آزمایش چهارم درس برنامه نویسی چند هسته ای

موازی سازی prefix sum

آرش حریرپوش
9731505

آرش حریرپوش
9731505

❖ روش اول

ابتدا آرایه x را بین نخ ها به صورت static تقسیم کنید (هر نخ n / 1) آرایه را پردازش می کند. هر نخ عملیات prefix sum را به صورت مستقل بر روی زیرآرایه خود انجام می دهد. به عنوان مثال با دو نخ داریم:

x:	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
y:	۱	۳	۶	۱۰	۱۵	۶	۱۳	۲۱	۳۰	۴۰

نخ اول نخ دوم

- پس از انجام کار هر نخ باید مقدار خانه آخر هر زیر آرایه را با تمامی خانه های زیرآرایه های بعد از آن جمع کنیم. مثلا مقدار خانه آخر زیرآرایه اول (15) باید با همه خانه های زیرآرایه دوم جمع شود. چرا؟

برای اینکه برای محاسبه prefix sum هر خانه باید تمام خانه های قبل از آن را جمع کنیم، این در حالی است که زمانی که آرایه را به قسمت هایی تقسیم کنیم و هر کدام از این قسمت ها را یک نخ انجام دهد، هر نخ به اطلاعات بخش قبلی که برابر درایه های قبلی این آرایه برای این درایه می باشد دسترسی ندارد؛ به همین دلیل prefix sum را مطابق درایه های آن بخش از آرایه که متعلق به آن نخ می باشد انجام می دهد، با توجه به اینکه مقدار Prefix sum باید بر اساس آرایه اصلی صورت پذیرد بنابراین نیاز است تا درایه ها هر قسمت با مجموع درایه های قبلی بخش های اصلی نیز جمع شود، و مقدار آخرین خانه هر قسمت برابر مجموع درایه های آن قسمت می باشد؛ به همین دلیل باید درایه های آخر هر قسمت را با قسمت های بعدی نیز جمع کنیم. (در این مثال مقدار prefix sum برای نخ دوم بر اساس آن قسمتی از آرایه که متعلق به آن نخ بوده محاسبه شده، شامل مقادیر درایه های قسمت قبل نبوده این در حالی است که برای محاسبه prefix sum باید تمام درایه های قبلی با هم جمع شوند و مجموع درایه های قبلی در آخرین خانه محاسبه شده هر نخ موجود است به همین دلیل باید مقدار خانه آخر زیر آرایه اول را که برابر مجموع درایه های آن بخش می باشد با همه خانه های زیر آرایه دوم جمع کنیم.)

آرش حریرپوش
9731505

- اگر در مثال بالا تعداد نخ ها چهار شود. برای به دست آوردن مقدار نهایی یک راه حل این است که هر نخ مقدار آخرین خانه محاسبه شده توسط خود را با تمام خانه های پس از آن جمع کند. خروجی این راه حل را به ازای 1 گیگ اندازه حافظه و 4 نخ بررسی کنید. دلیل اشتباهات احتمالی چیست؟

یک گیگ برابر 2^{30} بیت می باشد، و طول هر مقدار int برابر 4 بیت می باشد، بنابراین این آرایه دارای $2^{28} = 268,435,456$ درایه از جنس int می باشد.
زمان اجرا بخش محاسبات prefix sum در حالت سری برابر با: 0.255917 ثانیه
و زمان اجرا بخش محاسبات prefix sum پس از موازی سازی برابر است با:
0.864568 ثانیه

میزان تسریع: $Speed\ up = \frac{0.255917}{0.864568} = 0.29600$

دلایل اشتباهات احتمالی به این صورت است که، ممکن است هنگامی که می خواهیم آخرین مقدار محاسبه شده یک بخش را در یک آرایه دیگر ذخیره کنیم، ممکن است برای برخی از قسمت ها این کار به اتمام نرسیده باشد و آن آرایه درست مقدار دهی نشود در نتیجه باعث اشتباه شدن خروجی می شود، اگر از آرایه دیگری استفاده نکنیم و مستقیم از درایه های خروجی قبلی استفاده کنیم به این دلیل که در حال به روز رسانی مقادیر درایه های آرایه هستیم زمانی که می خواهیم آخرین درایه یک قسمت را برداریم آن درایه به دلیل اینکه در مراحل قبلی با prefix sum قسمت قبل جمع شده دیگر برابر مقدار prefix sum آن بخش نیست بلکه برابر مقدار prefix sum آرایه تا آن قسمت می باشد، به همین دلیل درایه های آخر قسمت های اول چند بار با درایه های بعد از جمع می شوند که خود باعث اشتباه شدن مقدار می شود (به عبارت دیگر در هر مرحله که ما در حال جمع کردن آخرین درایه یک قسمت با تمام درایه های قسمت های بعد هستیم مقدار آخرین درایه قسمت های بعد را نیز تغییر می دهیم و چون این درایه ها نیز خود باید با تمام درایه های بعد از در قسمت های بعد جمع شوند این باعث می شود تا درایه های آخر قسمت های قبل چندین بار با خانه های بعدی جمع شوند که برای حل این مشکل از یک آرایه میانی استفاده می کنیم)، ممکن است هنگامی که یک نخ کارش تمام می شود وارد حلقه بعدی شود تا کار محاسبه prefix sum آن بخش را انجام دهد در حالی که مقادیر درایه آخر قسمت های قبلی هنوز محاسبه نشده است، به همین دلیل با قرار دادن یک مانع قبل از محاسبه prefix sum کلی صبر می کنیم تا همه نخ ها کارشان تمام شود و مقدار آخرین درایه را در آرایه متناظر قرار دهند و سپس وارد بخش بعدی شوند، و یکی از مشکلات دیگر این است که آخرین درایه این آرایه به طول یک گیگ برابر $\frac{n*(n+1)}{2} = \frac{2^{28}*(2^{28}+1)}{2}$ می باشد و به دلیل اینکه این مقدار و بسیاری از مقادیر قبل از آن در int قابل نمایش نمی باشند سر ریز رخ می دهد و باعث اشتباه شدن مقدار نمایش داده شده می شود؛ در نهایت کد محاسبه prefix sum این قسمت برابر است با:

```
void prefix_sum(int *a, size_t n) {
	double starttime, elapsedtime;
	int i;
	int prefix_sum_parts[NUMOFTHREAD];
	int part_size = n / NUMOFTHREAD;
	starttime = omp_get_wtime();
	omp_set_num_threads(NUMOFTHREAD);
#pragma omp parallel
	{
		int id_threads = omp_get_thread_num();
		int n_threads = omp_get_num_threads();

		for (i = 1; i < part_size; ++i) {
			a[id_threads * part_size + i] = a[id_threads * part_size + i] +
a[id_threads * part_size + i - 1];
		}

		prefix_sum_parts[id_threads] = a[(id_threads+1) * part_size - 1];

		#pragma omp barrier
		for (int i = 0; i < id_threads; i++) {
			for (int j = 0; j < part_size; j++) {
				a[id_threads * part_size + j] += prefix_sum_parts[i];
			}
		}
	}
	// get ending time and use it to determine elapsed time
	elapsedtime = omp_get_wtime() - starttime;
	// report elapsed time
	printf("Time Elapsed: %f Secs \n",
		elapsedtime);
}
```

و همچنین نتیجه اجرای کد بالا برای مقدار حافظه 1 گیگ برابر است با:

```
[-] Configuration: Release.
[-] Platform: x64
[-] OpenMP is on.
[-] OpenMP version: 201611
[-] Maximum threads: 4
OMP: Info #273: omp_get_nested routine deprecated, please use omp_get_max_active_levels instead.
[-] Nested Parallelism: On
==============================
[-] Please enter N: 268435456
Time Elapsed: 0.864568 Secs
```

در این بخش فرض شده که اندازه آرایه های ورودی مضربی از تعداد نخ های مورد استفاده در بخش موازی سازی شده می باشد.

- در روش قبل به ازای هر خانه از خروجی باید به تعداد بخش های قبل از آن روی خانه موردنظر عمل جمع صورت بگیرد. چطور می توان با انجام یک سری محاسبات میانی کم هزینه بر روی خانه های آخر زیر آرایه ها، به ازای هر زیر آرایه مقداری به دست آورد که اگر با خانه های آن زیر آرایه یکبار جمع شود خروجی نهایی مستقیما به دست آید. کد موازی این روش را بنویسید و زمان اجرای آن را به ازای ۱ گیگ اندازه حافظه با حالت سریال مقایسه کنید.

در این بخش پس از اینکه هر آرایه مقدار *prefix sum* قسمت خود را محاسبه کرد، پس از اینکه مقدار آخرین درایه هر قسمت در آرایه میانی ذخیره شد، بعد از مانع اولین نخی که از مانع عبور کند مقدار *prefix sum* آن آرایه میانی را محاسبه می کند و سپس بقیه نخ ها پشت مانع بعد منتظر می مانند تا کار نخ قبل اتمام یابد سپس همه نخ ها بجز نخ اول که مقدار نهایی را به درستی محاسبه کرده مقدار درایه های آرایه های متناظر خود را با مقدار درایه متناظر آرایه میانی جمع می کنند؛ در نهایت کد محاسبه prefix sum این قسمت برابر است با:

```
void prefix_sum(int *a, size_t n) {
	double starttime, elapsedtime;
	int i;
	int prefix_sum_parts[NUMOFTHREAD];
	int part_size = n / NUMOFTHREAD;
	starttime = omp_get_wtime();
	omp_set_num_threads(NUMOFTHREAD);
#pragma omp parallel
	{
		int id_threads = omp_get_thread_num();
		int n_threads = omp_get_num_threads();

		for (i = 1; i < part_size; ++i) {
			a[id_threads * part_size + i] = a[id_threads * part_size + i] +
a[id_threads * part_size + i - 1];
		}

		prefix_sum_parts[id_threads] = a[(id_threads + 1) * part_size - 1];

#pragma omp barrier
#pragma omp single
		for (int i = 1; i < NUMOFTHREAD; ++i) {
			prefix_sum_parts[i] +=  prefix_sum_parts[i - 1];
		}
#pragma omp barrier
		for (int j = 0; j < part_size; j++) {
			if (id_threads == 0) {
				continue;
			}
			a[id_threads * part_size + j] += prefix_sum_parts[id_threads-1];
		}
	}
	// get ending time and use it to determine elapsed time
	elapsedtime = omp_get_wtime() - starttime;
	// report elapsed time
	printf("Time Elapsed: %f Secs \n",
		elapsedtime);
}
```

و همچنین نتیجه اجرای کد بالا برای مقدار حافظه 1 گیگ برابر است با:

```
------------ Info -------------
[-] Configuration: Release.
[-] Platform: x64
[-] OpenMP is on.
[-] OpenMP version: 201611
[-] Maximum threads: 4
OMP: Info #273: omp_get_nested routine deprecated, please use omp_get_max_active_levels instead.
[-] Nested Parallelism: On
==============================
[-] Please enter N: 268435456
Time Elapsed: 0.668276 Secs
```

در نتیجه، زمان اجرا بخش محاسبات prefix sum پس از موازی سازی برابر است با:
0.668276 ثانیه
بنابراین میزان تسریع برابر است با:

$$Speed\ up = \frac{0.255917}{0.668276} = 0.38295$$

همانطور که مشاهده می کنید، زمان اجرای این برنامه با موازی سازی افزایش می یابد که این افزایش به دلیل استفاده از موانع و سربار های موازی سازی می باشد و در برخی از قسمت ها از همه نخ ها به صورت بهینه استفاده نمی شود، و با استفاده از موانع برای کنترل کردن بخش های موازی زمان زیادی از بین می رود، در این قسمت از دو مانع استفاده شده و در بخشی که *prefix sum* آرایه میانی محاسبه می شود در عمل برنامه به صورت سری این کار را انجام می دهد، بنابراین سه نخ باید منتظر یک نخ بمانند تا کار خود را انجام دهد، و در نهایت سربار موازی سازی، استفاده از موانع و سری شدن بخشی از برنامه باعث شده که این برنامه به صورت سری بهتر از موازی اجرا شود و زمان اجرای این برنامه در حالت موازی افزایش یابد.

- (امتیازی): اگر ایده ای دارید که سریع تر به جواب می رسد آن را پیاده سازی کنید.

مسئله را بدین صورت حل می کنیم که، هر خانه خروجی برابر حاصل جمع خانه های قبلی می باشد و برای هر خانه خروجی تمام خانه های قبل از آن را جمع می کنیم، به این دلیل که این کار زمان بر می باشد یک آرایه کمکی دیگر در نظر می گیریم که مشخص می کند که آیا *prefix sum* خانه قبلی حساب شده است یا خیر و در صورت 0 بودن یعنی حساب نشده و در صورت 1 بودن یعنی حساب شده است، و زمانی که به خانه ای می رسیم که *prefix sum* آن قبلا حساب شده دیگر مقدار خانه های قبلی را جمع نمی کنیم و مقدار بیت *valid* خانه متناظر با آن خانه آرایه که مقدار *prefix sum* آن محاسبه شده را یک می کنیم و سپس مقدار *prefix sum* خانه بعدی را محاسبه می کنیم؛ پس از اینکه این برنامه را به صورت سری پیاده سازی کردیم، حلقه مربوط به محاسبه *prefix sum* هر خانه را موازی می کنیم تا هر نخ مسئول محاسبه *prefix sum* تعدادی از خانه های خروجی شود، و زمانی مقدار *prefix sum* آن خانه را بدست آورده است که به خانه ای برسد که مقدار بیت *valid* متناظر با آن خانه یک باشد؛ بنابراین کد محاسبه prefix sum این قسمت برابر است با:

```
void prefix_sum(int *a, size_t n) {
	double starttime, elapsedtime;
	int i, steps = (int)ceil(log2(n));
	int* valid = (int*)malloc(n * sizeof(int));

	starttime = omp_get_wtime();
	valid[0] = 1;
	omp_set_num_threads(NUMOFTHREAD);
#pragma omp parallel
	{
	#pragma omp for nowait
	for (int i = 1; i < n; ++i) {
		valid[i] = 0;
	}
		#pragma omp for schedule(static, 128)
		for (int i = 0; i < n; i++) {
			for (int j = i; j > 0; j--)
			{
				if (valid[j] == 1)
				{
					valid[i] = 1;
					break;
				}
				a[i] += a[j - 1];
			}
			valid[i] = 1;
		}
	}
	// get ending time and use it to determine elapsed time
	elapsedtime = omp_get_wtime() - starttime;
	// report elapsed time
	printf("Time Elapsed: %f Secs \n",
		elapsedtime);
	// Free allocated memory
	free(valid);
}
```

و همچنین نتیجه اجرای کد بالا برای مقدار حافظه 1 گیگ برابر است با:

```
------------ Info -------------
[-] Configuration: Release.
[-] Platform: x64
[-] OpenMP is on.
[-] OpenMP version: 201611
[-] Maximum threads: 4
OMP: Info #273: omp_get_nested routine deprecated, please use omp_get_max_active_levels instead.
[-] Nested Parallelism: On
==============================
[-] Please enter N: 268435456
Time Elapsed: 1.161942 Secs
```

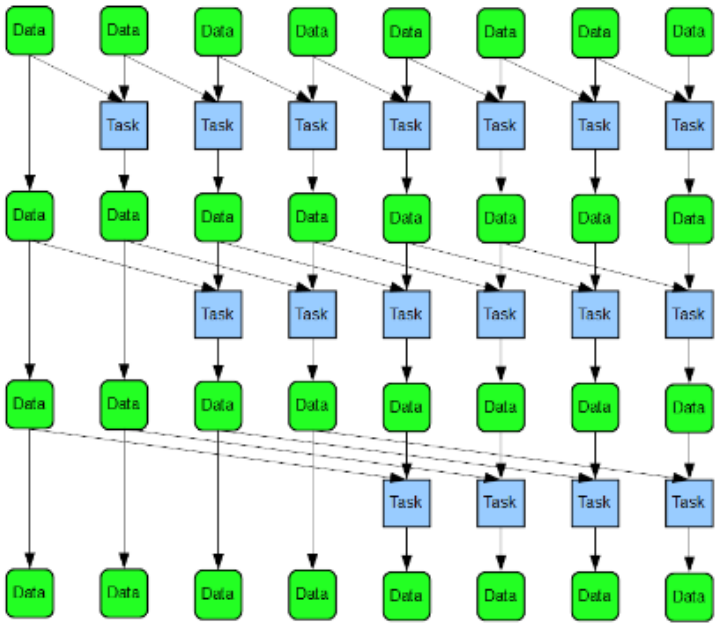
در نتیجه، زمان اجرا بخش محاسبات prefix sum پس از موازی سازی برابر است با:
1.161942 ثانیه
بنابراین میزان تسریع برابر است با:

$$Speed\ up = \frac{0.255917}{1.161942} = 0.22024$$

آرش حریرپوش
9731505

❖ روش دوم

یک الگوریتم برای محاسبه prefix scan الگوریتمی است به نام Hillis and Steele که در سال ۱۹۸۶ معرفی شده است. شکل زیر الگوی محاسبات آن را نشان میدهد:



در این شکل، آرایه ورودی دارای ۸ المان است و آرایه خروجی در پایین محاسبه شده است. هر مربع task یک جمع است.

- کد موازی این الگوریتم را بنویسید و زمان اجرای آن را به ازای ۱ گیگ اندازه حافظه با حالت سریال مقایسه کنید.

این کد به نوعی به صورت خط لوله عمل می کند، به این دلیل که در هر مرحله به پردازش مراحل قبل نیاز دارد، و برای موازی سازی آن باید تنها مراحلی که در هر بخش انجام می شود را به صورت موازی انجام داد، بنابراین برای پیاده سازی این روش به یک حلقه دو سطحه نیاز داریم که در سطح دوم خود دارای سه حلقه دیگر می شود، حلقه سطح اول تعداد تکرار هایی برابر $Log(NumOfArrayElements)$ (که برای اعدادی که در مبنای 2 نیستند نیاز است تا عدد را به بالا گرد کنیم) دارد و در هر مرحله $2^{numOfState}$ عنصر اول تغییر نمی کنند و $NumOfArrayElements - 2^{numOfState}$ تا عمل جمع بین عناصر دیگر رخ می دهد که این جمع ها را به صورت موازی انجام می دهیم، حال برای در داخل حلقه ای که مشخص کننده تعداد گام ها می باشد سه عمل باید صورت پذیرد که هر یک را با یک حلقه انجام می دهیم، ابتدا یک نخ باید درایه هایی که در آن مرحله ثابت هستند را در آرایه نهایی ذخیره کند، سپس در حلقه بعد عملیات های جمع بین درایه ها را انجام می دهیم، که در این مرحله به دلیل اینکه ترتیب عملیات ها مهم نیست تکرار حلقه را بین نخ ها تقسیم می کنیم، سپس پس از اتمام این مرحله توسط تمامی نخ ها یک نخ درایه های تغییر یافته را در آرایه اصلی وارد می کند و در همین مدت گام بعدی توسط بقیه نخ ها صورت می پذیرد، همانطور که مشاهده می شود این روش را به صورت خط لوله موازی کرده ایم ، به

این صورت که در هر گام ابتدا یک نخ شروع به ذخیره مقادیر ثابت از مرحله قبلی می کند و بقیه نخ ها له انجام محاسبات آن کام مشغول می شوند، سپس پس از اتمام کار تمامی نخ ها یک نخ اطلاعات تغییر یافته را در آرایه نهایی ذخیره می کند و باقی نخ ها وارد گام بعدی می شوند؛ در نهایت کد موازی سازی شده بخش محاسباتی prefix sum برابر است با:

```
void prefix_sum(int *a, size_t n) {
	double starttime, elapsedtime;
	int i, steps = (int) ceil(log2(n));
	int* copy = (int*)malloc(n * sizeof(int));

	starttime = omp_get_wtime();
	fill_array(copy, n);
	omp_set_num_threads(NUMOFTHREAD);
#pragma omp parallel
		{
		for (int s = 0; s < steps; s++) {

			int work_step = (int)pow(2, s);
			int work = work_step-1;

			#pragma omp single
			for (int w = 0; w < work_step; w++) {
				a[w] = copy[w];
			}
			#pragma omp for
			for (int w = 0; w < n - work_step; w++) {
				copy[w + work_step] += a[w];
			}

			#pragma omp single
			for (int w = 0; w < n - work_step; w++) {
				a[w + work_step] = copy[w + work_step];
			}
		}
	}
	// get ending time and use it to determine elapsed time
	elapsedtime = omp_get_wtime() - starttime;
	// report elapsed time
	printf("Time Elapsed: %f Secs \n",
			elapsedtime);
	// Free allocated memory
	free(copy);
}
```

و همچنین نتیجه اجرای کد بالا برای مقدار حافظه 1 گیگ برابر است با:

```
------------ Info -------------
[-] Configuration: Release.
[-] Platform: x64
[-] OpenMP is on.
[-] OpenMP version: 201611
[-] Maximum threads: 4
OMP: Info #273: omp_get_nested routine deprecated, please use omp_get_max_active_levels instead.
[-] Nested Parallelism: On
==============================
[-] Please enter N: 268435456
Time Elapsed: 12.594889 Secs
```

در نتیجه، زمان اجرا بخش محاسبات prefix sum پس از موازی سازی برابر است با:
12.594889 ثانیه
بنابراین میزان تسریع برابر است با:

$$Speed\ up = \frac{0.255917}{12.594889} = 0.02031$$

همانطور که مشاهده می کنید، زمان اجرای این برنامه به شدت بیشتر از زمان اجرای این برنامه به صورت سریال می باشد، که به این دلیل می باشد، سربار موازی سازی برای این قسمت زیاد می باشد و نسبت به حالت سریال کارهای بیشتری انجام می شود.

- الگوریتم دوم با اینکه کار موازی بیشتری تولید می کند ولی کندتر از الگوریتم اول و حتی الگوریتم سریال است . چرا؟

به این دلیل این الگوریتم از الگوریتم قبلی اول و سریال کند تر است که، برای رسیدن به مسئله مراحل بیشتر ی باید انجام دهیم و گام های این الگوریتم به یک دیگر مرتبط می باشد، بنابراین نمی توان گام ها را بین نخ های متعدد تقسیم کرد و تنها باید کار هایی که در هر گام انجام می شود را موازی کرد و تقریبا همانند خط لوله پیاده سازی می شود، و در نهایت به دلیل اینکه به یک مسئله memory bounded تبدیل می شود به نخ های زیادی نیاز داریم تا تاخیر حافظه را بپوشانیم و این در حالی است که در cpu به اندازه کافی نخ موازی نداریم.

توضیح دهید الگوریتم دوم در چه حالتی می تواند نسبت به الگوریتم اول مزیت داشته باشد (راهنمایی : این الگوریتم در GPU بسیار پر کاربرد است).

به دلیل اینکه این الگوریتم memory bounded می باشد بنابراین با افزایش تعداد نخ ها می توان سرعت مسئله را افزایش داد، و در حالتی که در پردازنده ای که تعداد نخ های موازی بیشتری را پشتیبانی کند می تواند سریع تر عمل کند (الگوریتم دوم زمانی که به اندازه درایه های آرایه پردازنده وجود داشته باشد خوب عمل می کند به همین دلیل در GPU بهتر عمل می کند).

(منبع: https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda)

آرش حریرپوش
9731505

عملیات prefix sum کاربرد فراوانی در حوزه های مختلف دارد. چند نمونه از آنها را ذکر کنید.

Prefix sum در برنامه های زیر کاربرد دارد:

- پیدا کردن index تعادل در آرایه (Equilibrium Indez)
- پیدا کردن اینکه زیر آرایه با مجموع صفر داریم یا خیر
- بزرگ ترین زیر آرایه ای که، همه زیر آرایه های در آن سایز جمع درایه هایشان کمتر از مقدار k باشد
- پیدا کردن اعداد اولی که می توانند به صورت جمع بیشترین تعداد عدد اول متوالی نوشته شوند
- پیدا کردن Longest Span با یک جمع در دو آرایه باینری
- تولید اعداد رندوم که از طریق توزیع احتمالات ریاضی محاسبه می شوند.

(منبع: https://www.geeksforgeeks.org/prefix-sum-array-implementation-applications-competitive-programming/)

همچنین کد سوال اول بخش اول در فایل prefixSum_part1.cpp و سوال اول بخش سوم در فایل prefixSum_part1.3.cpp و کد سوال اول بخش چهارم در فایل prefixSum_part1.4.cpp و کد سوال 2 نیز در فایل prefixSum_part2.cpp قرار دارد.