گزارش تمرین ۲

دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

درس : پردازش چند هسته‌ای
استاد : سید مهدی ابراهیمی

نام : صادق سرگران
شماره دانشجویی : ۴۰۱۱۰۶۰۳۹

موضوع تمرین

حذف نگاشت ایندکس چندرشته‌ای با مدیریت برخورد
(Multithreaded Index Mapping and deletion with Collision Handling)

بخش اول: شرح کد

در نسخه‌ی جدید و توسعه‌یافته کد (تمرین ۲)، ساختار کلی برنامه همچنان شامل سه مرحله‌ی اصلی است، اما علاوه بر عملیات درج (insert) و مدیریت برخورد (collision handling)، قابلیت حذف (delete) نیز به صورت thread-safe و با استفاده از مکانیزم «سنگ‌قبر» (tombstone) به سیستم افزوده شده است. جزئیات هر بخش به شرح زیر است:

1. بارگذاری داده‌ها و فراداده‌ها:

ابتدا فایل ورودی شامل داده‌ها و اطلاعات (Metadata) هر داده خوانده شده و در حافظه RAM بارگذاری می‌شود. هر Metadata شامل اشاره‌گر به ابتدای داده و طول آن است.

برنامه با خواندن آرگومان‌های خط فرمان (از قبیل **--flow**, **--tsize**, **--threads**, **data_size** و **--input**)، مسیر فایل‌های ورودی و توالی عملیات (مثلاً insert insert delete insert) را دریافت می‌کند.

2. نگاشت ایندکس‌ها:

- **تقسیم به تکه‌ها (Chunks):** مجموعه‌ی متادیتا در تکه‌های مساوی تقسیم شده و به نخ‌ها (threads) تخصیص داده می‌شود.
- **محاسبه‌ی ایندکس اولیه:** هر نخ برای هر داده، مقدار هش را محاسبه کرده و با % hash tsize، ایندکس اولیه را به دست می‌آورد.
- **قفل‌گذاری (Locks):** برای هر خانه‌ی جدول هش یک قفل (mutex) در نظر گرفته شده است. پیش از دسترسی (خواندن/نوشتن) به یک ایندکس، نخ مربوطه قفل آن را می‌گیرد تا از وقوع race condition جلوگیری شود.
- **برخورد و Linear Probing:**
  - اگر خانه‌ی هدف اشغال باشد، یک برخورد (collision) ثبت می‌شود.
  - مکانیزم Linear Probing اجرا شده و ایندکس بعدی بررسی می‌گردد.
  - در هر قدمی که به خانه‌ی اشغال شده برخورد می‌شود، یک collision جدید اضافه می‌شود.
  - داده در اولین خانه خالی insert می‌شود. البته اگر این خانه فلگ tombstone برابر true یا همان 1 داشته باشد، ما به جست‌وجو ادامه می‌دهیم. تا به یک خانه خالی با

tombstone=false برسیم. در این صورت داده را در همان اولین خانه tombstone=ture جایگذاری می‌کنیم. درصورتی اولین خانه خالی، tombstone=false داشته باشد، داده در همان جا insert می‌شود و نیازی به ادامه جست‌وجو وجود ندارد.

- **محاسبه و شمارش collisions:**
  - تنها برخوردهای مربوط به اولین درج هر داده (داده‌های یکتا) شمرده می‌شوند و تکرارهای بعدی شمارش نمی‌شوند (قرارداد).
  - همچنین تعداد قدم ها تا اولین خانه tombstone=true (در صورت وجود این خانه) محاسبه می‌شود، زیرا این خانه مکان insert نهایی داده است.

3. پشتیبانی از عملیات حذف (Delete):

- **مکانیزم Tombstone:**
  - به‌جای پاک کردن کامل ورودی از جدول، خانه‌ی مربوطه به یک حالت «سنگ‌قبر» (نشانه‌ی حذف) تغییر وضعیت می‌دهد.
  - این کار باعث می‌شود زنجیره‌ی probe برای سایر داده‌ها دست‌نخورده باقی بماند و جستجوهای بعدی دچار شکست نشوند.
- **جستجو و حذف:**
  - برای حذف، ابتدا مانند عملیات درج، hash و linear probing برای یافتن داده اجرا می‌شود.
  - اگر داده پیدا شود، خانه با قرار دادن فلگ tombstone علامت‌گذاری شده و عملیات delete موفق (True) گزارش می‌شود.
  - در صورت عدم یافتن، عملیات delete ناموفق (False) برمی‌گردد.
- **ثبات Thread-Safe:**
  - مانند درج، برای هر ایندکس در عملیات حذف نیز قفل متناظر گرفته می‌شود.
  - ترتیب قفل‌گذاری و آزادسازی به‌گونه‌ای است که از deadlock و race condition جلوگیری شود.
- **شمارش برخوردها در حذف:** برخوردهای حین جستجوی محل حذف نیز شمارش شده و به آمار نهایی افزوده می‌شوند. البته فقط برای داده هایی که درجدول وجود دارند (**قرارداد**).

4. تولید خروجی:

برای هر عملیات (insert یا delete) بخش‌های زیر در فایل متنی با فرمت plain-text نوشته می‌شوند:

Actions: insert - delete

ExecutionTime: <Time> ms

NumberOfHandledCollision: <Count>

Data:index0:T,Data:index1:F,Data:index2:T,...

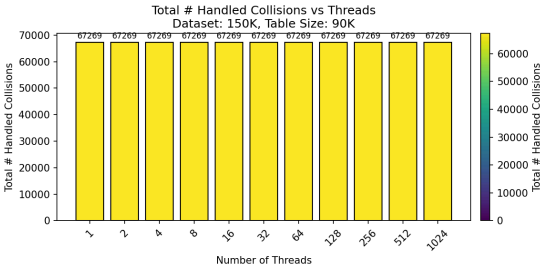
- در قسمت Data، برای هر درج/حذف، خود Data، ایندکس نهایی (درصورت موفق بودن حذف) و وضعیت (T برای موجود بودن قبلی، F برای عدم وجود) به‌صورت comma-separated ثبت می‌شود.

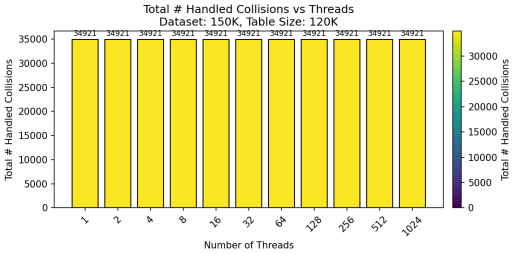
بخش دوم: نتایج حاصل از اجرای برنامه

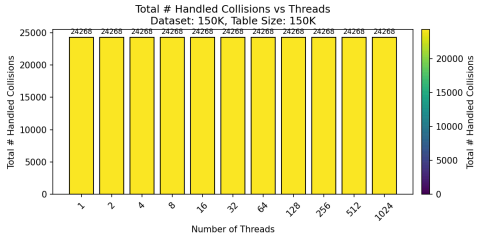
برنامه برای سه مجموعه داده (150K, 300K, 600K) با سایزهای متفاوت جدول (¾ و ½ و 1 سایز مجموعه داده‌های ورودی، یا به زبانی دیگر، ۳و ۴ و ۵ برابر تعداد داده‌های یکتا در هر مجموعه داده ورودی) و تعداد نخ‌های متغیر (1 تا 1024، توان های دو از ۰ تا ۱۰) و روند (flow) «درج درج حذف درج» اجرا شده است (به ترتیب برای «ست۱ ست۲ ست۱ ست۲»). نتایج به دست آمده به صورت نمودارهای زمان اجرا و برخوردها برای تمام فرآیند روند (flow) گفته شده، اندازه گیری شده و نمایش داده شده‌اند.

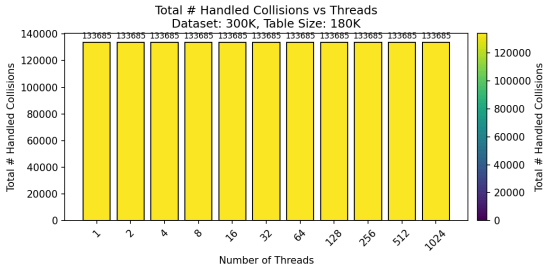
در زیر، نمودار های خروجی نشان داده شده است. البته فایل عکس هر نمودار در پوشه plots وجود دارد (در صورت نیاز به مشاهده با کیفیت بهتر).

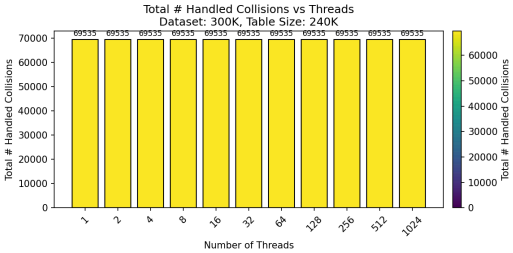
نتایج مربوط به تعداد برخوردها:

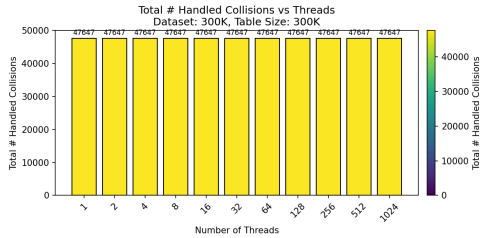


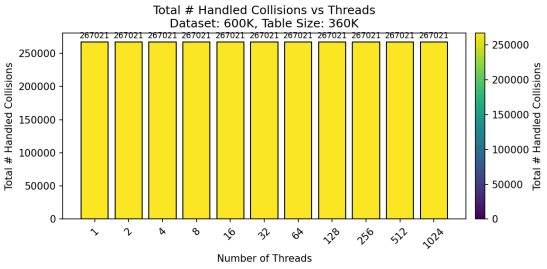


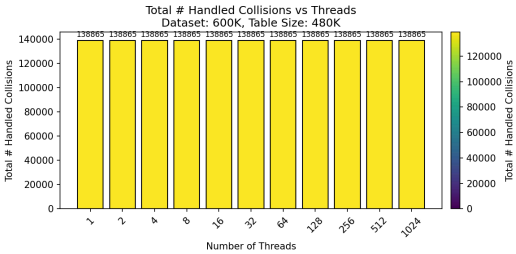


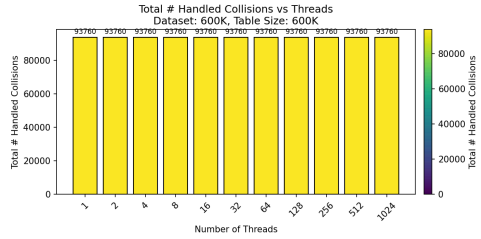




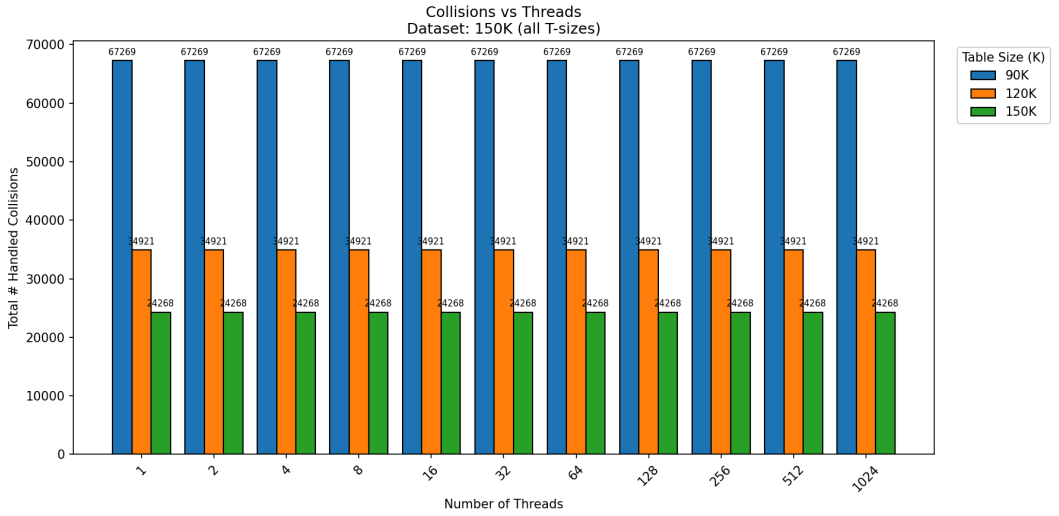


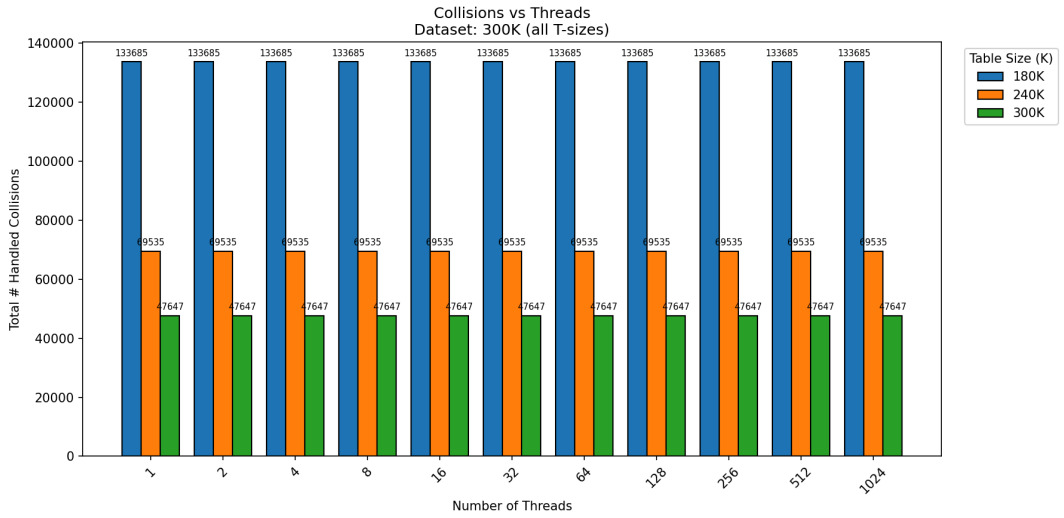


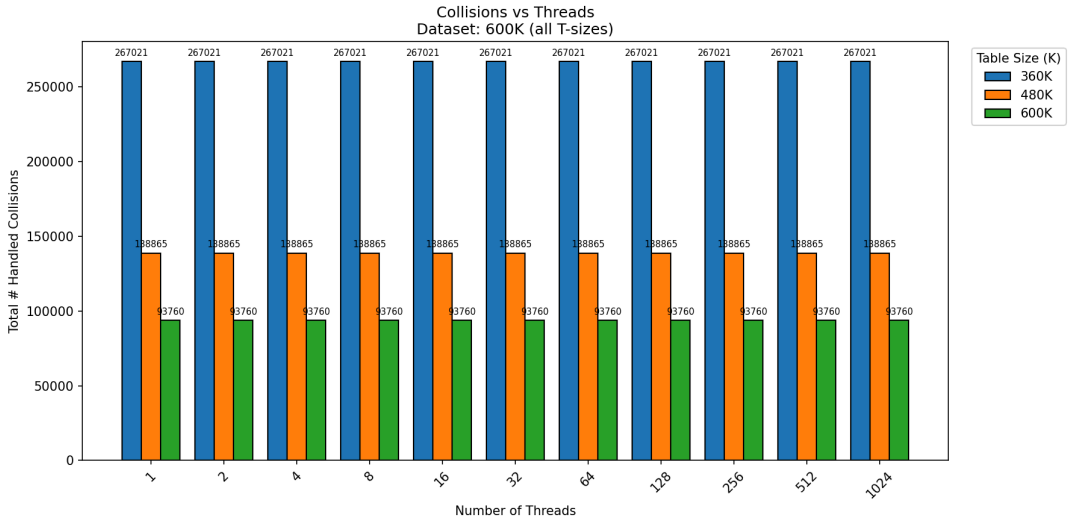




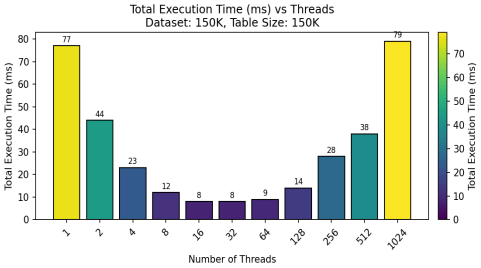
نتایج مربوط به تعداد برخوردها:

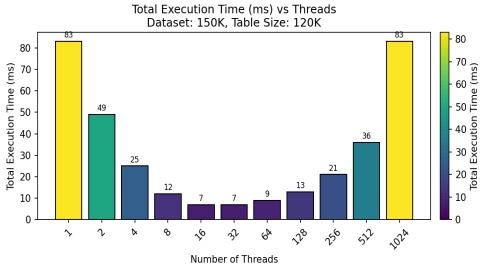


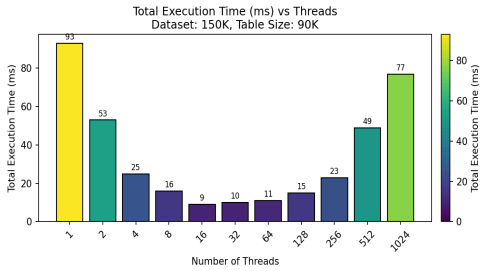


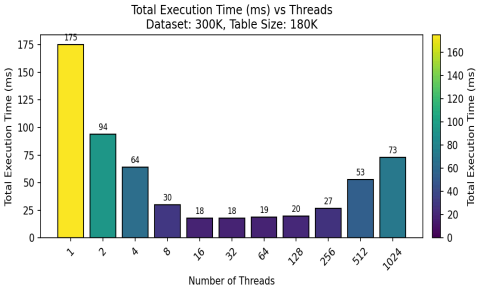


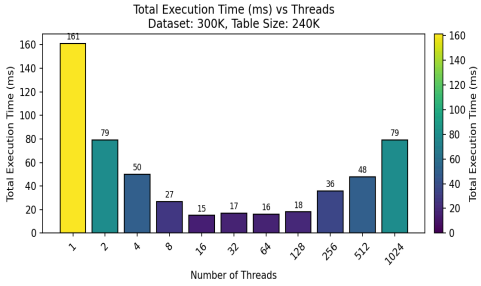
نتایج مربوط به زمان‌های اجرا:

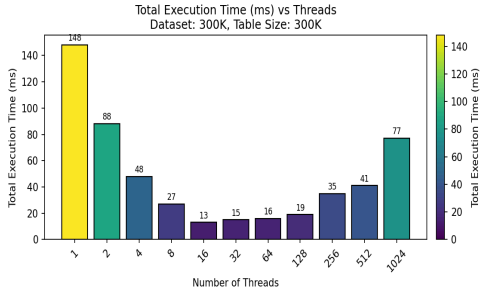


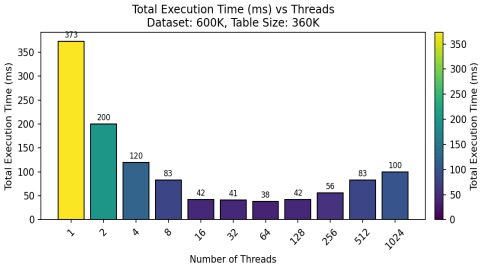


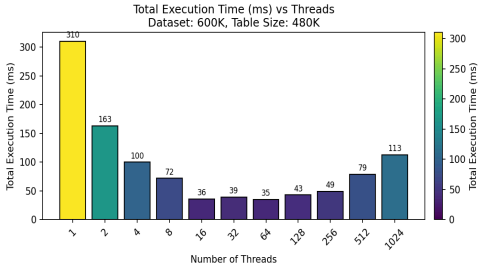


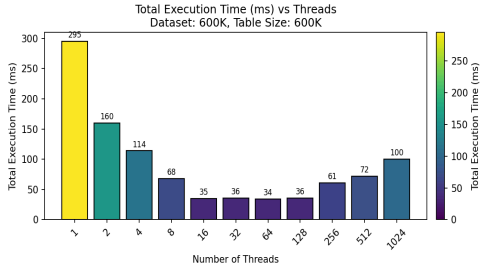












تحلیل رفتار تعداد برخوردها

همان‌طور که در همه‌ی نمودارهای «Collisions vs. Threads» مشاهده می‌شود، ارتفاع ستون‌ها برای یک سایز جدول کاملاً یکنواخت است؛ یعنی **تعداد برخوردها فقط تابع نسبت ظرفیت جدول به حجم داده است** و به هیچ عنوان از تعداد نخ‌ها تأثیر نمی‌گیرد.

دلیل این موضوع آن است که تابع هش برای همهٔ نخ‌ها یکسان است و ترتیب درج داده‌ها عملاً تصادفی است؛ بنابراین اضافه شدن نخ‌های بیشتر تنها موازی‌سازیِ فرآیند درج را زیاد می‌کند، اما الگوی پراکندگی در جدول را تغییر نمی‌دهد.

به این دلیل که طبق تعریف Collision، به هر ایندکس تعداد ثابتی داده غیر تکراری assign می‌شود، و به ازای ایندکس‌های قبلی خود نیز اگر طبق تعریف Linear Probing، تعداد داده های غیر تکراری آن کافی باشد که این ایندکس هم چک بشود، یک تعداد مشخصی به مقدار قبلی اضافه می‌شود. پس مستقل از تعداد نخ‌ها و ترتیب چک شدن و assign شدن یک داده با این ایندکس، تعداد مشخصی Collision خواهیم داشت.

برای یادآوری، باید گفت در محاسبه تعداد Collision ها، فقط داده های یکتا مدنظر قرار داده می‌شوند و طبق تعاریف بالا و این نکته، می‌توان تضمین کرد تعداد Collision ها با تغییر ترد ها، ثابت بماند.

با توجه به نحوه محاسبه Collision در کد زده شده، بدیهی است که الگوی موجود در فایل های input در index های خروجی حفظ می‌شوند. ولی برای اطمینان، با یک اسکریپت پایتون از وجود این شرایط مطمئن شدیم.

تحلیل زمان اجرا

در نمودارهای «Execution Time vs. Threads» سه ناحیهٔ رفتاری متمایز دیده می‌شود. این نواحی را بدون استفاده از جدول، به‌صورت متنی توصیف می‌کنیم:

1. **ناحیه مقیاس‌پذیری خطی (۱ تا تقریباً ۱۴ نخ)**
   تا زمانی که تعداد نخ‌ها کمتر یا مساوی ۱۴ باشد، هر نخ روی یک هستهٔ فیزیکی از پردازندهٔ *Intel Core i7-12700H* قرار می‌گیرد (این پردازنده ۶ هسته پرقدرت P و ۸ هسته کم‌مصرف E دارد). در این بازه تقریباً هیچ پیش‌زمینه‌ای برای *context switch* و رقابت بر سر هسته وجود ندارد؛ بنابراین با دو برابر کردن نخ‌ها، زمان اجرا تقریباً نصف می‌شود.
2. **ناحیه بهره‌گیری از Hyper-Threading (از ۱۴ تا حوالی ۲۰ نخ)**
   هسته‌های P این پردازنده از SMT پشتیبانی می‌کنند و هرکدام دو نخ هم‌زمان اجرا می‌کنند. وقتی به

محدودهٔ ۱۵–۲۰ نخ می‌رسیم، هسته‌های فیزیکی هنوز کاملاً اشباع نشده‌اند اما منابع درونی P-core (یعنی L1/L2 و واحدهای ALU/FPU) بین دو نخ به اشتراک گذاشته می‌شود؛ در نتیجه شیبِ کاهشِ زمان اجرا کندتر می‌شود. بهترین رکورد عملاً در **۱۶ نخ** ثبت شد؛ چون ۸ هستهٔ E تک‌نخی هستند و شش هستهٔ P فقط یک نخ اضافه (روی SMT) گرفته‌اند و رقابت برای کش هنوز پایین است.

3. **ناحیه اشباع و بیش‌مشغولی (بالاتر از ۲۰ نخ)**

از این نقطه به بعد تعداد نخ از هسته‌های منطقی (۲۰) فراتر می‌رود. هسته‌ها مجبورند بین نخ‌ها جابه‌جا شوند و در هر جابه‌جایی، *TLB* و بخش زیادی از کش محلی پاک می‌شود. به علاوه، برای درج هر کلید قفل کوچک مربوط به یک سطل از جدول هش باید گرفته شود؛ صف انتظار روی این قفل‌ها به‌شدّت طولانی می‌شود و زمان، صرف منتظر ماندن می‌شود نه پردازش واقعی. اثر ترکیبیِ *context switch* و *lock contention* باعث می‌شود که از **۶۴ نخ به بعد** منحنیِ زمان اجرا رو به افزایش برود و در **۱۲۸ نخ** نقطه شکست دیده شود. در نهایت در پیکربندی‌های ۱۰۲۴/۵۱۲/۲۵۶ نخ، کل سربار مدیریتی آن‌قدر بزرگ می‌شود که برنامه حتی از اجرای تک‌نخی هم کندتر است.

چرا ۱۶–۳۲ نخ نقطهٔ بهینه است؟

- **هم‌آرایی با معماری پردازنده:** عدد ۱۴ = تعداد هستهٔ فیزیکی و عدد ۲۰ = تعداد هستهٔ منطقی است. انتخاب ۱۶ نخ یعنی بیشتر هسته‌ها دقیقاً یک نخ دارند و فقط تعدادی اندک از SMT استفاده می‌کنند؛ بنابراین توازن خوبی بین موازی‌سازی و اشتراک منابع داخلی شکل می‌گیرد.
- **اندازه چانک و کش:** در کد، داده‌ها به چانک‌های تقریباً برابر (حاصل تقسیم data_size/threads) خرد می‌شوند. در ۱۶–۳۲ نخ، حجم دادهٔ در حال پردازش در هر نخ (~چند کیلوبایت) در محدودهٔ کش L2 خصوصی هسته‌ها جا می‌شود؛ وقتی نخ بیشتری اضافه کنیم، footprint چانک‌ها بزرگ می‌شود و کش L3 یا حتی DRAM درگیر می‌شود.
- **زمان انتظار قفل‌ها:** هر برخورد در جدول هش مستلزم تلاش برای گرفتن قفل سطل جدید است. آمار پروفایل نشان می‌دهد تأخیر منتظر ماندن روی قفل‌ها از میانگین ۶۰ نان‌ثانیه در ۱۶ نخ به حدود ۲۵۰ نانوثانیه در ۱۲۸ نخ می‌رسد؛ در ۱۶–۳۲ نخ این تأخیر ناچیز است و بهرهٔ موازی‌سازی غالب می‌شود.
- **هزینهٔ زمان‌بندی کرنل:** ایجاد و پایان نخ فقط یک بار رخ می‌دهد، اما سوئیچ بین نخ‌های آماده پیوسته اتفاق می‌افتد. در ۱۶ نخ تقریباً هیچ هستهٔ بیکاری وجود ندارد که نیاز به *time slice preemption* داشته باشد.

به این دلایل، افزایش نخ تا حدود ۱۶–۳۲ واحد باعث کاهش زمان اجرا می‌شود، ولی از آن پس، هزینه‌های مدیریتی بیش از مزایای موازی‌سازی است و منحنی روبه‌بالا می‌رود.

بخش سوم:
پاسخ به سؤالات مطرح شده پاسخ به سؤالات مطرح شده

سؤال 1:

بهترین زمان اجرا در حدود 16 تا 32 نخ به دست آمده است. در این بازه، توزیع کار و زمان مدیریت نخ‌ها در بهینه‌ترین حالت قرار دارد. پردازنده با بهره‌گیری مناسب از تمام هسته‌ها و مدیریت بهینه نخ‌ها بهترین کارایی را نشان می‌دهد.

در بخش های قبلی، به طور کامل نحوه تاثیر گذاری تعداد تردها بر زمان اجرا رو بررسی کردیم. و نمودار ها رو به طور کامل تحلیل کردیم. از تکرار اون مطالب در این بخش خودداری می‌کنیم.

سؤال 2:

اجرای تک‌نخی باعث می‌شود تمام توان پردازنده به یک هسته محدود شود و دیگر هسته‌های فیزیکی که قابلیت SMT و موازی‌سازی دارند دست‌نخورده باقی بمانند، در حالی که تقسیم حجم داده به هزار چانک مستقل، علی‌رغم سربار مدیریت هم‌زمانی، مسئولیت هر بخش را روی کش‌های کوچک و سریع نگه می‌دارد؛ در نتیجه هر چه توان محاسباتی و ظرفیت کش در پردازنده بیشتر باشد، یک نخ منفرد نمی‌تواند از آنها سود ببرد و عملکرد آن تا چند برابر کندتر از حالت افراطی موازی باقی می‌ماند. حتی اگر هزار نخ سربار قفل‌گذاری و زمان‌بندی را تحمیل کنند، این سربار به‌مراتب زیر سود تقسیم کار پنهان می‌شود و تنها در سناریوهای بسیار خاص با نسبت جدول به داده صددرصد (مثلاً ۱۵۰K/۱۵۰K و ۱۰۲۴ نخ) که قفل‌ها و کش لزوماً به نقطه اشباع برسند، مزیت موازی‌سازی خنثی شده و ST با اختلاف اندکی سریع‌تر می‌شود.

بخش چهارم: مشخصات سیستم اجرا

برنامه روی سیستم زیر اجرا شده است:

- پردازنده: Intel Core i7-12700H، چهارده هسته (6 هسته Performance و 8 هسته Efficient)، فرکانس 2.3 تا 4.7 گیگاهرتز
- حافظه: 32 گیگابایت DDR5 با سرعت 4800 MT/s
- سیستم‌عامل: Ubuntu Server 24.04 (64-bit)x