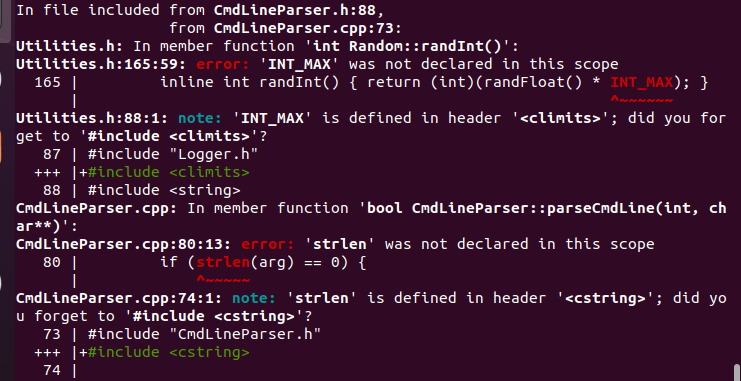
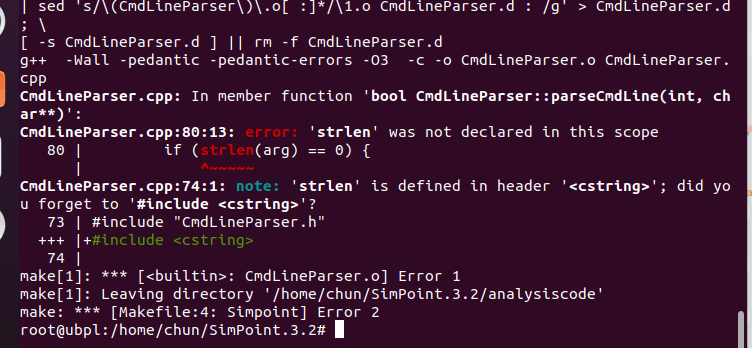
**Simpoint部署使用**

**Simpoint编译报错问题：**

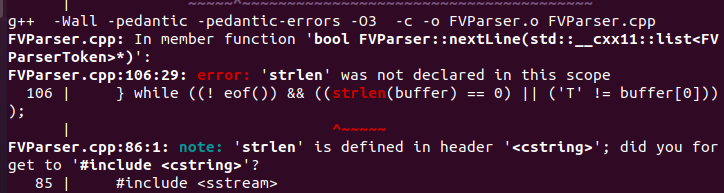
官方的Simpoint3.2直接编译是无法通过的，需要后面一步步修改



在Utilities.h文件中添加头文件:‘#include <climits>’ 解决

试着往CmdLineParser.cpp文件中添加头文件:‘#include <[cstring](https://so.csdn.net/so/search?q=cstring&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_43283275/article/details/_blank)>’

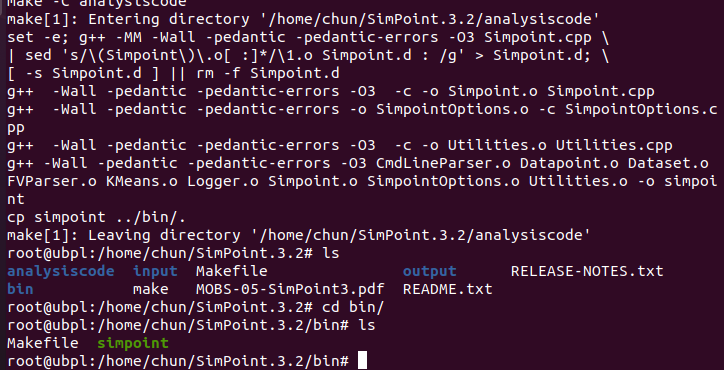
解决，但报别的错



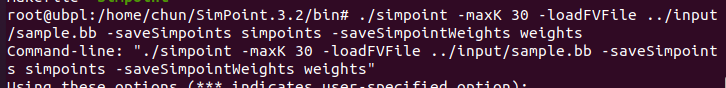
又报错

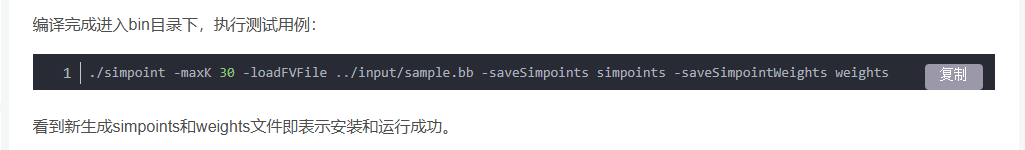
解决方法是在Simpoint.cpp文件中，在调用Utilities::check()函数的地方，129行，660 639 625 576 563 486 475 465 437 303 252 178 194 216 226每行的第一个参数前加强制类型转换bool。

改完后编译成功



执行simpoint本身自带的测试用例来生成simpoints和weights文件，但没有用到gem5







**环境配置：**

Ubentu 20.04

python3.9.7(与python版本关系不大，3.6以上就可以)

Simpoint3.2(官网的直接编译行不通，很多报错)

gem5 v20.1.0.4 (如使用gem5 v20版本，最好用ubentu20.04)

benchmark：graph500-2.1.4

交叉编译工具 aarch64-linux-gnu-gcc

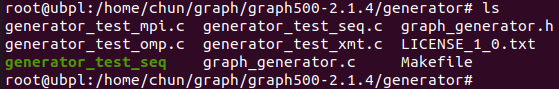
graph500：一套用来测试高性能集群的图计算性能的标准，可以根据用户的需求生成任意量级的数据用以测试产品的图计算能力。

在 Graph500 中，R-MAT（Recursive MATrix）的规模（由 -s 参数设置）通常指的是用于生成图的一个参数，R-MAT 的边因子（由 -e参数设置)，这些输入决定了生成的图的大小和复杂度，可以自行通过如 -o “-s 14 -e 12”来设置图的大小。

**Gem5 模拟运行graph500-2.1.4**

跑了graph500-2.1.4的2个用例

/home/chun/graph/graph500-2.1.4/generator/generator\_test\_seq 运行16个小时，这个用例耗时太长了，等了很久后面用另一个小点的seq-csr用例：



/home/chun/graph/graph500-2.1.4/seq-csr/seq-csr -s 13 -e 12 运行2小时20分：



实际命令：

build/ARM/gem5.opt configs/example/se.py \

--cpu-type O3\_ARM\_v7a\_3 \

--cpu-clock 2.5GHz \

--num-cpu 1 \

--mem-type DDR3\_2133\_8x8 --mem-size 16GB \

--l2-hwp-type StridePrefetcher \

--caches --l2cache --l1i\_size 64kB \

--l1d\_size 32kB --l2\_size 1MB \

--l1i\_assoc 8 --l1d\_assoc 8 \

--l2\_assoc 16 --cacheline\_size 128 \

-c /home/chun/graph/graph500-2.1.4/seq-csr/seq-csr \

-o "-s 13 -e 12" \

**Simpoint部署使用大致可以分为4个流程**

生成 Simpoint Checkpoint 分3步：

1. Profiling 采样，执行一轮 workload，收集程序行为信息，生成BBV
2. Cluster 聚类，得到权重最高的多个程序片段（节点），生成simpoints和weights
3. Checkpointing 生成，再执行一轮 workload，根据聚类的结果生成对应的 Checkpoint
4. 最后一步重载checkpoint

**1、分析工作负载并生成 SimPoint BBV 文件**

生成Simpoint文件不一定要用gem5，看到有人用Pinplay、simplescalar工具也可以。

gem5 运行文件夹中生成一个 SimPoint Basic 块向量文件（simpoint.bb.gz）：

相当于正常运行一遍程序（但是因为是NonCachingSimpleCPU，不需要像O3一样需要处理cache，比直接用gem5仿真会快很多）

实际命令：

build/ARM/gem5.opt configs/example/se.py \

--cpu-type=NonCachingSimpleCPU \

-c /home/chun/graph/graph500-2.1.4/seq-csr/seq-csr \

--simpoint-profile --simpoint-interval 10000000 \

-o ' -s 13 -e 12' \

--simpoint-profile

--simpoint-interval <间隔长度>表示指令数。默认间隔长度为 10M 指令

-c 指定工作负载，即编译生成的benchmark二进制文件

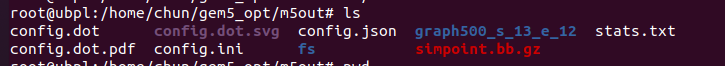
*# 注: gem5 只有 atomic cpu 支持生成BBV*

*否则会报错*

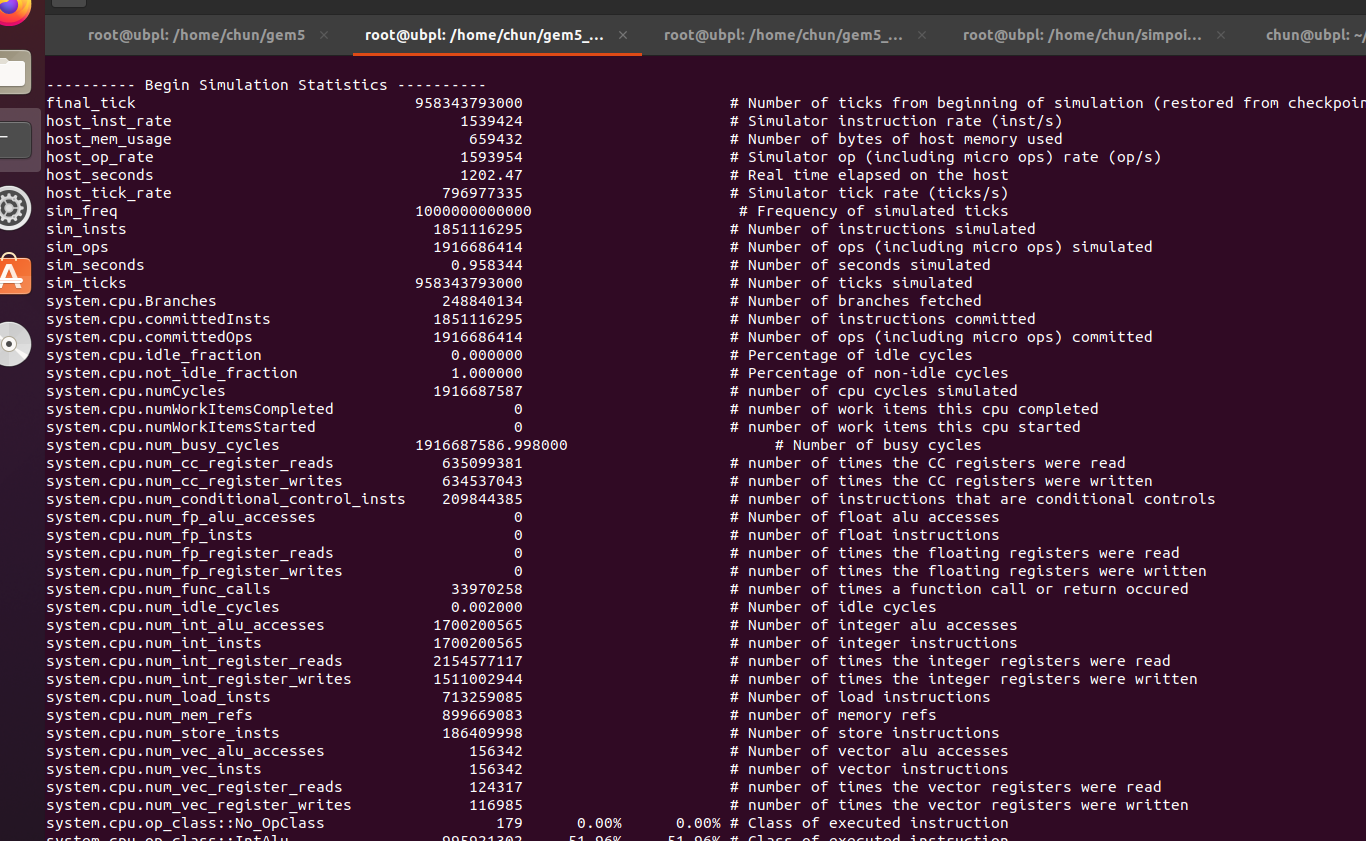
*# 注：SimPoint 分析应使用单个 AtomicSimpleCPU 配置运行。不支持多核模拟*

system.cpu.addSimPointProbe(20)*# 注: memory 也必须是 atomic 模式 (而不是 timing 模式)*

运行结果：



生成的stat.txt



## 2、SimPoint 分析，生成simpoints和weights文件

参数：

-maxK N 表示K的取值从1到N，然后用BIC选择最合适的k值。

-saveSimpoints 将最终选择的点信息放到文件

-saveSimpointWeights 将选择的每个点的权重放到文件

-loadFVFile 载入Frequence vector文件，就是步骤1生成的BBV文件

-inputVectorsGzipped 表明输入的FV文件是否经过gizpped压缩了，示例中的文件没有经过压缩，所以没有使用此参数

simpoint支持的参数还有很多，具体可以看SimPoint 3.0: Faster and More Flexible Program Phase Analysis 这篇论文

或者gem5 Usage: se.py [options]，

[https://github.com/prajyotgupta/ECE752/blob/e706c606166f7c2965f9dc5c4b450709b25f446e/gem5\_OG/param#L153](https://github.com/prajyotgupta/ECE752/blob/e706c606166f7c2965f9dc5c4b450709b25f446e/gem5_OG/param" \l "L153)

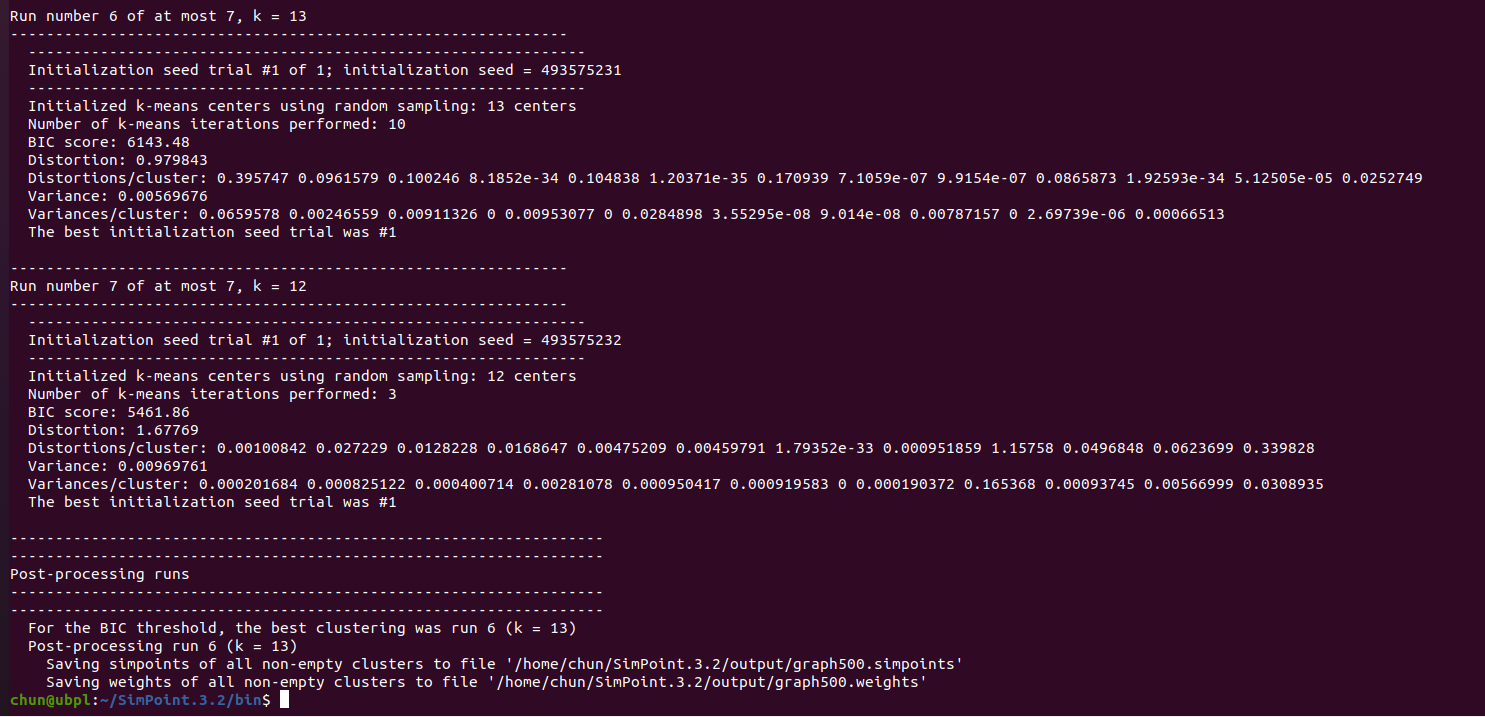
使用 SimPoint 3.2 基于 BBV 文件生成 SimPoint 分析的示例命令如下：

./simpoint -loadFVFile /home/chun/gem5\_opt/m5out/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12\_init/simpoint.bb.gz -numInitSeeds 1 -maxK 30 -saveSimpoints /home/chun/SimPoint.3.2/output/gem5/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12.simpoints -saveSimpointWeights /home/chun/SimPoint.3.2/output/gem5/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12.weights -inputVectorsGzipped

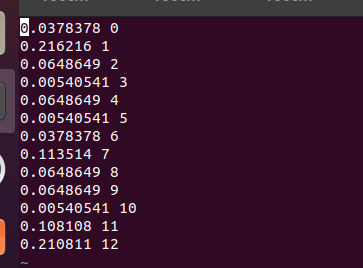
./simpoint -maxK 30 -numInitSeeds 1 -loadFVFile /home/chun/gem5\_opt/m5out/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12\_init/simpoint.bb.gz -inputVectorsGzipped -saveSimpoints /home/chun/SimPoint.3.2/output/gem5/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12.simpoints -saveSimpointWeights /home/chun/SimPoint.3.2/output/gem5/graph500\_s\_13\_e\_12/graph500\_s\_13\_e\_12.weights

./simpoint -maxK 30 -numInitSeeds 1 -loadFVFile /home/chun/gem5\_opt/m5out/simpoint.bb.gz -inputVectorsGzipped -saveSimpoints /home/chun/SimPoint.3.2/output/graph500.simpoints -saveSimpointWeights /home/chun/SimPoint.3.2/output/graph500.weights

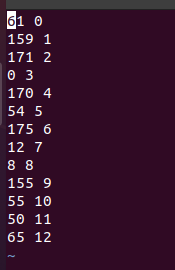
执行结果



Weights：



Simpoints：



****参数设置：****

****simpoint-interval length 10000000 即0.01 billion****

****warmup length 1000000 即0.001 billion****

其中1片段对应权重0.216，起始指令为1.59 billion

## 3、在 gem5 中采取 SimPoint 检查点，生成checkpoint

要根据 SimPoint 分析获取 gem5 检查点，使用以下命令：

% build/ARM/gem5.opt <base options> configs/example/fs.py --take-simpoint-checkpoint=<simpoint file path>,

<weight file path>,<interval length>,<warmup length>

<rest of fs.py options>

<simpoint 文件>和<weight 文件>由上述 SimPoint 分析步骤生成。<interval length>和<warmup length>都在指令数量中。gem5 检查点将在指定的 SimPoint 之前生成<预热长度>指令。

# 注：SimPoint 分析生成checkpoint应使用单个 AtomicSimpleCPU 配置运行。不支持多核模拟,否则会报错！

具体参考以下命令：

build/${ARCH}/gem5.${METHOD} --outdir=${SE\_OUT\_DIR\_CHECKPOINT} \

configs/example/se.py \

--take-simpoint-checkpoint=\

${SE\_simpoint\_file\_path},${SE\_weight\_file\_path},\

${interval\_length},${warmup\_length} \

--cpu-type=AtomicSimpleCPU \

${CACHE\_PRAM} \

${MEM\_PRAM} \

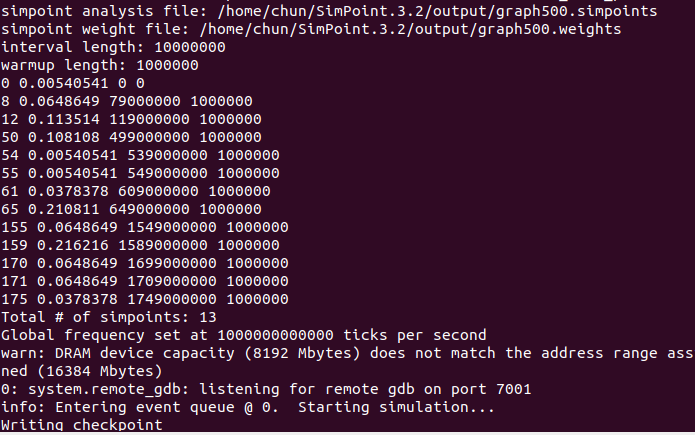
-c ${SE\_ELF\_ROUTE} \

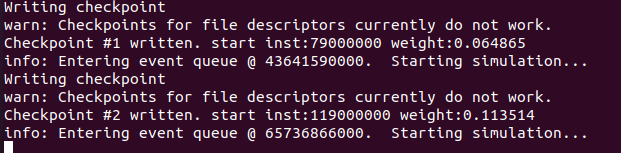
-o "${SE\_INPUT\_ROUTE}${OPTIONS}"

实际运行命令：



生成checkpoint用了一个小时

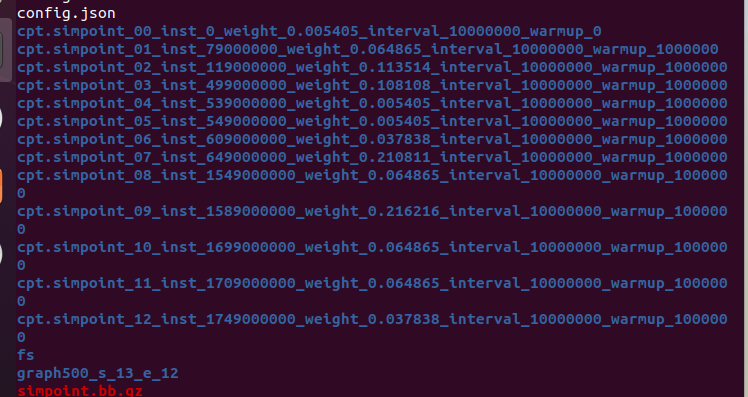




gem5使用该weight和simpoint产生checkpoint时，会在 0.079billion处产生checkpoint，然后经过0.001billion的instruction，结束warm up，进入真正的 0.119 的roi 指令。

实际运行结果：

生成了13个checkpoint



cpt.simpoint\_11\_inst\_1709000000\_weight\_0.064865\_int

erval\_10000000\_warmup\_1000000

│ ├── m5.cpt

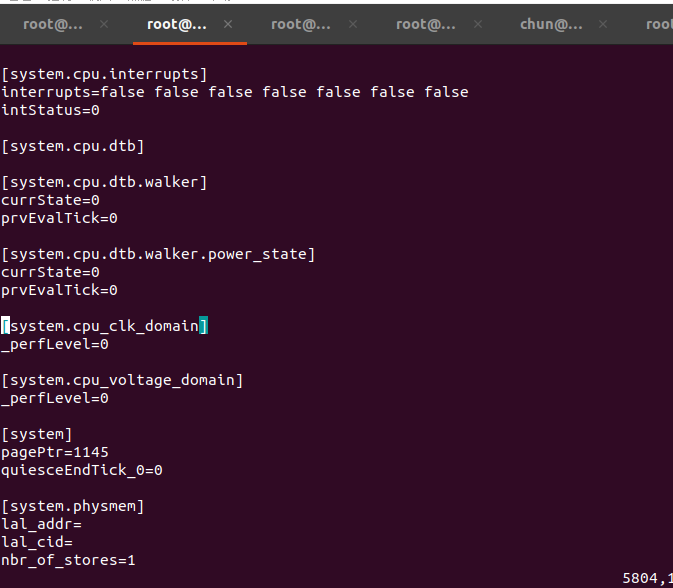
│ └── system.physmem.store0.pmem

m5.cpt即为当前cpu运行状态的快照，比如寄存器值，页表映射关系之类的。

system.physmem.store0.pmem则是当时的内存的状态。

因此有了cpt和pmem，gem5就可以从任意一个checkpoint restore，开始执行

m5.cpt：



## 4、gem5重载从checkpint 恢复

参数：

<base options> configs/example/fs.py

--restore-simpoint-checkpoint -r <N>

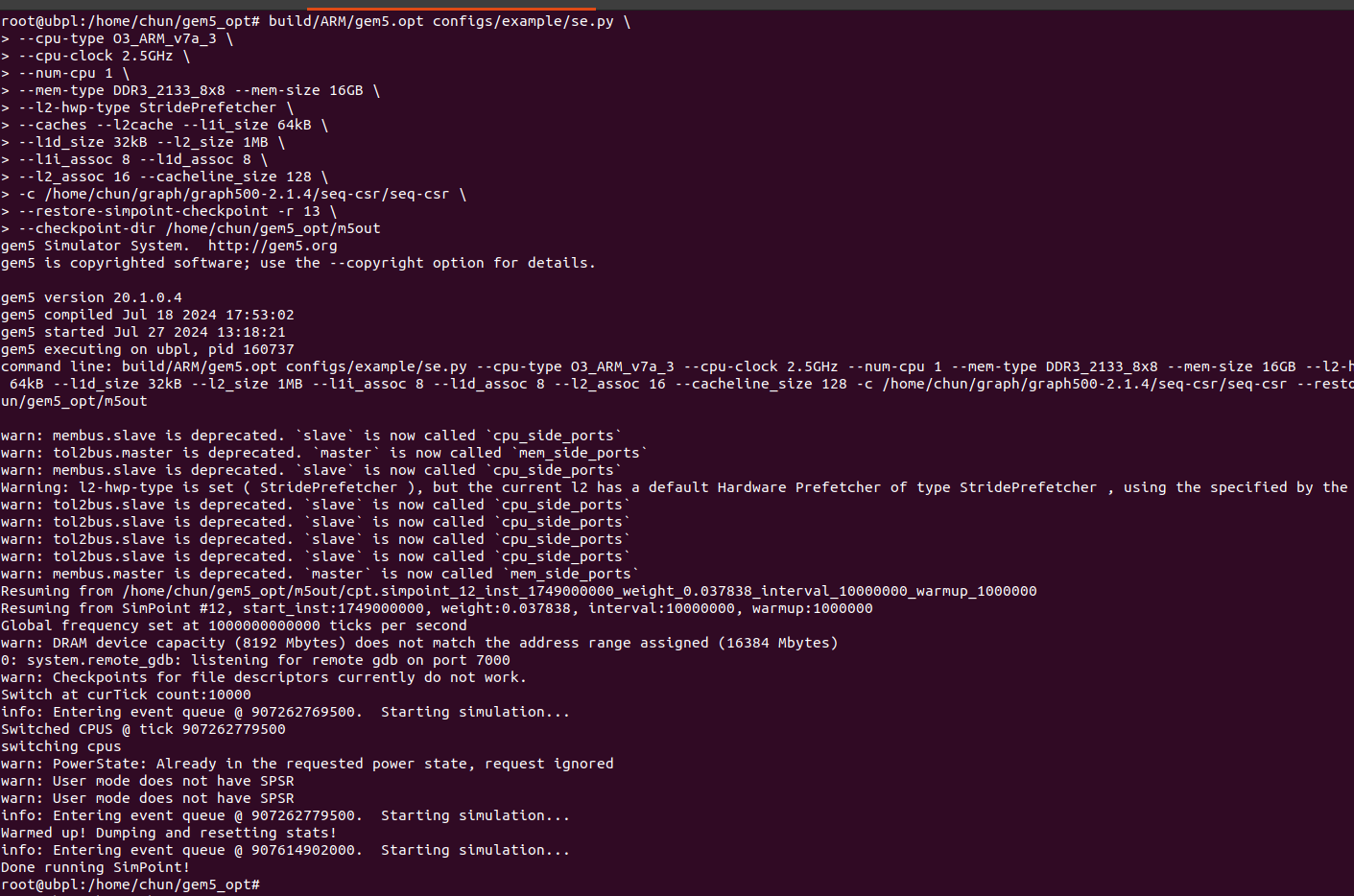
--checkpoint-dir <simpoint checkpoint path>

<rest of fs.py options>

<N> 是 （SimPoint 索引 + 1）。例如，“-r 1”将从 SimPoint #0 恢复。

此命令将从单个指定的 SimPoint 检查点恢复。模拟预热周期（从检查点文件夹名称自动识别）后，gem5 统计信息将被转储并重置。

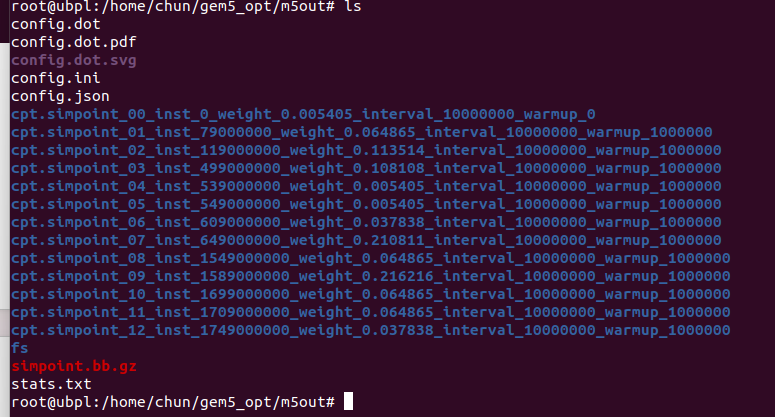
运行命令：

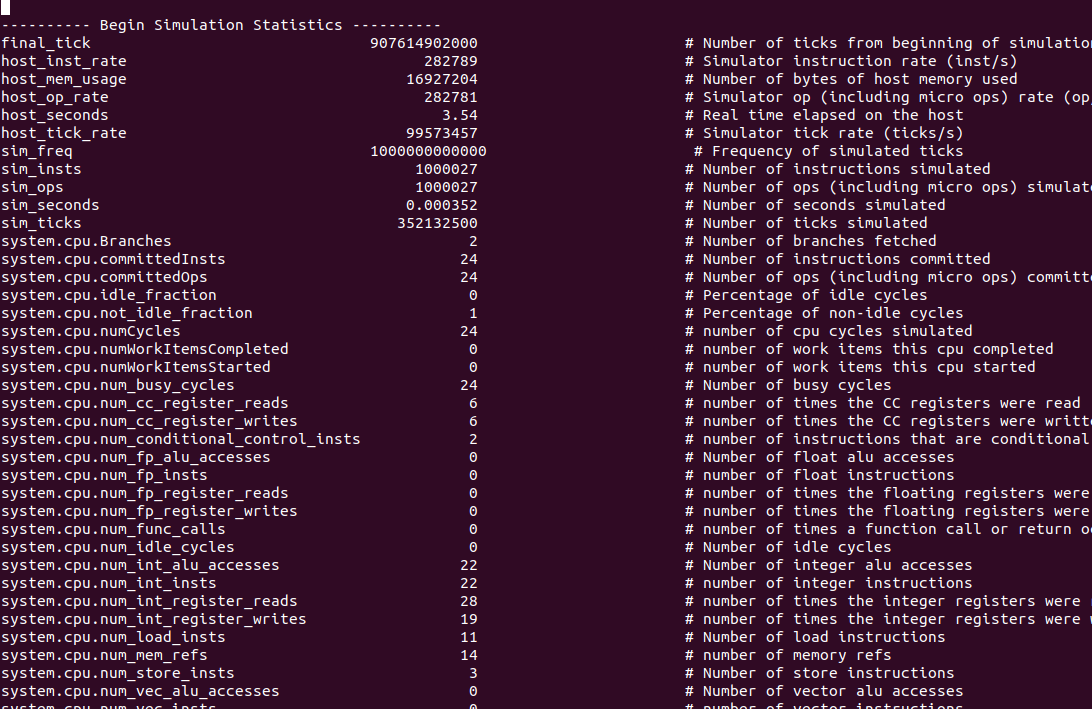


-r 13比较特殊，这里计数是按1开头的，因此-r 13实际上对应 第3步中的第12个checkpoint。

生成的stat.txt文件包含了gem5模拟过程中产生的各种统计信息，

用户可以通过分析这些统计信息来评估模拟计算机系统的性能情况。

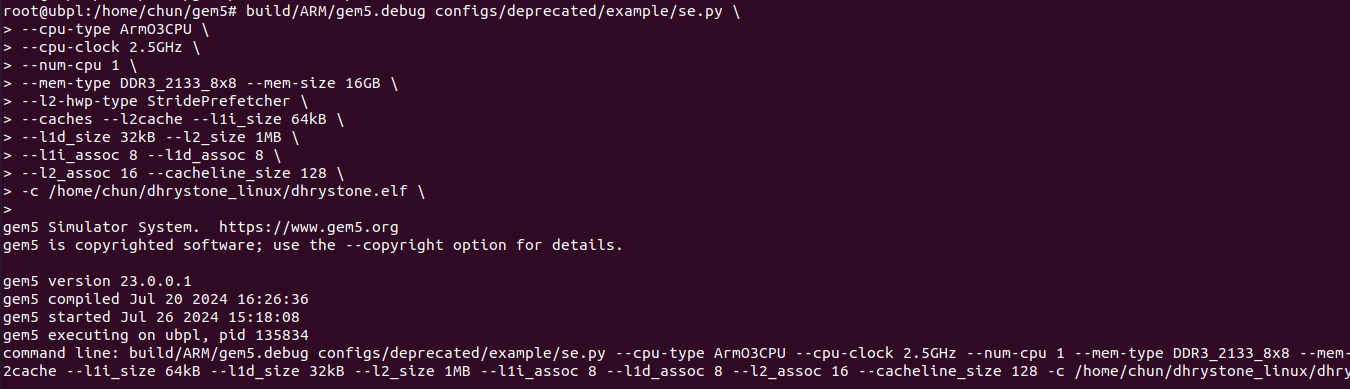


重载生成的stat.txt文件

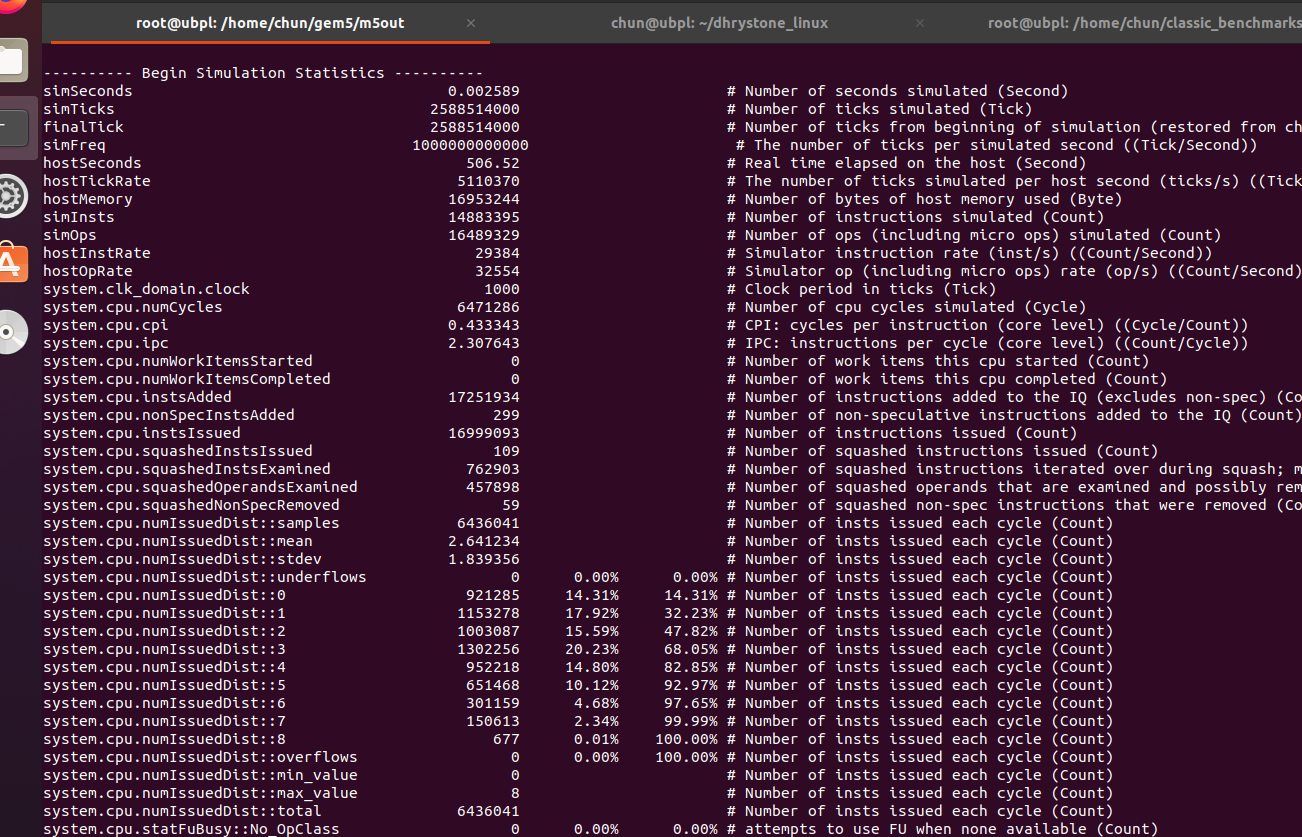
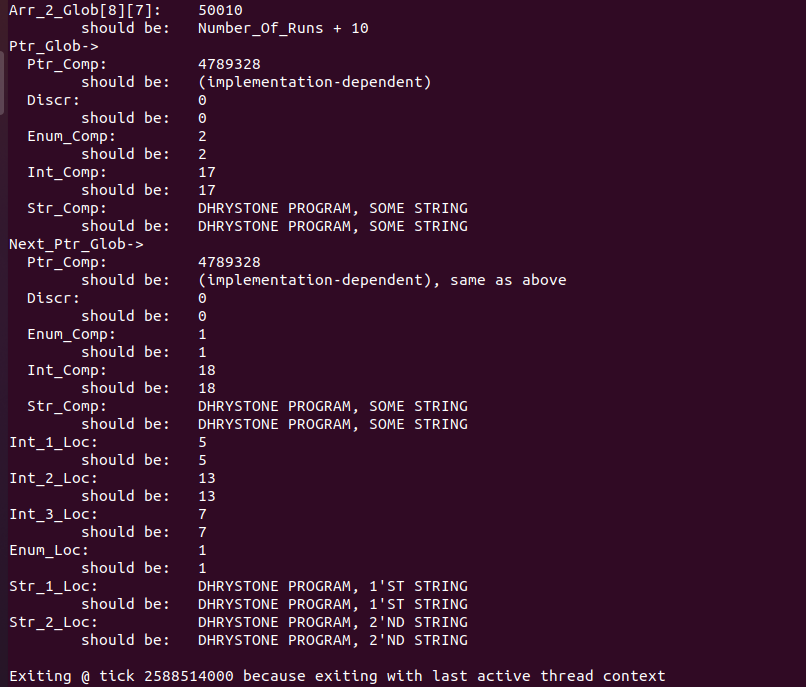
* simSeconds 表示模拟的秒数。这个值表示模拟的时钟的累计时间。
* simTicks 表示模拟的时钟滴答数，可以看作时钟周期。滴答（tick）是模拟中使用的最小时间单位。
* finalTick 表示模拟从开始到结束的时钟滴答数，即总的时钟周期数。注意，这个值不会重置，即使模拟从检查点恢复。
* simFreq 表示模拟时钟的频率。它表示每秒钟有多少滴答（时钟周期）。
* hostSeconds 指在宿主机上，表示真实时间中的秒数。这个值表示主机上实际流逝的时间。
* hostTickRate 指在宿主机上，表示每秒钟模拟的时钟周期数。它反映了模拟的速度。
* hostMemory 表示模拟器在宿主机上使用了多少字节的内存。
* simInsts：指模拟器模拟了多少条指令。指令（instruction）是计算机中执行的最小单位。这个数值越大，表明模拟器的性能越好。
* simOps：指模拟器模拟了多少个操作（包括微操作）。这个数值越大，表明模拟器的性能越好。
* hostInstRate：指模拟器的指令模拟速率，单位是指令/秒。这个数值越大，表明模拟器的性能越好。
* hostOpRate：指模拟器的操作模拟速率（包括微操作），单位是操作/秒。这个数值越大，表明模拟器的性能越好。

**Gem5模拟运行dhrystone**

运行命令(8分钟左右，这个benchmark太小了)：



运行结果：



**环境配置：**

Ubentu 20.04

python3.9.7(与python版本关系不大，3.6以上就可以)

Simpoint3.2(官网的直接编译行不通，很多报错)

gem5 v23.0.0.1 (如使用gem5 v20版本，最好用ubentu20.04)

benchmark： dhrystone

交叉编译工具 aarch64-linux-gnu-gcc

**1、分析工作负载并生成 SimPoint BBV 文件**

gem5\_23.0.0.1已经准备弃用se.py和fs.py，通过其相关介绍，gem5现在推荐使用gem5\_library。

arm-simpoint-benchmarks.py是基于gem5\_library构建的一个极简单的支持生成simpoint BBV的配置脚本，将此脚本放在config目录任意文件夹下即可，这里推荐使用目录：configs/example/gem5\_library/

实际命令：

build/ARM/gem5.debug configs/example/gem5\_library/arm-simpoint-benchmarks.py \

-c /home/chun/dhrystone\_linux/dhrystone.elf \

--cpu-type=NonCachingSimpleCPU \

--simpoint-profile --simpoint-interval 1000 \

#-o /home/chun/gem5\_opt/m5out

运行结果：



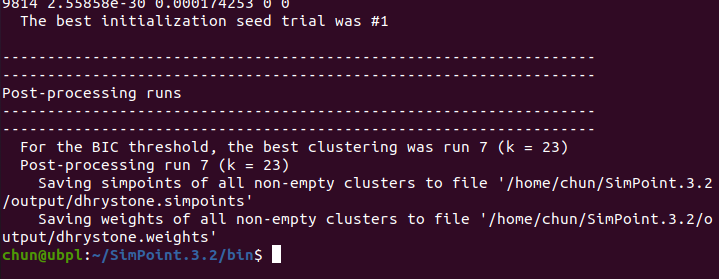
## 2、SimPoint 分析，生成simpoints和weights文件

基于 BBV 文件生成 SimPoint 分析的示例命令如下：

实际命令：

./simpoint -maxK 30 -numInitSeeds 1 -loadFVFile /home/chun/SimPoint.3.2/input/simpoint.bb.gz -inputVectorsGzipped -saveSimpoints /home/chun/SimPoint.3.2/output/dhrystone.simpoints -saveSimpointWeights /home/chun/SimPoint.3.2/output/dhrystone.weights

执行结果

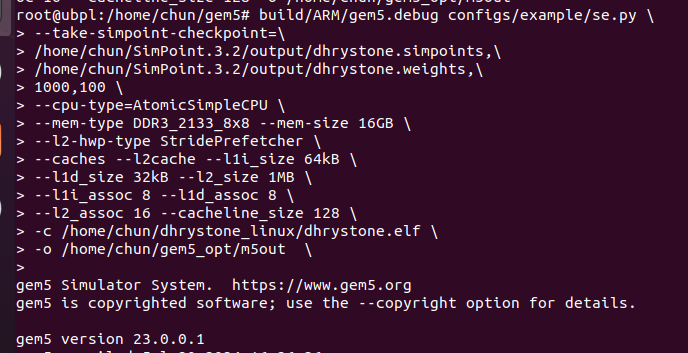




## 3、在 gem5 中采取 SimPoint 检查点，生成checkpoint

# 注：SimPoint 分析生成checkpoint应使用单个 AtomicSimpleCPU 配置运行。不支持多核模拟,否则会报错！

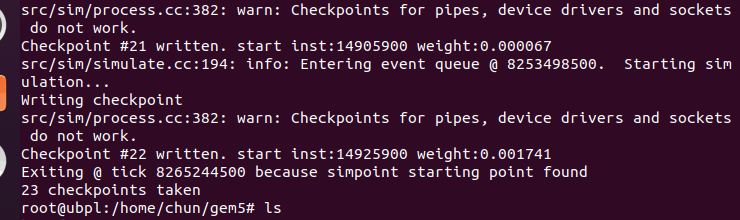
实际运行命令：

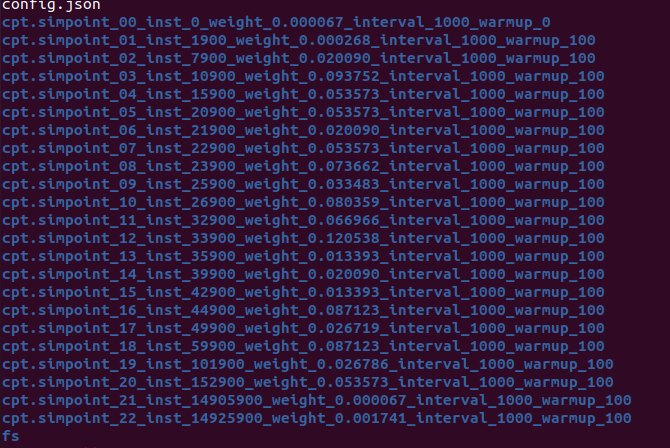


生成checkpoint用了一个小时(好像是因为设置的inteval太小了)

实际运行结果：

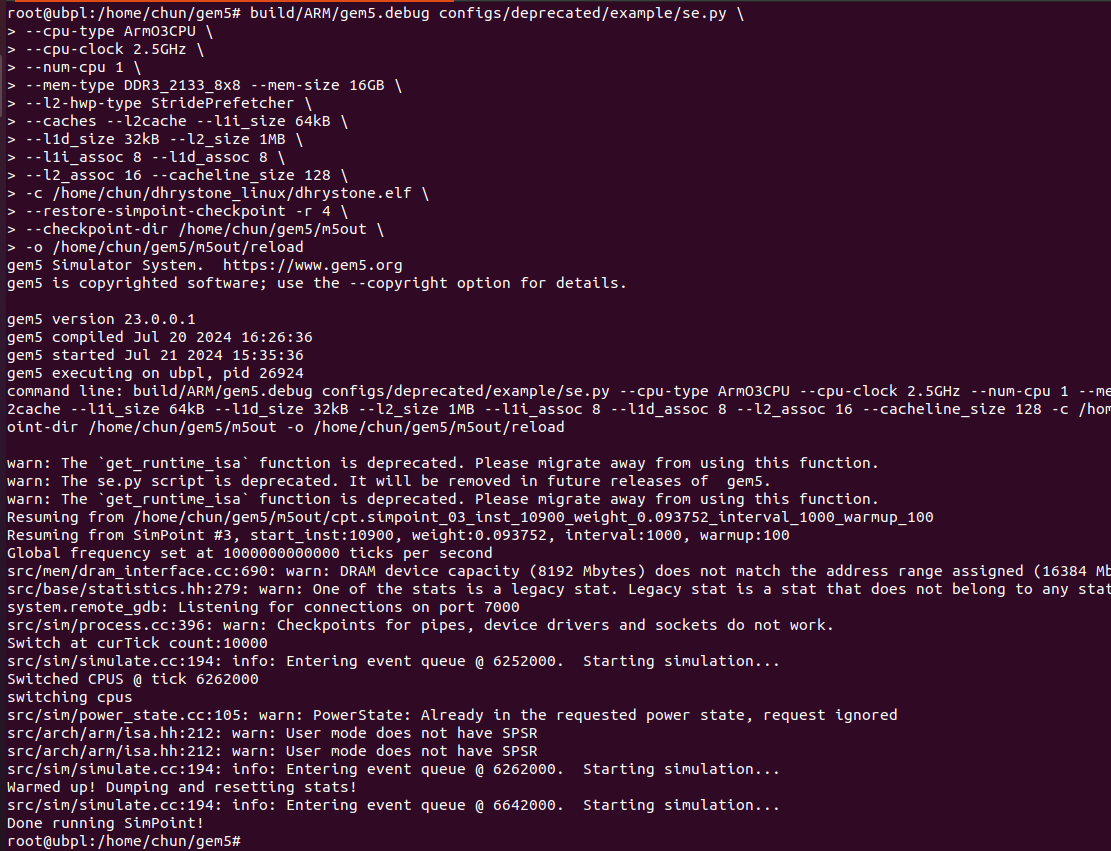
生成了23个checkpoint



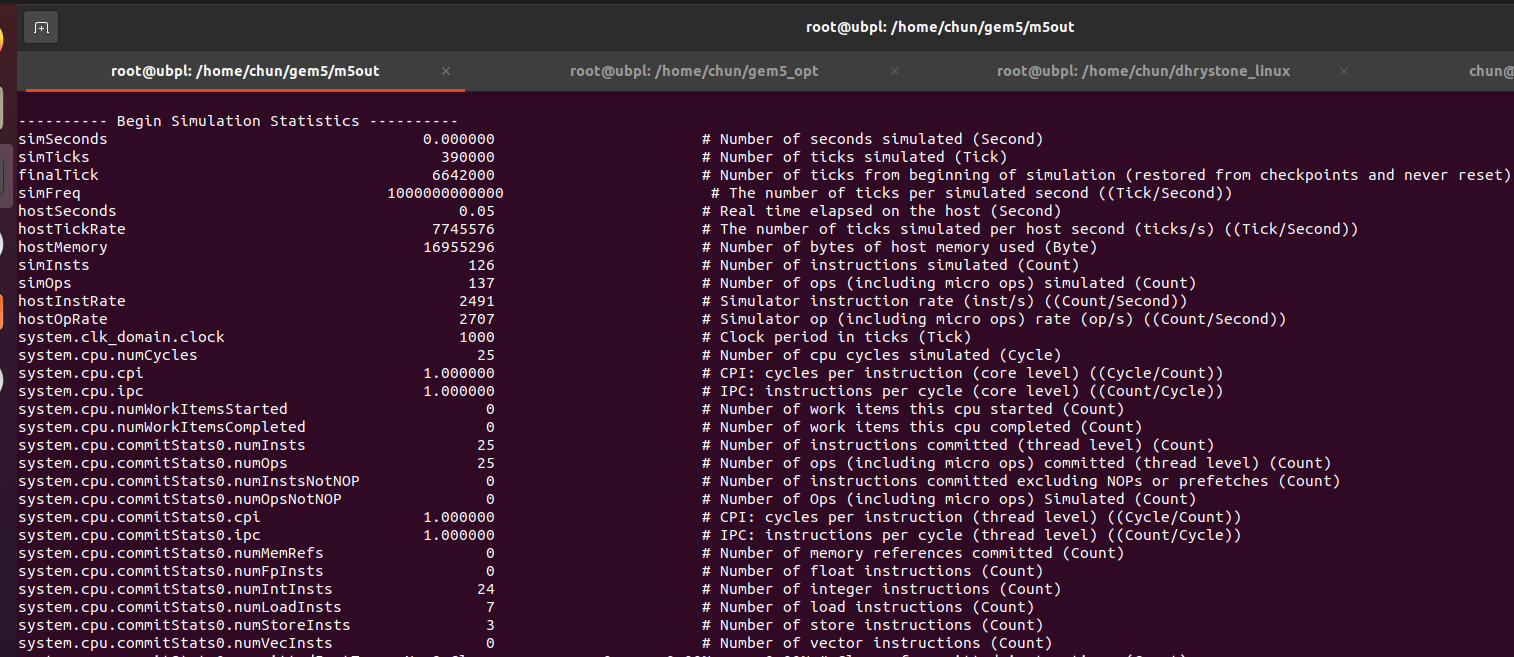


## 4、gem5重载从checkpint 恢复

实际运行了30秒左右，实际运行命令：





重载生成的stat.txt文件

**Reference：**

* [Simpoint 那些事 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/584300250?utm_psn=1800319227682484224)
* [Simpoint 在 GEM5 里加速仿真, 1 minute = N day(s) - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/453370789)
* http://www-cse.ucsd.edu/~calder/papers/ASPLOS-02-SimPoint.pdf
* SimPoint: Picking Representative Samples to Guide Simulation 2006 —— 对 simpoint整个过程比较好的阐述
* <https://old.gem5.org/Simpoints.html>
* SimPoint 3.0: Faster and More Flexible Program Phase Analysis 2005
* Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications
* https://cseweb.ucsd.edu/~calder/simpoint/sim-points-FAQ.htm