دومین سمینار ملی کنترل و بهینه سازی
۲۵-۲۴ آبان ۱۳۹۷
The 2nd National Seminar on Control and Optimization
15-16 November 2018
دانشگاه صنعتی شاهرود

افزایش سرعت بهینه سازی تُنُک با پردازش موازی روی GPU

بابک خوش‌نویس [1] ، محمود امین‌طوسی

دانشجو کارشناسی‌ارشد، دانشگاه حکیم سبزواری

استادیار، دانشگاه حکیم سبزواری

چکیده

امروزه بهینه‌سازی تُنک به عنوان یک مدل جدید و کارآمد در اکثر مسائل و مدل‌سازی‌ها مورد استفاده وسیع قرار می‌گیرد. اغلب حل این مسائل خصوصا در مورد داده‌های حجیم با پیچیدگی محاسباتی بالا و کندی عملکرد همراه است. در چنین شرایطی موازی‌سازی راه‌کاری موثر تلقی می‌گردد. یکی از جدیدترین و موثرترین روش‌های موازی‌سازی استفاده از پردازنده گرافیکی است. در این مقاله پیاده‌سازی الگوریتم جستجوی تطابقی روی پردازنده مرکزی با پیاده‌سازی آن روی پردازنده گرافیکی مورد مقایسه قرار گرفته است. این مقایسه توان محاسباتی بالا و عملکرد فوق‌العاده پردازنده گرافیکی را نشان می‌دهد.

واژه‌های کلیدی: بهینه‌سازی تُنُک، جستجوی تطابقی، محاسبات موازی، پردازنده گرافیک

۱ مقدمه

نمایش تنک مبتنی بر دانش گسترده‌ای‌ست که در دهه‌های اخیر گردآوری شده‌است. این مدل برپایه تبدیلات و تئوری‌ها در پردازش سیگنال، مفاهیم ریاضی و فراتر از این‌ها نمایش تنک ایده‌هایی از یادگیری ماشین در زمینه انطباق یک مدل با یک منبع داده را وام‌دار است. محصول تمامی این زیرساخت‌ها مدلی است که در راستای کاربردهای متعددی همچون فشرده‌سازی، نویززدایی، وضوح فوق‌العاده و ... نشان داده‌شده‌است. یکی از چالش‌هایی که الگوریتم‌های بهینه‌سازی تُنک با آن مواجه هستند، کاهش عملکرد و سرعت اجرای آن‌ها با افزایش حجم داده‌های ورودی می‌باشد. در الگوریتم جستجوی تطابقی با بزرگ‌تر شدن اندازه دیکشنری سرعت اجرا به شدت کاهش می‌یابد.

موازی‌سازی راه‌کاری بسیار کارآمد است که در اکثر مسائل برای افزایش سرعت و بهبود عملکرد مورد استفاده قرار می‌گیرد. با پیشرفت و توسعه پردازنده‌های گرافیکی و فراهم شدن امکان انجام محاسبات عمومی روی این دستگاه‌ها و همچنین ماهیت موازی و معماری منحصربفرد آن‌ها، علم محاسبات موازی متحول و دگرگون شده است. می‌توان از این امکان برای بهبود عملکرد عملیات بهینه‌سازی تُنُک بهره جُست.

[1] خوش‌نویس: بابک b.khoshnevis@gmail.com

۲ بهینه‌سازی تُنُک

اگر سیگنال ورودی بردار ستونی b باشد و بردار نمایش x باشد، آنگاه ارتباط بین این دو یک دستگاه خطی $Ax = b$ است که در آن ماتریس A دیکشنری می‌باشد که ستون‌های آن اتم‌ها هم اندازه سیگنال x هستند. با داشتن D و x ما به دنبال تنک‌ترین جواب در این دستگاه هستیم. در واقع هدف ارائه ساده‌ترین راه برای توصیف x به عنوان یک ترکیب خطی با کم‌ترین اتم‌های ممکن است.

هر چه تعداد اتم‌های موجود در دیکشنری بیش‌تر باشد، تخمین بهتری از سیگنال x خواهیم داشت. اگر دیکشنری D با ابعاد n در m باشد، $m > n$ خواهد بود. لذا با یک دستگاه خطی فرومعین [۲] مواجه هستیم. برای حل دستگاه فرومعین از منظم‌سازی [۳] استفاده می‌شود. ایده این روش این است که براساس میزان کیفیت هر جواب، یک رتبه به هریک از جواب‌های ممکن تخصیص داده‌شود و جواب با بالاترین رتبه به عنوان جواب نهایی انتخاب گردد که آن را به صورت رابطه زیر نشان می‌دهیم:

$$\min_x J(x) \quad s.t. \quad Ax = b$$

چالش اصلی نحوه انتخاب $J(x)$ می‌باشد. واضح است در صورتی که این تابع را به درستی انتخاب کنیم، به هدف خواهیم رسید. ترکیب محدب

در صورتی که تابع منظم‌سازی $J(x)$، محدب موکد باشد، آنگاه بهترین جواب در مجموعه $Ax = b$ منحصربفرد خواهد بود. [؟] بنابراین می‌توان به دنبال توابع منظم‌سازی محدب و یا محدب موکد بود. یک نوع از توابع که دارای این ویژگی هستند، خانواده نرم L_p (برای $p \geq ۱$) نام دارند.

با توجه به اینکه هدف یافتن تنک‌ترین جواب ممکن برای سیستم خطی $Ax = b$ است، می‌توان در این راستا p را صفر در نظر گرفت. در واقع نُرم $L_۰$ مجموع اعمال توابع نشانگر $I(x)$ روی تمامی ورودی‌های بردار x است.به بیان ساده‌تر، نُرم $L_۰$ تعداد عناصر غیرصفر در x را برمی‌گرداند. برخلاف این واقعیت که ما این تابع را نُرم $L_۰$ می‌نامیم، واضح است که این تابع یک نُرم معتبر نیست. چرا که این تابع به شدت غیرمحدب است. اما با توجه به این‌که جواب‌های تُنُک تولید می‌کند، برای حل دستگاه خطی $Ax = b$، می‌توان ازآن به عنوان تابع منظم‌سازی استفاده کرد.

تعریف ۱.۲ (مسئله $P_۰$). *مسئله $P_۰$ به دنبال تنک‌ترین جواب‌های معادلات با سیستم خطی نامعین، $Ax = b$، می‌باشد:*

$$(P_۰) \quad \min_x \|x\|_۰ \quad s.t. \quad Ax = b$$

باید تاکید داشت که $P_۰$ یک مسئله بسیار سخت است. یک رویکرد برای حل چنین مسائلی خانواده الگوریتم‌های حریصانه [۴] می‌باشد. الگوریتم جستجوی تطابقی یکی از روش‌های موثر حریصانه است که در ادامه به معرفی آن می‌پردازیم.

۳ الگوریتم جستجوی تطابقی

ایده الگوریتم‌های حریصانه بسیار ساده است و بر این اساس بنا شده است که مفهوم حقیقی حل مسئله $P_۰$ یک طبیعت حریصانه برای شناسایی تکیه‌گاه مناسب جواب دارد [؟]. جستجوی تطابقی می‌تواند با یک تخمین ارزیابی حریصانه حداقل مربعات به

[۲] Underdetermined linear system
[۳] Regularization
[۴] Greedy

عنوان یک ترکیب خطی اجزا (اتم‌های) یک دیکشنری، یک سیگنال تمیز را از یک نمونه نویزدار بازسازی نماید. این رویکرد بازسازی از محاسبات حجیم راه‌حل حداقل‌مربعات برای مسائل بزرگ جلوگیری می‌نماید، همچنین عملکرد بهبودیافته در زمینه نویز زدایی خصوصا در زمانی که سیگنال سالم تنک باشد بروز می‌دهد.

این روش ابتدا در [؟] مطرح شد. . در این روش در هر مرحله تنها ضریب یکی از اتم‌ها مشخص می‌شود. در هر مرحله یک اتم دیکشنری که بیشترین شباهت(بزرگ‌ترین ضرب داخلی) را به داده آزمون دارد، بعنوان عضو فعال در ترکیب خطی در نظر گرفته شده، ضریب مربوط به آن محاسبه می شود. در مرحله بعد، باقی‌مانده سیگنال آزمون و اتم نخست با بقیه اتم‌ها مقایسه شده و دوباره مشابه‌ترین اتم انتخاب می‌شود. یعنی تفاضل حاصل ضرب این تقریب ۱ تُنُک در دیکشنری از داده آزمون را به عنوان باقی مانده در نظر گرفته ، مراحل فوق تکرار می شوند. در هر مرحله جمع تقریب های ۱ ــ تُنُک به دست آمده با تقریب های قبلی به عنوان تقریب جدید در نظر گرفته می شود و این روند تاجایی ادامه می یابد که یا تعداد مراحل مشخصی طی شود و یا خطا از مقدار معینی کمتر شود.

یکی از مشکلات روش‌های بهینه‌سازی تُنُک و جستجوی تطابقی افت عملکرد این روش‌ها با افزایش حجم داده‌هاست. موازی‌سازی جی‌پی‌یو یکی از راه‌کارهای موثر و کارآمد است که در ادامه به معرفی آن می‌پردازیم.

۴ محاسبات جی‌پی‌یو

پردازنده‌های مرکزی تمایل به داشتن حافظه موقت [۵] بزرگ‌تری نسبت به پردازنده‌های گرافیکی دارند. پردازنده‌های گرافیکی تعداد ثبّات [۶] بیش‌تری برای پشتیبانی از تعداد بیش‌تری رشته [۷] نسبت به پردازنده‌های مرکزی دارند [؟]. پردازنده‌های گرافیکی واحدهای اس‌آی‌ام‌دی [۸] بیش‌تری نسبت به پردازنده‌های مرکزی دارند. پردازنده‌های مرکزی تمایل به داشتن منطق کنترل پیچیده‌ای [۹] دارند، درحالی‌که پردازنده‌های گرافیکی منطق کنترل ساده‌ای دارند اما در عوض دارای رشته‌های بیش‌تری برای مدیریت هستند.

معمولا به محاسباتی که هم از جی‌پی‌یو و هم سی‌پی‌یو برای اجرا استفاده می‌کنند، محاسبات موازی ناهمگون و به اختصار محاسبات جی‌پی‌یو [۱۰] گفته می‌شود. این عرصه به سرعت در حال رشد است. جی‌پی‌یوها عملکرد محاسباتی و پهنای باند حافظه مناسبی ارائه می‌دهند. همین موضوع باعث شده جی‌پی‌یو یک معماری جذاب برای اجرای الگوریتم‌های محاسباتی فشرده تلقی گردد. در حال حاضر دو مدل کدنویسی عمده برای توسعه برنامه روی جی‌پی‌یو در دسترس است، کودا (معماری دستگاه محاسبه یکپارچه) [۱۱] و اوپن‌سی‌ال (زبان محاسبات باز) [۱۲] . شرکت ان‌ویدیا کودا را در اواخر سال ۲۰۰۶ معرفی کرد و این معماری تنها به همراه جی‌پی‌یوهای این شرکت قابل استفاده است. زبان‌های برنامه‌نویسی عمومی مانند کودا و اوپن‌سی‌ال، بر پایه مدل پردازش جریان [۱۳] بنا شده‌اند. این مدل به دلیل ماهیت سطح پایین آن، برنامه نویسی جی‌پی‌یو را سخت می کند.

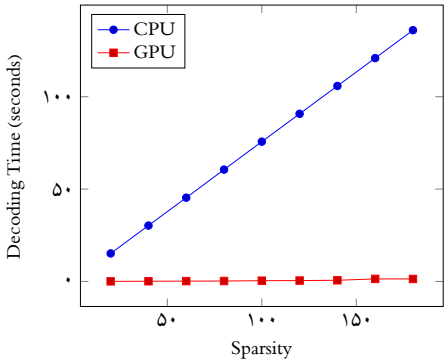
---

[۵] Cache
[۶] Register
[۷] Thread
[۸] Single Instruction Multiple Data (SIMD) Unit
[۹] Sophisticated Control Logic
[۱۰] GPU Computing
[۱۱] CUDA (Compute Unified Device Architecture)
[۱۲] OpenCL (Open Computing Language)
[۱۳] Stream Processing Model

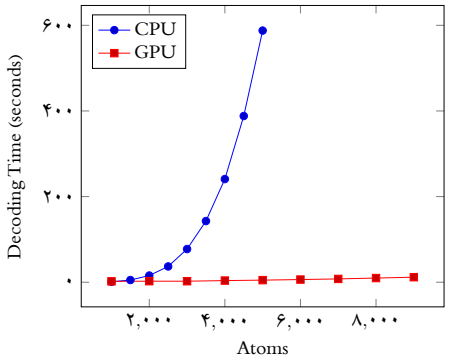
۵ آزمایشات و نتایج تجربی

در راستای مقایسه عملکرد جی‌پی‌یو و سی‌پی‌یو و بهبود عملکرد الگوریتم جستجوی تطابقی، این الگوریتم روی داده‌های تصادفی برای هر یک از این بسترها پیاده‌سازی شد و زمان اجرای هر یک محاسبه شد. در این آزمایش با استفاده از توابع کتابخانه CBLAS، این الگوریتم روی سی‌پی‌یو اجرا می‌شود و همچنین از توابع کتابخانه CUBLAS برای پیاده‌سازی این الگوریتم با کودا سی (زبان C) استفاده می‌گردد. [؟] برای مقایسه عملکرد هر یک از این روش‌ها، ورودی با تعداد اتم‌های دیکشنری متفاوت (شکل؟؟) و همچنین میزان تُنُکی متفاوت (شکل؟؟) اعمال شده است.

روال کلی عملیات در هر دو این پیاده‌سازی‌ها یکسان است. به این صورت که در ابتدا یک دیکشنری با ابعاد $n * m$ و یک بردار تُنُک s به صورت تصادفی تولید می‌شود. سپس با ضرب بردار تُنُک s در دیکشنری، یک سیگنال b تولید می‌شود (عملیات کدگذاری). در مرحله بعد عملیات جستجوی تطابقی برای یافتن نمایش (بردار) تُنُک x برای سیگنال b انجام می‌شود (عملیات کدگشایی) و در پایان بردار s و x برای محاسبه میزان خطا با هم مقایسه می‌شوند. در این آزمایش از یک پردازنده مرکزی اینتل Core i7 -7700HQ و یک پردازنده گرافیکی NVIDIA GeForce GTX 1050 استفاده شده‌است.



شکل ۱: مقایسه با تغییر میزان تُنُکی



شکل ۲: مقایسه با تغییر تعداد اتم‌ها

ملاحظه می‌شود که با بزرگ‌تر شدن اندازه دیکشنری و همچنین کاهش میزان تُنُکی، پیاده‌سازی روی سی‌پی‌یو با کاهش شدید عملکرد و افزایش زمان اجرا به میزان قابل توجه مواجه می‌گردد اما پیاده‌سازی ناهمگون عملکرد قابل قبولی از خود نشان می‌دهد.

مراجع

[۱] حامدی، ر. (۱۳۹۵)، بررسی نُرم‌های مختلف در نمایش تُنُک، *پایان نامه کارشناسی ارشد*، دانشگاه حکیم سبزواری.

[2] Elad M. (2010), *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, 1st Edition, Springer Publishing Company, Incorporated.

[3] Mallat et al. (1993), *Matching pursuits with time-frequency dictionaries*, Signal Processing, IEEE Transactions on, 41-12.

[4] Kirk, David B. and Hwu, Wen-mei W. (2010), *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA.

[5] M. Andrecut. (2009), Accelerating 2D Orthogonal Matching Pursuit Algorithm on GPU, *Engineering Letters*.