《Pthreads 并行程序计算稠密向量的 SAXPY》

--测评报告

洪瑶　1801111621

目录

一、并行算法描述

输入参数为：p（线程数量），n（2^n 大小的数组），α （系数）；

计算稠密向量的 SAXPY 的数学模型为：

$$B = \alpha A + B$$

其中 A，B 向量的大小为 2^n，最后更新 B 的值。

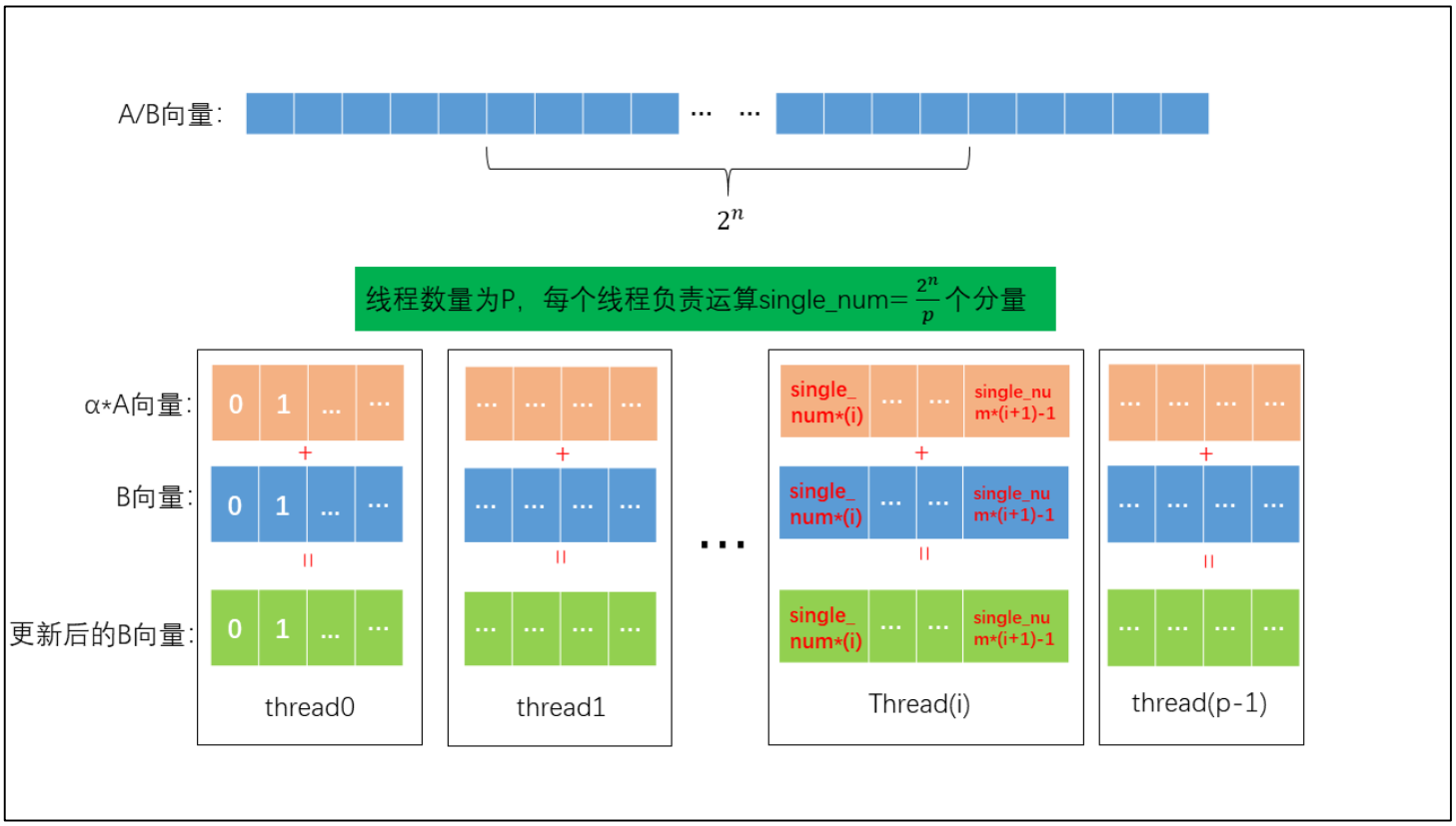


图 1 并行算法概图

从图 1 可以看到将需要求解的向量进行分线程求解，最后合并即得到新的向量。

二、改变线程数量和迭代空间规模，所编写并行程序的评测结果

（1）、控制 P=4，改变 N.

N=4:

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^4
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=4.1846e-04
reference parallel impl.: time_cost=1.6375e-04 speedup=2.555
your impl.              : time_cost=2.2370e-04 speedup=1.871
```

N=10:

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^10
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=3.8990e-04
reference parallel impl.: time_cost=1.7721e-04 speedup=2.200
your impl.              : time_cost=2.3822e-04 speedup=1.637
```

N=14:

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^14
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=7.3183e-04
reference parallel impl.: time_cost=2.9642e-04 speedup=2.469
your impl.              : time_cost=2.6048e-04 speedup=2.810
```

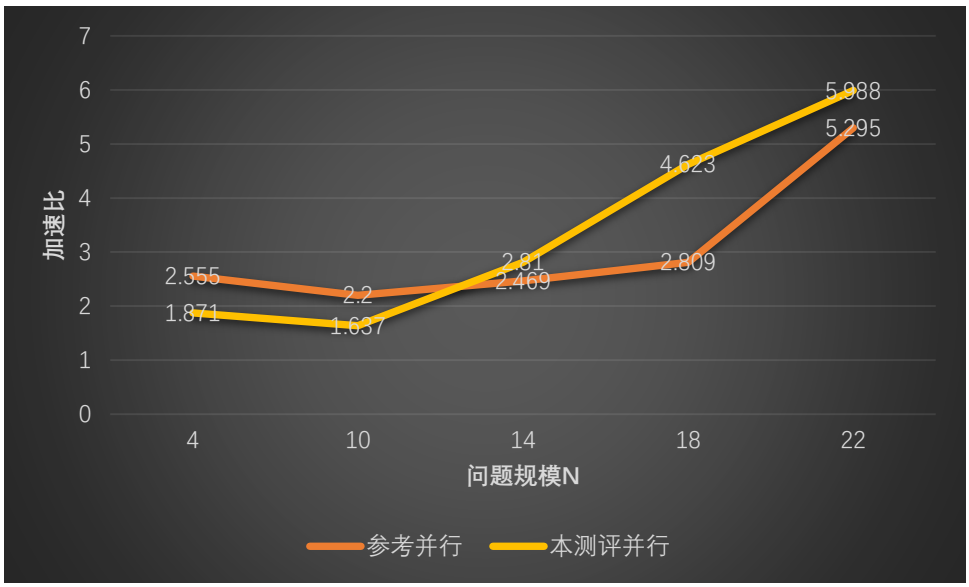
N=18：

```
**************************stdout from your program**************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^18
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=4.6742e-03
reference parallel impl.: time_cost=1.6642e-03 speedup=2.809
your impl.              : time_cost=1.0110e-03 speedup=4.623
```

N=22：

```
**************************stdout from your program**************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^22
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=6.9471e-02
reference parallel impl.: time_cost=1.3121e-02 speedup=5.295
your impl.              : time_cost=1.1601e-02 speedup=5.988
```

加速比变化图：



（2）、控制 N=24，改变 P

P=2：

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^24
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4119e-01
reference parallel impl.: time_cost=5.8905e-02 speedup=4.095
your impl.              : time_cost=4.2035e-02 speedup=5.738
```

P=4：

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^24
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.3687e-01
reference parallel impl.: time_cost=5.4639e-02 speedup=4.335
your impl.              : time_cost=4.3927e-02 speedup=5.392
```

P=8：

```
****************************stdout from your program****************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^24
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4206e-01
reference parallel impl.: time_cost=5.2586e-02 speedup=4.603
your impl.              : time_cost=4.2626e-02 speedup=5.679
```

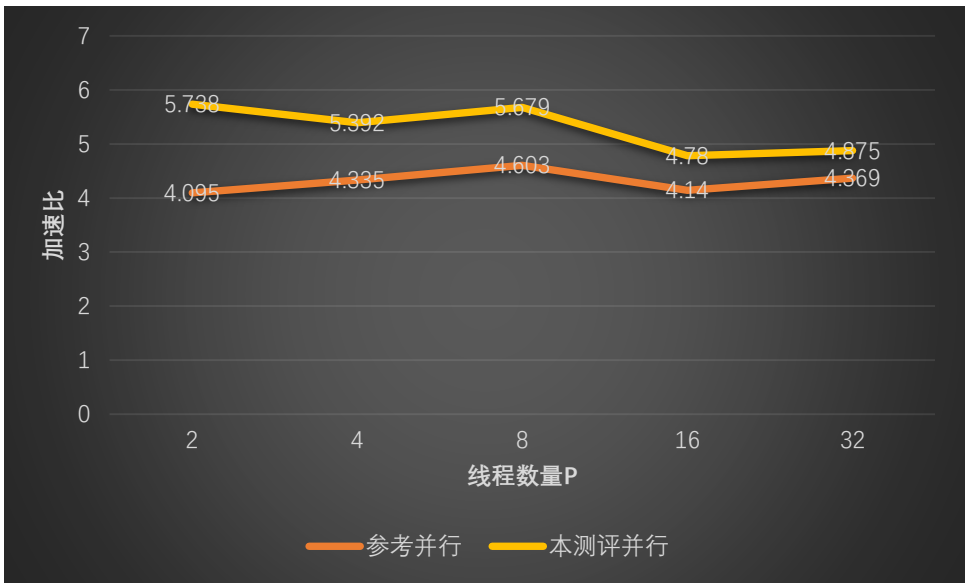
P=16：

```
**************************stdout from your program**************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^24
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.3840e-01
reference parallel impl.: time_cost=5.7584e-02 speedup=4.140
your impl.              : time_cost=4.9870e-02 speedup=4.780
```

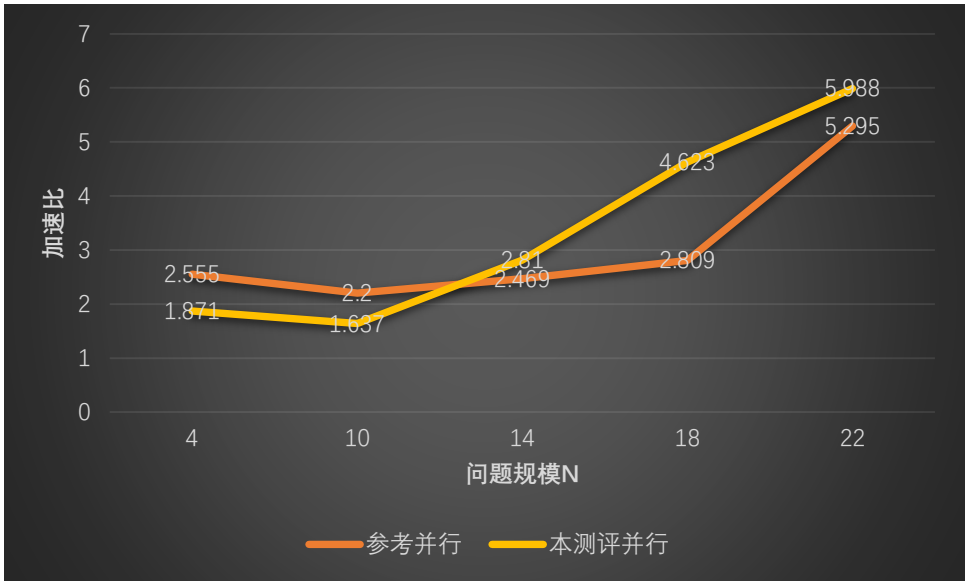
P=32：

```
**************************stdout from your program**************************
hello: thread is running!
***************value evaluation*****************
array-size = 2^24
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.3715e-01
reference parallel impl.: time_cost=5.4282e-02 speedup=4.369
your impl.              : time_cost=4.8648e-02 speedup=4.875
```

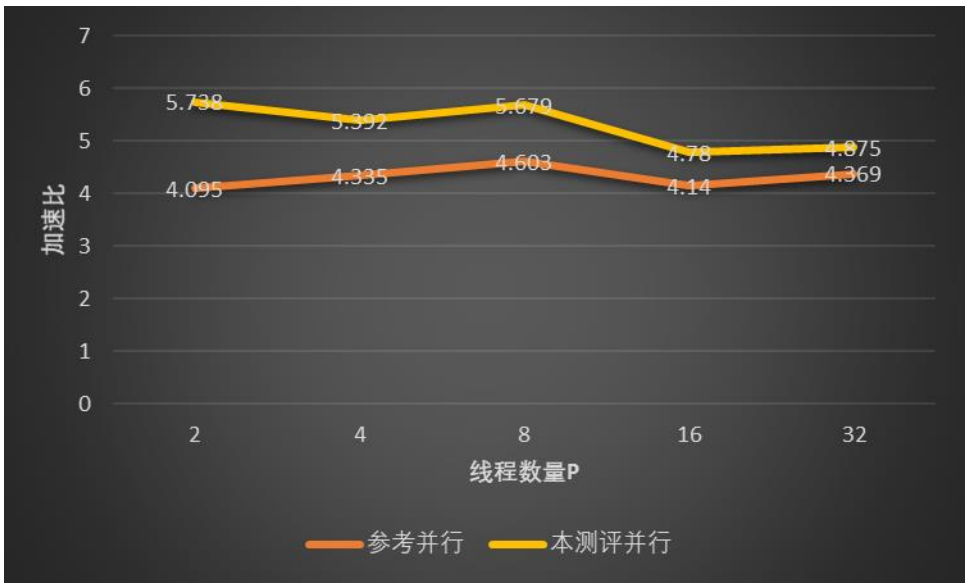
加速比变化图：



三、评测结果分析



上图为 P=4 时加速比随着问题规模的变化图，在 N 比较小的时候，程序的并行加速比比较小，随着 N 的增大，加速比也越来越大。在 N 比较小的时候加速比不如线程数量大，但是随着 N 的增加，加速比会超过 P=4, 这是因为在串行程序中，数据的导入导出需要时间，而在数据比较多的时候，并行将数据导入时间除去之后会超过线程的数量。另外，本测评的加速比在 N 较大时加速比超过参考加速比。



上图为控制 N=24,加速比随着线程数量的变化情况，可以看到和加速比随着问题规模的变化趋势完全不一样，加速比随着 P 的增加并没有增加，可见线程增加的同时，join 和 barrier 函数导致的等待时间等其他开销时间也相应增加，从而并不会导致加速比随着线程数量的增加而有明显的提升。

四、Pthread 程序

```
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <vector>
#include <math.h>
#include "fio.h"

int64_t n,n_num,sizeofA,sizeofB,single_num;
int32_t thread_num,threadid;
float *A,*B,alfa;
pthread_barrier_t barrier;

void  *worker(void *arg){
      int64_t i;
      int myID = __sync_fetch_and_add(&threadid, 1);
      for             (i=myID*single_num;i<(myID+1)*single_num;i++)
*(B+i)=(*(A+i))*alfa+*(B+i);
      return (void*)0;
}

int main(int argc, char *argv[]){
     int i;
     thread_num=atoi(argv[1]);        //      设置 P
     n=atoll(argv[2]);                        //      设置 N
     n_num=((int64_t)1<<n);           //      设置 2^N 大小的数组
     single_num=n_num/thread_num;      //单个线程的分配数量
     alfa=atof(argv[3]);
     threadid=0;
     sizeofA = (sizeof(float)) *n_num;       //A 数组空间大小
     sizeofB = (sizeof(float)) *n_num;       //B 数组空间大小
     A= (float *)malloc(sizeof(float)*n_num);   //开辟 A 数组空间
     B= (float *)malloc(sizeof(float)*n_num);     //开辟 B 数组空间
     input_data(A, sizeofA, B, sizeofB);        //输入 AB。
```

```
          printf("hello: thread is running!\n");
        // printf("A[2]=%f\n;B[2]=%f\nalfa=%f", A[2], B[2],alfa);  //检
查
         pthread_barrier_init(&barrier,NULL,thread_num);    //设置栅樟
         pthread_t *threads = new pthread_t[thread_num];
         for(int           i=0;           i<thread_num;           i++)
pthread_create(&(threads[i]),NULL,worker,&thread_num);
         for(int i=0; i<thread_num; i++)  pthread_join(threads[i],NULL);
         output_data(B,sizeofB ); //输出 B
         //printf("B[2]=%f\n", B[2]);
         //printf("B[1]=%f\n", *(B+1));
         delete[ ] threads;
         pthread_barrier_destroy(&barrier);      //删除栅樟
         return EXIT_SUCCESS;
    }
```