

Multicore Application Development with Zephyr RTOS

Alexey Brodkin,
Engineering manager, Synopsys
Linux Piter 2019, October 5, 2019



Alexey Brodtkin

Engineering manager at Synopsys

- Maintainer of U-Boot bootloader port for ARC
- Contributor to:
 - Linux kernel
 - Buildroot, OpenEmbedded/Yocto, OpenWrt
 - QEMU, uClibc etc
- Started contributions to Zephyr RTOS this year



Agenda

- Use-cases for multicore embedded processors
- Multicore support models review: AMP vs SMP
- AMP & SMP in Zephyr
- Challenges designing applications for SMP system with Zephyr
- Performance scaling on SMP system

Why go multicore?

Increase performance & responsiveness, decrease power consumption

Challenges

- Power consumption increases dramatically with increase of clock rate
- Limited MHz budget
 - Limited clock rate for a given tech process
- Multiple critical or resource-hungry task
- Very specific tasks (DSP, CNN, etc.)

Solution

- Use multiple CPUs or CPU cores
 - Scale MHz budget
 - Schedule critical tasks on dedicated CPU cores
 - Really parallel execution, not pseudo
- Use separate “accelerator” core(s)
 - Offload specific tasks
 - Use very specialized accelerators
 - DSP
 - Crypto-processor
 - Vector-processor
 - Application-specific instruction-set processor (ASIP)

Examples of multiprocessor/multicore solutions

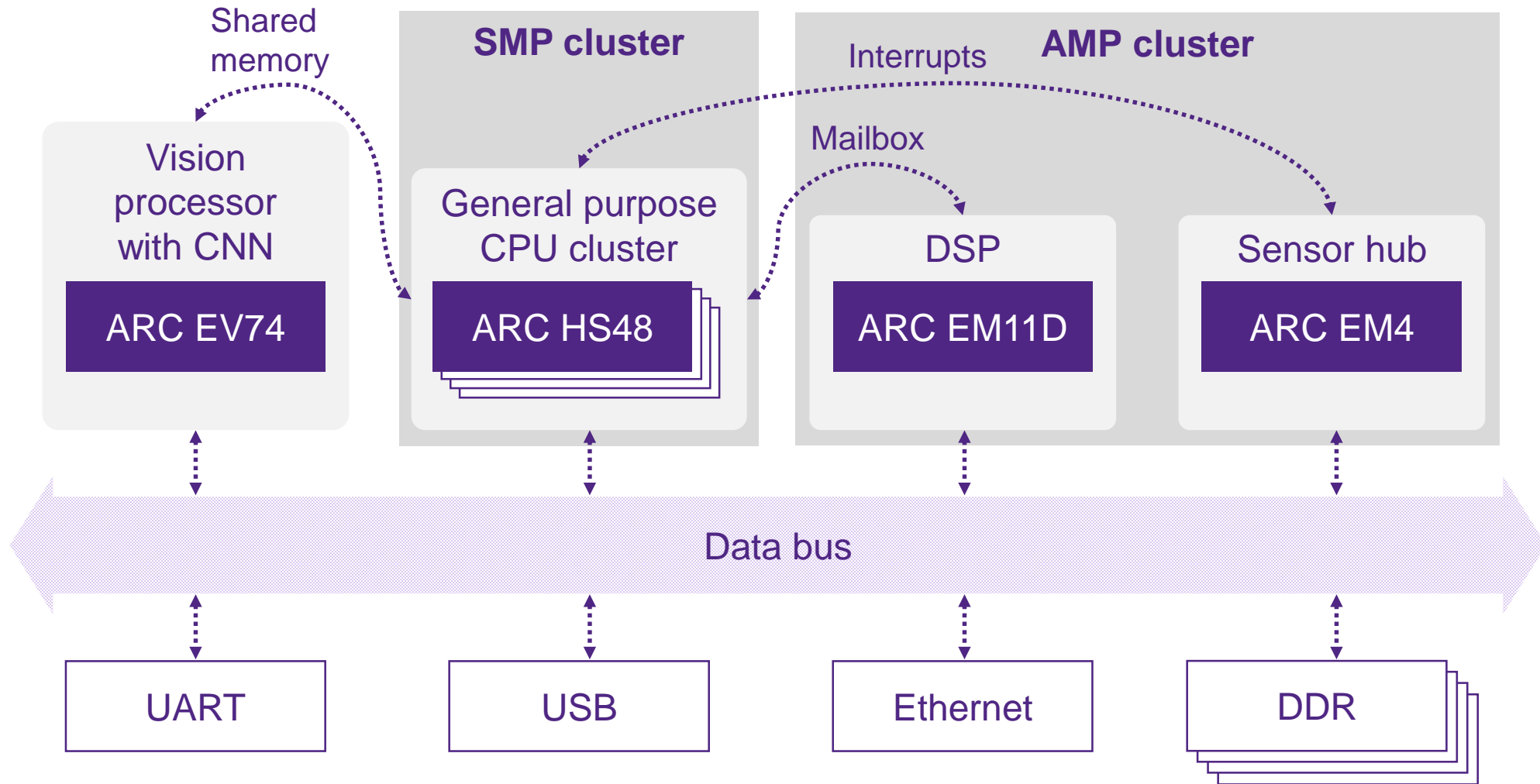
Increase performance & responsiveness, decrease power consumption

- 5G/LTE modems
 - Timing-critical non-interruptible handlers executed on dedicated CPU cores
 - Significant complexity of computations
 - High amount of data to process
- Audio DSPs
 - High-rate data I/O, specialized HW for efficient implementation of DSP algorithms
 - Cyclic buffers
 - Multiply-and-accumulate units
- Vision processors
 - Specific operations
 - Vector instructions
 - CNN
- Human-interface devices
 - Complex user-interface
 - High responsiveness



Modern heterogeneous SoC

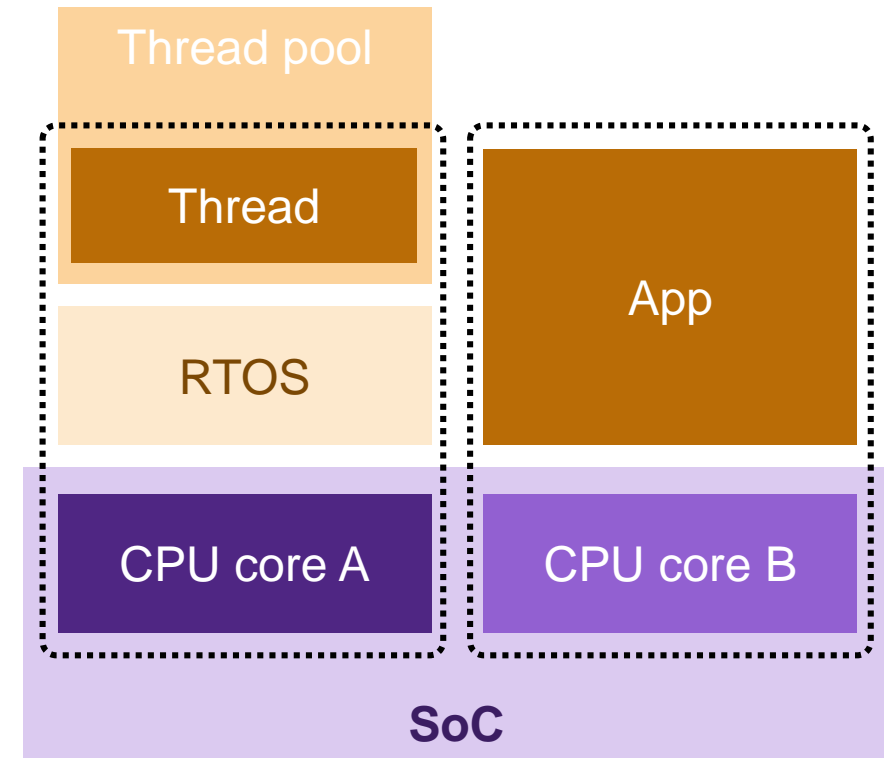
Combination of multiple different processors



Multicore support models: AMP

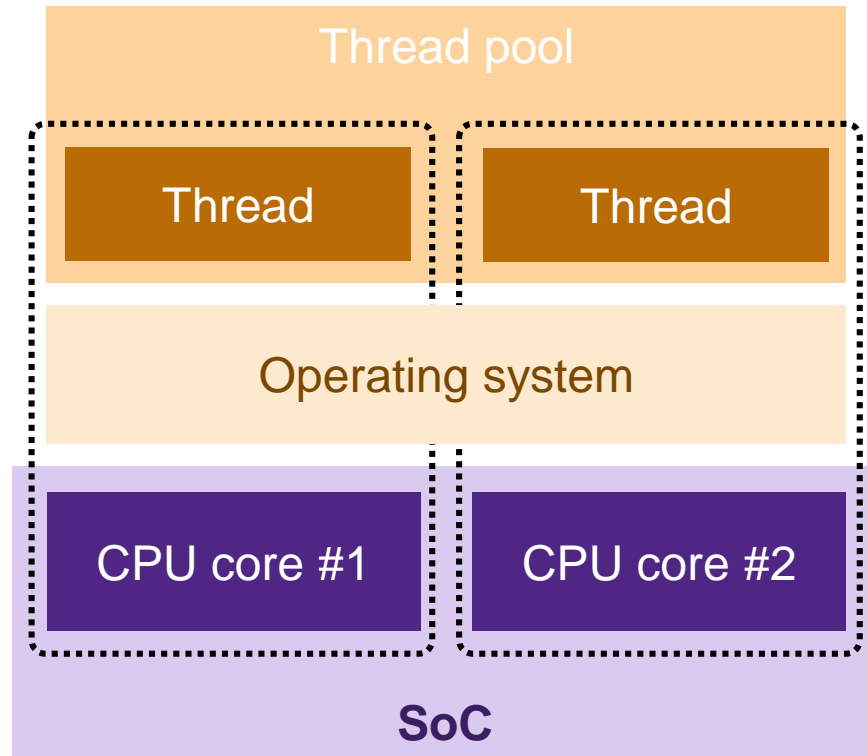
Asymmetric multiprocessing

- Completely custom system design
 - Each core might be unique
 - Multiple SW/API standards
- Wide variety of hardware IPCs
 - Shared memory
 - Proprietary mailboxes
 - Cross-core interrupts
 - HW semaphores etc
- Manual SW partitioning
- Non-scalable
 - There's no simple way to add yet another core



Multicore support models: SMP

Symmetric multiprocessing



- Strong requirements for system design
- CPU cores are equal from SW stand-point*
- Standard HW IPCs, typically
 - Shared memory
 - Coherent caches**
 - Cross-core interrupts
- Complex run-time scheduling
 - Load-balancing
 - Task pinning
- Scalable
 - Just add another core in cluster

* Except early start when there's a dedicated master and slaves are all the rest

** If CPUs have caches

Why use operating system

Significantly simplifies SW development

Abstractions of hardware

- Processor abstraction
 - Processes
 - Threads
- Peripherals abstraction
 - Complex subsystems
 - Device drivers

Implementation of complex services

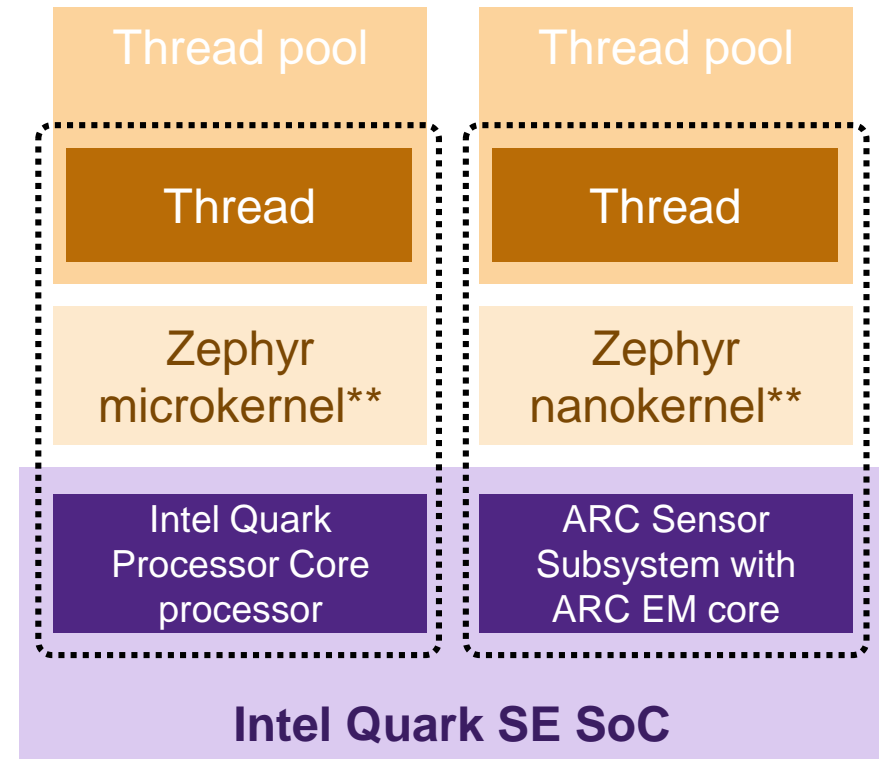
- Crypto
- File systems
- Communication
 - Ethernet
 - USB
 - CAN



AMP in Zephyr

Available from the very first commit

- Arduino/Genuino 101*
 - Intel Quark SE SoC
 - Intel x86 core
 - ARC EM4 core
 - Very limited IPCs
 - Shared memory: 80KiB @ 0xA8000000
 - ARC EM start/stop signals via Quark's SS_CFG regs
- NXP LPCXPRESSO54114
 - Arm Cortex M4F & dual Arm Cortex-M0
- ST STM32H747I Discovery
 - Arm Cortex-M7 & Arm Cortex-M4
- PSoC6 WiFi-BT Pioneer Kit
 - Arm Cortex-M4 & Arm Cortex-M0



* Arduino 101 support got dropped from Zephyr in v2.0.0

** Nano- & microkernels replaced by unified kernel in v1.14

Simplest AMP example

Arduino/Genuino 101

Intel Quark SE

```
/* Start of the shared 80K RAM */
#define SHARED_ADDR_START 0xA8000000
#define ARC_READY (1 << 0)
#define shared_data ((volatile struct shared_mem *) SHARED_ADDR_START)

struct shared_mem {
    u32_t flags;
};

int z_arc_init(struct device *arg)
{
    /* Start the CPU */
    SCSS_REG_VAL(SCSS_SS_CFG) |= ARC_RUN_REQ_A;

    /* Block until the ARC core actually starts up */
    while ((SCSS_REG_VAL(SCSS_SS_STS) & 0x4000) != 0U) {}

    /* Block until ARC's quark_se_init() sets a flag
     * indicating it is ready, if we get stuck here ARC has
     * run but has exploded very early */
    while ((shared_data->flags & ARC_READY) == 0U) {}
}
```

ARC EM4

```
static inline void quark_se_ss_ready(void)
{
    shared_data->flags |= ARC_READY;
}

static int quark_se_arc_init(struct device *arg)
{
    ARG_UNUSED(arg);
    quark_se_ss_ready();
    return 0;
}

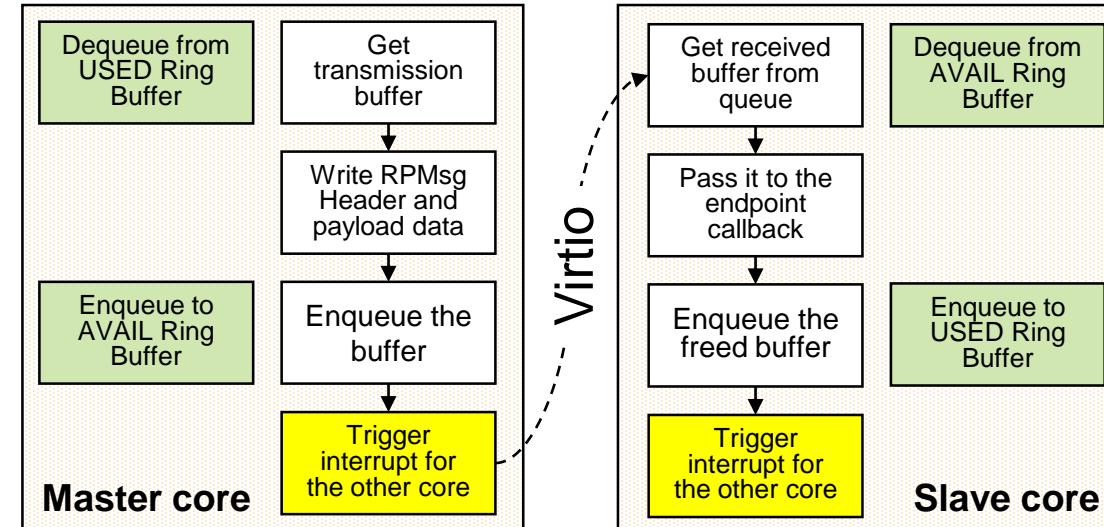
SYS_INIT(quark_se_arc_init, POST_KERNEL,
CONFIG_KERNEL_INIT_PRIORITY_DEFAULT);
```

OpenAMP in Zephyr

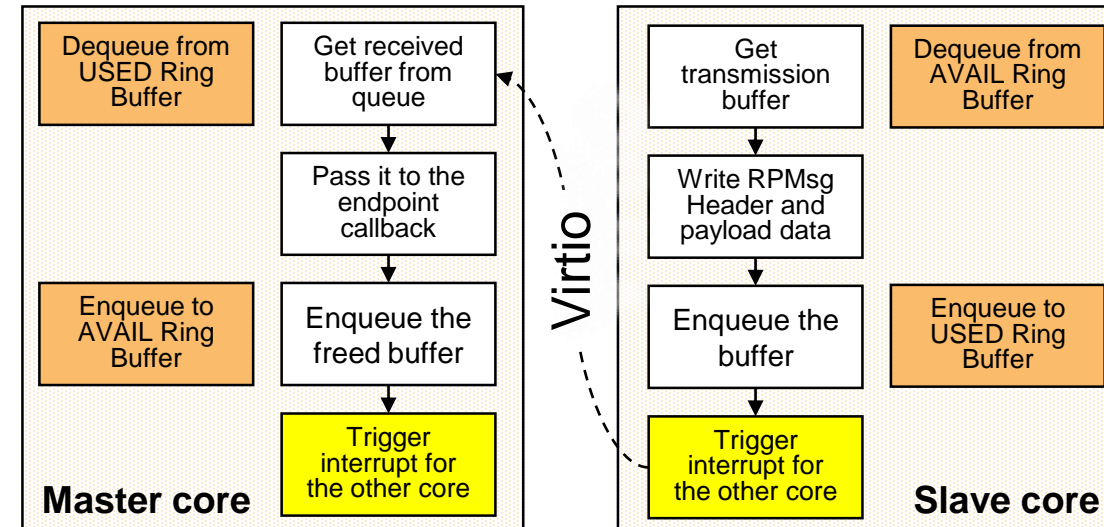
RPMsg-Lite – for devices with limited resources

- Communicate with other platforms
 - Linux
 - RTOS (Zephyr, FreeRTOS, QNX)
 - Bare-metal
- Virtio – transport abstraction layer
- Requires porting to each platform
 - Not per-architecture, but per-SoC!
 - Currently supported
 - NXP LPC54114
 - IMX6SX
- ELCE 2018 presentation by Diego Sueiro
[“Linux and Zephyr “talking” to each other in the same SoC”](#)

Host to co-processor



Co-processor to host



Using VRING0

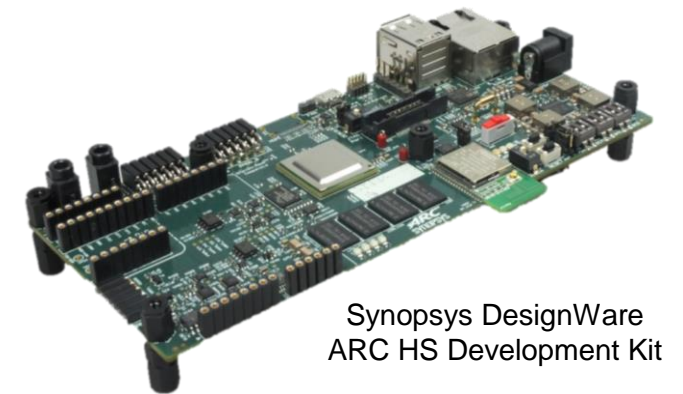
Using VRING1

Optional

SMP in Zephyr: historical overview

Requires OS-wide support, thus only emerging lately

- February 2018
 - Initial support with ESP32 in v1.11.0
<https://github.com/zephyrproject-rtos/zephyr/pull/6061>
- February 2019
 - Added x86_64 in v1.14.0
<https://github.com/zephyrproject-rtos/zephyr/pull/9522>
 - “...Virtualized SMP platform for testing under QEMU...”
- September 2019
 - ARC supported in v2.0.0
<https://github.com/zephyrproject-rtos/zephyr/pull/17747>
 - ARC nSIM instruction set simulator & ARC HS Development Kit board



* ESP32 only supports shared memory, no cross-core IRQs

* Author: Benoît Canet
Source: <http://lists.gnu.org/archive/html/qemu-devel/2012-02/msg02865.html>
Licensed under CC BY 3.0: <https://creativecommons.org/licenses/by/3.0/>

SMP in Zephyr: current status

Still in its early days

Ready

- Use of certain HW capabilities
 - Shared memory with coherent caches*
 - Cross-core IRQs**
 - Cluster-wise wall-clock
 - HW atomic instructions***
- Threads pinning to CPU(s)
 - Via CPU affinity API:
`k_thread_cpu_mask_xxx()`

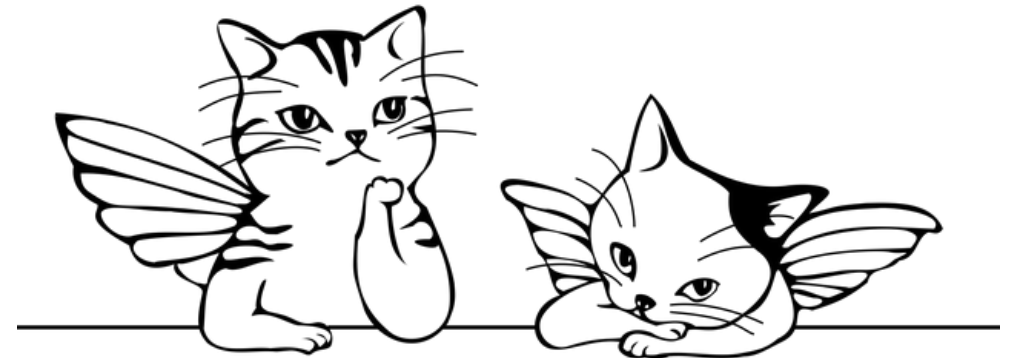
* If CPUs have caches

** For less latencies of premature thread termination

*** Used via compiler built-ins (`CONFIG_ATOMIC_OPERATIONS_BUILTIN`)
or via hand-written implementation in assembly
(`ATOMIC_OPERATIONS_CUSTOM`)

Needs more work

- More architectures and platforms
- More tests & benchmarks
- Support for more cores in the cluster
 - Now up-to 4 cores
- Fancier scheduler
 - Account for CPU migration costs
- SMP-aware Zephyr debugging with OpenOCD/GDB



Changes required to support SMP in Zephyr

Most of changes done in architecture & platform agnostic code

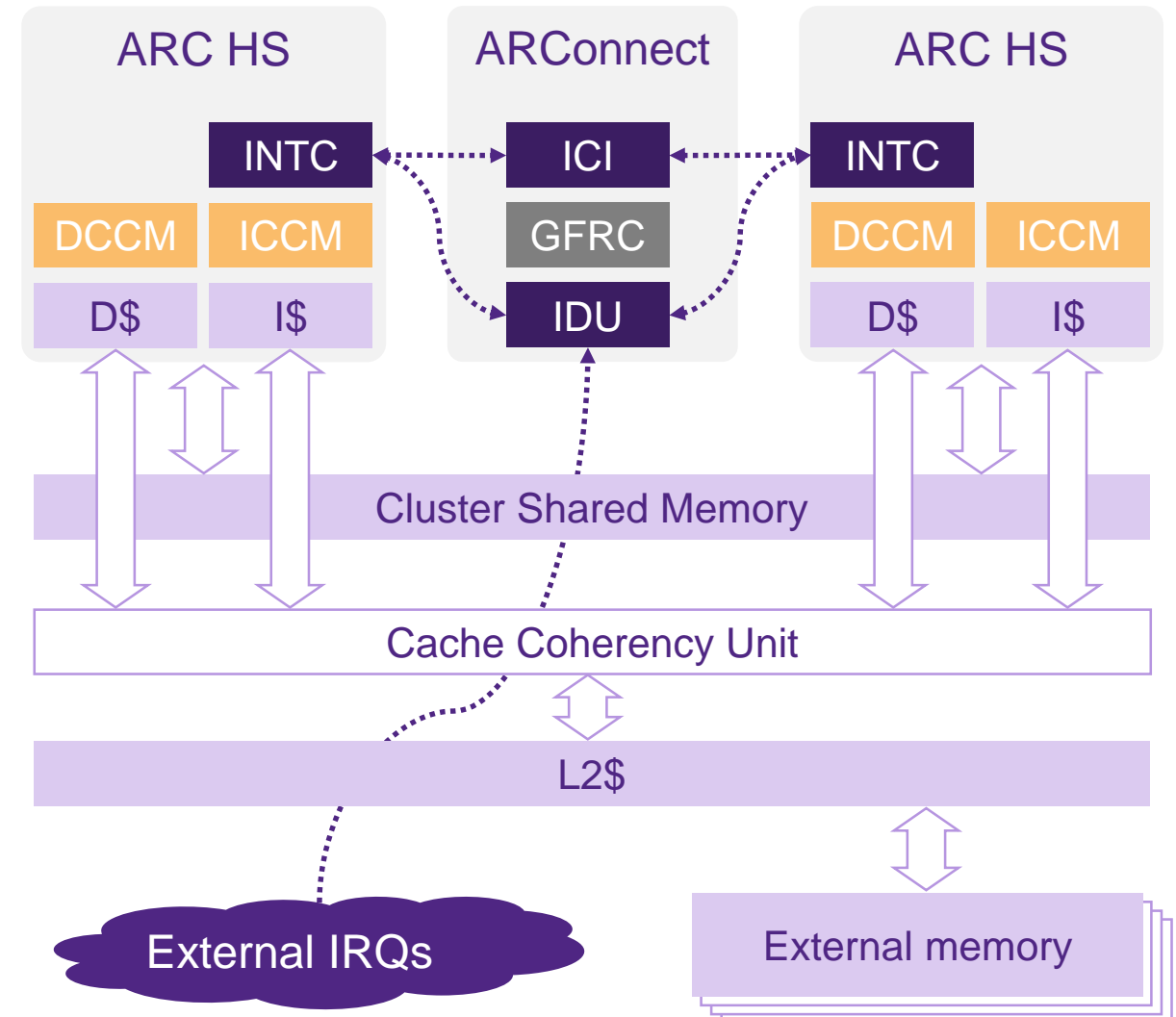
- Extra preparations for slave cores
 - Setup idle threads: `init_idle_thread()`
 - Allocate per-core IRQ stacks: `Z_THREAD_STACK_BUFFER(_interrupt_stackX)`
 - Start the core: `z_arch_start_cpu()`
- `irq_lock()`
 - UP: `z_arch_irq_lock()`
 - SMP: `z_smp_global_lock() = z_arch_irq_lock() + spinlock on global_lock`
- Prepare scheduler for SMP
 - `z_sched_abort()`
- `z_arch_switch(new_thread, old_thread)` instead of `z_swap()`
 - Lower-level
 - Scheduler unaware
 - No spinlocks inside architecture-specific assembly code



Hardware requirements for SMP

ARC HS primer

- Same ISA & functionally of all cores
- Shared memory
 - External RAM
 - SRAM
 - DDR
 - On-chip memory
 - Closely/Tightly-Coupled Memories are private (visible by 1 core)
 - ARC: ICCM/DCCM
 - ARM/RISC-V: ITCM/DTCM
- Flexible IRQ management
 - Inter-Core Interrupt Unit (ICI)
 - Interrupt Distribution Unit (IDU)
- Cluster-wise clock source
 - Global Free-Running Counter (GFRC)



Challenges designing application for SMP system

Software peculiarities

Task/thread scheduling

- Scheduler type
 - SCHED_DEADLINE
 - SCHED_DUMB
 - SCHED_SCALABLE
 - SCHED_MULTIQ
- Migration between CPU cores
 - Automatic core selection (by default)
 - Task/thread pinning
 - SCHED_CPU_MASK – only for SCHED_DUMB

Shared resources...

- Clock source
- Caches (while managing them)
- Peripherals
 - Serial port
 - SPI, I2C, Ethernet, etc

...add overhead for access serialization

- IRQ masking
 - irq_lock() → z_smp_global_lock()
- Spin-locks
- Additional SW/HW barriers

SMP benefits easily available with Zephyr

For supported architectures and platforms/boards

- Build SMP application in Zephyr
 - Change Zephyr configuration via Kconfig
 - Enable `CONFIG_SMP` by `make menuconfig`
 - SMP tests will do it for you via `tests/kernel/smp/prj.conf`
 - Optionally set `CONFIG_MP_NUM_CPUS=x`
 - Recompile with `make`
- Get performance scaling

```
/bin/wslbridge --distro-guid {afb13484-c906-44b0-b616-4e730a354ceb} -t -e APPDATA -C~
(Top) → General Kernel Options → SMP Options
Zephyr Kernel Configuration
-* Use new-style arch switch instead of swap
[*] Enable symmetric multithreading support
(4) Number of CPUs/cores
-* Architecture supports broadcast interprocessor interrupts

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                    [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode  [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

0 > ash 4* > make 0.0 0.0 0.1 2019-08-21 < 17:21 ru20arcgnu1
```

Utilization of multiple cores in SMP system is easy

SMP Pi example: <https://github.com/zephyrproject-rtos/zephyr/pull/18849>

```
#define THREADS_NUM    16
#define STACK_SIZE     XXX

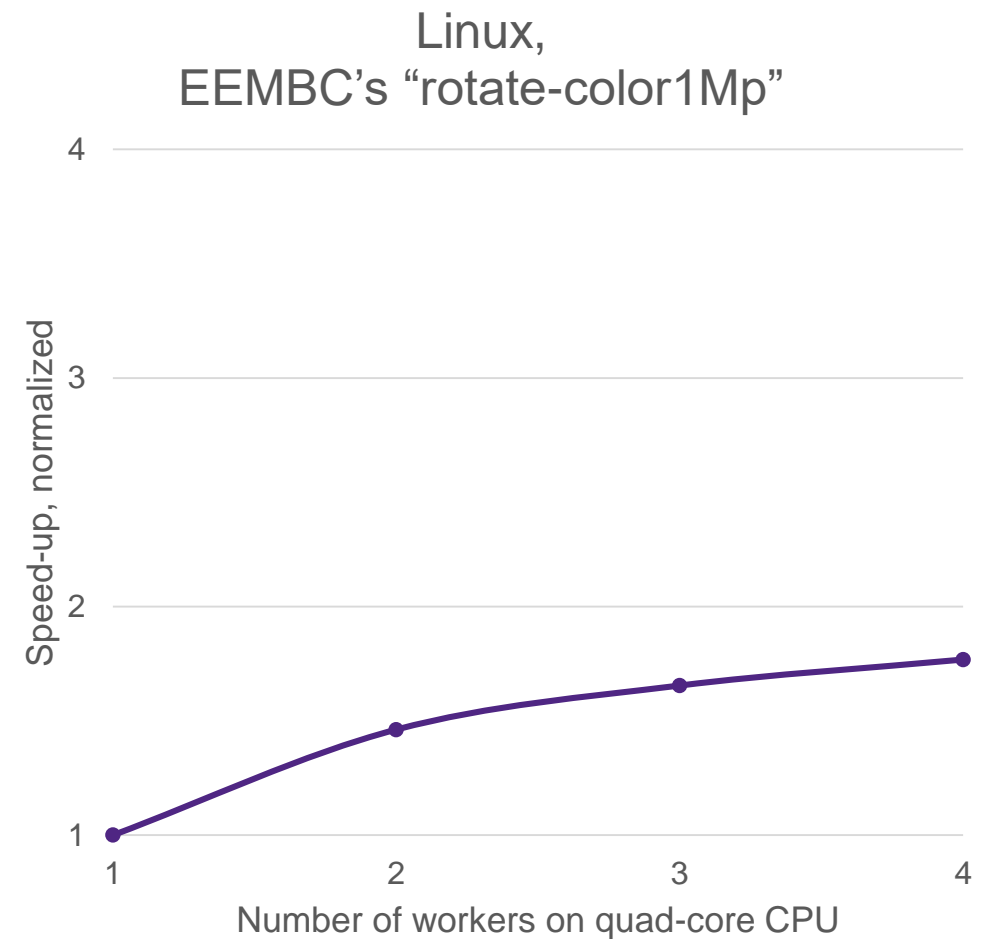
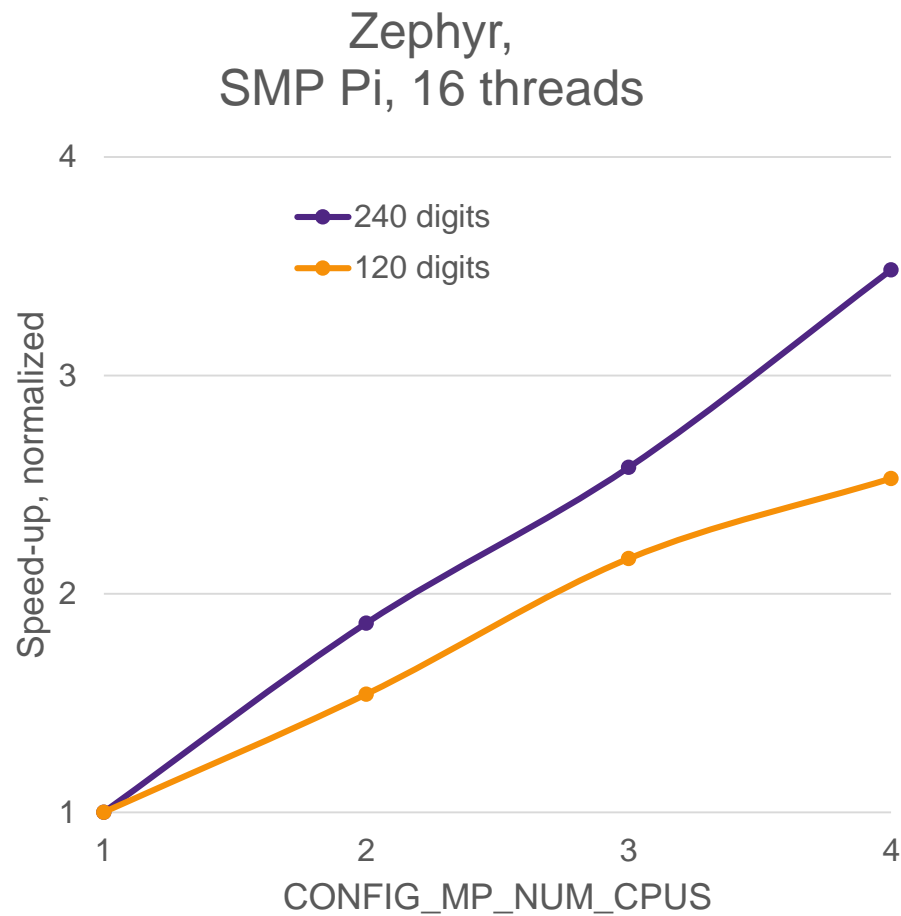
static K_THREAD_STACK_ARRAY_DEFINE(tstack, THREADS_NUM, STACK_SIZE);
static struct k_thread tthread[THREADS_NUM];

void pi_thread(void *arg1, void *arg2, void *arg3)
{
    ... calculate Pi value ...
}

void main(void)
{
    ...
    for (i = 0; i < THREADS_NUM; i++) {
        k_thread_create(&tthread[i], tstack[i], STACK_SIZE,
                       (k_thread_entry_t)pi_thread,
                       NULL, NULL, NULL,
                       K_PRIO_COOP(10), 0, K_NO_WAIT);
    }
    ...
}
```

Evaluation of SMP benefits for real applications

Scaling depends on workload



Conclusion

Zephyr RTOS allows utilization of powerful multi-core systems

- Depending on use-case and system configuration choose
 - SMP for a capable SoC
 - AMP for any HW
 - Heterogeneous multiprocessing (HMP) as a combination of AMP & SMP
- Align HW & SW design
 - Configure HW according to requirements of SW stack
 - Make sure given HW configuration is supported in SW
- SMP allows for easy scaling
 - Depending on use-case might scale better or worse

Participate

We need your help

- Get sources on GitHub: <https://github.com/zephyrproject-rtos/zephyr>
- Report bugs report & propose enhancements via GitHub issues
<https://github.com/zephyrproject-rtos/zephyr/issues>
- Contribute fixes & improvements via pull-requests
<https://github.com/zephyrproject-rtos/zephyr/pulls>



Thank You

