محاسبه prefix sum

۱-۱ کد سریال

کد سریال را اجرا می‌کنیم و میانگین زمان چند بار اجرای آن را در جدول ۱ ثبت می‌کنیم.

جدول ۱: نتایج کد سریال

ورودی (N)	میانگین زمان اجرا (ثانیه)
10^6	۰/۰۰۲۳۹۸
10^7	۰/۰۲۳۱۲۳
10^8	۰/۲۳۷۶۰۷
10^9	۲/۴۷۱۲۰۵

۱-۲ موازی‌سازی: روش اول

پس از پیاده‌سازی کد سریال، به سراغ موازی‌سازی آن و گرفتن تسریع می‌رویم. در این روش با شکستن آرایه، محاسبه prefix هر بخش را به آن می‌سپاریم. سپس برای ترکیب این زیربخش‌ها باید **جمع تمام عناصر قبلی یک زیربخش** (یعنی **عنصر آخر زیربخش قبلی**) به تمام اعداد آن زیربخش اضافه شود.

در اینجا نیز پس از محاسبه بازه شروع و پایان هر نخ (محدوده زیربخش‌ها) هر نخ prefix sum آن زیربخش را محاسبه می‌کند. پس از اتمام کار تمام نخ‌ها (با گذاشتن یک barrier)، یک prefix sum برای عناصر آخر هر زیربخش توسط یکی از نخ‌ها محاسبه می‌کنیم (آرایه last_sums). در نهایت نیز همه نخ‌ها (به جز نخ شماره صفر) عنصر مربوط به خود از آرایه last_sums را با تمام عناصر زیربخش خود جمع می‌کند.

در نهایت با میانگین گرفتن زمان چند اجرا، جدول ۲ را پر می‌کنیم. همانطور که می‌بینیم کمی موفق به گرفتن تسریع با این روش شده‌ایم.

پیاده‌سازی این الگوریتم در فایل **Lab_4_m1.c** آمده است.

جدول ۲: نتایج روش اول

تعداد نخ‌ها	میانگین زمان اجرا (ثانیه)				تسریع
	10^6	10^7	10^8	10^9	
۲	۰/۰۰۲۴۱۵	۰/۰۱۶۴۶۹	۰/۱۷۱۶۸۸	۱/۷۸۹۱۶۶	۱/۲۹
۴	۰/۰۰۲۲۹۳	۰/۰۱۳۷۱۷	۰/۱۵۱۸۹۵	۱/۴۵۲۱۲۴	۱/۴۹
۸	۰/۰۰۱۳۲۵	۰/۰۱۵۵۶۴	۰/۱۵۹۶۴۴	۱/۴۹۸۱۲۵	۱/۶۰

۳-۱ موازی‌سازی: روش دوم

ابتدا الگوریتم دوم معرفی شده را در حالت سریال می‌نویسیم. در این الگوریتم ابتدا نتایج میانی (نتایج taskها) را در آرایه partial_sum می‌ریزیم و پس از کامل شدن محاسبات این آرایه برای آن مرحله، نتایج میانی را داخل آرایه اصلی (آرایه a) برمی‌گردانیم. در نهایت نیز stride را ۲ برابر کرده و مراحل را دوباره تکرار می‌کنیم. این روند را تا زمانی که stride از طول آرایه کوچک‌تر است ادامه می‌دهیم.

پس از نوشتن و آزمودن کد سریال، با استفاده از راهنمای omp for، دو حلقه میانی را موازی می‌کنیم. سپس با میانگین گرفتن زمان چند بار اجرا، جدول ۳ را پر می‌کنیم.

پیاده‌سازی این الگوریتم در فایل **Lab_4_m2.c** آمده است.

جدول ۳: نتایج روش دوم

تعداد نخ‌ها	میانگین زمان اجرا (ثانیه)				تسریع
	10^6	10^7	10^8	10^9	
۲	۰/۰۱۳۵۷۸	۰/۲۵۴۵۴۱	۲/۹۱۷۳۳۴	۳۴/۲۶۵۰۹۱	۰/۱۰
۴	۰/۰۰۹۴۴۷	۰/۲۴۳۵۷۷	۲/۸۰۸۰۶۰	۳۲/۲۵۶۳۲۸	۰/۱۲
۸	۰/۰۰۸۴۳۳	۰/۲۴۸۶۰۱	۲/۷۹۷۱۴۲	۳۱/۹۱۹۳۱۶	۰/۱۳

همانطور که می‌بینیم، این الگوریتم بسیار کندتر از روش اول و حالت سریال است. دلیل این اتفاق، ساخت تسک‌های بسیار زیاد و بسیار کوچک در الگوریتم است. این اتفاق باعث می‌شود تا در اجرا روی CPU، سربار موازی‌سازی بسیار زیاد شود و موازی‌سازی آن به صرفه نباشد.

در سیستم‌های SIMD مانند GPUها، از آنجا که تعداد هسته‌های بسیار بیشتری داریم و از آنجا که تمامی تسک‌های یک مرحله مشابه هم هستند و تنها روی داده‌های مختلف عمل می‌کنند (درست مانند عملکرد SIMD)، اجرای این الگوریتم روی یک GPU می‌تواند بسیار سریع‌تر باشد و حتی زمان اجرایی بهتر از روش

اول و حالت سریال داشته باشد.