روشی کارا برای پیاده‌سازی موازی الگوریتم دسته‌بندی بسته درخت سلسله‌مراتبی بر روی واحد پردازش گرافیکی

میلاد رفیعی[۱]، دانشجوی کارشناسی ارشد، مهدی عباسی[۲]، استادیار، محمد نصیری[۳]، استادیار

۱- گروه مهندسی کامپیوتر- دانشکده مهندسی- دانشگاه بوعلی سینا- همدان- ایران- m.rafiee92@basu.ac.ir
۲- گروه مهندسی کامپیوتر- دانشکده مهندسی- دانشگاه بوعلی سینا- همدان- ایران- abbasi@basu.ac.ir
۳- گروه مهندسی کامپیوتر- دانشکده مهندسی- دانشگاه بوعلی سینا- همدان- ایران- m.nassiri@basu.ac.ir

چکیده: دسته‌بندی بسته‌ها، پردازشی اساسی در پردازنده‌های شبکه‌ای است. در این فرآیند، بسته‌های ورودی از طریق تطبیق یا مجموعه‌ای از فیلترها به جریان‌های مشخص طبقه‌بندی می‌شوند. پیاده‌سازی‌های نرم‌افزاری الگوریتم‌های دسته‌بندی با وجود هزینه کم‌تر و توسعه‌پذیری بیش‌تر نسبت به پیاده‌سازی‌های سخت‌افزاری، سرعت پایین‌تری دارند. در این مقاله، از قابلیت پردازش موازی پردازنده‌های گرافیکی برای تسریع الگوریتم درخت سلسله‌مراتبی دسته‌بندی بسته‌ها، استفاده نموده و سناریوهای متفاوتی را بر اساس معماری حافظه‌های سراسری و اشتراکی آن‌ها پیشنهاد می‌نماییم. نتایج پیاده‌سازی این سناریوها، ضمن تأیید پیچیدگی‌های زمانی و حافظه‌ای محاسبه‌شده، نشان می‌دهد کارایی سناریوهایی که مجموعه فیلتر را به‌صورت زیردرخت‌هایی کوچک‌تر یا مساوی حافظه اشتراکی تقسیم و به آن کپی می‌کنند کم‌تر از سناریویی است که کل ساختار داده را در حافظه سراسری نگه می‌دارد. کارایی این سناریوها، با کاهش تعداد زیردرخت‌ها و فیلترهای تکراری افزایش می‌یابد علاوه بر این، سناریویی که بتواند درخت سلسله‌مراتبی و مجموعه فیلترهای متناظر را، بدون افراز در حافظه اشتراکی جای دهد برترین سناریو است. نتایج آزمایش نشان می‌دهد که نرخ گذرداد حاصله در این سناریو نسبت به روش‌های موجود بر روی یک GPU یکسان تا ۲/۱ برابر بهبود می‌یابد.

واژه‌های کلیدی: دسته‌بندی بسته، الگوریتم درخت سلسله‌مراتبی، واحد پردازش گرافیکی، کودا، سلسله‌مراتب حافظه، پیچیدگی، کارایی

An Efficient Method for Parallel Implementation of H-Trie Packet Classification Algorithm on GPU

M. Rafiee, Master Student[1], M. abbasi, Assistant Professor[2], M. Nassiri, Assistant Professor[3]

1- Computer Engineering Department, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran, Email:m.rafiee92@basu.ac.ir
2- Computer Engineering Department, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran, Email: abbasi@basu.ac.ir
3- Computer Engineering Department, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran, Email: m.nassiri@basu.ac.ir

Abstract: Packet classification is a fundamental process in network processors. In this process, input packets are classified into distinct set of flows via matching against a set of filters. Software implementation of packet classification algorithms, though having lower cost and more scalability as compared with hardware implementations, are slower. In this paper, we use parallel processing capabilities of the graphical processors to accelerate Hierarchical-Trie packet classification algorithm and propose different scenarios based on the architecture of their global and shared memories. Results of implementing these scenarios, conforming computed time and memory complexities, show that the performance of the scenarios that divide the filter set into sub-trees, equal to/ smaller than the shared memory and copy them to it, is lower than that of a scenario which keeps the total data structure in the global memory. The performance of these scenarios increases by decreasing the number of sub-trees and duplicated filters. Moreover, a scenario that can keep hierarchical tree and corresponding filters in shared memory, without any partitioning, is the best scenario. The experimental results show that, on a same GPU, this scenario attains a throughput of approximately 2.1 times compared to the existing methods.

Keywords: Packet classification, H-trie algorithm, graphical processing unit, CUDA, memory hierarchy, complexity, performance.

تاریخ ارسال مقاله: ۹۴/۰۶/۰۳
تاریخ اصلاح مقاله: ۹۴/۰۹/۱۰
تاریخ پذیرش مقاله: ۹۴/۱۰/۲۶
نام نویسنده مسئول: مهدی عباسی
نشانی نویسنده مسئول: ایران - همدان – خیابان چهار باغ شهید احمدی روشن - دانشگاه بوعلی سینا - دانشکده مهندسی- گروه مهندسی کامپیوتر

۱- مقدمه

فرآیند طبقه‌بندی بسته‌های شبکه به جریان‌های مختلف در ابزارهای مختلف شبکه‌ای نظیر مسیریاب‌ها و سوئیچ‌ها را دسته‌بندی بسته‌ها می‌نامند. دسته‌بندی بسته‌ها در بسیاری از کاربردهای شبکه‌ای که در آن‌ها پردازش حجم بسیار زیادی از بسته‌ها در زمان محدود الزامی است، به‌عنوان یک راه‌حل کلیدی استفاده می‌شود. سرویس‌هایی چون اعمال سیاست‌های مدیریت ترافیک روی ارتباط‌های کاربران مختلف، اعمال سیاست‌های امنیتی در یک شبکه و از همه مهم‌تر مسیریابی در شاه‌راه‌های اینترنت تنها با استفاده از دسته‌بندی بسته‌ها امکان‌پذیر است؛ زیرا با دسته‌بندی بسته‌ها به جریان‌های مشخص، می‌توان یک فیلتر یکسان را به تمامی بسته‌های متعلق به یك جریان ترافیکی اعمال نمود [۱].

برای دسته‌بندی بسته‌ها روش‌های سخت‌افزاری و نرم‌افزاری ارائه‌شده است. در مهم‌ترین روش‌های سخت‌افزاری از آرایه گیت‌های منطقی برنامه‌پذیر و حافظه‌های تداعی‌گر سه‌وضعیتی[1] استفاده می‌شود [۲]. دسته‌بندهای مبتنی بر معماری‌های فوق علی‌رغم سرعت بالا و دستیابی به نرخ گذردادی در حد ۱۰۰MPPS[2]، به دلیل محدودیت منابع روی تراشه، قابلیت توسعه‌پذیری و سفارشی شدن را ندارند [۳, ۴]. علاوه بر این، هزینه طراحی سیستم‌های دسته‌بند سخت‌افزاری بسیار زیاد بوده و نسبت کارایی به هزینه در آن‌ها پایین است. بدین دلیل، اخیراً دسته‌بندهای نرم‌افزاری مورد توجه قرار گرفته‌اند [۸-۵]. دسته‌بندهای نرم‌افزاری متداول نیز با وجود توسعه‌پذیری، به دلیل پایین بودن سرعت پردازش متوالی دستورات در پردازنده‌های مرکزی، در شبکه‌های با پهنای باند زیاد کارایی مناسبی ندارند. بنابراین چالش مهمی تحت عنوان تسریع دسته‌بندهای نرم‌افزاری بسته‌های IP، انجام پژوهش‌های مرتبط روش‌های نرم‌افزاری و سخت‌افزاری جهت تسریع اجرای الگوریتم‌های دسته‌بندی بسته‌ها را ایجاب نموده است.

برخی از الگوریتم‌های دسته‌بندی بسته ساختار مناسبی برای موازی‌سازی در حد بسیار بالا دارند اما برخی دیگر به دلیل وابستگی‌های داده‌ای و کنترلی قابل موازی‌سازی نیستند. Taylor [۹] الگوریتم‌های دسته‌بندی مبتنی بر نرم‌افزار را در چهار کلاس زیر طبقه‌بندی کرده است:

جستجوی کلی: این الگوریتم‌ها کلیه عناصر درون یک لیست را بررسی می‌کنند تا آرگومان جستجو پیدا شود. جستجوی کلی برای کاهش افزونگی در مجموعه فیلترها تلاشی نمی‌کند. همین امر باعث محاسبات اضافی قابل‌توجهی در این الگوریتم می‌شود [۹].

تجزیه: الگوریتم‌های مبتنی بر تجزیه معمولاً شامل دو گام می‌باشند: در گام اول جستجویی به‌صورت مجزا بر اساس هر فیلد در مجموعه فیلترها انجام می‌شود. در گام دوم، نتایج تمام جستجوهای انجام‌شده روی فیلدهای متفاوت با هم تطبیق داده شده و ادغام می‌شود [۱۰]. طبق توضیح فوق واضح است که این الگوریتم‌ها قابلیت موازات بالایی دارند. با این وجود، به‌دلیل حجیم بودن ساختارهای داده‌ای لازم، از لحاظ میزان حافظه مصرفی مطلوب نیستند.

فضای چندتایی: در روش فضای چندتایی، فیلترهای موجود بر حسب تعداد بیت‌های معین‌شده در فیلدهای انتخابی جهت جستجو، تقسیم می‌شوند. بسته‌ها با استفاده از الگوریتم جستجوی ساده، با چندتایی‌های ایجادشده به‌صورت دقیق تطبیق داده شده و وارسی می‌شوند [۱۱]. این الگوریتم‌ها قابلیت موازی‌سازی را داشته و از ساختار داده کم حجم‌تری استفاده می‌کنند.

درخت تصمیم: آخرین رده، الگوریتم‌های درخت تصمیم می‌باشند. در این الگوریتم‌ها، مجموعه فیلترها بر اساس الگوهای دودویی در فیلدهای فیلترها، در درخت‌های جستجوی ذخیره می‌شوند. در ساخت و جستجوی درخت تصمیم بر اساس چندین فیلد، یک درخت تصمیم‌گیری ایجاد می‌شود که در آن، برگ‌های درخت حاوی فیلتری مشخص یا زیرمجموعه‌ای از فیلترها هستند. جهت یافتن بهترین فیلتر منطبق با بسته ورودی پیمایشی بر اساس محتویات دودویی فیلدهای موردنظر بر روی درخت جستجوی فیلترها انجام می‌شود. مزیت بالای این الگوریتم‌ها قابلیت موازی عمل کردن آن‌ها است [۱۲].

به‌منظور اجرای موازی یک الگوریتم می‌توان از پردازنده‌های چندهسته‌ای [۱۳] یا پردازنده‌ی گرافیکی استفاده نمود. اخیراً با توسعه واحد پردازش گرافیکی و به‌کارگیری پردازنده‌های متعدد در آن‌ها، استفاده از واحد پردازش گرافیکی برای انجام محاسبات موازی محبوبیت فراوانی یافته است. پردازنده گرافیکی شامل هسته‌های محاسباتی متعدد است که باعث شده است به‌عنوان سخت‌افزاری ارزان قیمت در پردازش‌های موازی استفاده شود. اخیراً، تحقیقات متعددی در راستای پیاده‌سازی موازی الگوریتم‌های دسته‌بندی بسته روی پردازنده گرافیکی مطرح شده است. تحقیقات مذکور تنها روی الگوریتم‌های جستجوی کلی و تجزیه انجام شده‌اند. همان‌گونه که قبلاً توضیح داده شد الگوریتم‌های مذکور به‌ویژه الگوریتم‌های مبتنی بر تجزیه به دلیل مصرف حافظه زیاد چندان مطلوب نیستند. در مقابل، تاکنون برای موازی‌سازی الگوریتم‌های درخت تصمیم و فضای چندتایی که مصرف حافظه متعادل‌تری نیز دارند تحقیقات قابل توجهی انجام نشده است.

در همین راستا، در این مقاله برای نخستین بار نسخه موازی الگوریتم درخت سلسله‌مراتبی[3] [۱۴] را برای دسته‌بندی بسته‌ها ارائه نموده‌ایم. الگوریتم درخت سلسله‌مراتبی یکی از روش‌های دسته‌بندی مبتنی بر درخت تصمیم می‌باشد. مهم‌ترین نوآوری‌های این مقاله به شرح زیر است:

- با در نظر گرفتن سلسله مراتب حافظه در معماری پردازنده گرافیکی، شش سناریوی متفاوت برای دسته‌بندی موازی بسته‌ها توسط الگوریتم درخت سلسله‌مراتبی ارائه شده است.

- برای نخستین بار پیچیدگی زمانی، حافظه‌ای و پردازنده سناریوهای پیشنهادی به‌صورت تحلیلی جهت پیش‌بینی کارایی آن‌ها ارائه و با هم مقایسه شده است.
- زمانی که تعداد فیلترها زیاد باشد و درخت سلسله‌مراتبی در حافظه اشتراکی جای نگیرد، لازم است مجموعه فیلتر شکسته شده و زیردرخت‌های حاصل بدون /با مجموعه فیلترهای نظیرشان از حافظه سراسری در حافظه اشتراکی بلوک‌های فعال کپی شوند. در این شرایط، کارایی سناریوهای مبتنی بر حافظه اشتراکی، به‌دلیل پردازش بسته‌های تکراری در زیردرخت‌های مذکور، از کارایی سناریویی که کل ساختار درخت سلسله‌مراتبی را در حافظه سراسری نگه می‌دارد کم‌تر بوده و با افزایش تعداد زیردرخت‌ها کاهش می‌یابد.
- کارایی سناریوهایی که از حافظه اشتراکی استفاده می‌کنند با تعداد SM[۲]ها، تعداد پردازش‌های نخی ممکن روی هر هسته، تعداد بلوک‌های فعال، تعداد زیردرخت‌های حاصل از تقسیم درخت سلسله‌مراتبی اصلی و تعداد فیلترهای موجود در هر زیردرخت متناسب است.
- کاراترین سناریو، سناریویی است که درخت سلسله‌مراتبی و مجموعه فیلترهای متناظر را بدون شکستن به زیردرخت‌ها، با هم در حافظه اشتراکی جای می‌دهد. در این حالت نرخ گذرداد سناریو بهینه روی پردازنده گرافیکی GTX 750 برابر با MPPS ۳۶/۳۴ است که می‌تواند روی پردازنده گرافیکی K20 به MPPS ۱۷۷/۱۵۷ برسد.

ساختار مقاله به‌صورت زیر سازمان‌دهی شده است. ابتدا در بخش دو ساختار واحد پردازش گرافیکی و ادامه، نحوه کار الگوریتم درخت سلسله‌مراتبی بیان می‌شود. در بخش سوم، کارهای پیشین در زمینه موازی‌سازی الگوریتم‌های دسته‌بندی بسته روی واحد پردازش گرافیکی بررسی می‌شود. سناریوهای پیشنهادی جهت موازی‌سازی الگوریتم درخت سلسله‌مراتبی روی واحد پردازش گرافیکی در بخش چهارم ارائه شده و نحوه پیاده‌سازی، تحلیل پیچیدگی سناریوها در بخش ۵ و ارزیابی کارایی هر یک پس از معرفی شاخص‌های کارایی، در بخش ۶ توضیح داده می‌شود. بخش پایانی مقاله به نتیجه‌گیری و ارائه راهکارهایی جهت پیشرفت تحقیق موجود اختصاص می‌یابد.

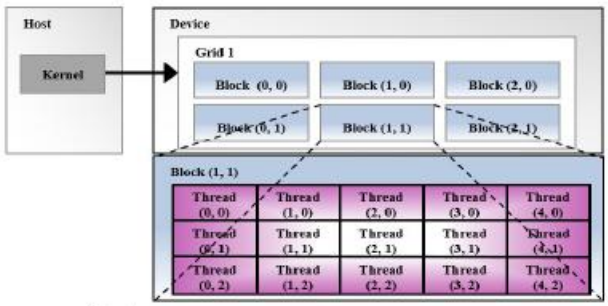
۲- ابزارها و الگوریتم‌های مرتبط

در این بخش ابتدا ساختار واحد پردازش گرافیکی و حافظه موجود در آن بررسی می‌شود. سپس، روش ایجاد درخت الگوریتم سلسله‌مراتبی و نحوه دسته‌بندی بسته‌ها توسط این الگوریتم توضیح داده می‌شود.

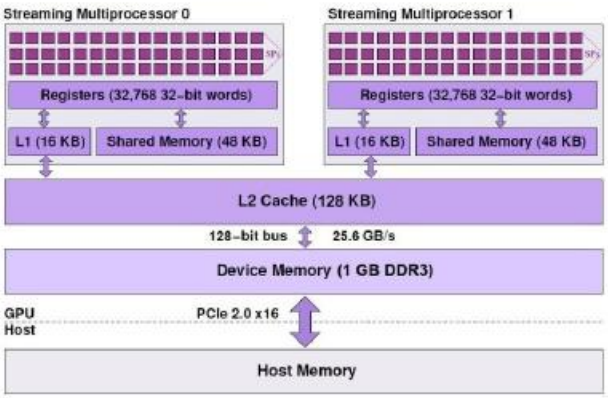
۲-۱- واحد پردازش گرافیکی

واحد پردازش گرافیکی سیستمی اختصاصی برای نمایش تصاویر گرافیکی در کامپیوترهای شخصی است. با انتشار بسته‌های توسعه نرم‌افزار روی این واحد، از سوی سازندگان بزرگی همچون Nvidia [۱۵] و ATI [۱۶] استفاده از واحد پردازش گرافیکی به‌عنوان یک واحد محاسباتی موازی قدرتمند به جای واحد پردازش مرکزی به‌عنوان رویکردی کلیدی در تسریع پردازش‌های محاسباتی پذیرفته شده است. دلیل اصلی این تحول بزرگ محاسباتی آن است که معماری واحد پردازش گرافیکی، مخصوصاً برای انجام محاسبات فشرده و عملیات موازی (موردنیاز برای نمایش تصاویر گرافیکی) طراحی شده است. بر این اساس، شرکت Nvidia در سال ۲۰۰۶ میلادی جهت اجرای محاسبات غیرگرافیکی روی پردازنده گرافیکی، بستر نرم‌افزاری بنام کودا[۵] عرضه کرده است [۱۵]. کودا امکاناتی را فراهم می‌آورد تا به واسطه آن‌ها برنامه‌نویسان بتوانند از قابلیت‌های سخت‌افزاری کارت‌های گرافیکی در برنامه‌های غیرگرافیکی خود بهره برده و به کمک قابلیت‌های محاسباتی آن‌ها، سرعت اجرای الگوریتم‌های پیچیده خود را افزایش دهند.

تاکنون، چندین کار از جمله [۱۸,۱۷] به استفاده از بستر کودا برای پیاده‌سازی موازی توابع شبکه از قبیل جستجوی IP در جداول مسیریابی با هدف دستیابی به گذرداد بالاتر، پرداخته‌اند. از این ابزار برای اجرای موازی الگوریتم‌های ژنتیک [۱۹]، شبکه‌های عصبی [۲۰] و برخی از الگوریتم‌های دیگر در حوزه هوش مصنوعی [۲۱] نیز استفاده شده است. همچنین در زمینه رمزنگاری، برای فشرده‌سازی پایگاه‌های داده و تسریع الگوریتم‌های رمزگذاری از قابلیت‌های برنامه‌نویسی موازی در بستر کودا استفاده شده است [۲۳,۲۲].



شکل ۱: بلوک و نخ در واحد پردازش گرافیکی [۲۴]



شکل ۲: ساختار واحد پردازش گرافیکی **Geforce 425M** [۱۵]

از دیدگاه برنامه‌نویسی، در بستر کودا دو پردازه درگیر در محاسبات وجود دارد: پردازه میزبان و پردازه دستگاه. پردازه میزبان، روی واحد پردازش مرکزی اجرا می‌شود و درواقع برنامه اصلی را اجرا می‌کند؛ درحالی‌که پردازه دستگاه روی واحد پردازش گرافیکی اجرا می‌شود. هر برنامه‌ای که در کودا نوشته می‌شود ممکن است از چندین هسته تشکیل شده باشد. هر هسته توسط یک گرید که خود از چندین بلوک تشکیل شده، اجرا می‌گردد. هر بلوک نیز از چندین پردازش نخی تشکیل می‌شود. درواقع پردازش‌های نخی عهده‌دار اجرای برنامه هستند. شکل ۱ ساختار یک هسته را با شش بلوک که هر یک دارای ۱۵ نخ است، نشان می‌دهد.

یکی از پردازنده‌های گرافیکی استفاده‌شده در این مقاله از نوع GT 425M می‌باشد که از دو SM که هر یک دارای ۴۸ SP است، تشکیل شده است. شکل ۲ ساختار سخت‌افزاری واحد پردازش گرافیکی مدل GT 425M را نشان می‌دهد. هر واحد پردازش گرافیکی دارای انواع مختلفی حافظه ازجمله، حافظه سراسری، حافظه ثابت، حافظه بافت، ثبات و حافظه مشترک است.

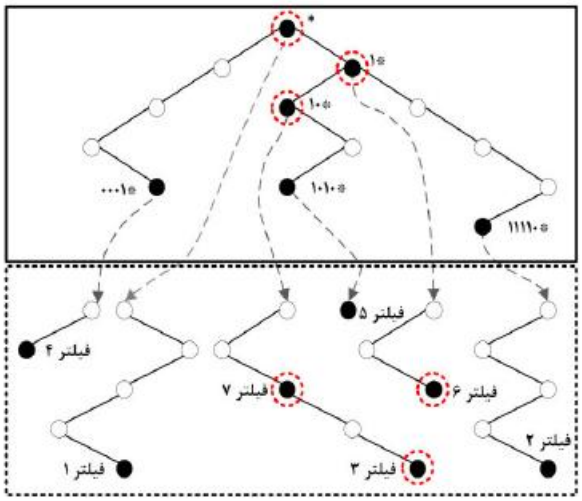
۲-۲- الگوریتم درخت سلسله‌مراتبی

یکی از الگوریتم‌های مبتنی بر درخت تصمیم، الگوریتم درخت سلسله‌مراتبی است. در الگوریتم درختی سلسله‌مراتبی برای دسته‌بندی بسته‌ها، از آدرس IP مبدأ و آدرس IP مقصد برای ساختن درخت تصمیم‌گیری استفاده می‌شود. به‌عنوان مثال، برای هفت فیلتر تصادفی موجود در جدول ۱، درخت سلسله‌مراتبی متناظر در شکل ۳ نشان داده شده است. در این جدول اولویت فیلترها از بالا به پایین کاهش می‌یابد. نحوه تشکیل درخت بدین صورت است که ابتدا آدرس IP مبدأ فیلترها به‌صورت بیت به بیت خوانده می‌شود؛ سپس، با مشاهده هر بیت "۰" یا "۱" به ترتیب سمت چپ یا راست درخت مورد پیمایش قرار می‌گیرد. وقتی که علامت "*" دیده شد یعنی تکمیل درخت بر اساس آدرس IP مبدأ آن فیلتر به پایان رسیده است. حال، با استفاده از یک اشاره‌گر به درخت مقصد برای تکمیل درخت از آدرس IP مقصد استفاده می‌شود. نحوه ساختن درخت توسط آدرس IP مقصد دقیقاً شبیه به نحوه ساختن درخت توسط آدرس IP مبدأ است [۱۴].

با دریافت یک بسته ورودی ابتدا آدرس‌های IP مبدأ و مقصد، آدرس درگاه مبدأ و مقصد و پروتکل از سرآیند بسته استخراج می‌شود. مثلاً در چندتایی (۰۱۱۱۰، ۱۰۰۰۰، ۵۴۲، ۲۷۸۸، ۴)، آدرس‌های مذکور از سرآیند بسته استخراج شده و به ترتیب از سمت راست به چپ نوشته شده‌اند. برای دسته‌بندی بسته فوق، ابتدا پیمایش از ریشه درخت سلسله‌مراتبی بر اساس بیت‌های فیلد آدرس مبدأ شروع می‌شود. در مسیر پیمایش روی درخت مبدأ، اگر از گره‌ای اشاره‌گر به درخت آدرس IP مقصد وجود داشته باشد، درون یک صف قرار می‌گیرد؛ با اتمام پیمایش درخت آدرس IP مبدأ، درخت‌های متناظر با گره‌های موجود در صف، بر اساس بیت‌های فیلد آدرس IP مقصد بسته پیمایش

جدول ۱: مثال از مجموعه فیلتر

فیلتر	IP مبدأ	IP مقصد	درگاه مبدأ	درگاه مقصد	پروتکل
فیلتر ۱	*	۱۰۰۱*	۵۳	۱۰۲۴-۶۵۵۳۵	۶
فیلتر ۲	۱۱۱۱۰*	۰۱۰۱*	۵۳	۴۴۳	۱۷
فیلتر ۳	۱۰*	۰۱۱۱*	۰-۶۵۵۳۵	۵۶۳۲	۴
فیلتر ۴	۰۰۰۱*	۰*	۰-۶۵۵۳۵	۲۵	۱۷
فیلتر ۵	۱۰۱۰*	*	۵۳	۴۴۳	۶
فیلتر ۶	۱*	۱۰*	۰-۶۵۵۳۵	۲۷۸۸	۴
فیلتر ۷	۱۰*	۰۱*	۵۳	۲۵	۱۷



شکل ۳: ساختار درخت الگوریتم سلسله‌مراتبی

می‌شوند. در صورت وجود یک فیلتر در مسیر پیمایش در درخت مقصد، انطباق دقیق بقیه فیلدهای بسته با آن فیلتر بررسی می‌شود و در صورت انطباق همه فیلدها، آن فیلتر ذخیره می‌شود. زمانی که در مسیر پیمایش چندین فیلتر با اولویت‌های مختلف با اطلاعات سرآیند بسته ورودی تطبیق داشته باشد، فیلتری که دارای بالاترین اولویت است به‌عنوان خروجی انتخاب می‌شود. مثال مطرح‌شده، از میان فیلترهای منطبق با بسته ورودی، فیلتر ۳ دارای بالاترین اولویت است. پیچیدگی زمانی الگوریتم درخت سلسله‌مراتبی برابر با $O(W^d)$ است که W حداکثر طول پیشوند هر فیلد و d تعداد فیلدهای مورد بررسی می‌باشند. همچنین پیچیدگی ذخیره‌سازی درخت $O(NdW)$ است که N تعداد فیلترهای دسته‌بند است.

۳- مروری بر کارهای پیشین

مروری بر تحقیقات اخیر در زمینه موازی‌سازی الگوریتم‌های دسته‌بندی بسته‌ها نشان می‌دهد که نیاز به انجام تحقیقات بیش‌تر در ابعاد مختلف موازی‌سازی وجود دارد. معدود کارهای انجام‌شده در این زمینه به بررسی امکان موازات الگوریتم‌ها و موازی‌سازی چند الگوریتم محدود شده‌اند. در ادامه به بررسی دستاوردهای مهم تحقیقات اخیر می‌پردازیم.

در مقاله A. Nottingham و همکاران موازی‌سازی تعدادی از الگوریتم‌های دسته‌بندی بسته به‌صورت نظری موردبررسی قرار گرفته

است [۲۵]. آن‌ها در مقاله خود، به بررسی امکان پیاده‌سازی موازی این الگوریتم‌ها در واحد پردازش گرافیکی پرداخته‌اند.

C.Hung و همکاران دو الگوریتم دسته‌بندی BPF[۸] را با هشت سناریوی مختلف و RFC [۹] را با چهار سناریوی مختلف در استفاده از حافظه واحد پردازش گرافیکی پیاده‌سازی و ارزیابی کرده‌اند [۲۴]. آن‌ها تنها از سه فیلتر به‌عنوان مجموعه فیلتر برای دسته‌بندی ۶۵ میلیون بسته تصادفی جهت پیاده‌سازی و ارزیابی استفاده نمودند. نتایج این آزمایش‌های محدود، عملکرد بهتر الگوریتم BPF را نسبت به RFC نشان می‌دهد.

Y.Deng و همکاران یک مدل ترکیبی را با تجمیع هسته‌های واحد پردازش مرکزی و واحد پردازش گرافیکی روی یک تراشه، که هر دو به یک ساختار حافظه معمولی متشکل از حافظه SRAM و حافظه معمولی DRAM دسترسی داشتند، ارائه کرده‌اند [۲۶]. الگوریتم مورد استفاده در پژوهش آن‌ها، الگوریتم جستجوی خطی است. مقایسه دو روش پردازش ترکیبی و پردازش متوالی روی پردازنده مرکزی نشان می‌دهد که مدل ترکیبی دارای قدرت پردازشی بیش‌تر و تأخیر کم‌تر است.

در مقاله K. Kang، پیاده‌سازی الگوریتم‌های جستجوی خطی و DBS[۱۰] که مبتنی بر جدول درهم‌سازی است، در واحد پردازش گرافیکی و واحد پردازش مرکزی موردبررسی قرار گرفته است [۲۷]. ارزیابی صورت گرفته در مورد کارایی الگوریتم‌های فوق در مجموعه فیلتر ساختگی FW4 که توسط ابزار ClassBench [۲۸] تولید شده است، با تعداد مختلفی از ۱۰۰ تا ۱۰۰k فیلتر، نشان می‌دهد که مدت‌زمان دسته‌بندی بسته‌های آزمون در جستجوی خطی موازی‌شده به میزان قابل‌توجهی کم‌تر از الگوریتم DBS است.

S. Zhou و همکاران در سال ۲۰۱۴ میلادی، از یک الگوریتم مبتنی بر تجزیه، بنام بردار بیت[۱۱] استفاده کردند [۲۹]. الگوریتم ارائه‌شده از k پردازش نخی برای دسته‌بندی بسته‌ها استفاده می‌کند. در این الگوریتم موازی‌شده، با فرض داشتن n فیلتر، در صورت ورود یک بسته هر نخ مسئول بررسی تطبیق آن بسته با تعداد $\frac{n}{k}$ فیلتر می‌باشد. نتیجه این مرحله، یافتن بهترین فیلتر منطبق با بسته در زیرمجموعه فیلترهای مذکور است. این نتیجه به‌صورت یک بردار بیتی که در آن بیت متناظر با فیلتر مذکور مقدار یک دارد در گام بعدی مورد استفاده قرار می‌گیرد. با عطف k بردار بیتی حاصل، نتیجه نهایی که بهترین فیلتر منطبق با بسته است مشخص می‌شود. نتایج تحقیق مذکور، بیانگر بالا بودن کارایی الگوریتم پیشنهادی نسبت به نسخه متوالی آن روی واحد پردازش مرکزی است.

M. Varvello و همکاران در سال ۲۰۱۴ از واحد پردازش گرافیکی برای افزایش سرعت الگوریتم‌های جستجوی خطی و جستجوی فضای چندتایی استفاده کردند [۳۰]. در موازی‌سازی پیشنهادی توسط آن‌ها، سرعت اجرای الگوریتم جستجوی خطی و الگوریتم جستجوی فضای چندتایی به ترتیب ۷ و ۱۱ برابر افزایش داشته است.

J. Zheng و همکاران نیز در سال ۲۰۱۵ میلادی الگوریتم دسته‌بندی برش هوشمندانه سلسله‌مراتبی[۱۲] را در واحد پردازش گرافیکی پیاده‌سازی کردند [۳۱]. نتایج ارزیابی الگوریتم مذکور نشان می‌دهد که میزان گذرداد الگوریتم دسته‌بندی برش هوشمندانه سلسله‌مراتبی در صورت موازی‌سازی روی واحد پردازش گرافیکی تا حد سه برابر افزایش می‌یابد.

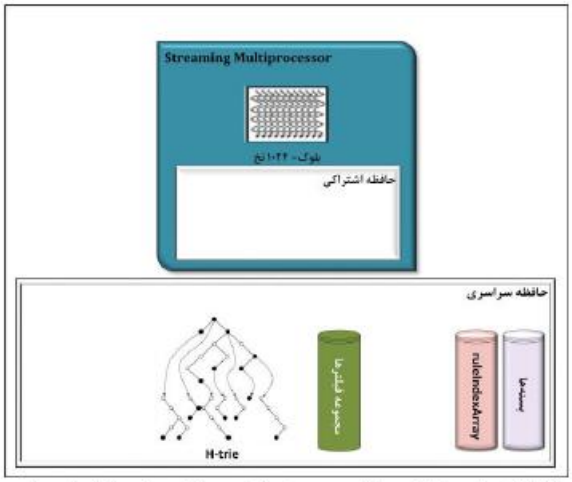
با توجه به کارهای انجام‌گرفته، نحوه استفاده از موازات و انواع مختلف ماژول‌های حافظه برای دسته‌بندی موازی بسته‌ها در هیچ‌یک از تحقیقات اخیر به‌صورت دقیق بررسی نشده و پیچیدگی زمانی، حافظه‌ای و پردازنده سناریوهای پیشنهادی تحلیل نشده است. همچنین، تاکنون تعداد کمی از الگوریتم‌هایی که از لحاظ میزان حافظه مصرفی مطلوب هستند جهت موازی‌سازی روی واحد پردازش گرافیکی و ارتقاء سرعت مورد بررسی قرار گرفته‌اند. الگوریتم‌های دسته‌بندی مبتنی بر ساختار درخت تصمیم مهم‌ترین این الگوریتم‌ها هستند. الگوریتم درخت سلسله‌مراتبی به‌عنوان مهم‌ترین الگوریتم از این رده، در صورت به کار بستن روشی که بتواند تأخیر پردازش را در پیاده‌سازی نرم‌افزاری با در نظر داشتن حداکثر موازات ممکن در واحد پردازش گرافیکی کاهش دهد، می‌تواند کارایی زمانی لازم را جهت دسته‌بندی حجم ترافیک بسیار بالا از خود نشان دهد.

بدین دلیل، در این مقاله برای نخستین بار سناریوهای کارآمدی برای افزایش نرخ گذرداد و میزان تسریع در پیاده‌سازی نرم‌افزاری الگوریتم دسته‌بندی مبتنی بر درخت سلسله‌مراتبی روی واحد پردازش گرافیکی ارائه نموده‌ایم. همچنین، با هدف تحلیل دقیق کارایی سناریوهای پیشنهادی، پیچیدگی آن‌ها را از دیدگاه زمان محاسبات و تعداد تراکنش‌های حافظه‌ای به‌صورت تحلیلی بررسی نموده‌ایم. نتایج ارزیابی سناریوها به کمک آزمون، ضمن تطبیق با پیچیدگی‌های پیش‌بینی شده در روابط تحلیلی، کارایی آن‌ها را تأیید می‌نماید.

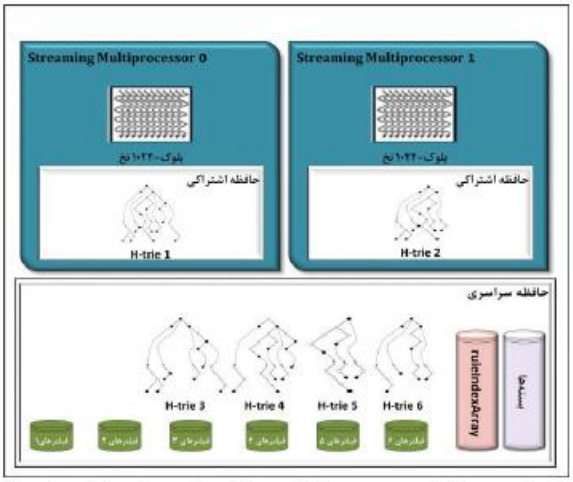
در ادامه به بررسی نحوه دسته‌بندی بسته‌ها توسط الگوریتم درخت سلسله‌مراتبی پرداخته می‌شود و ساختار آن از منظر موازات موردبررسی قرار می‌گیرد.

۴- پیاده‌سازی موازی درخت سلسله‌مراتبی

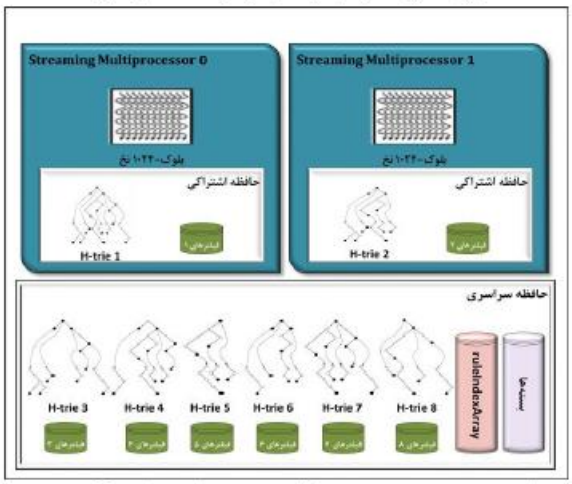
در این بخش، نحوه موازی‌سازی درخت سلسله‌مراتبی با استفاده از واحد پردازش گرافیکی بررسی می‌شود. همان‌طور که در بخش دوم مقاله بیان شد، واحد پردازش گرافیکی دارای چندین نوع حافظه با تأخیرهای متفاوت دسترسی است. ذخیره‌سازی مجموعه فیلترها با سرآیند بسته‌ها در هر یک از این حافظه‌ها، به دلیل زمان دسترسی متفاوتی که پردازش‌های نخی در دسترسی به هر یک دارند، روی زمان دسته‌بندی بسته‌ها بسیار تأثیرگذار است. بنابراین، در ادامه به بررسی پیکربندی‌های متفاوت استفاده از حافظه‌های واحد پردازش گرافیکی می‌پردازیم.



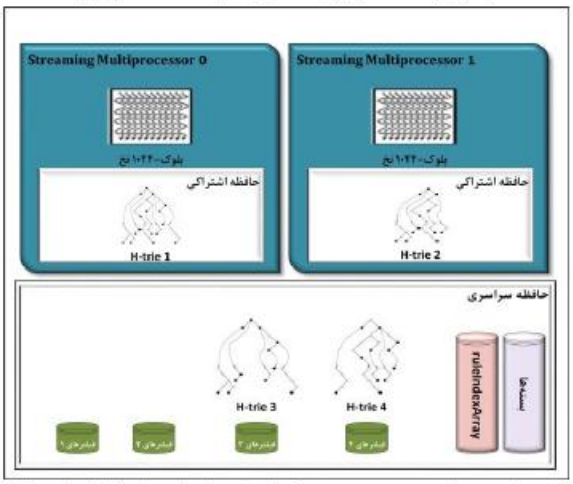
(الف) سناریو اول: درخت به‌صورت یکپارچه- گروه‌بندی نخ‌ها در یک بلوک- درخت و مجموعه فیلتر در حافظه سراسری



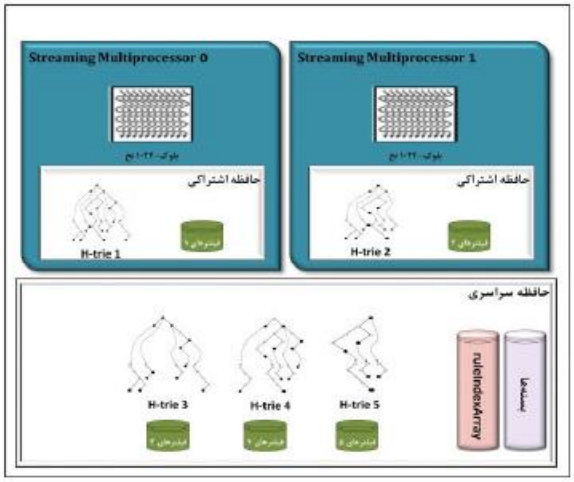
(ب) سناریو دوم: تقسیم مجموعه فیلتر به بخش‌های مساوی-گروه‌بندی نخ‌ها در دو بلوک- انتقال درخت فیلترها به حافظه اشتراکی.



(ج) سناریو سوم: تقسیم مجموعه فیلتر به بخش‌های مساوی-گروه‌بندی نخ‌ها در دو بلوک - انتقال درخت و مجموعه فیلتر به حافظه اشتراکی.



(د) سناریو چهارم: تقسیم مجموعه فیلتر با هدف استفاده کامل از حافظه اشتراکی- گروه‌بندی نخ‌ها در دو بلوک -انتقال درخت فیلترها به حافظه اشتراکی



(ه) سناریو پنجم، تقسیم مجموعه فیلتر با هدف استفاده کامل از حافظه اشتراکی- گروه‌بندی نخ‌ها در دو بلوک -انتقال درخت و مجموعه فیلتر به حافظه اشتراکی

شکل ۴: سناریوهای پیشنهادی جهت استفاده از حافظه‌های پردازنده گرافیکی در پردازنده گرافیکی **GT 425M**

جدول ۲: مشخصات سناریوها

سناریو	درخت		فیلترها		پر بودن حافظه اشتراکی
	حافظه سراسری	حافظه اشتراکی	حافظه سراسری	حافظه اشتراکی	
سناریو اول	×		×		
سناریو دوم		×	×		
سناریو سوم		×		×	
سناریو چهارم		×	×		×
سناریو پنجم		×		×	×

برای موازی‌سازی عمل دسته‌بندی بسته‌ها لازم است که ابتدا درخت متناظر با فیلترهای موجود توسط واحد پردازش مرکزی ساخته شود. سپس ساختار درخت، فیلترها، سرآیند بسته‌ها و آرایه‌ای جهت ذخیره‌سازی نتایج دسته‌بندی از حافظه میزبان به حافظه سراسری پردازنده گرافیکی کپی می‌شوند. در صورت انتخاب حافظه اشتراکی واحد پردازش گرافیکی برای ذخیره‌سازی مجموعه فیلترها و ساختار درخت تصمیم سلسله‌مراتبی، اگر اندازه حافظه موردنیاز درخت فیلترها بزرگ‌تر از اندازه حافظه اشتراکی باشد و درخت به‌صورت یکپارچه در حافظه اشتراکی جای نگیرد، لازم است درخت به بخش‌های کوچک‌تر تقسیم شده و هر بخش جداگانه به حافظه اشتراکی منتقل شود. در این راستا سناریوهای متفاوتی می‌توان ارائه نمود. این سناریوها در جدول ۲ معرفی شده است.

در حالت اول که به‌عنوان سناریو اول پیشنهاد می‌شود، ساختار درخت به‌صورت یکپارچه به همراه مجموعه فیلتر متناظر، در حافظه سراسری پردازنده گرافیکی قرار می‌گیرد اما در حالت دوم برای استفاده از حافظه اشتراکی، درخت فیلترها به زیردرخت‌هایی با اندازه کوچک‌تر یا مساوی با اندازه حافظه اشتراکی شکسته می‌شود. در شکل ۴ نحوه استفاده از حافظه‌های سراسری و اشتراکی پردازنده گرافیکی GT 425M توسط سناریوهای مذکور نمایش داده شده است. لازم به‌ذکر است در این مقاله، از پردازنده گرافیکی GTX 750 نیز برای اجرای سناریوهای مذکور استفاده شده است. جدول ۳ مشخصات هر دو

الگوریتم ۱: پیاده‌سازی درخت سلسله‌مراتبی با استفاده از حافظه سراسری

Input :rules R, H-trie T, headers H
Output :$ruleIndexArray$
1: global memory ← host memory(R,T,H)
2: **for all** $i \in \left[0,\frac{|H|}{1024}\right]$ **do**
3: $tid \leftarrow threadIdx + i * 1024$, $rIdx \leftarrow Null$
4: **if** $tid < |H|$ **then**
5: $P \leftarrow$ ReadPacket(tid)
6: $rIdx \leftarrow$ Classify(P,T,R)
7: **if** $rIdx \neq Null$ **then**
8: $ruleIndexArray(tid,rIdx)$
9: **end if**
10: **end if**
11: $i \leftarrow i + 1$
12: **end for**

پردازنده گرافیکی را نشان می‌دهد. با توجه به اینکه تعداد SMها در پردازنده GTX 750 دو برابر تعداد SMها در پردازنده GT 425M است، در اجرای سناریوهای دوم تا پنجم بر روی این پردازنده، از چهار بلوک همزمان می‌توان استفاده نمود. نحوه استفاده از حافظه اشتراکی در ادامه به تشریح سناریوهای پیشنهادی می‌پردازیم.

- سناریو اول: در این سناریو ساختار درخت، فیلترها و سرآیند بسته‌ها به‌همراه یک آرایه به طول تعداد بسته‌ها جهت ذخیره نتایج دسته‌بندی در حافظه سراسری پردازنده گرافیکی نگه‌داری می‌شوند. دلیل نگه‌داری این مقادیر در حافظه سراسری، قابل دسترس بودن آن برای تمام نخ‌ها است. این سناریو در شکل ۴-الف نشان داده شده است. با توجه به شبه‌کد الگوریتم ۱ ورودی این الگوریتم مجموعه فیلترها R، ساختار درخت T و بسته‌ها H است. هسته، اندیس بهترین فیلتر منطبق‌شده با هر بسته را در آرایه خروجی $ruleIndexArray$ ذخیره می‌کند. پس از کپی ساختار درخت سلسله‌مراتبی، مجموعه فیلترها و بسته‌ها از حافظه میزبان به حافظه سراسری پردازنده گرافیکی (خط ۱)، عملیات دسته‌بندی شروع می‌شود (خط‌های ۲-۱۲). عملیات اصلی دسته‌بندی در حلقه for انجام می‌شود. هر نخ از اندیس خود و آدرس شروع بسته‌ها برای انتخاب یک بسته و دسته‌بندی آن استفاده می‌کند (خط ۳). درصورتی‌که اندیس به‌دست‌آمده، از تعداد کل بسته‌ها کم‌تر باشد، نخ مذکور یک بسته را از حافظه

الگوریتم ۲: پیاده‌سازی درخت سلسله‌مراتبی با استفاده از حافظه سراسری و اشتراکی

Input :rules R, H-trie T, headers H
Output :$ruleIndexArray$
Data :rules R', tree T' **in shared memory**
Pre-processing:// performed within host (CPU)
1: break R into several R_i according to the scenario
2: construct all $T_i s$ from corresponding $R_i s$
Transfer: // from host to global memory of GPU
3: global memory ← host memory(R_i,T_i,H)// for all i
4: $treeOffset \leftarrow 0$
5: $nTrees \leftarrow$ number of T_is , $i \leftarrow 0$
6: **while** $treeOffset < nTrees$ **do**
7: $T' \leftarrow$ global memory$(T_i,blockIdx,treeOffset)$
8: $R' \leftarrow$ global memory$(R_i,blockIdx,treeOffset)$
9: $SyncThreads()$
10: **for all** $j \in \left[0,\frac{|H|}{1024}\right]$ **do**
11: $tid \leftarrow threadIdx + j * 1024$, $rIdx \leftarrow Null$
12: **if** $tid < |H|$ **then**
13: $P \leftarrow$ ReadPacket(tid)
14: $rIdx \leftarrow$ Classify(P,T',R')
15: **if** $rIdx \neq Null$ **then**
16: AtomicMax$(ruleIndexArray(tid),rIdx)$
17: **end if**
18: **end if**
19: $j \leftarrow j + 1$
20: **end for**
21: $SyncThreads()$
22: $treeOffset \leftarrow treeOffset + |Blocks|$
23: **end while**

جدول ۳: مشخصات سیستم‌ها

CPU		Intel(R) Core™ i7Q 740 @ 1.73GHz		
RAM		4 GB		
Operation System		Windows 7 Ultimate, 64-bit (Service Pack 1)		
GPU	Model	Nvidia GT 425M	Nvidia GTX 750	Nvidia K20
	Architecture	Fermi	Maxwell	Kepler
	Cuda Cores	96(48 CUDA Cores/SM)	512(128 CUDA Cores/SM)	2496(192 CUDA Cores/SM)
	Graphics clock	560 MHz	1020MHz	705MHz
	Memory Clock (effective)	1.6 GHz	5GHz	5.2 GHz
	Processing Power (GFLOPS)	215.04	1044	4106
	Memory bandwidth	25.6 GB/s	80 GB/s	208 GB/s
	SMs	2	4	13
	Bus width (bit)	128	128	320

سراسری برداشته و عمل دسته‌بندی را بر اساس درخت T و مجموعه فیلترهای R که هر دو در حافظه سراسری هستند انجام می‌دهد. ازآنجاکه ممکن است بسته ورودی با چند فیلتر تطبیق داشته باشد، اندیس متناظر با بهترین فیلتر تطبیقی به‌عنوان خروجی در خانه متناظر با بسته، در آرایه $ruleIndexArray$ ذخیره می‌شود. در هر تکرار حلقه for، ۱۰۲۴ نخ به‌صورت مجزا ۱۰۲۴ بسته را دسته‌بندی می‌کنند و این عملیات تا زمانی که تمام بسته‌ها دسته‌بندی شوند تکرار می‌گردد. در این سناریو جهت ایجاد شرایط پیاده‌سازی یکسان با سناریوهای دیگر و فراهم نمودن امکان مقایسه کارایی، تنها یک بلوک با ۱۰۲۴ نخ، استفاده می‌شود.

در حالت دوم برای استفاده از حافظه اشتراکی، پس از شکستن مجموعه فیلتر و ایجاد زیردرخت‌هایی با اندازه کوچک‌تر یا مساوی با اندازه حافظه اشتراکی، آن‌ها را با/بدون مجموعه فیلتر متناظر به‌صورت متوالی به حافظه اشتراکی انتقال داده و بسته‌ها را بر اساس آن‌ها دسته‌بندی می‌کنیم. در این حالت، می‌توان سناریوهای دوم تا پنجم را بر اساس جای‌گشت‌های جدول ۲ برای انتقال زیردرخت‌ها و مجموعه فیلترهای متناظر به حافظه اشتراکی تعریف نمود. مکانیسم کلی استفاده از حافظه اشتراکی به‌صورت زیر است. با توجه به محدودیت فضای حافظه اشتراکی و تعداد بلوک‌ها، ابتدا فیلترها به چندین قسمت تقسیم می‌شوند و سپس با هر تعداد فیلتر یک درخت سلسله‌مراتبی ایجاد می‌شود. شبه‌کد این مکانیسم در الگوریتم ۲ نشان داده شده است. ورودی این الگوریتم مجموعه فیلترهای R، ساختار درختی T و بسته‌های H است. پردازه هسته، اندیس بهترین فیلتر منطبق‌شده با هر بسته را در آرایه خروجی $ruleIndexArray$ ذخیره می‌کند. با توجه به محدودیت حافظه اشتراکی، در خطهای ۱ و ۲، مجموعه فیلتر به زیرمجموعه‌های مجزایی تقسیم شده و درخت‌های سلسله‌مراتبی کوچک‌تر متناظر با هر زیرمجموعه از فیلترها ایجاد می‌شود. سپس، درخت‌های سلسله‌مراتبی کوچک‌تر T'، زیرمجموعه فیلترهای متناظر با آن‌ها R'، بسته‌ها و آرایه خروجی به حافظه سراسری دستگاه ارسال می‌شوند (خط ۳). در خطهای ۴ و ۵ از الگوریتم، تعداد کل درخت‌ها و آدرس شروع آن‌ها به ترتیب در متغیرهای n و $treeOffset$ قرار می‌گیرد.

در این الگوریتم هر بلوک مسئول بررسی در مقابل یک ساختار واحد از درخت سلسله‌مراتبی است. خطهای ۱۰-۲۰ عملیات دسته‌بندی را انجام می‌دهند. در این الگوریتم، هر نخ با توجه به تعداد کل بسته‌ها، تعداد مشخصی بسته را دسته‌بندی می‌کند. در این الگوریتم هر نخ یک بسته را انتخاب کرده و آن را با استفاده از درخت T' و زیرمجموعه فیلترهای R' موجود در بلوک خود دسته‌بندی می‌کند. بعد از منطبق شدن بسته با یک فیلتر، خطهای ۱۵-۱۷ اندیس فیلتر منطبق شده دارای بالاترین اولویت را در آرایه خروجی می‌نویسد. تابع $AtomicMax(a,b)$ اگر مقدار b بزرگ‌تر از a باشد، مقدار b را در a ذخیره می‌کند. این تابع از شرایط رقابت بین بلوک‌ها جلوگیری می‌کند. در خط ۲۱ فراخوانی تابع $Syncthreads()$ تضمین می‌کند که تمام نخ‌های بلوک، عملیات دسته‌بندی خود را با درخت و فیلترهای موجود در حافظه اشتراکی قبل از کپی ساختار درخت و فیلترهای جدید تمام کرده باشند. در انتها، آدرس درخت‌ها به تعداد بلوک‌ها افزایش می‌یابد تا عملیات بررسی تطبیق بسته ورودی روی بقیه درخت‌های کوچک نیز انجام شود. در ادامه به بررسی جزییات سناریوها پرداخته و ویژگی‌های هریک را به‌دقت بررسی می‌کنیم.

- سناریو دوم: در این سناریو همان‌طور که در شکل ۴-ب نشان داده شده است، مجموعه فیلترهایی که در حافظه سراسری قرار دارند به چندین قسمت مساوی تقسیم می‌شود. هدف این تقسیم‌بندی آن است که با هر یک از زیرمجموعه فیلترهای ایجادشده، درختی با حداکثر ۲۴۵۷ گره ایجاد شود. زیرا دو بلوک با ۴۸ کیلوبایت حافظه اشتراکی در هرکدام در نظر گرفته شده است. از طرفی هر گره درخت نیاز به ۲۰ بایت فضا جهت ذخیره‌سازی دارد. بنابراین در هر بلوک حداکثر درختی با ۲۴۵۷ گره می‌توان ذخیره کرد. در هر مرحله از اجرای الگوریتم، به تعداد بلوک‌ها، درخت از زیرمجموعه فیلترهای متناظر ساخته شده به حافظه اشتراکی بلوک‌ها کپی می‌شوند. در این روش چون فیلترها به‌صورت مساوی تقسیم شده‌اند حافظه اشتراکی به‌صورت بهینه استفاده نمی‌شود.

- سناریو سوم: در زمان دسته‌بندی، وقتی که در مسیر جستجوی درخت فیلتری منطبق با آدرس‌های IP بسته وجود داشته باشد، پردازش نخی که مسئول دسته‌بندی بسته است، فیلدهای دیگر فیلتر را جهت تطبیق با بسته مقایسه می‌کند. اثر این مراجعات زیاد به فیلترها افزایش زمان دسته‌بندی بسته‌ها است. با در نظر گرفتن این موضوع و با توجه به این‌که تأخیر دسترسی به حافظه سراسری بسیار بالاتر از حافظه اشتراکی است، بهتر است که فیلترها در حافظه اشتراکی ذخیره شوند. بنابراین، در سناریوی سوم با ذخیره فیلترها در حافظه اشتراکی تأخیر دسترسی نخ‌ها به فیلترها کاهش می‌یابد و درنتیجه باعث افزایش سرعت دسته‌بندی می‌شود. در این سناریو، مجموعه فیلترها به زیرمجموعه‌هایی هم‌اندازه، تقسیم می‌شوند؛ این تقسیم‌بندی به‌گونه‌ای انجام می‌شود که هر زیرمجموعه فیلترها و درخت متناظر با آن باهم در حافظه اشتراکی جای گیرند. این سناریو در شکل ۴-ج نشان داده شده است.
- سناریو چهارم: در چهارمین سناریو فیلترها به‌صورتی تقسیم می‌شوند که کل فضای حافظه اشتراکی استفاده شود. در این صورت، تعداد درخت‌ها و گره‌های تکراری و درنتیجه تعداد دفعات پر و خالی شدن حافظه اشتراکی به حداقل می‌رسد. در این سناریو، همانند سناریوی دوم، فیلترها در حافظه سراسری هستند. شکل ۴-د این سناریو را نشان می‌دهد.
- سناریو پنجم: در آخرین سناریو، مجموعه فیلترها به‌گونه‌ای تقسیم‌بندی می‌شوند که گره‌های آن درخت به‌همراه هر زیرمجموعه حاصل تقسیم‌بندی به‌همراه درخت سلسله‌مراتبی نظیرش در حافظ اشتراکی جای گیرند و از کل این حافظه بیش‌ترین استفاده شود. شکل ۴-ه این سناریو را نشان می‌دهد.

۵- پیاده‌سازی و ارزیابی

در این بخش ابتدا ابزار ClassBench [۲۸] به اختصار معرفی می‌شود. از این ابزار برای تولید مجموعه فیلترهای ساختگی و سرآیندهای ساختگی استفاده می‌شود. سپس، شرایط لازم برای دسته‌بندی درخت سلسله‌مراتبی در واحد پردازش مرکزی و واحد پردازش گرافیکی بررسی می‌شود. در ادامه پیچیدگی زمانی، حافظه‌ای و پردازنده سناریوهای پیشنهادی بررسی شده و با توجه به آن‌ها، نتایج پیاده‌سازی الگوریتم روی مجموعه فیلتر از دیدگاه کارایی تحلیل و ارزیابی می‌شود.

۵-۱- ابزار ClassBench

ابزار ClassBench، یک شبیه‌ساز جهت تولید مجموعه فیلترهایی با توزیع‌های دلخواه در فضای هندسی فیلترها است. این ابزار متناظر با فیلترهای تولیدشده، سرآیندهایی ساختگی را نیز تولید می‌کند. درواقع، این ابزار با پارامترهای توزیعی که به‌عنوان ورودی به آن داده می‌شود فیلترها را ایجاد می‌کند. وجود این شبیه‌ساز نیاز توسعه‌دهندگان الگوریتم‌های دسته‌بندی بسته‌ها را به فیلترهای واقعی و ناهمگون موجود در دیواره‌های آتش، زنجیره‌های IP و لیست‌های کنترل دسترسی برطرف می‌نماید. مرور تحقیقات انجام‌شده در زمینه دسته‌بندی بسته‌ها نشان می‌دهد تحقیقات معدودی همچون [۲۴] مجموعه فیلترها و بسته‌های موردنیاز خود را به روش تصادفی تولید نموده‌اند. در اکثر تحقیقات [۲۶, ۲۷, ۲۹-۳۱]، به دلیل نیاز به فیلترها و بسته‌هایی که از لحاظ ویژگی‌های ساختاری و توزیع آماری به واقعیت نزدیک باشند از ابزار ClassBench جهت تولید ساختار داده‌های موردنیاز استفاده شده است. در این مقاله نیز از ابزار مذکور استفاده نموده و مجموعه فیلتر متناظر با پارامتر IPC2 حاوی تعداد 1k فیلتر به همراه تعداد ۸k تا ۶۴k بسته را جهت ارزیابی سناریوها تولید نموده‌ایم.

سناریوهای پیشنهادی روی سیستمی با مشخصات مندرج در جدول ۳ پیاده‌سازی شده است. در این پیاده‌سازی بستر برنامه‌نویسی کودا نسخه ۶/۵ مبتنی بر زبان C به کار گرفته شده است.

۵-۲- تقسیم و بررسی مجموعه فیلتر IPC در سناریوهای مختلف

مجموعه فیلتر IPC2 بعد از حذف فیلترهای تکراری دارای ۶۳۴ فیلتر متفاوت است. درخت سلسله‌مراتبی ایجادشده متناظر با مجموعه فیلترهای مذکور دارای ۷۲۱۵ گره است. ازآنجاکه هر گره در ساختار درخت نیاز به ۲۰ بایت فضای حافظه جهت ذخیره‌سازی دارد، کل درخت تقریباً به ۱۴۱ کیلوبایت فضای حافظه جهت ذخیره‌سازی نیاز دارد. بنابراین، جهت استفاده از حافظه اشتراکی و با توجه به محدودیت فضای آن، باید مجموعه فیلتر را به چندین قسمت تقسیم کرده و درختان سلسله‌مراتبی کوچک‌تری ایجاد کرد. همان‌طور که در جدول ۴ نشان داده شده است، تعداد درختان سلسله‌مراتبی و اندازه آن‌ها

جدول ۴: مشخصات مجموعه فیلتر IPC در سناریوهای مختلف- تعداد گره‌های هر زیردرخت (۶۳۴ فیلتر-۷۲۱۵ گره)

زیردرخت / سناریو	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	مجموع گره‌های زیردرختان	درصد گره‌های تکراری
سناریو دوم	۱۹۳۰	۲۱۱۴	۲۱۲۹	۱۸۹۵	۱۵۲۵	۶۹۴	۷۸۹	۷۰۱	۵۸۶	۶۲۳	۱۲۹۸۶	%۷۹/۹۹
سناریو سوم	۱۹۳۰	۲۱۱۴	۲۱۲۹	۱۸۹۵	۱۵۲۵	۶۹۴	۷۸۹	۷۰۱	۵۸۶	۶۲۳	۱۲۹۸۶	%۷۹/۹۹
سناریو چهارم	۲۴۵۳	۲۴۳۳	۲۴۵۲	۲۴۵۲	۱۱۸۱						۱۰۹۷۸	%۵۲/۱۵
سناریو پنجم	۲۳۵۲	۲۳۶۹	۲۳۴۶	۲۳۳۹	۱۴۸۶						۱۰۸۹۲	%۵۰/۹۶

متناظر با هر سناریو از سناریوهای پنج‌گانه متفاوت است. به‌عنوان مثال، در سناریو دوم، ۱۰ درخت سلسله‌مراتبی از کل مجموعه فیلتر ایجاد شده است. در صورت استفاده از دو بلوک در این سناریو، عمل دسته‌بندی در پنج مرحله با کپی کردن درخت‌ها در حافظه اشتراکی بلوک‌ها انجام می‌گیرد. مجموع گره‌های این زیردرختان نسبت به درخت یکپارچه این مجموعه فیلتر دارای ۷۹/۹۹ درصد گره اضافی است. بنابراین زمان دسته‌بندی این سناریو نسبت به سناریو اول افزایش خواهد یافت. تعداد درختان در سناریوی سوم برابر با سناریوی دوم است با این تفاوت که در سناریوی سوم قسمتی از حافظه اشتراکی، برای ذخیره‌سازی فیلترهای متناظر با درخت استفاده شده است. در چهارمین سناریو با توجه به اینکه تقسیم‌بندی فیلترها به‌گونه‌ای است که از کل حافظه اشتراکی استفاده می‌شود، تعداد زیر درختان به پنج کاهش یافته است. ازاین‌رو سناریو چهارم در صورت تعریف دو بلوک، با سه مرحله کپی‌کردن ساختار درخت از حافظه سراسری به حافظه اشتراکی عمل دسته‌بندی را انجام می‌دهند. در آخر سناریو پنجم همانند سناریو چهارم است، با این تفاوت که مجموعه فیلترها نیز در حافظه اشتراکی قرار گرفته‌اند. درنتیجه، سناریو پنجم نسبت به سناریو چهارم سریع‌تر است.

در ادامه ضمن بررسی مدل‌های پیچیدگی الگوریتم‌های موازی روی پردازنده‌های گرافیکی، پیچیدگی زمانی و حافظه‌ای سناریوهای پیشنهادی را بررسی می‌کنیم.

۵-۳- تحلیل پیچیدگی

یکی از ابزارهای مهم برای پیش‌بینی و تحلیل کارایی الگوریتم‌ها، محاسبه پیچیدگی زمانی و حافظه‌ای آن‌ها است [۳۲]. مدل‌های تحلیلی متفاوتی برای بررسی کارایی الگوریتم‌های موازی‌سازی شده در بستر پردازنده‌های گرافیکی ارائه شده است. برخی از مدل‌ها همچون TMM[۱۳] [۳۳]، PGM[۱۴] [۳۴]، BSP[۱۵] [۳۵]، HMM[۱۶] [۳۶]، DMM[۱۷] [۳۷]، UMM[۱۸] [۳۷] و MMM[۱۹] [۳۸] به‌صورت مجانبی عمل می‌نمایند؛ به‌عبارت‌دیگر، این مدل‌ها علاوه بر ویژگی‌های الگوریتم موازی برخی ویژگی‌های اصلی مربوط به معماری پردازنده گرافیکی را در محاسبه پیچیدگی محاسباتی و پیچیدگی حافظه‌ای در نظر می‌گیرند. به‌عنوان‌مثال، در مدل TMM با فرض کامل بودن زمان‌بندی، بیشینه پیچیدگی محاسباتی و پیچیدگی حافظه تقریب زده می‌شود.

در مقابل، برخی دیگر از مدل‌های تحلیلی بنام مدل‌های کالیبره، جزییات بیش‌تری را که در مدل‌های مجانبی اهمیت ندارند در تحلیل کارایی خود وارد نموده و با دقت بیش‌تری به تحلیل پیچیدگی و تقریب کارایی الگوریتم موازی روی پردازنده گرافیکی می‌پردازند. به‌عنوان

جدول ۵: پارامترهای مدل ترکیبی [۴۱]

پارامتر	تعریف
Q	تعداد پردازنده‌های روی هر SM
Z	اندازه حافظه اشتراکی در هر SM
L	زمان دسترسی به حافظه سراسری
C	حداکثر تعداد دسترسی‌های هم‌زمان به حافظه
P	مجموع هسته‌های پردازشی
X	بیشینه پردازش‌های نخی مجاز بر روی هر هسته پردازشی
T_1	تعداد کل عملیات در برنامه سری
M	تعداد انتقال خواندن و نوشتن از / به حافظه (پیچیدگی حافظه)
τ	تعداد پردازش‌های نخی به ازای هر هسته
n	تعداد فیلترهای دسته‌بند
a	تعداد بسته‌های ورودی
B_a	تعداد بلوک‌های فعال بر روی هر SM
B_r	تعداد کل بلوک‌های موردنیاز برای یک‌بار اجرای موازی الگوریتم

جدول ۶: مقادیر پارامترهای موردنیاز جهت محاسبه پیچیدگی سناریوهای پیشنهادی و مدل **Zhou**

سناریو	پردازنده گرافیکی	سناریو	P	Q	B_r	B_a	T_1	τ
سناریوهای پیشنهادی	GT 425 M	اول	۹۶	۴۸	۱	$\frac{۱}{۲}$	$O((\log n)^2)$	$\frac{۱۰۲۴}{۹۶} = ۱۰/۷$
		دوم			۱۰	۱		$\frac{۲۰۴۸}{۹۶} = ۲۱/۳$
		سوم						
		چهارم			۵			
		پنجم						
		ششم			۱	$\frac{۱}{۲}$		
	GTX 750	اول	۵۱۲	۱۲۸	۱	$\frac{۱}{۴}$		$\frac{۱۰۲۴}{۵۱۲} = ۲$
		دوم			۱۰	۱		$\frac{۴۰۹۶}{۵۱۲} = ۸$
		سوم						
		چهارم			۵			
		پنجم						
		ششم			۱	$\frac{۱}{۴}$		
Zhou	**K20**		۲۴۹۶	۱۹۲	۱۳	۱	$O(5\log n + n)$	$\frac{۱۳۳۱۷}{۲۴۹۶} = ۵/۳$

جدول ۷: پیچیدگی زمانی، پیچیدگی حافظه و پیچیدگی پردازنده سناریوهای پیشنهادی

پردازنده گرافیکی	سناریو	پیچیدگی زمانی	پیچیدگی حافظه	پیچیدگی پردازنده
GT 425 M	اول	$max\left(O((\log n)^2+n), \frac{O((\log n)^2+n)\times 100}{10.7}\right)\times\frac{1}{96}$	$O((\log n)^2+n)\times 100$	$O(a)$
	دوم	$max\left(O((\log n)^2+n), \frac{O(n^2+n)\times 100+O((\log n)^2)}{21.3}\right)\times\frac{5}{96}$	$O(n^2+n)\times 100+O((\log n)^2)$	$O(a\times t)$
	سوم	$max\left(O((\log n)^2+n), \frac{O(n^2)\times 100+O((\log n)^2+n)}{21.3}\right)\times\frac{5}{96}$	$O(n^2)\times 100+O((\log n)^2+n)$	
	چهارم	$max\left(O((\log n)^2+n), \frac{O(n^2+n)\times 100+O((\log n)^2)}{21.3}\right)\times\frac{3}{96}$	$O(n^2+n)\times 100+O((\log n)^2)$	
	پنجم	$max\left(O((\log n)^2+n), \frac{O(n^2)\times 100+O((\log n)^2+n)}{21.3}\right)\times\frac{3}{96}$	$O(n^2)\times 100+O((\log n)^2+n)$	
GTX 750	اول	$max\left(O((\log n)^2+n), \frac{O((\log n)^2+n)\times 100}{2}\right)\times\frac{1}{512}$	$O((\log n)^2+n)\times 100$	$O(a)$
	دوم	$max\left(O((\log n)^2+n), \frac{O(n^2+n)\times 100+O((\log n)^2)}{8}\right)\times\frac{3}{512}$	$O(n^2+n)\times 100+O((\log n)^2)$	$O(a\times t)$
	سوم	$max\left(O((\log n)^2+n), \frac{O(n^2)\times 100+O((\log n)^2+n)}{8}\right)\times\frac{3}{512}$	$O(n^2)\times 100+O((\log n)^2+n)$	
	چهارم	$max\left(O((\log n)^2+n), \frac{O(n^2+n)\times 100+O((\log n)^2)}{8}\right)\times\frac{2}{512}$	$O(n^2+n)\times 100+O((\log n)^2)$	
	پنجم	$max\left(O((\log n)^2+n), \frac{O(n^2)\times 100+O((\log n)^2+n)}{8}\right)\times\frac{2}{512}$	$O(n^2)\times 100+O((\log n)^2+n)$	

نمونه، مدل ارائه‌شده توسط Hong و همکاران [۳۹] یا مدل Liu [۴۰] و همکاران، از روش کالیبره استفاده نموده‌اند. از مشکلات این مدل آن است که دستیابی به برخی پارامترهای موردنیاز در روابط تحلیلی نیاز به دانش عمیق در مورد جزییات سخت‌افزاری پردازنده گرافیکی دارد. مشکل دیگر این مدل آن است که تأثیر برخی پارامترها، نظیر نرخ برخورد در حافظه نهان پردازنده گرافیکی را در نظر نمی‌گیرد.

اخیرا، مدل جدیدی توسط Lin Ma و همکاران [۴۱] برای تحلیل پیچیدگی الگوریتم‌های موازی روی پردازنده‌های گرافیکی پیشنهاد شده است. این مدل از ترکیب مدل مجانبی با مدل کالیبره به دست می‌آید. در این مدل، علاوه بر تحلیل مجانبی، پارامترهایی نظیر زمان پردازش سری، تعداد هسته‌های پردازنده، تعداد انتقال‌های به/از حافظه و تعداد پردازش‌های نخی در هر هسته در کنار تأثیر حافظه نهان و زمان‌بند سیستم چندپردازنده در نظر گرفته می‌شود. در این مقاله مدل ترکیبی فوق به همراه پارامترهای جدول ۵ استفاده شده است.

در مدل ترکیبی، زمان کلی اجرای الگوریتم روی پردازنده گرافیکی با رابطه زیر بیان می‌شود:

$$Time \propto \max\left(T_1, \frac{ML}{\tau}\right)\times\left\lceil\frac{QB_r}{PB_a}\right\rceil\times\frac{1}{\mathrm{P}} \qquad (۱)$$

در رابطه (۱)، وابستگی مجانبی به پارامترهای پیچیدگی محاسباتی و تعداد دسترسی‌ها به حافظه به ترتیب با عبارت‌های $\mathrm{T_1}$ و $\frac{\mathrm{ML}}{\tau}$ بیان می‌شوند. دقت کنید که $\mathrm{T_1}$ و M به نوع الگوریتم و سناریو انتخابی بستگی دارد. همچنین، با توجه به اینکه P/Q برابر تعداد SMها است، در صورتی $\mathrm{B_r} > \mathrm{B_a}\times \mathrm{P/Q}$، لازم است برای تکمیل پردازش و استفاده از همه تعداد بلوک‌های موردنیاز، بلوک‌های موجود به تعداد $\left\lceil\frac{QB_r}{PB_a}\right\rceil$ پر و خالی شوند. این عامل که بیانگر تعداد دفعات جایگزینی[۲۰] بلوک‌های فعال است، در دومین عبارت رابطه (۱) لحاظ شده است. مجموع تعداد تراکنش‌های حافظه، پیچیدگی حافظه نامیده می‌شود که با (ML) در رابطه (۱) نمایش داده می‌شود [۴۱].

در جدول ۶ مقادیر پارامترهای موردنیاز برای محاسبه پیچیدگی سناریوهای پیشنهادی به تفکیک پردازنده‌های گرافیکی مختلف ارائه شده است. بر همین اساس، جدول ۷ پیچیدگی محاسباتی، حافظه‌ای و پردازنده‌ای سناریوهای مذکور را نشان می‌دهد.

پیچیدگی عملیات انتقال درخت و مجموعه فیلترها از حافظه سراسری به حافظه اشتراکی، جستجو در درخت سلسله‌مراتبی و بررسی تطبیق فیلدهای دیگر به ترتیب $O(n^2+n)$، $O((\log n)^2)$ و $O(n)$ است. همچنین، ضرایب نرمال ۱۰۰ و ۱ متناظر با تأخیر دسترسی به حافظه سراسری و اشتراکی در محاسبه پیچیدگی حافظه استفاده شده است.

در جدول ۸ پیچیدگی پردازنده متناظر با هر سناریو نیز درج شده است. پیچیدگی پردازنده نشان‌دهنده حداکثر تعداد هسته‌های پردازشی است که برای اجرای سناریوی موازی می‌توان استفاده نمود [۴۲]. در سناریو اول این پارامتر متناسب با تعداد بسته‌ها $O(a)$ و در سناریوهای دیگر به دلیل وابستگی به تعداد بسته‌ها و تعداد زیردرخت‌ها $O(a\times t)$ است.

جدول ۸: نتایج پیاده‌سازی الگوریتم درخت سلسله‌مراتبی روی سناریوهای متفاوت

تعداد بسته	سناریو / نتایج پیاده‌سازی	GeForce GT 425M					GeForce GTX 750				
		سناریو اول	سناریو دوم	سناریو سوم	سناریو چهارم	سناریو پنجم	سناریو اول	سناریو دوم	سناریو سوم	سناریو چهارم	سناریو پنجم
۸ k	زمان محاسبه هسته (میلی‌ثانیه)	۳/۸۶	۱۰/۵۸	۱۰/۵۳	۷/۳۷	۴/۱۹	۳/۳۵	۶/۸۴	۶/۷	۶/۰۴	۳/۷۹
	زمان انتقال (میلی‌ثانیه)	۰/۸۶	۲/۵۵	۲/۱۴	۱/۳۹	۱/۸	۰/۴۹	۰/۹۲	۰/۸۸	۰/۴۹	۰/۹۴
	مجموع زمان (میلی‌ثانیه)	۴/۷۲	۱۳/۱۳	۱۲/۶۷	۸/۷۷	۵/۹۹	۳/۸۳	۷/۷۵	۷/۵۹	۶/۵۳	۴/۷۳
	تأخیر پردازش بسته (μs)	۰/۴۷	۱/۲۹	۱/۲۸	۰/۹	۰/۵۱	۰/۴۱	۰/۸۳	۰/۸۲	۰/۷۴	۰/۴۶
	گذرداد (MPPS)	۲/۰۲	۰/۷۴	۰/۷۴	۱/۰۶	۱/۸۶	۲/۳۳	۱/۱۴	۱/۱۷	۱/۲۹	۲/۰۶
	تسریع	۵/۲۸	۱/۹۳	۱/۹۴	۲/۷۷	۴/۸۷	۶/۰۹	۲/۹۸	۳/۰۴	۳/۳۸	۵/۳۸
۱۶ k	زمان محاسبه هسته (میلی‌ثانیه)	۷/۲۵	۱۸/۰۵	۱۷/۸۷	۱۴/۰۳	۷/۸۱	۵/۱۹	۱۳/۱۵	۱۲/۹۸	۱۱/۴۳	۶/۳۴
	زمان انتقال (میلی‌ثانیه)	۱/۱۹	۲/۰۲	۲/۱۷	۱/۸۵	۲/۰۹	۰/۵۱	۱/۰۳	۰/۹۱	۰/۵۷	۰/۹۵
	مجموع زمان (میلی‌ثانیه)	۸/۴۳	۲۰/۰۷	۲۰/۰۴	۱۵/۸۸	۹/۹	۵/۱۹	۱۴/۱۹	۱۳/۸۹	۱۲	۷/۳۳
	تأخیر پردازش بسته (μs)	۰/۴۴	۱/۱	۱/۰۹	۰/۸۶	۰/۴۸	۰/۳۲	۰/۸	۰/۷۹	۰/۷	۰/۳۹
	گذرداد (MPPS)	۲/۱۵	۰/۸۶	۰/۸۷	۱/۱۱	۲	۳/۰۱	۱/۱۹	۱/۲	۱/۳۷	۲/۴۶
	تسریع	۵/۱	۲/۰۵	۲/۰۷	۲/۶۴	۴/۷۴	۷/۱۳	۲/۸۱	۲/۸۵	۳/۲۴	۵/۸۳
۳۲ k	زمان محاسبه هسته (میلی‌ثانیه)	۱۲/۲۳	۳۳/۳۴	۳۳/۱۲	۲۲/۷۹	۱۳/۱۲	۱۰/۰۵	۲۰/۶۶	۲۰/۳۷	۱۷/۵۷	۱۰/۴۵
	زمان انتقال (میلی‌ثانیه)	۱/۷۳	۲/۴۱	۲/۷۷	۱/۵۲	۲/۷۲	۰/۵۹	۰/۹۸	۰/۹۷	۰/۷۹	۱/۰۲
	مجموع زمان (میلی‌ثانیه)	۱۳/۹۶	۳۵/۷۵	۳۵/۸۹	۲۴/۳۱	۱۵/۸۴	۱۱/۰۹	۲۱/۶۴	۲۱/۳۵	۸/۳۶	۱۱/۴۶
	تأخیر پردازش بسته (μs)	۰/۳۷	۱/۰۲	۱/۰۱	۰/۶۹	۰/۴	۰/۳۱	۰/۶۳	۰/۶۲	۰/۵۴	۰/۳۲
	گذرداد (MPPS)	۲/۵۵	۰/۹۴	۰/۹۴	۱/۳۷	۲/۳۸	۳/۱۱	۱/۵۱	۱/۵۳	۱/۷۸	۲/۹۹
	تسریع	۵/۵۶	۲/۰۴	۲/۰۵	۲/۹۸	۵/۱۸	۶/۷۷	۳/۲۹	۳/۳۴	۳/۸۷	۶/۵۱
۶۴ k	زمان محاسبه هسته (میلی‌ثانیه)	۱۳/۳۹	۴۰/۸۴	۳۹/۸۲	۲۹/۸۹	۱۷/۷۸	۱۱/۲۴	۲۵/۹۱	۲۴	۱۵/۵	۱۲/۸۹
	زمان انتقال (میلی‌ثانیه)	۱/۷۸	۲/۸۳	۲/۹۵	۲/۷۴	۳/۹۹	۰/۷۴	۱/۱۳	۱/۱۲	۰/۷۲	۱/۳۶
	مجموع زمان (میلی‌ثانیه)	۱۵/۱۷	۴۳/۶۶	۴۲/۷۸	۳۲/۶۳	۲۱/۷۷	۱۱/۹۸	۲۷/۰۴	۲۵/۱۳	۱۶/۲۲	۱۴/۲۵
	تأخیر پردازش بسته (μs)	۰/۲	۰/۶۲	۰/۶۱	۰/۴۶	۰/۲۷	۰/۱۷	۰/۳۹	۰/۳۷	۰/۲۴	۰/۲
	گذرداد (MPPS)	۴/۶۷	۱/۵۳	۱/۵۷	۲/۰۹	۳/۵۱	۵/۵۶	۲/۴۱	۲/۶	۴/۰۳	۴/۸۵
	تسریع	۹/۳۳	۳/۰۶	۳/۱۴	۴/۱۸	۷/۰۳	۱۱/۱۲	۴/۸۲	۵/۲۱	۸/۰۶	۹/۷

۶- ارزیابی نتایج

در این بخش، کارایی مکانیسم پیشنهادی جهت موازی‌سازی روی واحد پردازش گرافیکی از جنبه‌های مختلف کارایی نظیر زمان دسته‌بندی، گذرداد و تسریع بررسی می‌شود. زمان دسته‌بندی، مدت‌زمانی است که بسته‌ها توسط واحد پردازش گرافیکی دسته‌بندی می‌شوند. گذرداد میزان دسته‌بندی بسته‌های ورودی در واحد ثانیه است. همچنین، نسبت زمان دسته‌بندی بسته‌ها در پردازشگر گرافیکی به زمان دسته‌بندی بسته‌ها در پردازنده مرکزی، تسریع نامیده می‌شود.

ازآنجاکه طبق جدول ۷ پیچیدگی محاسباتی و حافظه‌ای سناریوهای پیشنهادی به تعداد فیلترها وابسته است، از دو مجموعه فیلتر IPC2 حاوی تعداد ۶۳۴ و ۳۳۴ به‌عنوان مجموعه فیلترهای بزرگ و کوچک در ارزیابی سناریوها استفاده شد.

جدول ۸ شامل نتایج اجرای سناریوها برای دسته‌بندی بسته‌های آزمون بر مبنای مجموعه فیلتر بزرگ IPC2، روی دو پردازنده گرافیکی معرفی‌شده در جدول ۳ است. در این جدول، به ازای هر سناریو، زمان دسته‌بندی، گذرداد، تسریع و همچنین زمان انتقال ساختار داده موردنیاز دسته‌بند بین پردازنده مرکزی و پردازنده گرافیکی اندازه‌گیری و ارائه شده است. واحد زمان برای محاسبات در این جدول بر اساس میلی‌ثانیه و واحد گذرداد میلیون بسته در ثانیه است.

مقایسه نتایج مربوط به اجرای سناریوها روی دو پردازنده گرافیکی GTX 750 و GT 425M در جدول ۸ نشان می‌دهد که پردازنده اول به دلیل بالاتر بودن تعداد SM و تعداد هسته‌های پردازشی از نظر زمان محاسبه، گذرداد و تسریع برتر از پردازنده دوم است.

همچنین، طبق جدول ۴، در سناریوهای دوم تا پنجم، با زیاد شدن تعداد فیلترها، درخت اصلی به تعداد B_r زیردرخت قابل ذخیره در حافظه اشتراکی تجزیه می‌شود. برای جستجو در این زیردرخت‌ها، لازم است بلوک‌های موجود به تعداد $\left\lceil \frac{QB_r}{PB_a} \right\rceil$ پر و خالی شوند. هر بار پرکردن بلوک‌های موجود مستلزم انتقال زیردرخت سلسله‌مراتبی فیلترها و مجموعه فیلترهای مرتبط با آن از حافظه سراسری به حافظه اشتراکی است. درنتیجه، طبق جدول ۷ می‌توان پیش‌بینی کرد که با افزایش تعداد فیلترهای دسته‌بند، به دلیل افزایش چشمگیر در پیچیدگی حافظه‌ای و پیچیدگی زمانی سناریوهای مذکور، میزان تسریع و نرخ گذردادشان کم‌تر از سناریوی اول شود. بررسی نتایج ارائه‌شده در جدول ۸ نشان می‌دهد که بیش‌ترین میزان تسریع و نرخ گذرداد در دسته‌بندی

جدول ۹: پیچیدگی زمانی و حافظه‌ای سناریوهای اول، ششم و **Zhou**

سناریو	پیچیدگی زمانی	پیچیدگی حافظه
سناریو اول **GTX 750**	$max\left(O((\log n)^2+n), \frac{O((\log n)^2+n)\times 100}{2}\right)\times\frac{1}{512}$	$O((\log n)^2+n)\times 100$
سناریو ششم **GTX 750**	$max\left(O((\log n)^2+n), \frac{O(n+(\log n)^2)}{8}\right)\times\frac{1}{512}$	$O(n+(\log n)^2)$
مدل **Zhou** **K20**	$max\left(O(5\log\ n+n), \frac{O\left(\log\frac{n}{32}\right)+2O\left(\frac{n}{32}\right)}{5.3}\right)\times\frac{1}{192}$	$O\left(\log\frac{n}{k}\right)+2O\left(\frac{n}{k}\right)$

جدول ۱۰: مجموعه فیلتر **IPC**– ۳۴۴ فیلتر- ۱ درخت با ۲۰۸۵ گره

سناریو	تعداد بسته‌ها / نتایج پیاده‌سازی	GeForce GT 425M				GeForce GTX 750			
		۸ k	۱۶ k	۳۲ k	۶۴ k	۸ k	۱۶ k	۳۲ k	۶۴ k
سناریو اول	زمان محاسبه هسته (میلی‌ثانیه)	۳/۳	۵/۹۶	۹/۳۸	۱۰/۸۸	۲/۳	۴/۷۱	۷/۸۶	۹/۰۸
	زمان انتقال (میلی‌ثانیه)	۰/۹۲	۱/۳۶	۱/۲۹	۱/۶۶	۰/۴۵	۰/۴۷	۰/۵۴	۰/۷۲
	مجموع زمان (میلی‌ثانیه)	۴/۲۲	۷/۳۱	۱۰/۶۷	۱۲/۵۴	۲/۸۵	۵/۱۸	۸/۴	۹/۸
	تأخیر پردازش بسته (µs)	۰/۴	۰/۳۶	۰/۲۹	۰/۱۷	۰/۲۸	۰/۲۹	۰/۲۴	۰/۱۴
	گذرداد (MPPS)	۲/۳۷	۲/۶۲	۳/۳۳	۵/۷۴	۳/۴	۳/۳۲	۳/۹۷	۶/۸۸
	تسریع	۵/۶۶	۶/۲۶	۸/۳۲	۸/۴۳	۸/۱۳	۷/۹۲	۹/۹۲	۱۰/۱
سناریو ششم	زمان محاسبه هسته (میلی‌ثانیه)	۰/۶۴	۱/۲۴	۲/۴۶	۲/۷۸	۰/۳۹	۰/۸۱	۱/۵	۱/۷۲
	زمان انتقال (میلی‌ثانیه)	۰/۳	۰/۳۴	۰/۲۳	۰/۳۱	۰/۱۴	۰/۱۵	۰/۰۹	۰/۱۸
	مجموع زمان (میلی‌ثانیه)	۰/۹۴	۱/۵۸	۲/۶۹	۳/۰۹	۰/۵۳	۰/۹۷	۱/۶	۱/۹
	تأخیر پردازش بسته (µs)	۰/۰۸	۰/۰۷	۰/۰۷	۰/۰۴	۰/۰۵	۰/۰۵	۰/۰۴	۰/۰۳
	گذرداد (MPPS)	۱۲/۲۱	۱۲/۶	۱۲/۷	۲۲/۴۸	۲۰/۰۳	۱۹/۲۹	۲۰/۸۳	۳۶/۳۴
	تسریع	۲۹/۲۱	۳۰/۱	۳۱/۷۱	۳۳/۰۲	۴۷/۹۵	۴۶/۰۷	۵۲	۵۳/۳۴

بسته‌ها توسط سناریوی اول به دست آمده و به ترتیب برابر با ۱۱/۱۲ و ۵/۵۶ است.

طبق رابطه (۱)، در سناریوهایی که از حافظه اشتراکی استفاده می‌کنند می‌توان با دو ایده کلیدی زیر پیچیدگی زمانی را کاهش داده و درنتیجه کارایی را افزایش داد. ایده نخست استفاده کامل از کل فضای حافظه اشتراکی روی بلوک‌ها برای ذخیره‌سازی زیردرخت‌ها و فیلترها است که باعث کاهش تعداد بلوک‌های موردنیاز B_r و درنتیجه، تعداد دفعات پر و خالی شدن بلوک‌های موجود، $\left\lceil\frac{QB_r}{PB_a}\right\rceil$ می‌شود. ایده دوم انتقال مجموعه فیلترها علاوه بر زیردرخت‌ها به حافظه اشتراکی بلوک‌های موجود است که باعث می‌شود پیچیدگی حافظه‌ای در دسترسی نخ‌ها به مجموعه فیلترها حدود ۱۰۰ برابر کاهش یابد. بنابراین، می‌توان انتظار داشت سناریوهای پنجم و دوم به ترتیب کم‌ترین و بیش‌ترین پیچیدگی زمانی و درنتیجه به ترتیب بیش‌ترین و کم‌ترین کارایی را در میان سناریوهای مذکور داشته باشند. مقایسه مقادیر تأخیر پردازش هر بسته، تسریع و گذرداد سناریوهای دوم تا پنجم در جدول ۸ این موضوع را تأیید می‌کند.

بررسی و مقایسه پیچیدگی زمانی و حافظه‌ای سناریو پنجم با سناریوی اول نشان می‌دهد درصورتی‌که درخت ایجادشده از مجموعه فیلترها به‌قدری کوچک باشد که در یک بلوک حافظه اشتراکی جای گیرد، می‌توان انتظار داشت کارایی سناریوی پنجم از سناریوی اول بهتر شود. بدین دلیل، در آزمایش بعدی تعداد فیلترهای دسته‌بند به‌گونه‌ای تعیین شد که درخت سلسله‌مراتبی متناظر با آن‌ها به همراه مجموعه فیلترها در یک حافظه اشتراکی ۴۸ کیلوبایتی جای بگیرند. در این حالت خاص از سناریو پنجم که سناریوی ششم نام دارد، برای مجموعه فیلتر IPC با ۳۴۴ فیلتر یک درخت سلسله‌مراتبی با ۲۰۸۵ گره ایجاد می‌شود. فضای موردنیاز برای ذخیره درخت برابر با ۴۰/۷۲ کیلوبایت است که با درنظر گرفتن فضای موردنیاز برای ذخیره ۳۴۴ فیلتر به ۴۷/۴۴ کیلوبایت می‌رسد. کل این ساختار داده در یک حافظه اشتراکی جای می‌گیرند.

جدول ۹ پیچیدگی محاسباتی سناریوهای اول و ششم را روی پردازنده گرافیکی GTX 750 نشان می‌دهد. در جدول ۱۰ نیز، زمان محاسبه هسته، زمان انتقال، مجموع زمان، تأخیر پردازش هر بسته، گذرداد و تسریع سناریوهای اول و ششم در دسته‌بندی بسته‌های آزمون با هم مقایسه شده است. طبق این جدول بیش‌ترین میزان تسریع و گذرداد و کم‌ترین تأخیر پردازش هر بسته در هر دو سناریو بر روی پردازنده GTX 750 حاصل می‌شود. در اجرای سناریوهای مذکور بر روی این پردازنده گرافیکی، بیشینه تسریع و گذرداد و همچنین کمینه تأخیر پردازش هر بسته در دسته‌بندی ۶۵۵۳۶ بسته به دست آمده

است. درحالی‌که سناریو اول در هر ۰/۱۴ میکروثانیه یک بسته را پردازش نموده و بیشینه گذردادش ۶/۸۸ MPPS است، سناریوی ششم با صرف زمان بسیار کم‌تری در حد ۰/۰۳ جهت پردازش هر بسته، به نرخ گذرداد ۳۶/۳۴ MPPS دست می‌یابد. مقایسه میزان تسریع دو سناریوی مذکور نیز نشان می‌دهد که سناریوی ششم کاراتر از سناریو اول است. این نتیجه با بررسی پیچیدگی زمانی سناریوهای مذکور به ازای n= ۳۴۴، در جدول ۹ منطبق است.

می‌توان سناریو ششم را با سناریو پیاده‌سازی‌شده توسط Zhou و همکاران مقایسه نمود [۲۹]. برای انجام مقایسه حالتی از سناریو Zhou را در نظر می‌گیریم که در آن تعداد فیلترها کم‌تر از ۵۱۲ بوده و کل ساختار داده دسته‌بند با یک مرتبه انتقال در حافظه اشتراکی جای می‌گیرد. در پیچیدگی حافظه‌ای و زمانی هر دو سناریو با پیروی از سناریو Zhou پیچیدگی انتقال ساختار داده از حافظه سراسری به حافظه اشتراکی در نظر گرفته نمی‌شود. پیچیدگی حافظه‌ای و زمانی این دو سناریو در جدول ۹ ارائه شده است.

با در نظر گرفتن n=۳۴۴ در رابطه پیچیدگی‌های زمانی سناریوی مذکور و سناریو ششم می‌توان پیش‌بینی کرد که سناریوی ششم در مدت زمان کم‌تری نسبت به سناریوی Zhou می‌تواند تعداد یکسانی از بسته‌ها را پردازش کند. مقایسه نتایج جدول ۱۰ با نتایج Zhou در دسته‌بندی ۶۵۵۳۶ بسته با مجموع ۵۱۲ فیلتر تحلیل فوق را تأیید نموده و نشان می‌دهد درحالی‌که سناریوی ششم قادر است بر روی پردازنده گرافیکی GTX 750 با ۵۱۲ هسته پردازشی، هر بسته را در ۰/۰۳ میکروثانیه پردازش کند سناریو Zhou با وجود در اختیار داشتن ۲۴۹۶ هسته روی پردازنده گرافیکی K20، قادر نیست پردازش هر بسته را در زمانی کم‌تر از ۴/۹ میکروثانیه به اتمام رساند. در این شرایط نرخ گذرداد سناریوی ششم و سناریو Zhou به ترتیب برابر با MPPS ۳۶/۳۴ و MPPS ۸۵ است. ازآنجاکه طبق جدول ۳ تعداد هسته‌های پردازنده مورداستفاده در سناریو ششم حدود ۴/۸۷۵ برابر کم‌تر از تعداد هسته‌های پردازنده مورد استفاده در سناریو Zhou است و با توجه به این‌که نرخ گذرداد سناریوی ششم متناسب با تعداد هسته‌های پردازشی است، می‌توان انتظار داشت گذرداد سناریوی ششم روی پردازنده گرافیکی K20، حدود MPPS ۱۷۷/۱۵۷ شود. چنین دسته‌بندی می‌تواند در شبکه‌های مبتنی بر استاندارد OC-768، که بیشینه نرخ گذرداد بسته در آن‌ها MPPS ۱۲۵ است، جایگزین دسته‌بندهای سخت‌افزاری گران‌قیمتی شود که با استفاده از حافظه‌های تداعی‌گر سه‌وضعیتی طراحی می‌شوند [۳, ۴].

۷- نتیجه‌گیری

دسته‌بندی بسته‌ها پردازشی پایه‌ای در اکثر سیستم‌های شبکه‌ای پردازنده بسته است که در آن بر اساس مجموعه فیلترهای تعریف‌شده روی فیلدهای سرآیند بسته‌ها، بسته‌ها به جریان‌های مختلف، تقسیم‌بندی می‌شود؛ در این صورت، سیستم با سرعت بسیار بالاتری کنش‌های متناظر را به جریان‌های تفکیک‌شده اعمال می‌نماید. عملیات کلیدی در این پردازش، جستجو جهت بررسی تطبیق سرآیند بسته ورودی با فیلترهای تعریف‌شده در مجموعه فیلترها و درنهایت یافتن بهترین فیلتر منطبق است. الگوریتم‌های متفاوتی برای دسته‌بندی بسته‌ها معرفی شده‌اند. الگوریتم درخت سلسله‌مراتبی از مهم‌ترین الگوریتم‌های دسته‌بندی بسته‌ها است که مبتنی بر نگاشت فیلترهای دسته‌بند به یک درخت تصمیم است. این الگوریتم با وجود قابلیت توسعه‌پذیری به دلیل محدودیت‌های پردازشی روی واحد پردازش مرکزی کارایی مناسبی ندارد. در این مقاله برای نخستین بار، ایده موازی‌سازی این الگوریتم روی واحد پردازش گرافیکی مطرح و پیاده‌سازی و ارزیابی شد. در این راستا، سناریوهای متفاوتی، جهت پیاده‌سازی موازی الگوریتم درخت سلسله‌مراتبی با در نظر گرفتن حالت‌های ممکن جهت ذخیره‌سازی مجموعه فیلترها و درخت تصمیم متناظر، در ماژول‌های مختلف حافظه واحد پردازش گرافیکی ارائه شد. در این سناریوها از حافظه اشتراکی جهت بهره بردن از سرعت بالای دسترسی آن، استفاده شده است. مهم‌ترین چالش در استفاده از حافظه اشتراکی، توجه به میزان فضای موردنیاز جهت نگه‌داری ساختار درخت است. زیرا اندازه ساختار درخت بسیار بزرگ‌تر از حافظه اشتراکی است. برای حل این مشکل نشان داده شد که می‌توان با تقسیم مجموعه فیلتر در حافظه سراسری واحد پردازش گرافیکی، درخت‌هایی با اندازه کوچک‌تر یا مساوی حافظه اشتراکی ایجاد نمود و درخت‌های کوچک‌تر را حتی به انضمام زیرمجموعه فیلترهای متناظر به حافظه اشتراکی انتقال داد. بر این اساس، جای‌گشت‌های ممکن برای انتقال زیردرخت‌ها و مجموعه فیلترهای متناظر به حافظه اشتراکی را در قالب پنج سناریو بیان نموده و برای هریک از آن‌ها، پیچیدگی زمانی، حافظه‌ای و پردازنده‌ای را به‌صورت تحلیلی محاسبه و با هم مقایسه کردیم.

در ادامه، برای ارزیابی سناریوهای مذکور، آن‌ها را در بستر کودا پیاده‌سازی و با مجموعه فیلتر IPC برای دسته‌بندی تعداد متفاوتی از بسته‌های آزمون، به تعداد ده بار اجرا کردیم. نتایج به‌دست‌آمده، با پیچیدگی‌های محاسبه‌شده منطبق بوده، کارایی سناریوهای پیشنهادی را جهت موازی‌سازی الگوریتم درخت سلسله‌مراتبی روی پردازنده گرافیکی نسبت به نسخه متوالی الگوریتم روی واحد پردازش مرکزی تائید می‌نماید. همچنین، تحلیل پیچیدگی سناریوها و تطبیق آن‌ها با نتایج ارزیابی نشان می‌دهد در صورت افزایش تعداد زیردرخت‌ها، کارایی سناریوهای مبتنی بر حافظه اشتراکی، به دلیل پردازش بسته‌های تکراری، از کارایی سناریویی که کل ساختار درخت سلسله‌مراتبی را در حافظه سراسری نگه می‌دارد کم‌تر بوده و با افزایش تعداد زیردرخت‌ها کاهش می‌یابد. علاوه بر این، بهترین سناریو، سناریویی است که درخت سلسله‌مراتبی و مجموعه فیلترهای متناظر را بدون شکستن به زیردرخت‌ها، باهم در حافظه اشتراکی جای می‌دهد. در این حالت نرخ گذرداد سناریوی بهینه روی پردازنده گرافیکی GTX

750 برابر با ۳۶/۳۴ MPPS است که می‌تواند روی پردازنده گرافیکی K20 به ۱۷۷/۱۵۷ MPPS برسد.

اخیراً خوشه پردازنده‌های گرافیکی، به بستری نوین برای محاسبات توزیع‌شده با موازات چشمگیر تبدیل شده است [۴۳, ۴۴]. در ادامه تحقیق، برای افزایش موازات در اجرای الگوریتم دسته‌بندی درخت سلسله‌مراتبی از خوشه پردازنده‌های گرافیکی استفاده خواهد شد. در چنین بستری می‌توان میزان تسریع و نرخ گذرداد دسته‌بندی بسته‌ها را تا حد چشمگیری افزایش داد.

مراجع

[1] D. Pao and Z. Lu, "A multi-pipeline architecture for high-speed packet classification," *Computer Communications*, vol. 54, pp. 84-96, 2014.

[2] B. S. Tumari and W. Lakshmipriya, "FPGA Implementation of Binary-tree-based High Speed Packet ClassificationSystem," *International Journal of Combined Research & Development (IJCRD)*, vol. 2, pp. 17-22, 2014.

[3] K. Zheng, H. Che, Z. Wang and B. Liu, "TCAM-based distributed parallel packet classification algorithm with range-matching solution," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, pp. 293-303, 2005.

[4] K. Zheng, H. Che, Z. Wang, B. Liu and X. Zhang, "DPPC-RE: TCAM-based distributed parallel packet classification with range encoding," *Computers, IEEE Transactions on*, vol. 55, pp. 947-961, 2006.

[5] Z. Cao, M. Kodialam and T. Lakshman, "Traffic steering in software defined networks: planning and online routing," *ACM SIGCOMM Computer Communication Review - SIGCOMM'14*, vol. 44, pp. 65-70, 2014.

[6] K. Guerra Perez, X. Yang, S. Scott-Hayward and S. Sezer, "A configurable packet classification architecture for Software-Defined Networking," in *27th IEEE International System-on-Chip Conference (SOCC)*, pp. 353-358, 2014.

[7] S. Han, K. Jang and S. Moon, "PacketShader: a GPU-accelerated software router," *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 195-206, 2011.

[8] K. G. Perez, X. Yang, S. Scott-Hayward and S. Sezer, "Optimized packet classification for Software-Defined Networking," in *International Conference on Communications (ICC), IEEE*, pp. 859-864, 2014.

[9] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys (CSUR)*, vol. 37, pp. 238-275, 2005.

[10] S. Zhou, Y. R .Qu and V. K. Prasanna, "Multi-core implementation of decomposition-based packet classification algorithms," in *Parallel Computing Technologies*, vol. 7979, ed: Springer, pp. 105-119, 2013.

[11] V. Srinivasan, S. Suri and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Computer Communication Review*, vol. 29, pp. 135-146, 1999.

[12] H. Lim, Y. Choe, M. Shim and J. Lee, "A Quad-Trie Conditionally Merged with a Decision Tree for Packet Classification," *Communications Letters, IEEE*, vol. 18, pp. 676 - 679, 2014.

[۱۳] سعید پارسا و محمد حمزه‌ئی، «کاشی‌بندی حلقه‌های تودرتو با در نظر گرفتن محلیت داده‌ها به‌منظور اجرای موازی بر روی پردازنده‌های چندهسته‌ای»، مجله مهندسی برق دانشگاه تبریز، جلد ۴۵، شماره ۳، صفحه ۱۷-۲۶، ۱۳۹۴.

[14] H. Lim, S. Lee and E. E. Swartzlander Jr, "A new hierarchical packet classification algorithm," *Computer Networks*, vol. 56, pp. 3010-3022, 2012.

[15] *NVIDIA. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, version 6.5, August 2015,* http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[16] *AMD: Global Provider of Innovative Graphics, Processors, August 2015* Available: http://www.amd.com

[17] Y. Li, D. Zhang, A. X. Liu and J. Zheng, "GAMT: a fast and scalable IP lookup engine for GPU-based software routers," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 1-12, 2013.

[18] T.-H. Li, H.-M. Chu and P.-C. Wang, "IP address lookup using GPU," in *14th International Conference on High Performance Switching and Routing (HPSR), IEEE*, pp. 177-184, 2013.

[19] R. S. Sinha, S. Singh, S. Singh and V. K. Banga, "Speedup Genetic Algorithm Using C-CUDA," in *Fifth International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 1355-1359, 2015.

[20] M. Beyeler, N. Oros, N. Dutt and J. L. Krichmar, "A GPU-accelerated cortical neural network model for visually guided robot navigation," *Neural Networks*, In Press, 2015.

[21] Y. Lu, Y. Zhu ,M. Han, J. S. He and Y. Zhang, "A survey of GPU accelerated SVM," in *Proceedings of the 2014 ACM Southeast Regional Conference*, pp. 15-23, 2014.

[22] P. Przymus and K. Kaczmarski, "Dynamic compression strategy for time series database using GPU," in *New Trends in Databases and Information Systems*, ed: Springer, pp. 235-244, 2014.

[23] G. Vasiliadis, E. Athanasopoulos, M. Polychronakis and S. Ioannidis, "PixelVault: Using GPUs for securing cryptographic operations," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1131-1142, 2014.

[24] C.-L. Hung, Y.-L. Lin, K.-C. Li, H.-H. Wang and S.-W. Guo, "Efficient GPGPU-based parallel packet classification," in *Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1367-1374, 2011.

[25] A. Nottingham and B. Irwin, "GPU packet classification using OpenCL: a consideration of viable classification methods," in *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pp. 160-169, 2009.

[26] Y. Deng, X. Jiao, S. Mu, K. Kang and Y. Zhu, "NPGPU: Network Processing on Graphics Processing Units," in *Theoretical and Mathematical Foundations of Computer Science*, ed: Springer, pp . 313-321, 2011.

[27] K. Kang and Y. S. Deng, "Scalable packet classification via GPU metaprogramming," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-4, 2011.

[28] D. E. Taylor and J. S. Turner, "Classbench: A packet classification benchmark," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 2068-2079, 2005.

[29] S. Zhou, S. G. Singapura and V. K. Prasanna, "High-Performance Packet Classification on GPU", in *High Performance Extreme Computing Conference (HPEC)*, pp. 1-6, 2014.

[30] M. Varvello, R. Laufer, F. Zhang and T. Lakshman, "Multi-Layer Packet Classification with Graphics Processing Units," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 109-120, 2014.

[31] J. Zheng, D. Zhang, Y. Li and G. Li, "Accelerate Packet Classification Using GPU: A Case Study on HiCuts," in *Computer Science and its Applications*, ed: Springer, pp . 231-238, 2015.

[۳۲] احسان اولیائی ترشیزی و حسین شریفی، «ارائه دو الگوریتم دیکدینگ هیبرید جدید با عملکرد بسیار خوب و پیچیدگی بسیار کم برای دیکدینگ کدهای LDPC»، مجله مهندسی برق دانشگاه تبریز، جلد ۴۵، شماره ۴، صفحه ۲۷-۳۷، ۱۳۹۴.

[23] L. Ma, K. Agrawal and R. D. Chamberlain", A memory access model for highly-threaded many-core architectures," *Future Generation Computer Systems*, vol. 30, pp. 202-215, 2014.

[34] J. S. Kirtzic, O. Daescu, "A parallel algorithm development model for the GPU architecture," in *Proc. of International Conf. on Parallel and Distributed Processing Techniques and Applications*, pp. 1-9, 2012.

[35] M. Amaris, D. Cordeiro, A. Goldman and R. Y. de Camargo, "A Simple BSP-based Model to Predict Execution Time in GPU Applications," in *22nd annual IEEE International Conference on High Performance Computing (HiPC 2015)*, pp. 285-294, 2015.

[36] K. Nakano, "The hierarchical memory machine model for GPUs," in *IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pp. 591-600, 2013.

[37] K. Nakano, "Simple memory machine models for GPUs," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 29, pp. 17-37, 2014.

[38] S. A. Haque and N. Xie, "A many-core machine model for designing algorithms with minimum parallelism overheads," in *arXiv preprint arXiv:1402.0264*, 2014.

[39] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *ACM SIGARCH Computer Architecture News*, pp. 152-163, 2009.

[40] W. Liu, W. Müller-Wittig and B. Schmidt, "Performance predictions for general-purpose computation on GPUs," in *International Conference on Parallel Processing (ICPP)*, pp. 50-50, 2007.

[41] L. Ma, R. D. Chamberlain and K. Agrawal, "Performance modeling for highly-threaded many-core GPUs," in *25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 84-91, 2014.

[42] S. H. Roosta, *Parallel processing and parallel algorithms: theory and computation*, Springer Science & Business Media, 2012.

[43] D. A. Jacobsen, J. C. Thibault and I. Senocak, "An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters," in *48th AIAA aerospace sciences meeting and exhibit*, pp. 1-16, 2010.

[44] M. Bernaschi, M. Bisson and M. Fatica, "Colloquium: Large scale simulations on GPU clusters," *The European Physical Journal B*, vol. 88, pp. 1-10, 2015.

زیرنویس‌ها

۱ Field-Programmable Gate Array (FPGA)
۲ Millions Packets PerSecond
۳ Hierarchical trie (H-trie)
۴ Streaming Multiprocessor (SM)
۵ Compute Unified Device Architecture (CUDA)
۶ Streaming Processor (SP)
۷ Wildcard
۸ BSD Packet Filter
۹ Recursive Flow Classification (RFC)
۱۰ Discrete Bit Selection (DBS)
۱۱ Bit Vector
۱۲ Hierarchical Intelligent Cuttings
۱۳ Threaded Multi-core Memory
۱۴ Parallel GPU Model
۱۵ Bulk Synchronous Parallel
۱۶ Hierarchical Memory Module
۱۷ Discrete Memory Machines
۱۸ Unified Memory Module
۱۹ Many-core Machine Model
۲۰ Occupancy