به نام خدا

دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

برنامه‌نویسی چندهسته‌ای

گزارش کار آزمایش ۴

سید محمد حجازی حسینی

۹۷۳۳۰۲۰

بهار ۱۴۰۱

تست‌ها بر روی کامپیوتر شخصی با 4 thread گرفته شده است.

سریال)

n	time
10 ^ 6	0.005000
10 ^ 7	0.044000
10 ^ 8	0.432000
10 ^ 8 x 2	0.867000

موازی روش ۱)

کد مربوط به این روش:

```
void prefix_sum(int *a, size_t n) {
    int nthreads = NUM_THREADS;
    int ithread;
    int sum;
    int *suma = (int*)malloc(sizeof(int) * (nthreads + 1));;

    #pragma omp parallel private(sum, ithread)
    {
        ithread = omp_get_thread_num();
        sum = 0;

        #pragma omp for nowait
        for (int i = 0; i < n; i++)
        {
            sum += a[i];
            a[i] = sum;
        }
        suma[ithread] = sum;

        #pragma omp barrier

        int offset = 0;
        for(int i = 0; i < ithread; i++)
            offset += suma[i];

        #pragma omp for schedule(static)
        for (int i = 0; i < n; i++)
            a[i] += offset;
    }
}
```

ابتدا آرایه را به قسمت‌های مساوی تقسیم کرده و جمع را می‌سپاریم به هر thread. سپس یک barrier ایجاد می‌کنیم تا همه‌ی جمع‌های محلی حساب شوند. سپس هر thread باید مقادیر آخرین درایه بخش‌های قبلی خود را جمع کند و به عنوان offset به هر خانه‌ی خود اضافه کند که این کار را هم parallel انجام می‌دهیم. در نتیجه offset هر بخش هم جمع شده و آرایه نهایی بدست می‌آید.

n	time
10 ^ 6	0.004000
10 ^ 7	0.031000
10 ^ 8	0.317000
10 ^ 8 x 2	0.609000

n	speedup
10 ^ 6	1.25
10 ^ 7	1.41
10 ^ 8	1.36
10 ^ 8 x 2	1.42

موازی روش ۲)

کد مربوط به این روش:

```
void prefix_sum(int *a, size_t n) {
    int step = (int) log2(n);
    int neighbour;
    #pragma omp parallel
    {
        for (int i = 0; i <= step; i++)
        {
            neighbour = (int) pow(2, i);
            #pragma omp single
            for (int j = n; j >= neighbour; j--)
            {
                #pragma omp task
                {
                    a[j] = a[j] + a[j - neighbour];
                }
            }
        }
    }
}
```

با یک thread همه‌ی task ها را می‌سازیم. برای جمع زدن از آخر شروع می‌کنیم که بتوانیم in-place جمع‌ها را انجام دهیم. خود thread ای که task ها را می‌سازد، پس از ساخت آن‌ها می‌تواند مقداری از آن‌ها را اجرا کند در نتیجه از همه thread ها به خوبی استفاده می‌شود.

n	time
10 ^ 6	0.825883
10 ^ 7	1.284498
10 ^ 8	2.934161
10 ^ 8 x 2	5.787013

n	speedup
10 ^ 6	0.0006
10 ^ 7	0.0342
10 ^ 8	0.1472
10 ^ 8 x 2	0.1498

این روش برای اجرا روی CPU مناسب نیست زیرا task های بسیار زیادی تولید شده و overhead زیادی دارد. اما اگر روی GPU اجرا کنیم از آنجایی که همه‌ی task ها مانند هم و سبک هستند، می‌توانیم task های جمع و ذخیره را به راحتی و با سرعت بسیار زیادی انجام دهیم.