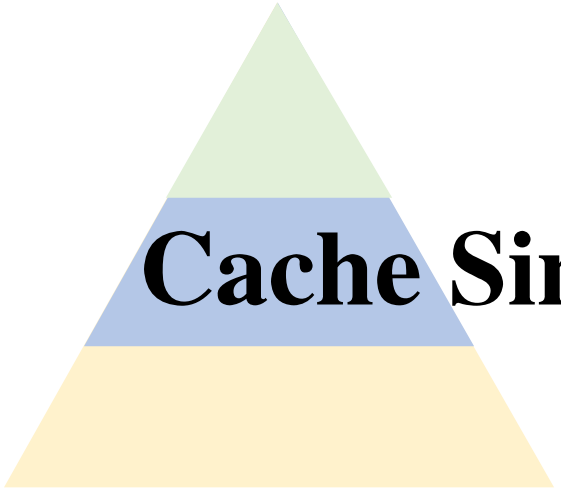


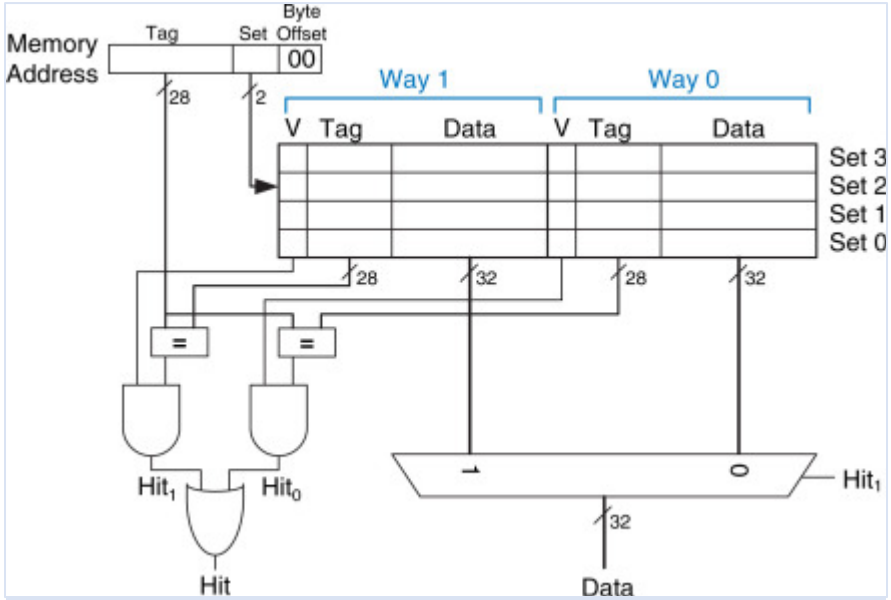
Cache Sim简单实现思路

分享人: 诸晨冬

分享时间: 2024年7月28日

一、简单可配置的CacheSim

- 需要哪些配置？
- ✓ 组的数量（Set Num）
- ✓ 路的数量（Way Num）
- ✓ Cacheline的大小（Cacheline Size）
- ✓ 替换策略（Replacement Policy）



2路组相联缓存（2-Way Set-Associative Cache）*

* 图片来源：https://www.sciencedirect.com/topics/computer-science/set-associative-cache

一、简单可配置的CacheSim

怎么实现配置?

```
winterdotc@ysyx:~$ ./CacheSim -s4 -w2 -c2 -f"bintrace.txt"
winterdotc@ysyx:~$ ./CacheSim -E -T10
```

```
  while((opt = getopt(argc, argv, "f:s::w::c::r::E::T::j::h")) != -1) {
    switch(opt) {
      case 'f': {
        sprintf(binITraceFileName,"%s", optarg);
        break;
      }
      case 's': {
        SetNumLog2 = atoi(optarg);
        break;
      }
  ......
}
```

命令行传参

```
public class Cache {
  public void initial(){
    ...          // 通用的Cache初始化
  }
  public void accessAddr(){
    ...          // 通用的地址访问
  }
}

public class LRUCache extends Cache {
  @override
  public void initial(){
    ...          // 覆写初始化函数，实现多态
  }
}
```

多态

```
  // 函数指针
  void (*cache_init)(Cache *, int, int, int) = NULL;
  void (*cache_access)(Cache *, uint32_t, uint64_t *,
uint64_t *, int , int, int) = NULL;
  // 根据替换策略选择函数
  switch(design->rp){
    case RP_LRU: {
      cache_init = LRUCache_init;
      cache_access = LRUCache_access;
    }
    default: {
      cache_init = LRUCache_init;
      cache_access = LRUCache_access;
    }
  }
```

函数指针

一、简单可配置的CacheSim

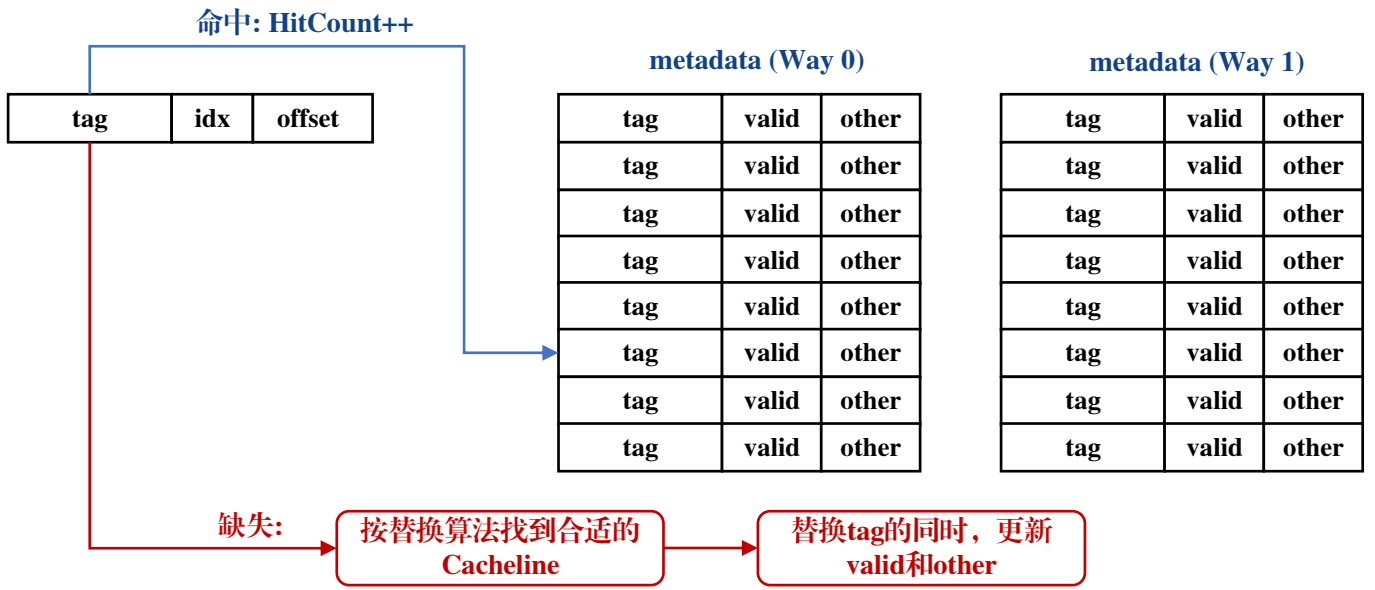
- CacheSim大致实现思路：

```
typedef struct {
  uint32_t  tag;            // 用以比较
  bool      valid;          // 用以判断是否合法
  uint8_t   *other;         // 不同种替换算法所需的域，需要自
行开辟大小
} Metadata;

typedef struct {
  Metadata *ways;       // 每个Set是多路的，数量由Way决定
} CacheSet;

typedef struct {
  CacheSet *sets;
} Cache;
```

模拟Cache，总得要有个Cache的结构体吧



还得要用代码实现替换算法

- Cache的内容不需要在意，只需要维护metadata，有了metadata就能够得出是否命中，进一步就可以求命中率。

二、设计空间探索

- **太多配置了，需要一个个命令行试出最佳的Cache配置吗?**
  - **Cache能缓存的总大小是 TotalSize=SetNum*WayNum*CachelineSize**

$$log_2(\text{TotalSize}) = log_2(\text{SetNum} * \text{WayNum} * \text{CachelineSize})$$

$$log_2(TotalSize) = log_2(SetNum) + log_2(WayNum) + log_2(CachelineSize)$$

  - **如果限定了Cache的总大小（如8KB），$log_2(8\text{KB}) = 13$，加上Cacheline至少32位的限制，大概有近百种配置的可能性，如果一一尝试的话，就可以摸鱼一个下午**
  - **现代的Cache一般更大，~~那就可以摸鱼大半年子，~~就需要利用CacheSim进行设计空间探索了**

二、设计空间探索

- 设计空间探索的简单实现：

```
// 设计报告
typedef struct {
  uint64_t RequestCnt;     // 总事务数量
  uint64_t HitCnt;         // Cache命中次数
  uint64_t MissCnt;        // 未命中次数
  uint64_t HitCost;        // 命中代价
  uint64_t MissCost;       // 未命中代价
  double Area;             // 面积
  double Freq;             // 频率
  double Score;            // 分数
} Report;

// 一个设计（解）
typedef struct {
  int SetNumLog2;          // CacheSet 大小
  int WayNumLog2;          // Way 大小
  int CacheLineSizeLog2;   // CacheLine的大小
  int rp;                  // 替换策略
  Report report;           // 设计对应的设计报告
} Design;
```

设计空间探索，总得定义个设计的结构体吧

SetNumLog2	WayNumLog2	CacheLineSizeLog2	rp
3	4	6	LRU
3	5	5	RANDOM
3	6	4	LRU
…	…	…	…

生成设计空间

循环评估每一种设计方案

评估设计（跑一遍CacheSim，算一算命中率）

- 其实就是省去了输入命令行的时间，还得花一个下午，能不能更快一点？

二、设计空间探索

- 设计空间探索的优化实现：
  - 剪枝：已知目前搜索到的最优方案是命中了900次（一共1000次），此时搜索的方案已经缺失了101次，还需要继续搜索吗？更多剪枝的可能性？
  - 多线程：反正设计空间都已经有了，多来几个线程一块跑试试。
  - 启发式算法：如果设计空间太大，很可能没法遍历所有的设计，把设计空间探索想象成一个最优化问题，其实就是在诸多解空间里面找到一个最优解（有可能是多目标优化）。
    - 模拟退火
    - 遗传算法
    - 粒子群算法
    - ……

二、设计空间探索

- 多线程×设计空间探索：
  - 将地址流文件读入内存后进行操作（如果每个线程都去读文件，文件读写的效率会很低）
  - 互斥锁（避免评估重复的设计）

```
void *worker(void *arg) {
  // 工人函数
  bool endflag = false;
  int DesignID = 0;
  while(true){
    // 取任务
    pthread_mutex_lock(&mutex);
    if(curDesignID >= DesignNum) endflag = true;
    else {
      DesignID = curDesignID;
      curDesignID++;
    }
    pthread_mutex_unlock(&mutex);
    if(endflag) break;    // 所有设计空间均已得到探索
    // 探索设计空间
    eval(&(designs[DesignID]));
  }
}
```

每个线程需要干的事情

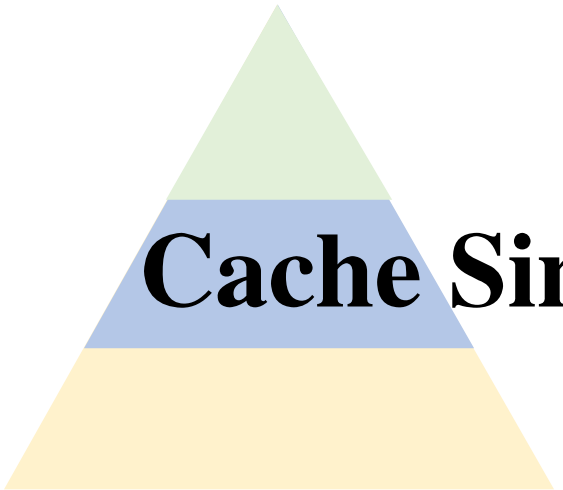
二、设计空间探索

```
winterdotc@Scientist:~/ysyx/ysyx-workbench/CacheSim$ time ./CacheSim -f"bintrace.bin" -E -T12 -j4
Sys init!

real    9m24.473s
user    36m51.163s
sys     0m0.480s
winterdotc@Scientist:~/ysyx/ysyx-workbench/CacheSim$ time ./CacheSim -f"bintrace.bin" -E -T12 -j8
Sys init!

real    9m23.426s
user    37m18.275s
sys     0m0.496s
winterdotc@Scientist:~/ysyx/ysyx-workbench/CacheSim$
[0] 0:am-kernels- 1:npc  2:top  3:bash*
```

- 更多优化的可能性：
- 不同设计的评估时间不同（LRU替换策略下，WayNum大更耗时间），因此可能需要提前评估不同设计的耗时并将设计进行分组，让各个组之间的任务相对均衡。
- 每个评估任务的耗时都较短，导致线程一部分的时间在等待互斥锁，每次取多个任务，降低访问互斥资源的频率。



Cache Sim简单实现思路

分享人: 诸晨冬

分享时间: 2024年7月28日

代码仓库地址：https://github.com/WinterDotC/CacheSim