Project4: Matrix Multiplication in C

12010508华羽霄

1. Implement a function matmul_plain () in a straightforward way using several loops as the benchmark.

```
#include <stdio.h> //预处理命令，基本的输入输出函数库
#include <stdlib.h>//随机函数rand()所在库
#include <time.h>


void matmul_plain(const float *p1, const float * p2, size_t nSize){
    float each[nSize*nSize];
    //下面这句话不知道为什么不起效果
    // memset(each,0,sizeof(each));
    for(size_t r_init=0;r_init<nSize;r_init++){
        for(size_t c_init=0;c_init<nSize;c_init++){
            // 初始化结果矩阵为全0
            each[r_init*nSize+c_init]=0;
        }
    }

    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            for(size_t i=0;i<nSize;i++){
                //用一维数列表示矩阵乘法有点麻烦
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
            }
        }
    }

}


int main(int argc, char ** argv){
    //用命令行输入的方式确定矩阵的行列大小
    size_t nSize = atoi(argv[1]);
    size_t total_element=nSize*nSize;

    //用malloc分配两个矩阵内存
    float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        //两个矩阵内所有元素均为0到1之间的随机数
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    matmul_plain(p1, p2, nSize);
    end = clock();
    int time_ms=end-start;
    //如果时间较短，用ms作单位，否则用s
    if(time_ms<1000){
        printf("plain,%dms\n",time_ms);
    }else{
        int time_s=time_ms/CLOCKS_PER_SEC;
        printf("plain,%ds\n",time_s);
    }

    //释放内存
    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % gcc test_plain.c
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out 16
```

```
plain,63ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out 128
plain,0s
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out 1000
plain,5s
# 这个结果显然是有问题的，后面会做修改
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out 8000
zsh: segmentation fault  ./a.out 8000
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out 64000
zsh: segmentation fault  ./a.out 64000
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % gcc test_plain.c -o3
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./3 16
plain,40ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./3 128
plain,0s
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./3 1000
plain,5s
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./3 8000
zsh: segmentation fault  ./3 8000
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./3 64000
zsh: segmentation fault  ./3 64000
```

2. Implement a function matmul_improved () using SIMD, OpenMP and other technologies to improve the speed. You can compare it with matmul_plain()'.

(1) unloop

```
#include <stdio.h> //预处理命令，基本的输入输出函数库
#include <stdlib.h>//随机函数rand()所在库
#include <time.h>


void matmul_unloop(const float *p1, const float * p2, size_t nSize){
    //必须确保矩阵的行列大小为8的倍数，这里我注释掉了
    // if(nSize % 8 != 0){
    //     std::cerr << "The size n must be a multiple of 8." <<std::endl;
    //     return;
    // }
    float each[nSize*nSize];
    for(size_t r_init=0;r_init<nSize;r_init++){
        for(size_t c_init=0;c_init<nSize;c_init++){
            each[r_init*nSize+c_init]=0;
        }
    }

    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            for(size_t i=0;i<nSize;i+=8){
                //一次性计算8次乘法运算，然后再加起来
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+1]*p2[c+(i+1)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+2]*p2[c+(i+2)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+3]*p2[c+(i+3)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+4]*p2[c+(i+4)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+5]*p2[c+(i+5)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+6]*p2[c+(i+6)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+7]*p2[c+(i+7)*nSize];
            }
        }
    }

}


int main(int argc, char ** argv){

    size_t nSize = atoi(argv[1]);
    size_t total_element=nSize*nSize;

    float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
```

```
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    matmul_unloop(p1, p2, nSize);
    end = clock();
    int time_ms=end-start;
    if(time_ms<1000){
        printf("unloop,%dms\n",time_ms);
    }else{
        int time_s=time_ms/CLOCKS_PER_SEC;
        printf("unloop,%ds\n",time_s);
    }

    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % gcc test_unloop.c
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./a.out 16
unloop,30ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./a.out 128
unloop,0s
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./a.out 1000
unloop,5s
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./a.out 8000
zsh: segmentation fault  ./a.out 8000
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./a.out 64000
zsh: segmentation fault  ./a.out 64000
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % gcc test_unloop.c -o3
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3 16
unloop,33ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3 128
unloop,0s
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3 1000
unloop,5s
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3 8000
zsh: segmentation fault  ./3 8000
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3 64000
zsh: segmentation fault  ./3 64000
```

(2) neon

```
#include <stdio.h> //预处理命令，基本的输入输出函数库
#include <stdlib.h>//随机函数rand()所在库
#include <time.h>


void matmul_unloop(const float *p1, const float * p2, size_t nSize){
    //必须确保矩阵的行列大小为8的倍数，这里我注释掉了
    // if(nSize % 8 != 0){
    //     std::cerr << "The size n must be a multiple of 8." <<std::endl;
    //     return;
    // }
    float each[nSize*nSize];
    for(size_t r_init=0;r_init<nSize;r_init++){
        for(size_t c_init=0;c_init<nSize;c_init++){
            each[r_init*nSize+c_init]=0;
        }
    }

    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            for(size_t i=0;i<nSize;i+=8){
                //一次性计算8次乘法运算，然后再加起来
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+1]*p2[c+(i+1)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+2]*p2[c+(i+2)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+3]*p2[c+(i+3)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+4]*p2[c+(i+4)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+5]*p2[c+(i+5)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+6]*p2[c+(i+6)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+7]*p2[c+(i+7)*nSize];
            }
        }
```

```
    }

}


int main(int argc, char ** argv){

    size_t nSize = atoi(argv[1]);
    size_t total_element=nSize*nSize;

    float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    matmul_unloop(p1, p2, nSize);
    end = clock();
    int time_ms=end-start;
    if(time_ms<1000){
        printf("unloop,%dms\n",time_ms);
    }else{
        int time_s=time_ms/CLOCKS_PER_SEC;
        printf("unloop,%ds\n",time_s);
    }

    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % gcc test_neon.c
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./a.out 16
neon,6ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./a.out 128
neon,9ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./a.out 1000
neon,5ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./a.out 8000
neon,8ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./a.out 64000
neon,4ms
```

(3) openmp

```
#include <stdio.h> //预处理命令，基本的输入输出函数库
#include <stdlib.h>//随机函数rand()所在库
#include <time.h>


void matmul_openmp(const float *p1, const float * p2, size_t nSize){
    float each[nSize*nSize];
    for(size_t r_init=0;r_init<nSize;r_init++){
        for(size_t c_init=0;c_init<nSize;c_init++){
            each[r_init*nSize+c_init]=0;
        }
    }

    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            for(size_t i=0;i<nSize;i++){
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
            }
        }
    }

}


int main(int argc, char ** argv){
    size_t nSize = atoi(argv[1]);
```

```
    size_t total_element=nSize*nSize;

    float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=1;
        p2[i]=1;
  }

    clock_t start,end;
    start = clock();
    //openmp方法与plain方法的唯一区别在于这句话，启用了并行计算
    #pragma omp parallel for
    matmul_openmp(p1, p2, nSize);
    end = clock();
    int time_ms=end-start;
    if(time_ms<1000){
        printf("openmp,%dms\n",time_ms);
    }else{
        int time_s=time_ms/CLOCKS_PER_SEC;
        printf("openmp,%ds\n",time_s);
    }

    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % gcc test_openmp.c
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./a.out 16
openmp,53ms
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./a.out 128
openmp,0s
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./a.out 1000
openmp,5s
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./a.out 8000
zsh: segmentation fault  ./a.out 8000
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./a.out 64000
zsh: segmentation fault  ./a.out 64000
```

3. Test the performance using 16×16, 128×128, 1K×1K, 8K8K and 64K×64K matrices. You can generate some matrics with random values to test your function.

不难发现，当数组长度超过一定限度之后，会发生segmentation fault的错误。查阅一番资料后，我决定在数组创建时候，加入static前缀，这样就能将数组存储在data segment中，而非stack中。这二者的区别就在于，data segment的空间比stack大得多，因此能存储下更大的数组。但同样会带来一个问题，nSize是由用户指定的，在创建可变长度数组时，不可以使用static，所以我不得不将数组的可变长改为定长。另外，我编写了一个print_time方法，能够根据花费时间的长短选取合适的单位（毫秒、秒、分钟）。

(1) plain

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//宏定义数组大小，看情况自行修改
#define nSize 16


void matmul_plain(const float *p1, const float * p2){
    static float each[nSize*nSize];

    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            //这里优化了一下，把初始化和计算写在一起，减少2次循环
            each[r*nSize+c]=0;
            for(size_t i=0;i<nSize;i++){
```

```
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
            }
            //这段话是为了观察计算进度
            // float sequence=(float)r*nSize+c;
            // float total=(float)nSize*nSize;
            // float percent=sequence/total*100;
            // printf("%luth,%.2f%c\n",r*nSize+c,percent,'%');
        }
    }
}


void print_time(int time_ms){
    //如果时间较短，用毫秒作单位，否则用秒,超过60秒改用分钟
    printf("plain,nSize=%d,time=",nSize);
    int time_s=time_ms/1000;
    if(time_ms<1000){
        printf("%dms\n",time_ms);
    }else if(time_s<60){
        printf("%ds\n",time_s);
    }else{
        int time_m_part=time_s/60;
        int time_s_part=time_s%60;
        printf("%dm%ds\n",time_m_part,time_s_part);
    }
}


int main(){
    size_t total_element=nSize*nSize;

    static float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    static float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    matmul_plain(p1, p2);
    end = clock();
    int time_ms=(int)(end-start)/1000;

    print_time(time_ms);

    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % gcc final.c
(base) hyx13701490089@huayuxiaodeMacBook-Pro plain % ./a.out
plain,nSize=16,time=0ms
plain,nSize=128,time=7ms
plain,nSize=1000,time=3s
plain,nSize=8000,time=53m4s
# nSize=64K时，计算时间超级超级长，就没有记录了
```

(2) unloop

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define nSize 16


void matmul_unloop(const float *p1, const float * p2){
    static float each[nSize*nSize];
    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            each[r*nSize+c]=0;
            for(size_t i=0;i<nSize;i+=8){
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
```

```
                each[r*nSize+c]+=p1[r*nSize+i+1]*p2[c+(i+1)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+2]*p2[c+(i+2)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+3]*p2[c+(i+3)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+4]*p2[c+(i+4)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+5]*p2[c+(i+5)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+6]*p2[c+(i+6)*nSize];
                each[r*nSize+c]+=p1[r*nSize+i+7]*p2[c+(i+7)*nSize];
            }
            //这段话是为了观察计算进度
            float sequence=(float)r*nSize+c;
            float total=(float)nSize*nSize;
            float percent=sequence/total*100;
            printf("%luth,%.2f%c\n",r*nSize+c,percent,'%');
        }
    }
}


void print_time(int time_ms){
    printf("unloop,nSize=%d,time=",nSize);
    int time_s=time_ms/1000;
    if(time_ms<1000){
        printf("%dms\n",time_ms);
    }else if(time_s<60){
        printf("%ds\n",time_s);
    }else{
        int time_m_part=time_s/60;
        int time_s_part=time_s%60;
        printf("%dm%ds\n",time_m_part,time_s_part);
    }
}


int main(){
    size_t total_element=nSize*nSize;

    static float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    static float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    matmul_unloop(p1, p2);
    end = clock();
    int time_ms=(int)(end-start)/1000;
    print_time(time_ms);

    free(p1);
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % gcc final.c -o3
(base) hyx13701490089@huayuxiaodeMacBook-Pro unloop % ./3
unloop,nSize=16,time=0ms
unloop,nSize=128,time=25ms
unloop,nSize=1000,time=6s
unloop,nSize=8000,time=35m47s
```

(3) neon

```
#include <stdio.h> //预处理命令，基本的输入输出函数库
#include <stdlib.h>//随机函数rand()所在库
#include <time.h>
#include <arm_neon.h>
#define nSize 16


void matmul_neon(float32_t *A, float32_t *B, float32_t *C){
  //相比于上一版，在创建变量的时候更简洁高效
  float32x4_t A0,A1,A2,A3;
```

```
  float32x4_t B0,B1,B2,B3;
  float32x4_t C0,C1,C2,C3;

  A0 = vld1q_f32(A);
  A1 = vld1q_f32(A+4);
  A2 = vld1q_f32(A+8);
  A3 = vld1q_f32(A+12);

  C0 = vmovq_n_f32(0);
  C1 = vmovq_n_f32(0);
  C2 = vmovq_n_f32(0);
  C3 = vmovq_n_f32(0);

  B0 = vld1q_f32(B);
  C0 = vfmaq_laneq_f32(C0, A0, B0, 0);
  C0 = vfmaq_laneq_f32(C0, A1, B0, 1);
  C0 = vfmaq_laneq_f32(C0, A2, B0, 2);
  C0 = vfmaq_laneq_f32(C0, A3, B0, 3);
  vst1q_f32(C, C0);

  B1 = vld1q_f32(B+4);
  C1 = vfmaq_laneq_f32(C1, A0, B1, 0);
  C1 = vfmaq_laneq_f32(C1, A1, B1, 1);
  C1 = vfmaq_laneq_f32(C1, A2, B1, 2);
  C1 = vfmaq_laneq_f32(C1, A3, B1, 3);
  vst1q_f32(C+4, C1);

  B2 = vld1q_f32(B+8);
  C2 = vfmaq_laneq_f32(C2, A0, B2, 0);
  C2 = vfmaq_laneq_f32(C2, A1, B2, 1);
  C2 = vfmaq_laneq_f32(C2, A2, B2, 2);
  C2 = vfmaq_laneq_f32(C2, A3, B2, 3);
  vst1q_f32(C+8, C2);

  B3 = vld1q_f32(B+12);
  C3 = vfmaq_laneq_f32(C3, A0, B3, 0);
  C3 = vfmaq_laneq_f32(C3, A1, B3, 1);
  C3 = vfmaq_laneq_f32(C3, A2, B3, 2);
  C3 = vfmaq_laneq_f32(C3, A3, B3, 3);
  vst1q_f32(C+12, C3);
}


void print_time(int time_ms){
    printf("neon,nSize=%d,time=",nSize);
    int time_s=time_ms/1000;
    if(time_ms<1000){
        printf("%dms\n",time_ms);
    }else if(time_s<60){
        printf("%ds\n",time_s);
    }else{
        int time_m_part=time_s/60;
        int time_s_part=time_s%60;
        printf("%dm%ds\n",time_m_part,time_s_part);
    }
}


int main(int argc, char ** argv){
    size_t total_element=nSize*nSize;

    static float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    static float * p2;
    p2=(float*)malloc(total_element*sizeof(float));
    static float * p3;
    p3=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
        p3[i]=0;
  }

    clock_t start,end;
    start = clock();
    matmul_neon(p1, p2, p3);
    end = clock();
    int time_ms=(int)(double)(end-start)/1000;
    print_time(time_ms);

    free(p1);
    free(p2);
```

```
    free(p3);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % gcc final.c -o3
(base) hyx13701490089@huayuxiaodeMacBook-Pro neon % ./3
neon,nSize=16,time=0ms
neon,nSize=128,time=0ms
neon,nSize=1000,time=0ms
neon,nSize=8000,time=0ms
```

我就实话和老师说吧，这个neon的程序是有问题的，按照程序的意思，是只算了矩阵数乘，所以结果会全0。它应该是这样的：A,B,C分别对应前矩阵的一行，后矩阵的一列，输出结果。这三者只是大矩阵的一个部分，但我把它写成了全部，这就导致了在计算时，只算了两个一维矩阵的乘法，而非二维矩阵的乘法。但我来不及写完了，所以有些遗憾。

(4) openmp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <arm_neon.h>
#include <omp.h>
//这里'omp.h'头文件找不到，这个问题暂时还没有解决
#define nSize 16


void matmul_openmp(const float *p1, const float * p2){
    static float each[nSize*nSize];
    for(size_t r=0;r<nSize;r++){
        for(size_t c=0;c<nSize;c++){
            each[r*nSize+c]=0;
            for(size_t i=0;i<nSize;i++){
                each[r*nSize+c]+=p1[r*nSize+i]*p2[c+i*nSize];
            }
        }
    }
}


void print_time(int time_ms){
    printf("openmp,nSize=%d,time=",nSize);
    int time_s=time_ms/1000;
    if(time_ms<1000){
        printf("%dms\n",time_ms);
    }else if(time_s<60){
        printf("%ds\n",time_s);
    }else{
        int time_m_part=time_s/60;
        int time_s_part=time_s%60;
        printf("%dm%ds\n",time_m_part,time_s_part);
    }
}


int main(int argc, char ** argv){
    size_t total_element=nSize*nSize;

    static float * p1;
    p1=(float*)malloc(total_element*sizeof(float));
    static float * p2;
    p2=(float*)malloc(total_element*sizeof(float));

    for(int i=0;i<total_element;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
  }

    clock_t start,end;
    start = clock();
    #pragma omp parallel for
    matmul_openmp(p1, p2);
    end = clock();
    int time_ms=(int)(double)(end-start)/1000;
    print_time(time_ms);

    free(p1);
```

```
    free(p2);

    return 0;
}
```

```
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % gcc final.c -o3
(base) hyx13701490089@huayuxiaodeMacBook-Pro openmp % ./3
openmp,nSize=16,time=0ms
openmp,nSize=128,time=7ms
openmp,nSize=1000,time=3s
openmp,nSize=8000,time=53m4s
```

以下表格是所有测试样例的数据汇总：

	16	128	1K	8K	64K
plain	0ms	7ms	3s	53m4s	inf
unloop	0ms	25ms	6s	35m47s	inf
neon	0ms	9ms	5ms	8ms	4ms
openmp	0ms	7ms	3s	53m4s	inf

4. Compare your implementation with the matrix multiplication in OpenBLAS (https://www.openblas.net/). The results by your implementation should be the same or ver similar with those by OpenBLAS. Can your implementation be as fast as OpenBLAS?

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include 'cblas.h'
#define nSize 16


static void matmul_openblas(const float * p1, const float * p2, const float * p3) {
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, nSize, nSize, nSize, 1.0f, *p1, nSize, *p2, nSize, 1.0f, *p3, nSize);
}


void print_time(int time_ms){
    printf("plain,nSize=%d,time=",nSize);
    int time_s=time_ms/1000;
    if(time_ms<1000){
        printf("%dms\n",time_ms);
    }else if(time_s<60){
        printf("%ds\n",time_s);
    }else{
        int time_m_part=time_s/60;
        int time_s_part=time_s%60;
        printf("%dm%ds\n",time_m_part,time_s_part);
    }
}


int main(int argc, const char * argv[]) {
    size_t total=nSize*nSize;

    static float * p1;
    p1=(float*)malloc(total*sizeof(float));
    static float * p2;
    p2=(float*)malloc(total*sizeof(float));
    static float * p3;
    p3=(float*)malloc(total*sizeof(float));

    for(int i=0;i<total;i++){
        p1[i]=(double)rand()/RAND_MAX;
        p2[i]=(double)rand()/RAND_MAX;
        p3[i]=0;
  }
```

```
    clock_t start,end;
    start = clock();
    matmul_openblas(p1, p2, p3);
    end = clock();
    int time_ms=(int)(end-start)/1000;

    print_time(time_ms);

    free(p1);
    free(p2);
    free(p3);

    return 0;
}
```

这个程序有一些bug，因为参考程序是C++，转换过来有点问题。’cblas.h’是我从官网上下载的头文件，太长了这里不展示了。

5. Test your program on X86 and ARM platforms, and describe the differences.

我有两台笔记本，MacBook Pro M1 2020（arm架构）和Honor Matebook X14（x86架构），不过win版vscode的环境暂时还没配置好，所以暂时没能完成这部分。后期我会补上，虽然已经不算分了。