Model Presentation

2018年3月9日 14:06

总览

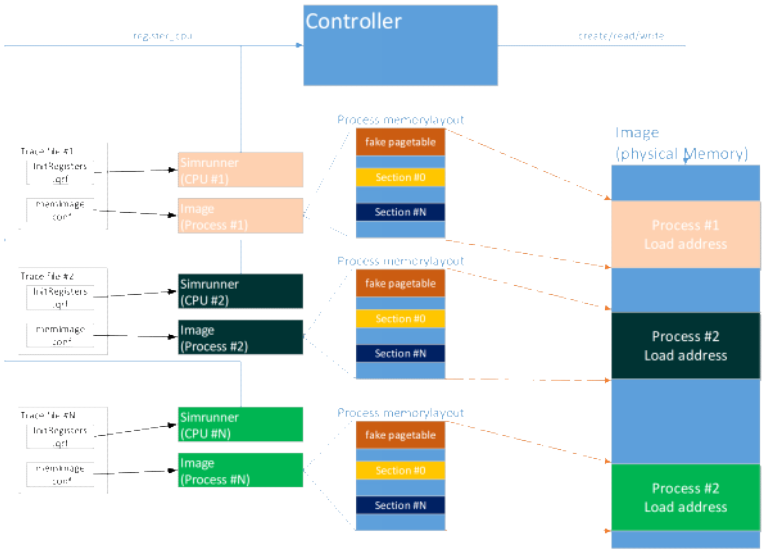
目前看到的model只包括CPU-only的模型，没有任何device模型，不是一个完整的系统simulator。但是可以运行CPU benchmark，比如drynstone, SPEC20XX (基于trace格式)

- Function model uISA模拟器，实现了将x86指令翻译为对应uops的过程，但是没有任何时间信息，仅仅实现了x86->uop的翻译并执行的过程，以及需要的辅助的运行时环境， 共有678890行代码(没有注释，包含无效行)，其中uop解码相关代码623716行，463995行是x86解码过程，159720行是存储的解析后的uops序列和信息
- Timing model 详细实现Processor内部各个硬件组成和Pipeline结构， 共有53683行代码(没有注释，包含无效行)，执行效率大约为2KIPS
- Support library 用于支持function/timing model实现的辅助代码 共有21293行代码(没有注释，包含无效行)

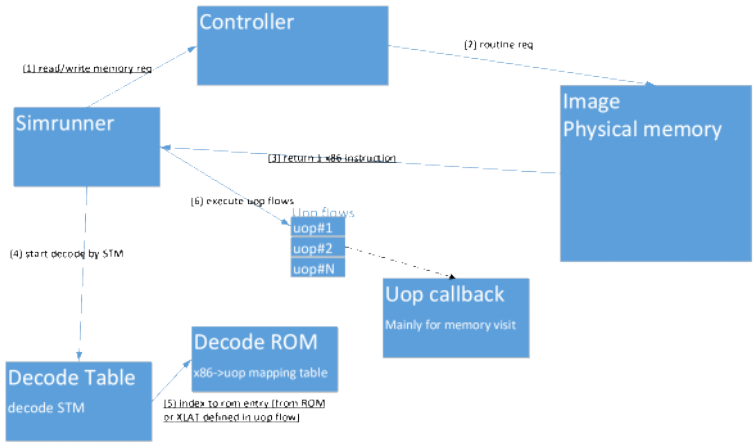
Function Model

Working Flow

- Load Phase



- Execute Phase



Trace文件

model中使用的trace有两个部分组成

- conf process加载进入内存后的memory layout，分成若干节(是否与某个具体的执行格式相关，未知)
- qrf 指定process初始状态下的system register的初始值

trace文件应该通过类似于PIN的工具trace生成，目前系统中包含spec2000/2006的所有trace文件

Controller

Function model执行过程中的控制器，类似于系统中的chipset功能。用于访问simulated physical memory，中断嵌入，读取仿真对象的属性信息等功能。猜测，还有外围的system仿真环境，用于仿真外围device，并用于注入中断，由controller负责转发device的中断处理

Image

主要包含两部分功能

- 仿真物理memory，管理物理memory region，并提供加速访问技术
- 加载process的memory image，建立VA->PA的映射关系，伪造pagetable

Simrunner

Function model的CPU仿真部分，是一个uISA模拟器，实现了uISA的执行环境

- fetch-decode-execute循环执行处理每条x86指令
- 支持systemcall emulation(有限制，比如不支持fork, clone等)

uISA

uISA context

Function model定义了某款CPU的uISA环境

- uop个数： 519
- uop寄存器：10个通用tmp寄存器，6个地址通用tmp寄存器，3个段tmp寄存器，2个fuse信息寄存器，1个repeat前缀寄存器，8个浮点tmp寄存器
- uop标志：trigger_microrom，x86_start/end_marker，branch_indicator等

x86 group

在现有的model实现中，uop按照实现的不同功能分类为33组（base+expandXX)，base为model运行中必须支持的x86指令，而其余32组expand指令组可以在命令行

中选择解码与否

Decode (x86->uop)

Function model的90%以上代码都用于定义x86->uop翻译的状态机以及每条x86指令对应的uop序列

指令翻译状态机代码，463995行

x86指令对应的uop序列代码， 159720行， 哪些指令trap ucode哪些从xlat解码由uop标志进行标记

所有的代码都以C的array数组形式存储，这些代码应该是通过更高级的DSL语言转换而来，但是目前的源码中没有这部分DSL的代码

Timing Model

Simulator总体结构

Framework

```
int main(int argc, char *argv[]) {
  build_system(config_file);

  // cycle simulation
  while (true) {
    if (halt_condition_meet)
      break;
    FSB_simulate();
    foreach CORE
      CORE_PIPELINE_simulate();
    thread_cycle++;
  }
}
```

从上面伪码可以看出，Timing model的整体仿真方式是每cycle迭代仿真所有的仿真对象一次

Timing model与function model

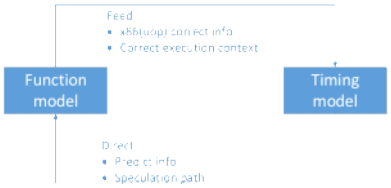
Timing model不能单独运行，其运行必须借助于function model，由function model首先产生运行的结果，然后送给timing model进行对应的时序仿真。这样做的优点在于可以将function model和timing model分离，可以编写不同精度的timing model可不用担心功能的正确性；简化timing model的编写复杂度

Function model完成：

- 指令解码，指令信息产生
- 异常/中断的产生
- 指令执行
- 实际的内存访问（物理地址产生，memory访问）

Timing model完成：

- 具体仿真的CPU系统结构的构建
- pipeline的timing仿真，但不完成实际的功能，功能在function model中实现
- 指导function model进行投机执行



Timing的仿真

内部的thread_cycle变量表示base clock；系统中不同的仿真对象可以设置不同的工作频率

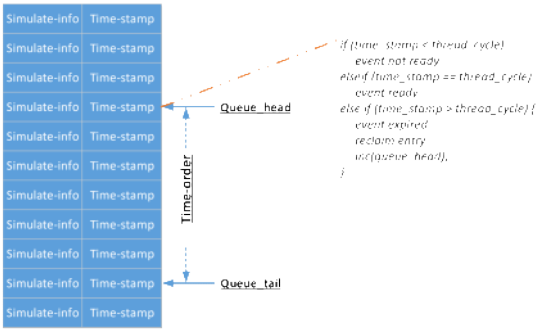
在系统仿真的配置文件中可以指定不同仿真对象的工作频率，当已知某一个仿真对象的工作频率与base clock的ratio关系时，其余仿真对象的工作频率与base clock的ratio关系可以通过仿真对象间的工作频率计算得出。(base clock不代表某个具体的频率值，只是仿真使用的时间变量)

用于timing演进的仿真结构-SIMQ

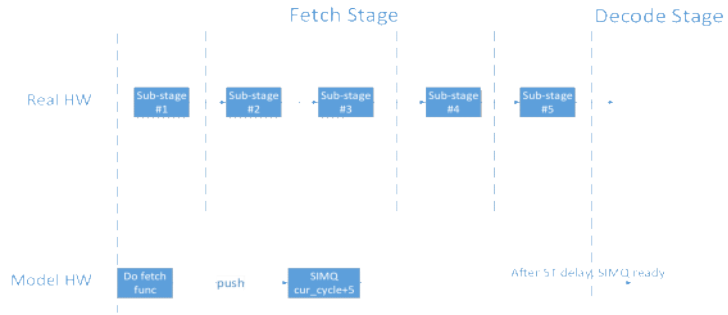
为了描述Timing系统中某个Event的delay，系统中大量使用SIMQ的数据结构。这个结构构建了整个timing model的时间演进过程

SIMQ结构

SIMQ是每个entry带有时间信息的FIFO，entry之间按照时间信息的先后进行排序



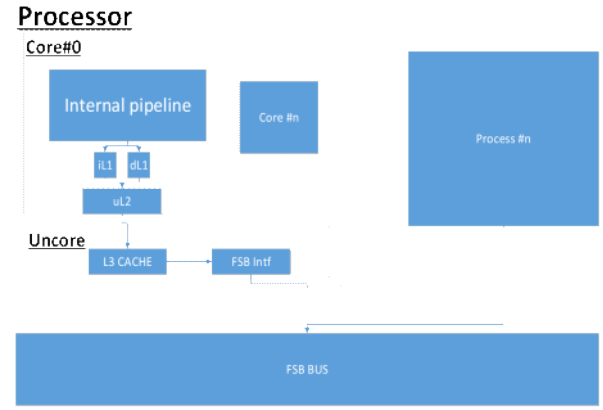
SIMQ用于仿真delay



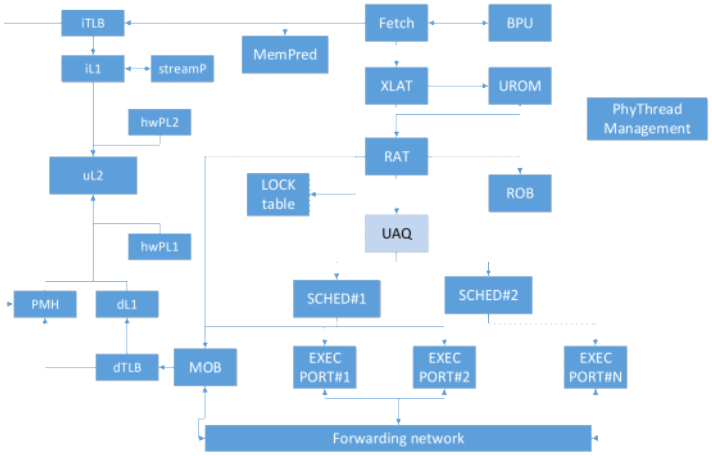
Timing model中所有的delay的仿真都基于SIMQ实现，不同的delay可以设置不同的delay长度的SIMQ。如果要进行精细仿真，则将仿真系统中的delay进行进一步分解，并用更多的SIMQ就可以达到更高的精度

仿真的系统结构

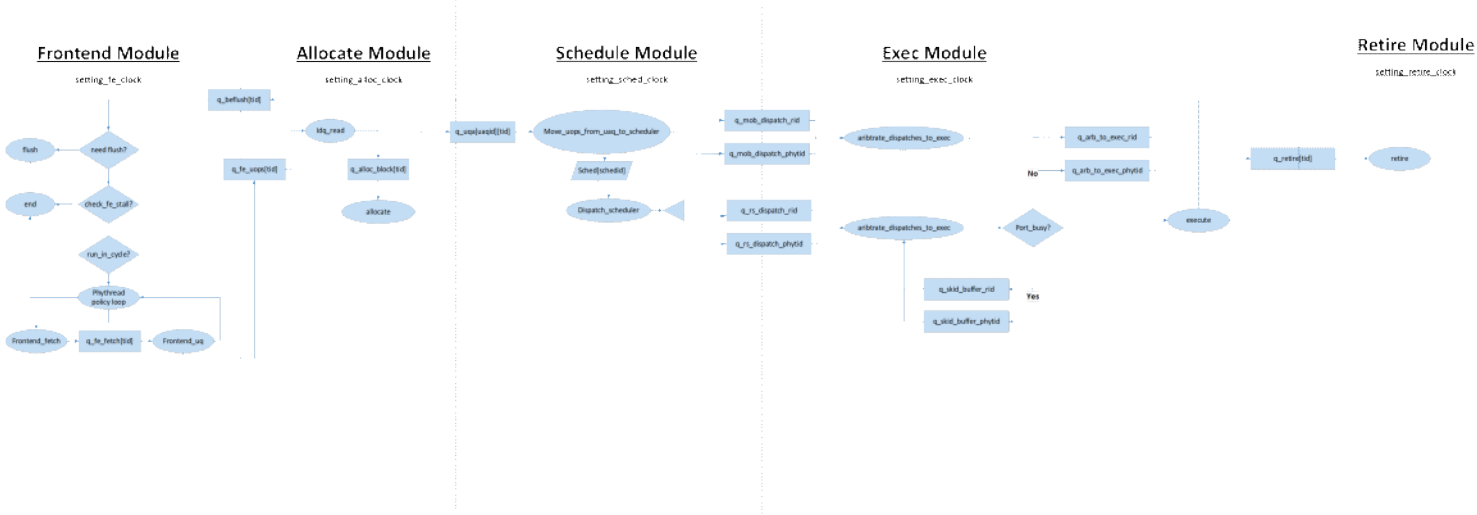
Hierarchy



Core internal



Core Pipeline的实现



SMT的考虑

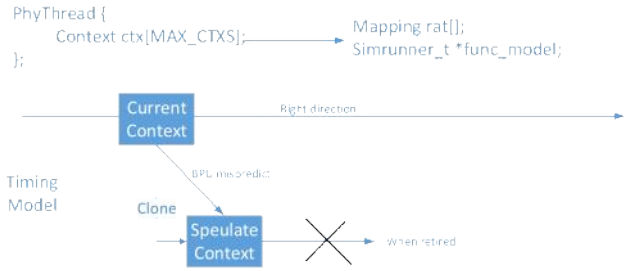
Thread的调度策略

```
enum {
  POLICY_INDEPENDENT;
  // 遍历所有的physical thread，只要满足处理条件全部处理
  POLICY_PHYTHREAD_INTERLEAVE;
```

```
    // 找出满足条件的physical thread，处理一次，如果没有处理，则查找下个满足条件的physical thread，直到有一个可以处理或是遍历所有physical thread
    POLICY_PHYTHREAD_ROUNDROBIN;
    // 当前T遍历所有的满足条件的physical thread
    POLICY_ICOUNT;
    // 首先找出使用资源最少的phyiscal thread进行处理；如果ThreadState.count已经处理过了，则当前T结束，否则轮询下个资源使用最少的physical thread
    POLICY_OLDEST_FIRST;
    // 同PHYTHREAD_ROUNDROBIN
    POLICY_LEAST_RECENT_SERVICED;
    // 同PHYTHREAD_ROUNDROBIN
    POLICY_REPLICATED;
    // 同PHYTHREAD_ROUNDROBIN
    POLICY_TIME_INTERLEAVE;
    // 只处理当前初始选择的physical thread，无论处理条件是否满足
    POLICY_THREAD_UNWARE;
    // 同PHYTHREAD_ROUNDROBIN
};
```

投机执行的实现方式

对于现代处理器来说，BPU会预测下一条指令的执行地址，并投机执行，为了支持投机执行机制，目前Model采取了如下的策略



投机时需要保存的信息主要由两部分构成

- Register renaming信息，clone RAT table
- Function model的执行context以及physical memory的memory write；通过Simrunner_t， 即function model clone当前正确点的function model获得

Next Plan to introduce

- Frontend BPU/Fetch/Decode
- RAT
- Schedule/execution
- MOB
- Retirement
- Uncore Cache-System/FSB/HW-Prefetch