LoongArch: What we will do next
● Who we are
● What we’ve done
● What we’ll do next
● Q & A time
Who we are

• 陈华才 (CHEN Huacai) @chenhuacai
  - arch/loongarch maintainer

• 王雪瑞 (WANG Xuerui) @xen0n
  - Gentoo dev, arch/loongarch reviewer, among countless other roles
  - Proud to be that Hobbyist hanging around!
What is LoongArch?

- “a new RISC ISA, a bit like MIPS or RISC-V”
- Some numbers
  - 3 ISA subsets (LA32{R,S} and LA64)
  - 4 privilege levels (PLV0 ~ PLV3)
  - 32 GPRs, 32 FPR/VRs, 8 FCCs
- Models
  - Loongson 3A5000, 3C5000(L), 2K1000LA, 2K0500, etc.
- Further information
  - Check out the official docs
  - And don’t miss @xen0n’s unofficial FAQ
What we’ve done

- Overview of upstream status
- Current status of upstream kernel
Overview of upstream status

- Essential support mostly upstreamed
  - Done: binutils, gcc, linux, glibc, go, libffi, libunwind, systemd, etc.
  - Porting ongoing / pending reviews: LLVM, Rust, musl, libseccomp, etc.
- ELF psABI just got revised slightly incompatibly
- Overall ABI stable, multiple distros already available
  - Gentoo
  - Arch Linux (unofficial, two efforts) by @yetist and @shipujin
  - Slackware (unofficial) by @shipujin
  - CLFS (unofficial) by @sunhaidong1978
- Actually I should be presenting on a LoongArch laptop right now!
Status of upstream kernel

• Supports UEFI+ACPI systems

• Timeline
  - v5.19: Arch support & UAPI
  - v6.0: irqchip, PCI & provisional ACPI definitions
    • also vDSO getcpu etc.
  - v6.1 should mostly work OOTB!
    • Final ACPI definitions, proper EFI boot support, eBPF JIT, qspinlock, perf events
  - More to come: suspend/resume, LS7A sound, ...
What we’ll do next

- The “old world” problem
- Alternative boot protocols
- Way forward for EFI zboot flow
  - Sorted out, kudos to @ardb!
The “old world” problem

• Background: “Tale of two worlds”
• Incompatibilities
  - psABI
  - Firmware & boot protocol
  - Linux UAPI
  - Userland (libc symbol versions etc.)
• Ways forward?
Tale of two worlds

• Earliest LoongArch ports were basically copy-paste of MIPS code with mass-replaced strings
  - Rushed for non-technical reasons
  - Little gems like BogoLOONGARCH and LBT_LOONGARCH
  - Obviously this is not going to fly...

• New ABI largely modeled after that of RISC-V
  - ELF psABI and calling convention *mostly* unaffected (fortunately)
  - Other parts not so much; differences at every layer
Incompatibilities – psABI

- Relocation types
  - Stack-machine relocations in OW modeled after rl78 and rx relocations
  - Classic-style relocations in NW; transition largely complete

- ELF `e_flags[7:6]`
  - 0x1 for objects produced with very recent NW toolchains, 0x0 for OW

- Implications
  - Upstream LLVM/mold cannot understand stack relocations, and cannot be taught to do so
  - Multiple downstream projects need adaptation
Disassembly of section .text:

```
0: 00100000  move $a5, $z0
0000000000000000 <start>:
```

* each of these RELA records takes up 24 bytes

```
4: 1a000000  pcarr u121
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL _GLOBAL_OFFSET_TABLE_
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SOP_PUSH_PCREL main
4: R_LARCH_SOP_ADD +ABS+
4: R_LARCH_SLinux Plumbers Conference 2022

Relocs: before vs after

Dublin, Ireland / September 12-14, 2022

```
```
Incompatibilities - Firmware

- **UEFI tables**
  - Pointers in VA in old-world ("OW")
    - Possible rationale: it’s the same DMW config as arch/loongarch expects
  - PA in new-world ("NW") as is the case with everyone else
- **ACPI tables**
  - Different and incompatible layouts
- **Boot protocol**
  - `struct bootparamsinterface` ("BPI")
    - for OW & early iterations of NW kernel via special GRUB
  - EFI stub for NW
Incompatibilities - UAPI

• _NSIG
  - 128 in OW (same as MIPS), 64 in NW

• Syscalls
  - \{get, set\}rlimit $\rightarrow$ prlimit64
  - fstat, newfstatat $\rightarrow$ statx

• ptrace, sigcontext differences
Incompatibilities - Userland

- libc symbol version
  - GLIBC_2.27 in OW (you guessed that)
  - GLIBC_2.36 in NW
- ld.so path
  - /lib64/ld.so.1 in OW (ditto)
  - /lib64/ld-linux-loongarch-lp64d.so.1 in NW
Uniting the two worlds?

- Goal: Digital preservation, possibly by allowing **OW binaries** on **NW kernel**
  - Do we even want to go this way?
  - Layered approach if we ever decide to try
  - WINE-like approach otherwise for sanity

- **Firmware ↔ Kernel**: run **NW kernel** on either **OW** or **NW** firmware
  - Means supporting BPI upstream
  - Some early 3A5000 systems might never get updated FW; do we care?

- **Kernel & userland ABI**
  - Separate chroot/sysroot likely needed for sanity, but UX might get hurt
  - Handle the rest with userland shim / in-kernel?
How to do it if we try?

- Dividing line
  - syscall boundary / in-kernel mechanism?
  - How do we know if a process is speaking OW ABI?
    By looking at e_flags, or implied _NSIG on 1st sigprocmask call?
  - How to handle cross-world execs?

- Entrypoint
  - Via binfmt_misc: how do we identify OW binaries?
  - As ld.so replacement
    - libc symbol versioning hacks – probably not upstreamable
    - What about statically linked binaries?

- Shimming
  - Marking of ABI flavor: ptrace / personality?
Alternative boot protocols

- Why other boot protocols matter
- Possibility: BPI compatibility
- Possibility: DT boot
Why other boot protocols matter?

• Old-world/BPI compatibility
  - Some early hardware (esp. laptops) may never get NW firmware
  - Users don’t want (semi-)planned obsolescence

• Resource-constrained use cases
  - DT boot where full-fledged UEFI is too heavy
  - Do we want vanilla Linux on these devices?

• FLOSS firmware (coreboot etc.)
  - Projects & users may not want to / cannot support UEFI
  - Choices in general
Possibility: BPI compatibility

- What does a BPI boot look like?
  - UEFI present, but differently placed & with VA pointers
  - Differently shaped memory map
- Shimming
  - Again: at which layer?
  - Chain-load unmodified kernel if done before kernel
  - Effectively another EFI-stub-like entry point, if done in kernel
Possibility: DT boot

• Likely doable without much friction (unlike what’s expected for BPI)

• DT standardization
  - Both Loongson presenters are not working on DT kernel AFAIK
  - To the people working on this: Communicate, communicate, communicate!
Acknowledgements

- Obligatory thanks to my employer and Loongson
- Community power!
  - dilfridge and sam from Gentoo
  - @FlyGoat, @HougeLangley, @phorcys, @prcups, @Rabenda, @xry111, and others in the Telegram Loongson user group
  - Countless others
Thanks!

and Q & A time