中山大学计算机院本科生实验报告

（2024 学年春季学期）

课程名称：并行程序设计	批改人:
实验：4	专业（方向）：计算机科学与技术
学号：22336226	姓名：王泓沣
Email：wanghf59@mail2.sysu.edu.cn	完成日期：2025/4/22

1 实验目的

使用 OpenMP 实现并行通用矩阵乘法，并通过实验分析不同进程数量、矩阵规模、调度机制时该实现的性能。

输入： m,n,k 三个整数，每个整数的取值范围均为 [128, 2048]

问题描述： 随机生成 $m \times n$ 的矩阵 A 及 $n \times k$ 的矩阵 B，并对这两个矩阵进行矩阵乘法运算，得到矩阵 C .

输出： A,B,C 三个矩阵，及矩阵计算所消耗的时间 t 。

要求： 使用 OpenMP 多线程实现并行矩阵乘法，设置不同线程数量（1-16）、矩阵规模（128-2048）、调度模式（默认、静态、动态调度），通过实验分析程序的并行性能。

2 实验过程和核心代码

使用 omp 库自带的三种调度模式：默认、静态、动态

```
static void multiply_omp(const double *A, const double *B, double *C, int
    m, int n, int k, int num_threads, const char *sched, int chunk)
{
    /* Configure threads and (optionally) schedule policy */
    omp_set_num_threads(num_threads);

    if (strcmp(sched, "default") != 0) {
        omp_sched_t stype = (strcmp(sched, "static")  == 0) ?
            omp_sched_static : (strcmp(sched, "dynamic") == 0) ?
            omp_sched_dynamic : omp_sched_guided; /* fallback */
        omp_set_schedule(stype, chunk);
    }

    /* Parallelised i-loop; inner loops are private per thread */
```

```
    #pragma omp parallel for schedule(runtime)
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < k; ++j) {
            double sum = 0.0;
            for (int p = 0; p < n; ++p)
                sum += A[i * (long long)n + p] * B[p * (long long)k + j];
            C[i * (long long)k + j] = sum;
        }
    }
}
```

默认模式下 OpenMP 将所有迭代预先分为 chunk=num_iterations/num_threads 块，静态模式下将所有迭代预先分为指定 chunk 个块，然后轮询地分给各个线程，动态模式下将迭代同样按 chunk 划分，但直到线程空闲后才请求下一个块。

3 实验结果

表 1: 矩阵乘法 $C = A \times B$ 在不同矩阵规模、线程数与 OpenMP 调度策略下的执行时间（秒）

规模 ($m = n = k$)	线程数	default	static	dynamic
128	1	0.002795	0.002096	0.002036
	2	0.001234	0.001094	0.000976
	4	0.001017	0.000544	0.000866
	8	0.000477	0.000545	0.000704
	16	0.000482	0.000570	0.000648
256	1	0.016911	0.013508	0.012773
	2	0.007665	0.006490	0.006714
	4	0.005063	0.004217	0.003626
	8	0.002709	0.002689	0.002958
	16	0.002688	0.002684	0.002699
512	1	0.120386	0.120066	0.119333
	2	0.062146	0.062250	0.063300
	4	0.038561	0.039039	0.038420
	8	0.026171	0.027528	0.025571
	16	0.025216	0.025294	0.025823
1024	1	1.020314	1.071334	1.082719
	2	0.538968	0.560008	0.553239
	4	0.324746	0.320664	0.309923
	8	0.277214	0.276154	0.283870
	16	0.285287	0.277281	0.274888
2048	1	17.643392	17.827585	17.785308
	2	10.907776	11.097452	10.916473
	4	11.456329	11.453770	11.314458
	8	7.951397	7.969373	8.056342
	16	7.960207	7.948762	7.965987

4 实验感想

omp 库和 IntelMKL 库在具体使用方式上有些类似，但配置较为简单。