LKM Preresolver

A Lightweight Prelinker for Linux Kernel Modules

Embedded Linux Conference – Europe 2010

Carmelo Amoroso

Software Design Manager

carmelo.amoroso@st.com

Rosario Contarino

contarino.rosario@gmail.com

STMicroelectronics

www.stlinux.com

Agenda

- Fast boot & modules
  - pros & cons
- Symbols resolution
  - State of art
- Preresolver
  - Overview
  - Basics
  - Implementation
  - Pros & Cons
  - Examples & Benchmarks
- Next steps

Speed-up boot time

- Kernel bootup time is a hot topic
  - for embedded systems
  - for laptop and netbooks
  - and for desktop system as well

- Modular kernels is becoming an valuable option
  - There are pros and cons
  - Trade-off between boot-time and runtime optimizations

Using modules

- Modules allow to boot with a *thin* kernel
  - Thinner kernels are faster to boot
  - Fit better on small boot flash device

- Defer initialization of some device drivers
  - Move such drivers to be “*modules*”
  - Beneficial for many drivers which are not needed to boot the system
    - Ethernet (PHY reset) can take some time
    - USB initialization is very slow
    - SATA HDDs can take a lot of time
  - ***Recall, all driver initialization will take some time.***

Using modules (cont'd)

- Moving to a module has impacts anyway
  - It will require some time for module loading for
    - memory allocation, copying from user space
    - symbols resolution
  - Trade-off between boot-time and run-time!
- In summary
  - For best **boot-time**, modularize many device drivers
  - For best **run-time**, use no (few) modules

- We will focus on optimizing modules loading
  - ***In particular the symbol resolution process***

Symbols resolution: state of art

- Symbol resolution process is an avoidable step
  - It is time consuming, never optimised
- Until last year almost nobody took care of it !!!

- At ELC-E 2009 a new strategy was proposed
  - Using hash table for speeding-up symbol resolution
  - Based on SysV hash table used in C runtime library
  - With further optimisations to pre-compute hash values
  - Implemented in STLinux kernel (git.stlinux.com)
  - ***Unfortunately not up-streamed up to now !!!***

Symbols resolution: state of art

- Another solution was developed @ ST again
  - Based on GNU hash table
  - No changes required into the Kernel Symbol Table
  - Added optional Bloom Filtering
  - Coming soon in STLinux kernel
  - ***Hopefully it will be up-streamed soon***

- Another solution was proposed by A. Jenkins
  - Based on a binary search in the kernel symbol tables
  - Proposed on the LKML in Oct / Nov 2009
  - Follow http://lkml.org/lkml/2009/11/2/289

Preresolver: overview

- LKM Preresolver is another solution to speed-up module loading
  - It is a software tool working at kernel build time
  - It is based on the standard concept of prelinking
  - The goal is to perform the **resolution** of the undefined symbols at build time
  - But it does not perform all symbol **relocations**
  - It is fully compatible with any module loader implementation

- **So it is a sort of *lightweight* prelinker**

Preresolver: basics

- vmlinux is a statically linked binary
- Kernel symbols addresses are absolute
  - They are accessible from the ELF by inspecting the kernel symbol tables
- Kbuild guarantees that there are no duplicated exported symbols

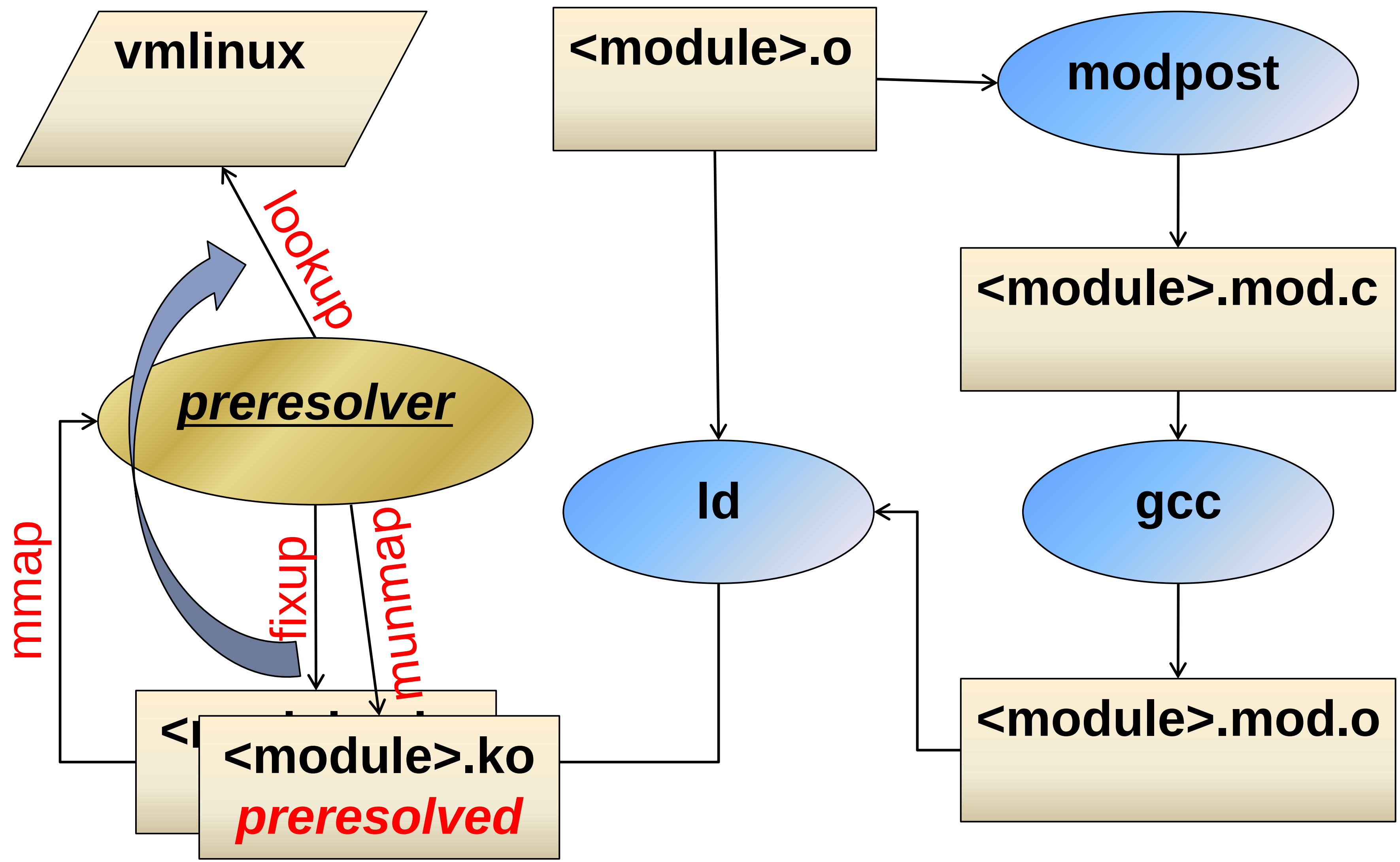
Preresolver: strategy

- Lookup undefined symbols into the kernel symbol tables
- Update the LKM by fixing up the symbol table with the proper values
  - Preresolved symbols are marked as SHN_ABS
  - Symbols not resolved into the vmlinux are kept unchanged (SHN_UND)
- Mark the module as preresolved
  - By adding an empty ELF section (.preresolved)

Preresolver: pros & cons

- Benefits
  - Less undefined symbols to be resolved at load time
  - In some case, 100% of symbol resolution can be optimized away!
    - When all undefined symbols are exported by the vmlinux
  - Module loader can be instructed to perform symbol lookup in the loaded modules only
    - No symbols are expected to be resolved in the kernel symbol tables
- Drawbacks
  - Preresolved modules cannot be used with different kernels

Preresolver: flow



Preresolver: examples (1)

- Examples using a kernel 2.6.32.16 (82 modules)

```
LD      vmlinux
... [SNIP] ...
Building modules, stage 2.
MODPOST 82 modules
...[SNIP] ...
CC      drivers/ata/libata.mod.o
LD [M]  drivers/ata/libata.ko
PRERESV drivers/ata/libata.ko (104/104 *fully* preresolved)
CC      drivers/ata/pata_platform.mod.o
LD [M]  drivers/ata/pata_platform.ko
PRERESV drivers/ata/pata_platform.ko (11/26 preresolved)
CC      drivers/ata/sata_stm.mod.o
LD [M]  drivers/ata/sata_stm.ko
PRERESV drivers/ata/sata_stm.ko (20/41 preresolved)
CC      drivers/hid/usbhid/usbhid.mod.o
LD [M]  drivers/hid/usbhid/usbhid.ko
PRERESV drivers/hid/usbhid/usbhid.ko (59/74 preresolved)
CC      drivers/i2c/i2c-core.mod.o
LD [M]  drivers/i2c/i2c-core.ko
PRERESV drivers/i2c/i2c-core.ko (47/47 *fully* preresolved)
```

Preresolver: examples (2)

- Inspecting the usbcore.ko .symtab

```
readelf -s usbcore.ko | grep UND | grep printk$
725: 00000000     0 NOTYPE    GLOBAL DEFAULT UND printk
```

- Inspecting the .symtab after the Preresolver

```
readelf -s usbcore.ko | grep printk$
727: 80192b44     0 NOTYPE    GLOBAL DEFAULT ABS printk
```

- Inspecting the vmlinux symbol table, we have

```
readelf -s vmlinux | grep \ printk$
14176: 80192b44  40 FUNC      GLOBAL DEFAULT 2 printk
```

  - The printk is actually resolved with the absolute address
  - The symbol type is changed

Preresolver: figures (best cases)

	Undefined symbols			Undefined symbols	
Module	Std	Preres	Module	Std	Preres
xfs.ko	242	1	ntfs.ko	95	0
ext4.ko	276	39	jbd2.ko	87	0
nfs.ko	243	44	hostap.ko	88	2
sunrpc.ko	188	0	mmc_core.ko	77	0
jfs.ko	166	0	lockd.ko	104	34
usbcore.ko	163	0	snd.ko	67	1
cifs.ko	152	0	squashfs.ko	61	0
fat.ko	108	0	pegasus.ko	68	9
ide-core.ko	108	0	usbhid.ko	74	15
libata.ko	104	0	usb-storage.ko	75	16
jffs2.ko	110	6	mmc_block.ko	59	7
smbfs.ko	103	0	usbnet.ko	61	10

Preresolver: figures (worst cases)

	Undefined symbols	
Module	**Std**	**Preres**
sata_stm.ko	41	21
net1080.ko	22	11
pata_platform.ko	26	15
aead.ko	22	13
cdc_ether	17	11
chainiv.ko	18	14
ecb.ko	13	12
arc4.ko	2	2
cdc_subset.ko	6	6
aes_generic.ko	2	2

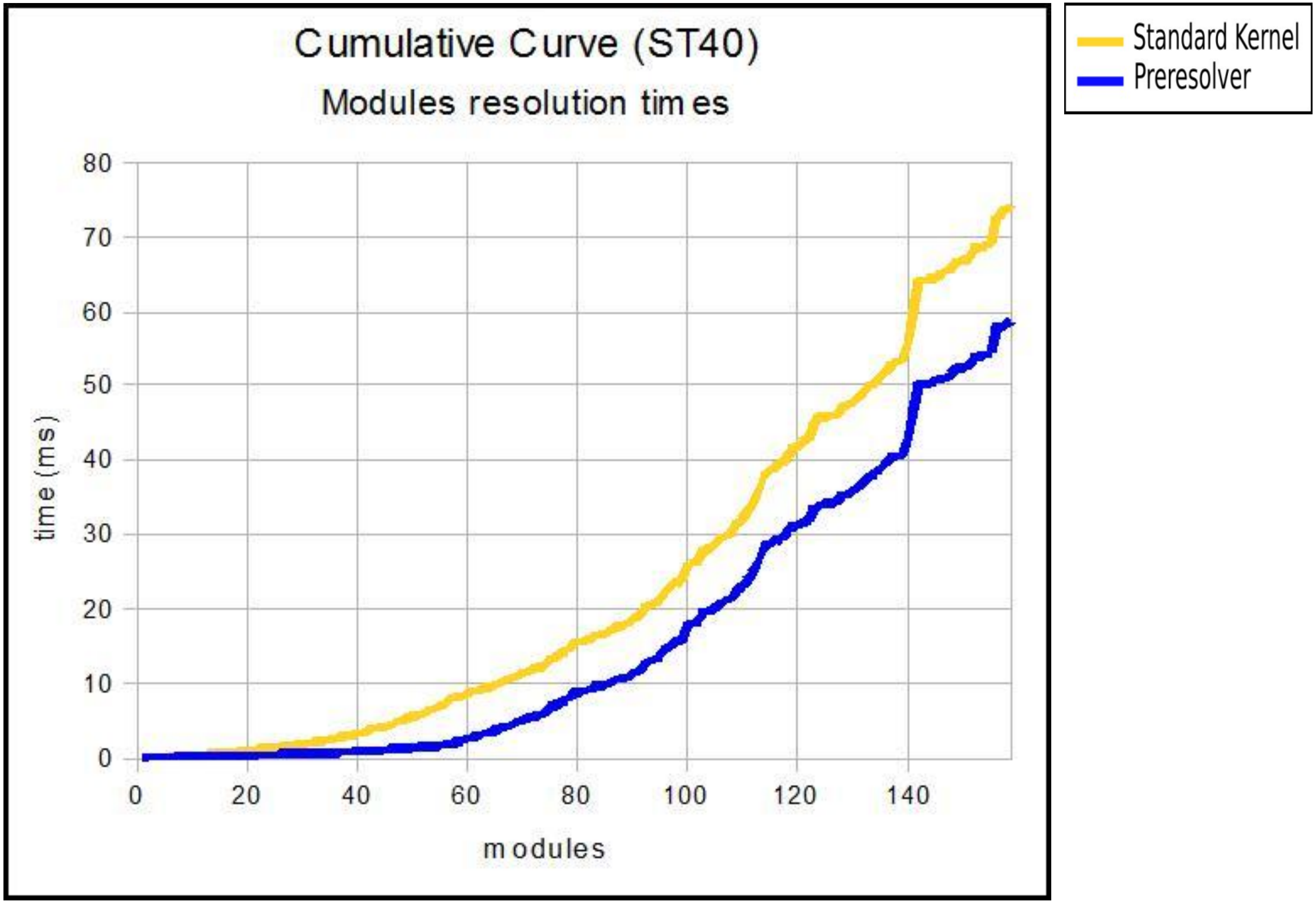
Preresolver: benchmarks (1)

- Scenario 1: **Embedded Set-Top-Boxes**
  - ST40 cpu (SH-4 based)
  - Running STLinux distro w/ kernel 2.6.32.16
    - GNU hash loader was used
    - 160 modules loaded/unloaded sequentially

Benchmarks		
160 Modules	**Gain (times)%**	
No. deps	***Symbol resolution***	***Module loading***
0	**-68,53%**	**-0,89%**
1	**-17,25%**	**-0,47%**
> 1	**-15,85%**	**-0,45%**

Times measured using gettimeofday
*Gain is ((new-old)/old)*100*

Preresolver: benchmarks (2)



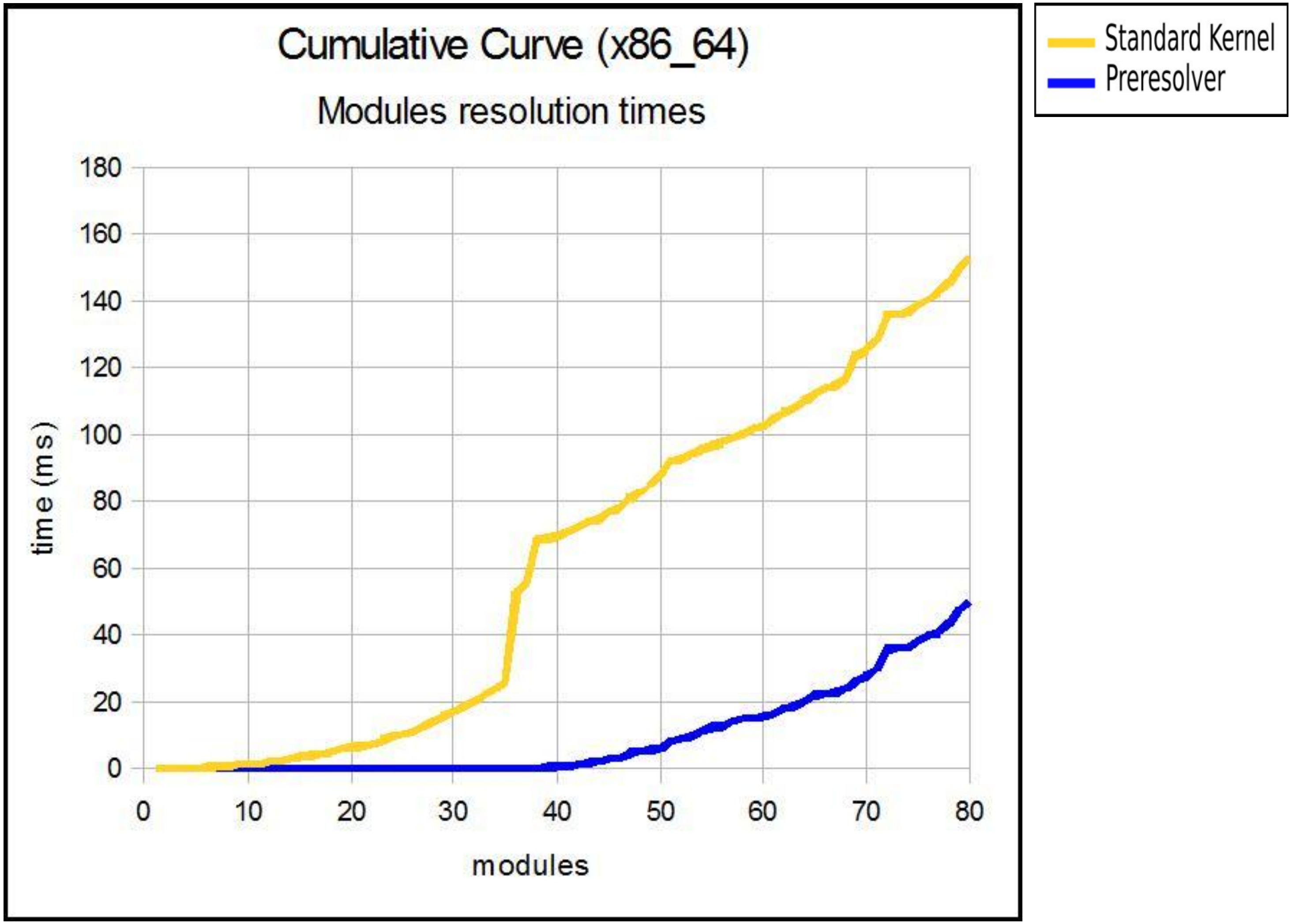
Preresolver: benchmarks (3)

- Scenario 2: **laptop/desktop systems**
  - Based on Intel x86_64 cpu
  - Running ArchLinux distro w/ kernel 2.6.33.3
    - 84 modules (default), “modprobed”

Benchmarks		
84 Modules	**Gain (times) %**	
No. deps	***Symbol resolution***	***Module loading***
0	**-98,72%**	**-50,55%**
1	**-69,08%**	**-48,22%**
> 1	**-32,08%**	**-30,05%**

Times measured using gettimeofday
*Gain is ((new-old)/old)*100*

Preresolver: benchmarks (4)



Next steps

- **Trying to upstream !!!**

- Further optimisations
  - Strip down kernel by removing kernel symbol tables
    - Works only with preresolved modules
    - Acceptable ? In an embedded scenario likely yes
  - Direct binding
    - Use dependencies information in .modinfo section
    - Perform lookup into a subset of modules
- Merge all together (GNU hash, Preresolver, Direct binding) for ***fastest module loading***

Thanks for your attention