一、设计目的

在多核体系模拟器中实现面向混合式内存结构的共享缓存替换算法，并对其性能和功耗进行分析比较。

二、前序工作

关于下载、编译和运行gem5的介绍

1. 确保正在用的Ubuntu版本是15.04（64bit）或者是更高的版本，更新方法：

sudo apt-get update;

sudo apt-get dist-upgrade

2. 安装以下必要的软件包

sudo apt-get install git mercurial scons build-essential swig libfreetype6-dev python-dev python-pip python-lxml python-pydot zlib1g-dev libgoogle-perftools-dev protobuf-compiler libprotobuf-dev m4 graphviz

pip install objectpath yattag pytz pygal cairosvg tinycss cssselect seaborn matplotlib pandas flufl.enum

3. 检测 gem5:

git clone https://github.com/mcai/gem5\_hmm\_llc\_replacement.git.

4. 构建 gem5:

cd gem5/;./compile\_ALPHA.sh

5. 下载磁盘映象和Linux内核文件作为在gem5中ALPHA全系统仿真的所需的文件：

a) 全系统文件: http://www.m5sim.org/dist/current/m5\_system\_2.0b3.tar.bz2

b) linux dist: http://www.m5sim.org/dist/current/linux-dist.tgz

提取这些文件并且放置于 /home/<current\_user>/Tools/GEM5/system/.

准备就绪的目录布局应该像以下这样：

.

├── binaries

│   ├── console

│   ├── tsb\_osfpal

│   ├── vmlinux

│   ├── vmlinux-22-22-64

│   ├── vmlinux\_2.6.27-gcc\_4.3.4

│   ├── x86\_64-vmlinux-2.6.22.9

│   └── x86\_64-vmlinux-2.6.22.9.smp

├── configs

│   ├── linux-2.6.22.9

│   ├── linux-2.6.22.9.smp

│   ├── linux-2.6.25.1

│   └── linux-2.6.28.4

└── disks

├── BigDataBench-gem5.img

├── linux-bigswap2.img

├── linux-parsec-2-1-m5-with-test-inputs.img

├── linux-x86.img

└── x86root-parsec.img

6. 运行 gem5:

a) 例如运行一个多核试验

./run\_all\_experiments\_ALPHA.py

实验结果在 results/alpha/中.

b) 例如运行 CC-NUMA multi-core experiments:

./run\_all\_experiments\_ALPHA\_no\_checkpoints.py

实验结果在 results/alpha\_ccnuma\_no\_checkpoints/中.

三、对于三个文档的说明

|  |  |
| --- | --- |
| TODOS | 过程中的工作细节 |
| CHANGES | 内部文件的修改和外部文件的修改 |
| run\_all\_experiments\_CPU\_X86.SE | 运行测试程序 |
| analyze\_all\_experiments\_result\_CPU2006 | 分析测试结果 |

【TODOS】

1、过程中需要抓紧解决的问题

（1） 测试多个工作负载的映射和运行

（2） 支持CPU2006的运行

2、过程中优先完成的工作

（1） 修复在各个工作负载执行中的Bug

（2） 合并gem5 CPU2006脚本。

3、工作细节

（1）将near memory 和far memory的带宽比设为: 3:1，内存比设为1GB :3GB Bandwidth Utilization ?= Bandwidth Ratio

class NearMemory(m5.objects.SimpleMemory):

bandwidth = '51.2GB/s' = 12.8 \* 4

class FarMemory(m5.objects.SimpleMemory):= '12.8GB/s' # representative of a x64 DDR3-1600 channel.

Bandwidth Utilization:

1573338118

537934346

system.mem\_ctrls0.num\_reads::total 3825587 # Number of read requests responded to by this memory

system.mem\_ctrls0.num\_writes::writebacks 2146150 # Number of write requests responded to by this memory

system.mem\_ctrls0.num\_writes::total 2146150 # Number of write requests responded to by this memory

system.mem\_ctrls1.num\_reads::total 1076130 # Number of read requests responded to by this memory

system.mem\_ctrls1.num\_writes::writebacks 965645 # Number of write requests responded to by this memory

system.mem\_ctrls1.num\_writes::total 965645 # Number of write requests responded to by this memory

1. 工作集的随机划分

（每个时期）内存带宽利用率跟踪16芯率模式：例如，在所有的内核相同的CPU2006程序

void

Process::allocateMem(Addr vaddr, int64\_t size, bool clobber)

{

int npages = divCeil(size, (int64\_t)PageBytes);

Addr paddr = system->allocPhysPages(npages);

pTable->map(vaddr, paddr, size, clobber ? PageTableBase::Clobber : 0);

}

Addr

System::allocPhysPages(int npages)

{

Addr return\_addr = pagePtr << PageShift;

pagePtr += npages;

Addr next\_return\_addr = pagePtr << PageShift;

AddrRange m5opRange(0xffff0000, 0xffffffff);

if (m5opRange.contains(next\_return\_addr)) {

warn("Reached m5ops MMIO region\n");

return\_addr = 0xffffffff;

pagePtr = 0xffffffff >> PageShift;

}

if ((pagePtr << PageShift) > physmem.totalSize())

fatal("Out of memory, please increase size of physical memory.");

return return\_addr;

}

（3）目的: 确保每时期near memory与far memory利用率之比=near memory与far memory带宽比。

解决方案： 分配的方式。

实施:

 附加硬件：在每个缓存行中添加一个“hmm\_memory\_type”位：“near memory（True）”或“far memory（False）”）。

在高速缓存块插入：

规则：“near memory”数据只能以“hmm\_memory\_type”位设置为“near memory”的方式插入。

同样，“far memory”数据只能以“hmm\_memory\_type”位设置为“far memory”的方式插入。

在每个阶段结束：

检索“near:far memory”带宽比。

检索“near:far memory”带宽利用率

如果“near:far memory”带宽利用率>“near:far memory”带宽比：

含义：“near memory”被过度利用，应该节省更多的“near memory”高速缓存未命中。

行为：从“far memory”分配一个LRU方式到“near memory”。

否则如果“near：far memory”带宽利用率<“near：far memory”带宽比：

含义：“far memory”被过度利用，应该节省更多的“far memory”高速缓存未命中。

行为：从“near memory”分配一个LRU方式到“far memory”。

其他：

没有行为

4、添加统计每个近存远存的页面分配。

\* 如果分配一页页处理后，混合存储系统配置将在 Process::allocatemem（..）中完成。

void

Process::allocateMem(Addr vaddr, int64\_t size, bool clobber)

{

int npages = divCeil(size, (int64\_t)PageBytes);

Addr paddr = system->allocPhysPages(npages);

pTable->map(vaddr, paddr, size, clobber ? PageTableBase::Clobber : 0);

}

【CHANGES】

1、对内部文件的修改

（1） 在src/mem/cache/tags/中添加了一个hmm.hh和hmm.cc文件

hmm算法是在传统的LRU算法上进行了修改。

**Hmm.hh**

class HMM : public BaseSetAssoc

{

public:

typedef HMMParams Params;

/\*\*

\*构建和初始化这个标签存储。

\*/

HMM(**const** Params \*p);

/\*\*

\* 析构函数

\*/

~HMM() {}

/\*\*

\*命中块

\*/

CacheBlk\* accessBlock(ThreadID threadId, Addr pc, Addr addr, bool is\_secure, Cycles &lat,

int context\_src);

/\*\*

\*寻找替换块

\*/

CacheBlk\* findVictim(ThreadID threadId, Addr pc, Addr addr);

void insertBlock(PacketPtr pkt, BlkType \*blk);

void invalidate(CacheBlk \*blk);

void endOfPhase();

EventWrapper<HMM, &HMM::endOfPhase> endOfPhaseEvent;

float nearFarMemoryChannelsRatio;

float perPhaseNearFarMemoryBandwidthsRatio;

int nearMemoryAssoc;

};

#endif // \_\_MEM\_CACHE\_TAGS\_HMM\_HH\_\_

**Hmm.cc**

HMM::HMM(**const** Params \*p)

: BaseSetAssoc(p),

endOfPhaseEvent(**this**),

nearFarMemoryChannelsRatio(0),

perPhaseNearFarMemoryBandwidthsRatio(0)

{

nearMemoryAssoc = assoc / 2; //每个缓存组包含的缓存块个数

**for**(**int** set = 0; set < numSets; set++) {

**for** (**int** i = 0; i < assoc; i++) {

BlkType \*b = sets[set].blks[i];

b**->**hmm\_memory\_type = i < nearMemoryAssoc;

}

}

schedule(endOfPhaseEvent, curTick() + 1);

}

/\*\*

\*命中块

\*/

CacheBlk\*

HMM::accessBlock(ThreadID threadId, Addr pc, Addr addr, **bool** is\_secure, Cycles &lat, **int** master\_id)

{

CacheBlk \*blk = BaseSetAssoc::accessBlock(threadId, pc, addr, is\_secure, lat, master\_id);

**if** (blk != NULL) {

*//将此块移动到MRU列表的头部*

sets[blk**->**set].moveToHead(blk);

DPRINTF(CacheRepl, **"set %x: moving blk %x (%s) to MRU\n"**,

blk**->**set, regenerateBlkAddr(blk**->**tag, blk**->**set),

is\_secure ? **"s"** : **"ns"**);

}

**return** blk;

}

/\*\*

\*寻找替换块

\*/

CacheBlk\*

HMM::findVictim(ThreadID threadId, Addr pc, Addr addr)

{

Addr farMemoryAddrStart= cache**->**system**->**farMemoryAddrStart;

**bool** hmm\_memory\_type = addr < farMemoryAddrStart;

**int** set = extractSet(addr);

*// grab a replacement candidate*

BlkType \*blk = NULL;

**for** (**int** i = assoc - 1; i >= 0; i--) {

BlkType \*b = sets[set].blks[i];

**if** (b**->**way < allocAssoc && b**->**hmm\_memory\_type == hmm\_memory\_type) {

blk = b;

**break**;

}

}

assert(!blk || blk**->**way < allocAssoc);

**if** (blk && blk**->**isValid()) {

DPRINTF(CacheRepl, **"set %x: selecting blk %x for replacement\n"**,

set, regenerateBlkAddr(blk**->**tag, set));

}

**return** blk;

}

/\*\*

\*插入块

\*/

**void**

HMM::insertBlock(PacketPtr pkt, BlkType \*blk)

{

BaseSetAssoc::insertBlock(pkt, blk);

**int** set = extractSet(pkt**->**getAddr());

sets[set].moveToHead(blk);

}

/\*\*

\*无效块

\*/

**void**

HMM::invalidate(CacheBlk \*blk)

{

BaseSetAssoc::invalidate(blk);

*// should be evicted before valid blocks*

**int** set = blk**->**set;

sets[set].moveToTail(blk);

}

**void**

HMM::endOfPhase()

{

**if**(nearFarMemoryChannelsRatio == 0)

nearFarMemoryChannelsRatio = **float**(cache**->**system**->**nearMemoryChannels) / cache**->**system**->**farMemoryChannels;

perPhaseNearFarMemoryBandwidthsRatio = cache**->**system**->**perPhaseNearFarMemoryBandwidthsRatio / 2 + perPhaseNearFarMemoryBandwidthsRatio / 2;

**if**(perPhaseNearFarMemoryBandwidthsRatio >0 && perPhaseNearFarMemoryBandwidthsRatio > nearFarMemoryChannelsRatio && nearMemoryAssoc < assoc - 1) {

nearMemoryAssoc++;

**for**(**int** set = 0; set < numSets; set++) {

**for** (**int** i = assoc - 1; i >= 0; i--) {

BlkType \*b = sets[set].blks[i];

**if** (!b**->**hmm\_memory\_type) {

b**->**hmm\_memory\_type = **true**;

**break**;

}

}

}

}

**else if** (perPhaseNearFarMemoryBandwidthsRatio >0 && perPhaseNearFarMemoryBandwidthsRatio < nearFarMemoryChannelsRatio && nearMemoryAssoc > 1) {

nearMemoryAssoc--;

**for**(**int** set = 0; set < numSets; set++) {

**for** (**int** i = assoc - 1; i >= 0; i--) {

BlkType \*b = sets[set].blks[i];

**if** (b**->**hmm\_memory\_type) {

b**->**hmm\_memory\_type = **false**;

**break**;

}

}

}

}

schedule(endOfPhaseEvent, curTick() + cache**->**system**->**numTicksPerPhaseBwTotalHist);

}

HMM\*

HMMParams::create()

{

**return new** HMM(**this**);

}

1. 在src/mem/cache/tags/Tags.py中添加了HMM： **(没找到Tags.py)**

class HMM(BaseSetAssoc):

type = 'HMM'

cxx\_class = 'HMM'

cxx\_header = "mem/cache/tags/hmm.hh"

1. 在src/mem/cache/tags/SConscript中添加Source('hmm.cc')
2. （4）在src/mem/cache/blk.hh中添加了class CacheBlk

In class CacheBlk:

```

/\*\*使用HMM缓存替换策略。\*/

bool hmm\_memory\_type;

```

1. 在src/mem/中添加了abstract\_mem.hh和abstract\_mem.cc

in `class AbstractMemory`:

```

/\*\* 计算当前时期从内存读的总字节数\*\*/

signed long perPhaseBytesRead;

/\*\* 计算当前时期从内存写的总字节数 \*\*/

signed long perPhaseBytesWritten;

/\*\* 记录上个时期完成时的sim ticks的数量\*\*/

signed long simTicksEndOfLastPhase;

/\*\* 直方图记录内存中总带宽在每个时期的间隔\*\*/

Stats::Histogram perPhaseBwTotalHist;

void endOfPhase();

EventWrapper<AbstractMemory, &AbstractMemory::endOfPhase> endOfPhaseEvent;

```

（6）在src/sim/system.cc做出修改 **没找到**

In `class System`:

```

*///* *分配npages连续的未使用的物理页面*

*/// 返回第一页的起始地址*

Addr allocPhysPages(int npages);

```

（7）对src/sim/system.hh and src/sim/system.cc做出修改

In `class System`:

```

Addr nearMemoryPagePtr;

Addr farMemoryPagePtr;

Addr farMemoryAddrStart;

uint64\_t init\_param;

bool hybridMemorySystem;

int nearMemoryChannels;

int farMemoryChannels;

int currentMemoryId;

/\*\* 用于测量此存储器总带宽的程序阶段中经过的周期数\*/

signed long numTicksPerPhaseBwTotalHist;

float perPhaseNearMemoryBandwidth;

float perPhaseFarMemoryBandwidth;

float perPhaseNearFarMemoryBandwidthsRatio;

/\*\*用于记录每相的直方图接近：远内存带宽比 \*/

Stats::Histogram perPhaseNearFarMemoryBwRatioHist;

void endOfPhase();

EventWrapper<System, &System::endOfPhase> endOfPhaseEvent;

Addr nearMemoryPagePtr;

Addr farMemoryPagePtr;

Addr farMemoryAddrStart;

（8）同样需要对src/sim/System.py中进行修改

In `class System`:

```

hybrid\_memory\_system = Param.Bool(**False**, **"Hybrid memory system"**)

near\_mem\_channels = Param.Int(**False**, **"Near memory channels"**)

far\_mem\_channels = Param.Int(**False**, **"Far memory channels"**)

```

（9）在src/sim/process.cc中修改

```

Process::allocateMem(Addr vaddr, int64\_t size, **bool** clobber)

{

**int** npages = divCeil(size, (int64\_t)PageBytes);

**if**(system**->**hybridMemorySystem) {

**for** (**int** i = 0; i < npages; i++) {

Addr paddr = system**->**allocPhysPages(1);

pTable**->**map(vaddr + (int64\_t)PageBytes \* i, paddr, (i == npages - 1) ? size - ((int64\_t)PageBytes \* i) : (int64\_t)PageBytes, clobber ? PageTableBase::Clobber : 0);

}

}

**else** {

Addr paddr = system**->**allocPhysPages(npages);

pTable**->**map(vaddr, paddr, size, clobber ? PageTableBase::Clobber : 0);

}

}

```

（10）在configs/common/MemConfig.py中修改

```

**def** config\_hybrid\_mem(options, system):

```

2、对外部文件的修改（即添加相应的接口）

在configs/common/Options.py中

```

*#内存选择*

parser.add\_option(**"--list-mem-types"**,

action=**"callback"**, callback=\_listMemTypes,

help=**"List available memory types"**)

parser.add\_option(**"--mem-type"**, type=**"choice"**, default=**"DDR3\_1600\_x64"**,

choices=MemConfig.mem\_names(),

help = **"type of memory to use"**)

parser.add\_option(**"--mem-channels"**, type=**"int"**, default=1,

help = **"number of memory channels"**)

parser.add\_option(**"--mem-ranks"**, type=**"int"**, default=**None**,

help = **"number of memory ranks per channel"**)

parser.add\_option(**"--mem-size"**, action=**"store"**, type=**"string"**,

default=**"512MB"**,

help=**"Specify the physical memory size (single memory)"**)

*# 混合内存选择*

parser.add\_option(**"--hybrid-memory-system"**, action=**"store\_true"**)

parser.add\_option(**"--near-mem-size"**, action=**"store"**, type=**"string"**, default=**"512MB"**,

help = **"Specify the near memory size"**)

parser.add\_option(**"--far-mem-size"**, action=**"store"**, type=**"string"**, default=**"4GB"**,

help = **"Specify the far memory size"**)

parser.add\_option(**"--near-mem-channels"**, type=**"int"**, default=1,

help = **"number of near memory channels"**)

parser.add\_option(**"--far-mem-channels"**, type=**"int"**, default=1,

help = **"number of far memory channels"**)

```

【run\_all\_experiments\_CPU\_X86.SE】

测试所设计的HMM算法

**--hybrid-memory-system --near-mem-size=1GB --far-mem-size=3GB --near-mem-channels=8 --far-mem-channels=2'** \(刚开始设置near-mem-size=4GB --far-mem-size=12GB，但由于电脑内存的限制，后改为 near-mem-size=1GB --far-mem-size=3GB)

具体细节

**def** run(benchmark, input\_set, l2\_size, l2\_assoc, l2\_tags, num\_threads, num\_phase\_ticks):

/\*\*

\*实验结果的存储位置

\*/

dir = **'../gem5\_results/x86\_SE/'** + benchmark + **'/'** + input\_set + **'/'** + l2\_size + **'/'** + str(l2\_assoc) + **'way/'** + l2\_tags + **'/'** + str(num\_threads) + **'c/'** + str(num\_phase\_ticks) + **'/'**

os.system(**'rm -fr '** + dir)

os.system(**'mkdir -p '** + dir)

cmd = **'build/X86\_MESI\_Two\_Level/gem5.opt -r -e -d '** + dir + **' configs/example/se.py --num-cpus='** + str(num\_threads)\

+ **' --cpu-type=timing --mem-type=DDR3\_1600\_x64 --mem-channels=10'** \

+ **' --hybrid-memory-system --near-mem-size=1GB --far-mem-size=3GB --near-mem-channels=8 --far-mem-channels=2'** \

+ **' --caches --l2cache --num-l2caches=1 --l2\_size='** + l2\_size + **' --l2\_assoc='** + str(l2\_assoc) + **' --l2\_tags='** + l2\_tags \

+ **' --l1i\_size=32kB --l1d\_size=32kB --l1i\_assoc=4'** \

+ **' --fast-forward=200000000 --maxinsts=2000000000'** \

+ **' --bench='** + benchmark + **' --num-phase-ticks='** + str(num\_phase\_ticks)

**print** cmd

os.system(cmd)

experiments = []

**def** run\_experiments():

num\_processes = mp.cpu\_count()

**if** num\_processes > 2:

num\_processes -= 2

pool = mp.Pool(num\_processes)

pool.map(run\_experiment, experiments)

pool.close()

pool.join()

**def** run\_experiment(args):

benchmark, input\_set, l2\_size, l2\_assoc, l2\_tags, num\_threads, num\_phase\_ticks = args

run(benchmark, input\_set, l2\_size, l2\_assoc, l2\_tags, num\_threads, num\_phase\_ticks)

**def** add\_experiment(benchmarks, num\_threads):

**for** l2\_size **in** [**'256kB'**,**'1MB'**,**'4MB'**]: *//所设定的三种内存*

**for** l2\_tag **in** [**'LRU'**, **'HMM'**]:*//分别使用LRU HMM算法*

**for** num\_phase\_ticks **in** [10000000]:

experiments.append((**'-'**.join(benchmarks), **'ref'**, l2\_size, 16, l2\_tag, num\_threads, num\_phase\_ticks))

/\*\*

\*测试的CPU2006中的多个工作负载

\*/

benchmarks = [

***'400.perlbench',***

***'401.bzip2',***

***'403.gcc',***

***'410.bwaves',***

***'416.gamess',***

***'429.mcf',***

***'433.milc',***

***'434.zeusmp',***

***'435.gromacs',***

***'436.cactusADM',***

***'437.leslie3d',***

***'444.namd',***

***'445.gobmk',***

***'450.soplex',***

***'453.povray',***

***'454.calculix',***

***'456.hmmer',***

***'458.sjeng',***

***'459.GemsFDTD',***

**'462.libquantum',**

***'464.h264ref',***

***'470.lbm',***

***'471.omnetpp',***

***'473.astar',***

***'482.sphinx3'***

]

num\_threads =1 *#4 //分别测试了线程数为1和4*

**for** benchmark **in** benchmarks:

add\_experiment([benchmark] \* num\_threads, num\_threads)

run\_experiments()

【analyze\_all\_experiments\_result\_CPU2006】

分析测试结果

/\*\*

\*测试的CPU2006中的多个工作负载

\*/

benchmarks = [

**'400.perlbench'**,

**'401.bzip2'**,

**'403.gcc'**,

**'410.bwaves'**,

**'416.gamess'**,

**'429.mcf'**,

**'433.milc'**,

**'434.zeusmp'**,

**'435.gromacs'**,

**'436.cactusADM'**,

**'437.leslie3d'**,

**'444.namd'**,

**'445.gobmk'**,

**'450.soplex'**,

**'453.povray'**,

**'454.calculix'**,

***'456.hmmer',***

***'458.sjeng',***

***'459.GemsFDTD',***

***'462.libquantum',***

***'464.h264ref',***

***'470.lbm',***

***'471.omnetpp',***

***'473.astar',***

***'482.sphinx3'***

]

num\_threads = 1 *#4 //分别测试了线程数为1和4*

/\*\*

\*分析结果

\*/

**def** analyze\_general\_results():

results = []

**for** benchmark **in** benchmarks:

**for** l2\_size **in** [**'256kB'**, **'1MB'**,**'4MB'**]:*//所设定的三种内存*

**for** l2\_tag **in** [**'LRU'**, **'HMM'**]:*//分别使用LRU HMM算法*

**for** num\_phase\_ticks **in** [10000000]:

result\_dir = **'../gem5\_results/x86\_SE/'** + **'-'**.join([benchmark] \* num\_threads) + **'/ref/'** \

+ l2\_size + **'/16way/'** + l2\_tag + **'/'** + str(num\_threads) + **'c/'**+str(num\_phase\_ticks)

**print**(**'Parsing result from '** + result\_dir + **'\n'**)

results.append(

parse\_result(result\_dir,

benchmark=benchmark,

num\_threads=num\_threads,

l2\_size=l2\_size,

l2\_tag=l2\_tag,

num\_phase\_ticks=num\_phase\_ticks)

)

**def** num\_cycles(r):

**return** int(r.stats[0][**'system.{}.numCycles'**.format(**'switch\_cpus' if** num\_threads == 1 **else 'switch\_cpus0'**)])

**def** committed\_insts(r):

**if** num\_threads == 1:

result = int(r.stats[0][**'system.switch\_cpus.committedInsts'**])

**else**:

result = 0

**for** i **in** range(0, num\_threads):

result += int(r.stats[0][**'system.switch\_cpus{}.committedInsts'**.format(i)])

**return** result

**def** ipc(r):

**return** float(committed\_insts(r)) / num\_cycles(r) **if** num\_cycles(r) > 0 **else** 0.0

fields = [

(**'Benchmark'**, **lambda** r: r.props[**'benchmark'**]),

(**'# Threads'**, **lambda** r: r.props[**'num\_threads'**]),

(**'L2 Size'**, **lambda** r: r.props[**'l2\_size'**]),

(**'L2 Replacement Policy'**, **lambda** r: r.props[**'l2\_tag'**]),

(**'L2 Size+L2 Replacement Policy'**, **lambda** r: r.props[**'l2\_size'**] + **'+'** + r.props[**'l2\_tag'**]),

(**'L2 Miss Rate'**, **lambda** r: r.stats[0][**'system.l2.overall\_miss\_rate::total'**]),

(**'# Cycles'**, num\_cycles),

(**'Committed Insts'**, committed\_insts),

(**'Simulation Time'**, **lambda** r: r.stats[0][**'host\_seconds'**]),

(**'IPC'**, ipc),

(**'# Near Memory Pages'**, **lambda** r: r.stats[0][**'near\_memory\_pages'**]),

(**'# Far Memory Pages'**, **lambda** r: r.stats[0][**'far\_memory\_pages'**]),

(**'Near:Far Memory Page Ratio'**, **lambda** r: float(r.stats[0][**'near\_memory\_pages'**]) / float(r.stats[0][**'far\_memory\_pages'**])),

(**'system.perPhaseNearFarMemoryBwRatioHist.mean'**, **lambda** r: float(r.stats[0][

**'system.perPhaseNearFarMemoryBwRatioHist::mean'**] **or** 0.0)),

(**'system.perPhaseNearFarMemoryBwRatioHist.stdev'**, **lambda** r: float(r.stats[0][

**'system.perPhaseNearFarMemoryBwRatioHist::stdev'**] **or** 0.0)),

]

to\_csv(**'../gem5\_results/general.csv'**, results, fields)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/num\_cycles.pdf'**, **'Benchmark'**, **'# Cycles'**,

**'L2 Size+L2 Replacement Policy'**, **'# Cycles'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/committed\_insts.pdf'**, **'Benchmark'**, **'Committed Insts'**,

**'L2 Size+L2 Replacement Policy'**, **'Committed Insts'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/ipc.pdf'**, **'Benchmark'**, **'IPC'**,

**'L2 Size+L2 Replacement Policy'**, **'IPC'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/l2\_miss\_rate.pdf'**, **'Benchmark'**, **'L2 Miss Rate'**,

**'L2 Size+L2 Replacement Policy'**, **'L2 Miss Rate'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/simulation\_time.pdf'**, **'Benchmark'**, **'Simulation Time'**,

**'L2 Size+L2 Replacement Policy'**, **'Simulation Time (seconds)'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/near\_memory\_pages.pdf'**, **'Benchmark'**, **'# Near Memory Pages'**,

**'L2 Size+L2 Replacement Policy'**, **'# Near Memory Pages'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/far\_memory\_pages.pdf'**, **'Benchmark'**, **'# Far Memory Pages'**,

**'L2 Size+L2 Replacement Policy'**, **'# Far Memory Pages'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/near\_far\_memory\_page\_ratio.pdf'**, **'Benchmark'**, **'Near:Far Memory Page Ratio'**,

**'L2 Size+L2 Replacement Policy'**, **'Near:Far Memory Page Ratio'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/system.perPhaseNearFarMemoryBwRatioHist.mean.pdf'**, **'Benchmark'**,

**'system.perPhaseNearFarMemoryBwRatioHist.mean'**,

**'L2 Size+L2 Replacement Policy'**, **'Avg. Per Phase Near:Far Memory Bandwidth Ratio'**)

generate\_plot(**'../gem5\_results/general.csv'**,

**'../gem5\_results/system.perPhaseNearFarMemoryBwRatioHist.stdev.pdf'**, **'Benchmark'**,

**'system.perPhaseNearFarMemoryBwRatioHist.stdev'**,

**'L2 Size+L2 Replacement Policy'**, **'Stdev. Per Phase Near:Far Memory Bandwidth Ratio'**)

**return** results

**def** analyze\_mem\_ctrls\_results():

results = []

**for** benchmark **in** benchmarks:

**for** l2\_size **in** [**'256kB'**,**'1MB'**,**'4MB'**]:

**for** l2\_tag **in** [**'LRU'**, **'HMM'**]:

num\_phase\_ticks = 10000000

**for** mem\_index **in** range(0, 10):

result\_dir = **'../gem5\_results/x86\_SE/'** + **'-'**.join([benchmark] \* num\_threads) + **'/ref/'** \

+ l2\_size + **'/16way/'** + l2\_tag + **'/'** + str(num\_threads) + **'c/'**+str(num\_phase\_ticks)

results.append(

parse\_result(result\_dir,

benchmark=benchmark,

num\_threads=num\_threads,

l2\_size=l2\_size,

l2\_tag=l2\_tag,

mem\_index=mem\_index,

num\_phase\_ticks=num\_phase\_ticks)

)

/\*\*

\*描绘直方图

\*/

fields = [

(**'Benchmark'**, **lambda** r: r.props[**'benchmark'**]),

(**'# Threads'**, **lambda** r: r.props[**'num\_threads'**]),

(**'L2 Size'**, **lambda** r: r.props[**'l2\_size'**]),

(**'L2 Replacement Policy'**, **lambda** r: r.props[**'l2\_tag'**]),

(**'L2 Miss Rate'**, **lambda** r: r.stats[0][**'system.l2.overall\_miss\_rate::total'**]),

(**'# Cycles'**, **lambda** r: r.stats[0][**'system.{}.numCycles'**.format(**'switch\_cpus' if** num\_threads == 0 **else 'switch\_cpus0'**)]),

(**'mem\_index'**, **lambda** r: **'Mem #{}'**.format(r.props[**'mem\_index'**])),

(**'mem.PerPhaseBw.mean'**, **lambda** r: float(r.stats[0][ **'system.mem\_ctrls{0}.perPhaseBwTotalHist::mean'**.format(

r.props[**'mem\_index'**])] **or** 0.0)),

(**'mem.PerPhaseBw.stdev'**, **lambda** r: float(r.stats[0][

**'system.mem\_ctrls{0}.perPhaseBwTotalHist::stdev'**.format(

r.props[**'mem\_index'**])] **or** 0.0)),

]

to\_csv(**'../gem5\_results/hmm.csv'**, results, fields)

generate\_plot(**'../gem5\_results/hmm.csv'**,

**'../gem5\_results/mem.PerPhaseBw.mean.pdf'**, **'Benchmark'**, **'mem.PerPhaseBw.mean'**,

**'mem\_index'**, **'Avg. Per Phase Bandwidth (# GB/s)'**)

generate\_plot(**'../gem5\_results/hmm.csv'**,

**'../gem5\_results/mem.PerPhaseBw.stdev.pdf'**, **'Benchmark'**, **'mem.PerPhaseBw.stdev'**,

**'mem\_index'**, **'Stdev. Per Phase Bandwidth (# GB/s)'**)

**return** results

analyze\_general\_results()

analyze\_mem\_ctrls\_results()