

Lab07

Exercise 1 - A Couple of Memory Access Scenarios

在本任务中，我们需要修改程序的参数和 cache 的参数，去看在每种情况下 cache 的表现如何。

不得不说 venus 真是个好东西。

Exercise 2 - Loop Ordering and Matrix Multiplication

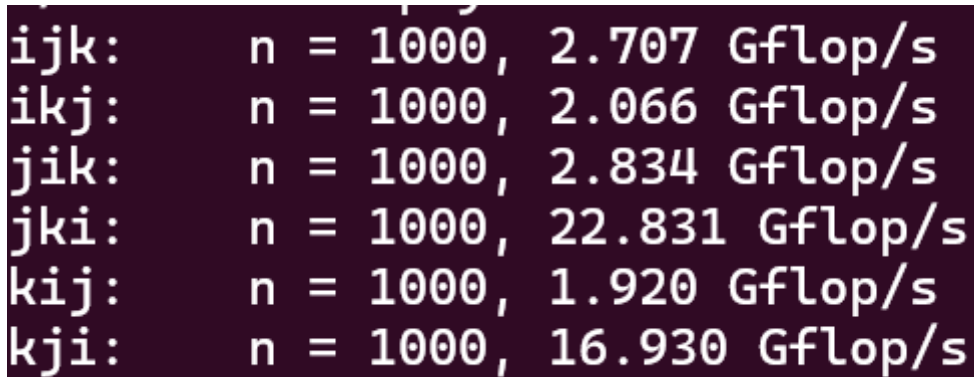
我们先来看一个矩阵乘法的例子：

```
1 for (int i = 0; i < n; i++)
2     for (int j = 0; j < n; j++)
3         for (int k = 0; k < n; k++)
4             c[i+j*n] += A[i+k*n] * B[k+j*n];
```

在这个例子中，三个矩阵均存储为一个一维数组，其中 i 表示列指标， j 表示行指标， $i+j*n$ 表示 (i,j) 。

那么上面这个例子表示的是：每次固定 A 的第 i 列，循环 B 的每一行，计算出 C 的第 i 列所有元素。注意本代码在计算 $B*A = C$

显然的是，我们任意改变遍历 i,j,k 的顺序并不会改变最终的结果，但是不同的遍历顺序会有截然不同的运行速度



ijk:	n = 1000,	2.707 Gflop/s
ikj:	n = 1000,	2.066 Gflop/s
jik:	n = 1000,	2.834 Gflop/s
jki:	n = 1000,	22.831 Gflop/s
kij:	n = 1000,	1.920 Gflop/s
kji:	n = 1000,	16.930 Gflop/s

其中 jki 是最快的,而 kij 是最慢的

- jki : 当 $j=0, k=0$ 时, $c[i] += a[i] * b[0]$

当 $j=0, k=1$ 时, $c[i] += a[i+n] * b[1]$

我们可以发现，每次计算都是遍历 a 的某一行， b 某个固定的位置， c 的某一行

这样可以极大的利用 spatial locality

- `kij` 当 `k=0, i=0` 时, $c[j * n] += a[0] * b[j * n]$

当 `k=0, i=1` 时, $c[1 + j * n] += a[1] * b[j * n]$

每次都是按照列去遍历 `c, b`, 完全不用 `locality`.