SAIL-RISCV 内存模型解析与重构

使用 newtype 进行类型安全重构

黄烁

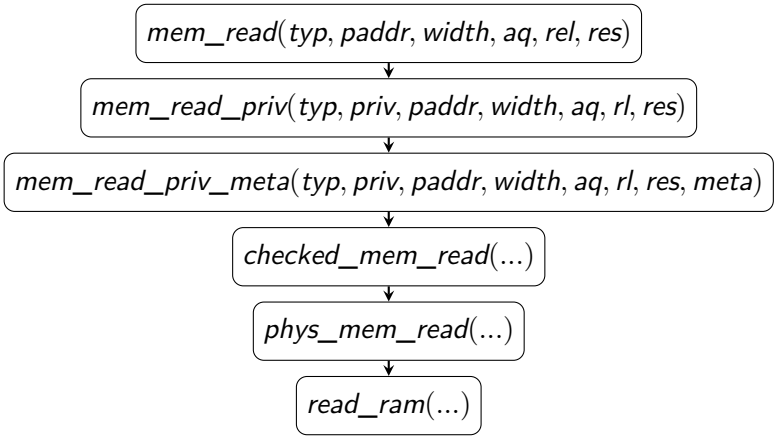
2024 年 11 月 13 日

目录

SAIL-RISCV 内存模型

物理内存 (Physical Memory)

- 基础内存访问接口



SAIL-RISCV 内存模型

物理内存 (Physical Memory)

- 基础内存访问接口

```
function read_ram(rk, physaddr(addr), width, read_meta) = {
  let meta = if read_meta then __ReadRAM_Meta(addr, width)
             else default_meta;
  let request = struct {
    access_kind = match rk {
      ...
    },
    va = None(),
    pa = addr,
    translation = (),
    size = width,
    tag = false,
  };
  match sail_mem_read(request) {
    Ok((value, _)) => (value, meta),
    Err() => exit(),
```

SAIL-RISCV 内存模型

虚拟内存 (Virtual Memory)

- 虚拟地址入口类型
  - PC Fetch 类型

```
union Ext_FetchAddr_Check ('a : Type) = {
  Ext_FetchAddr_OK  : virtaddr,
  Ext_FetchAddr_Error : 'a
}
```

SAIL-RISCV 内存模型

虚拟内存 (Virtual Memory)

- 虚拟地址入口类型
  - PC Fetch 类型

```
union Ext_FetchAddr_Check ('a : Type) = {
  Ext_FetchAddr_OK  : virtaddr,
  Ext_FetchAddr_Error : 'a
}
```

  - PC 控制流类型

```
union Ext_ControlAddr_Check ('a : Type) = {
  Ext_ControlAddr_OK : virtaddr,
  Ext_ControlAddr_Error : 'a
}
```

SAIL-RISCV 内存模型

虚拟内存 (Virtual Memory)

- 虚拟地址入口类型
  - PC Fetch 类型

```
union Ext_FetchAddr_Check ('a : Type) = {
  Ext_FetchAddr_OK  : virtaddr,
  Ext_FetchAddr_Error : 'a
}
```

  - PC 控制流类型

```
union Ext_ControlAddr_Check ('a : Type) = {
  Ext_ControlAddr_OK : virtaddr,
  Ext_ControlAddr_Error : 'a
}
```

  - 数据访问类型

```
union Ext_DataAddr_Check ('a : Type) = {
  Ext_DataAddr_OK : virtaddr,
  Ext_DataAddr_Error : 'a
}
```

SAIL-RISCV 内存模型

虚拟内存系统 (Virtual Memory System)

- Sv32: 32 位虚拟地址
  - 2 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[1] (10 位)	VPN[0] (10 位)	页内偏移 (12 位)

SAIL-RISCV 内存模型

虚拟内存系统 (Virtual Memory System)

- Sv32: 32 位虚拟地址
  - 2 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[1] (10 位)	VPN[0] (10 位)	页内偏移 (12 位)

- Sv39: 39 位虚拟地址
  - 3 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[2] (9 位)	VPN[1] (9 位)	VPN[0] (9 位)	页内偏移 (12 位)

SAIL-RISCV 内存模型

虚拟内存系统 (Virtual Memory System)

- Sv32: 32 位虚拟地址
  - 2 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[1] (10 位)	VPN[0] (10 位)	页内偏移 (12 位)

- Sv39: 39 位虚拟地址
  - 3 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[2] (9 位)	VPN[1] (9 位)	VPN[0] (9 位)	页内偏移 (12 位)

- Sv48: 48 位虚拟地址
  - 4 级页表,4KB 页大小
  - 虚拟地址划分:

VPN[3] (9 位)	VPN[2] (9 位)	VPN[1] (9 位)	VPN[0] (9 位)	页内偏移 (12 位)

SAIL-RISCV 内存模型

虚拟地址翻译为物理地址

- 确定虚拟内存模式

```
function translationMode(priv : Privilege) -> SATPMode = {
if priv == Machine then
  Sbare
else if xlen == 32 then
  match Mk_Satp32(satp)[Mode] {
    0b0 => Sbare,
    0b1  => Sv32
  }
else if xlen == 64 then {
  let arch = architecture(get_mstatus_SXL(mstatus));
  match arch {
    Some(RV64) => { let mbits : bits(4) = satp[63 .. 60];
                    match satp64Mode_of_bits(RV64, mbits) {    // see riscv_types.sail
                      Some(m) => m,
                      None()  => internal_error(__FILE__, __LINE__,
                                                "invalid RV64 translation mode in satp")
```

SAIL-RISCV 内存模型

虚拟地址翻译为物理地址

- 确定虚拟地址是否有效

```
let (valid_va, sv_params) : (bool, SV_Params) = match mode {
  Sbare => return TR_Address(physaddr(virtaddr_bits(vAddr)), init_ext_ptw),
  Sv32  => (true,                                  sv32_params),
  Sv39  => (is_valid_vAddr(sv39_params, vAddr_64b), sv39_params),
  Sv48  => (is_valid_vAddr(sv48_params, vAddr_64b), sv48_params),
  // Sv57 => (is_valid_vAddr(sv57_params, vAddr_64b), sv57_params),    // TODO
};
...
function is_valid_vAddr(struct { va_size_bits, _ } : SV_Params,
                        vAddr                      : bits(64)) -> bool =
 vAddr == sign_extend(vAddr[va_size_bits - 1 .. 0])
```

SAIL-RISCV 内存模型

虚拟地址翻译为物理地址

- 查询 TLB

SAIL-RISCV 内存模型

虚拟地址翻译为物理地址

- 查询 TLB
- 查询页表

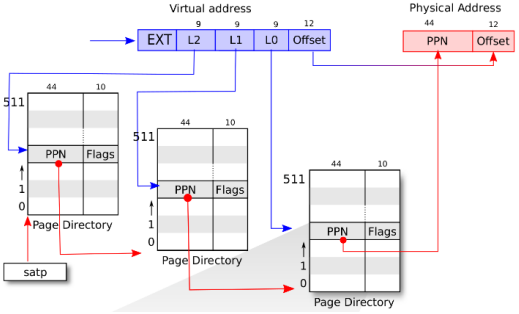


图: sv39 页表遍历过程 (from MIT 6.828)

当前实现存在的问题

- 当前地址实现

```
(* 物理内存读取 *)
val read_ram : (read_kind, xlenbits, int('n), bool) -> (bits(8 * 'n), mem_meta)
(* 虚拟地址检查 *)
union Ext_DataAddr_Check ('a : Type) = {
  Ext_DataAddr_OK : xlenbits,    /* Address to use for the data access */
  Ext_DataAddr_Error : 'a
}
```

当前实现存在的问题

- 当前地址实现

```
(* 物理内存读取 *)
val read_ram : (read_kind, xlenbits, int('n), bool) -> (bits(8 * 'n), mem_meta)
(* 虚拟地址检查 *)
union Ext_DataAddr_Check ('a : Type) = {
  Ext_DataAddr_OK : xlenbits,    /* Address to use for the data access */
  Ext_DataAddr_Error : 'a
}
```

- 类型安全问题，地址混淆风险，缺乏语义区分
  在实际的编写中，可能会将物理和虚拟地址混淆，导致 Bug 的产生

SAIL 语言中的 newtype

- newtype 是 SAIL 语言中的类型封装机制

SAIL 语言中的 newtype

- newtype 是 SAIL 语言中的类型封装机制
- 语法：`newtype name = constructor : type`

SAIL 语言中的 newtype

- newtype 是 SAIL 语言中的类型封装机制
- 语法：`newtype name = constructor : type`
- 示例：

```
newtype meters = Meters : int
newtype feet = Feet : int

val add_meters : (meters, meters) -> meters
function add_meters(Meters(x), Meters(y)) = Meters(x + y)

// 编译错误：feet 不能和 meters 相加
let wrong = add_meters(Meters(5), Feet(10))
```

重构方案

使用 newtype 进行类型安全重构

- 定义物理地址和虚拟地址

```
newtype physaddr = physaddr : xlenbits
newtype virtaddr = virtaddr : xlenbits

function physaddr_bits(physaddr(paddr) : physaddr) -> xlenbits = paddr

function virtaddr_bits(virtaddr(vaddr) : virtaddr) -> xlenbits = vaddr
```

重构方案

使用 newtype 进行类型安全重构

- 定义物理地址和虚拟地址

```
newtype physaddr = physaddr : xlenbits
newtype virtaddr = virtaddr : xlenbits

function physaddr_bits(physaddr(paddr) : physaddr) -> xlenbits = paddr

function virtaddr_bits(virtaddr(vaddr) : virtaddr) -> xlenbits = vaddr
```

- Sail-RISCV 内存模型内部，规范各个函数的接口类型检查

重构方案

使用 newtype 进行类型安全重构

- 定义物理地址和虚拟地址

```
newtype physaddr = physaddr : xlenbits
newtype virtaddr = virtaddr : xlenbits

function physaddr_bits(physaddr(paddr) : physaddr) -> xlenbits = paddr

function virtaddr_bits(virtaddr(vaddr) : virtaddr) -> xlenbits = vaddr
```

- Sail-RISCV 内存模型内部，规范各个函数的接口类型检查
- 外部对接部分，保留向外输出的 xlenbits 类型

总结

- SAIL-RISCV 内存模型现状
  - 复杂的内存访问层次结构
  - 物理地址和虚拟地址使用相同类型 (xlenbits)
  - 缺乏类型安全保证，存在地址混淆风险

总结

- SAIL-RISCV 内存模型现状
  - 复杂的内存访问层次结构
  - 物理地址和虚拟地址使用相同类型 (xlenbits)
  - 缺乏类型安全保证，存在地址混淆风险
- 重构方案
  - 使用 newtype 机制封装地址类型
  - 分别定义 physaddr 和 virtaddr 类型
  - 在内部实现类型安全检查

总结

- SAIL-RISCV 内存模型现状
  - 复杂的内存访问层次结构
  - 物理地址和虚拟地址使用相同类型 (xlenbits)
  - 缺乏类型安全保证，存在地址混淆风险
- 重构方案
  - 使用 newtype 机制封装地址类型
  - 分别定义 physaddr 和 virtaddr 类型
  - 在内部实现类型安全检查
- 预期效果
  - 避免地址混淆导致的 bug
  - 保持对外接口兼容性

感谢聆听