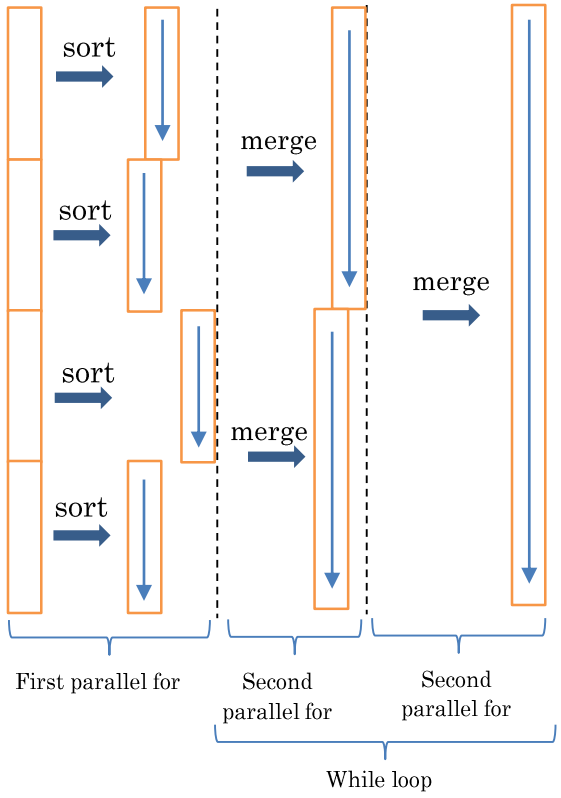
تمرین درس پردازش موازی سری دوم

نام: سینا قادرمرزی شماره‌ی دانشجویی: ۹۳۱۳۱۰۳۸

در این تمرین الگوریتم tim sort به صورت موازی توسط OpenMP پیاده‌سازی می‌شود.

روش استفاده شده:



روش اول) تجزیه‌ی عملیات به طور کلی به این صورت بوده که آرایه‌ی کلی به قطعه‌هایی با طول مشخص تقسیم شده است. (جهت جلوگیری از false sharing آرایه به صورت align شده‌ی ۶۴ بایتی در حافظه ایجاد شده و برای این که هر کدام از نخ‌ها از بلاک‌های جداگانه‌ی cache استفاده نمایند، سایز قطعه‌ها را ۶۴ بایت یا ۱۶ عدد integer انتخاب نموده‌ایم.)

ابتدا قطعه‌ها به صورت موازی در داخل خود مرتب‌سازی می‌شوند(با استفاده از insertion sort). این کار توسط یک ساختار for موازی با زمان‌بندی dynamic انجام می‌شود. پس از این که همه‌ی قطعه‌ها در داخل خود مرتب‌سازی شدند، در مرحله‌ی دوم در یک ساختار for دوم قطعه‌ها دو به دو ادغام می‌شوند. نتیجه‌ی این عملیات دوبرابر شدن طول قطعه‌های مرتب شده و نصف شدن تعداد آن‌هاست. و این عملیات باید آن‌قدر تکرار شود تا طول قطعه برابر طول کل آرایه گردد. (در این عملیات فرض شده که طول آرایه توانی از ۲ است که البته با یک سری تغییرات می‌توان این فرض را حذف کرد)

روش دوم) در این روش از قابلیت Task در OpenMP استفاده شده است و با استفاده از روش بازگشتی که در merge sort وجود دارد، هر فراخوانی merge sort تبدیل به یک Task در OpenMP می‌شود و به صورت پویا زمان‌بندی شده و توسط یکی از thread های ساخته شده اجرا می‌گردد.

هرچند نتایج به دست آمده نشان می‌دهد که این روش تسریع کمتری ایجاد می‌کند اما به دلیل این که روند طبیعی الگوریتم merge sort در آن استفاده شده در این‌جا آورده شده است.

نحوه‌ی اجرای برنامه به این صورت است که برنامه در خط فرمان به صورت زیر اجرا می‌گردد

```
./main16 <array_size> <#threads>
```

۱- ابتدا الگوریتم merge sort عادی ۵ بار اجرا شده و زمان آن اندازه‌گیری می‌شود و به همراه میانگین زمان اجرا گزارش می‌گردد.

۲- سپس الگوریتم tim sort سریال ۵ بار اجرا شده و زمان آن اندازه‌گیری می‌شود. و به همراه میانگین زمان اجرا و میزان تسریع گزارش می‌شود.

۳- سپس الگوریتم tim sort موازی (روش اول) ۵ بار اجرا شده و میانگین زمان اجرا و تسریع آن نسبت به tim sort سریال گزارش می‌شود.

۴- سپس الگوریتم tim sort موازی (روش دوم مبتنی بر task) ۵ بار اجرا شده و میانگین زمان اجرا و تسریع آن نسبت به tim sort سریال گزارش می‌شود.

به دلیل این که در محیط parallel studio و ویندوز به دلایل نامشخصی هنگام بالا رفتن تعداد threadها زمان اجرا به جای بهبود یافتن بدتر می‌شد. در نهایت سورس کد برای g++ تنظیم شده ، و اجرای برنامه در محیط لینوکس و کامپایلر g++ انجام گرفته است.

همانطور که می‌دانیم در الگوریتم tim sort برای مرتب کردن زیر آرایه‌ها هنگامی که اندازه‌ی آن‌ها از حدی کوچکتر می‌شود، آن را به insertion sort می‌سپاریم. در این برنامه اندازه‌ی قطعه‌هایی که باید به Insertion sort سپرده شود، در یک ماکرو به نام segment_size قرار داده شده است و سورس کدهای main4 ، main8 ، main16، main32و main64 با نسبت دادن مقادیر متناظر ۴، ۸ ، ۱۶ ، ۳۲ و ۶۴ به segment_size ایجاد شده و کامپایل گردیده‌اند. تا با اجرای هرکدام از این برنامه‌ها تاثیر این عامل مشاهده گردد.

نتایج

segment_size	# of threads	speedup ($1MB \sim 2^{20}B$)	speedup ($10MB \sim 2^{23}B$)	speedup ($100MB \sim 2^{27}B$)	speedup ($1GB \sim 2^{30}B$)
4	2	2.782999	2.406698	2.382715	?
	4	4.399320	3.446273	2.938736	?
	8	4.652689	4.678148	2.486360	?
8	2	3.065223	2.874384	2.343103	?
	4	3.880807	4.322674	3.186781	?
	8	6.360669	4.603671	2.634015	?
16	2	3.120741	3.060110	2.856844	?
	4	5.080948	4.438980	4.276991	?
	8	5.062926	4.482665	3.746407	?
32	2	3.022475	2.986325	2.755344	?
	4	3.944716	4.335279	3.125338	?
	8	5.290780	4.643464	3.063123	?
64	2	2.768403	2.757283	2.785749	?
	4	4.295042	3.663392	3.646281	?
	8	5.135854	5.155970	3.000446	?

جدول۱- تسریع الگوریتم tim sort موازی نسبت به tim sort سریال

segment_size	# of threads	speedup ($1MB \sim 2^{20}B$)	speedup ($10MB \sim 2^{23}B$)	speedup ($100MB \sim 2^{27}B$)	speedup ($1GB \sim 2^{30}B$)
4	2	1.604142	1.539117	1.598320	?
	4	1.570591	1.494417	1.598289	?
	8	1.482070	1.611138	1.558635	?
8	2	1.611688	1.501750	1.600279	?
	4	1.505926	1.560978	1.575510	?
	8	1.581017	1.570810	1.609599	?
16	2	1.597420	1.591189	1.667568	?
	4	1.635730	1.548525	1.709763	?
	8	1.498316	1.475581	1.671397	?
32	2	1.506640	1.525774	1.576331	?
	4	1.563478	1.525434	1.570020	?
	8	1.452163	1.523756	1.717336	?
64	2	1.526943	1.502768	1.672776	?
	4	1.512526	1.487220	1.622922	?
	8	1.407075	1.572138	1.613002	?

جدول ۲- تسریع الگوریتم tim sort موازی مبتنی بر task نسبت به tim sort سریال

segment_size	speedup ($1MB \sim 2^{20}B$)	speedup ($10MB \sim 2^{23}B$)	speedup ($100MB \sim 2^{27}B$)	speedup ($1GB \sim 2^{30}B$)
4	0.993510	1.004124	1.002694	?
8	0.999043	1.001294	1.005619	?
16	1.006447	1.016038	1.003513	?
32	1.010938	1.005042	1.005890	?
64	1.017879	1.004044	1.002071	?

جدول ۳- تسریع الگوریتم tim sort سریال نسبت به merge sort

نتایج نشان می‌دهد که الگوریتم موازی اول تسریع بسیار خوبی دارد اما این تسریع با افزایش اندازه‌ی ورودی‌ها در حال کاهش است. الگوریتم نوع دوم تسریع کم و کاملاً ثابتی دارد که کمی عجیب به نظر می‌رسد. و الگوریتم tim sort سریال نسبت به merge sort هیچ گونه تسریعی ندارد. برای سایز ۱گیگ زمان اجرا بسیار طولانی بوده و بنابراین از آن صرف نظر شده است.