并行与分布式作业

“”

第二次作业

姓名：谷正阳
班级：行政一班
学号：18308045

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 算法等计算矩阵两个 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 4K x 4K. (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

问题 1：我查阅资料并编写了几种矩阵乘法的代码，然后在另一台电脑上安装 ubuntu，在上面使用 perf 工具

问题 2：经过查阅资料，memory bandwidth 可通过 stream 工具测试，而 cr0 寄存器的修改在 user mode 下无法进行，所以没有实现评估不同级的存储，仅将代码打包在压缩文件中。

问题 3:

∵ Cache line size is four words

∴ Every 4 iterations, there are 2 cache misses for a[i] and b[i]

∵ Every 4 iteration, there are 8 ops

∴ 8ops / (2 * 100cycles / 1GHz) = 40MFLOPS

问题 4:

∵ Cache line size is four words

∵ 32 KB > 16KB + 1word

∵ An optimal cache placement policy

∴ In the first outer iteration, every 4 iterations, there are 2 cache misses for a[i] and b[i], and in other outer iterations, every 4 iterations, there are 1 cache misses

∵ Every 4 iteration, there are 8 ops

∴ 8ops * 4K / ((2 * 1 + 1 * (4K – 1)) * 100cycles / 1GHz) = 79.98000499875MFLOPS

三、 实验结果

1.

```
Correct!

 Performance counter stats for './multiply':

           74,584      cache-misses
    1,085,040,586      cycles
    2,929,575,458      instructions              #    2.70  insn per cycle
                0      mem-loads

      0.411669623 seconds time elapsed

      0.411662000 seconds user
      0.000000000 seconds sys
```

```
Correct!

 Performance counter stats for './divide_and_conquer':

           80,806      cache-misses
    2,852,178,453      cycles
    7,110,703,063      instructions              #    2.49  insn per cycle
                0      mem-loads

      1.066457271 seconds time elapsed

      1.066293000 seconds user
      0.000000000 seconds sys
```

```
Correct!

 Performance counter stats for './strassen':

         1,652,537      cache-misses
   387,399,301,810      cycles
   700,493,472,069      instructions              #    1.81  insn per cycle
                 0      mem-loads

     143.908669099 seconds time elapsed

     143.901721000 seconds user
       0.000000000 seconds sys
```

三种乘法对 300 x 300 两个单位矩阵相乘，得出结果也为 300 x 300 单位矩阵则输出“Correct!”。并分别分析性能，结果 mem-loads 均为 0，其他指标分治算法比普通的算法性能差，因为分治算法递归的过程种不断地创建新变量，而且加法乘法次数并没有减少。Strassen 算法性能比其他两个算法性能都差，预测因为递归过程中构建太多变量（比分治多，因为分治实现没有构建临时矩阵，而 Strassen 实现构建了临时矩阵）。

2.

使用 gcc 编译 stream.c 成二进制可执行文件

gcc -O3 -mcmodel=medium -fopenmp -DSTREAM_ARRAY_SIZE=100000000 -DNTIMES=100 stream.c -o stream

参数说明：

-O：编译器编译优化级别；

-mcmodel=mediu：当单个 Memory Array Size 大于 2GB 时需要设置此参数。

-fopenmp：适应多处理器环境；开启后，程序默认线程为 CPU 线程

数。

-DSTREAM_ARRAY_SIZE=100000000：指定计算中 a[],b[],c[]数组的大小，

-DNTIMES=100：执行的次数，并且从这些结果中选最优值。

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 100000000 (elements), Offset = 0 (elements)
Memory per array = 762.9 MiB (= 0.7 GiB).
Total memory required = 2288.8 MiB (= 2.2 GiB).
Each kernel will be executed 100 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 4
Number of Threads counted = 4
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 146522 microseconds.
   (= 146522 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           10363.7     0.154758     0.154385     0.156930
Scale:           7659.4     0.209338     0.208894     0.213734
Add:             8744.0     0.274918     0.274473     0.278521
Triad:           8726.4     0.276013     0.275027     0.280865
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

最终结果，bandwidth = data_size / time，数组的复制 10363.7MB/s，数组的尺度变换 7659.4MB/s，数组的矢量求和 8744MB/s，数组的复合矢量求和 8726.4MB/s。

四、遇到的问题及解决方法

问题 1：mem-loads 都为 0。

解决 1：经过上网查询，mem-loads 不是统计 loads 总数的指标，而是用来统计代码中的哪些指令发送 load 请求的时候占用多余一个特定时钟周期完成的。如果想统计 load uops retired（微操作完成）总数，要使用 L1-dcache-loads。

```
Correct!

 Performance counter stats for './multiply':

           95,474      cache-misses

    1,082,862,676      cycles

    2,929,604,509      instructions              #    2.71  insn per cycle

    1,049,200,135      L1-dcache-loads


       0.409524907 seconds time elapsed

       0.409540000 seconds user
       0.000000000 seconds sys
```

```
Correct!

 Performance counter stats for './divide_and_conquer':

          119,566      cache-misses

    2,856,133,164      cycles

    7,110,675,256      instructions              #    2.49  insn per cycle

    2,867,823,470      L1-dcache-loads


       1.071096381 seconds time elapsed

       1.063146000 seconds user
       0.008023000 seconds sys
```

```
Correct!

 Performance counter stats for './strassen':

        2,241,527      cache-misses

   387,634,803,077      cycles

   700,492,931,006      instructions              #    1.81  insn per cycle

   238,365,157,305      L1-dcache-loads


     144.097338806 seconds time elapsed

     144.094827000 seconds user
       0.008000000 seconds sys
```

可以看到 L1-dcache-loads 不再是 0，且 L1-dcache-loads 越大，性能越小。

问题 2：Strassen 算法从除 mem-loads 外各个指标上都比其他两个算法性能差。

解决 2：尝试 Strassen 一次，后面使用普通算法，来看有没有提升。

```
Correct!

 Performance counter stats for './strassen_n':

          204,244      cache-misses

      985,325,541      cycles

    2,704,225,395      instructions              #    2.74  insn per cycle

      965,742,512      L1-dcache-loads


      0.371906282 seconds time elapsed

      0.367902000 seconds user
      0.003998000 seconds sys
```

可以看到结果有提升，且比普通算法更快。其中 instructions 比普通算法少，普通算法相当于分为 4 块，按照分块矩阵乘法做 8 次乘法，而 Strassen 只需要做 7 次。而在 L1-dcache-loads 小的情况下 cache-misses 大于普通算法，可能因为 Strassen 算法实现时分配了栈内存建立临时临时矩阵，导致空间连续性不好。