并行与分布式作业

第二次作业

姓名：陈琮昊
班级：人工智能与大数据
学号：18340013

一、 问题描述

（1）分别采用不同的算法（非分布式算法）例如一般算法、分治算法和 Strassen 算法等计算两个 300x300 的矩阵乘积，并通过 Perf 工具分别观察 cache miss、CPI、mem_load 等性能指标。

（2）Design an experiment (i.e. design and write programs and take measurements) to determine the memory bandwidth of your computer and to estimate the caches at various levels of the hierarchy. Use this experiment to estimate the bandwidth and L1 cache of your computer. Justify your answer. (Hint: To test bandwidth, you do not want reuse. To test cache size, you want reuse to see the effect of the cache and to increase this size until the reuse decreases sharply.)

（3）Consider a memory system with a level 1 cache of 32 KB and DRAM of 512 MB with the processor operating at 1 GHz. The latency to L1 cache is one cycle and the latency to DRAM is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is four words). What is the peak achievable performance of a dot product of two vectors? Note: Where necessary, assume an optimal cache placement policy.

```
/* dot product loop */
for (i = 0; i < dim; i++)
        dot_prod += a[i] * b[i];
```

（4）Now consider the problem of multiplying a dense matrix with a vector using a two-loop dot-product formulation. The matrix is of dimension 4*K* x 4*K*. (Each row of the matrix takes 16 KB of storage.) What is the peak achievable performance of this technique using a two-loop dot-product based matrix-vector product?

```
/* matrix-vector product loop */
for (i = 0; i < dim; i++)
        for (j = 0; i < dim; j++)
                c[i] += a[i][j] * b[j];
```

二、 解决方案

（3）不妨设所有数据均在 cache 内，则 4K 个 word 需要 $400\mu s$，而每个向量有 2K 个 word，共需要进行 4K 次计算，即 1K 个周期，时间是 $1\mu s$，故结果为 $\frac{4K}{401\mu s}=99.75FLOPS$

（4）每一行为 16KB，则两行时 cache 就已经满了。剩下的数据则要从 DRAM 中取。从 cache 中取数据，一次取 4K 个 word，大约耗时 $400\mu s$，取 2K 次则需要：$400\mu s\times 2K=0.8s$；从 DRAM 中取需要 2×10^5 个周期，即 $200\mu s$。一共进行了 $2\times\left(4\times10^3\right)^3=2^7\times10^9$ 次运算，故结果为 $\frac{2^7\times10^9}{800200\mu s}=160GFLOPS$

三、 实验结果

(3)、(4)结果在解决方案一栏已给出，此处则给出(1)的结果：

注：下面的截图从上至下依次是普通、分治、Strassen。

(i)CPI/IPC：可以看到 Strassen 在更多指标上占优。

```
        440.65 msec task-clock                #    0.999 CPUs utilized
             4      context-switches          #    0.009 K/sec
             0      cpu-migrations            #    0.000 K/sec
           380      page-faults               #    0.862 K/sec
 1,362,822,498      cycles                    #    3.093 GHz
 3,900,997,637      instructions              #    2.86  insn per cycle
   439,235,442      branches                  #  996.780 M/sec
        95,754      branch-misses             #    0.02% of all branches

   0.440972185 seconds time elapsed
```

```
      6,041.41 msec task-clock                #    1.000 CPUs utilized
            31      context-switches          #    0.005 K/sec
             0      cpu-migrations            #    0.000 K/sec
       108,330      page-faults               #    0.018 M/sec
18,684,443,005      cycles                    #    3.093 GHz
41,279,398,630      instructions              #    2.21  insn per cycle
 6,495,400,655      branches                  # 1075.147 M/sec
    34,541,361      branch-misses             #    0.53% of all branches

   6.042349567 seconds time elapsed
```

```
      6,838.43 msec task-clock                #    1.000 CPUs utilized
            29      context-switches          #    0.004 K/sec
             0      cpu-migrations            #    0.000 K/sec
       143,510      page-faults               #    0.021 M/sec
21,142,843,677      cycles                    #    3.092 GHz
48,061,861,312      instructions              #    2.27  insn per cycle
 7,510,559,969      branches                  # 1098.287 M/sec
    11,688,952      branch-misses             #    0.16% of all branches
```

(ii)mem-load：正常，三者均为 0

```
             0      mem-loads

  0.441286802 seconds time elapsed
```

```
             0      mem-loads

  6.030203902 seconds time elapsed
```

```
             0      mem-loads

 6.879902739 seconds time elapsed
```

(iii)cache-miss：一般算法最简单，没有递归所以 cache miss 最少；而分治算法和 Strassen 就多很多，时间也长很多；当然由于 Strassen 算法更复杂，cache miss 也就多一些。

四、 遇到的问题及解决方法

第(2)题电脑配置不是很给力且较为复杂所以没有完成。三个方法的代码编程实现起来并不难。主要是在安装 perf 的过程中可以说是碰到了不少的小问题，通过不断地网上查博客最终得以解决。