به نام خدا

گزارشکار آزمایش چهارم (prefix sum)

برنامه‌نویسی چندهسته‌ای

زهرا لطیفی 9923069

آیدا احمدی پارسا 9923003

هدف از انجام این آزمایش پیاده‌سازی عملیات prefix sum به دو روش موازی‌سازی می‌باشد.

روش اول به این صورت است که آرایه ورودی با سایز n را به چند بخش مختلف (به تعداد نخ‌ها) تقسیم می‌کنیم. فرض می‌کنیم تعداد نخ‌ها num_thread می‌باشد. اندازه هر زیرآرایه برابر خواهد بود با n/num_threads . سپس انجام عملیات prefix sum برای هر زیرآرایه را به یک نخ می‌سپاریم. درنهایت پس از محاسبه prefix sum هر زیرآرایه باید آخرین درایه از زیرآرایه‌های قبلی را با درایه‌های چانک بعدی جمع کنیم. به زبان ساده‌تر برای هر چانک دیتا باید local prefix sum حساب کنیم و نهایتا درایه‌های هر چانک را با استفاده از روش ذکر شده به‌روزرسانی کنیم.

حال به توضیح جزیی کد ارائه شده برای روش اول موازی‌سازی، می‌پردازیم.

```
void parallel_prefix_sum(int *a, size_t n) {

	int i , j;
	int *last_sums, *starts, *ends;

#pragma omp parallel num_threads(4)
	{
		int tid = omp_get_thread_num();
		int num_threads = omp_get_num_threads();
#pragma omp single
		{
			last_sums = (int *)malloc(num_threads * sizeof(int));
			starts = (int *)malloc(num_threads * sizeof(int));
			ends = (int *)malloc(num_threads * sizeof(int));
		}

		int chunk_size = n / num_threads;
		starts[tid] = tid * chunk_size;
		ends[tid] = starts[tid] + chunk_size;

		// Local prefix sum over subarray
		for (int i = starts[tid] + 1; i < ends[tid]; i++) {
			a[i] += a[i - 1];
		}
		#pragma omp barrier


		#pragma omp single
		{
			last_sums[0] = 0;
			for (int i = 1; i < num_threads; i++) {
				last_sums[i] = a[starts[i] - 1] + last_sums[i - 1];
			}
		}
```

```
            if (tid != 0) {
                  for (int i = starts[tid]; i < ends[tid]; i++) {
                        a[i] += last_sums[tid];
                  }
            }     }
}
```

بخش 1) در این قسمت ابتدا سایز چانک‌ها را بر اساس تعداد نخ‌ها مشخص می‌کنیم. سپس نقطه شروع و پایان هر زیرآرایه را نیز تعیین می‌کنیم. به این صورت که نقطه شروع برابر خواهد بود با شماره نخ ضرب در سایز چانک‌ها و نقطه پایان برابر است با نقطه شروع به اضافه سایز چانک.

بخش 2) در ابتدای این قسمت یک omp parallel قرار داده‌ایم سپس prefix sum هر نخ را محاسبه می‌کنیم. نکته قابل توجه این است که پیش از رفتن به بخش بعدی باید یک barrier قرار دهیم تا تمام نخ‌ها کار خود را به پایان برسانند و پس از آن مقادیر چانک‌های دوم تا آخر را به‌روزرسانی کنیم.

بخش 3) در این بخش یک آرایه جدید به نام last_sums می‌سازیم. هر کدام از درایه‌های این آرایه مقادیر مختلفی دارد که هر یک باید با مقادیر یکی از چانک‌های دوم تا آخر جمع شود. هر درایه به این صورت محاسبه می‌شود که آخرین درایه چانک قبلی را با مقدار قبلی last_sum جمع می‌کنیم.

بخش 4) در بخش نهایی باید مقادیر هر چانک را با مقدار last_sum مربوط به همان چانک، جمع کنیم تنها در صورتی که در چانک اول داده قرار نداشته باشیم.

سوالات دستور کار:

علت جمع کردن درایه آخر چانک قبلی با چانک‌های بعدی چیست؟

درواقع درایه آخر هر چانک مجموع تمام درایه‌های پیشین است درنتیجه برای تاثیر دادن چانک‌های قبلی روی سایر داده‌ها کافی است که عدد آخر را با بقیه داده‌ها جمع کنیم.

روش دوم

کد ارائه شده برای این بخش به شکل زیر است:

```
void hillis_prefix_sum(int *a, size_t n) {
```

```
	int stride, i;
	int *partial_sum = malloc(n * sizeof(int));

	for (stride = 1; stride < n; stride *= 2) {
#pragma omp parallel num_threads(4) private(i)
		{
			int stop = n - stride;

#pragma omp for
			for (i = 0; i < stop; i++) {
				partial_sum[i + stride] = a[i + stride] + a[i];
			}

#pragma omp for
			for (i = stride; i < n; i++) {
				a[i] = partial_sum[i];
			}
		}
	}
	free(partial_sum);
}
```

در این بخش متغیری به نام stride تعریف می‌کنیم که هر بار ضرب در 2 شده و به‌روزرسانی می‌شود. تعداد taskهای انجام شده در هر مرحله به تعداد $n - stride$ است. سپس داده‌ها را با فاصله stride بایکدیگر جمع می‌کنیم. این عملیات باید موازی انجام شود.

نتایج به این صورت که زمان اجرای روش دوم کندتر از روش اول و حتی کندتر از حالت سریال است.

سوالات دستور کار:

دلیل این اتفاق چیست؟

همواره یکی از موضوعات مهم در موازی‌سازی تریدآف مطرح شده میان سربار موازی‌سازی و بهینه‌سازی اجرای برنامه می‌باشد. در این روش تعداد تسک‌های ایجاد شده بسیار زیاد است درنتیجه برای cpu سربار تولید می‌کند. با توجه به اینکه GPU یک نوع سیستم SIMD است و تعداد هسته‌های بیشتری دارد این عملیات روی GPU بهتر انجام می‌شوند. چرا که درواقع یک دستور مشخص روی چند دیتا در حال اجراست.

نتایج و زمان‌بندی

2 نخ

Size	1MB÷4B	10MB÷4B	100MB÷4B	1GB÷4B	Speed up
Serial	0.000332	0.003162	0.033039	0.351406	
Method1	0.000134	0.001109	0.009625	0.112602	2.94
Method2	0.004648	0.045616	0.635245	7.722143	0.0596

4 نخ

Size	1MB÷4B	10MB÷4B	100MB÷4B	1GB÷4B	Speed up
Serial	0.000332	0.003162	0.033039	0.351406	
Method1	0.000130	0.001129	0.011359	0.116658	2.81
Method2	0.003453	0.048235	0.623747	7.581853	0.0652

8 نخ

Size	1MB÷4B	10MB÷4B	100MB÷4B	1GB÷4B	Speed up
Serial	0.000332	0.003162	0.033039	0.351406	
Method1	0.000129	0.001322	0.009769	0.107762	2.902
Method2	0.005362	0.044144	0.600049	7.149038	0.23