计算机系统结构(A)

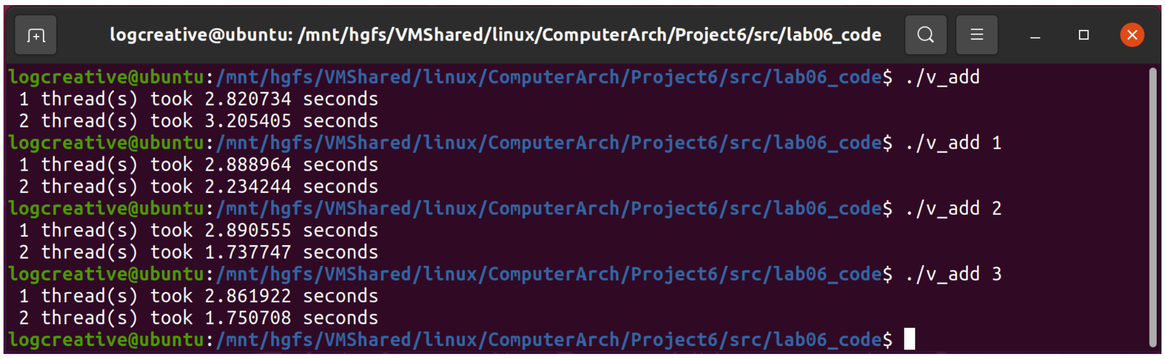
实验 6

李子龙 518070910095

2021 年 6 月 1 日

一. 向量加法

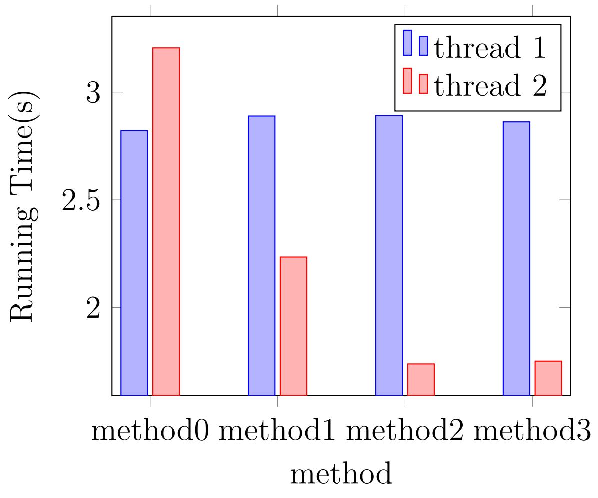
1. 运行结果如下：



logcreative@ubuntu: /mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code

```
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./v_add
 1 thread(s) took 2.820734 seconds
 2 thread(s) took 3.205405 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./v_add 1
 1 thread(s) took 2.888964 seconds
 2 thread(s) took 2.234244 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./v_add 2
 1 thread(s) took 2.890555 seconds
 2 thread(s) took 1.737747 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./v_add 3
 1 thread(s) took 2.861922 seconds
 2 thread(s) took 1.750708 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ 
```

假共享问题，访问同一个数组，导致一个处理器修改数组后，多个处理器的数组已经不再有效，需要通过广播更改数组。`method_1()` 受到 cache 大小和处理器个数的影响，虽然使用了 `gap` 在一定程度上缓解了逐步访问相邻块导致的假共享，但是由于处理器仍然访问相邻块，cache大小足够小时才会有更好的性能，否则依然会 dirty。



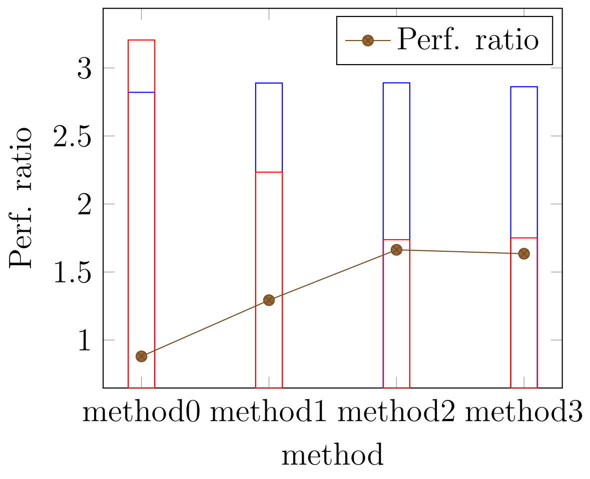


图 1: 不同方法的性能

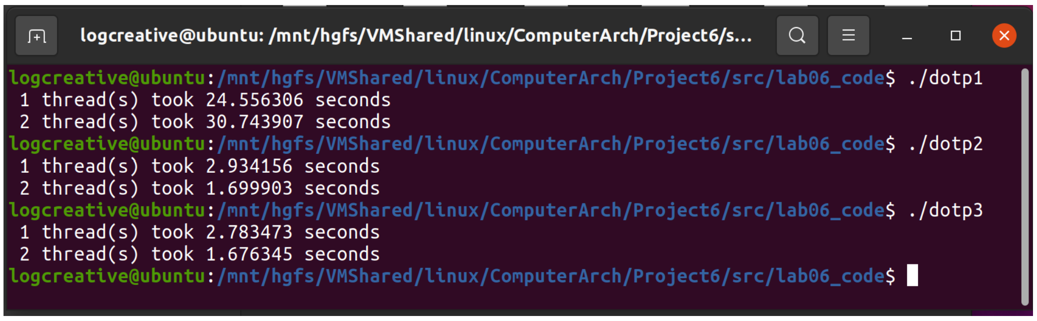
2. 达到了同等的性能。

```c
void method_3(double* x, double* y, double* z) {
#pragma omp parallel
{
  // your code here:
  int block = omp_get_num_threads();
  int n = omp_get_thread_num();
  int block_size = ARRAY_SIZE / block + 1;
  int low = n * block_size;
  int high = (n + 1) * block_size;
  for(int i = low; i < high && i < ARRAY_SIZE; ++i){
    z[i] = x[i] + y[i];
  }
}

}
```

二. 点积

1. 运行结果如下：



```
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./dotp1
 1 thread(s) took 24.556306 seconds
 2 thread(s) took 30.743907 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./dotp2
 1 thread(s) took 2.934156 seconds
 2 thread(s) took 1.699903 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$ ./dotp3
 1 thread(s) took 2.783473 seconds
 2 thread(s) took 1.676345 seconds
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/ComputerArch/Project6/src/lab06_code$
```

线程数目越多，反而性能越差。当线程数增多时，进入临界区的请求就会变多，有时候反而就会因为排队时间的增长而导致较长的运行时间。

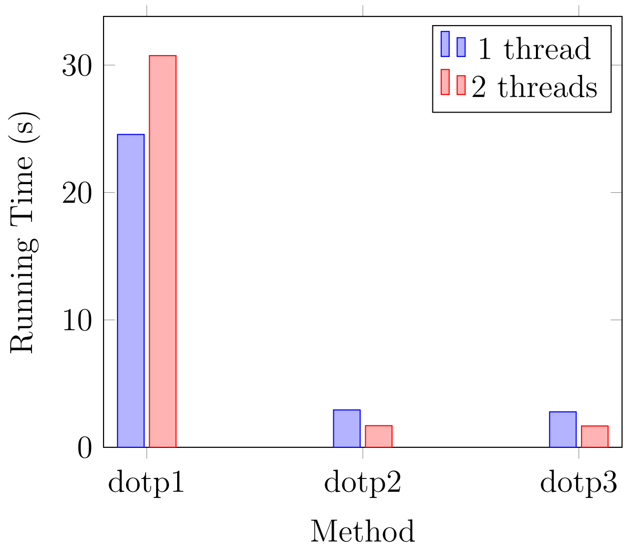


图 2: 点积程序运行结果

2. 性能得到了巨大的改变。

```c
double dotp_2(double* x, double* y) {
   double global_sum = 0.0;
   #pragma omp parallel
   {
     // your code here: modify dotp_1 to improve performance
      double local_sum = 0.0;
      #pragma omp for
      for (int i = 0; i < ARRAY_SIZE; i++) {
         local_sum += x[i] * y[i];
      }
      #pragma omp critical
          global_sum += local_sum;
   }
   return global_sum;
}
```

3. reduction 的作用[1]:

> *A private copy for each list variable is created and initialized for each thread. At the end of the reduction, the reduction variable is applied to all private copies of the shared variable, and the final result is written to the global shared variable.*

和 `dotp_2` 的方法几乎一致，性能与 `dotp_2` 也几乎一致。

[1] https://hpc.llnl.gov/openmp-tutorial#REDUCTION