به نام خدا

برنامه‌نویسی چندهسته‌ای

پاسخ سوالات دستور کار آزمایشگاه ۴

امیرمحمد پیرحسینلو ۹۵۳۱۰۱۴

مهدی صفری

هدف از این آزمایش، انجام عملیات Prefix sum بر روی یک آرایه است.

عملیات Prefix sum به‌صورت زیر تعریف می‌شود:

$y[i] = \sum_{j=0}^{i} x[j]$

به‌عبارت‌دیگر، هر درایه در آرایه خروجی، جمع همه درایه‌های قبل از خود در آرایه ورودی است.

این الگوریتم معمولاً به دو شیوه inclusive و exclusive پیاده می‌شود که تفاوت این دو را در ادامه می‌توانید ببینید:

Inclusive:

x	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
y	۱	۳	۶	۱۰	۱۵	۲۱	۲۸	۳۶	۴۵	۵۵

Exclusive:

x	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
y	۰	۱	۳	۶	۱۰	۱۵	۲۱	۲۸	۳۶	۴۵

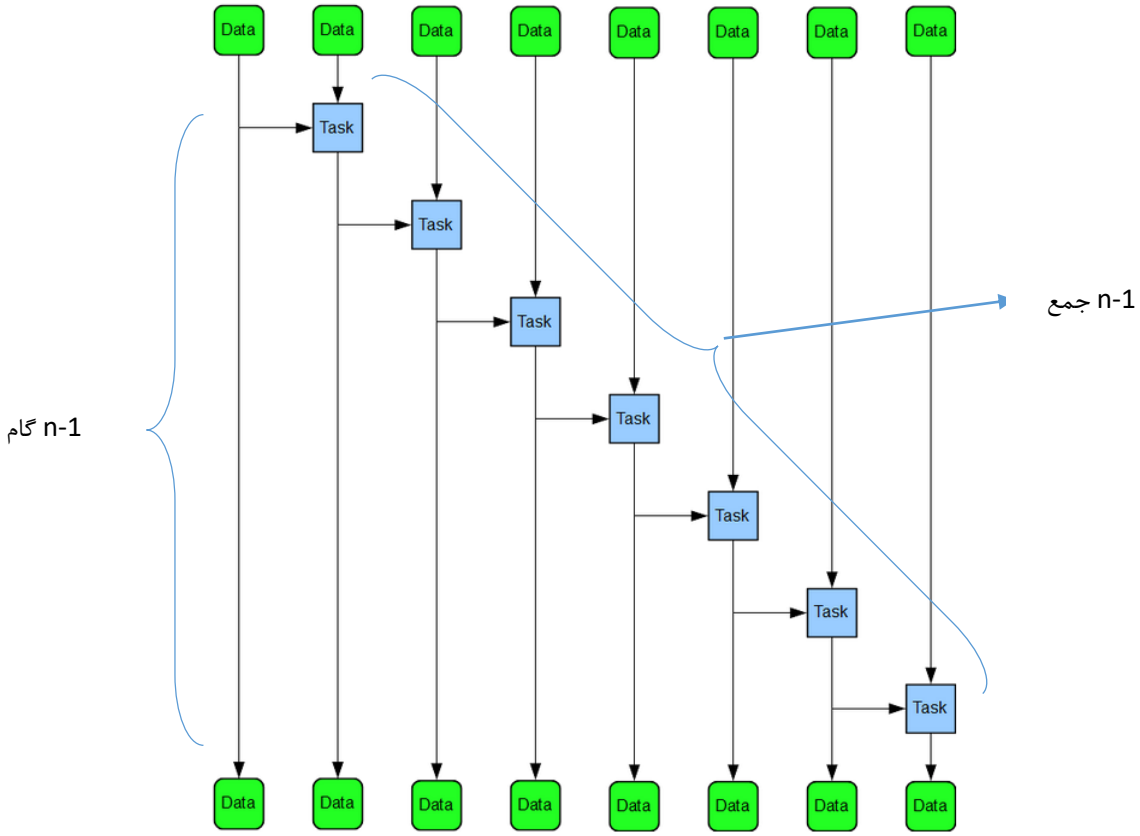
این عملیات کاربرد فراوانی در حوزه‌های مختلف دارد. به‌عنوان‌مثال:

اگر آرایه x همه تراکنش‌های مالی یک حساب (میزان کاهش یا افزایش حساب) است، آرایه y مقدار موجودی حساب تا پایان هر تراکنش خواهد بود.

و یا اگر آرایه x مقادیر تابع احتمالی PDF باشد، آرایه y مقادیر تابع احتمالی تجمعی CDF خواهد شد.

کد سریال این الگوریتم با نام lab_4_serial.c در اختیار شما قرار دارد.

نحوه انجام این عملیات به‌صورت شماتیک در شکل زیر مشخص است:



در این آزمایش، سه روش موازی‌سازی الگوریتم Prefix sum معرفی می‌شود.

توجه کنید در الگوریتم بالا n-1 عمل جمع و نیز n-1 گام برای محاسبه خروجی لازم است.

۱- روش اول:

ابتدا آرایه x را بین نخ‌ها به‌صورت static تقسیم کنید (هر نخ 1/n آرایه را پردازش می‌کند). هر نخ عملیات prefix sum را به‌صورت مستقل بر روی زیرآرایه خود انجام می‌دهد. به‌عنوان‌مثال 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 خانه‌های زیرآرایه‌های بعد از آن جمع کنیم. مثلاً مقدار خانه آخر زیرآرایه اول (۱۵) باید با همه خانه‌های زیرآرایه دوم جمع شود. چرا؟
  زیرا در غیر این صورت اثر خانه های ابتدایی در خانه های انتهایی دیده نمی شود.
- اگر در مثال بالا تعداد نخ‌ها چهار شود برای به دست آوردن مقدار نهایی یک‌راه حل این است که هر نخ مقدار آخرین خانه محاسبه‌شده توسط خود را با تمام خانه‌های پس‌ازآن جمع کند. این روش علاوه بر داشتن race، بیش‌ازاندازه مورد نیاز عمل‌های جمع انجام می‌دهد. در این روش به ازای هر خانه از خروجی باید به تعداد بخش‌های قبل از آن روی خانه موردنظر عمل جمع صورت بگیرد. چگونه می‌توان به ازای هر خانه خروجی فقط با انجام یک عمل جمع این به‌روزرسانی را انجام داد و از روی آرایه بالا جواب نهایی را به دست آورد؟
  برای مثال آرایه به ۴ بخش تقسیم شده و prefix sum در هر بخش محاسبه شده است. عنصر انتهای بخش اول را باید به عناصر بخش دوم اضافه کنیم. این کار را توسط ۴ نخ انجام می دهیم(به صورت استاتیک). حال باید عنصر آخر بخش دوم را به عناصر بخش سوم اضافه کنیم. این کار را هم توسط ۴ نخ انجام می دهیم. به همین منوال که جلو برویم به ازای هر خانه خروجی فقط با انجام یک عمل جمع این به‌روزرسانی را انجام داده ایم.
- برای این الگوریتم تعداد جمع‌ها و نیز تعداد گام‌ها را به دست آورید

#threads = t
#size = n

sum: n – t + (t-1) * (n/t)

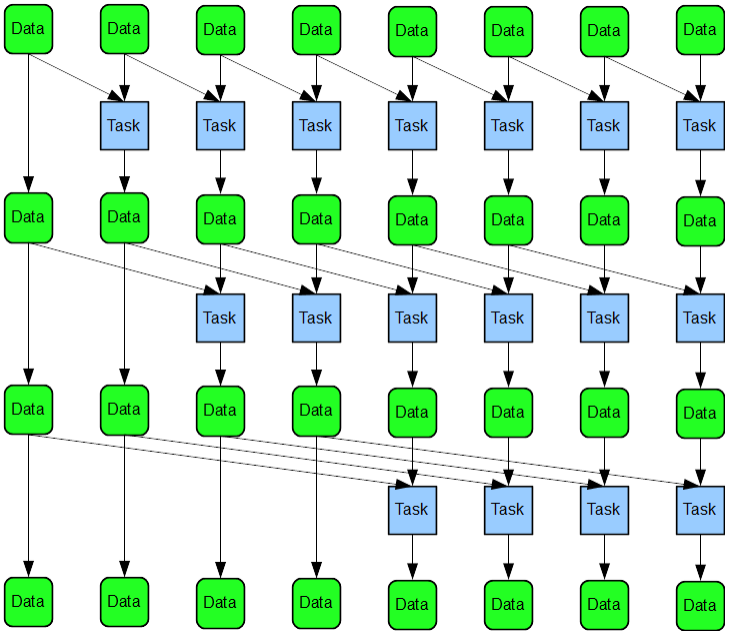
steps: $\left(\frac{n}{t}\right) - 1 + (t-1)\left(\frac{n}{t^2}\right)$

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

در حالت موازی ۰,۶۵۵۰ ثانیه طول کشید در حالی که در حالت سریال ۰,۷۶۶۰ ثانیه طول کشید. دلیل عدم دست یابی به بازدهی بالا این بود که زیاد به parallel region وارد و خارج می شدیم و سرباز زیادی داشت. میزان تسریع: ۱,۱۶۹۴

۲- 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 محاسبه‌شده است. هر مربع task یک جمع است.

این الگوریتم time-optimal است؛ به این معنا که تعداد گام‌ها را کمینه کرده است بنابراین اگر به اندازه داده ورودی مسئله نخ داشته باشیم (تا تمام گام‌ها در یک واحد زمانی انجام شوند) آنگاه زمان اجرای برنامه در حالت موازی کمینه (log n) خواهد شد؛ اما می‌دانیم در cpu این تعداد نخ موجود نیست و بنابراین کارهای مربوط به هر گام باید بین نخ‌ها تقسیم‌شده و بنابراین بیشتر از یک واحد زمانی طول خواهد کشید.

- با توجه به توضیحات بالا می‌توانید محاسبه کنید به ازای یک گیگ ورودی چه تعداد نخ لازم داریم تا روی کاغذ بتوان تسریع گرفت؟

معادله زیر را حل کنیم تا مقدار n بر حسب t به دست آید:

$$nlog(n) - n + 1 \leq n - t + n - \frac{n}{t}$$

- برای این الگوریتم تعداد جمع‌ها و نیز تعداد گام‌ها را به دست آورید.

#threads = t
#size = n

sum: $(n-2^0)+(n-2^1)+(n-2^2)+\cdots+\left(n-2^{\log(n)-1}\right)=nlog(n)-n+1$

steps = $ceil(\log(n))$

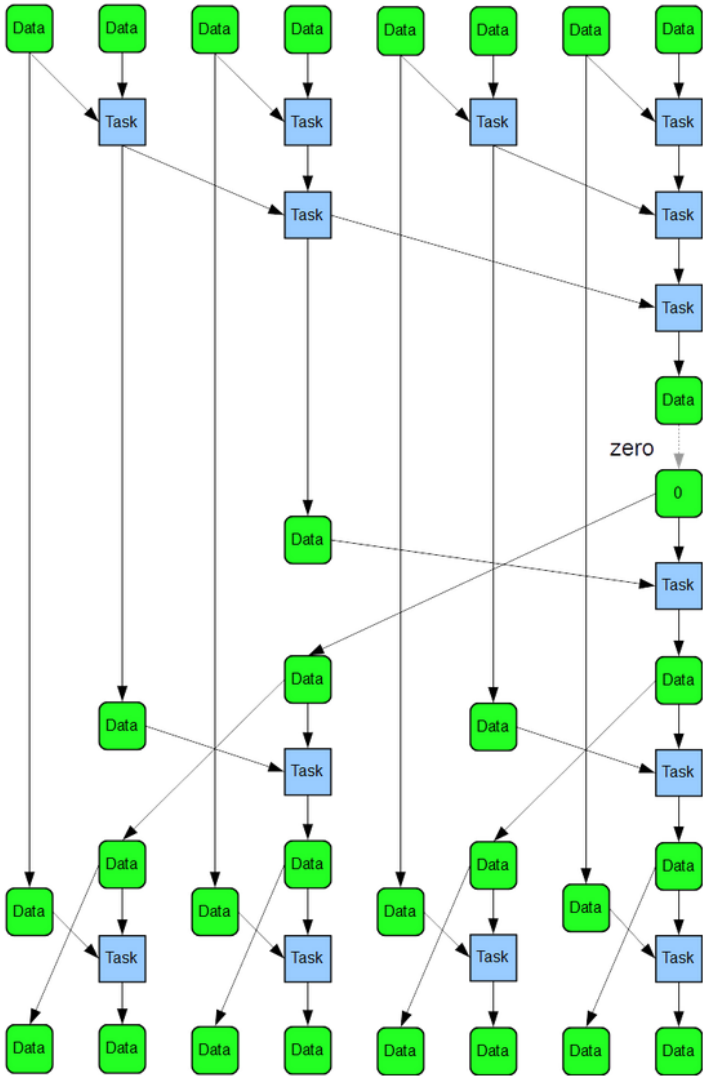
- کد موازی این الگوریتم را بنویسید و زمان اجرای آن را به ازای ۱ گیگ اندازه حافظه با حالت سریال مقایسه کنید.

در این حالت ۱۲,۸۵۱۰ ثانیه طول کشید کارایی خیلی پایین آمد زیرا تعداد نخ ها کم بود و عملیات کپی کردن بین آرایه های a و b زمان زیادی می برد.

۳- روش سوم:

این الگوریتم توسط guy blelloch بر اساس balanced-tree در سال ۱۹۹۳ ارائه شد.

توجه کنید که این الگوریتم برخلاف دو روش قبل ماهیت exclusive دارد.



نحوه پیاده‌سازی این الگوریتم را در قطعه کد زیر می‌بینید:

```
void work_efficient_parallel_prefix_sum(int *a, int n) {
    for (int i = 1; i < n; i <<= 1)
        for (int j = 0; j < n; j += 2 * i)
            a[2 * i + j - 1] = a[2 * i + j - 1] + a[i + j - 1];

    a[n - 1] = 0;

    for (int i = n / 2; i > 0; i >>= 1) {
        for (int j = 0; j < n; j += 2 * i) {
            int temp = a[i + j - 1];
            a[i + j - 1] = a[2 * i + j - 1];
            a[2 * i + j - 1] = temp + a[2 * i + j - 1];
        }
    }
}
```

این الگوریتم work-optimal است؛ به این معنا که به دلیل محدودیت ما در داشتن هر تعداد نخ، تعداد عمل‌های جمع برای اجرای این الگوریتم به صورت موازی کمینه شده است و برای رسیدن به این هدف تعداد گام‌ها دو برابر شده است.

- قطعه کد بالا را موازی کرده و زمان اجرای این الگوریتم را با روش‌های قبل مقایسه کنید.
- توضیح دهید این الگوریتم که درواقع بهینه‌شده الگوریتم روش دوم است در چه حالتی می‌تواند نسبت به الگوریتم اول مزیت داشته باشد (راهنمایی: این الگوریتم در GPU بسیار پرکاربرد است).

در صورتی که بتوان برای هر تسک در هر سطح یک نخ اختصاص داد تا همه کار های هر سطح موازی اجرا شوند.

- این الگوریتم تنها برای توان‌های دو عملکرد صحیحی دارد. چگونه می‌توان برای هر عددی از این الگوریتم استفاده نمود؟

می توان برای نزدیک ترین عدد توان ۲ محاسبه کرد. یک بار هم مقدار باقی مانده را با روش های ۱ یا ۲ (ذکر شده در بالا) محاسبه کرد. جواب این قسمت را هم با قسمت بزرگتر(توان ۲) جمع می کنیم.

- چگونه می‌توان این الگوریتم را به صورت inclusive نیز پیاده‌سازی کرد؟ (نیازی به پیاده‌سازی نیست)

کافی است به هر عضو a هنگام عملیات به چشم a[i] + i نگاه کنیم. به عبارتی هر جا که میخوایم مقدار a[i] رو مورد پردازش قرار دهیم، a[i]+i را پردازش کنیم.