计算机系统结构(A)

实验 4

李子龙 518070910095

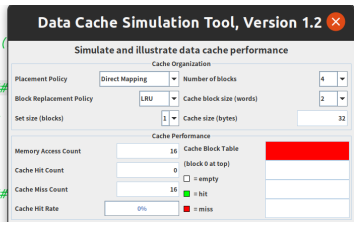
2021 年 5 月 12 日

一. Cache 可视化工具

(1) 场景一

- Cache 命中率为 0% 。

```
15
16          .data
17  array:  .space  2048           # max array size specified in BYTES (
18
19          .text
20  ###############################################################
21  # You MAY change the code below this section
22  main:   li      $a0, 128                # array size in BYTES (power
23          li      $a1, 8          # step size  (power of 2 > 0)
24          li      $a2, 4  # rep count  (int > 0)
25          li      $a3, 0  # 0 - option 0, 1 - option 1
26  # You MAY change the code above this section
27  ###############################################################
28
```



- `stepsize` 被设定为 8，按照 `int`（4 Bytes）存储，写入（`option`为0）需要跳跃 32 bytes，只有一组，而一组正好是

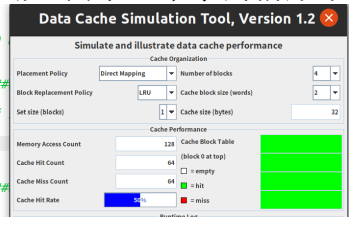
$$4 \text{ blocks} \times 2 \text{ words} \times 4 \text{ bytes} = 32 \text{ bytes}$$

  将会导致每一次的写入都会失效。

- 增加 `repcount` 也无法提高命中率，因为上文所述的间隔无法被改变，会一直失效。

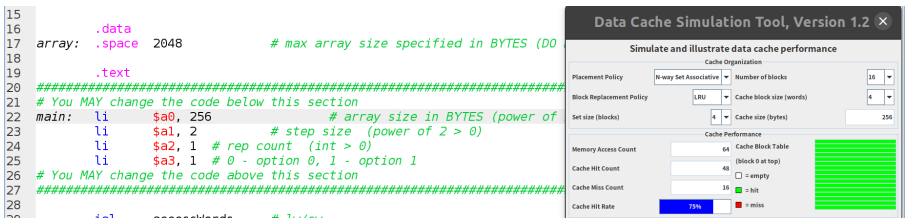
- 将 `stepsize` 更改为 1，可以将命中率提高至 50% 。
  每对访问第一个失效，第二个命中，在块大小为 2 个字的情形下。

```
15
16          .data
17  array:  .space  2048           # max array size specified in BYTES (DO
18
19          .text
20  ###############################################################
21  # You MAY change the code below this section
22  main:   li      $a0, 128                # array size in BYTES (power of
23          li      $a1, 1          # step size  (power of 2 > 0)
24          li      $a2, 4  # rep count  (int > 0)
25          li      $a3, 0  # 0 - option 0, 1 - option 1
26  # You MAY change the code above this section
27  ###############################################################
28
```

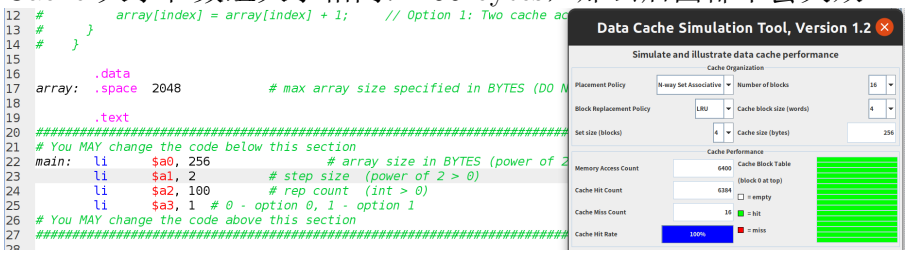


(2) 场景二

- 命中率为 75 % 。

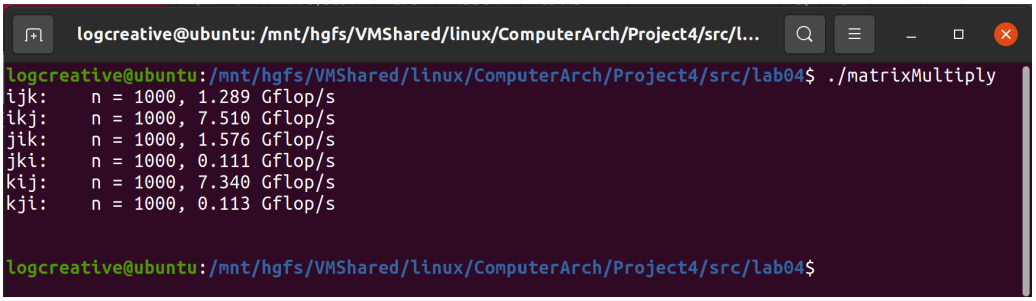


- `stepsize` 是 2，一块 4 个字，那么相邻的两次读+写，除了第一个读失效，其余均为命中，命中率为 75% 。
- 命中率会接近于 100% 。因为以第一重复后，所有的数据都进入了 Cache，Cache 大小和数组大小相同：256 bytes，那么后面都不会失效。



二. 矩阵乘法

- ikj 性能最好，jki 性能最差。

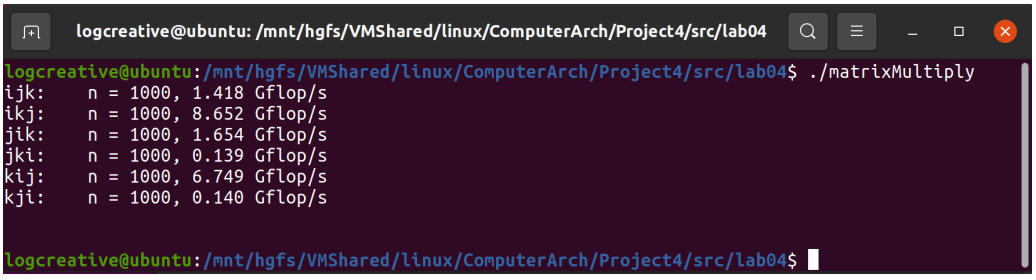


- 一致。

	未命中总次数	速率1	速率2
AB	1.25	ijk 1.289	jik 1.576
AC	2.00	jki 0.111	kji 0.113
BC	0.50	kij 7.340	ikj 7.510

不同的算法步长会导致不同的Cache 命中率，从而影响访问时间，也就影响了性能。

- 性能得到了改善，除去 kij。



使用局部变量进行局部加和可以显著减少 Cache 失效率。在有Cache 失效率的情形下，使用局部变量提前预取可以提升性能。

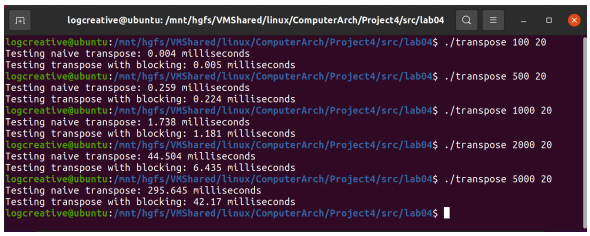
- 硬件预取可以大幅降低最低两种算法的失效率，接近于0，而对于另外一些算法效果不明显。

3. 矩阵转置

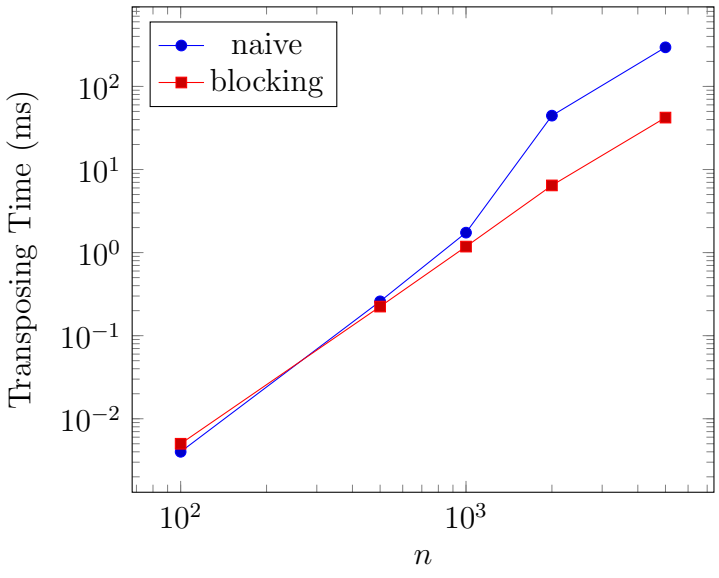
`transpose_blocking()` 函数的实现如下：

```
/* Implement cache blocking below. You should NOT assume that n is a
* multiple of the block size. */
void transpose_blocking(int n, int blocksize, int *dst, int *src) {
    // YOUR CODE HERE
    for(int i = 0; i < n; i += blocksize)
        for(int j = 0; j < n ; j += blocksize)
            // (j,i) is the starting element
            // of the block
            for(int x = 0; x < blocksize && i + x < n; ++x)
                for(int y = 0; y < blocksize && j + y < n; ++y)
                    // (j+y,i+x) is the transposing element
                    // of the matrix.
                    // (i+x,j+y) is the final position.
                    dst[j+y + (i+x)*n] = src[i+x + (j+y)*n];
}
```

(Part 1) 测评结果如下：



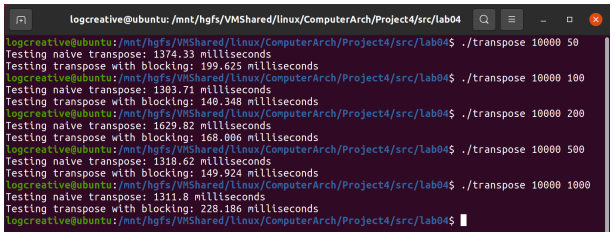
Part 1: blocksize=20

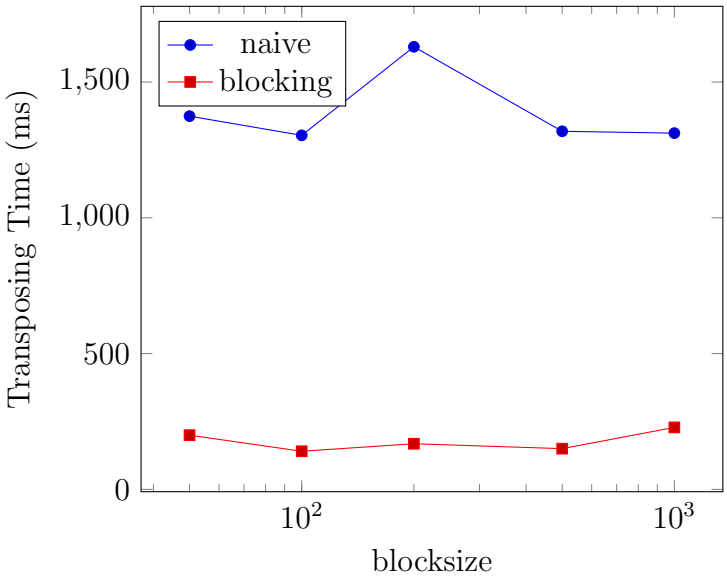


只有当 n 达到 1000 以上时，分块才会比普通方法块，而且规模越大越明显。小型矩阵会因为分块与总规模接近而近似不分块，普通算法和分块算法都不一定能占满 Cache，而且多层循环会增加开销。

(Part 2) 测评结果如下：

Part 2: $n = 10000$





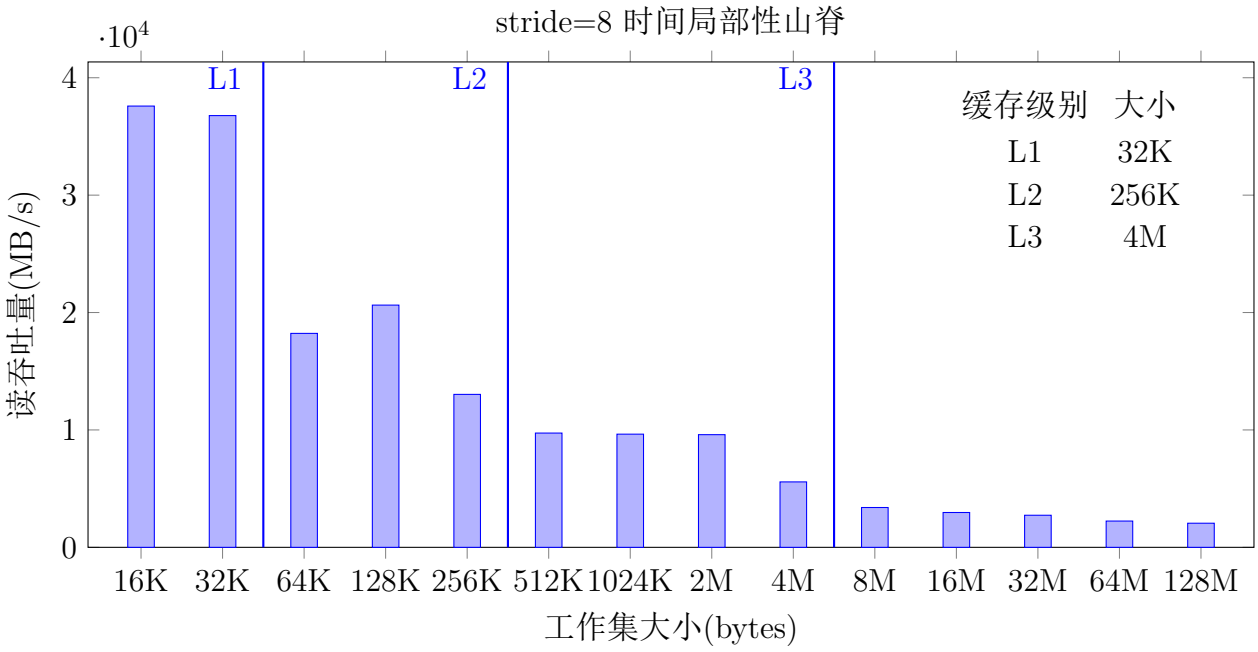
blocksize 增加时性能没有非常明显的变化，会有颠簸现象。因为 Cache 比较大的原因，在一定分块范围内命中率都会比较高，blocksize 并不占据主导因素。比如当 blocksize 为 5000 时，时间立刻上去并逼近普通算法。

```
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04$ ./transpose 10000 5000
Testing naive transpose: 1342.76 milliseconds
Testing transpose with blocking: 1226.07 milliseconds
```

4. 内存山脊

- 运行结果如下：

```
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04$ cd mountain
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain$ make
gcc -Wall -O3 -D__i386__ -o mountain mountain.c fcyc2.c clock.c
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain$ ./mountain
Clock frequency is approx. 2496.0 MHz
Memory mountain (MB/sec)
        s1      s2      s3      s4      s5      s6      s7      s8      s9      s10     s11     s12 s13     s14     s15
128m    12546   7750    5310    4003    3137    2518    1889    2032    1891    1727    1672    1529    1504    1426    1397
64m     13824   7734    5492    3899    3321    2846    2449    2235    2166    1913    1901    1783    1720    1652    1568
32m     16245   9031    6486    5058    3899    3310    2906    2515    2447    2272    2173    2059    1977    1931    1947
16m     16601   9868    7145    5404    4297    3617    3155    2836    2661    2532    2383    2263    2201    2165    2080
8m      18023   11005   7975    6105    4930    4227    3663    3282    3058    3076    2929    2885    2813    2761    2879
4m      26491   18157   14617   11719   7492    6596    5660    5079    5534    5860    5982    6001    6189    6123    6115
2m      33840   25730   21389   17335   14518   12489   10918   9647    9242    8895    8561    8305    8093    7937    7755
1024k   33789   26149   21576   17293   14445   12420   10872   9599    9204    8879    8591    8306    8096    7925    7769
512k    32371   26585   22237   18039   15289   13153   11513   10168   9839    9505    9319    9035    9059    8969    8786
256k    33783   26261   23099   19581   17342   15079   13191   11578   11518   11751   11541   12787   12204   13217   13665
128k    36743   29559   28987   27763   24451   21706   19522   17154   17766   17703   17808   17865   17500   17082   16829
64k     36677   29569   29096   28010   24342   22200   20041   18256   17542   17512   20093   20906   18130   18546   30124
32k     33741   46897   45894   45438   43271   43688   42028   41900   43264   43962   41759   41549   40320   40004   38938
16k     46156   44450   42864   41224   40083   40083   37440   44067   42062   42588   42233   46041   42499   40560   42588
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain$
```



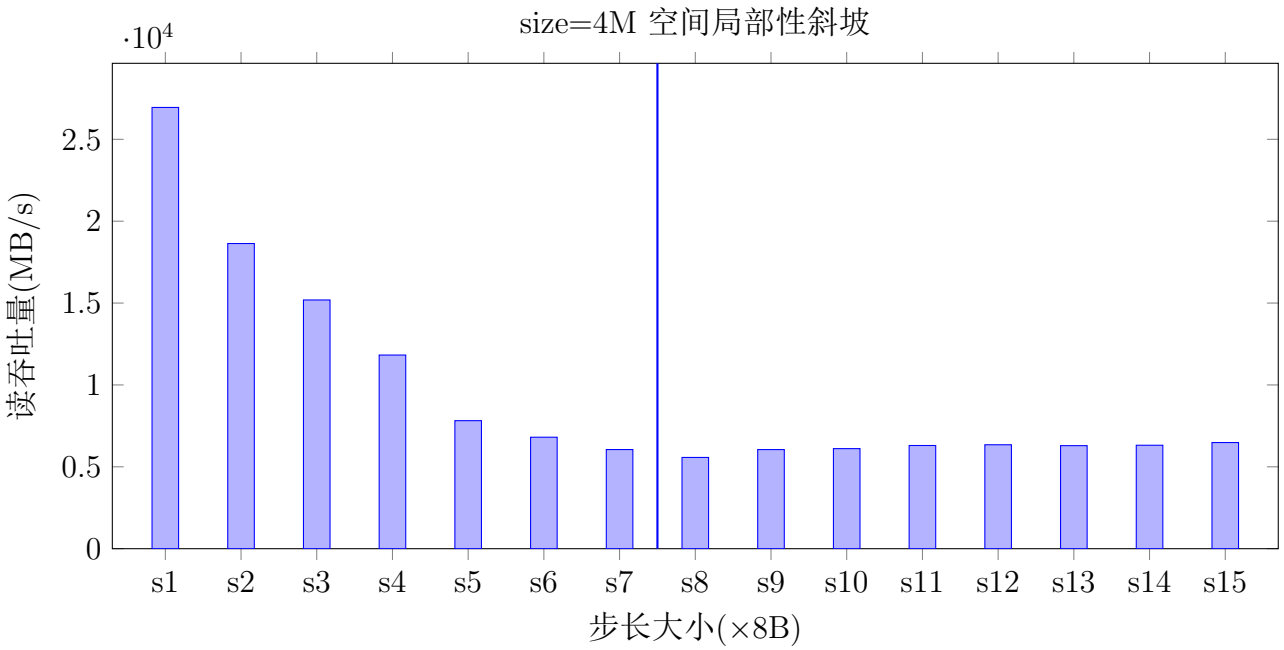
- 截图得到

```
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE                 32768
LEVEL1_ICACHE_ASSOC                8
LEVEL1_ICACHE_LINESIZE             64
LEVEL1_DCACHE_SIZE                 32768
LEVEL1_DCACHE_ASSOC                8
LEVEL1_DCACHE_LINESIZE             64
LEVEL2_CACHE_SIZE                  262144
LEVEL2_CACHE_ASSOC                 4
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  4194304
LEVEL3_CACHE_ASSOC                 16
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC                 0
LEVEL4_CACHE_LINESIZE              0
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project4/src/lab04/mountain$
```

缓存级别	大小（字节）
L1	32768
L2	262144
L3	4194304

结果一致。

- 固定数组长度为 4MB，有



图像是一致的。块的大小为 $8 \times 8 = 64$ bytes，因为从8开始就是每个高速缓存行一次访问，所以0~8都是块大小以内的范围。