动机

随着计算机系统结构的不断复杂和程序长度的不断提高，计算机系统模拟器所关注的结构也在不断增加，导致模拟速度愈发减缓。

采样模拟是一种简单有效的方式缓解这一问题，在采样模拟中，不再模拟完整的基准测试，而是仅对少量样本进行模拟，以减少模拟运行的指令总数，从而加速模拟过程，常用的方法是SMARTS与Simpoint，这两种采样方法都存在同一种问题，为了缓解冷启动的微体系结构状态对模拟结果的干扰，其需要进行一定程度的预热，以缓存为例，冷启动时缓存为空，命中率极低，与程序稳态执行的缓存状态差异显著；通过预热，可使缓存命中率趋近于正常，保证采样片段的性能具有代表性。预热的典型实现方式是执行目标片段前的一段连续指令序列，预热长度通常为目标片段长度的 1-2 倍。这就导致了详细模拟的一大半时间都花费到本无需模拟的的指令上。

所以这个项目的目的是期望直接构建不同配置下的微体系结构状态，跳过预热阶段，从而加速模拟过程，目前只实现了Cache相关信息的恢复，但是实验数据表明，Cache恢复后，精度已经很好，此外也可以按照这类思路和接口完成其他微体系结构状态的恢复。

NEMU修改部分

1.添加功能

添加了memtrace选项，开启后，在模拟的过程中会记录对内存访问的记录，并在生成checkpoint时，输出到对应的checkpoint所在的目录下。

2.主要修改代码

2.1 Kconfig

在Kconfig选项中添加了`config MEMTRACE`与`MEMTRACE_PATH`，用于编译时确定是否要开启memtrace功能，以及memtrace的暂存路径。

特别地，以下所有修改都严格使用了#ifdef CONFIG_MEMTRACE进行控制，确保不开启功能时完全对原有程序无影响。

2.2 include/util.h

增加了记录信息所需的结构体`pkt_data_used_small`，以及获取checkpoint输出路径的`get_memtrace_file_path`，增加了`memtrace_dump`

用于将捕捉的内存访问记录增添到缓冲区，以及输出到对应文件，生成checkpoint时会触发`memtrace_flush()`，将全部内容输出到对应的checkpoint目录下，触发nemu的good_trap的时候会触发`memtrace_trapflush()`，把剩余的记录输出到memtrac的暂存路径中。

具体实现在src/utils/memtrace.c中完成

2.3 src/checkpoint/serializer.cpp

在对应的checkpoint生成时，对PMem序列化时，调用了`memtrace_flush()`，并且使用uniform生成checkpoint时，增加了开启nemu checkpoint生成功能后附近的checkpoint生成

```
-   nextUniformPoint = intervalSize;
+   nextUniformPoint = 0;
```

2.4 src/isa/riscv64/instr/special.h

增加了good_trap触发时的`memtrace_trapflush()`调用。

2.5 src/memory/host-tlb.c | src/memory/paddr.c

在两者途径的内存访问中截获所需要的访问记录信息，调用`memtrace_dump`进行输出。

3.目的

捕捉该访存记录，作为输出，使用程序https://github.com/greenfool/CacheReplay.git，可以完成指定配置下Cache的模拟，用于得到Cache对应的微体系结构状态。

GEM5修改部分

1.添加功能

基于上述的https://github.com/greenfool/CacheReplay.git所产生的指定配置下Cache的微体系结构状态，使用NEMU产生的checkpoint进行恢复的同时，恢复对应配置下Cache的微体系结构状态，无需使用预热进行Cache的微状态恢复。

2.主要修改代码

2.1 configs/common/Options.py

增加`--memtrace_path`选项，用于指定CacheReplay所产生的文件，并在后续中使用该选项确保，不指定该选项时，不会触发该功能，不影响GEM5其余功能的正常使用。

2.2 configs/common/Simulation.py | src/python/m5/simulate.py

修改`m5.instantiate`的输入参数，传入memtrace_path对应选项`mem_trace_file`，以及checkpoint的指定选项`pmem_file_path`,并在其余的SimObject完成初始化后，调用L3Cache的恢复功能

```
        for obj in root.descendants():
            if str(obj) == "system.l3.tags" and pmem_file_path is not None
and mem_trace_file is not None:
             obj.warmupState(pmem_file_path,mem_trace_file)
```

2.3 src/mem/cache/tags/base.cc

此处是具体的恢复流程，根据文件内容，依次恢复cacheline，完成内容恢复后，需要更新其LRU的时间戳信息`updateRp`,以及CacheBlk的相关读写权限，所有者信息的恢复，特别注意的是，目前只完成了单核的适配

2.4 src/mem/cache/replacement_policies/lru_rp.cc

增加了使用memtrace的时候，LRU时间戳的恢复，并只适配了lru_cp.cc这一种替换策略，其余策略暂且都是使用空，后续如果想要补充更多策略，在CacheReply中实现后，可以利用此接口恢复。

2.5 src/mem/snoop_filter.cc

由于本身只适配了单核，所以在此处的snoop侦听进行了更加宽松的检查

2.6 src/sim/globals.cc | src/sim/root.cc

GEM5使用NEMU的checkpoint进行恢复时，全局的tick是从0开始，导致之前恢复的LRU时间戳会晚于模拟时间，所以需要使用`Globals::memtrace_init()`完成全局时钟模拟的推迟，并使用`Root::memtrace_init()`完成所有事件队列事件的推迟

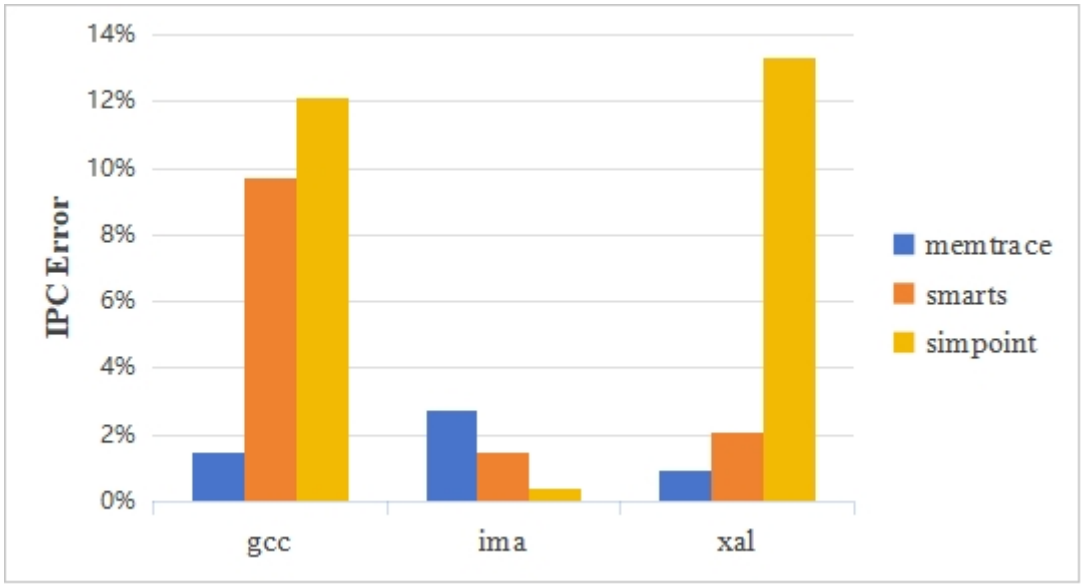
实验数据

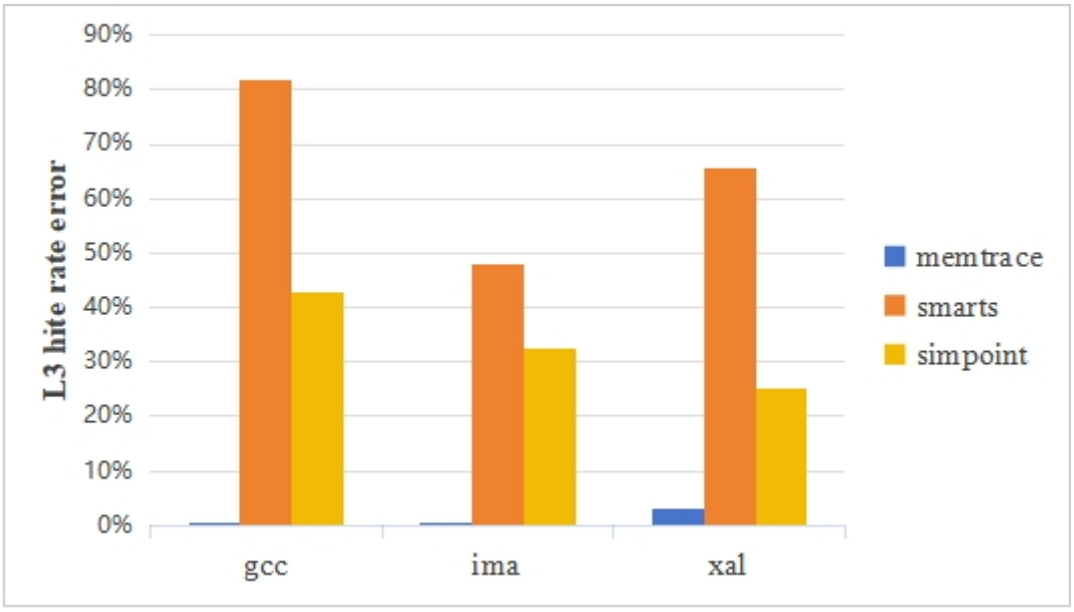
spec2017：602.gcc_s、623.xalancbmk_s、638.imagick_s

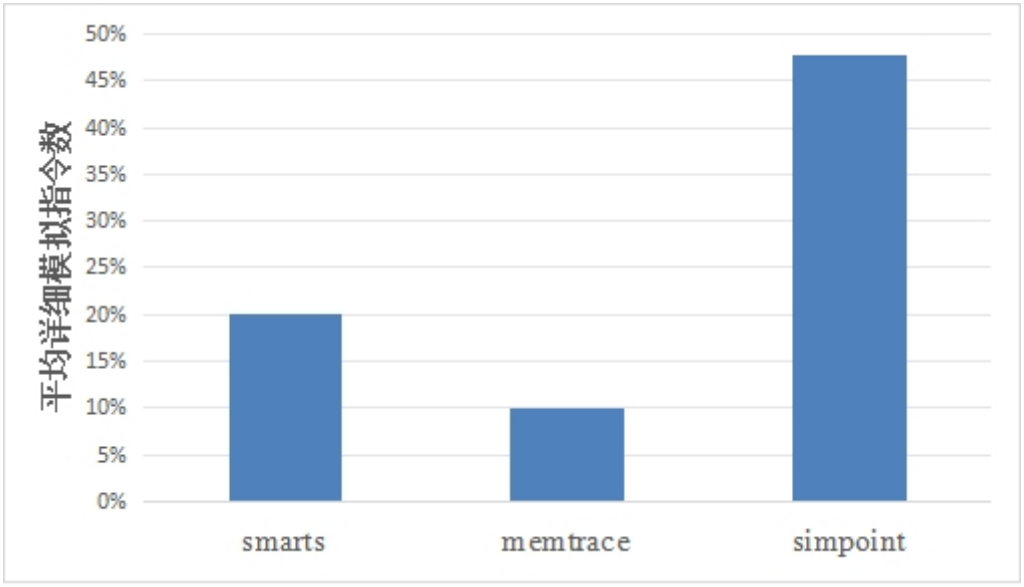
baseline是采用FullSystem下全部详细模拟，所以只选择了几个运行速度较快的

模拟运行CPU参数

参数类型	模拟值
L1i/L1d cache size/assoc	64KB/4
L2 cache size/assoc	1MB/4
L3 cache size/assoc	16MB/16
load entries	72
store entries	56
reorder buffer entries	160
frequency	3GHz
memory bandwidth	25396MB/s
memory latency(CL)	10.080ns







访存轨迹方法平均IPC误差更小且更为稳定，平均误差仅为1.7%，加之图14，当L3的命中率误差较小时，IPC的误差也偏小，这说明对缓存的恢复起到了较好的效果，但并不绝对，这是因为微体系结构并不只缓存一种。对于模拟指令数来说，由图15可知，应用了访存轨迹技术比起单纯SMARTS少了一半的指令数，这是由于无需进行传统预热，此外在这三个测试集上，Simpoint模拟指令数最多，一方面是由于其需要进行较长的传统预热，另一方面是因为所选取的负载不够长，重复的阶段并不多，导致所占的平均详细模拟指令数较高，在更长的负载中，其平均详细模拟指令数应当下降。