

Popcorn Unikernel/HEXO Tutorial

Antonio Barbalace, Stevens Institute of Technology

Pierre Olivier, Virginia Tech

Binoy Ravindran, Virginia Tech

10/26/2019

Table of contents

HEXO Presentation

Installation

Tutorial Setup

Sample Application

Writing Your Own Application

Post-Copy Migration

Template Makefile

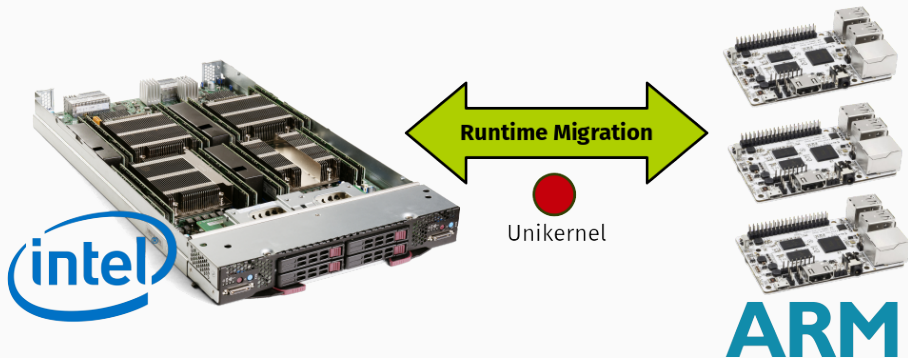
Miscellaneous Information

HEXO Presentation

HEXO Presentation

Heterogeneous EXecution Offloading

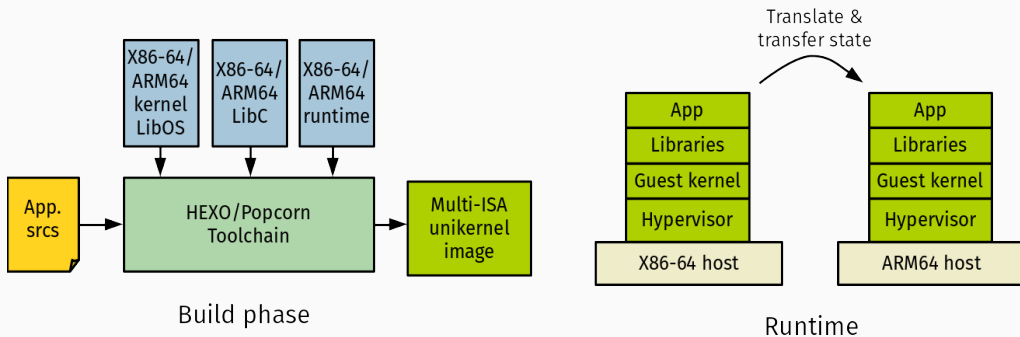
- Uses Popcorn Linux's technology to migrate micro-VMs, *unikernels*, between x86-64 servers and ARM64 embedded boards



Pierre Olivier, A K M Fazla Mehrab, Stefan Lankes, Mohamed Karaoui, Rob Lierly, and Binoy Ravindran, "HEXO: Offloading HPC Compute-Intensive Workloads on Low-Cost, Low-Power Embedded Systems", in The 28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), 2019

HEXO Components

1. **Toolchain** to create multi-ISA unikernel images ready for migration between x86-64 and ARM64
 - Compiler, linker, custom C library, runtime transformation libraries
 - Kernel library OS based on the HermitCore unikernel
 - <https://hermitcore.org/>
2. **Micro-hypervisor** (Uhyve, KVM-based) to execute and migrate unikernels



- VM state transfer: checkpoint/restart or post-copy (on demand) mode

Objective of this Tutorial

Objective

- Build and migrate a few simple C applications between an x86-64 laptop and an ARM64 embedded board
 - Use some example code
 - Write your own application
 - Use checkpointing as well as post-copy state transfer

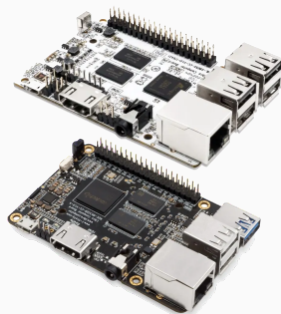
Installation

Installing HEXO

For this tutorial: no need to install anything

To use HEXO at home, requirements are:

- x86-64 machine with KVM enabled
- Librecomputer Leopotato or ROC single board computer



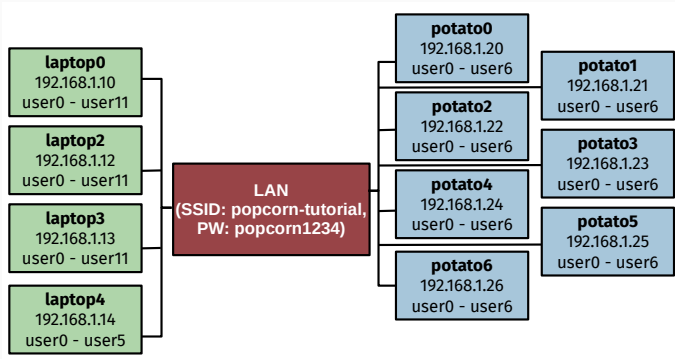
```
apt update && apt install ...  
git clone https://github.com/ssrg-vt/popcorn-compiler.git  
cd popcorn-compiler  
git checkout hermit-master  
./install_compiler.py
```

Detailed instructions:

<https://github.com/ssrg-vt/popcorn-compiler/tree/hermit-master>

Tutorial Setup

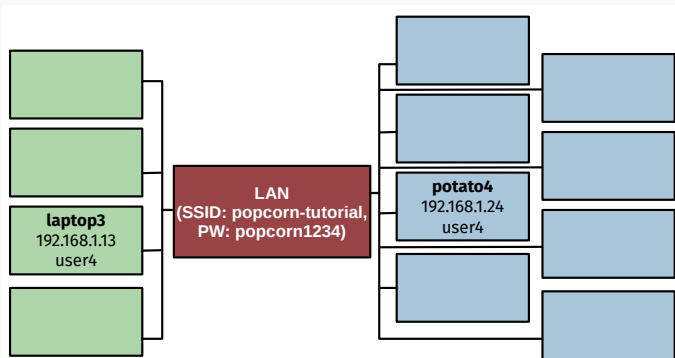
This Tutorial's Setup



- 4 x86-64 laptops, 7 ARM64 boards on the LAN

1. Use WiFi to connect to SSID 'popcorn-tutorial', password: 'popcorn1234'
2. Use ssh to connect to the laptop account, the credentials and IP address are on the distributed sheet of paper

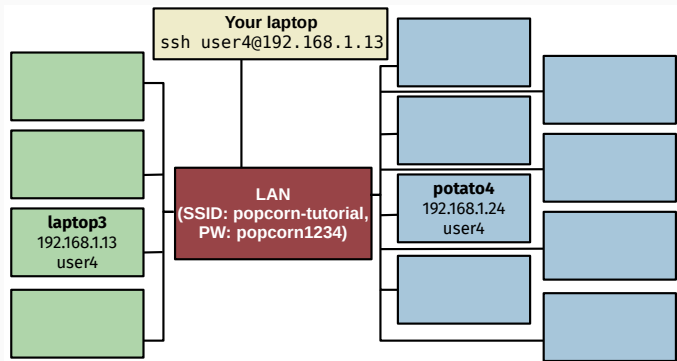
This Tutorial's Setup



- 4 x86-64 laptops, 7 ARM64 boards on the LAN
- Each user gets one account on a laptop, 1 account on a board

1. Use WiFi to connect to SSID 'popcorn-tutorial', password: 'popcorn1234'
2. Use ssh to connect to the laptop account, the credentials and IP address are on the distributed sheet of paper

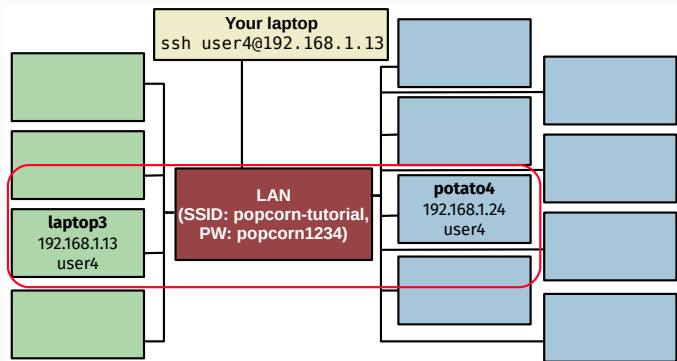
This Tutorial's Setup



- 4 x86-64 laptops, 7 ARM64 boards on the LAN
- Each user gets one account on a laptop, 1 account on a board
- Connect to the laptop account to start the tutorial

1. Use WiFi to connect to SSID 'popcorn-tutorial', password: 'popcorn1234'
2. Use ssh to connect to the laptop account, the credentials and IP address are on the distributed sheet of paper

This Tutorial's Setup



- 4 x86-64 laptops, 7 ARM64 boards on the LAN
- Each user gets one account on a laptop, 1 account on a board
- Connect to the laptop account to start the tutorial
- Migrate a unikernel between the laptop and the board

1. Use WiFi to connect to SSID 'popcorn-tutorial', password: 'popcorn1234'
2. Use ssh to connect to the laptop account, the credentials and IP address are on the distributed sheet of paper

Connecting to the Laptop's Account

Example of credentials:

Connection information and credentials (SSID: <i>popcorn-tutorial</i> , password: <i>popcorn1234</i>)				
	Machine name	IP address	User name	Password
x86-64 laptop	laptop2	192.168.1.12	user10	user10
aarch64 board	potato3	192.168.1.23	user4	user4

```
$ ssh user10@192.168.1.12 # From your machine
user10@192.168.1.12's password:
$ ls
hermit-popcorn/ popcorn-compiler/
```

- `hermit-popcorn/` contains most of the toolchain libraries, binaries and sources
- `popcorn-compiler/` contains
 - Sample applications: `popcorn-compiler/apps`
 - Makefile to ease running tests: `popcorn-compiler/util/hermit/Makefile`

All further commands in this tutorial are executed on the laptop

Sample Application

Example Application: NPB Integer Sort

Sample code in `popcorn-compiler/apps/npb-is`:

```
$ cd ~/popcorn-compiler/apps/npb-is
$ ls
c_print_results.c  is.c          npbparams-A.h  npbparams-C.h  npbparams-S.h  wtime.h
c_timers.c        Makefile      npbparams-B.h  npbparams.h    wtime.c
```

The `Makefile` can be used, in combination with its variables, to configure and launch the build/execution of applications

```
$ ls -l Makefile
lrwxrwxrwx 1 user10 user10 26 Oct 22 14:47 Makefile -> ../../util/hermit/Makefile
```


Example Application: NPB Integer Sort (2)

Simply use `make` to build

```
$ make
[x86_64-cc] c_print_results.c.x86.o
[x86_64-cc] c_timers.c.x86.o
[x86_64-cc] is.c.x86.o
[x86_64-cc] wtime.c.x86.o
[ld-unaligned] prog_x86-64
[aarch64-cc] c_print_results.c.x86.o
[aarch64-cc] c_timers.c.x86.o
[aarch64-cc] is.c.x86.o
[aarch64-cc] wtime.c.x86.o
[ld-unaligned] prog_aarch64
[align] prog_x86-64 prog_aarch64
[ld-aligned] prog_x86-64_aligned
[ld-aligned] prog_aarch64_aligned
$ ls prog*aligned
prog_aarch64_aligned  prog_x86-64_aligned
```

Example Application: NPB Integer Sort (3)

Run the application locally on the laptop:

```
$ make test-x86
...
NAS Parallel Benchmarks (NPB3.3-SER) - IS Benchmark
...
```

Run the application remotely on the board (automatically copies all necessary files to the board and uses ssh to trigger execution remotely):

```
$ make test-arm
...
NAS Parallel Benchmarks (NPB3.3-SER) - IS Benchmark
...
```

- In both case you can use ctrl+c to interrupt execution
- Currently timing measurements are wrong on arm ;)

Example Application: NPB Integer Sort (4)

Starts execution on x86 and sends a signal to checkpoint and translate the VM state after 4 seconds:

```
$ MIGTEST=4 make test-x86
...
Uhyve: exiting
$ ls *.bin*
bss.bin  data.bin  fds.bin  heap.bin  mdata.bin  stack.bin.1  tls.bin.1
```

Transfer the checkpoint to the board (may take a bit of time):

```
$ make transfer-full-checkpoint-to-arm
```

Resume on the board:

```
$ RESUME=1 make test-arm
```

Writing Your Own Application

Writing Your Own Application

Create a simple C file:

```
cd ~ && mkdir test-app && cd test-app  
vim test.c
```

```
#include <stdio.h>  
#include <hermit/migration.h> // This needs to be included in all programs  
  
extern void sys_msleep(int ms);  
int main(void) {  
    int i=0;  
    for(i=0; i<10; i++) {  
        popcorn_check_migrate(); // checkpoint if checkpointing signal received  
        printf("Iteration %d/10\n", i);  
        sys_msleep(1000);  
    }  
}
```

Writing Your Own Application

Use the template Makefile to build the binaries:

```
$ ln -s ~/popcorn-compiler/util/hermit/Makefile .  
$ make
```

Start on the board, checkpoint and translate after 5 seconds:

```
$ MIGTEST=5 make test-arm
```

Transfer checkpoint to the laptop:

```
$ make transfer-full-checkpoint-from-arm
```

Restart on the laptop:

```
$ RESUME=1 make test-x86
```

How is Migration Triggered?

Sources are instrumented as follows:

```
#include <hermit/migration.h>
...
int my_function(void) {
    ...
    popcorn_check_migrate(); // _potential_ migration point
    ...
}
```



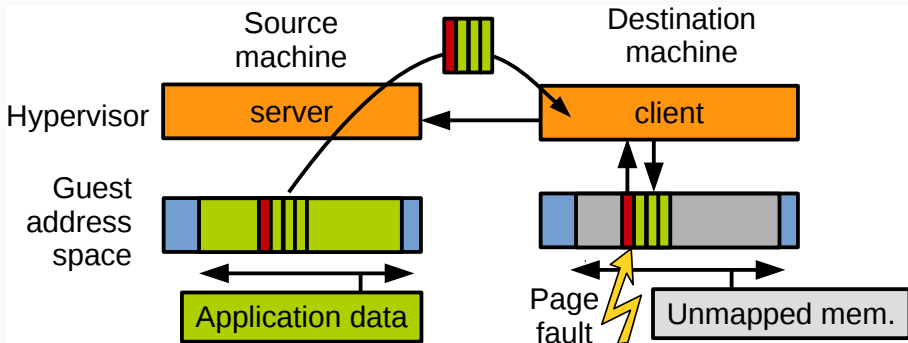
- Signal received by the hypervisor (regular process on the Linux host)

Post-Copy Migration

Post-Copy Migration

Post-copy: transfer a minimal checkpoint, let the VM resumes on the target

- Page faults trigger **on-demand state transfer**



Post-Copy Migration (2)

```
#include <stdio.h>
#include <string.h>
#include <hermit/migration.h>

#define DATA_SIZE    (100*1024*1024)
char data[DATA_SIZE];

extern void sys_msleep(int ms);
int main(void) {
    memset(data, 0x0, DATA_SIZE);

    force_migration_flag(1); /* force checkpointing */
    popcorn_check_migrate();

    for(int i=0; i<10; i++) {
        printf("Iteration %d/10\n", i);
        memset(data + i*(DATA_SIZE/10), 0x0, DATA_SIZE/10);
    }
}
```

Post-Copy Migration (3)

Full checkpoint mode:

```
$ make test-x86
$ make transfer-full-checkpoint-to-arm # relatively slow
$ RESUME=1 make test-arm             # relatively fast
```

Post-copy mode:

```
$ FULL_CHKPT_SAVE=0 make test-x86
$ make transfer-checkpoint-to-arm # relatively fast
$ RESUME=1 FULL_CHKPT_RESTORE=0 SERVER=192.168.1.12 make test-arm # slower
```

(replace the IP address for `SERVER` with the one from the laptop you use)

Template Makefile

Template Makefile in `popcorn-compiler/util/hermit/Makefile`

- Provided for easy building and testing of applications, intended for invocation on the x86-64 machine
- Multiple useful **targets** for various actions such as building and running applications, transferring checkpoints, etc.
- Can be tuned with **variables**:
 - Can be set directly by editing the Makefile
 - Or passing them as environment variables on the command line

Template Makefile (2)

Interesting targets

- `make`: build VM images from C source files in the local directory
- `make test-x86`: launch local execution on the x86-64 machine, can be tuned with variable
- `make test-arm`: remotely launch execution on the arm board, can be tuned with variables
- `make transfer-full-checkpoint-to-arm`: transfer full (C/R) checkpoint after dump from the x86 machine to the board
- `make transfer-full-checkpoint-from-arm`: same but from the board to the x86 machine
- `make transfer-checkpoint-to-arm` and `make transfer-checkpoint-from-arm`: transfer minimal checkpoint between machines (for post-copy)
- `make check`: check binaries alignment

Template Makefile (3)

Interesting Variables

- Variables for connection with the board:
 - MEM=X: memory to give to the VM, use the classical format (for example X can be 2G, 50M, etc.)
 - ARM_TARGET_IP=IP: IP address of the board
 - ARM_TARGET_USER=name: username on the board
- Variables to use with `make test-{x86|arm}`:
 - MIGTEST=X: send the migration signal after X seconds
 - RESUME=1: reload checkpoint rather than starting from the application entry point
 - FULL_CHKPT_SAVE=X: when migration is requested, save a full checkpoint for C/R (X=1) or a minimal one and switch to page server for post-copy (X=0)
 - FULL_CHKPT_RESTORE=X: when resuming, restore a full checkpoint for C/R (X=1) or a minimal one and connect to page server for post-copy
 - SERVER=IP: IP address of the page server when resuming in post-copy
 - PORT=X: port number for the page server when resuming in post-copy

Miscellaneous Information

Anatomy of an Execution

```
$ HERMIT_ISLE=uhyve HERMIT_MEM=2G HERMIT_CPUS=1 \  
HERMIT_VERBOSE=0 HERMIT_MIGTEST=0 \  
HERMIT_MIGRATE_RESUME=0 HERMIT_DEBUG=0 \  
HERMIT_NODE_ID=0 ST_AARCH64_BIN=prog_aarch64_aligned \  
ST_X86_64_BIN=prog_x86-64_aligned \  
HERMIT_MIGRATE_PORT=5050 HERMIT_MIGRATE_SERVER=0 \  
HERMIT_FULL_CHKPT_SAVE=1 \  
HERMIT_FULL_CHKPT_RESTORE=1 \  
/home/pierre/hermit-popcorn/x86_64-host//bin/proxy prog_x86-64_aligned
```

In this example:

- `proxy` is the hypervisor
- `prog_x86-64_aligned` is the unikernel image

Installation Folder Organization

Default location is `~/hermit-popcorn`

```
$ ls ~/hermit-popcorn  
aarch64-hermit/  x86_64-hermit/  x86_64-host/
```

Please do not modify these, they are symlinked for every user of this tutorial!

`aarch64-hermit/` and `x86_64-hermit` contain include and libraries for the targets,
`x86_64-host` contains host software

```
$ ls ~/hermit-popcorn/x86_64-host  
aarch64-hermit/  bin/  include/  lib/  share/  src/  x86_64-hermit/
```

- `bin` contains the toolchain binaries: compiler, and binutils (including linker)
- `src` contains the toolchain sources

Installation Folder Organization (2)

```
$ ls ~/hermit-popcorn/x86_64-host/src  
binutils/  HermitCore/  llvm/  newlib/  pte/
```

HermitCore contains the kernel and hypervisor sources:

```
$ ls ~/hermit-popcorn/x86_64-host/src/HermitCore  
arch/          cmake/          drivers/  libkern/        mm/           usr/  
build-aarch64/ CMakeLists.txt img/        LICENSE         README.md  
build-x86-64/  config/         include/  local-cmake.sh  tests.sh  
caves/         docker/         kernel/   lwip/           tools/
```

Popcorn HEXO's Team and Future Directions

The Team

- Pierre Olivier
- A K M Fazla Mehrab
- Stefan Lankes
- Antonio Barbalace
- Binoy Ravindran

Future Directions

- Support for more applications
- Support for more languages (C++?)
- Multi-threading
- In the context of offloading jobs from servers to embedded systems, **address the low amount of RAM in embedded systems**
 - Aggregate the memory of multiple embedded systems?
 - Use the memory of a large x86-64 server as swap for the embedded system?

