支持一致性缓存的Spike仿真器

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

宋 威

信息安全国家重点实验室

songwei@iie.ac.cn

学生参与者：

李真真，李博雅

Spike: RISC-V的标准实现（Golden Model）

- Spike: RISC-V ISA Simulator
  - https://github.com/riscv/riscv-isa-sim

- RISC-V的标准实现（Golden Model）
  - 在一个ISA扩展成为标准之前，一般会在Spike进行功能测试和性能评估
  - 最早实现RISC-V指令集各种更新的仿真器
  - 可用于对标其他实现，检验其他实现在指令执行上的功能正确性

- Spike仿真器的意义（除了标准实现之外）
  - 快速指令仿真：不同于Gem5，不模拟流水线的细节，不模拟时钟
  - 可以以近似QEMU的速度启动Linux操作系统
  - 面向体系结构：不同于QEMU，Spike保证指令执行序列的准确性，没有二进制翻译，严格遵守指令执行的顺序（program order）
  - 面向硬件和基础软件开发者：暴露最基本的硬件特新（异常处理、中断控制、内存保护、简单外设模拟、总线控制），可用于底层代码的开发。

Spike的缓存模型

鉴于Spike的目标（面向体系结构和底层硬软件开发的迅速迭代），Spike并没有实现准确的缓存模型。

- Spike现有的缓存模型是可选的，在实现上接近一个玩具（toy model）
  - 开启缓存仿真和不开启缓存仿真会造成10倍以上的仿真速差
  - 缓存模型不存储数据，只是一个访问统计计数器
  - 不支持私有缓存：所有的核共享一个一级缓存
  - 不支持缓存一致性：没有私有缓存，自然没必要支持一致性。从缓存的角度来看，系统是单核的。
  - 不支持虚拟地址访问：所有访问均为物理地址
  - 只有简单的数据统计：访问数、命中数、缺失数、写回数
  - TLB不访问一级缓存
- 对于多核缓存性能的研究，Spike现有模型基本没用！
  - 似乎当前仅剩的选择只有Gem5，可是Gem5的速度令人捉急。。。

我们的需求

- 我们是谁?
  - 信息工程研究所信安国重，体系结构安全研究组
  - 关于缓存侧信道攻击的攻击分析及其防御

- 我们的需求?
  - 能够让我们迅速构建新型缓存结构的平台
    - 不同的置换算法、缓存映射算法、缓存预取、缓存一致性协议
  - 能够让我们迅速仿真的平台
    - 修改了缓存之后，能迅速的执行攻击程序或者正常程序
  - 可以运行操作系统或正常C/C++程序
  - 可以仔细查看缓存的内部状态
    - 统计信息
    - 缓存组的访问分布
    - 运行时的缓存访问/命中率/置换率变化
    - 跟踪地址

Gem5：太慢，实现过于复杂

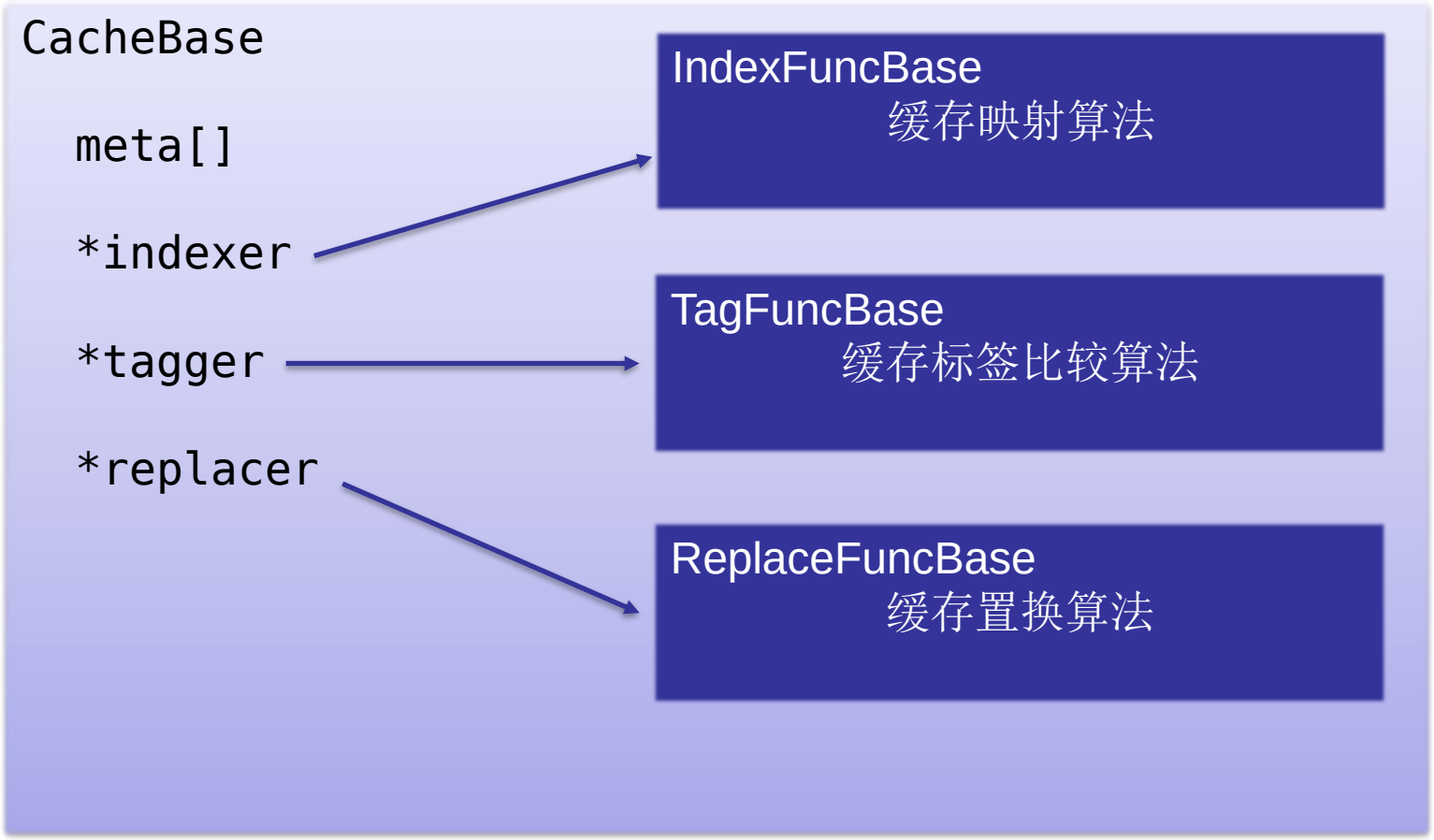
QEMU：远离硬件，没有缓存模型

具体目标

- 缓存模型
  - 支持一致性：MSI
  - 支持私有缓存
  - 支持各种缓存结构：模块化设计+缓存结构配置文件
  - 支持缓存状态查看：一个复杂的全局统计记录器
  - 快速仿真：行为级，事件驱动、也不记录数据

- Spike
  - 替换缓存模型
  - 降低缓存仿真的速差
  - 支持TLB对一级缓存的访问

缓存的基类：直接硬推上干货 ;-)



缓存的基类：基本API

 # 从地址获得缓存组编号

```
virtual uint32_t get_index(uint64_t addr)
```

 # 获取一个缓存块的metadata

```
uint64_t get_meta(uint32_t idx, uint32_t way) const
```

 # 设置一个缓存块的metadata

```
void set_meta(uint32_t idx, uint32_t way, uint64_t m_meta)
```

 # 检测一个地址是否命中（包括如果命中，返回位置）

```
virtual bool hit(uint64_t addr)
virtual bool hit(uint64_t addr, uint32_t *idx, uint32_t *way)
```

 # 选择被置换的缓存路，帮助置换算法记录缓存块访问（访问和无效）

```
virtual uint32_t replace(uint32_t idx)
virtual void access(uint32_t idx, uint32_t way)
virtual void invalid(uint32_t idx, uint32_t way)
```

缓存的基类：询问API

 # 询问机制允许上层软件在运行时查询缓存的当前状态

 # 询问某一个缓存块的当前状态

```
virtual void query_block(uint32_t idx, uint32_t way, CBInfo *info) const
```

 # 询问某一个缓存组的状态

```
virtual void query_set(uint32_t idx, SetInfo *info) const
```

 # 询问某两个地址在这个缓存种是否被映射到同一个缓存组（congruent关系）

```
virtual bool query_coloc(uint64_t addrA, uint64_t addrB)
```

 # 直接询问如果一个地址被缓存在该缓存，会被存放在什么位置

```
virtual LocInfo query_loc(uint64_t addr);
```

缓存的基类：工厂API

 # 缓存参数很多，这里使用工厂机制，为后面的缓存配置文件解析搭架子

 # 缓存构造的工厂函数：可以构造一个子类（特殊缓存），但返回基类指针

```
static CacheBase *factory(uint32_t nset, uint32_t nway,
                          indexer_creator_t ic,
                          tagger_creator_t tc,
                          replacer_creator_t rc,
                          uint32_t level,
                          int32_t core_id,
                          uint32_t cache_id)
```

只有缓存编号需要指定

 # 利用构造工厂函数，返回一个更简洁的构造函数对象：隐藏具体配置信息

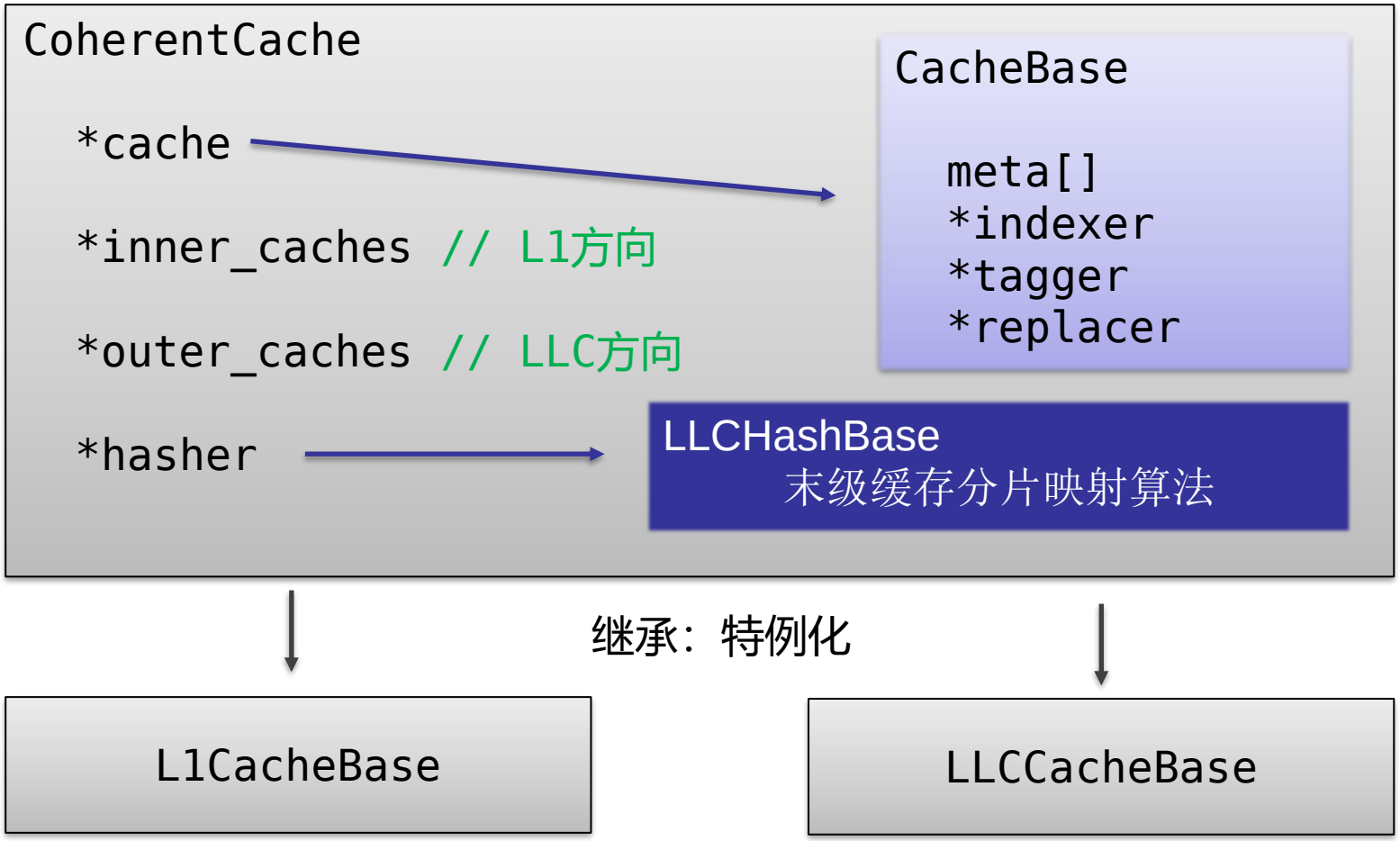
```
static cache_creator_t gen(uint32_t nset, uint32_t nway,
                           indexer_creator_t ic,
                           tagger_creator_t tc,
                           replacer_creator_t rc,
                           uint32_t delay = 0)
```

这些都省掉了

 # 例子，创建一个缓存对象：

```
CacheBase *pcache = cache_gen(level, core_id, cache_id)
```

一致性缓存的包装（Wrapper）



一致性缓存：一致性API（部分）

 # 服从一致性的缓存读写接口

```
  virtual void read(uint64_t addr, uint32_t inner_id)
  virtual void write(uint64_t addr, uint32_t inner_id, bool to_dirty =
false)
```

 # 上一级缓存向本级写回数据（Release）

```
  virtual void release(uint64_t addr, uint32_t inner_id)
```

 # 刷新缓存：缓存块或整个缓存

```
  virtual void flush(uint64_t addr, int32_t levels, uint32_t inner_id)
  virtual void flush_cache(int32_t levels, uint32_t inner_id)
```

 # 下一级缓存向本级询问缓存块状态

```
  virtual void probe(uint64_t addr, bool invalid)
```

全局统计记录器（Reporter）

- 设计思路：
  - 在缓存关键位置插桩，记录缓存的变化（事件）
  - 全局缓存状态数据库：按需求只记录需要查看的（尽量降低速度损失）
    - 可以具体到某一级缓存的某一个缓存的某一组
  - 任何时候可以开启/清除/关闭/暂停/重启一个事件的记录

- 记录基本信息

```
// 记录2号处理器核的一级代码缓存的访问信息
reporter.register_cache_access_tracer(1, 2, 0);
// 询问2号处理器核的一级代码缓存的写回次数
reporter.check_cache_writeback(1, 2, 0);
```

- Trace一个地址（帮助调试）

```
// 开启针对0xffffffff00008400的trace
reporter.register_address_tracer(0xffffffff00008400);
// 在此之后，所有关于该地址缓存变化，都会自动打印输出，类似：
// 0xffffffff00008400 is accessed at level 2 core 0 cache 0 set 16 way 1 [1]
```

- 还可以构造其他更复杂的功能

缓存配置文件：cache.json (cache_config_parser)

```
{
  "config": {
    "default" : ["1x64x8", "1x1024x16"],
    "spike-default" : ["2x64x8", "1x1024x16"]
  },
  "cache": {
    "1x64x8": {
      "number": 1,
      "set"   : 64,
      "way"   : 8,
      "type"  : "norm",
      "indexer"    : "norm",
      "tagger"     : "norm",
      "replacer"   : "lru",
      "hasher"     : "norm",
      "delay"      : 0
    },
    "2x64x8": {
      "base": "1x64x8", "number": 2
    },
    "1x1024x16": {
      "base": "1x64x8",
      "set": 1024, "way": 16
    }
  },
  "indexer": {
    "norm": {
      "type"  : "norm",
      "delay" : 0
    }
  },
  "tagger": {
    "norm": {
      "type"  : "norm",
      "delay" : 0
    }
  },
  "replacer": {
    "lru": {
      "type"  : "lru",
      "delay" : 0
    }
  },
  "hasher": {
    "norm": {
      "type"  : "norm",
      "delay" : 0
    }
  }
}
```

Spike的整合

```
$ spike -h
Spike RISC-V ISA Simulator 1.0.1-dev

usage: spike [host options] <target program> [target options]
Host Options:
  -p<n>                 Simulate <n> processors [default 1]
  -m<n>                 Provide <n> MiB of target memory [default 2048]
  -m<a:m,b:n,...>       Provide memory regions of size m and n bytes
                          at base addresses a and b (with 4 KiB alignment)
  -d                    Interactive debug mode
  -g                    Track histogram of PCs
  -l                    Generate a log of execution
  -h, --help            Print this help message
  -H                    Start halted, allowing a debugger to connect
  --isa=<name>          RISC-V ISA string [default RV64IMAFDC]
  --varch=<name>        RISC-V Vector uArch string [default v128:e32:s128]
  --pc=<address>        Override ELF entry point
  --hartids=<a,b,...>   Explicitly specify hartids, default is 0,1,...
  --cache-cfile=<name>  the configuration file for the cache model
                        [default cache_model/config/cache.json]
  --cache-model=<name>  name of the cache mod
  --extension=<name>    Specify RoCC Extension
  --extlib=<name>       Shared library to load
  --rbb-port=<port>     Listen on <port> for remote bitbang connection
  --trace               trace the cache activity
. . . .
```

例子: spike **--cache-model=spike-default --trace** pk hello

Spike的内存寻址

- 原有的寻址方式
  - 软件TLB：用于仿真器访存的地址翻译加速，并不模拟硬件TLB功能
  - 直接主机地址寻址：在回添TLB时，直接回添VA到Host Address (HA)的映射
  - 当需要访存时（数据或指令），TLB直接给出HA，仿真器根据HA读取内存
  - 如果VA在缓存记录范围，使用慢速模式：
    - 强制读取页表进行地址翻译
    - 向缓存模型发出访问请求
    - 根据页表查询结果，PA到HA的转换，最终从主机内存读数
  - 这就是原有Spike在开启缓存后仿真速度下降10倍的原因！

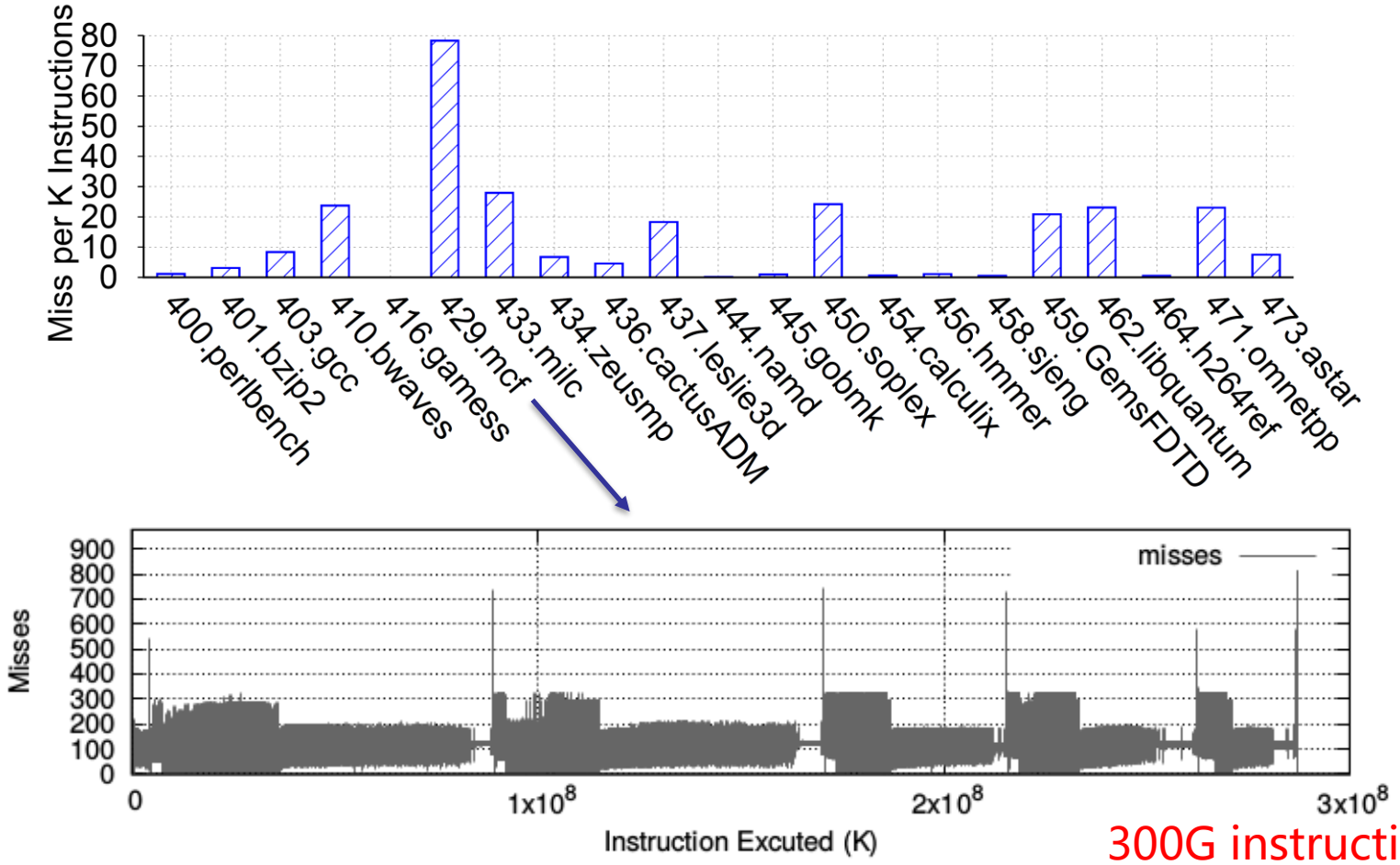
- Spike-cache的方式
  - 软件TLB和直接地址寻址没有改变
  - 如果配置了缓存类型参数（--cache-model），开启缓存仿真
  - 如果VA在缓存记录范围内，直接向缓存模型发出请求
  - 原有慢速模式只有当软件TLB miss时启动，和缓存模型无关

Spike的TLB和硬件TLB支持

- 原有Spike的TLB为软件TLB，并不模拟硬件TLB的功能
  - 不记录页属性
  - 不检查页属性
  - 任何关于页表的修改，都导致TLB刷新
    - fense.vma
    - MSTATUS, MPMPCFG, SATP
    - 优先级切换

- Spike-cache的TLB支持
  - 原有软件TLB保留不变
  - 如果配置了缓存类型参数（--cache-model），开启硬件TLB仿真
  - 参照硬件TLB进行VA-PA翻译，检查页表属性
  - TLB miss需要访问页表时，同时向L1-D发出请求
  - 不发生任何异常（假设所有异常已经被软件TLB捕获）
  - 基本只在fense.vma时发生刷新
  - 结果和软件TLB不一致时发出assertion错误（功能设计错误）

SPEC CPU 2006仿真



平均仿真速度1.5~2 million instruction per second (MIPS)

开源啦!

https://github.com/comparch-security/spike-cache

```
$ git clone https://github.com/comparch-security/spike-cache
$ cd spike-cache
$ mkdir build
$ cd build
$ ../configure –prefix=$RISCV
$ make -j8 libcache_model.a
$ make -j8 librandomgen.a
$ make -j8 install
$ cd ..
$ spike --cache-model=spike-default --trace pk hello
bbl loader
Hello!
------- statistics --------
Core 0 runs 383410 instructions
Core(0)-L1(0): miss 316, evict 0, and writeback 0 in 122626 access, miss
rate 0.002577
Core(0)-L1(1): miss 1397, evict 885, and writeback 862 in 84673 access,
miss rate 0.016499
L2: miss 1670, evict 0 and writeback 0 in 2609 access, miss rate 0.640092
```

后面的工作

- 不断添加新的缓存结构和功能
  - 随机缓存
  - 更复杂的缓存trace
  - 软件访问缓存pfc的CSR接口
  - 更多的一致性协议，non-inclusive缓存，directory支持
- 提高稳定性
  - Spike的编译流程
  - 缓存配置文件的默认地址
  - 暴露更多的参数
- 上游整合?
  - 不太确定SiFive对我们将缓存模型替换掉的态度
  - 现在的code size有些大，已经不太适合分步整合了
  - 也许作为可选项进入上游？需要和他们谈一谈。

谢谢！欢迎合作PR ;-)

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS