montavista

XIP: the past, the present... the future?

Vitaly Wool,

Lead Engineer, MontaVista ODC

FOSDEM 2007

What the heck XIP is?

montavista

- XIP = eXecute In Place
  - The code is being executed from non-volatile storage
  - Requires linear read access
    - Effectively that means NOR flash is required
- XIP types
  - Firmware (bootloader) XIP
  - Kernel XIP
  - Application XIP
- Architectures
  - Only ARM currently
  - MIPS work ongoing
  - Makes no sense for most PowerPC- and x86-based SoCs

XIP: first steps into...

montavista

- XIP is not directly related to Linux
  - Bootloaders are often XIP
  - You're welcome to XIP whatever :)
- Firmware: something should be XIP
  - Once the hardware is powered, the control is passed to a non-volatile storage
    - Either the firmware is a complete XIP
    - Or it initializes RAM, loads the main bootloader and runs it

XIP application

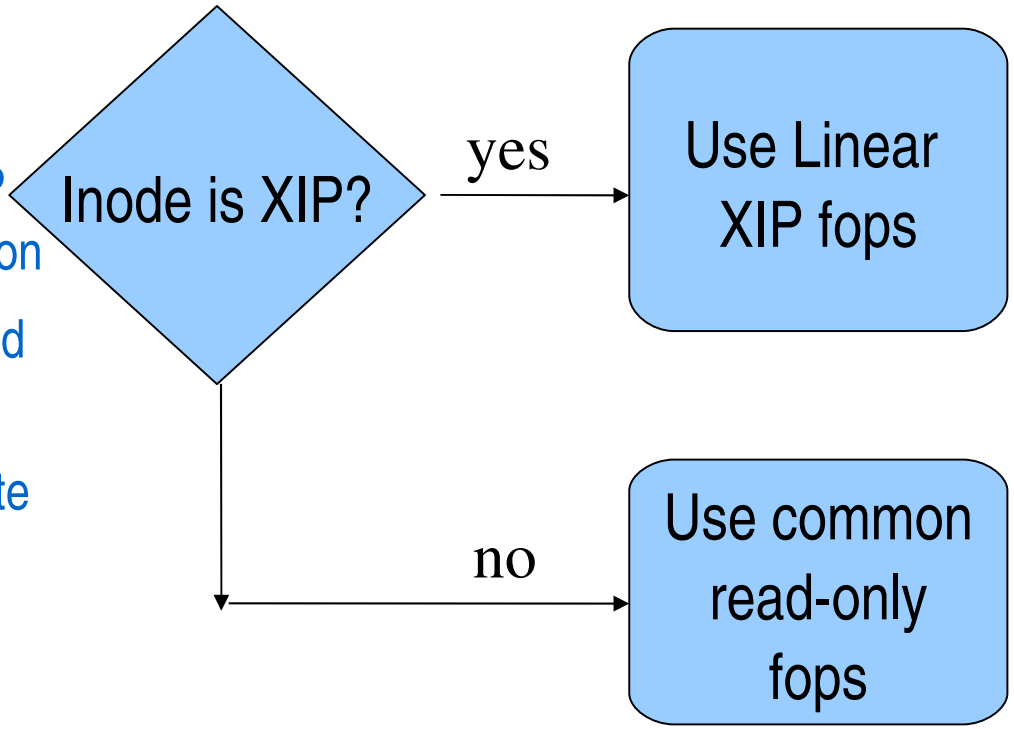
montavista

- Requirements
  - Storage that is linear on read
    - NOR flash
    - RAM (*yes, that makes sense*)
  - Implies uncompressed XIP file storage
  - Executable code stored consequently
    - Only read-only filesystem can guarantee that
    - MontaVista has XIP support for cramfs
    - Work on squashfs XIP support is in alpha stage
- Objectives
  - Decreases the RAM usage
  - Decreases the time-to-start parameter

XIP application: how it works

montavista

- HOWTO
  - Specify the executable files for XIP during the cramfs filesystem creation
  - Usually these files are distinguished by a specific flag
  - /sbin/init is usually a good candidate



  - NB: The gain depends on NOR flash speed vs RAM speed

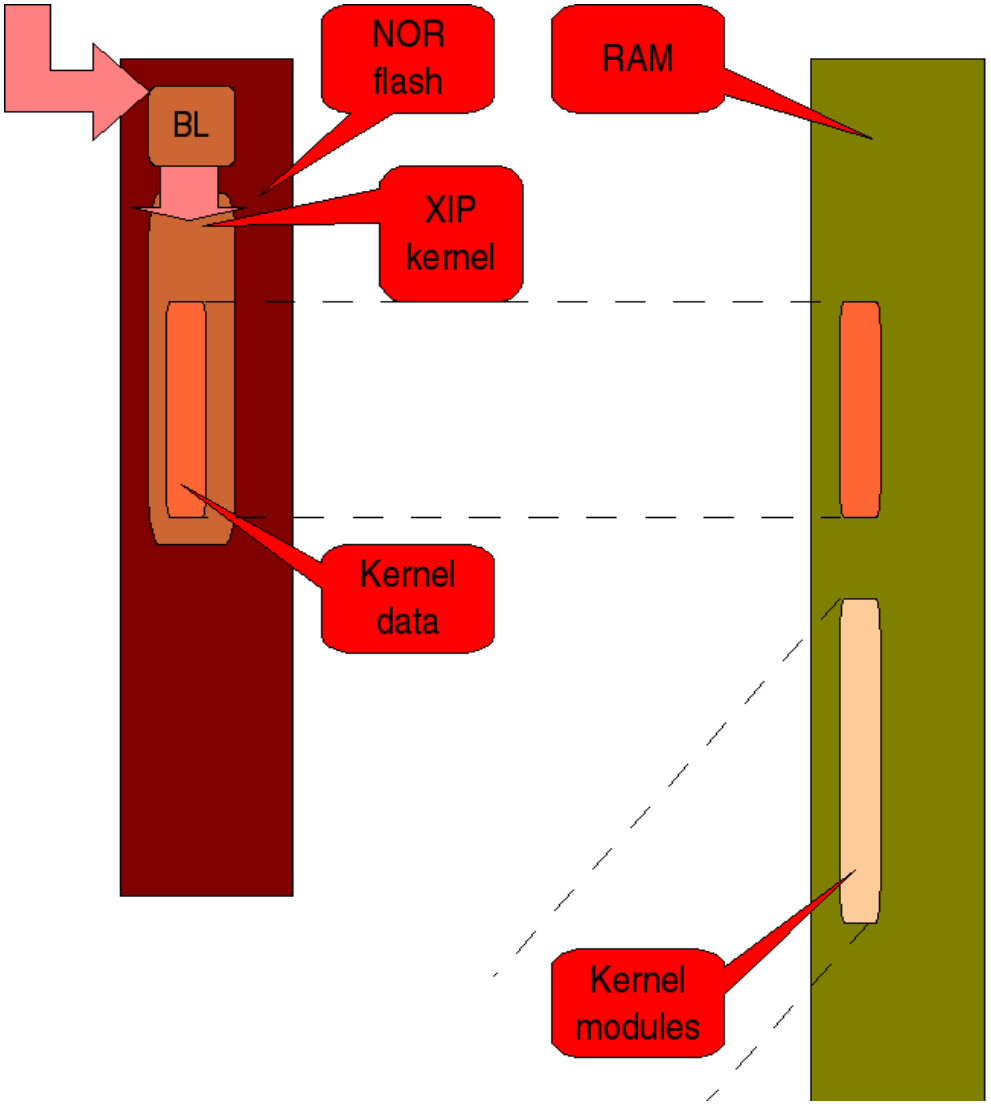
XIP kernel

montavista

- NB: The most interesting case
  - Requirements
    - Storage linear on read
  - Objectives
    - Decreases the “time to splashcreen” value
    - Decreases the overall boot time
    - Decreases the RAM usage
    - Decreases the power consumption
- MTD/XIP
  - The code can't be executed when a flash is not in array mode
    - Some MTD bits are copied to ram (along with data sections)

XIP kernel: how it works

- Bootloader gets control
  - Initializes peripherals and RAM
  - Runs xipImage
- XipImage gets control
  - copies data sections to RAM
  - Proceeds just as a common kernel
- NB: modules will be loaded to memory

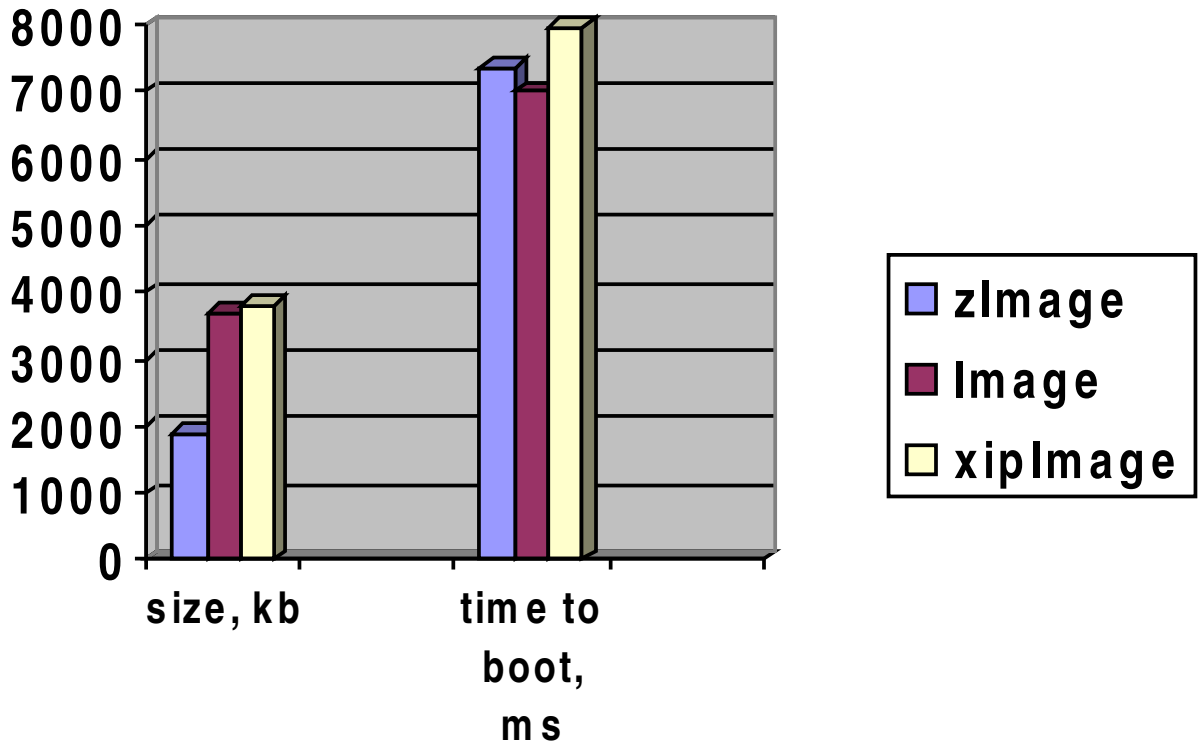


Summarizing…

montavista

- XIP pros
  - Less RAM consumption
  - Less power consumption
  - Shorter time-to-splashscreen
  - Shorter overall time-to-boot
- XIP cons
  - Needs NOR flash which is expensive
  - Takes a ***lot*** of NOR flash space
    - Code can not be stored compressed
  - Possible execution performance degradation
    - The RAM frequency is usually a lot higher than NOR flash's
    - …and timing values are also less

XIP on ARM920@180 MHz

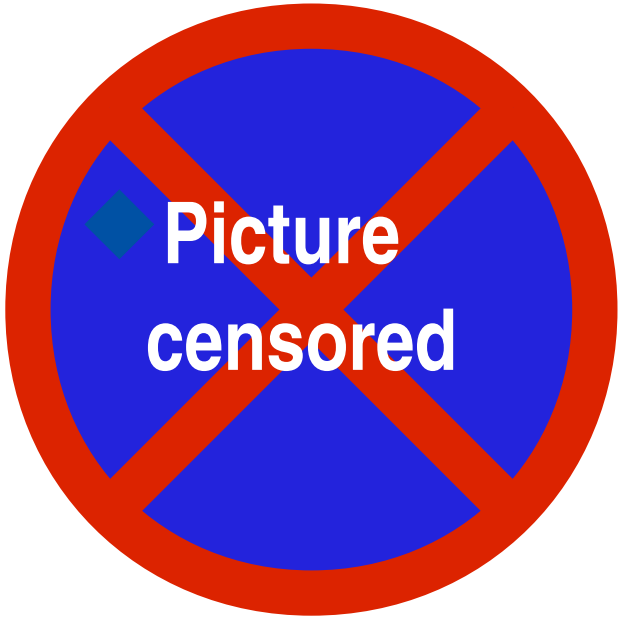


- 180-MHz ARM920EJS AT91RM9200-based custom board
- Reported by Marc Pignat

XIP on ARM920@180 Mhz: what's wrong?

montavista

- Too big a kernel
  - XIP kernel should be minimalistic, unless you're concerned only with RAM usage
  - Modules will reside and RAM and be working faster
- Slow NOR flash
  - Planing to heavily use XIP, you should make sure the NOR flash performance doesn't slow dramatically things down



Hardware objectives

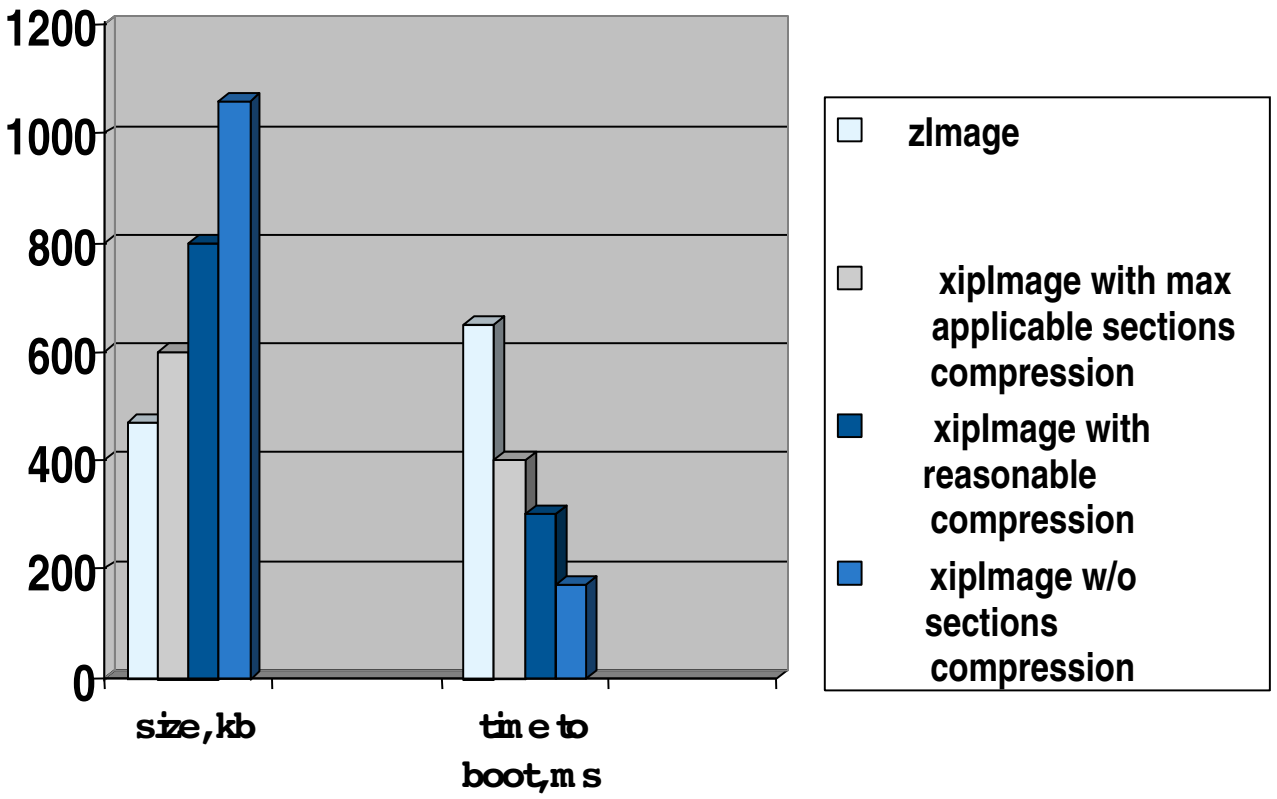
- NAND flashes get cheaper and bigger
  - HW vendors started getting rid of NOR flash in the design
- Higher CPU and RAM speeds
  - The degradation in execution speed for XIP grows
  - The boot time decreases just extensively
  - XIP needs really fine selection of built-in drivers vs modules
- So… *is there a future for XIP?*

XIP improved: partial compression

montavista

- Traditional XIP image (uncompressed)
  - Consumes a lot of NOR flash space
  - Some sections of XIP image are anyway copied to RAM (data sections)
- Partially compressed XIP image
  - Any of the data sections can be stored compressed
  - Configurability
  - Flexibility to choose between space consumption and speed

XIP/compressed on ARM926@108 MHz



- 108-MHz ARM926EJS Integrator-like board with 4M NOR

XIP improved: only XIP .init

montavista

- Traditional XIP image
  - All the code is executed directly from NOR flash
  - It causes performance degradation
  - But saves time on first stage of boot
- What if
  - Start executing from NOR and then continue from RAM?
- XIP .init
  - Start kernel thread copying kernel code to RAM
  - Run .init code

XIP improved: separate sections

montavista

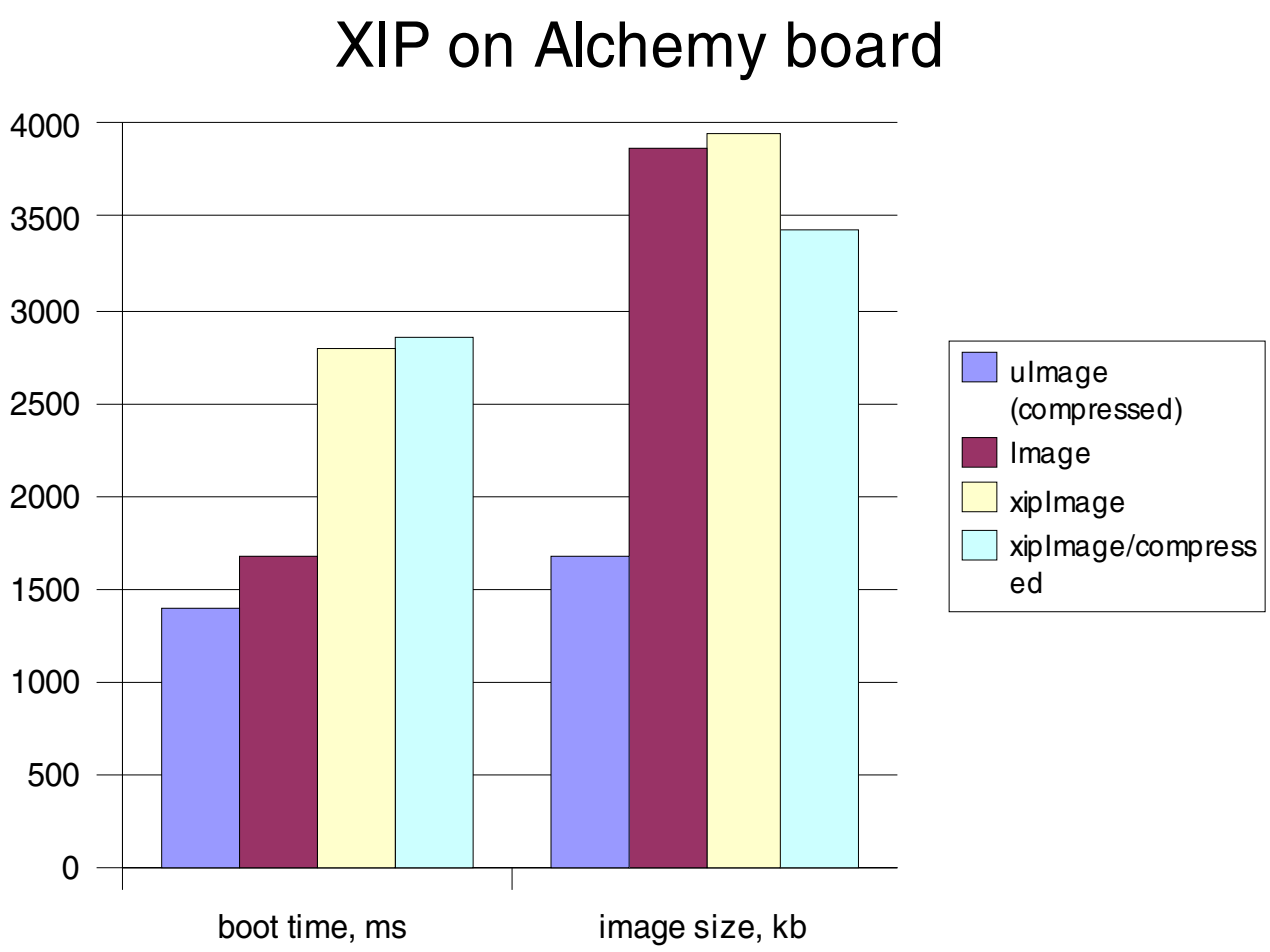
- Traditional XIP image
  - All the sections are in one image
  - Not a rational way to consume expensive NOR flash!
- For each of the two latter mods, what if store sections targeted for RAM separately
  - NAND or ATA is an ideal place
  - Especially when the device is DMA-able
  - Almost no additional CPU usage

MIPS XIP

montavista

- Traditional XIP approach
  - At the beta stage
  - Implemented by Konstantin Baidarov and Vitaly Wool
  - Verified mostly on Alchemy board
- Test board configuration
  - 400 MHz CPU Alchemy board
  - 32 MB NOR flash
  - 64 MB RAM
  - Very slow NOR flash (approx. 20 times slower on read than RAM)
- Main results
  - Wow, it works ☺
  - The performance degradation is crucial ☹

MIPS XIP: Alchemy board



- Performance degradation for XIP case is dramatic...
- But that's due to the NOR flash having read speed 17x slower than RAM

Unexpected XIP: in a ramdisk

montavista

- What if
  - The ramdisk is read-only filesystem-based
  - …which is XIP-able
  - And we mark an application as XIP?
- This app
  - Will start faster
  - Will not consume additional RAM
  - Will make the Ramdisk size bigger
- Useful for…
  - /sbin/init or any other init code

Conclusions

montavista

- Think over hardware design
  - Review smartly the requirements
  - A NOR flash? A NAND? An ATA disk?
  - Try to consider as many variants as possible at prototyping stage
- Think over software design
  - Do you need XIP at all?
    - review requirements
    - remember XIP drawbacks
  - NOR flash should be consumed basically with only XIP-able stuff
    - consider the optimizations covered in this talk