Cache测试

## Cache说明

### 概述

实现了与主存直接相连的Cache，主存与Cache之间的数据传输是以块为单位的。主要参数和策略如下（指令集采用32位物理地址）：

* + - Tag与数据分离，每行内包含Valid和Dirty两位和一个数据块。
    - 块大小为8，相联度为4，Cache总容量为4096个字节。
    - Cache采用了随机替换的替换策略。
    - 写Cache时采用写回，写不命中时采用写分配。
    - 加入了写缓冲机制，并不直接写回主存，而是写入写缓冲，由写缓冲择机写入主存。写缓冲的大小为8个行（Cache行），读写时以行为单位。同时实现了写合并机制

### 参数选择

* + - 块大小的选择考虑了我们实现的Y86指令集中所有访存指令都是以8字节为单位。考虑到Cache与主存间只能以块为单位进行数据交互。而指令从Cache中访存也是以块为单位。故选择块大小为4.
    - 相联度可以调节，经过多次测试，最终得到最佳相联度16

### 工作流程



#### 读Cache

读Cache时，将32位地址分为|Tag|Index|Block\_offset|三部分。

进行行匹配时，在对应的组内逐行匹配，若Tag匹配，且此行Valid有效，则命中。

不命中时，需要按以下流程处理：

* + - 1. 由于加入了写缓冲机制，首先去写缓冲中查找是否有匹配的行，若有，将其取出，这时需要进行以下检查：
         1. 若组内有空闲行，将其放入空闲行
         2. 无空闲行，按照替换策略，选择一个行替换，将取出的行放入这行，若被替换的行Dirty位有效，先将其写回写缓冲

#### 写Cache

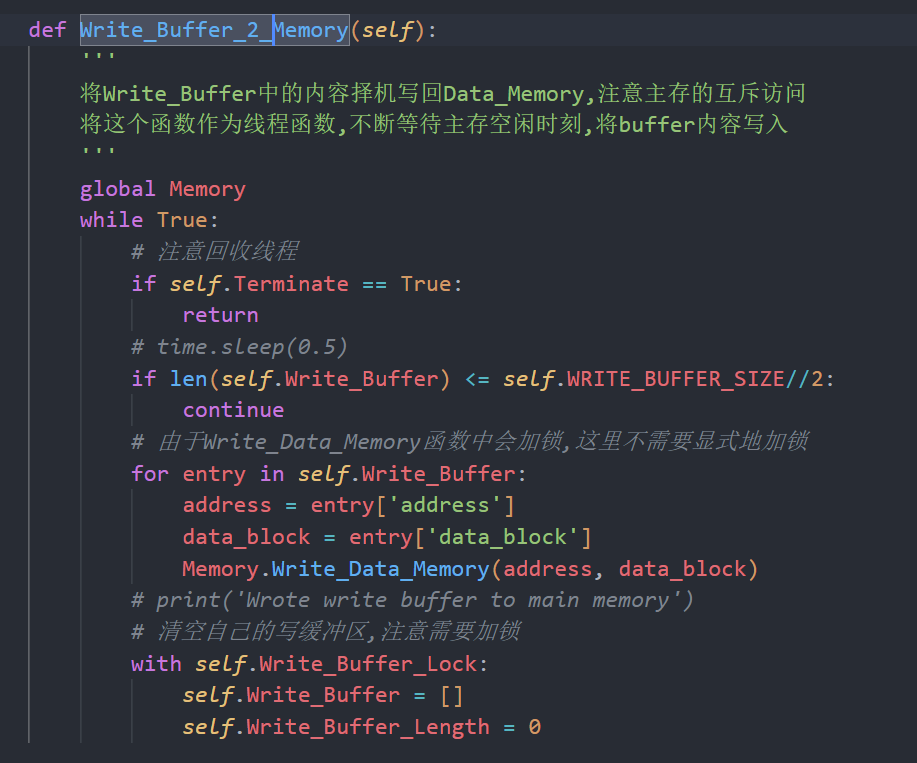
写Cache时，同样进行行匹配。

命中时，写入找到的行，若这行有效且Dirty有效，先将其写入写缓冲。

不命中时，若有空闲行，写入空闲行。无空闲行时，按照替换策略，选择一个行替换，将取出的行放入这行，若被替换的行Dirty位有效，先将其写回写缓冲。

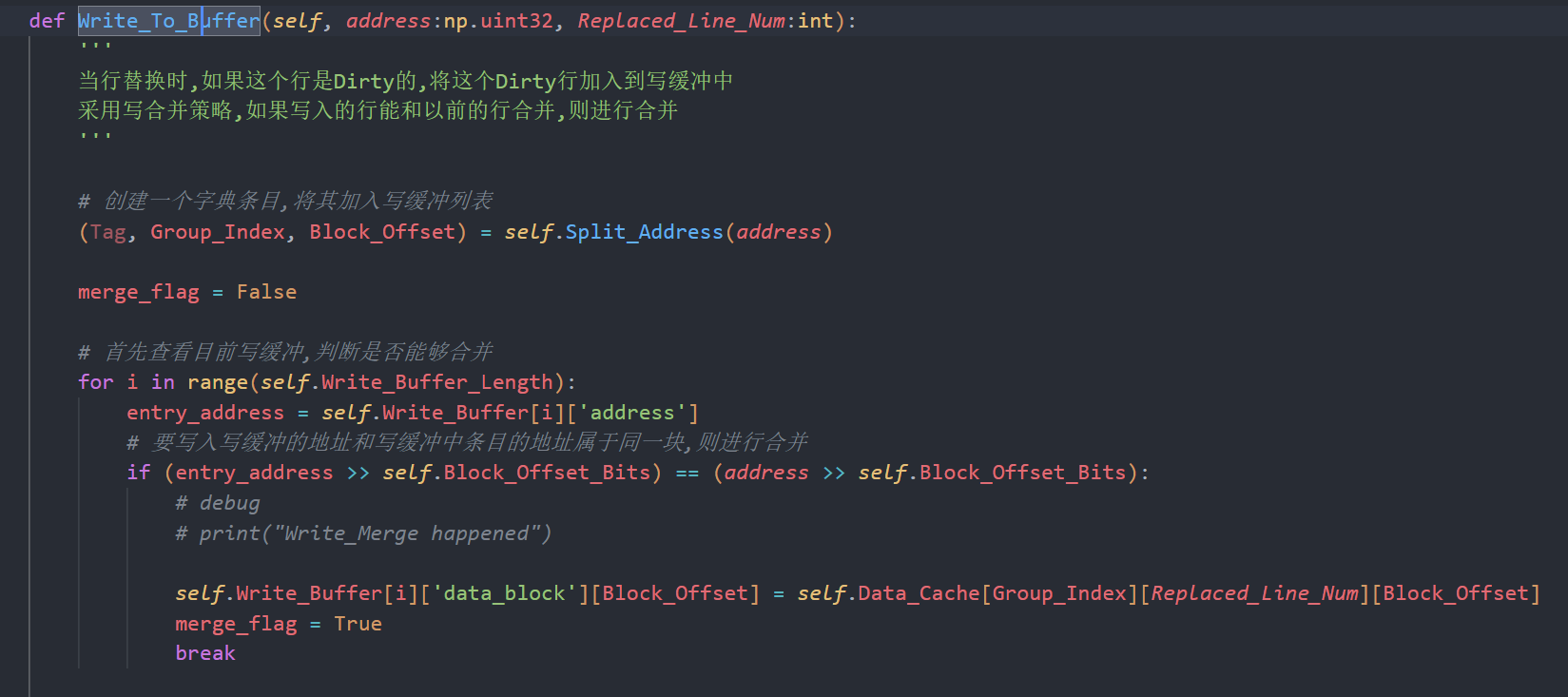
#### 写缓冲的实现

如上所述，当Dirty的行写回主存时，并不直接写回主存，而是写回写缓冲。写缓冲择机将缓冲中的数据块写入主存。在代码实现中，通过分配一个线程来实现这一点：



这个线程不断运行检查，当主存空闲时将自身数据写入。注意，为了减少写回次数，我们要求当前缓冲中数据块数量大于最大容量的一半时才写入。

#### 写缓冲中写合并的实现



如上图，当写入的多个字节位于同一个数据块中时，只占用写缓冲的一个条目。

## 测试说明

用于测试的Cache直接与主存相联，块大小为8，相联度为16，Cache总容量为4096个字节。Cache和主存之间都是以块为单位进行数据传输的。

Cache采用了随机替换的替换策略，写Cache时采用写回，写不命中时采用写分配。同时加入了写缓冲机制，并不直接写回主存，而是写入写缓冲，由写缓冲择机写入主存。

采用分阵乘对Cache进行测试，矩阵规模为64\*64，矩阵中每个元素的长度为8字节。分别对直接矩阵乘和分块矩阵乘法两种访存方式对Cache进行测试。两种方式的访存序列在结果部分给出

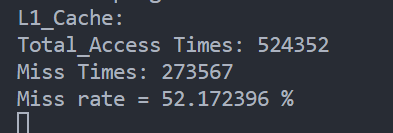
## 测试结果

### 直接矩阵乘

访存伪代码如下：

for i in range(N):  
 for j in range(N):  
 for k in range(N):  
 # 读y和z  
 r += y[i][k] \* z[k][j]  
 # 写x  
 x[i][j] = r

Python

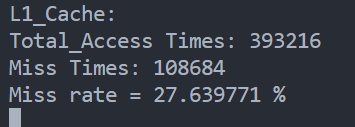
测试结果如下：

### 分块矩阵乘

访存的伪代码如下：

for jj in range(0, N, B):  
 for kk in range(0, N, B):  
 for i in range(N):  
 for j in range(min(jj+B-1, N)):  
 r=0  
 for k in range(kk, min(kk+B-1, N)):  
 # 读y读z  
 r += y[i][k] \* z[k][j]  
 # 先读x，再写x  
 x[i][j] = x[i][j] + r

Python

分块因子B取4时，结果如下：