步骤一：运行报错修改：Error setting memory limit: Argument list too long

[root@test100 workshop0311-main]# ./workshop Error setting memory limit: Argument list too long

Initialize 函数部分源代码：

```
snprintf(mlim_str, sizeof(mlim_str), "%dM", MEMORY_LIMIT_MB);
if (write(mlim_fd, mlim_str, sizeof(mlim_str)) == -1) {
    perror("Error setting memory limit");
    close(mlim_fd);
    exit(EXIT_FAILURE);
}
```

修改后：

```
snprintf(mlim_str, sizeof(mlim_str), "%d", MEMORY_LIMIT_MB * 1024 * 1024);
if (write(mlim_fd, mlim_str, sizeof(mlim_str)) == -1) {
    perror("Error setting memory limit");
    close(mlim_fd);
    return(EXIT_FAILURE);
}
close(mlim_fd);
```

步骤二：目前运行结果，7-11s 范围内

```
[root@test100 workshop0311-main]# ./workshop
Matrix multiply iteration 1: cost 7.453 seconds
Matrix multiply iteration 2: cost 7.266 seconds
Matrix multiply iteration 3: cost 10.560 seconds
Matrix multiply iteration 4: cost 7.665 seconds
Matrix multiply iteration 5: cost 7.902 seconds
Matrix multiply iteration 6: cost 8.398 seconds
Matrix multiply iteration 7: cost 11.036 seconds
Matrix multiply iteration 8: cost 7.680 seconds
Matrix multiply iteration 9: cost 9.245 seconds
Matrix multiply iteration 10: cost 8.426 seconds
Matrix multiply iteration 11: cost 9.720 seconds
Matrix multiply iteration 12: cost 7.160 seconds
Matrix multiply iteration 13: cost 9.506 seconds
Matrix multiply iteration 14: cost 9.240 seconds
```

步骤三：优化 C = A * B 矩阵乘法实现

（1）块矩阵优化（减少 cache miss）

（2）SIMD（AVX2）优化（提高计算吞吐量）

（3）OpenMP 并行（利用 CPU 多核加速）

mul.c 优化前：

```
#include "mul.h"

void mul(int msize, int tidx, int numt, Vector *vec, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{
    int i,j,k;

    for (i = tidx; i < msize; i = i + numt) {
        for (j = 0; j < msize; j++) {
            for (k = 0; k < msize; k++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
            vector_append(vec, c[i][j]);
        }
    }
}
```

mul.c 优化后：

```
#include "mul.h"
#include <omp.h>
#include <immintrin.h>
#include <stdlib.h>

#define BLOCK_SIZE 64 // 设定 Block Matrix 计算块大小

void mul(int msize, int tidx, int numt, Vector *vec, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{
    int i, j, k, ii, jj, kk;

    omp_set_num_threads(4);

    #pragma omp parallel for private(ii, jj, kk, i, j, k) shared(a, b, c) schedule(dynamic)
    for (ii = 0; ii < msize; ii += BLOCK_SIZE) {
        for (jj = 0; jj < msize; jj += BLOCK_SIZE) {
            for (kk = 0; kk < msize; kk += BLOCK_SIZE) {
                for (i = ii; i < ii + BLOCK_SIZE && i < msize; i++) {
                    for (j = jj; j < jj + BLOCK_SIZE && j < msize; j++) {
                        __m256d sum = _mm256_setzero_pd();
                        for (k = kk; k < kk + BLOCK_SIZE && k < msize; k += 4) {
                            __m256d va = _mm256_load_pd(&a[i][k]);
                            __m256d vb = _mm256_load_pd(&b[k][j]);
                            sum = _mm256_fmadd_pd(va, vb, sum);
                        }
```

```
                    double sum_arr[4];
                    _mm256_store_pd(sum_arr, sum);
                    c[i][j] += sum_arr[0] + sum_arr[1] + sum_arr[2] + sum_arr[3];
                }
            }
        }
    }
}
```

步骤四：优化后的运行结果

```
[root@test100 workshop0311-main]# ./workshop
Matrix multiply iteration 1: cost 4.546 seconds
Matrix multiply iteration 2: cost 4.194 seconds
Matrix multiply iteration 3: cost 4.444 seconds
Matrix multiply iteration 4: cost 4.047 seconds
Matrix multiply iteration 5: cost 4.466 seconds
Matrix multiply iteration 6: cost 4.178 seconds
Matrix multiply iteration 7: cost 4.560 seconds
Matrix multiply iteration 8: cost 4.240 seconds
Matrix multiply iteration 9: cost 3.695 seconds
Matrix multiply iteration 10: cost 4.594 seconds
Matrix multiply iteration 11: cost 4.503 seconds
Matrix multiply iteration 12: cost 5.260 seconds
Matrix multiply iteration 13: cost 4.969 seconds
Matrix multiply iteration 14: cost 5.717 seconds
Matrix multiply iteration 15: cost 6.097 seconds
Matrix multiply iteration 16: cost 5.049 seconds
Matrix multiply iteration 17: cost 4.995 seconds
Matrix multiply iteration 18: cost 5.891 seconds
Matrix multiply iteration 19: cost 6.070 seconds
Matrix multiply iteration 20: cost 5.487 seconds
Matrix multiply iteration 21: cost 5.106 seconds
Matrix multiply iteration 22: cost 4.951 seconds
Matrix multiply iteration 23: cost 5.167 seconds
Matrix multiply iteration 24: cost 5.725 seconds
Matrix multiply iteration 25: cost 5.977 seconds
Matrix multiply iteration 26: cost 5.887 seconds
Matrix multiply iteration 27: cost 5.675 seconds
Matrix multiply iteration 28: cost 5.251 seconds
Matrix multiply iteration 29: cost 4.993 seconds
Matrix multiply iteration 30: cost 5.926 seconds
Matrix multiply iteration 31: cost 5.664 seconds
Matrix multiply iteration 32: cost 4.995 seconds
```

Matrix multiply iteration 33: cost 5.638 seconds
Matrix multiply iteration 34: cost 5.420 seconds
Matrix multiply iteration 35: cost 6.161 seconds
Matrix multiply iteration 36: cost 5.246 seconds
Matrix multiply iteration 37: cost 5.320 seconds
Matrix multiply iteration 38: cost 5.471 seconds
Matrix multiply iteration 39: cost 5.700 seconds
Matrix multiply iteration 40: cost 4.985 seconds
Matrix multiply iteration 41: cost 4.955 seconds
Matrix multiply iteration 42: cost 4.944 seconds
Matrix multiply iteration 43: cost 5.325 seconds
Matrix multiply iteration 44: cost 5.322 seconds
速度后面开始变慢，可能是由于 CPU 长时间高负载运行，导致降频

步骤五：结果总结
原始运行时间：平均 9.1 秒（7.16～11.04 秒）
优化后运行时间：平均 4.9 秒（3.7～6.1 秒）

速度提高了约 1.86 倍
计算性能提升 46.15%