مجله محاسبات نرم، جلد 14، شماره 1 (بهار و تابستان 1404)، صفحه 88ـ107
https://doi.org/10.22052/scj.2024.252862.1145

مجله محاسبات نرم
SOFT COMPUTING JOURNAL
تارنمای مجله: scj.kashanu.ac.ir

دانشگاه کاشان
University of Kashan

پیش‌بینی کارآیی برنامه‌های منظم کودا از طریق روش‌های یادگیری ماشین✧

علی ریاحی[1]، استادیار، عبدالرضا سوادی[2*]، استادیار، محمود نقیب‌زاده[2]، استاد

[1] گروه مهندسی کامپیوتر، دانشکده مهندسی برق و کامپیوتر، دانشگاه علم و فناوری مازندران، بهشهر، ایران.

[2] گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه فردوسی مشهد، مشهد، ایران.

اطلاعات مقاله

تاریخچه مقاله:
دریافت 9 اردیبهشت ماه 1402
پذیرش 19 تیر ماه 1403

کلمات کلیدی:
پردازنده گرافیکی
کودا
مدل کارآیی
پیش‌بینی زمان اجرا
یادگیری ماشین
انتخاب ویژگی

چکیده

پیش‌بینی کارآیی پردازنده‌های گرافیکی یک چالش برای برنامه‌نویسان کودا است و در تنظیم برنامه‌ها کاربرد وسیعی دارد چراکه سربارهای برنامه‌های کودا ممکن است باعث شوند که اجرای برنامه‌ها روی پردازنده گرافیکی مقرون به صرفه نباشند. پیش‌بینی کارآیی به برنامه‌نویس کمک می‌کند تا تنظیم برنامه‌ها را آگاهانه‌تر انجام دهند و به جای آزمون و خطا، نقطه بهینه تنظیم را دقیق‌تر پیدا کند. وجود یک مدل که بتواند کارآیی را پیش‌بینی کند، می‌تواند به برنامه‌نویسان در تنظیم دستی و به توسعه‌دهندگان کامپایلرهای مبدا به مبدا در تنظیم خودکار برنامه‌های کودا کمک کند. در این مقاله، ما یک مدل کارآیی برای پیش‌بینی زمان اجرای یک کرنل کودا ارائه کرده‌ایم. در این مدل، ابتدا با استفاده از پروفایلینگ و با استفاده از تحلیل ایستای کدهای کودا و PTX، دو مجموعه داده برای زمان اجرای کرنل‌های محاسبه‌گرا و حافظه‌گرا ایجاد شده است. سپس با استفاده از روش‌های شبکه عصبی مصنوعی، ماشین بردار پشتیبان، و ماشین یادگیری افراطی، زمان اجرای کرنل کودا پیش‌بینی شده است. نتایج آزمایشگاهی نشان می‌دهند که روش ماشین یادگیری افراطی می‌تواند زمان اجرای یک کرنل محاسبه‌گرا را با حداکثر خطای 3/42 درصد و زمان اجرای یک کرنل حافظه‌گرا را با حداکثر خطای 9/84 درصد پیش‌بینی کند. برای اعتبارسنجی بهتر نتایج از روش اعتبارسنجی متقاطع استفاده شده است. همچنین با استفاده از روش‌های انتخاب ویژگی، میزان تاثیر هر یک از ویژگی‌های ورودی روی زمان اجرای کرنل مشخص شده است.

1. مقدمه

با افزایش قابلیت‌های پردازشی پردازنده گرافیکی، از این پردازنده در کاربردهای بسیار وسیعی از پردازش موازی ازجمله محاسبات علمی، پردازش داده‌های بزرگ، داده‌کاوی، مسیریاب‌های اینترنت و بیوانفورماتیک استفاده شده است و عنوان پردازنده همه منظوره[1] بر آن قرار داده شده است. این پردازنده‌ها به عنوان یک شتاب‌دهنده[2] و یا به عنوان یک پردازنده همکار[3] توسط پردازنده مرکزی مورد استفاده قرار می‌گیرند. به همین دلیل معمولا به

✧ نوع مقاله: پژوهشی
* نویسنده مسئول
پست(های) الکترونیک: riahi@mazust.ac.ir (ریاحی)
savadi@um.ac.ir (سوادی)
naghibzadeh@.um.ac.ir (نقیب‌زاده)

[1] General Purpose GPU (GPGPU)
[2] Accelerator
[3] Co-processor

نحوه ارجاع به مقاله: علی ریاحی، عبدالرضا سوادی، محمود نقیب‌زاده، «پیش‌بینی کارآیی برنامه‌های منظم کودا از طریق روش‌های یادگیری ماشین»، مجله محاسبات نرم، جلد 14، شماره 1، ص88ـ107، بهار و تابستان 1404.

پردازنده مرکزی اصطلاح میزبان[1] و به پردازنده گرافیکی اصطلاح دستگاه[2] اطلاق می‌شود. میزبان تابعی که کرنل[3] نامیده می‌شود را برای دستگاه ارسال کرده و پس از خاتمه اجرای کرنل، نتایج جهت استفاده در ادامه برنامه به سمت میزبان برگردانده می‌شوند [1].

مدل برنامه‌نویسی کودا توسط شرکت انویدیا برای طراحی برنامه‌های موازی و اجرای آنها روی پردازنده گرافیکی این شرکت ارائه شده است. این زبان امکانات پردازشی فوق‌العاده‌ای به طراحان برنامه‌های موازی می‌دهد که استفاده از هسته‌های زیاد پردازنده‌های گرافیکی برای تسریع برنامه‌ها را فراهم می‌کند. این کدها قبل از ترجمه به زبان صفر و یک، توسط کامپایلر NVCC به کدهای PTX[4] ترجمه می‌شوند. این کدها یک روایت شبه اسمبلی از کد کودا بوده و به زبان ماشین نزدیک هستند. با تحلیل همزمان کدهای کودا و PTX اطلاعات بیشتری درباره برنامه به دست می‌آید.

چالش اصلی در استفاده از پردازنده‌های گرافیکی این است که جزئیات ناچیزی از نحوه عملکرد آنها توسط توسعه‌دهندگان منتشر شده است و جنبه‌های ناشناخته زیادی برای برنامه‌نویسان وجود دارد. به همین دلیل پیش‌بینی کارآیی کرنل دشوار بوده و در برخی موارد، اجرای برنامه‌ها روی پردازنده گرافیکی مقرون به صرفه نیست [2]، [3]. از طرفی باتوجه به سطح پایین بودن زبان برنامه‌نویسی کودا، لازم است تا تنظیم[5] برنامه‌ها بر اساس ویژگی‌های سخت‌افزار و جزئیات سطوح پایین معماری انجام شود [4]. منظور از تنظیم برنامه، بکارگیری هر تکنیکی است که باعث بهبود کارآیی یک برنامه شود. به دلیل دانش کم از معماری، استفاده از یک تکنیک تنظیم ممکن است باعث بهبود یا افت کارآیی شود. درنتیجه بسیاری از تنظیم‌ها در برنامه‌های کودا توسط برنامه‌نویس به صورت آزمون و خطا انجام می‌شوند [4]. از طرف دیگر در زبان‌های سطح بالاتر مثل زبان‌های مبتنی بر راهنما[6]، کامپایلرهای مبدا به مبدا[7] سعی می‌کنند که علاوه بر ترجمه برنامه‌ها به زبان‌های سطح پایین‌تر، تنظیم خودکار[8] نیز انجام دهند [5] و این عمل ممکن است باعث افت کارآیی شده و عملکرد کرنل کاهش یابد. در نتیجه کامپایلرهای مبدا به مبدا در استفاده از بسیاری از تکنیک‌های تنظیم ناتوان هستند.

یک راه‌کار برای مرتفع کردن چالش‌های مطرح شده، استفاده از یک مدل کارآیی است [6]. مدل کارآیی به شناسایی جنبه‌های ناشناخته دستگاه و گلوگاه‌های کارآیی آن کمک می‌کند. همچنین با استفاده از مدل می‌توان کارآیی یک کرنل را پیش‌بینی کرد [7]. باید توجه شود که در این مقاله منظور از کارآیی، زمان اجرای کرنل است. مدل کارآیی با دریافت یک کرنل ورودی، اطلاعاتی از کرنل استخراج کرده و با استفاده از آنها زمان اجرا را پیش‌بینی می‌کند. بنابراین قبل از اجرای برنامه روی پردازنده گرافیکی، برآوردی خواهیم داشت که آیا اجرای برنامه روی این پردازنده باعث بهبود کارآیی شده و تسریع مناسبی دارد یا خیر؟ از طرفی مدل کارآیی می‌تواند به تنظیم دستی و خودکار برنامه‌ها نیز کمک کند [8]. به این ترتیب که از بین گزینه‌هایی که برای تنظیم وجود دارد، گزینه‌ای انتخاب می‌شود که کرنل حاصل از آن، زمان اجرای کمتری داشته باشد. به این ترتیب از آزمون و خطا توسط برنامه‌نویس پرهیز می‌شود و کامپایلرهای مبدا به مبدا نیز می‌توانند تنظیم بهتری را به صورت خودکار روی کرنل ورودی اعمال کنند [9].

ما مدل‌های کارآیی را به دو دسته تحلیلی و مبتنی بر یادگیری ماشین تقسیم می‌کنیم. در مدل‌های تحلیلی، کارآیی کرنل توسط روابط ریاضی بیان می‌شوند. همچنین ویژگی‌ها و اجزای دستگاه نیز به صورت پارامتر ارائه می‌شوند. مزیت این مدل‌ها این است که قابلیت حمل بالایی دارند و روی دستگاه‌های مختلف قابل بکارگیری هستند [10]. در عوض، با توجه به اینکه جزئیات دستگاه نامشخص است، ایجاد یک مدل تحلیلی بسیار سخت است و معمولا با خطای زیادی همراه است. در مدل‌های مبتنی بر یادگیری ماشین ویژگی‌ها و عملکرد دستگاه توسط روش‌های

[1] Host
[2] Device
[3] Kernel
[4] Parallel Thread Execution
[5] Tuning

[6] Directive-based Language
[7] Source-to-source Compiler
[8] Auto-tuning

یادگیری ماشین عمومی‌سازی[1] می‌شوند و عملا ایجادکننده مدل نیازی به پرداختن به جزئیات ندارد. در عوض این مدل‌ها قابلیت حمل پایینی دارند و برای هر دستگاه باید به طور جداگانه ایجاد شوند [11]. علاوه بر پیش‌بینی زمان اجرای کرنل، از روش‌های یادگیری ماشین برای تخمین بسیاری از پارامترهای و ویژگی‌های پردازنده‌های گرافیکی که به طور مستقیم روابطی برای تخمین آنها نداریم نیز استفاده شده است [12].

در این مقاله یک مدل مبتنی‌بر یادگیری ماشین ایجاد شده است که زمان اجرای یک کرنل کودای ورودی را پیش‌بینی می‌کند. با توجه به تنوع دستگاه‌ها و برنامه‌ها، هیچ مجموعه داده استانداردی برای این مساله وجود ندارد و در هر پژوهش مجموعه داده به صورت مستقل ایجاد می‌شود. بنابراین در این پژوهش نیز ابتدا یک مجموعه داده برای این مدل ایجاد می‌شود. این مجموعه داده بر اساس پروفایلینگ محک‌های واقعی و همچنین با تحلیل ایستای کدهای کودا و PTX کرنل‌ها ایجاد می‌شود. سپس برای پیش‌بینی زمان اجرای کرنل، روش‌های یادگیری ماشین روی این مجموعه داده اعمال می‌شوند. همچنین با استفاده از روش‌های انتخاب ویژگی میزان تاثیر هر یک از ویژگی‌های مجموعه داده روی زمان اجرای کرنل (برچسب خروجی) مشخص می‌شود. با توجه به اینکه برنامه‌های کودا منحصرا روی پردازنده‌های گرافیکی تولید شده توسط شرکت انویدیا اجرا می‌شوند، بنابراین موضوع این مقاله منحصرا این دستگاه‌ها می‌باشند. مهم‌ترین نوآوری این تحقیق این است که معیارهای ایستای مهمی برای کرنل‌های کودا معرفی شده است که برخی از آنها بر اساس تحلیل همزمان کدهای کودا و PTX و برخی دیگر به کمک کامپایلر NVCC ارائه می‌شوند. همچنین با استفاده از سه روش یادگیری ماشین مدل‌های کارآیی برای پیش‌بینی زمان اجرای کرنل کودا ایجاد شده است.

در ادامه این مقاله، ابتدا در بخش 2 پیش‌زمینه‌های مورد نیاز ارائه می‌شوند. سپس در بخش 3، مهم‌ترین کارهای مرتبط با طرح پیشنهادی که پیش از این ارائه شده‌اند، مورد بررسی قرار می‌گیرند. در ادامه در بخش 4، مدل پیشنهادی ارائه شده و در بخش 5 مورد ارزیابی قرار می‌گیرد. سپس در بخش‌های 6 و 7 بحث و نتیجه‌گیری انجام می‌شود.

2. پیش‌زمینه

پردازنده‌های گرافیکی از جهت وضعیت حافظه مشترک با پردازنده مرکزی، به دو دسته یکپارچه[2] و گسسته[3] دسته‌بندی می‌شوند [13]. پردازنده‌های یکپارچه دارای فضای حافظه و آدرس‌دهی مشترک با پردازنده مرکزی می‌باشند، در حالی که پردازنده‌های گرافیکی گسسته حافظه‌های مجزا داشته و برای اجرای یک برنامه روی آنها باید ابتدا داده‌ها منتقل شوند و پس از اجرای برنامه نیز نتایج به حافظه اصلی برگردانده شوند. این رفت و برگشت داده‌ها تاثیر مستقیمی روی کارآیی برنامه‌ها دارد و یک سربار برای اجرای برنامه‌ها محسوب می‌شود [14]. پردازنده‌های گرافیکی که بر روی اکثر کامپیوترهای شخصی نصب شده‌اند، از این نوع هستند.

نسل‌های مختلفی از معماری پردازنده‌های گرافیکی شرکت انویدیا ارائه شده است که در زمان نگارش این مقاله به صورت جدول (1) می‌باشند و از مراجع [15] و [16] استخراج شده است. جزئیات معماری این پردازنده‌ها کمی متفاوت است، اما ساختار کلی آنها مشابه هستند. هر یک از این نسل‌ها، قابلیت‌های محاسباتی متفاوتی دارند که از 1 تا 9 شماره‌گذاری شده‌اند. درک ساختار و جزئیات معماری این پردازنده‌ها در مدل‌سازی آنها و در طراحی و تنظیم برنامه‌های موازی روی آنها نقش مهمی ایفا می‌کند. بنابراین در این بخش سعی شده است تا در حد نیاز این جزئیات ارائه شوند.

1.2. معماری پردازنده‌ی گرافیکی

معماری پردازنده‌های گرافیکی انویدیا که در شکل (1) نشان داده شده است، شامل یک حافظه سراسری، یک حافظه نهان سطح دو، یک زمانبند سطح اول به نام گیگاترد[4]، یک واسط

[1] Generalization

[2] Integrated
[3] Discrete
[4] GigaThread

میزبان[1] و تعدادی چندپردازنده جریانی[2] است که از طریق یک شبکه ارتباطی با هم در ارتباط هستند. چندپردازنده‌های جریانی واحدهایی هستند که هسته‌های پردازشی در داخل این واحد قرار گرفته‌اند و به لحاظ پردازشی مهم‌ترین بخش پردازنده‌های گرافیکی هستند. تعداد و ساختار این واحدها در معماری‌های مختلف متفاوت است. اما در هر دستگاه، چند پردازنده‌های جریانی کاملا همسان هستند. حافظه سراسری تنها حافظه‌ای است که می‌توان از سمت حافظه اصلی میزبان (پردازنده مرکزی) به آن داده رد و بدل کرد. حافظه نهان سطح دو مشترک بین چن پردازنده‌های جریانی است. واحد گیگاترد زمانبندی سطح یک است و بلوک‌هایی از نخ‌ها را جهت اجرا بین چند پردازنده‌های جریانی توزیع می‌کند. واسط میزبان وظیفه برقراری ارتباط و هماهنگی بین میزبان و دستگاه را به عهده دارد.

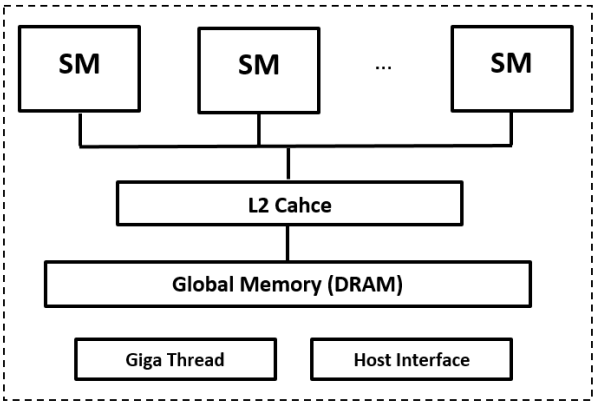
جدول (1): نسل‌های معماری پردازنده‌های گرافیکی انویدیا

معماری	سال انتشار	قابلیت محاسباتی
Tesla	2007	1.x
Fermi	2009	2.x
Kepler	2011	3.x
Maxwell	2013	5.x
Pascal	2015	6.x
Volta	2017	7, 7/2
Turing	2019	7/5
Ampere	2020	8, 8/6, 8/7
Ada Lovelace	2022	8/9
Hopper	2022	9

به لحاظ پردازشی، مهم‌ترین بخش این معماری چندپردازنده‌های جریانی هستند که هسته‌های پردازشی را در خود جای داده‌اند. هر چند پردازنده جریانی شامل تعدادی هسته بوده که پردازنده جریانی[3] نامیده می‌شوند و به صورت یک دستورالعمل چندین داده[4] بکار گرفته شده و نخ‌ها را به طور موازی اجرا می‌کنند.

[1] Host Interface
[2] Streaming Multiprocessor (SM)
[3] Streaming Processor (SP)
[4] Single Instruction Multiple Data (SIMD)

علاوه بر این، در یک چند پردازنده جریانی واحدهای محاسبات با دقت مضاعف[5]، واحد تابعی خاص[6] برای توابع غیرجبری، واحد تنسورفلو[7] برای محاسبات مربوط به یادگیری عمیق[8] و واحدهای بارگذاری و ذخیره داده‌ها[9] وجود دارند. یکی از مهمترین اجزای درونی یک چند پردازنده جریانی واحد زمانبند وارپ[10] است که زمانبند سطح دوم در پردازنده گرافیکی است.

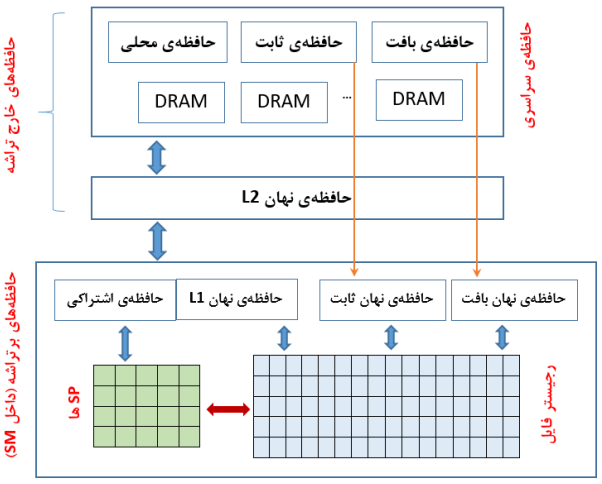


شکل (1): نمای کلی اجزای درونی پردازنده‌های گرافیکی انویدیا

یکی از مهمترین جنبه‌های معماری پردازنده گرافیکی، سلسله مراتب حافظه آن است که در شکل (2) نشان داده شده است. درک این سلسله مراتب و اندازه‌های واحدهای حافظه در بهینه‌سازی کدها تاثیر مهمی دارند. حافظه‌های محلی[11]، بافت[12] و ثابت[13] با حافظه‌ی سراسری یکپارچه هستند. سطح دسترسی حافظه محلی در سطح نخ‌ها است و در صورتی توسط یک نخ استفاده می‌شود که ثبات به اندازه کافی در دسترس نخ نباشد. این حافظه‌ها خارج تراشه[14] بوده و تاخیر دسترسی آنها زیاد است. چند پردازنده‌های جریانی شامل حافظه اشتراکی، حافظه‌های نهان دستورالعمل، حافظه نهان سطح یک، حافظه نهان بافت[15] و

[5] Double Precision Unit (DPU)
[6] Special Function Unit (SFP)
[7] Tensor Processing Unit (TPU)
[8] Deep Learning
[9] Load/Store (LD/ST)
[10] Warp Scheduler
[11] Local Memory
[12] Texture Memory
[13] Constant Memory
[14] Off-chip
[15] Texture Cache memory

حافظه نهان ثابت[1] و تعداد زیادی ثبات هستند که حافظه‌های برتراشه[2] محسوب می‌شوند و سرعت دسترسی بالایی دارند. حافظه اشتراکی در معماری‌های فرمی و کپلر با حافظه نهان سطح یک یکپارچه است [17]، [18]. ولی در معماری مکسول حافظه اشتراکی به عنوان یک واحد مستقل در چند پردازنده جریانی قرار گرفته است و حافظه‌های نهان سطح یک و بافت یکپارچه هستند [19]. از معماری پاسکال به بعد حافظه اشتراکی و حافظه نهان سطح یک یکپارچه شده و قابل پیکربندی هستند [20]- [22]. حافظه نهان سطح یک از معماری فرمی به پردازنده‌های گرافیکی انویدیا اضافه شده‌اند [23]. حافظه‌های بافت و ثابت توسط حافظه‌های نهان بافت و ثابت کش می‌شوند. برخلاف حافظه سراسری که فقط برای خواندن کش می‌شود، حافظه محلی هم برای خواندن و هم برای نوشتن کش می‌شود [24]. نخ‌هایی که روی یک چندپردازنده جریانی اجرا می شوند از طریق منابع آن می‌توانند با هم همگام شوند [25].



شکل (2): سلسله‌مراتب حافظه پردازنده‌های گرافیکی انویدیا

2.2. مدل برنامه‌نویسی کودا

یک برنامه کودا به دو بخش میزبان و دستگاه تقسیم می‌شود. بخش دستگاه، تابعی است که کرنل نامیده شده و برای اجرا به دستگاه ارسال می‌شود تا جریان‌های ورودی را به جریان‌های خروجی تبدیل کند [1]. اجرای یک برنامه کودا شامل اجرای یک یا چند کرنل چندنخی است که به صورت داده‌موازی[3] روی پردازنده گرافیکی اجرا می‌شوند. محاسبات به وسیله سلسله مراتبی از نخ‌ها انجام می‌شوند. در واقع کرنل یک برنامه ترتیبی است که بر اساس آن تعداد زیادی نخ ایجاد می‌شود. این تعداد از طریق پیکربندی کرنل تعیین می‌شود. هنگام راه‌اندازی کرنل (از طریق برنامه میزبان)، همه نخ‌ها در قالب یک گرید و در قالب چندین بلوک سازماندهی می‌شوند که پیکربندی کرنل را ایجاد می‌کنند. این پیکربندی به صورت $<<< n_b . n_t >>>$ مشخص می‌شود. n_b اندازه گرید (تعداد بلوک‌ها) و n_t اندازه بلوک (تعداد نخ‌ها در هر بلوک) است. اندازه بلوک‌ها ثابت و مشابه هستند. تعداد نخ‌های ایجاد شده برابر $n_b \times n_t$ است. البته اندازه‌های گرید و بلوک می‌توانند چندبعدی نیز باشند. به طور معمول این اعداد برحسب اندازه داده‌های ورودی یا اندازه مساله تعیین می‌شوند و نقش مهمی در موازی‌سازی دارند. نخ‌های یک بلوک روی یک چندپردازنده جریانی اجرا می‌شوند و می‌توانند با هم همگام شده و روی داده‌های مشترک کار کنند. مدل برنامه‌نویسی کودا برای موازی‌سازی از نگاشت نخ‌ها به هسته‌های پردازشی استفاده می‌کند که نقش بسیار مهمی در عملکرد برنامه‌ی موازی دارد. نخ‌ها در قالب دسته‌های 32 تایی به نام وارپ دسته‌بندی می‌شوند که کوچک‌ترین واحد زمانبندی در پردازنده گرافیکی هستند. نخ مفهومی نرم‌افزاری و وارپ مفهومی سخت‌افزاری است. در عمل این وارپ‌ها هستند که اجرا می‌شوند و در درون چندپردازنده‌های جریانی توسط زمانبندهای وارپ زمانبندی می‌شوند. نخ‌های یک وارپ به طور همزمان کدگشایی[4]، صادر[5] و اجرا می‌شوند. هر 32 نخ عضو یک وارپ به طور همزمان در حال اجرای یک دستورالعمل مشابه بوده و ثبات شمارنده برنامه مشترک دارند. ارتباط بین میزبان و دستگاه از طریق متن کودا[6] میسر می‌شود. با فراخوانی اولین تابع کتابخانه‌ای از سمت میزبان، این متن ایجاد می‌شود [24]. به دلیل

[1] Constant Cache Memory
[2] On-chip

[3] Data-parallel
[4] Decode
[5] Issue
[6] CUDA Context

متمایز بودن حافظه اصلی از حافظه سراسری پردازنده گرافیکی گسسته، برای راه‌اندازی یک کرنل، ابتدا داده‌ها باید از حافظه اصلی میزبان به حافظه سراسری پردازنده گرافیکی ارسال شوند و پس از انجام محاسبات، نتایج باید برگردانده شوند. در واقع مدل برنامه‌نویسی روی این پردازنده‌ها به صورت یک برنامه چندین داده[1] است [26].

3. مرور کارهای گذشته

در بخش 1، مدل‌های کارآیی که برای پردازنده گرافیکی ارائه شده‌اند را به دو دسته تحلیلی و مبتنی‌بر یادگیری ماشین دسته‌بندی کردیم. در این بخش مدل‌هایی را بررسی می‌کنیم که از روش‌های یادگیری برای مدل‌سازی استفاده کرده‌اند. این مدل‌ها با استفاده از روش‌های هوش مصنوعی سعی می‌کنند که رابطه‌ای بین داده‌ها استخراج کنند. این عمل ازطریق داده‌های از قبل شناخته شده انجام می‌شود و مرحله آموزش نام دارد. سپس این نتایج برای پیش‌بینی داده‌های ناشناخته بکار می‌روند [27]. اونیل و همکاران [28]، با اجرای بارکاری روی شبیه‌ساز، داده‌های آماری را جمع‌آوری کرده و با روش‌های یادگیری ماشین چارچوبی برای پیش‌بینی کارآیی ارائه کردند. آنها برای تحلیل‌های خطی از روش رگرسیون و برای تحلیل‌های غیرخطی از روش رگرسیون جنگل تصادفی استفاده کرده‌اند و به خطای ۷ تا 9 درصد دست یافتند. وو و همکاران [29]، مدلی بر اساس یادگیری ماشین برای پیش‌بینی توان و کارآیی پردازنده گرافیکی ارائه کردند. آنها مجموعه‌ای از کرنل‌ها را روی پردازنده‌های گرافیکی مختلف اجرا کرده و داده‌ها را با استفاده از شمارنده کارآیی[2] جمع‌آوری کردند. آنها کرنل‌ها را بر اساس ویژگی‌های آنها و با استفاده از الگوریتم خوشه‌بندی K-میانگین[3] به 22 دسته مختلف دسته‌بندی کردند. این ویژگی‌ها شامل نرخ خواندن و نوشتن با حافظه، نرخ پهنای باند مورد استفاده و غیره بوده است. سپس هر دسته را با استفاده از شبکه‌های عصبی آموزش داده و به نرخ خطای 10 تا 15 درصد دست یافته‌اند.

در برخی از پژوهش‌ها روش‌های یادگیری ماشین برای محاسبه یک پارامتر از مدل‌های تحلیلی استفاده شده‌اند و یا در بخشی از مدل برای اجرای یک عملیات مثل دسته‌بندی استفاده شده‌اند. هوانگ و همکاران [30]، برای محاسبه تاخیرهایی که توسط سیاست زمانبندی قابل پنهان‌سازی است، مفهوم دستورالعمل‌های غیرهمپوشانی را معرفی کردند. آنها روند اجرای دستورالعمل‌های هر وارپ را ردیابی و از الگوریتم خوشه‌بندی K-میانگین برای یافتن یک وارپ به عنوان نماینده استفاده کردند. این وارپ محور محاسبه زمان اجرای کرنل است. سپس آنها مجموع هزینه‌های محاسباتی را با تعداد سیکل‌هایی از ارتباطات که توسط محاسبات پنهان نشده‌اند را جمع می‌کنند. رسیوس و هلدرمن [31]، یک مدل‌سازی تحلیلی برای تخمین کارآیی کرنل ارائه کرده‌اند که در آن برای تخمین پارامترهای کارآیی و انتقال داده از ریزمحک‌زنی استفاده کرده‌اند. در این مدل مدت زمان سربار انتقال داده‌ها بین میزبان و دستگاه از طریق رگرسیون تخمین زده شده است. در مرجع [12] برای تخمین زمان انتقال داده‌ها بین حافظه اصلی میزبان و حافظه سراسری دستگاه، از روش‌های یادگیری ماشین شبکه‌های عصبی مصنوعی، جنگل تصادفی و رگرسیون خطی استفاده شده است. همچنین در این پژوهش یک مدل تحلیلی ارائه شده که در آن برای محاسبه پارامتر λ از روش‌های یادگیری ماشین استفاده شده است. آماریس و همکاران [32]، مدل تحلیلی که در مرجع [33] کرده بودند را با سه روش یادگیری ماشین مقایسه کردند. آنها نشان دادند که روش‌های یادگیری ماشین برای پیش‌بینی زمان اجرا از مدل تحلیلی آن‌ها دقت بهتری ارائه می‌کند.

در جدول (2) به طور خلاصه برخی مدل‌های مبتنی بر یادگیری ماشین که در سایر پژوهش‌ها ارائه شده‌اند بررسی شده است. در برخی از روش‌ها انتخاب ویژگی انجام شده است و در برخی از پژوهش‌ها کرنل‌ها دسته‌بندی شده و برای هر دسته یک مدل جداگانه‌ای ساخته شده است. این جدول نشان می‌دهد که روش‌هایی وجود دارند که هنوز روی این مساله بکارگرفته نشده‌اند و در بخش بعدی این روش‌ها را امتحان کنیم.

[1] Single Program Multiple Data (SPMD)
[2] Performance Counter
[3] K-means

جدول (2): مرور مدل‌های کارآیی مبتنی بر یادگیری ماشین

مرجع	سال	روش‌های یادگیری	انتخاب ویژگی	دسته‌بندی کرنل‌ها	درصد خطا	
					میانگین	بدترین
[34]	2018	SVM, LReg, K-NN			3/32	9
[35]	2020	K-NN	✓	✓	گ.ن.	10
[36]	2021	OLS, MLP, CNN, SVM, RF	✓		8 ~ 26	گ.ن.
[37]	2017	Lasso, LReg, NNLS, RF, OLS			14	23
[38]	2016	SVM, MLP			7 ~ 14	16/37
[32]	2016	RF, LReg, SVM			گ.ن.	2/08
[39]	2015	SVM		✓	10 ~ 15	گ.ن.
[40]	2016	K-NN, LReg			10	35
[7]	2022	DL	✓		11	27
[41]	2018	DL, CNN			10	23

4. مدل پیشنهادی

مراحل ایجاد مدل پیشنهادی در شکل (3) نشان داده شده است. در گام اول، مجموعه داده باید ایجاد شود و این مجموعه داده به دو بخش آموزش و آزمون تقسیم می‌شود. مجموعه داده آموزش به روال آموزش، و مجموعه داده آزمون به روال آزمون ارسال می‌شوند. در روال آموزش سه روش ماشین بردار پشتیبان، شبکه‌های عصبی مصنوعی و ماشین یادگیری افراطی اجرا می‌شوند. حاصل این مرحله مدل‌های آموزش‌یافته است که برای روال آزمون ارسال می‌شوند. همچنین در این مرحله از اعتبارسنجی متقاطع K-دسته[1] نیز استفاده می‌شود. در روال آزمون مدل‌های آموزش‌یافته با استفاده مجموعه داده آزمون ارزیابی می‌شوند و خطای هر مدل گزارش می‌شود. در ادامه هر یک از این مراحل به تفصیل تشریح می‌شوند.

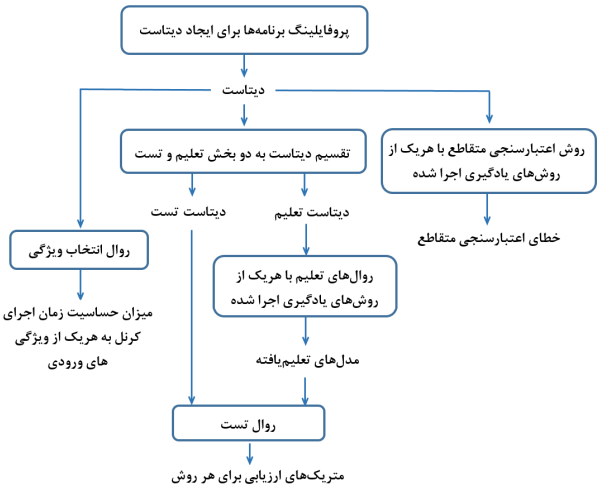
1.4. ایجاد مجموعه داده

برای ایجاد مجموعه داده از محک‌های دنیای واقعی و مجموعه مثال‌های انویدیا استفاده شده است که در بخش بعدی معرفی می‌شوند. برای این منظور فقط برنامه‌های منظم بکار گرفته شده‌اند زیرا زمان اجرای کرنل‌های کودا به شدت به دسترسی‌های حافظه وابسته است. در برنامه‌های منظم الگوی دسترسی به حافظه تقریبا مشخص است و بنابراین احتمال پیگیری موفق داده‌ها در سلسله مراتب حافظه بیشتر است. در عوض، در برنامه‌های نامنظم احتمال موفقیت یا خطای حافظه نهان چندان قابل پیش‌بینی نیست. از طرفی زمان دسترسی به حافظه سراسری و حافظه‌های نهان تفاوت بسیار محسوسی دارند و بنابراین پراکندگی زمان اجراهای ثبت شده برای این برنامه‌ها بسیار متنوع هستند و این مساله باعث می‌شود که در مرحله ایجاد مجموعه داده، ممکن است داده‌های بد تولید شوند. به همین دلیل در این پژوهش فقط به برنامه‌های منظم می‌پردازیم. علاوه بر این، ما برای کرنل‌های حافظه‌گرا و محاسبه‌گرا مجموعه داده‌های مجزایی ایجاد کردیم. نتایج نشان دادند که در این حالت عمومی‌سازی زمان اجرای کرنل توسط روش‌های یادگیری ماشین بهتر انجام شده است.

مهمترین نکته در ایجاد مجموعه داده مشخص کردن معیارهایی است که ویژگی‌های ورودی مجموعه داده را تشکیل می‌دهند. این معیارها با استفاده از تحلیل ایستای کدهای کودا و PTX استخراج می‌شوند. کد PTX یک ترجمه شبه‌اسمبلی از کد کودا است که برای ایجاد آن نیازی به اجرای برنامه نیست و با استفاده از گزینه ترجمه ptx- ازطریق مترجم NVCC به دست می‌آید.

[1] K-fold Cross Validation

در جدول (3) این معیارها لیست شده‌اند. بنابراین مجموعه داده شامل 16 ویژگی ورودی و یک برچسب خروجی است. برچسب خروجی زمان اجرای کرنل است که با پروفایلینگ اجرای کرنل اندازه‌گیری می‌شود. این زمان برحسب میلی‌ثانیه ثبت می‌شود.



شکل (3): مراحل ایجاد مجموعه داده و مدل و انجام آزمایشات

هفت ویژگی اول در جدول (3) با استفاده از تحلیل کدهای کودا محاسبه می‌شوند. n اندازه ورودی است و فرض بر این است که عددی مشخص است. ویژگی cpx پیچیدگی محاسباتی (مرتبه اجرا) الگوریتم کرنل است که به طور معمول به صورت مجانبی برحسب اندازه ورودی تعیین می‌شود. با توجه به مشخص بودن اندازه ورودی این ویژگی نیز به صورت یک عدد محاسبه می‌شود. s اندازه مساله است و بر اساس ماهیت مساله و الگوریتم آن تعیین می‌شود. به عنوان مثال در مورد مساله ضرب ماتریس‌ها با ابعاد $n \times n$ اندازه مساله n^2 و پیچیدگی آن n^3 است. α پارامتر شدت محاسباتی است که در راهنمای برنامه‌نویسی انویدیا [42] معرفی شده است و از تقسیم تعداد دستورالعمل‌هایی که از حافظه برتراشه (دستورالعمل‌های محاسباتی) استفاده می‌کنند، بر تعداد دستوراتی که از حافظه خارج تراشه (دستورالعمل‌های حافظه) استفاده می‌کنند، به دست می‌آید. در محاسبه این پارامتر، دستورالعمل همگام‌سازی درون بلاکی (__synchthreads) جزء دستورالعمل‌های برتراشه محسوب می‌شود. چراکه این عملیات با استفاده از حافظه اشتراکی انجام می‌شود. این پارامتر با تحلیل کدهای کودا و با شمارش تعداد دستورالعمل‌های کرنل استخراج می‌شود. صرف نظر از اینکه این دستورات چند بار اجرا می‌شوند. پارامترهای n_b و n_t و w به ترتیب اندازه گرید، اندازه بلوک و تعداد وارپ‌ها در یک بلوک هستند و با توجه به پیکربندی کرنل تعیین می‌شوند.

جدول (3): معیارهای استخراج شده به عنوان ویژگی‌های ورودی.

معیار	توضیحات
n	اندازه ورودی
s	اندازه مساله
α	شدت محاسباتی[1]
n_b	تعداد بلوک‌های کرنل (اندازه گرید)
n_t	تعداد نخ‌ها در یک بلوک (اندازه بلوک)
w	تعداد وارپ‌ها در یک بلوک
cpx	پیچیدگی محاسباتی کرنل
l	تعداد کل دستورالعمل‌های PTX
l_c	تعداد دستورالعمل‌های محاسباتی
l_{mr}	تعداد دستورالعمل‌های خواندن از حافظه سراسری
l_{mw}	تعداد دستورالعمل‌های نوشتن در حافظه سراسری
l_{shr}	تعداد دستورالعمل‌های خواندن از حافظه اشتراکی
l_{shw}	تعداد دستورالعمل‌های نوشتن در حافظه اشتراکی
l_s	تعداد دستورالعمل‌های همگام‌سازی
l_{dpu}	تعداد دستورالعمل‌های اعشاری دقت مضاعف
l_{sfu}	تعداد دستورالعمل‌های واحد تابعی خاص
dpc	درجه وابستگی داده‌ای
$ShMem$	تعداد کلمات حافظه اشتراکی مورد نیاز یک بلوک
n_{reg}	تعداد رجیسترهای مورد نیاز یک نخ

هشت ویژگی بعدی در جدول (3) با استفاده از تحلیل کدهای PTX به دست می‌آیند. معیارهای l، l_c، l_{mr}، l_{mw}، l_{shr} و l_{shw} و l_s تعداد دستوراتی از کدهای PTX هستند که اجرا می‌شوند. این پارامترها با تعداد دستورات متفاوت است. مواردی نظیر حلقه‌های تکرار باعث می‌شوند که برخی از دستورات بیش از

[1] Arthmetic Intensity

یک بار اجرا شوند. باید توجه شود که در اینجا فرض ما بر این است که تعداد تکرار حلقه‌ها در کرنل‌ها به مقدار داده ورودی بستگی ندارند. چراکه در این حالت پیش‌بینی دقیق تعداد تکرارهای حلقه به صورت ایستا غیرممکن است و اساسا پیش‌بینی ایستای زمان اجرا نیز با خطای بسیار زیادی همراه خواهد بود. بنابراین ما کرنل‌هایی را انتخاب کردیم که تعداد تکرار حلقه‌ها به اندازه ورودی یا اندازه مساله بستگی داشته باشد و مستقل از مقادیر ورودی داده‌ها باشند. معیار l_c تعداد دستورالعمل‌های محاسباتی اجرا شده است که شامل مواردی است که نتایج در یک حافظه برتراشه ذخیره می‌شوند. l_{mr} و l_{mw} به ترتیب تعداد دستورالعمل‌های خواندن و نوشتن در حافظه سراسری هستند. باید توجه شود که خیلی مهم است که این دو دستورالعمل تفکیک شوند. چون عملیات نوشتن در حافظه سراسری برخلاف خواندن از آن کش نمی‌شوند [1] و بنابراین تاخیر این دو عملیات با هم متفاوت است. l_{shr} و l_{shw} به ترتیب تعداد دستورالعمل‌های خواندن و نوشتن در حافظه اشتراکی هستند. ما از طریق ریزمحک‌زنی تاخیر خواندن و نوشتن در حافظه اشتراکی را روی دستگاه 940MX که معماری مکسول است، امتحان کردیم. تاخیر خواندن از حافظه اشتراکی این دستگاه 6 سیکل ساعت و تاخیر نوشتن در آن 20 سیکل ساعت است. به همین دلیل این دو پارامتر را از دستورالعمل‌های محاسباتی تفکیک کردیم. بین ویژگی‌های مرتبط با تعداد دستورالعمل‌های اجرا شده رابطه (1) باید برقرار باشد.

$$l = l_c + l_{shr} + l_{shw} + l_{mr} + l_{mw} + l_s + l_{DPU} + l_{SFU} \quad (1)$$

ویژگی dpc برآوردی از میزان وابستگی داده‌ای بین دستورالعمل‌های PTX را نشان می‌دهد. وابستگی داده‌ای زمانی رخ می‌دهد که نتیجه یک دستورالعمل، به عنوان ورودی در دستورالعمل دیگری استفاده شود. وابستگی داده‌ای یک گلوگاه مهم برای کارآیی برنامه‌های کودا است. چراکه باعث می‌شود اجرای موازی دستورالعمل‌ها در واحدهای تابعی[1] درون SM ها به تاخیر بیافتند. به این ترتیب که اگر دستورالعمل j-ام به دستورالعمل i-ام ($i < j$) وابستگی داده‌ای داشته باشد، آنگاه صدور[2] دستورالعمل j-ام تا کامل شدن اجرای دستورالعمل i-ام به تاخیر می‌افتد [1]. همچنین اگر بین دو دستورالعمل متوالی وابستگی داده‌ای وجود داشته باشد، صدور دوگانه[3] دستورالعمل‌ها توسط زمانبند وارپ انجام نمی‌گیرد. برای محاسبه dpc، بردار U معرفی می‌شود. در کد PTX کرنل، اگر دستورالعمل j-ام اولین دستورالعملی باشد که دستورالعمل i-ام ($i < j$) به آن وابستگی داده‌ای داشته باشد، آنگاه $U[i] = j$ خواهد بود و به ازای این دستور مقدار dpc به اندازه $\frac{1}{U[i]-i}$ افزوده می‌شود. در جدول (4) کدهای PTX برنامه NN از محک Rodinia ارائه شده است. در این جدول بردار U و مقادیری که به ازای هر دستورالعمل به dpc اضافه می‌شوند نیز مشخص شده‌اند. مقدار این پارامتر برای هر برنامه فقط یکبار و از طریق رابطه (2) محاسبه می‌شود. در این رابطه k تعداد خطوط برنامه PTX است و در جدول (4) این عدد برابر 29 است. در واقع از مجموع مقادیر میانگین گرفته شده است.

$$dpc = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{U[i]-i} \quad (2)$$

پارامتر dpc هرچقدر مقدار کمتری داشته باشد، نشان می‌دهد که امکان اجرای موازی دستورالعمل‌ها در واحدهای تابعی بیشتر است. توجه کنید که هرچه وابستگی داده‌ای بین دستورالعمل‌هایی باشد که فاصله بیشتری نسبت به هم داشته باشند (به عبارت دیگر نتیجه یک دستورالعمل دیرتر مورد استفاده قرار گرفته باشد)، این پارامتر مقدار کمتری دارد. به عبارت دیگر هرچقدر مقدار $U[i] - i$ مقدار کوچکتری داشته باشد، بدان معنی است که نتیجه دستورالعمل i-ام زودتر مورد استفاده قرار می‌گیرد و بنابراین موازی‌سازی در واحدهای تابعی کمتر است. همچنین احتمال صدور دوگانه دستورالعمل‌ها در زمانبند وارپ نیز کاهش می‌یابد. این پارامتر برای هر کرنل فقط یک بار محاسبه می‌شود و این عمل با استفاده از تحلیل کدهای PTX انجام می‌شود.

[1] Functional Units

[2] Issue
[3] Dual-issue

جدول (4): نحوه محاسبه معیار DPC در برنامه NN.

#Inst	Instructions	U	$\frac{1}{\mathrm{U}[i]-i}$
1	ld.param.u64 %rd1,[Kernel_param_0];	15	1/(15-1)
2	ld.param.u64 %rd2,[Kernel_param_1];	16	1/(16-2)
3	ld.param.u32 %r2,[Kernel_param_2];	13	1/(13-3)
4	ld.param.f32 %f1,[Kernel_param_3];	22	1/(22-4)
5	ld.param.f32 %f2,[Kernel_param_4];	24	1/(24-5)
6	mov.u32 %r3, %ctaid.y;	9	1/(9-6)
7	mov.u32 %r4, %nctaid.x;	9	1/(9-7)
8	mov.u32 %r5, %ctaid.x;	9	1/(9-8)
9	mad.lo.s32 %r6, %r3, %r4, %r5;	12	1/(12-9)
10	mov.u32 %r7, %ntid.x;	12	1/(12-10)
11	mov.u32 %r8, %tid.x;	12	1/(12-11)
12	mad.lo.s32 %r1, %r6, %r7, %r8;	13	1/(13-12)
13	setp.ge.s32 %p1, %r1, %r2;	14	1/(14-13)
14	@%p1 bra BB0_2;	0	0
15	cvta.to.global.u64 %rd3, %rd1;	20	1/(20-15)
16	cvta.to.global.u64 %rd4, %rd2;	18	1/(18-16)
17	mul.wide.s32 %rd5, %r1, 4;	18	1/(18-17)
18	add.s64 %rd6, %rd4, %rd5;	28	1/(28-18)
19	mul.wide.s32 %rd7, %r1, 8;	20	1/(20-19)
20	add.s64 %rd8, %rd3, %rd7;	21	1/(21-20)
21	ld.global.f32 %f3, [%rd8];	22	1/(22-21)
22	sub.f32 %f4, %f1, %f3;	26	1/(26-22)
23	ld.global.f32 %f5, [%rd8+4];	24	1/(24-23)
24	sub.f32 %f6, %f2, %f5;	25	1/(25-24)
25	mul.f32 %f7, %f6, %f6;	26	1/(26-25)
26	fma.rn.f32 %f8, %f4, %f4, %f7;	27	1/(27-26)
27	sqrt.rn.f32 %f9, %f8;	28	1/(28-27)
28	st.global.f32 [%rd6], %f9;	0	0
--	BB0_2:	0	0
--	ret;	0	0

دو ویژگی آخر در جدول (3) یعنی معیارهای $ShMem$ و n_{reg} با کمک مترجم NVCC محاسبه می‌شوند. برای این منظور از گزینه ترجمه ptxas-options=-v-- استفاده می‌شود. هر کلمه در حافظه را معادل 4 بایت در نظر می‌گیریم که معادل اندازه یک عدد اعشاری است. اهمیت این دو معیار به دلیل تاثیر در زمانبندی سطح یک در گیگاترد است. هرچقدر ثبات‌های تخصیصی و یا میزان حافظه اشتراکی مورد استفاده یک بلوک بیشتر باشد، آنگاه تعداد بلوک‌هایی توسط گیگاترد به صورت همزمان به یک چندپردازنده جریانی تخصیص می‌یابد، کاهش می‌یابد. لذا اجرای همروند بلوک‌ها در یک چندپردازنده جریانی کاهش می‌یابد. این مساله به دلیل محدودیت منابع درون یک چندپردازنده جریانی است. بنابراین این معیارها در زمان اجرای کرنل تاثیرگذار هستند و به همین دلیل به عنوان ویژگی‌های ورودی مجموعه داده در نظر گرفته شده‌اند.

پس از ایجاد مجموعه داده، با استفاده از روش حداکثر-حداقل[1] عملیات نرمال‌سازی روی مجموعه داده اجرا شده است و ویژگی‌های ورودی به بازه صفر تا یک نگاشت شده‌اند. سپس به صورت تصادفی 80 درصد داده‌ها برای آموزش و 20 درصد باقیمانده را برای آزمون در نظر گرفته شده‌اند. دو مجموعه داده برای کرنل‌های محاسبه‌گرا و حافظه‌گرا ایجاد شده است و از هر دو مجموعه داده برای عملیات‌های انتخاب ویژگی و اعتبارسنجی متقاطع K-دسته‌ای استفاده کردیم.

2.4. روال‌های آموزش و آزمون

روال‌های آموزش و آزمون با استفاده از جعبه ابزار متلب برای سه روش شبکه‌های عصبی مصنوعی، ماشین بردار پشتیبان و ماشین یادگیری افراطی اجرا شده است. همچنین همان‌طور که در شکل (3) نشان داده شده است، از روش اعتبارسنجی متقاطع K-دسته نیز استفاده شده است. برای این منظور مقدار K=10 انتخاب شده است. هال [43]، نشان داده است که در مسائل پیش‌بینی، بهترین مقدار برای K برابر 10 است. روش اعتبارسنجی متقاطع روی کل داده‌های نرمال‌شده (قبل از تفکیک

[1] Max-Min

به مجموعه داده‌های آموزش و آزمون) اجرا شده است.

3.4. انتخاب ویژگی

در جدول (3) تعداد 16 معیار ایستا را معرفی کردیم که روی زمان اجرای کرنل کودا موثر هستند. این معیارها ویژگی‌های ورودی مجموعه داده را تشکیل می‌دهند. با استفاده از روش‌های انتخاب ویژگی می‌توان تعیین کرد که میزان حساسیت زمان اجرای کرنل (برچسب خروجی در مجموعه داده) به هریک از ویژگی‌های ورودی مجموعه داده چقدر است. این اطلاعات راهنمای بسیار خوبی برای برنامه‌نویس در تنظیم دستی برنامه‌های کودا است. با شناسایی مهمترین پارامترهای موثر بر زمان اجرا، در هنگام تنظیم برنامه، برنامه‌نویس گزینه‌های کمتری پیش رو خواهد داشت و بنابراین می‌تواند تنظیم را آگاهانه‌تر، دقیق‌تر و سریع‌تر انجام دهد [44]. روش انتخاب ویژگی استفاده شده در اینجا روش حداقل افزونگی حداکثر ارتباط [1] است و با استفاده از جعبه ابزار متلب پیاده‌سازی شده است. از روش‌های انتخاب ویژگی برای کاهش تعداد ویژگی‌ها و در برخی موارد بهبود کارآیی مدل هم استفاده می‌شود [45]، [46]. اما در اینجا هدف کاهش تعداد ویژگی نیست.

5. ارزیابی مدل پیشنهادی

آزمایشاتی برای تعیین میزان کارآیی مدل‌ها انجام شده است. هدف تعیین دقت مدل‌های آموزش‌یافته در پیش‌بینی زمان اجرای کرنل‌های کودا و همچنین تعیین میزان تاثیرگذاری هر یک از ویژگی‌های ورودی روی زمان اجرای کرنل‌های کودا است.

1.5. روش‌شناسی آزمایشات

مشخصات دستگاه استفاده شده در آزمایشات در جدول (5) نشان داده شده است. معماری این دستگاه مکسول است که از نسل پنجم پردازنده‌های گرافیکی انویدیا است. برای ایجاد مجموعه‌های داده، محک‌های دنیای واقعی در محیط اوبونتو 16/1 با استفاده از کودای 9 اجرا شده‌اند. برای پیاده‌سازی روال‌های یادگیری ماشین از محیط متلب 2021 در سیستم‌عامل ویندوز 10 استفاده شده است.

جدول (5): مشخصات دستگاه استفاده شده در آزمایشات.

پارامتر	مقدار
Video Card	GeForce 940MX
GPU Architecture	Maxwell
GPU Model	GM107-B
CUDA Capability (CC)	5
# CUDA Cores	512
SM# × Cores/SM	4 × 128
Number of 32-bit registers per SM	64 K
Shared Memory Size	48 KB
L1 Cache Size	48 KB
L2 Cache Size	1 MB
Size of L2 Cache Line	32 B
Size of Global Memory	2 GB
Maximum number of threads per block	1024
Maximum number of threads per SM	2048
CUDA Version	9
CPU	Intel Corei7

مهمترین معیار ارزیابی در آزمایشات، میزان دقت پیش‌بینی است که به صورت درصد خطا ارائه می‌شود. زمان ارائه شده توسط مدل و زمان اندازه‌گیری شده در اجرای واقعی کرنل با هم مقایسه شده و میزان خطای مدل تعیین می‌شود. علاوه بر این، برای هر یک از روش‌های یادگیری ماشین، مدت زمان روال‌های آموزش و آزمون نیز گزارش خواهند شد.

برای ایجاد مجموعه‌های داده از محک‌های دنیای واقعی و معتبر استفاده شده است که فهرست آنها در جدول (6) ارائه شده است. در این جدول مشخص شده است که محک‌ها از چه مجموعه‌هایی انتخاب شده‌اند و در کدام دسته از کرنل‌های حافظه‌گرا یا محاسبه‌گرا قرار گرفته‌اند. در این مقاله روی کرنل‌های منظم متمرکز شده است و کرنل‌های نامنظم موضوع

[1] Minimum Redundancy Maximum Relevance (MRMR)

بحث این مقاله نیست. محاسبه‌گرا یا حافظه‌گرا بودن کرنل‌ها بر مبنای پژوهش شکفته و همکاران [47]، انجام شده است که وضعیت 25 کرنل را از این جهت مشخص کرده‌اند. هر سطر از رکوردهای ذخیره شده در مجموعه داده در اثر اجرای یک نمونه از کرنل‌ها با پیکربندی‌ها و اندازه‌های ورودی‌های مختلف استخراج شده‌اند. برای هر کرنل ویژگی‌های ورودی در جدول (3) محاسبه شده‌اند و پس از اجرای کرنل و محاسبه زمان اجرای آن یک سطر از مجموعه داده به دست می‌آید.

جدول (6): محک‌های استفاده شده برای ساخت مجموعه‌های داده.

C/M[1]	کرنل	برنامه	محک
M	srad_cuda_1	SRAD v2	Rodinia 3.1
M	srad_cuda_2	SRAD v2	Rodinia 3.1
M	euclid	NN	Rodinia 3.1
M	addKernel	Vec Add	CUDA Samples
C	calculate_temp	Hotspot	Rodinia 3.1
C	matrixMulCUDA	Matrix Multiply	CUDA Samples
C	Fan1	Gaussian	Rodinia 3.1
C	Fan2	Gaussian	Rodinia 3.1

2.5. نتایج آزمایشات

در این بخش نتایج حاصل از اجرای روش‌های یادگیری ماشین روی مجموعه‌های داده ارائه می‌شوند. مجموعه‌های داده بر اساس ویژگی‌های ایستای کرنل‌ها ایجاد شده‌اند. برخی از ویژگی‌ها مختص هر کرنل بوده و برای هر کرنل فقط یکبار محاسبه می‌شوند. در جدول (7) مقادیر محاسبه شده برای کرنل‌های محاسبه‌گرا و در جدول (8) مقادیر محاسبه شده برای کرنل‌های حافظه‌گرا ارائه شده‌اند. مقدار ویژگی cpx در مجموعه داده کرنل‌های حافظه‌گرا برای همه نمونه‌ها برابر یک است. لذا این ویژگی از این مجموعه داده حذف شده است.

جدول (7): ویژگی‌های استخراج شده برای کرنل‌های محاسبه‌گرا.

ویژگی	Hotspot	MM	Gussian,Fan1	Gussian,Fan2
α	64/33	50/66	1	2
cpx	1	3	1	1
l	272	1145	32	57
l_c	227	461	28	49
l_{mr}	2	22	2	6
l_{mw}	1	1	1	2
l_{shr}	14	640	0	0
l_{shw}	5	1	0	0
l_s	5	20	0	0
n_{dpu}	14	0	0	0
n_{sfu}	4	0	1	0
dpc	0/6198	0/5624	0/6826	0/6385
$ShMem$	768	2048	0	0
n_{reg}	34	22	12	11

جدول (8): ویژگی‌های استخراج شده برای کرنل‌های حافظه‌گرا

ویژگی	NN	Srad,Kernel1	Srad,Kernel2	VecAdd
α	8/33	3/78	4/125	1
cpx	1	1	1	1
l	28	276	149	18
l_c	24	232	115	15
l_{mr}	2	9	10	2
l_{mw}	1	5	1	1
l_{shr}	0	6	5	0
l_{shw}	0	12	7	0
l_s	0	4	5	0
n_{dpu}	0	0	6	0
n_{sfu}	1	4	0	
dpc	0/5738	0/5717	0/5852	0/5943
$ShMem$	0	6144	5120	0
n_{reg}	9	19	17	10

[1] Computation-bound/Memory-bound

سه روش ماشین یادگیری افراطی (ELM)، شبکه‌های عصبی پرسپترون چندلایه (MLP) و ماشین بردار پشتیبان رگرسیونی (SVMR) برای پیش‌بینی زمان اجرا پیاده‌سازی شده‌اند. خطای پیش‌بینی این روش‌ها روی دو مجموعه داده محاسبه‌گرا و حافظه‌گرا به ترتیب در جداول (9) و (10) گزارش شده‌اند. برای هر روش خطای اعتبارسنجی متقاطع 10-دسته‌ای و خطای آزمون گزارش شده‌اند. روش‌های یادگیری با پیکربندی‌های مختلف فراخوانی می‌شوند که این پیکربندی‌ها در دقت آنها تاثیرگذار هستند. در روش MLP تعداد لایه‌ها و تعداد نورون‌های هر لایه و در روش ELM تعداد نورون‌های لایه‌ی مخفی و تابع فعالیت[1] استفاده شده در فرآیند آموزش روی دقت تاثیر مستقیمی دارند. ما با استفاده از آزمون و خطا بهترین پیکربندی را یافته و آنها را گزارش کرده‌ایم.

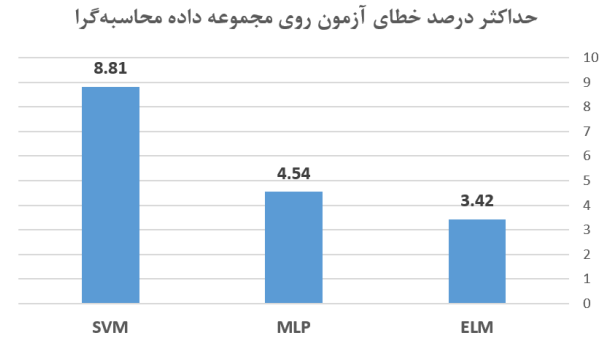
جدول (9): دقت روش‌های یادگیری روی مجموعه داده محاسبه‌گرا.

روش	معیار ارزیابی	درصد خطا	
		حداکثر	میانگین
ELM	خطای اعتبارسنجی 10-دسته‌ای	3/42	0/97
	خطای آزمون	2/78	1/006
MLP	خطای اعتبارسنجی 10-دسته‌ای	5/18	0/007
	خطای آزمون	4/54	0/006
SVMR	خطای اعتبارسنجی 10-دسته‌ای	10/78	0/016
	خطای آزمون	8/81	0/003

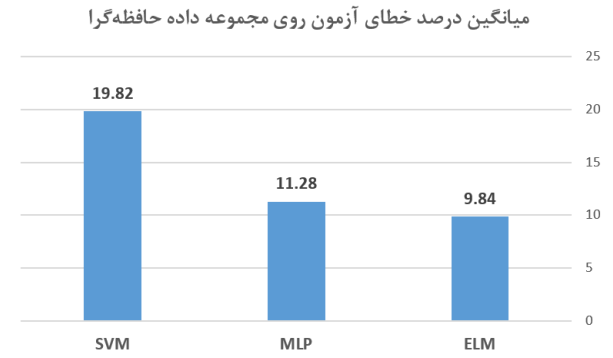
جدول (10): دقت روش‌های یادگیری روی مجموعه داده حافظه‌گرا.

روش	معیار ارزیابی	درصد خطا	
		حداکثر	میانگین
ELM	خطای اعتبارسنجی 10-دسته‌ای	10/32	3/64
	خطای آزمون	9/84	3/83
MLP	خطای اعتبارسنجی 10-دسته‌ای	11/76	0/006
	خطای آزمون	11/28	0/089
SVMR	خطای اعتبارسنجی 10-دسته‌ای	21/4	0/36
	خطای آزمون	19/82	0/16

در شکل‌های (4) و (5) حداکثر و میانگین خطای روش‌ها به صورت مقایسه‌ای روی نمودار ستونی نشان داده شده‌اند. در این آزمایشات بهترین نتایج هر روش گزارش شده است. علاوه بر این سه روش، ما روش‌های جنگل تصادفی و رگرسیون لجستیکی را نیز امتحان کردیم که دارای خطای زیادی بودند و گزارش نشده‌اند. برای ارزیابی مدت زمان فرآیندهای آموزش و آزمون، در جدول (11) مدت زمان اجرای این دو مرحله در هر سه روش روی مجموعه داده محاسبه‌گرا گزارش شده‌اند و نشان می‌دهد روش ELM سریع‌ترین روش است.



شکل (4): مقایسه‌ی دقت روش‌ها روی مجموعه داده محاسبه‌گرا



شکل (5): مقایسه‌ی دقت روش‌ها روی مجموعه داده حافظه‌گرا

جدول (11): زمان‌های آموزش و آزمون روی مجموعه داده محاسبه‌گرا.

روش	زمان آموزش (ثانیه)	زمان آزمون (ثانیه)
ELM	0/001118	0/000399
MLP	2/240111	0/007542
SVM	0/015613	0/000511

[1] Activation Function

3.5. تاثیر جداسازی مجموعه‌های داده

در آزمایش بعدی مجموعه‌های داده محاسبه‌گرا و حافظه‌گرا را با هم ادغام کردیم و سه روش یادگیری ماشین را روی این مجموعه داده اجرا کردیم. در واقع دو مجموعه داده آموزش را باهم و دو مجموعه داده آزمون را نیز با هم ادغام کردیم تا نتایج به خوبی میزان تاثیر تفکیک مجموعه‌های داده محاسبه‌گرا و حافظه‌گرا را نشان دهند. نتایج مربوط به دقت مدل‌های یادگیری روی این مجموعه داده در جدول (12) گزارش شده‌اند.

جدول (12): دقت روش‌های یادگیری روی مجموعه داده ادغام شده.

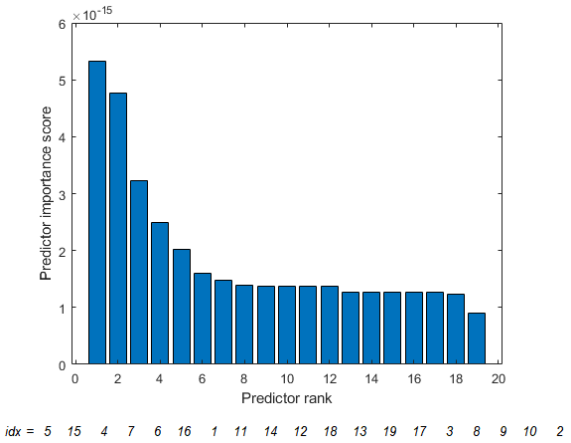
	حداکثر خطا	میانگین خطا
درصد خطای آزمون روش ELM	13/92	2/7
درصد خطای آزمون روش MLP	24/66	0/02
درصد خطای آزمون روش SVMR	35/62	0/31

4.5. تاثیرگذاری هریک از ویژگی‌های ورودی

برای ارزیابی میزان تاثیر هر یک از ویژگی‌های ورودی روی زمان اجرای کرنل کودا، روال انتخاب ویژگی روی مجموعه داده‌ی نرمال‌شده (قبل از تفکیک به مجموعه‌های داده آموزش و آزمون) فراخوانی شده است. نتایج در شکل‌های (6) و (7) و جدول (13) ارائه شده‌اند. خروجی روال انتخاب ویژگی به صورت یک نمودار ارائه شده است که در آن ویژگی‌ها به ترتیب نزولی (برحسب میزان تاثیر) مرتب شده‌اند. در زیر هر نمودار اندیس ویژگی‌ها نیز به ترتیب نشان داده شده‌اند. شکل (6) مربوط به مجموعه داده محاسبه‌گرا و شکل (7) مربوط به مجموعه داده حافظه‌گرا است. در جدول (13) میزان حساسیت هر ویژگی گزارش شده است. ویژگی *cpx* به دلیل داشتن مقادیر ثابت از مجموعه داده حافظه‌گرا حذف شده است.

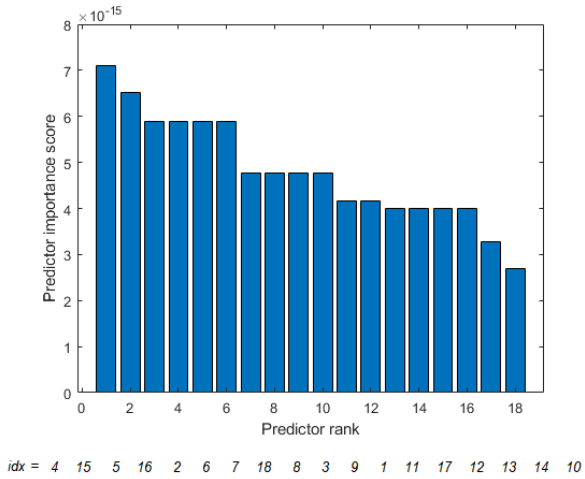
6. بحث

نتایج گزارش شده در بخش قبلی نشان می‌دهند که روش ELM بهترین تخمین را از زمان اجرای کرنل کودا ارائه کرده است. این روش زمان اجرای کرنل‌های محاسبه‌گرا را با خطای 3/42 درصد و زمان اجرای کرنل‌های حافظه‌گرا را با خطای 9/84 درصد پیش‌بینی می‌کند که در مقایسه با دو روش دیگر دقت خوبی ارائه کرده است. در هر سه روش پیاده‌سازی شده دقت کرنل‌های محاسبه‌گرا بهتر از کرنل‌های حافظه‌گرا بوده است. دلیل این مساله این است که اساسا عمومی‌سازی کارآیی کرنل‌های حافظه‌گرا به نسبت محاسبه‌گرا بیشتر مستعد خطاست. در کرنل‌های حافظه‌گرا تراکنش‌های حافظه‌ی بیشتری وجود دارد. از طرفی پیش‌بینی ایستای زمان سرویس‌دهی به تراکنش‌های حافظه به دلیل احتمال موفقیت یا خطای حافظه کش به طور دقیق قابل انجام نیست. بدان معنی که در برخی موارد ممکن است یک تراکنش حافظه در یک اجرای کرنل با خطای کش مواجه شود و در اجرای دیگر با موفقیت کش مواجه شود. ضمن اینکه به‌روزرسانی‌های تناوبی حافظه نیز ممکن است باعث افزایش تاخیر تراکنش‌های حافظه شود. بنابراین پراکندگی زمان اجراهای ثبت شده در کرنل‌های حافظه‌گرا ممکن است عمومی‌سازی کارآیی را با خطا مواجه کند. به همین دلیل است که در تمام روش‌های سه‌گانه‌ای که اجرا کردیم، دقت پیش‌بینی روی مجموعه داده محاسبه‌گرا بهتر از حافظه‌گرا به دست آمده است. به لحاظ مدت زمان آموزش و آزمون نیز روش ELM در مقایسه با روش‌های MLP و SVMR عملکرد بهتری ارائه کرده است. با توجه به اینکه روش ELM بر اساس شبکه‌های عصبی پیش‌رو[1] ایجاد شده است و تنها یک لایه مخفی دارد، سرعت بسیار خوبی در روال‌های آموزش و آزمون ارائه کرده است.



شکل (6): خروجی روال انتخاب ویژگی روی مجموعه داده محاسبه‌گرا.

[1] Feed-forwarding Neural Network



شکل (7): خروجی روال انتخاب ویژگی روی مجموعه داده حافظه‌گرا.

جدول (13): مقادیر خروجی انتخاب ویژگی برای هر ویژگی

ویژگی	محاسبه‌گرا (1.0e-14*)	حافظه‌گرا (1.0e-14*)
n	0/1476	0/4162
s	0/0896	0/5885
α	0/127	0/4771
n_b	0/3234	0/7105
n_t	0/5329	0/5885
w	0/2026	0/5885
cpx	0/2486	--
l	0/127	0/4776
l_c	0/127	0/4776
l_{mr}	0/1233	0/4162
l_{mw}	0/1394	0/27
l_{shr}	0/1376	0/3999
l_{shw}	0/1376	0/3999
l_s	0/1376	0/3999
n_{dpu}	0/4758	0/3273
n_{sfu}	0/1604	0/6519
dpc	0/127	0/5885
$ShMem$	0/1376	0/3999
n_{reg}	0/127	0/4776

در این تحقیق چندین معیار ایستا برای کرنل‌های کودا ارائه شده‌اند که مهمترین نوآوری این تحقیق محسوب می‌شود. پارامتر α ویژگی چندان موثری برای پیش‌بینی زمان اجرای کرنل نیست. چراکه این ویژگی بر اساس شمارش دستورالعمل‌های محاسباتی و حافظه در کدهای کودا محاسبه می‌شود و دستورالعمل‌های محاسباتی همگی یکسان در نظر گرفته شده‌اند. در حالی که دستورالعمل‌های محاسباتی که روی واحدهای تابعی نظیر واحد تابعی خاص و واحد دقت مضاعف اجرا می‌شوند، زمان‌بر هستند و در برخی موارد تاخیر آنها از تاخیر دستورالعمل‌های حافظه نیز بیشتر است. در عوض ما ویژگی‌های l، l_c، l_{mr}، l_{mw}، l_{shr}، l_{shr}، l_{DPU} و l_{SFU} را معرفی کردیم که بر اساس کدهای PTX به دست می‌آیند و هر یک تخمینی از تعداد اجرای دستورالعمل‌ها هستند. محاسبه این ویژگی‌ها با شمارش تعداد دستورالعمل‌ها انجام نشده و بر اساس پیگیری روند اجرای کرنل انجام می‌شود. به همین دلیل است که ما کرنل‌هایی را انتخاب کردیم که تعداد تکرار حلقه‌ها به مقادیر ورودی بستگی نداشته باشند و به صورت ایستا قابل تعیین باشند. تعداد اجرای دستورالعمل‌های PTX تخمین بهتری ارائه می‌کند. چراکه با ضرب تعداد دستورالعمل در تاخیر آن مدت زمان اجرای آنها به دست می‌آید و بنابراین عمومی‌سازی می‌تواند بهتر انجام شده و روش‌های یادگیری ماشین می‌توانند عملکرد بهتری ارائه کنند. درواقع ما در اینجا برای کمک به روش یادگیری ماشین، از ویژگی‌هایی استفاده کردیم که در مدل‌های تحلیلی می‌توانند استفاده شوند. اما سوال مهم این است که چرا به همین روش زمان اجرای کرنل محاسبه نشده است؟ در پاسخ باید ذکر کنیم که روند اجرای کرنل به شدت به زمانبندی کرنل در واحدهای گیگاترد و زمانبند وارپ بستگی دارد. نخ‌ها به صورت بلوک‌ها دسته‌بندی می‌شوند و در درون یک بلوک هم به وارپ‌ها که دسته‌های 32 تایی از نخ‌ها هستند دسته‌بندی می‌شوند. زمانبندی وارپ‌ها در یک بلوک به صورت نوبتی گردشی است که وابستگی زیادی به تراکنش‌های حافظه دارد. زمانبندی بلوک‌ها روی یک چندپردازنده جریانی و اجرای همروند چندین بلوک روی یک چندپردازنده جریانی نیز قابل

ردیابی نیست و اطلاعات چندانی نیز در این خصوص ارائه نشده است. ما آزمایشاتی انجام دادیم که نشان می‌دهند ترتیب اجرای بلوک‌ها و وارپ‌ها بسیار متنوع هستند و بنابراین تخمین دقیق ترتیب آنها دشوار است.

در این تحقیق مجموعه‌های داده جداگانه‌ای برای کرنل‌های محاسبه‌گرا و حافظه‌گرا ایجاد شده است. در جدول (12) نشان دادیم که درصورتی که دو مجموعه داده محاسبه‌گرا و حافظه‌گرا با هم ادغام شوند، خطای پیش‌بینی تا 13/92 درصد افزایش می‌یابد. البته این خطا بهترین مقداری است که به دست آمده است و در روش‌های ELM و MLP خطاهای بیشتری نیز به دست آمده است. این نتایج نشان می‌دهند که جداسازی دو نوع کرنل به عمومی‌سازی بهتر کارآیی کرنل کمک کرده است.

در جدول (2) پژوهش‌های قبلی که برای پیش‌بینی زمان اجرای کرنل کودا از روش‌های یادگیری ماشین استفاده کرده‌اند، گزارش شده‌اند و میزان دقت هر یک نیز قید شده است. تنها مرجع [32] دقت بهتری نسبت به مدل پیشنهادی ما در این تحقیق ارائه کرده است. اما نکته بسیار مهمی که باید به آن توجه شود این است که در مرجع [32] برای ساختن مجموعه داده همزمان از ویژگی‌های ایستا و پویای کرنل استفاده شده است. نویسندگان در این پژوهش معیارهایی مانند اشغال[1] را نیز به عنوان ویژگی‌های ورودی در مجموعه داده لحاظ کردند. این عمل با هدف ایجاد مدل کارآیی در تناقض است. چراکه معیارهای پویا در اثر اجرای کرنل و انجام پروفایلینگ به دست می‌آیند. مدل کارآیی باید بتواند بدون اجرای کرنل زمان اجرای آن را پیش‌بینی کند. بنابراین مدل پیشنهادی این پژوهش بهترین عملکرد را به لحاظ دقت ارائه کرده است.

در جدول (13) نتایج روش انتخاب ویژگی گزارش شده است. در مجموعه داده محاسبه‌گرا ویژگی n_t و در مجموعه داده حافظه‌گرا ویژگی n_b بالاترین میزان تاثیر را روی زمان اجرا دارند. این نتیجه رابطه مستقیمی با زمانبندی در پردازنده گرافیکی دارد. درصورتی که تمام وارپ‌های یک بلوک در انتظار عملیات‌های حافظه باشند، این بلوک مسدود شده و بلوک دیگری جایگزین آن می‌شود. از طرفی باتوجه به اینکه زمانبندی وارپ‌ها در درون یک بلوک به صورت نوبتی گردشی هستند، بنابراین تراکنش‌های حافظه این وارپ‌ها تقریبا همزمان رخ می‌دهند. به همین دلیل در کرنل‌های حافظه‌گرا که تراکنش‌های حافظه بیشتری دارند، تنظیم تعداد بلوک‌ها اهمیت بیشتری نسبت به تعداد نخ‌ها دارد و افزایش تعداد بلوک‌ها می‌تواند کارآیی را افزایش دهد. اما در مورد کرنل‌های محاسبه‌گرا هرچه تعداد نخ‌های یک بلوک بیشتر باشد، بنابراین این نخ‌ها سریع‌تر اجرا شده و از تعویض متن بلوک‌ها پرهیز می‌شود. به همین دلیل در این کرنل‌ها اندازه بلوک اهمیت بیشتری دارد. نتایج ارائه شده در جدول (13) این نکات را تایید می‌کنند.

مزیت مدل پیشنهادی در مقایسه با مدل‌های تحلیلی این است که استفاده‌کنندگان از پرداختن به جزئیات سطح پایین عملکرد سخت‌افزاری و نرم‌افزاری دستگاه بی‌نیاز می‌شوند. جزئیات عملکردی پردازنده‌های گرافیکی به طور کامل توسط تولیدکنندگان ارائه نشده است. به همین دلیل ساخت یک مدل تحلیلی برای این پردازنده‌ها بسیار سخت‌تر از ساخت یک مدل مبتنی بر یادگیری ماشین است و این یک مزیت دیگر مدل پیشنهادی در این تحقیق است. نقطه ضعف مدل پیشنهادی این است که برای هر دستگاه باید به طور جداگانه ساخته شود و قابلیت حمل پایینی دارد. در عوض، مدل‌های تحلیلی فقط یکبار ایجاد می‌شوند و برای دستگاه‌های مختلف قابل بکارگیری هستند. البته به دلیل تنوع بسیار زیاد در نسل‌های مختلف پردازنده‌های گرافیکی مدل‌های تحلیلی ساخته شده روی برخی نسل‌ها ممکن است در برخی دیگر کارآمد نباشد. به عنوان مثال از معماری ولتا به بعد مفهوم زمانبندی نخ‌های مستقل[2] توسعه-یافته است و این تغییر باعث می‌شود که بسیاری از مدل‌های تحلیلی که بر اساس نسل‌های قبلی ایجاد شده‌اند، روی نسل‌های ولتا به بعد دارای خطای زیادی باشند. مدل‌هایی که براساس روش‌های یادگیری ماشین ساخته می‌شوند از این نقیصه مبرا هستند.

[1] Occupancy

[2] Independent Thread Scheduling

7. نتیجه‌گیری و کارهای آینده

در این تحقیق یک مدل کارآیی مبتنی بر یادگیری ماشین برای پیش‌بینی زمان اجرای کرنل‌های کودا ارائه شده است. در طرح پیشنهادی دو مدل جداگانه برای کرنل‌های محاسبه‌گرا و حافظه-گرا ساخته شده است. ابتدا معیارهای مهمی برای کرنل معرفی کردیم که با استفاده از تحلیل ایستای کدهای کودا و PTX مقادیر آنها تعیین می‌شود. برای تعیین این مقادیر نیازی به اجرای برنامه‌ها نیست. برای ساختن مجموعه داده از چهار کرنل محاسبه‌گرا و چهار کرنل حافظه‌گرا استفاده کردیم. با اجرای این کرنل‌ها و انجام عملیات پروفایلینگ، زمان اجرای کرنل‌ها را اندازه‌گیری کرده و به عنوان برچسب خروجی در مجموعه داده ثبت کردیم. هر رکورد از مجموعه داده حاصل اجرای یک کرنل کودا است. ما مجموعه‌های داده جداگانه‌ای برای کرنل‌های محاسبه‌گرا و حافظه‌گرا ایجاد کردیم. سپس سه روش یادگیری ماشین ELM، MLP و SVMR را روی این مجموعه‌های داده اجرا و نتایج آزمایشات را گزارش کرده‌ایم. نتایج نشان دادند که روش ELM زمان اجرای کرنل‌های محاسبه‌گرا را با حداکثر خطای 3/42 و زمان اجرای کرنل‌های حافظه‌گرا را با حداکثر خطای 9/84 درصد پیش‌بینی می‌کند. برای اعتبارسنجی فرآیندهای آموزش و آزمون ما اعتبارسنجی متقاطع 10-دسته‌ای را روی هر سه روش اجرا و نتایج را گزارش کردیم. همچنین برای اینکه میزان تاثیر هر یک از ویژگی‌های ورودی در مجموعه داده را روی برچسب خروجی تعیین کنیم، روش انتخاب ویژگی حداقل افزونگی حداکثر ارتباط را روی هر دو مجموعه داده اجرا کردیم و مقادیر حساسیت را برای هر ویژگی گزارش کرده‌ایم. به عنوان کارهای آینده پیشنهاد می‌کنیم که سایر روش‌های یادگیری ماشین مثل یادگیری عمیق را روی این مجموعه‌های داده اجرا کنیم. همچنین پیشنهاد می‌کنیم که مجموعه داده‌ای از معیارهای پویا ایجاد کرده و با استفاده از روش‌های انتخاب ویژگی میزان تاثیر این ویژگی‌ها را روی زمان اجرای کرنل‌های کودا تعیین کنیم.

تعارض منافع: نویسندگان اعلام می‌کنند که هیچ تعارض منافعی ندارند.

مراجع

[1] NVIDIA, “CUDA C programming guide, version 10.1,” NVIDIA Corp., Santa Clara, CA, USA, 2019.

[2] S. Liu et al., “Fastgr: Global routing on CPU-GPU with heterogeneous task graph scheduler,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 42, no. 7, pp. 2317-2330, 2022, doi: 10.1109/TCAD.2022.3217668.

[3] F. Zhang et al., “Performance evaluation and analysis of sparse matrix and graph kernels on heterogeneous processors,” CCF Trans. High Perform. Comput., vol. 1, pp. 131-143, 2019, doi: 10.1007/s42514-019-00008-6.

[4] R. Schoonhoven, B. van Werkhoven, and K.J. Batenburg, “Benchmarking optimization algorithms for auto-tuning GPU kernels,” IEEE Trans. Evol. Comput., vol. 27, no. 3, pp. 550-564, 2023, doi: 10.1109/TEVC.2022.3210654.

[5] D. Mustafa, “A survey of performance tuning techniques and tools for parallel applications,” IEEE Access, vol. 10, pp. 15036-15055, 2022, doi: 10.1109/ACCESS.2022.3147846.

[6] J. Lemeire, J.G. Cornelis, and E. Konstantinidis, “Analysis of the analytical performance models for GPUs and extracting the underlying pipeline model,” J. Parallel Distrib. Comput., vol. 173, pp.

32-47, 2023, doi: 10.1016/j.jpdc.2022.11.002.

[7] M. Lattuada, E. Gianniti, D. Ardagna, and L. Zhang, "Performance prediction of deep learning applications training in GPU as a service systems," Cluster Comput., vol. 25, no. 3, pp. 1279-1302, 2022, doi: 10.1007/s10586-021-03428-8.

[8] N. Weber and M. Goesele, "MATOG: Array layout auto-tuning for CUDA," ACM Trans. Archit. Code Optim., vol. 14, no. 3, pp. 1-26, 2017, doi: 10.1145/3106341.

[9] M.A. Al-Mouhamed, A.H. Khan, and N. Mohammad, "A review of CUDA optimization techniques and tools for structured grid computing," Computing, vol. 102, no. 4, pp. 977-1003, 2020, doi: 10.1007/s00607-019-00744-1.

[10] S. Madougou et al., "The landscape of GPGPU performance modeling tools," Parallel Comput., vol. 56, pp. 18-33, 2016, doi: 10.1016/j.parco.2016.04.002.

[11] A. Reuther et al., "Survey and benchmarking of machine learning accelerators," in Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC), 2019, doi: 10.1109/HPEC.2019.8916327.

[12] A. Riahi, A. Savadi, and M. Naghibzadeh, "Comparison of analytical and ML-based models for predicting CPU-GPU data transfer time," Computing, vol. 102, no. 9, pp. 2099-2116, 2020, doi: 10.1007/s00607-019-00780-x.

[13] J. Peddie, "What is a GPU?" in The History of the GPU-Steps to Invention. Cham, Switzerland: Springer, 2023, pp. 333-345, doi: 10.1007/978-3-031-10968-3_7.

[14] J. Wang, W. Pang, C. Weng, and A. Zhou, "D-Cubicle: Boosting data transfer dynamically for large-scale analytical queries in single-GPU systems," Front. Comput. Sci., vol. 17, no. 4, 2023, doi: 10.1007/s11704-022-2160-z.

[15] NVIDIA, "CUDA C++ programming guide, version 11.7," NVIDIA Corp., Santa Clara, CA, USA, 2022.

[16] NVIDIA, "CUDA C++ programming guide, version 12.1," NVIDIA Corp., Santa Clara, CA, USA, 2023.

[17] NVIDIA, "Fermi whitepaper," NVIDIA Corp., Santa Clara, CA, USA, 2009.

[18] NVIDIA, "GTX 680 whitepaper," NVIDIA Corp., Santa Clara, CA, USA, 2012.

[19] NVIDIA, "980: Featuring Maxwell, the most advanced GPU ever made," NVIDIA Corp., Santa Clara, CA, USA, 2014.

[20] NVIDIA, "Whitepaper NVIDIA GeForce GTX 1080 gaming perfected," NVIDIA Corp., Santa Clara, CA, USA, 2016.

[21] NVIDIA, "NVIDIA Turing GPU architecture: Graphics reinvented," NVIDIA Corp., Santa Clara, CA, USA, 2018.

[22] NVIDIA, "Tesla V100 GPU architecture: The world's most advanced datacenter GPU," NVIDIA Corp., Santa Clara, CA, USA, 2017.

[23] P.N. Glaskowsky, "NVIDIA's Fermi: The first complete GPU computing architecture," White paper 18, 2009.

[24] NVIDIA, "CUDA C++ programming guide, version 11.7," NVIDIA Corp., Santa Clara, CA, USA, 2022.

[25] M.P. Akbarpour, K. Khamforoosh, and V. Maihami, "An approach to Improve Particle Swarm Optimization Algorithm Using CUDA," Soft Comput. J., vol. 8, no. 2, pp. 2-21, 2020, doi: 10.22052/8.2.2 [In Persian].

[26] S. Ghike et al., "Directive-based compilers for GPUs," in Proc. 27th Int. Worksh. Lang. Compilers Parallel Comput. (LCPC), 2015, doi: 10.1007/978-3-319-17473-0_2.

[27] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 1, pp. 272-281, 2015, doi: 10.1109/TPDS.2014.2308216.

[28] K. O'Neal et al., "HALWPE: Hardware-assisted light weight performance estimation for GPUs," in

Proc. 54th Annu. Des. Autom. Conf. (DAC), 2017, doi: 10.1145/3061639.306225.

[29] G. Wu, J.L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in Proc. 21st Int. Symp. High Perform. Comput. Archit. (HPCA), 2015, pp. 564-576, doi: 10.1109/HPCA.2015.7056063.

[30] J.-C. Huang, J.H. Lee, H. Kim, and H.-H.S. Lee, "GPUMech: GPU performance modeling technique based on interval analysis," in Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), 2014, pp. 268-279, doi: 10.1109/MICRO.2014.59.

[31] A. Resios and V. Holdermans, "GPU performance prediction using parametrized models," Utrecht Univ., Utrecht, Netherlands, 2011.

[32] M. Amaris, R.Y. de Camargo, M. Dyab, A. Goldman, and D. Trystram, "A comparison of GPU execution time prediction using machine learning and analytical modeling," in Proc. 15th Int. Symp. Netw. Comput. Appl. (NCA), 2016, pp. 326-333, doi: 10.1109/NCA.2016.7778637.

[33] M. Amaris, D. Cordeiro, A. Goldman, R.Y. de Camargo, "A simple BSP-based model to predict execution time in GPU applications," in Proc. 22nd Int. Conf. High Perform. Comput. (HiPC), 2015, pp.285-294, doi: 10.1109/HiPC.2015.34.

[34] J. Yun et al., "A novel performance prediction model for mobile GPUs," IEEE Access, vol. 6, pp. 16235-16245, 2018, doi: 10.1109/ACCESS.2018.2816040.

[35] T. Remmelg et al., "High-level hardware feature extraction for GPU performance prediction of stencils," in Proc. 13th Annu. Worksh. Gen. Purpose Process. Graphics Process. Unit (GPGPU), 2020, doi: 10.1145/3366428.3380769.

[36] H. Bouzidi et al., "Performance prediction for convolutional neural networks on edge GPUs," in Proc. 18th ACM Int. Conf. Comput. Frontiers (CF), 2021, doi: 10.1145/3457388.3458666.

[37] K. O'neal et al., "GPU performance estimation using software rasterization and machine learning," ACM Trans. Embedded Comput. Syst., vol. 16, no. 5, pp. 1-21, 2017, doi: 10.1145/3126557.

[38] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Sparse matrix format selection with multiclass SVM for SpMV on GPU," in Proc. 45th Int. Conf. Parallel Process. (ICPP), 2016, pp. 496-505, doi: 10.1109/ICPP.2016.64.

[39] T.T. Dao, J. Kim, S. Seo, B. Egger, and J. Lee, "A performance model for GPUs with caches," IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 7, pp. 1800-1813, 2015, doi: 10.1109/TPDS.2014.2333526.

[40] C. Lehnert et al., "Performance prediction and ranking of SpMV kernels on GPU architectures," in Euro-Par 2016: Parallel Processing, Cham, Switzerland: Springer, 2016, doi: 10.1007/978-3-319-43659-3_7.

[41] E. Gianniti, L. Zhang, and D. Ardagna, "Performance prediction of GPU-based deep learning applications," in Proc. 30th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD), 2018, doi: 10.1109/CAHPC.2018.8645908.

[42] NVIDIA, "CUDA C programming guide, version 7.0," NVIDIA Corp., Santa Clara, CA, USA, 2015.

[43] M.A. Hall, Practical Machine Learning Tools and Techniques. Burlington, MA, USA: Morgan Kaufmann, 2011.

[44] P. Carvalho, "Using machine learning techniques to analyze the performance of concurrent kernel execution on GPUs," Future Gener. Comput. Syst., vol. 113, pp. 528-540, 2020, doi: 10.1016/j.future.2020.07.038.

[45] E. Saberi, E. Radmand, J. Pirgazi, and A. Kermani, "Buying and selling strategy in the Iranian stock market using machine learning models, with feature selection using the Cuckoo Search algorithm," Soft Comput. J., vol. 12, no. 2,

pp. 130-145, 2024, doi: 10.22052/scj.2023.252793.1144 [In Persian].

[46] H. Veisi, H.R. Ghaedsharaf, and M. Ebrahimi, "Improving the Performance of Machine Learning Algorithms for Heart Disease Diagnosis by Optimizing Data and Features," Soft Comput. J., vol. 8, no. 1, pp. 70-85, 2019, doi: 10.22052/8.1.70 [In Persian].

[47] S.K. Shekofteh, H. Noori, M. Naghibzadeh, H.S. Yazdi, and H. Froning, "Metric selection for GPU kernel classification," ACM Trans. Archit. Code Optim., vol. 15, no. 4, pp. 1-27, 2019, doi: 10.1145/3295690.