缓存测量实验

计01　容逸朗　2020010869

实验机器参数

- L1 Cache: 32 KB (per core)
- L2 Cache: 256 KB (per core)
- L3 Cache: 6 MB
- Cache associativity: 8
- Cache Line Size: 64 B

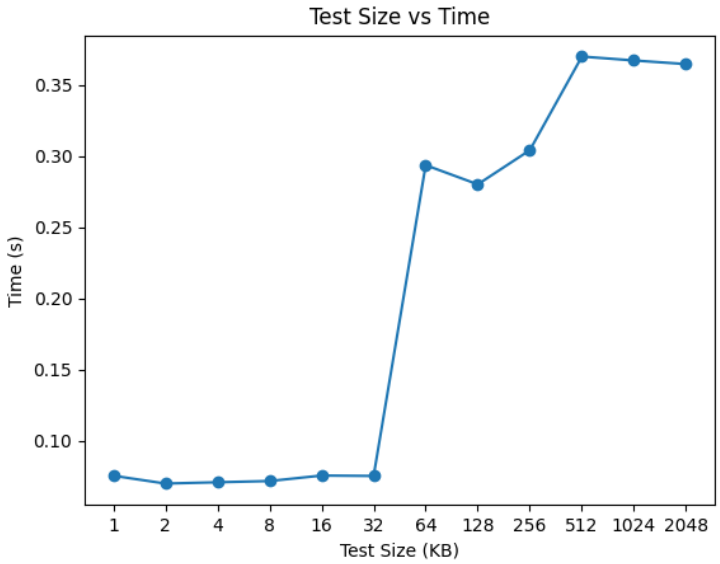
测量内容

1. 测量缓存大小 (Cache Size)

实验方式

- 以 uint8_t (1 byte) 为单位，分别访问大小为 1KB 至 2048KB 的数组 2^{27} 次。
- 访存序列：0, 64, 128, ... , size - 64, 0, 64, 128, ...（间隔 64 byte 访存）

实验结果



- 从上图可见，L1 DCache 结果和系统参数匹配，而 L2 Cache 结果偏小。

思考题

1. 理论上 L2 Cache 的测量与 L1 DCache 没有显著区别。但为什么 L1 DCache 结果匹配但是 L2 Cache 不匹配呢？你的实验有出现这个现象吗？请给出一个合理的解释。

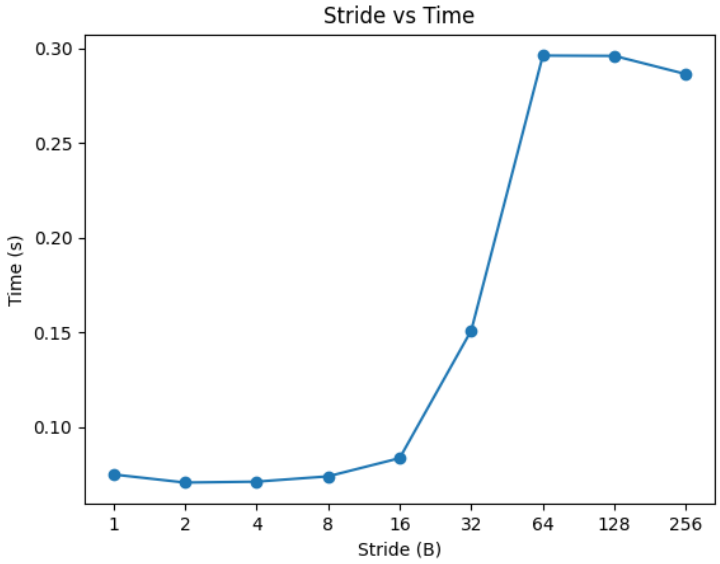
- 因为 L1 DCache 只会存放数据，故实验结果匹配，但 L2 Cache 除了存放数据外，还存放了指令，因此实验结果会偏小。

2. 测量缓存行大小 (Cache Line Size)

实验方式

- 以 uint8_t (1 byte) 为单位，分别以步长 1 - 256 访问大小为 64KB 的数组各 2^{27} 次。
- 访存序列：0, stride, stride * 2, ... （间隔 stride (byte) 访存）

实验结果



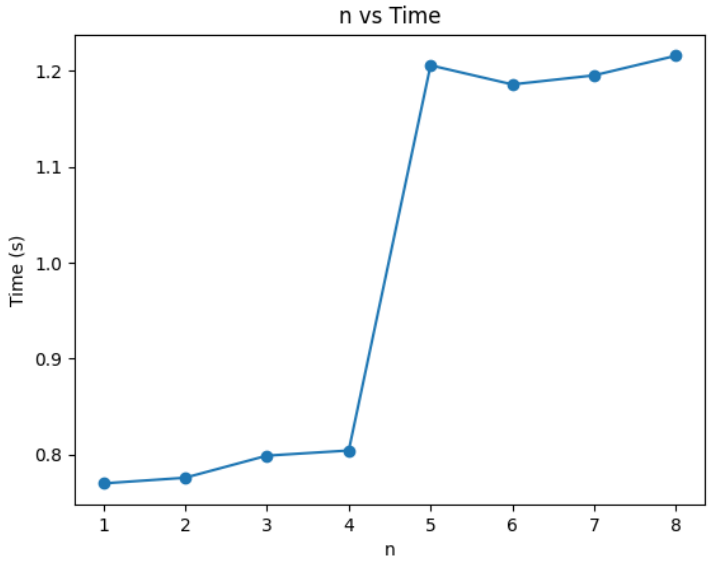
- 从中可见，运行时间在 64B 步长之后显著提升，与预期相符。

3. 测量缓存相联度 (Cache Associativity)

实验方式

- 采用实验思路 1；
- 取 $n = 1, 2, \ldots, 8$ ，测试使用 64KB 大小的数组，然后将数组平均分成 2^n 块，只访问其中的奇数块。
- 假设每块的大小为 sz，则此时访存序列为：$sz, sz+1, \ldots, 2sz-1, 3sz, 3sz+1, \ldots, 4sz-1, 5sz, \ldots, (2^n-1)sz, (2^n-1)sz+1, \ldots, 2^n sz-1$

实验结果



- 由图中可见，从 $n = 5$ 开始访问时间显著增加，故 $2^{5-2} = 8$ 为相联度，与预期相符。

算法分析

- 不妨记 Cache 的参数为：
  - Cache line size: $B = 2^b$ bits
  - Cache size: 2^s bits
  - Cache associativity: 2^w
- 那么 Cache line 共有 $X = 2^{s-b-w}$ 个可映射的值，每个值包含 2^w 个槽位。
- 接下来考虑实验相关的参数：
  - 数组大小为 2^{s+1} bits
  - 每次操作把数组分为 2^n 块
  - 每块包含了 $M = 2^{s-n+1}$ bits，对应 $L = 2^{s-n+1-b}$ 个 Cache line size 大小的块（即 $M = L \cdot B$）
  - 特别地，当访存地址为 M 时，表示访问 $[M, M + B)$ 中的所有数据
- 此时访存阵列记为：$M, M + B, M + 2B, \ldots, M + (L - 1)B, 3M, 3M + B, \ldots, 3M + (L - 1)B,$ $\ldots, (2^n - 1) \cdot M, (2^n - 1) \cdot M + B, \ldots, (2^n - 1) \cdot M + (L - 1)B$
- 显然地，当 $X|L$ ，即 $n \leq w + 1$ 时：
  - 访问每一块 $[xM, xM + M)$ 中的所有数据，对应位 Index 为 $xL, xL + 1, \ldots xL + L - 1$，模 X 后得到 $k = \frac{L}{X}$ 组 $0, 1, \ldots, X - 1$，因此需要 X 个 Cache Index 各 $k = 2^{w-n+1}$ 个槽位来储存这一块的数据。
  - 每次遍历访存阵列时，我们共访问了 2^{n-1} 个这样的块，对于每个 Cache Index，各需要 $2^{n-1} \cdot k = 2^w$ 个槽位，故所有 Cache Line 都能被数据填满，但又不至于发生替换。
- 接下来考虑 $n = w + 2$ 时的情况：
  - 此时有 $X = 2L$，访问奇数块 $[2xM - M, 2xM)$ 中的所有数据时，对应位置的 Index 为 $(2x - 1)L, (2x - 1)L + 1, \ldots, 2xL - 1$，模 X 后得到 $L, L + 1, L + 2, \ldots, 2L - 1$
  - 每次遍历访存阵列时，我们访问了 2^{n-1} 个奇数块，因此对于上述计算所得的每个 Cache Index，都需要槽位 $2^{n-1} = 2^{w+1} > 2^w$。由于 Cache 的槽位不足，所以每次访存是都必定会出现 Cache Miss 从而导致访存时间显著上升。
- 故当访存时间显著增加时，n 满足条件 $n = w + 2$，由此可知相联度为 $2^w = 2^{n-2}$。

4. 矩阵乘法优化 (MatMul Optimization)

优化方法

- 首先更改 j 和 k 的循环次序，使得最终的循环由外而内分别为 $i-k-j$:
  - 这使得运算时用到的三个矩阵都能以连续方式存取。
- 我们还可以手动循环展开，减少分支预测所需的计算量;
- 由于实验只能使用 O0 优化（即不使用任何优化），因此我们需要手动 “提取公因式”:
  - 在计算 $d[i][j]=a[i][k]*b[k][j]$ 时，可以把 $a[i][k]$ 的值放入寄存器中（因为最内层循环 j 和 $a[i][k]$ 的值无关，故可以提取），减少访存次数。
  - 同时由于我们采用了 “循环展开” 的方式，因此也可以把 $d[i][j_0]$ 和 $b[k][j_0]$ 的地址作为基址，在后续访存时使用基址 + 偏移量的方式访存也可以有效减少地址计算时所需的时间。

实验结果

- 最终在本机上得到 8 倍的加速比。

```
(base) BoxWorld:cache boxworld$ ./mat_mul
time spent for original method : 9.49442 s
time spent for new method : 1.11592 s
time ratio of performance optimization : 8.50813
```

其他

- 本次实验较为友好，整体难度尚可。