分布式系统作业

第 2 次作业

姓名：唐晨轩

班级：人工智能与大数据

学号：19335182

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, 找出特征或者规律。

2. 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];
```

3. 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

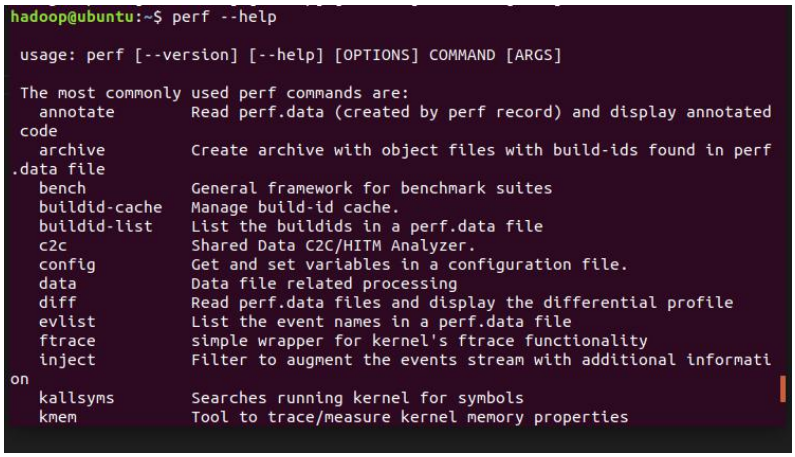
1. 根据传统算法，两个 n 阶矩阵相乘，对于 n2 个元素，每个元素想要被计算出来，至少要进行 n 次乘法和 n-1 次加法，算法复杂度达到 O(n^3)。

2. 这里的分治算法是利用矩阵分块将矩阵分为 4 个子矩阵，这样就可以对两个二乘二的矩阵进行乘法，最后再合并起来得到结果。简单的分治算法的时间复杂度仍然是 O(n^3)的。

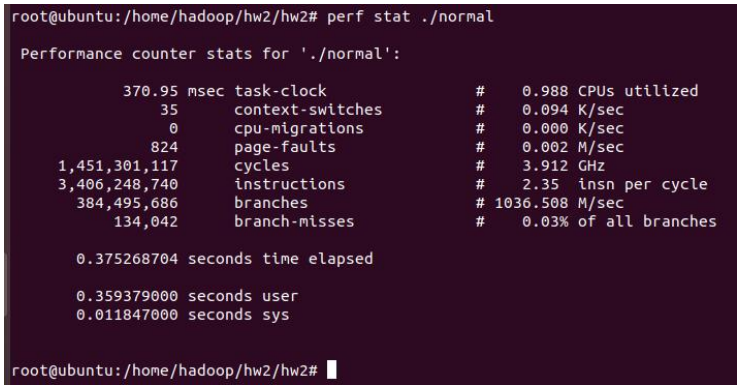
3.strassen 乘法：通过数学构造，将 8 个不同的小方阵转化为 7 个不同的小方阵，从而减少相乘的次数。时间复杂度大概为 O(n^2.8)

使用 perf 观察 cache-misses,cpi,mem-loads 等性能指标:

先安装好 perf:



一般算法:



```
root@ubuntu:/home/hadoop/hw2/hw2# perf stat -e cache-misses,instructions,cycles ./normal

 Performance counter stats for './normal':

         2,279,219      cache-misses
     3,405,922,916      instructions              #    2.40  insn per cycle
     1,417,669,403      cycles

       0.366522579 seconds time elapsed

       0.364537000 seconds user
       0.000000000 seconds sys


root@ubuntu:/home/hadoop/hw2/hw2#
```

可以看到 cache-misses 为 2279219 次,指令数量为 3485922916 条,平均 cpu cycles 只有 1417669483 条，计算得到 cpi = 1417669483/3485922916=0.4067，mem_loads 我查看不了，一查看就报错所以这里没有贴出。

普通分治：

```
root@ubuntu:/home/hadoop/hw2/hw2# perf stat ./DC

 Performance counter stats for './DC':

          1,639.84 msec task-clock                #    0.985 CPUs utilized
               336      context-switches          #    0.205 K/sec
                 1      cpu-migrations            #    0.001 K/sec
             1,250      page-faults               #    0.762 K/sec
     6,387,289,364      cycles                    #    3.895 GHz
    12,001,587,660      instructions              #    1.88  insn per cycle
     1,827,387,864      branches                  # 1114.369 M/sec
         9,155,824      branch-misses             #    0.50% of all branches

       1.664528344 seconds time elapsed

       1.591386000 seconds user
       0.047622000 seconds sys


root@ubuntu:/home/hadoop/hw2/hw2#
```

```
root@ubuntu:/home/hadoop/hw2/hw2# perf stat -e cache-misses,instructions,cycles ./DC

 Performance counter stats for './DC':

         5,640,918      cache-misses
    11,999,935,245      instructions              #    1.86  insn per cycle
     6,439,834,621      cycles

       1.680727758 seconds time elapsed

       1.619714000 seconds user
       0.027857000 seconds sys


root@ubuntu:/home/hadoop/hw2/hw2#
```

cache-misses 为 5640918 次，与一般算法在一个数量级，指令数量为 11999935245 条，平均 cpu cycles 有 6439834621 条，计算得到 cpi = 6439834621/11999935245=0.5367，CPI 有所提升，但是可以看到平均时间消耗从 0.36 秒上升到 1.68 秒，说明程序在递归调用时消耗大量时间。

Strassen 算法：

```
root@ubuntu:/home/hadoop/hw2/hw2# perf stat ./strassen

 Performance counter stats for './strassen':

          265.54 msec task-clock                #    0.969 CPUs utilized
              33      context-switches          #    0.124 K/sec
               1      cpu-migrations            #    0.004 K/sec
           2,156      page-faults               #    0.008 M/sec
   1,041,513,645      cycles                    #    3.922 GHz
   2,915,537,385      instructions              #    2.80  insn per cycle
     337,739,779      branches                  # 1271.910 M/sec
         382,694      branch-misses             #    0.11% of all branches

       0.274157564 seconds time elapsed

       0.257965000 seconds user
       0.007937000 seconds sys


root@ubuntu:/home/hadoop/hw2/hw2#
```

```
root@ubuntu:/home/hadoop/hw2/hw2# perf stat -e cache-misses,instructions,cycles ./strassen

 Performance counter stats for './strassen':

         1,802,099      cache-misses
     2,915,165,533      instructions              #    2.80  insn per cycle
     1,042,630,925      cycles

       0.268183294 seconds time elapsed

       0.248712000 seconds user
       0.016309000 seconds sys


root@ubuntu:/home/hadoop/hw2/hw2#
```

cache-misses 为 1802899 次，与上面是一个数量级，指令数量为 2915165533 条，平均 cpu cycles 有 1042630925 条，计算得到 cpi = 1042630925/2915165533=0.3577.

3. 在这里我假设数组的元素都是 4 字节的 float 类型，一个 word 是 4 个字节，cache 分配时不会把上一次刚分配的块覆盖掉，i,dot_prod 两个变量放在寄存器里。则对每一次循环：

①如果 i%4==0，则读取 a[i]时需要从内存中取出 a[i],a[i+1],a[i+2],a[i+3]放在 cache 里，需要 100ns，读取 b[i]时需要从内存中取出 b[i],b[i+1],b[i+2],b[i+3]放在 cache 里，需要 100ns。则这次循环需要 100ns+100ns+4ns(比较 i 与 dim 要 1ns,i++要 1ns,a[i]b[i]要 1ns，将结果加到 dot_prod 要 1ns)，共 204ns。

②如果 i%4！=0，则 a[i],b[i]可以从 cache 里取出，需要 2ns。则这次循环需要 2ns+2ns+4ns，共 8ns。

所以,每次循环平均需要 (8*3+204*1)/4=57ns 故每秒钟能进行 17.5M 次循环。考虑到每次循环是进行了两次浮点数运算，所以能达到的最高性能是 35MFLOPS

3.我们仍然假设 cache 不会把刚刚分配的块覆盖掉，在取 a[i][j]时只需访问一次内存 cache，i 和 j 两个变量放在寄存器里，且第一重循环中 i<dim，i++这两个操作可以忽略不计。值得注意的是 cache 的大小为 32k,而 b 数组大小为 16k，矩阵 a 的一行也是 16k，如果 cache 分配策略是最优的，则 b 数组会在 i=0 时全部装进 cache 里，只有 a 数组需要重复取出，所以在内存里取出 b 数组的时间也可以忽略。

则对每一次循环：

① 如果 j%4==0 ， 则读取 a[i][j] 时需要从内存中取出 a[i][j],a[i][j+1],a[i][j+2],a[i][j+3]放在 cache 里，需要 100ns。c[i]只有一开始会从内存中取到 cache 里(这一次的用时也忽略不计)，之后一直在寄存器里放着。

则这次循环需要 100ns+6ns(假设比较 j 与 dim 要 1ns,j++要 1ns,a[i][j]b[j]要 1ns，取出 c[i]要 2ns，将结果加到 c[i]要 1ns)，共 106ns。

②如果 i%4！=0，则 a[i][j],b[j]可以从 cache 里取出，需要 2ns。则这次循环需要 2ns+2ns+6ns，共 10ns。

所以，每次循环平均需要 （3*10+106*1）/4=34ns,每秒钟能执行 29.4M

次循环，能达到的最高性能是 59.8MFLOPS。

三、 实验结果

四、遇到的问题及解决方法