Lab04 Data Locality and Cache Blocking and Memory Mountain

徐薪-519021910726

Exercise 1: Cache Visualization

场景 1：（使用 cache.s）

1.

Cache 命中率为 0%。

2.

因为$\text{step size} = 8\text{words} = 32\text{bytes}, \text{cache block size} = 2\text{ words} = 8\text{bytes}$. 数据是以 block 为单位读入 cache 里面的，在这种情况下，下一次写入的数据地址超过了上一次数据写入的 block 范围，所以每一次都是 misshit。

3.

不可以。因为每一次 step size 的大小刚好是 cache 的 size 大小。又因为第一次访问一定 miss（cache 里面为空），并且以后的每次访问会访问到相同的 cache block，而该 block 又被上次的访问数据所占用，所以每次访问都是 occupied，因此不论 Rep Count 如何增大，只要 step size 不变，这种持续占用的情况会一直持续下去，从而 cache hit rate = 0%。

4.

我们可以减小 step size 的大小，也可以增大 step size 的大小。比如当我们令 step size 为 1 时，cache hit rate= 50%；当 step size = 32 时，cache hit rate = 75%。当 step size = 32 时，如果无限增大 Rep Count 的大小， cache hit rate 可以无限接近 100%。

场景 2（使用 cache.s）

1.

cache hit rate=75%。

2.

cache size = array size，所以一共只有一个区，所以不会出现不同区之间相同组共用 cache 的情况。而 step size=8bytes，所以每一次因为 cache empty 造成 miss 所读入的数据块都会使下一次的 read 和 write hits。所以一次遍历（Rep Count = 1）里的 miss 总数 = 总块数 = 16。而一共进行了读写命令$256 \div 8 \times 2 = 64$次，所以 cache hit rate = $(64 - 16) \div 64 \times 100\% = 75\%$。

3.

当重复无数次时，命中率为 100%。因为在一次遍历后，cache 已经将所需的数据全部读入完毕，所以以后的每次遍历所需的地址都已经存在 cache 里面了，所以第一次之后的 cache miss rate = 100%。当 Rep Count 不断增大时，只存在第一次遍历时的 16 miss，其余访问全是 hit，所以无限增大 Rep Count 的值，命中率会无限接近于 100%。

Exercise 2: Loop Ordering and Matrix Multiplication

1.

ikj 嵌套顺序性能最好，kji 嵌套顺序性能最差。

2.

和我观察到的结果一致。因为最内层循环的部分是程序执行次数最多的部分，如果最内层的数据访问步长超过了每次读入的数据块的长度，那么每一次运算都会造成一次 cache miss，

CPU 需要重新从内存中读取数据，从而降低性能。

3.

性能好的变得更好，性能差的变得更差。说明对于好的性能算法来说，增加变量定义的赋值时间小于减少的重复计算时间，从而性能更好；但是对于性能较差的算法来说，增加变量定义的赋值时间大于减少的重复计算时间，从而性能更差。(但是我认为性能应该都变好才对，可是跑出来的结果并不是这样)。

4.

因为分析中只考虑了 cache 的命中率，并没有考虑 memory latency 等因素。数据的读取也会有延迟，并且在 memory 中读取数据所需的延迟时间远大于在 cache 中读取数据，所以在实际情况中，运算性能差距更大。

Exercise 3: Cache Blocking and Matrix Transposition

1.

当矩阵大小比较小时，矩阵分块实现转置比不用矩阵分块的方法要慢。只有当矩阵大小达到一定程度，矩阵分块算法才会优于不用矩阵分块的算法。因为当矩阵比较小时，对矩阵进行分块操作计算和增加循环层数所需的代价远远大于分块计算减少 cache 命中失败的代价。

2.

分块算法的性能先变好后变坏。因为当 block size 太小或太大时，分块算法都会退化为不分块的算法。

Exercise 4: Memory Mountain

1.

```
Clock frequency is approx. 2208.0 MHz
Memory mountain (MB/sec)
       s1      s2      s3      s4      s5      s6      s7      s8      s9      s10     s11     s12     s13     s14     s15
128m    11686   6104    3921    2993    2257    1933    1731    1609    1450    1470    1273    1220    1119    678     1111
64m     11539   6117    4178    3118    2563    2191    1063    1521    1291    1398    1320    1254    783     721     721
32m     10835   6038    4341    3330    2484    2058    1372    1262    1171    1395    825     1411    906     1274    1241
16m     12238   6664    5221    4167    3361    2933    2360    2070    1642    1349    1844    1197    1756    1477    1630
8m      14901   8920    7166    5452    4390    3415    3480    2712    2777    2988    3019    3106    3292    3249    3480
4m      22993   17564   14687   12243   10239   8913    7789    6917    6627    6399    6088    5894    5736    5551    5544
2m      24856   18491   15381   12365   10339   8934    7820    6929    6612    6382    6162    5939    5781    5663    5561
1024k   26532   19786   16441   13243   11043   9554    8419    7439    7128    6858    6601    6296    6134    6018    5890
512k    26953   20743   17602   14208   11968   10331   9063    8063    7741    7529    7307    7190    6981    6997    6933
256k    30137   23502   21216   18896   16566   14773   13112   11692   11403   11208   11115   11473   11428   10914   11539
128k    31685   25236   24129   22795   20701   18975   16684   14826   14603   14781   14455   15866   15123   15023   14083
64k     32694   26291   25988   24949   23005   21191   19176   17980   18310   16708   16400   16207   14455   14979   27251
32k     40647   39236   38279   37683   35993   35464   34224   33496   33211   33801   37790   36312   31978   35388   36532
16k     40195   37218   36762   35054   33801   32760   30754   31845   35251   34776   32877   25982   24840   30754   32583
```

2.

我的判断：根据相同步长，不同工作集下的断层情况：

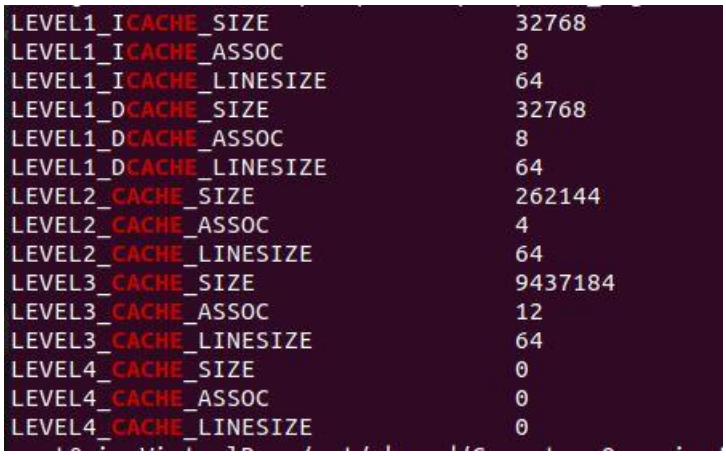
一级缓存：32k

二级缓存：256k

三级缓存：8m

3.

系统中高速缓存配置如下：



发现 L1 的指令 cache 和数据 cache 都是 32k，所以 L1 的 cache 总大小应该为 64k，我的判断是 32k，不够准确。L2 的 cache 大小判断准确。L3 的 cache 大小不够精确，可能是因为程序中工作集的间隔划分太大。

4.

我们将 L3 的大小设为 size，则 size=8M，观察到在 s8 时程序的吞吐量到达了最低点，所以 block size=8words=64bytes.