# CacheLab-Report

**学号：21302010042**

**姓名：侯斌洋**

*实验过程：*

（1）首先进行环境搭建，在VMware上安装Ubuntu20.04系统，并在此系统上利用vscode完成本次lab。然后阅读附带的文档了解本次lab的整体要求以及一些注意事项。

（2）接下来是关于PartA以及PartB实现的一些细节，包括代码思路以及遇到的问题等。

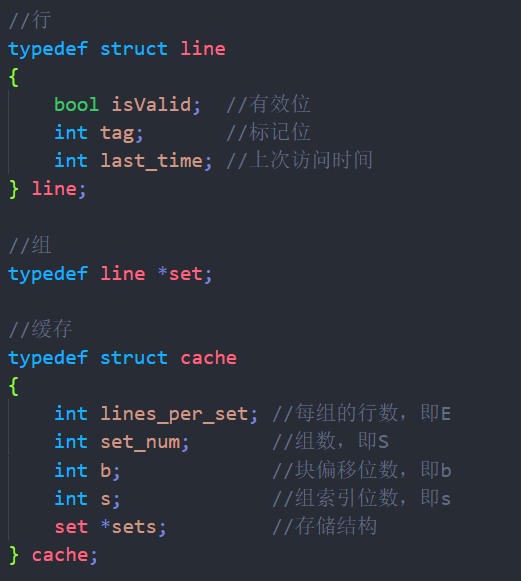
此外，Lab托管在GitHub上，并且每次进行修改后均会上传，可访问<https://github.com/HBY-STAR/ICS_lab/tree/master/cachelab-handout> 查看整个实验进行的过程细节以及修改记录。

①PartA：

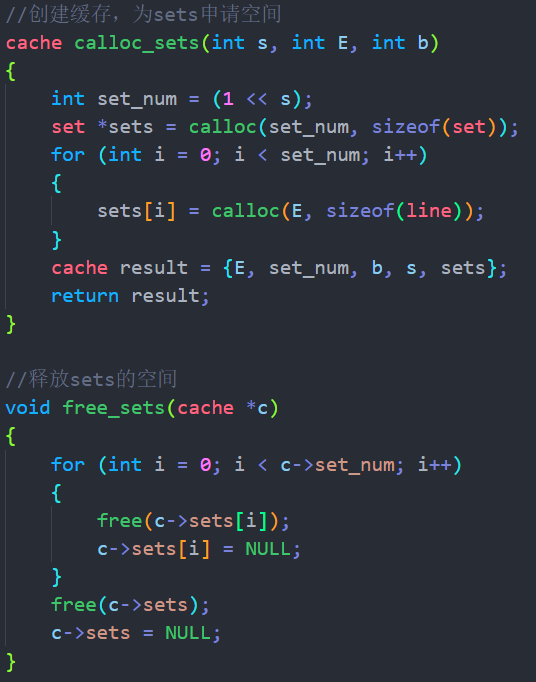
本实验部分要求实现一个高速缓存模拟器，要求能够识别并处理数据操作并计算每次处理过程中的hit、miss、以及eviction数。

由于该模拟器在运行时要处理命令行参数，根据PDF提示包含<getopt.h>头文件，于是去查询getopt函数的用法并通过while加switch语句实现对命令行参数的解析，详见代码中的void get\_argvs(int argc, char \*\*argvs, int \*s, int \*E, int \*b, FILE \*\*fp, bool \*isDisplay)函数。

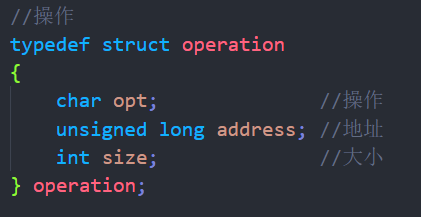
然后是实现高速缓存结构，如下分别定义了行、组和缓存结构。



由于高速缓存的大小是由命令行参数给出的，故这里在定义结构的时候要用指针，然后在运行时再申请空间，申请空间与释放空间的函数如下



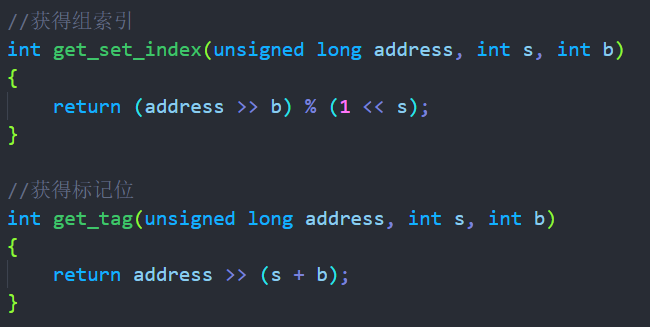
然后为了处理操作也定义了操作结构如下：



之后实现对于文件中操作的获取函数如下：



获取了操作之后还要对操作中的address进行解析，找出组索引和标记位。

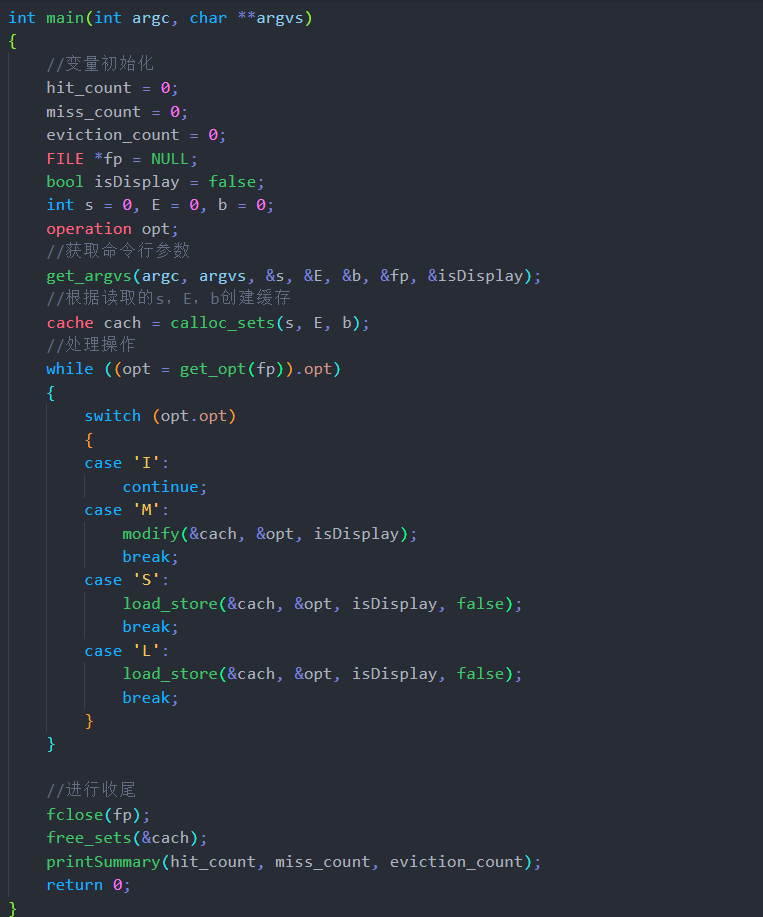


由此对于文件的操作读取部分以完成，接下来便是对于操作的处理。

由于load和store在高速缓存上的操作是相同的，故实现为一个函数void load\_store();而modify的操作是load与store的组合，实现为一个函数void modify()。

由于代码很长，故不在此列出，load和store的思路如下：首先根据操作的组索引找到对应的组，在对应的组内遍历所有行寻找该组内是否已有标记位和有效位均吻合的行，若有，则hit，更新时间后结束操作。若无则miss，在该组内寻找空余的行，若有空余的行，则该行为新的该操作对应的行，之后更新该行的标记以及有效位之后再更新时间，然后结束操作。若无空余的行则要eviction，遍历该组寻找上次访问时间最晚的行，执行驱逐更新该行的标记以及有效位之后再更新时间，然后结束操作。modify的思路如下：直接执行一次load然后执行一次store，在此过程中注意一下opt即可。此外函数中的变量bool isDisplay;为是否有命令行参数-v并每次操作都展示hit、miss和eviction的标记。

最后便是组织上面的操作并实现主函数，如下：



②PartB：

本次Lab的第二部分相对于第一部分代码量会小很多，但需要深入思考的问题却更难了，尤其是对于64×64矩阵的优化方案。

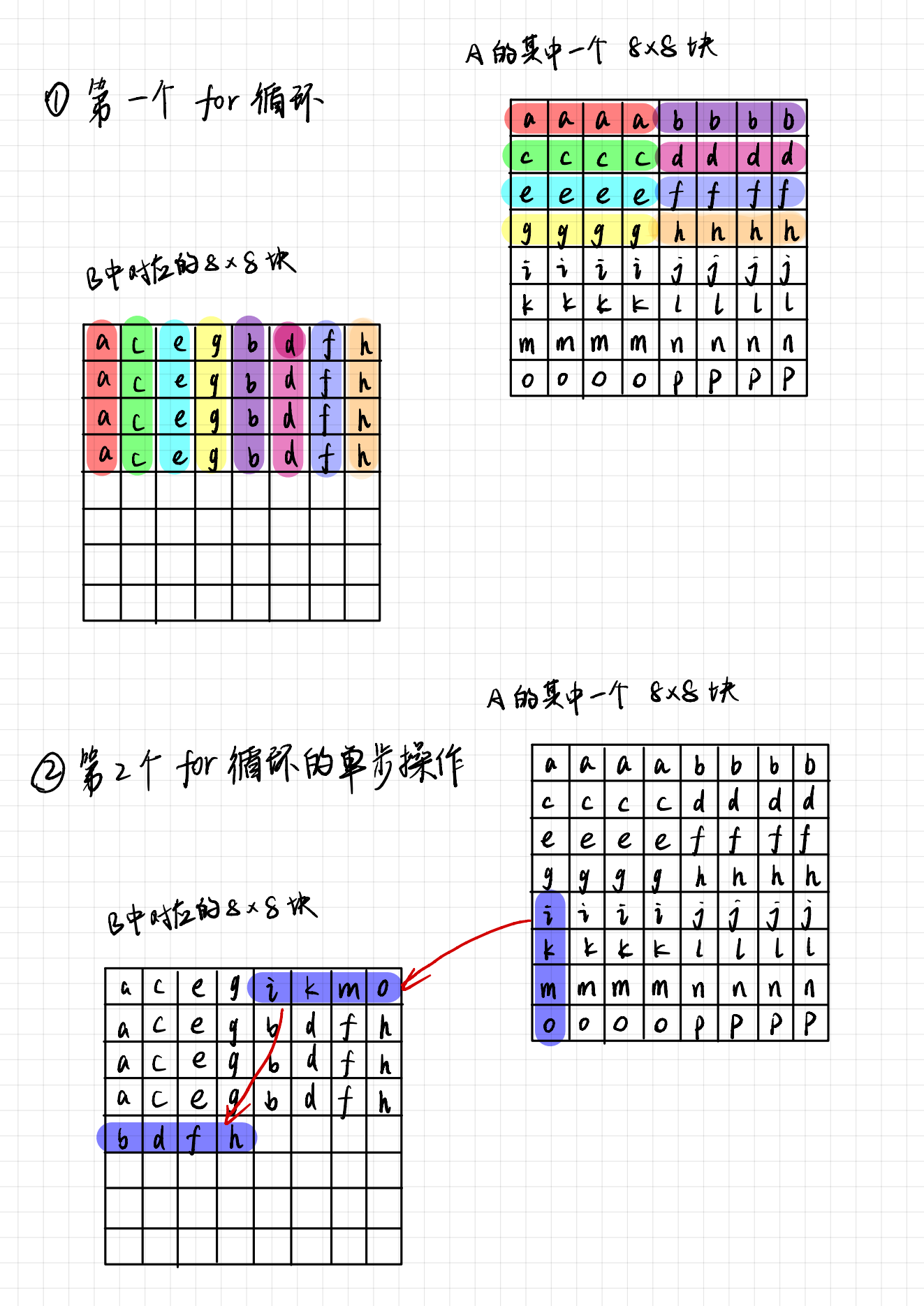
（i）32×32

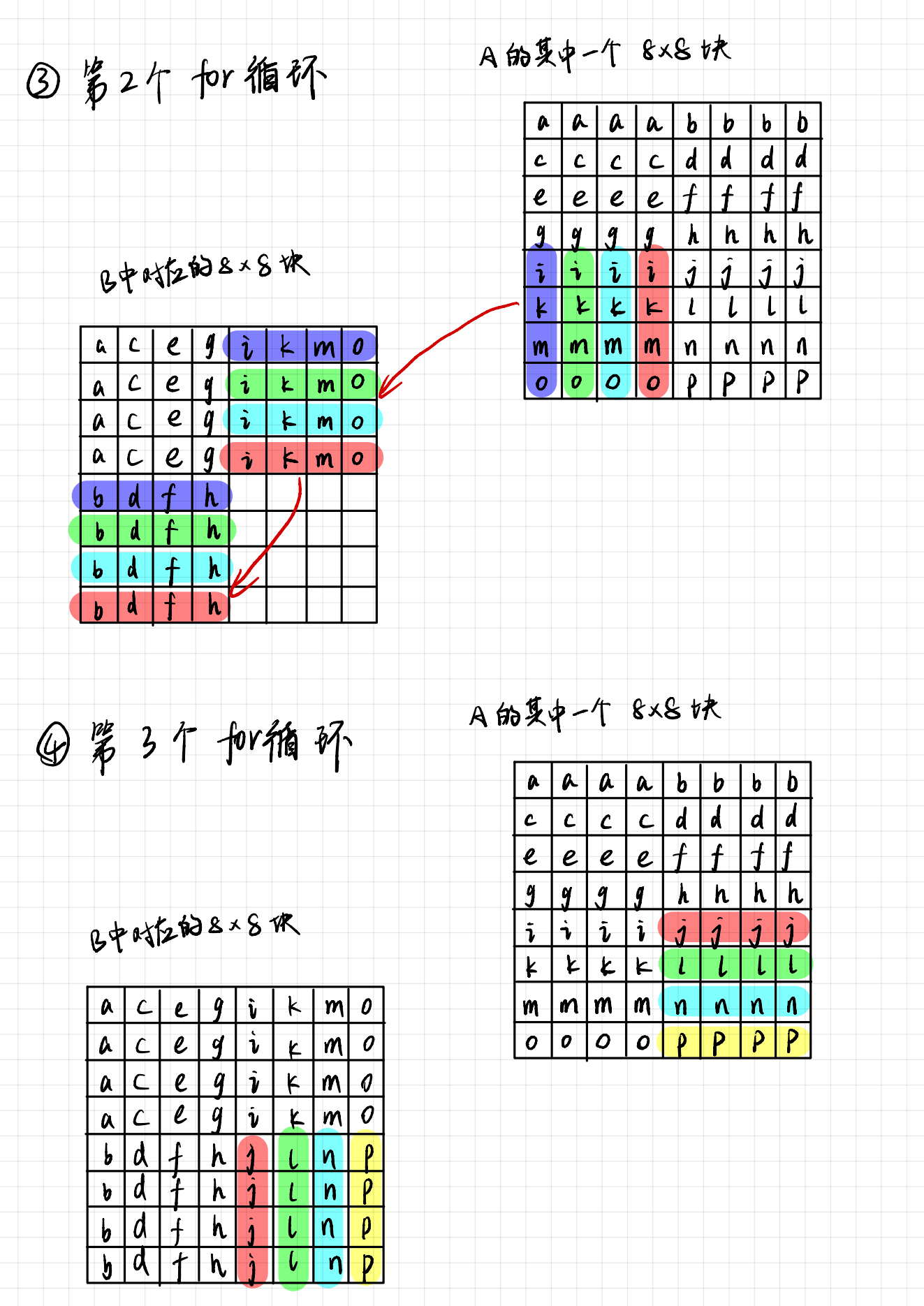
根据PDF的提示进行分块处理，这里观察到cache的大小正好够填满数组的前8行，因此cache的映射应该是整个cache分别对应0-7、8-15、16-23、24-31这四个部分，举例来说就是第0行的前8个元素块与第8行的前8个元素块对应cache中的一个相同的块。由以上分析分成8×8的块，这样对于对角线之外的块A和B就不会冲突，从而减少了miss数。这里为了处理对角线上的8×8块A和B数组同时占用cache导致大量不命中的情况，故一次读取A中的一行8个数据并存入临时变量中，这样A每块只有一次冷不命中，而B对角线上的块部分有两次不命中，其他位置的块也是只有一次冷不命中，已达到最优要求。

具体实现请看代码。

（ii）64×64

由以上32×32的分析，这次cache的大小正好够填满数组的前4行，故首先尝试分成4×4的块，此时的miss数已经达到了1600左右，虽然有了很大的减少但并未达到最优要求。事实上由于cache的块大小为8个int值，故分成4×4的块并未充分利用cache的空间，想要充分利用cache的空间还是要分成8×8的块，但块内操作需要思考新的方案，否则还是会有大量不命中。下面是对于8×8的块中进行的操作的图示，分别对应于代码中的三个for循环的操作：

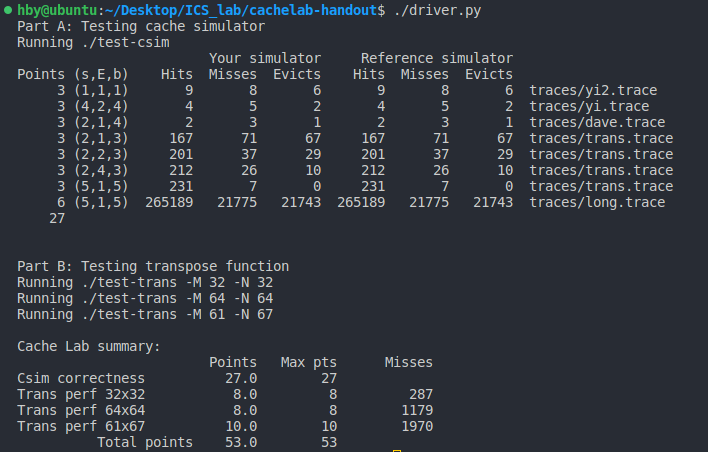




（iii）61×67

根据PDF的提示进行分块处理，由于给出的最优要求的miss数较多，达到了2000，且该缓存大小并不规则，故先对不同分块进行测试，当分块为17×17时已达到要求，并不需要额外优化了，故就此结束。具体实现请看代码。

*本地运行结果：*（如测试结果不同请联系我，谢谢）

**

*总结：*

在本次lab中主要加深了对于高速缓存的理解，在PartA中弄清了每次数据处理时高速缓存是如何工作的，hit、miss以及eviction时高速缓存内部的状态，此外还实现了一个高速缓存模拟器。在PartB中加深了对高速缓存与主存关系的理解，并注意到在处理大矩阵时分块策略在高速缓存层面上对于函数性能的提高，此外不断优化函数的过程也是一次充满挑战的经历。