped to a NULL pointer

Task Randomly Segfaulting (cont)

- TLB entries have an 8-bit ASID to allow entries with same vaddr to coexist (corresponding to different tasks) - avoids flush every Task switch
- A new task (fork/execve) is assigned an ASID and when it is *schedule()*ed, MMU PID Reg is set with task's ASID
- ASID allocation using a simple wrapping atomic counter - a new request gets counter++
- When ASID rolls over, kernel flushes the TLB and restarts the allocation cycle - **task ASID (re)assignments however remain unchanged** as that is done “lazily”
- Algorithm by default allows for ASID “stealing”: if ASID counter is at “N”, a new request gives “N+1” even if it is already allocated
- Unless rest of algorithm is carefully written, the 2 prev points combined can cause a ASID reuse - meaning “stale” TLB entries to be used by a task

Task Randomly Segfaulting (cont2)



- Solution was to always keep task->asid behind @ASID_counter
- So in step #2 above,Task-A would be forced to refresh it's ASID, give up 125 and get 31
- In general, for @ASID_counter = “N”, TLB entries would never pre-exist for “N+1”, thus ASID stealing won't cause stale TLB entries reuse.

```
static inline void switch_mm(struct mm_struct *prev, struct mm_struct *next,  struct task_struct *tsk)
{
-       if (next->context.asid == NO_ASID)
+       if (next->context.asid > asid_cache)
                get_new_mmu_context(next);
```

How we debugged these !

- Some of the issues took many weeks
- Tools
  - A very accurate Instruction Set Simulator (ISS) with Instruction Trace
  - A JTAG Debugger allowing to peek/poke almost any architectural CPU element (TLB, Cache, memory)
- Wrote low level event capture (poor man's LTT)
  - And pouring through 100's of events
- Some luck and lots of patience
  - In one case, a crash caused CPU to jump to a random location containing zeros (encoding for Branch-to-self instruction). Thus it would just spin-loop. The JTAG debugger was still connected. For 2 days we analysed that dead-but-live system - and eventually found that interrupts had not been disabled in a critical TLB programming section of code

Agenda

- Introduction
- Early challenges
- Optimizations
  - Software
    - Kernel
    - uClibc
  - Tools assisted
  - Hardware assisted
- Community

Now that we can walk, Run !

- Not all Bugs are bad !
  - Most of the early “opportunities” for optimization came straight out of staring at objdumps while debugging - without any “directed” profiling
  - Not afraid to ask “stupid” questions
- Marketing Benchmarking Requests helped the cause too
  - LMBench was/is standard measuring stick and for a hacker, the excitement of beating a competitor in a benchmark is lesser only to a few other things in life, mentionable in a public forum :-)

uaccess optimization

- Bootup init.d/rcS makes several mount calls for pseudo/real filesystems (e.g. proc/sysfs/devpts/nfs-shares..)
- We saw excessive number of D-TLB Miss Exceptions during boot, due to mount syscall copying the “strings” from userspace

```
sys_mount( ) => copy_mount_options( ) => exact_copy_from_user()

        while (n) {      /* 8192 */
              if (__get_user(c, f)) {     --> this loop continued for 8k iterations
                    memset(); break;
               }
               f++; n--;
```

- How __get_user( ) works

```
__asm__ __volatile__ (
    "1:   ld      %1, [%2]   \n"        ACTUAL COPY OPERATION
    "      mov  %0, 0   \n"             (default success case sets r0 = 0)
    "2:   nop                 \n"

    "  .section .fixup, \"ax\"  \n"
    “3:   mov  %0, %3    \n”            FIXUP code
    "      j        2b          \n"     (error case, sets r0 = -EFAULT)
    "     .previous          \n"

    "  .section __ex_table,\"a\"\n"     EXCEPTION Table Entry
    "     .word   1b, 3b        \n"     (links Copy code to Fixup code)
    "     .previous             \n"
```

uaccess: What Joy can a Half line diff bring

- The misplaced asm label for fixup caused .ex_table entry to NOT point to actual fixup code
- This caused __get_user( ) to NOT return -EFAULT when early end of page was encountered, continuing for 8k iterations.

```
172 #define __get_user_asm(x,addr,err,op) \
173     __asm__ __volatile__(                \
174     "1: "op"    %1,[%2]\n"               \
175     "   mov %0, 0x0\n"                   \
176     "2: nop\n"                           \
      "3: .section .fixup, \"ax\"\n"     \
      "   mov %0, %3\n"                    \
179     "   j   2b\n"                        \
180     "   .previous\n"                     \
181     "   .section __ex_table, \"a\"\n"    \
182     "   .align 4\n"                      \
183     "   .word 1b,3b\n"                   \
```

```
172 #define __get_user_asm(x,addr,err,op) \
173     __asm__ __volatile__(                \
174     "1: "op"    %1,[%2]\n"               \
175     "   mov %0, 0x0\n"                   \
176     "2: nop\n"                           \
      "   .section .fixup, \"ax\"\n"     \
      "3: mov %0, %3\n"                    \
179     "   j   2b\n"                        \
180     "   .previous\n"                     \
181     "   .section __ex_table, \"a\"\n"    \
182     "   .align 4\n"                      \
183     "   .word 1b,3b\n"                   \
```

- After fix, total boot time D-TLB Exceptions went down by 2 orders of magnitude: **from 87,000 to 650 !**

Signal Handling Optimization

- A userspace signal handler has to return to kernel to restore original user context using SIGRETURN syscall
- Handler return “prep”ed by setting call return register to a userspace asm stub which invokes sigreturn syscall
- In Orig Kernel code the asm stub was “synthesized”
  - “inject”ed opcodes for sigreturn trap on user stack
  - Code modification needed I$ + D$ line flush as well as Page Execute permission wiggle
- New Design
  - A “real” asm stub in uClibc, passed by sigaction() to kernel via SA_RESTORER
  - No need to muck with user stack at all

```
.globl __rt_sa_restorer;
.type __rt_sa_restorer, @function
.align 4;
__rt_sa_restorer:
      mov r8, __NR_rt_sigreturn
      swi
```

Signal Handling Optimization (cont)

- Also changed to “batch” copying of user mode registers (vs. itemized copy)

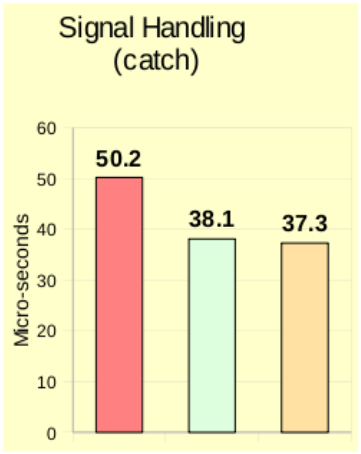
```
static int setup_sigframe()
{
  err = __put_user(r->r0, &sf->uc.mc.r0);
  err = __put_user(r->r0, &sf->uc.mc.r1);
....
 err |= __put_user(r->r12, &sf->uc.mc.r12);
```

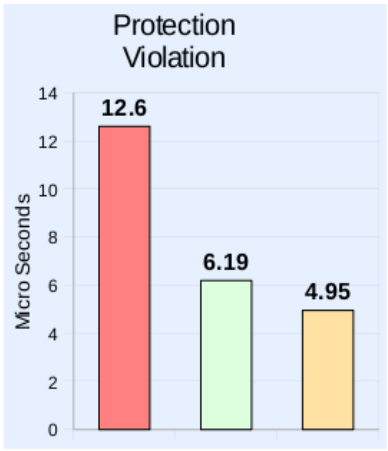
-->

```
static int setup_sigframe()
{
 unsigned long *src = &(r->bta);
 unsigned long *dst = &(sf->uc.mc.bta);
 unsigned int sz = &(r->r0) - &(r->bta) + 4;

 err = __copy_to_user(dst, src, sz);
```

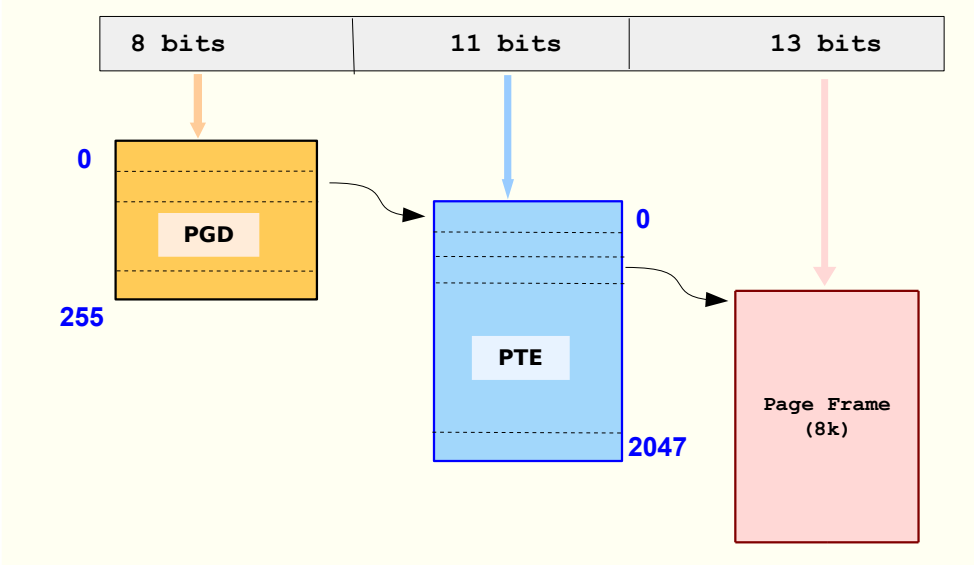
- LMBench lat_sig {catch, prot-v} improved significantly





Page Table Walking Address Split

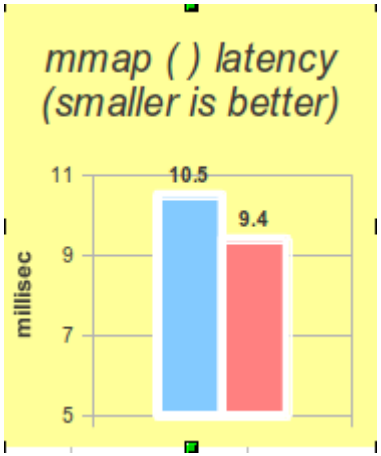
- TLB Exception Handler walks the Page Tables to find the Virtual-to-Physical mapping for creating the TLB Entry
- ARC Linux implements a 2-tier paging: PGD-Tbl and PTE-Tbl
- Indexing into each level requires vaddr to be split: originally **8:11:13** which governs the paging geometry
  - PGD-Tbl 256 entries (1KB)
  - PTE-Tbl 2K entries (8KB)



- Given page sized allocations for all tables, 7k wasted per PGD-Tbl per process

Page Table Walking Addr Split (cont)

- When a vaddr is faulted, needing a new PTE-Tbl, the entire Table has to be memzero()-ed
  - With 2k entries, as many 'ST 0, [mem]' instructions needed
  - A 2k entries deep PTE-Tbl covers 16M of vaddr space which is too coarse/large of granularity for vaddr space allocation
- Switched to **11:8:13**
  - PTE-Tbl now spans 2M of address space, still a reasonable granularity
  - A new PTE-Tbl only needs 256 instructions to initialize
  - PGD-Tbl fits w/o memory waste
- A few more minor optimizations
  - pgtable_t made ulong instead of struct page *
  - Allowed Inline memset instead of clear_page()
- lat_mmap (16k) improved by **~10%**



uClibc syscall wrappers

- Simple asm stubs which load syscall NR and args in ABI designated registers before invoking TRAP into kernel and in return possibly set “errno”
- At heart is one “C” macro with inline asm
  - Uses Recursive macro expansion
- Existing version generated lot of useless insn
- Rewrote INLINE_SYSCALL()
  - A critical reg var r0 missing
  - errno set out-of-line
  - -fomit-frame-pointer
- Total uClibc shrunk by **~5%**, while all syscall wrappers combined by **~13%**

NEW

```
00c244 <__libc_nanosleep>:
 c244:  mov    r8,162
 c248:  swi
 c24c:  brge   r0,0,c25c
 c250:  st.a   blink,[sp,-4]
 c254:  bl     ad5c
 c258:  ld.ab  blink,[sp,4]
 c25c:  j_s        [blink]
 c25e:  nop_s
```

OLD

```
00d6b0
<__libc_nanosleep>:
 d6b0:  push_s  blink
 d6b2:  st.a    r13,[sp,-8]
 d6b6:  st.a    fp,[sp,-4]
 d6ba:  mov     fp,sp
 d6be:  mov_s   r3,r0
 d6c0:  mov     r8,162
 d6c4:  mov     r0,r3
 d6c8:  swi
 d6cc:  nop
 d6d0:  nop
 d6d4:  mov     r13,r0
 d6d8:  cmp     r13,-126
 d6dc:  bls.d   d6f2
 d6e0:  mov.ls  r0,r13
 d6e4:  bl      ada4
 d6e8:  rsub    r2,r13,0
 d6ec:  st_s    r2,[r0,0]
 d6ee:  mov     r0,-1
 d6f2:  ld.ab   fp,[sp,4]
 d6f6:  ld.as   blink,[sp,2]
 d6fa:  ld_s    r13,[sp,0]
 d6fc:  j_s.d   [blink]
 d6fe:  add     sp,sp,12
 d702:  nop_s
```

Agenda

- ARC Architecture
- Early challenges
- Optimizations
  - Software
  - Tools assisted
    - gcc toggles/peephole
    - New gcc features
  - Hardware assisted
- Community

Making Toolchain work for us

- **-fomit-frame-pointer** made default across the board as it caused needless stack operations (w/o helping with debugging)
- Kernel built with **-O3** improved most LMBench numbers by ~8%
- By default GP reg not picked by Register Allocator as it is typically reserved for small data relocations, which kernel can't use. So re-purposed it as a GPR (**-fcall-used-gp**)
- Peephole for 1-bit multiply
  - page_add_file_ramp( ) accesses a 2 entry array nodes_zone[ ]
  - Indexing requires a multiply of constant with 1 bit number
  - Gcc was generating MPY instructions - which could instead be done with a TST + conditional ADD

Gcc: builtin for alignment check

- Unaligned load/store not supported in Hardware, needing additional code (Branches) to check for alignment first
- Branches bad for CPU pipelines: Mispredicts / I-Cache misses...
- If data start/end are aligned, tight inline loops could be generated specially given ARC Zero Delay Loop (ZDL) instructions
  - e.g. memzero function call requires 2 instructions to setup args, and 1 instruction for the Branch - doable inline in 3 instructions with ZDL
- New **__builtin_arc_aligned()** compile time detects alignment of data types: allowed 55% of kernel memset calls (292 out of 529) to be inlined

```
#define memzero(dst, sz)                                                    \
({                                                                          \
  if (__builtin_constant_p((sz)) && !((sz)%4) && __builtin_arc_aligned(dst, 4) ) { \
     tail_n_head_aligned_memzero(dst, sz/4);          // tight inline loop   \
  } else {                                                                  \
     extern void * memzero(void *, __kernel_size_t);                        \
     memzero(dst, sz);                          // fall back to function call \
  }                                                                         \
  dst;                                                                      \
})
```

- ~3% performance gains (pending analysis and not yet applied to things like memcpy / copy_(to|from)_user

Agenda

- ARC Architecture
- Early challenges
- Optimizations
  - Software
  - Tools assisted
  - Hardware assisted
    - New Instructions
    - Architectural Changes
- Community

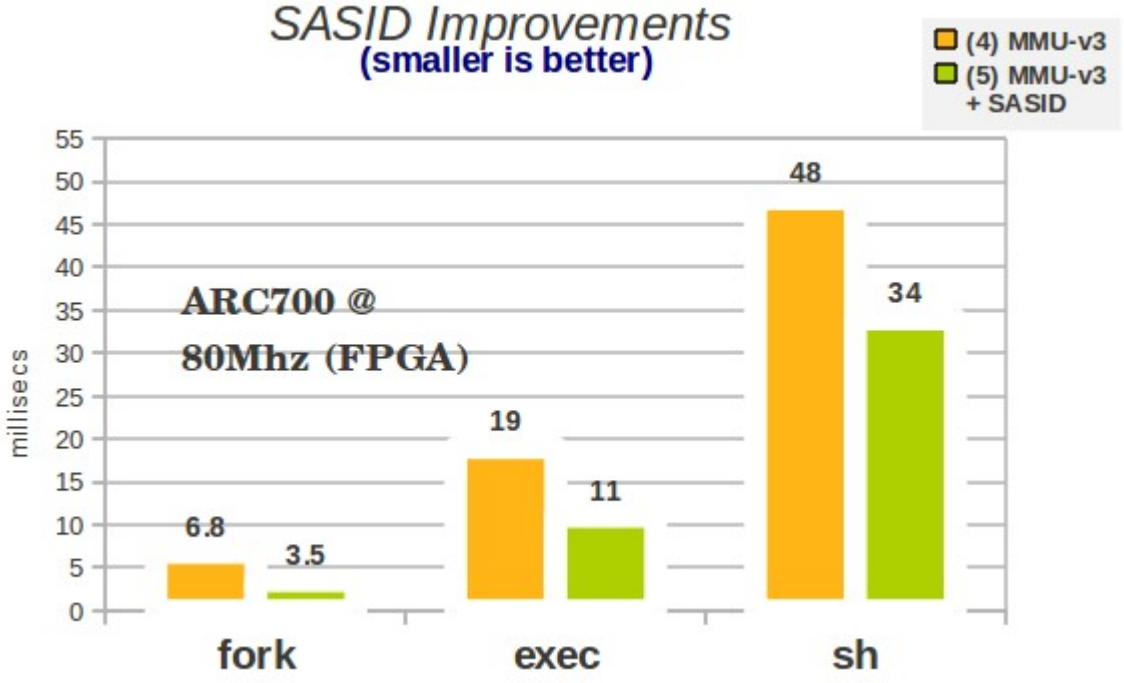
Atomic Read-Modify-Write

- As typical of a RISC architecture, ARC700 originally did not support atomic read-modify-write
  - However kernel is littered with atomic bitops: set_bit( ) / ... and ALU ops: atomic_add( ) / …
  - Linux originally disabled interrupts around such code and SMP model resorted to using additional global hashed spin lock
- ARC700 4.10 introduced LLOCK/SCOND instructions
  - A load from memory “marks” a hardware critical section, and the paired store only commits if no IRQs were taken in between. If failed, code has to retry !
  - All atomic operations converted to these instructions, kernel code reduced by ~2%

MMU Shared Address Spaces (SASID)

- Shared library code despite being “shared” at Page level, still needs per-process TLB entries (due to *ASID*)
  - e.g. System with 10 processes and 10 code pages in libc will require 100 TLB entries for libc code alone
- While *ASID* allows **segregation** of TLB entries (per task), a way to **aggregate** TLB entries (per block of code/data) is needed
  - Each sharable block of code is assigned a unique SASID (e.g. libc 1, libm 2, libpthread 3...)
  - A new type of TLB entry introduced in MMU which uses a SASID to tag TLB entries
  - Task “subscribes” to group of SASID(s): e.g. Per above, 0x5 gives it access to libc / libpthread. Currently 32 SASIDs possible
  - Only requirement from software (loader/kernel) is to map shared blocks at exact same vaddr across processes

MMU Shared Address Spaces (cont)



- Why we gain
  - New task mapping existing mapped lib is essentially a “free loader”. It reuses the existing TLB entry w/o CPU faulting. Additionally complete Linux page fault handling code is short circuited.
  - For fork, parent need to Copy-on-write it's entire address space. W/o SASID, all TLB entires (code/data) need to be invalidated. With SASID, majority of code mappings remain valid.

Agenda

- ARC Architecture
- Early challenges
- Optimizations
- Community
  - Upstreaming
  - Possible ABI Fallout

Upstreaming

- As Linux matured (stability/performance/customer-base), attention focused towards community contribution
  - Customers demand more closer following of upstream revisions
  - Miss the automatic bucket fixes which get applied to all arches (e.g. Recent signal handling updates / UAPI split)
  - geek factor of your code being in kernel.org and sharing space with smarter people
  - It's simply the right thing to do!
- Homework
  - Pretending like a new port despite being old is a serious handicap
  - Lindent / checkpatch / sparse / remove C99 comments
  - Refactored for cleaner seperation of platform / drivers / core ARCH code.
  - Flattened the existing code in 3.2 port (500+ patches since 2.6.30) and then recarved into logical patches (headers, IRQ, syscall...)
  - Read Documentation/Submitting*

Upstreaming (cont)

- Still lot of anxiety in letting “the world” loose on code which had never faced “critical” public review
  - Contacted some of the key kernel developers: Arnd Bergmann, Paul Mundt, GKH all of whom offered welcome advice
  - Arnd suggested restructuring the patches into 2 series
    - #1: Basic features to get a building/running kernel with console
    - #2: Any additional features added incrementally: ptrace, SMP, perf,....
  - Follow the last merged architecture to know common mistakes / current “trends”
    - Same image across platforms
    - Reuse the device tree bindings...
  - Don't be afraid - it's your code not you that's criticized !
  - Be prepared to fix / refix.

Generic Headers and userspace ABI

- With recent kernels, a primary recommendation/requirement for a new port is to use asm-generic headers as much as possible
- Some of the headers are mere code switch - no semantical changes, however some can cause userspace ABI change
  - e.g. generic unistd.h removes some of the syscalls, changes existing syscall numbers
  - This means older libc will NOT be compatible with new kernel
  - This may or may not be serious issue depending on existing customer base
  - Situation partially mitigated by introducing an ELF header based ABI versioning check, allowing early detection of non-compatibility
  - upstream kernel version can be used as a final checkpoint for a definitive ABI switch

```
[ARCLinux]$ /mnt/arc/ltp/testcases/bin/mmap01
  ABI mismatch - you need newer toolchain
  ABI mismatch - you need newer toolchain
  Segmentation fault
```

Summary

- Every bug is an opportunity to dig deeper and if one pays attention, most lead to one or more performance optimization(s)
- Never be afraid to ask “stupid” questions :-)
- System Optimization spans 3 areas: Software, Tools, Hardware
- ARC Linux has been a fun ride
- The second phase of fun has barely started
  - By the time I'm presenting this, the first set of kernel patches would have hopefully hit lkml and linux-arch for first review !

Thank You !

SYNOPSYS®
Accelerating Innovation