计算机系统结构第一次实验

李雨田　2010012193　计 14

March 26, 2014

Contents

1 测量数据缓存的大小

1.1 实验原理

基本思想是对一段大小的数据反复读取，并且逐次增加数据的大小。当数据的大小超过缓存的大小时，频繁读取会导致频繁替换缓存，使得吞吐

量下降。所以只要测量吞吐量，观察发现突变的点，即可得到缓存的大小。

但是最新的 Intel 处理器自带硬件预读取功能，即会根据程序执行的步长预测下一次访问，并且提前读取到内存里。如果按照顺序访问数组的方法，结果发现吞吐量一直不会下降，正是因为硬件预读取提前替换了缓存，没有影响到读取的效率。

为了不让处理器预测出访问的步长，可以每次产生一个伪随机数作为下标访问。但是产生随机数会影响吞吐量的测试，并且调用外部函数时会导致内存访问，使得之前的缓存失效。所以只能仿照链表的实现方式，把下一次访问的地址放在这次访问的地址所对应的变量中。并且合理增大步长，加大替换缓存的次数。

1.2 实验结果和数据分析

运行程序，对 1KB 到 2048KB 之间大小都进行测试，并且得出吞吐量。吞吐量的单位是 MB/s, 但是因为使用 `clock()` 函数记时并不是很准确。不过最重要的是相对数值，所以并不影响测量缓存的大小。

源代码为 `cache-size.c`, 运行时通过命令行提供两个参数，分别为起始和结束测量的大小。然后程序会给出相应大小下的吞吐量。

程序输出如图1所示，结果如图2所示。注意到横轴对应的是 2^xKB. 可以看出在 32KB 和 256KB 处有明显的吞吐量的突降，于是知道 L1 和 L2 数据缓存的大小分别为 32KB 和 256KB.

```
size: 1KB, throughput: 4723.100180
size: 2KB, throughput: 4498.637010
size: 4KB, throughput: 3501.863357
size: 8KB, throughput: 2790.821903
size: 16KB, throughput: 2286.950924
size: 32KB, throughput: 2200.802842
size: 64KB, throughput: 1297.379616
size: 128KB, throughput: 1298.050199
size: 256KB, throughput: 1282.880066
size: 512KB, throughput: 707.177437
size: 1024KB, throughput: 713.854618
size: 2048KB, throughput: 709.695294
```

Figure 1: 数据缓存大小程序输出

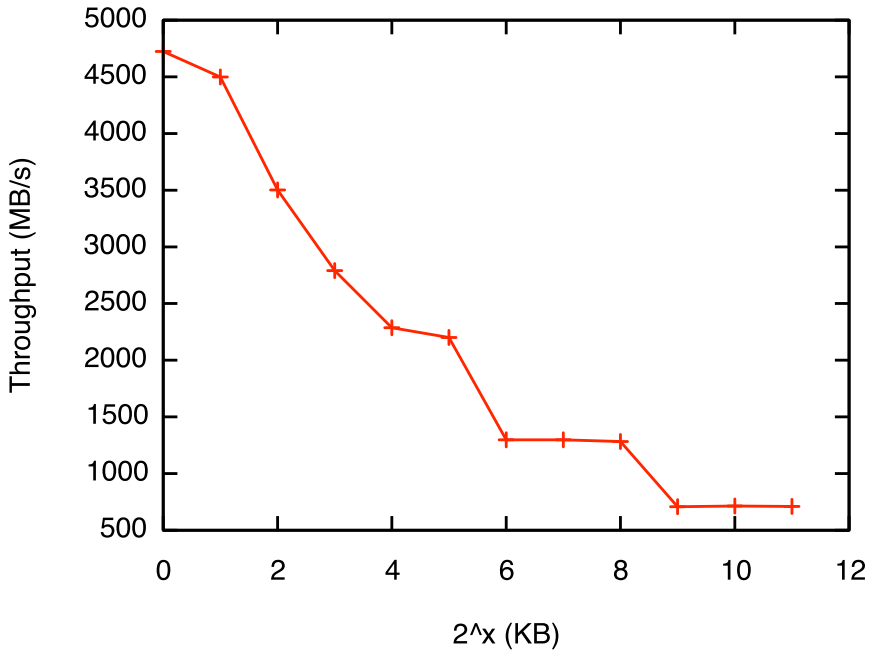

Figure 2: 数据缓存大小数据

2 测量数据缓存的块大小

2.1 实验原理

同样是对内存进行顺序访问，但是读到块里的某一个字节时，整个块都会被缓存进来。所以如果按字节顺序访问，仅仅会在访问该块的第一个字节的时候对访问更低级存储，接下来在该块内的访问会更快。如果不每个字节都依次访问，加大访问的步长，可以预见当步长等于块大小的时候，每一次读取就会要访问更低级存储，将整个块都加载进来，这样的吞吐量是最低的。每次逐次加大步长的话，将会出现吞吐量先降后升。最低点对应的步长即是块大小。

这里任然要注意到硬件预读取带来的影响，同样使用类似链表的数据结构。

2.2 实验结果和数据分析

程序源代码为 `block-size.c`. 运行程序，对步长从 1 到 32 进行测试。这里的步长是指 `uint64_t` 的长度，即 64B.

得到程序输出如图3

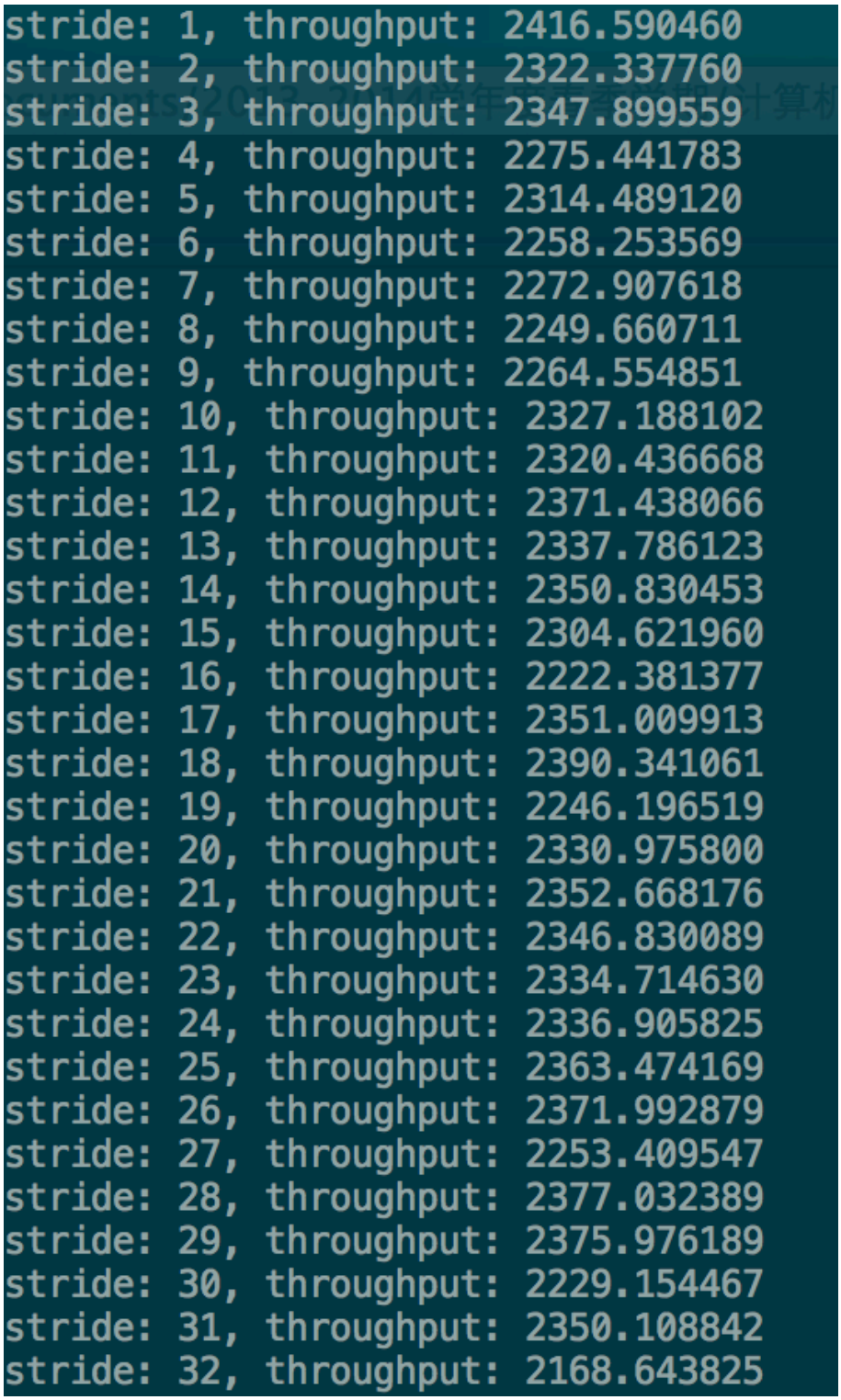

Figure 3: 数据缓存块大小程序输出

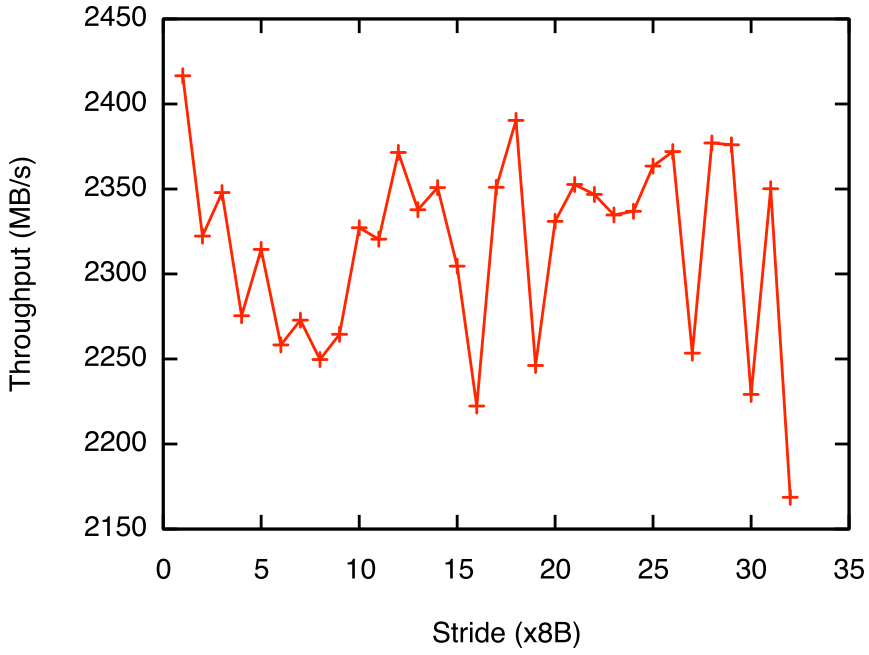

Figure 4: 数据缓存块大小数据

3 测量数据缓存的相连度

3.1 实验原理

到现在已经知道块大小是 64B，缓存的大小也已经测出来了。下面只要测出来一共有多少个组，就能知道每个组的大小和相连度了。

已知块大小是 64B，占了地址最低的 6 位。取一个掩码，用来分割地址前面的标签和后面的索引和偏移量部分。实际上可以取这个掩码加一的值，每次往地址上加这个值。如果掩码没有盖住索引和偏移量部分，那么往地址上加的时候会改变索引，会从一个组跳到另一个组。如果掩码正好盖住或者超过了索引和偏移量，那么往地址上递增的时候就只会改变标签，而任然还在同一个组内。

所以通过依次左移掩码，当掩码正好盖住索引和偏移量的时候，所有读取的内容都属于同一个缓存组，缓存冲突频率最大，吞吐量最低。通过观察吞吐量下降到极值这个点，即可算出相连度。

3.2 实验结果和数据分析

程序源代码为 `associativity.c`. 依次左移掩码，得到程序输出如图5. 这里的 `mask` 实际上是掩码加一的值，即往地址上累加的值。

数据如图6. 可以看出当掩码为 12 位的时候吞吐量最低，即地址的低 12 位是索引和偏移量。偏移量为 6 位，所以索引为 6 位，共有 64 组。一级缓存共有 32KB，而块大小之前已经算出来是 64B，计算得到一组里有 8 个块，即时 8 相连的。

考虑到一级缓存和二级缓存有一定的一致性，猜测二级缓存也是 8 相连的。之前已经得到二级缓存共 256KB，算得一共应有 512 组。对应的掩码有 15 位，可以看到如图6当掩码为 15 位时，吞吐率的增长趋势得到抑制，验证猜测是正确的。

对于图6后面吞吐量开始上升，是因为程序采用固定读取量，测量时间得到吞吐量。当掩码变大时，实际上每一步跳跃距离变大，但数组大小不变，为了达到同样的读取量，只能多次重复读取。很可能即使是在同一组中，但因为不同的地址的数量太少，无法使这个组的缓存填满，更不用说替换缓存了，所以吞吐量会变高。

```
mask: 64, throughput: 1792.765735
mask: 128, throughput: 1713.085742
mask: 256, throughput: 1735.105773
mask: 512, throughput: 1675.176147
mask: 1024, throughput: 1563.110590
mask: 2048, throughput: 1187.185469
mask: 4096, throughput: 1083.229929
mask: 8192, throughput: 1490.857035
mask: 16384, throughput: 2022.253677
mask: 32768, throughput: 4039.332208
mask: 65536, throughput: 5217.116626
mask: 131072, throughput: 3767.756796
```

Figure 5: 数据缓存相连度程序输出

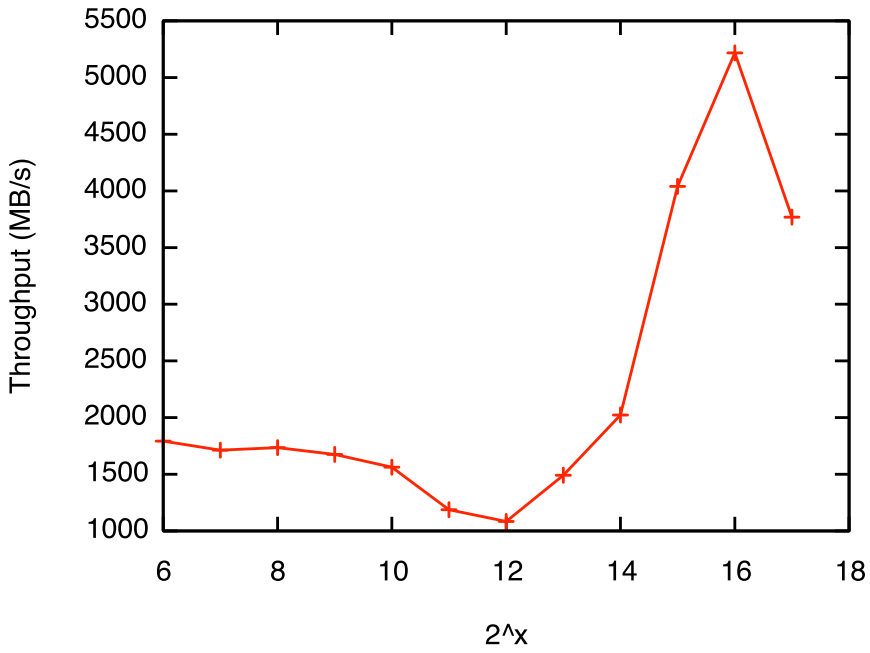

Figure 6: 数据缓存相连度数据

4 对所给程序 matrix_mul.cpp 进行优化

对于 $A \times B = C$ 的矩阵乘法。对于指数的顺序，一共有三种方式。其中 ijk 和 jik 等效，jki 和 kji 等效，kij 和 ikj 等效。

根据 *Computer Systems A Programmer's Perspective* 上的结论，有如表1所示结论。

所以只要采用 kij 或者 ikj 方式，就能大幅利用空间局部性提高效率。经测试使用原来的方法耗时7 658 105 ms，使用 ikj 方式只需 2790251ms.

Matrix multiply version	Loads per iter.	Stores per iter.	A misses per iter.	B misses per iter.	C misses per iter.	Total misses per iter.
ijk & jik	2	0	0.25	1.00	0.00	1.25
jki & kji	2	1	1.00	0.00	1.00	2.00
kij & ikj	2	1	0.00	0.25	0.25	0.50

Table 1: 矩阵乘法效率

5 测量数据缓存的块大小

5.1 实验原理

5.2 实验结果和数据分析