



SmartNICでMilvusVisorを 動かす話

2024/11/09(土) Kernel/VM探検隊@北陸 Part 7
産業技術総合研究所 超分散コンピューティング研究チーム 森 真誠



本発表の内容

本発表ではMilvusVisorをSmartNIC(AMD Zynq UltraScale+ MPSoCs)で動作させる過程で遭遇した問題やその解決方法について説明

注意: 本発表での意見・感想は個人的なものであり、組織を代表するものではありません



MilvusVisorとは

Armv8-A以降の実行モードであるAArch64で動作するThin Hypervisor

- Thin Hypervisor: OSを一つしか動かさない代わりにデバイスの仮想化しないHypervisor
 - そのため動作が軽量で性能劣化が少ない
- MilvusVisorではOSの挙動に制限をかけたり、環境を復元する機能を搭載

GitHub: <https://github.com/RIKEN-RCCS/MilvusVisor>

v1.4.1までを公開中



背景

- [Kernel/VM探検隊@北陸 Part 6](#)で「AArch64 ThinHypervisor 開発記録」という内容でArmデバイスのデモ話をしました
- [2024年2月OS研究発表会](#)で「I/Oデバイス内部の汎用プロセッサにおける軽量の隔離実行環境」の題目で発表をしました

SmartNICに処理をオフロードする過程において、「各タスクを隔離するための手法」として「軽量のハイパーバイザ」を導入する研究を行っています

詳細は論文をご確認ください

SmartNICについて

提案手法を実装するためにNVIDIA BlueField-2というSmartNICに実装

NVIDIA BlueField-2 DPUの仕様

- Arm Cortex-A72 x8 搭載
 - NICの中でCPUが動作
- Generic Timer + Generic Interrupt Controller version 3
- UEFI BIOS + ACPI
- SmartNICの中にPCI Express搭載
 - SmartNICの中のCPUからホストと別のPCI空間が見える
 - ConnectX-6 のネットワークカードが見える



提案と設計・実装(ざっくり)

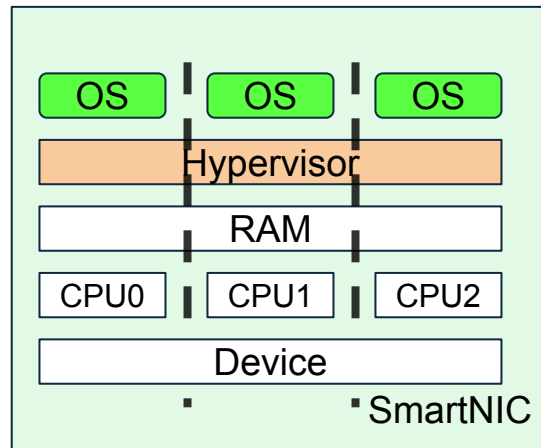
動機: SmartNICで処理を行う際に軽量の隔離実行環境がほしい


提案: 論理分割と隔離実行

- 論理分割: **仮想デバイスをつくらず**に、物理デバイスを論理的に分割
 - CPUはコアごと、メモリは区画ごと、デバイスは複数口を作成
- 分割したデバイスをそれぞれのOSに割り当てて複数環境を動作
- 隔離: 各OSが割り当てられたリソース以外にアクセスしようとした際にHypervisorでブロック

実装: MilvusVisorを拡張して論理分割と隔離実行環境を実装

詳細は論文をご確認ください





BlueField-2で論理分割

提案手法をMilvusVisorを改造してBlueField-2上で実装しようとした



BlueField-2はネットワークデバイスが**分割できなかった**

- 複数のネットワークデバイスを作成できなかった
- SR-IOVみたいなのを期待していた..



各環境と通信するための**仮想デバイス**を実装するはめに(本末転倒)



Virtio-Netデバイスを実装しようとした結果

既存のPCIバスやGICv3(割り込みコントローラ)に寄生する形で実装しようとした結果、
前回の発表のような問題に遭遇しました



PCI Base Address Registerは 絶対アドレスじゃ無かった話

↑BlueField-2で遭遇した問題→



Arm用の割り込みコントローラ
GICv3とPCIのITS-MSI割り込みについて

今年の方針

SmartNICを変更し「XILINX ALVEO U25」と呼ばれるFPGA + Arm CPUのSmartNICに変更

XILINX ALVEO U25の仕様

- Zynq UltraScale+ XCU25 FPGA
- Arm Cortex-A53 x4
- Generic Interrupt Controller version 2
- U-BootとLinuxが動くらしい
 - 公式ドキュメント曰く



U-Boot + Linuxが動いたら ...
MilvusVisorも動くやろ (適当)

~~Of course it runs MilvusVisor~~ (小声)



起動方法

現状

1. FSBL(First Stage Boot Loader)が起動(EL3)
2. FreeRTOSが起動(EL3)

方針

1. FSBLが起動(EL3)
2. U-Bootが起動(EL2)
3. MilvusVisorが起動(EL2)
4. FreeRTOSが起動(EL1)

EL0

Application

EL1

OS

EL2

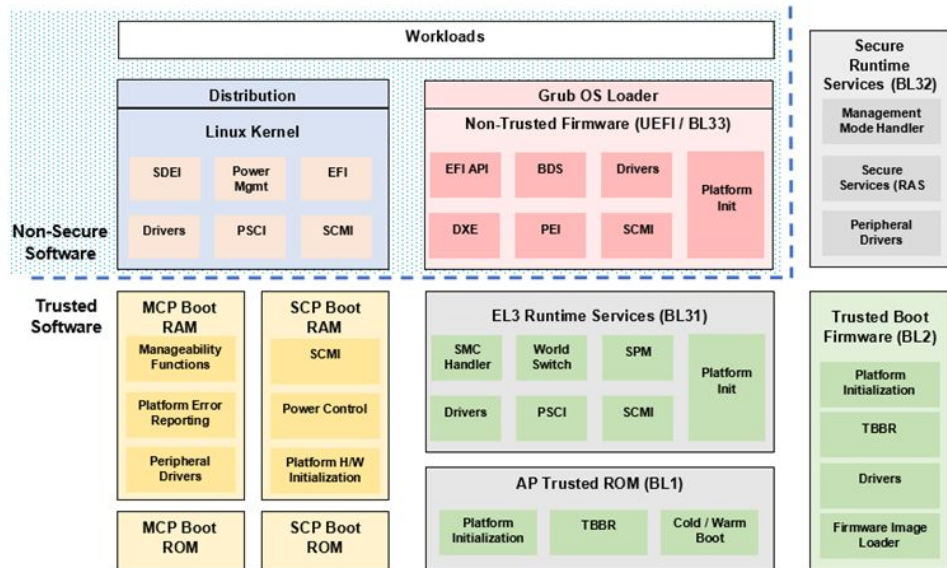
HyperVisor

EL3

Firmware

Armのブートまわり

- First Stage Boot Loader(BL1)
 - プラットフォーム固有の初期化
- Arm Trusted Firmware(BL31)
 - Secure Monitor Callなどをハンドル
 - PSCIをハンドル
- U-Boot/EDK2等(BL33)
 - OSをロード
 - 今回はU-BootのUEFI(EBBR)を使用



<https://developer.arm.com/documentation/102337/0000/Software-stack/About-the-software>



PSCI: Power State Coordination Interface

Armの電源管理インターフェース

- シャットダウンやリブートなどを行う
 - ACPIと違ってとっても簡単
- コアの起動や停止も管理
 - Local APICと違ってとっても簡単
- Secure Monitor Callを発行することで実行
 - 下にファームウェアが動作していることが条件となる

PSCIの実態はArm Trusted Firmware(ATF)を呼び出すことらしい

MilvusVisorはコアの起動にPSCIを使用しているため、ATFは必須ということになる



起動順序

1. FSBLが起動(EL3)
2. ATFが起動(EL3) ←追加
3. U-Bootが起動(EL2)
4. MilvusVisorが起動(EL2)
5. FreeRTOSが起動(EL1)

以上の順に起動するようにバイナリを準備・スクリプトを修正



ZynqMP用のFSBL・ATF・U-Bootのビルド

公式リポジトリ等のコードを自前でビルドしてみたものの動作せず。(起動不可・DTBロード失敗等)



公式ドキュメントではPetaLinuxで関連バイナリをビルドしていたため、それに倣うことに

PetaLinux(Tools)

- Linuxをビルド・実行するために必要なソフトウェア群
 - コンパイラ・ドライバ・ブートローダ等
- Yocto Projectみたいなもの
 - PetaLinuxはYocto Projectをベースにしているみたい[1]

1: <https://adaptivesupport.amd.com/s/question/0D52E00006iHwoTSAS/project-yocto-vs-petalinux>



petalinux-configでの修正

Arm Trusted FirmwareとU-Bootの設定の修正

一回 petalinux-config を走らせて設定ファイルを生成

PetaLinux自体の設定の以下を編集(各々のバイナリが衝突しないように)

- CONFIG_SUBSYSTEM_PRELOADED_BL33_BASE
- CONFIG_SUBSYSTEM_ATF_MEM_BASE

U-Boot関連の.configを編集

- ロードされるアドレスの修正
- EFI関連の機能の有効化

U-Bootのパッチ

一度 petalinux-build を走らせると一連のバイナリがビルドされる

バージョンなどによってはリロケーションエラーが発生する可能性があるためパッチを当てて修正

AArch64 アセンブリで「#:lo12:LabelName」と記述すると
LabelName の下位12ビットの値が取り出せるらしい

(AArch64 では命令長の関係で1命令でアドレスが
ロードできない場合がありその場合に便利)

```
diff --git a/arch/arm/cpu/armv8/start.S b/arch/arm/cpu/armv8/start.S
index 99d126660d..ea9dcf93c6 100644
--- a/arch/arm/cpu/armv8/start.S
+++ b/arch/arm/cpu/armv8/start.S
@@ -67,8 +67,10 @@ pie_fixup:
    adr x0, __start      /* x0 <- Runtime value of __start */
    ldr x1, __TEXT_BASE  /* x1 <- Linked value of __start */
    sub x9, x0, x1       /* x9 <- Run-vs-link offset */
-   adr x2, __rel_dyn_start /* x2 <- Runtime &__rel_dyn_start */
-   adr x3, __rel_dyn_end   /* x3 <- Runtime &__rel_dyn_end */
+   adrp x2, __rel_dyn_start /* x2 <- Runtime &__rel_dyn_start */
+   add x2, x2, #:lo12:__rel_dyn_start
+   adrp x3, __rel_dyn_end   /* x3 <- Runtime &__rel_dyn_end */
+   add x3, x3, #:lo12:__rel_dyn_end
```


Boot.binの中身

bootgen(Vivadoのコマンド)でBIFファイルを使用して
Boot.binを作成した後、Boot.binの中身を見える

Debian系の”u-boot-tools”の”dumpimage -l”を使用すると
ファイル構造を確認可能

ファイル内での各ファイルのオフセットやサイズが
わかる

```
Image Type   : Xilinx ZynqMP Boot Image support
Image Offset : 0x00002800
Image Size   : 86488 bytes (86488 bytes packed)
Image Load   : 0xffffc0000
Checksum     : 0xfd1b8891
Modified Interrupt Vector Address [0]: 0x14000000
Modified Interrupt Vector Address [1]: 0x14000000
Modified Interrupt Vector Address [2]: 0x14000000
Modified Interrupt Vector Address [3]: 0x14000000
Modified Interrupt Vector Address [4]: 0x14000000
Modified Interrupt Vector Address [5]: 0x14000000
Modified Interrupt Vector Address [6]: 0x14000000
Modified Interrupt Vector Address [7]: 0x14000000
Custom Register Initialization:
  @ 0xffff41a040 -> 0x00000003
FSBL payload on CPU a5x-0 (PS):
  Offset      : 0x00017a00
  Size        : 262216 (0x40048) bytes
  Load        : 0x00000000
  Attributes  : EL3
  Checksum    : 0xffffc9981
FSBL payload on CPU a5x-1 (PS):
  Offset      : 0x00057a80
  Size        : 237640 (0x3a048) bytes
  Load        : 0x20000000
  Attributes  : EL3
  Checksum    : 0xbffbe040
```



U-BootからMilvusVisorを起動

起動後、U-Bootがプロンプトを出すため以下のコマンドを実行

1. `setenv filesize (BOOTAA64.EFIのサイズ)`
 - **これがないとbootefiが失敗する(本来はファイル read時に自動セットされる)**
2. `sf probe 0 0`
3. `sf read $kernel_addr_r (Boot.binでのBOOTAA64.EFIの位置) $filesize`
 - QSPI Flashからread
4. `bootefi $kernel_addr_r`

これでMilvusVisorのhypervisor_bootloader部分は起動します



MilvusVisorの起動シーケンス

1. BOOTAA64.EFI(hypervisor_bootloader)が起動
2. デバイスの検知
3. **起動したデバイスから hypervisor_kernelをロード**
4. hypervisor kernelを起動
5. hypervisor_kernelがハイパーバイザ権限の設定やデバイスハンドラなどをセットアップ
6. EL2からEL1に移行
7. UEFIに制御を返還

UEFIのEfiFileProtocolやPxeProtocolなどを用いてhypervisor_kernelを取得しているが、
Alveo U25ではSDカードなどは無くQSPI FlashのみでEfiFileProtocolが使えない



回避策

1. U-Bootが認識できるようなFATファイルシステムをどうにか構築する
 - かなり面倒そう
2. FTPなどで読み込めるようにする
 - 実験環境やU-Bootを改変する必要あり、めんどくさい
3. U-Bootで予めメモリにロードしておく
 - 読み込む先のメモリアドレスの確保やリロケーションがやや面倒

最終奥義:

bootloaderにkernelを埋め込む



embed_kernel

MilvusVisor version 1.4.1から登場

- 先にhypervisor_kernelをビルド
- hypervisor_bootloaderをビルド時に“include_bytes!”でkernelを [u8](配列)として読み込む
- あとはEfiFileProtocolの時と同様

MilvusVisor version 1.4.1ではEfiFileProtocolとPxeProtocol、及びembed_kernelの共通部分を整理
(各固有部分をCallback関数として分離し、コア機能を一つの関数にした)

MilvusVisor on ALVEO U25

ここまでの手順でようやくMilvusVisorが動くように🎉

```
Xilinx Zynq MP First Stage Boot Loader
Release 2020.2   May 13 2024 - 12:17:54
NOTICE: ATF running on XGZUUNKN/silicon v4/RTL5.1 at 0x40000000
NOTICE: BL31: v2.2(release):xlnx_rebase_v2.2_2020.3
NOTICE: BL31: Built : 01:55:51, Jul  1 2024
```

```
U-Boot 2020.01 (Jul 01 2024 - 01:28:46 +0000)
```

```
Board: Xilinx ZynqMP
DRAM:  4 GiB
trace: enabled
PMUFW: v1.1
EL Level:      EL2
Chip ID:       unknown
NAND:  0 MiB
MMC:
In:     serial@ff000000
Out:    serial@ff000000
Err:    serial@ff000000
Bootmode: QSPI_MODE
Reset reason: EXTERNAL
Hit any key to stop autoboot: 0
```

```
Warning: SPI speed fallback to 100 KHz
SF: Detected n25q00a with page size 256 Bytes, erase size 64 KiB, total 128 MiB
device 0 offset 0x400000, size 0x80000
SF: 524288 bytes @ 0x400000 Read: OK
QSPI: Trying to boot script at 0x20000000
## Executing script at 20000000
SF: Detected n25q00a with page size 256 Bytes, erase size 64 KiB, total 128 MiB
device 0 offset 0x152740, size 0x20400
SF: 132096 bytes @ 0x152740 Read: OK
device 0 offset 0x172b40, size 0x183218
SF: 1585688 bytes @ 0x172b40 Read: OK
Found 0 disks
MilvusVisor Bootloader Version 1.5.0(c63fa5e114a3100c16a7aa58d4bb075dafaf4475)
Compiler Information: rustc 1.84.0-nightly (c1db4dc24 2024-10-25)
Allocated 0x67D6F000 ~ 0x77D6F000
Reading the embedded hypervisor_kernel
Call the hypervisor(Entry Point: 0x7FC0004168)
MilvusVisor Kernel Version 1.5.0(c63fa5e114a3100c16a7aa58d4bb075dafaf4475)
Compiler Information: rustc 1.84.0-nightly (c1db4dc24 2024-10-25)
```

ここまでの手順を自動化

boot.scrを作成し、U-Boot起動時に自動実行してもらうようにする

1. 一旦空のboot.scrを作成
2. boot.bifにboot.scrを入れる
 - “offset”はU-Bootのconfigと揃える
3. bootgenなどでboot.binを作成

```
the_ROM_image:
{
    [bootloader]» » » » » » » » zynqmp_fsbl.elf
    [trustzone,exception_level=el-3]» » » » bl31.elf
    [destination_cpu=a53-0, exception_level=el-2]» u-boot.elf
    [pmufw_image]» » » » » » » pmufw.elf
    partition
    {
        » id=0x01,
        » partition_owner=u-boot,
        » file=B00TAA64.EFI
    }
    partition
    {
        » id=0x03,
        » offset=0x00400000,
        » partition_owner=u-boot,
        » file=boot.scr
    }
}
```

boot.scrの自動作成

dumpimageの出力を加工してMilvusVisorの
ロードコマンドを自動作成

これでboot.scrを作成し、再度bootgenを
実行



自動的にMilvusVisorが起動するように

```
#!/bin/bash

PAYLOAD_INFO=(`dumpimage -l boot.bin | grep -A 2 'U-Boot payload'`)
BOOT_OFFSET=${PAYLOAD_INFO[8]}
BOOT_SIZE=`echo ${PAYLOAD_INFO[12]} | sed 's/[\\(\\)]//g'`
GUEST_OFFSET=${PAYLOAD_INFO[23]}
GUEST_SIZE=`echo ${PAYLOAD_INFO[27]} | sed 's/[\\(\\)]//g'`

echo setenv filesize $BOOT_SIZE > boot.txt
echo sf probe 0 0 0 >> boot.txt
echo sf read \\$kernel_addr_r $BOOT_OFFSET \\$filesize >> boot.txt
echo sf read \\$ramdisk_addr_r $GUEST_OFFSET $GUEST_SIZE >> boot.txt
echo bootefi \\$kernel_addr_r >> boot.txt
mkimage -A arm64 -T script -C none -d boot.txt boot.scr
```



ところで...(余談)

MilvusVisorの公式リポジトリではGitHub Actionsで毎日Nightly BuildのrustcでMilvusVisorのコンパイルが走っています

理由:

MilvusVisorはRustのunstableな機能を使用していて、
それらの仕様変更で突然ビルドがコケる可能性があるため



最近のGitHub Actions

毎日13:30の時報を送るメールボットと化した
GitHub Actions君→

✖ Build Hypervisor	Build Hypervisor #205: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #204: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #203: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #202: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #201: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #200: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #199: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #198: Scheduled	main
✖ Build Hypervisor	Build Hypervisor #197: Scheduled	main



最近のMilvusVisorのビルドがコケている理由

- naked_functionの安定化のために、#[naked]の付いている関数内ではasm!からnaked_asm!を使用するように変更が入った
- Rust Edition 2024にむけてmutable referenceの仕様変更が色々入っている
 - &raw (const|mut)という表現が入った?
 - 現在はWarningのみ

内部版では修正済み(これ以上の仕様変更がないかを見極め中..)



MilvusVisor version 1.5.0

- naked_asm!を使用
- Unstable Featuresを削除(予定)
 - naked_functionsが安定化すれば..
- Rust Edition 2024対応
- CPUレジスタの設定まわりの修正
 - 3年前に書いた内容があまり良くないことが分かったので修正

いつか出ます...



まとめ

- 近年SmartNICでMilvusVisorを動かしている
 - 去年はNVIDIA BlueField-2
 - 今年はXILINX ALVEO U25
- Armのブート周りのコンポーネントについて紹介
- AMD Zynq UltraScale+ MPSoCs(今回はAlveo U25)でMilvusVisorを動作させる方法を紹介
 - petalinux-configやpetalinux-buildのコマンドを一部紹介
- MilvusVisor version 1.4.1での変更について紹介
- MilvusVisor version 1.5.0(予定)の変更について紹介

MilvusVisorでは皆様のコントリビューションをお待ちしております!