

## پیش‌بینی کارایی برنامه‌های منظم کودا از طریق روش‌های یادگیری ماشین

علی ریاحی<sup>1</sup>، استادیار، عبدالرضا سوادی<sup>2\*</sup>، استادیار، محمود نقیب‌زاده<sup>3</sup>، استاد

<sup>1</sup>گروه مهندسی کامپیوتر، دانشکده مهندسی برق و کامپیوتر، دانشگاه علم و فناوری مازندران، بهشهر، ایران

<sup>2</sup> و <sup>3</sup>گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه فردوسی مشهد، مشهد، ایران

<sup>1</sup>[riahi@mazust.ac.ir](mailto:riahi@mazust.ac.ir), <sup>2</sup>[savadi@um.ac.ir](mailto:savadi@um.ac.ir), <sup>3</sup>[naghibzadeh@um.ac.ir](mailto:naghibzadeh@um.ac.ir)

چکیده: پیش‌بینی کارایی 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 پردازنده‌ی گرافیکی مقرون‌به‌صرفه نباشند. پیش‌بینی کارایی به برنامه‌نویس کمک می‌کند تا تنظیم برنامه‌ها را آگاهانه‌تر انجام دهند و به‌جای آزمون و خطا، نقطه‌ی بهینه‌ی تنظیم را دقیق‌تر پیدا کند. وجود یک مدل که بتواند کارایی را پیش‌بینی کند، می‌تواند به برنامه‌نویسان در تنظیم دستی و به توسعه‌دهندگان کامپایلرهای مبدأ به مبدأ در تنظیم خودکار برنامه‌های کودا کمک کند. در این مقاله، ما یک مدل کارایی 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 شبکه‌ی 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 حافظه-گرا 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. واژه‌های کلیدی: پردازنده‌ی گرافیکی، کودا، مدل کارایی، پیش‌بینی زمان اجرا، یادگیری ماشین، انتخاب ویژگی.

## 1- مقدمه

وجود دارد. به همین دلیل پیش‌بینی کارایی کرنل دشوار بوده و در برخی موارد، اجرای برنامه‌ها روی پردازنده‌ی گرافیکی مقرون‌به‌صرفه نیست [2,3]. از طرفی باتوجه به سطح پایین بودن زبان برنامه‌نویسی کودا، لازم است تا تنظیم<sup>8</sup> برنامه‌ها براساس ویژگی‌های سخت‌افزار و جزئیات سطوح پایین معماری انجام شود [4]. منظور از تنظیم برنامه، بکارگیری هر تکنیکی است که باعث بهبود کارایی یک برنامه شود. به دلیل دانش کم از معماری، استفاده از یک تکنیک تنظیم ممکن است باعث بهبود یا افت کارایی شود. در نتیجه 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<sup>9</sup>، کامپایلرهای مبدأ به مبدأ<sup>10</sup> سعی می‌کنند که علاوه بر ترجمه‌ی برنامه‌ها به زبان‌های سطح پائین تر، تنظیم خودکار<sup>11</sup> نیز انجام دهند [5] و این عمل ممکن است باعث افت کارایی شده و عملکرد کرنل کاهش یابد. در نتیجه کامپایلرهای مبدأ به مبدأ در استفاده از بسیاری از تکنیک‌های تنظیم ناتوان هستند.

یک راه‌کار برای مرتفع کردن چالش‌های مطرح شده، استفاده از یک مدل کارایی است [6]. مدل کارایی به شناسایی جنبه‌های ناشناخته‌ی دستگاه و گلوگاه‌های کارایی آن کمک می‌کند. همچنین با استفاده از مدل می‌توان کارایی یک کرنل را پیش‌بینی کرد [7]. باید توجه شود که در این مقاله منظور از کارایی، زمان اجرای کرنل است. مدل کارایی 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 پردازنده‌ی گرافیکی، برآوردی خواهیم داشت که آیا اجرای برنامه روی این پردازنده باعث بهبود کارایی شده و تسریع مناسبی دارد یا خیر؟ از طرفی مدل کارایی می‌تواند به تنظیم دستی و خودکار برنامه‌ها

با افزایش قابلیت‌های پردازشی پردازنده‌ی گرافیکی، از این پردازنده در کاربردهای بسیار وسیعی از پردازش موازی از جمله محاسبات علمی، پردازش داده‌های بزرگ، داده‌کاوی، مسیریاب-های اینترنت و بیوانفورماتیک استفاده شده است و عنوان پردازنده‌ی همه‌منظوره<sup>1</sup> بر آن قرار داده شده است. این پردازنده-ها به عنوان یک شتاب‌دهنده<sup>2</sup> و یا به‌عنوان یک پردازنده‌ی همکار<sup>3</sup> توسط پردازنده‌ی مرکزی مورد استفاده قرار می‌گیرند. به همین دلیل معمولاً به پردازنده‌ی مرکزی اصطلاح میزبان<sup>4</sup> و به پردازنده‌ی گرافیکی اصطلاح دستگاه<sup>5</sup> اطلاق می‌شود. میزبان تابعی که کرنل<sup>6</sup> نامیده می‌شود را برای دستگاه ارسال کرده و پس از خاتمه‌ی اجرای کرنل، نتایج جهت استفاده در ادامه‌ی برنامه به سمت میزبان برگردانده می‌شوند [1].

مدل برنامه‌نویسی کودا توسط شرکت انویدیا برای طراحی برنامه‌های موازی و اجرای آن‌ها روی پردازنده‌ی 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<sup>7</sup> ترجمه می‌شوند. این کدها یک روایت شبه‌اسمبلی 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 نحوه‌ی عملکرد آن‌ها توسط توسعه‌دهندگان منتشر شده است و جنبه‌های ناشناخته زیادی برای برنامه‌نویسان

<sup>1</sup> General Purpose GPU (GPGPU)  
<sup>2</sup> Accelerator  
<sup>3</sup> Co-processor  
<sup>4</sup> Host  
<sup>5</sup> Device  
<sup>6</sup> Kernel  
<sup>7</sup> Parallel Thread Execution

<sup>8</sup> Tuning  
<sup>9</sup> Directive-based Language  
<sup>10</sup> Source-to-source Compiler  
<sup>11</sup> Auto-tuning

این مجموعه داده براساس 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 (برجسب خروجی) مشخص می‌شود. باتوجه به اینکه برنامه‌های کودا منحصر روی پردازنده‌های گرافیکی تولید شده توسط شرکت انویدیا اجرا می‌شوند، بنابراین موضوع این مقاله منحصر این دستگاه‌ها می‌باشند. مهم‌ترین نوآوری این تحقیق این است که معیارهای ایستای مهمی برای کرنل‌های کودا معرفی شده است که برخی از آن‌ها براساس 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 ادامه‌ی این مقاله، ابتدا در بخش 2 پیش‌زمینه‌های موردنیاز ارائه می‌شوند. سپس در بخش 3 مهم‌ترین کارهای مرتبط با طرح پیشنهادی که قبلاً ارائه شده‌اند، مورد بررسی قرار می‌گیرند. در ادامه در بخش 4 مدل پیشنهادی ارائه شده و در بخش بعدی مورد ارزیابی قرار می‌گیرد. سپس در بخش‌های 6 و 7 بحث و نتیجه‌گیری انجام می‌شود.

## 2- پیش‌زمینه

پردازنده‌های گرافیکی از جهت وضعیت حافظه‌ی مشترک با پردازنده‌ی مرکزی، به دو دسته‌ی یکپارچه<sup>۲</sup> و گسسته<sup>۳</sup> دسته‌بندی می‌شوند [13]. پردازنده‌های یکپارچه دارای فضای حافظه و آدرس‌دهی مشترک با پردازنده‌ی مرکزی می‌باشند، درحالی‌که پردازنده‌های گرافیکی گسسته حافظه‌های مجزا داشته و برای

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 مبدأ به مبدأ نیز می‌توانند تنظیم بهتری را به صورت خودکار روی کرنل ورودی اعمال کنند [9].

ما مدل‌های کارآیی را به دو دسته‌ی 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، باتوجه به اینکه جزئیات دستگاه نامشخص است، ایجاد یک مدل تحلیلی بسیار سخت است و معمولاً با خطای زیادی همراه است. در مدل‌های مبتنی بر یادگیری ماشین ویژگی‌ها و عملکرد دستگاه توسط روش‌های یادگیری ماشین عمومی‌سازی<sup>۱</sup> می‌شوند و عملاً ایجادکننده‌ی 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 مبتنی بر یادگیری ماشین ایجاد شده است که زمان اجرای یک کرنل کودای ورودی را پیش‌بینی می‌کند. باتوجه به تنوع دستگاه‌ها و برنامه‌ها، هیچ مجموعه داده استاندارد برای این مسئله 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.

<sup>2</sup> Integrated  
<sup>3</sup> Discrete

<sup>1</sup> Generalization

میزبان<sup>۲</sup> و تعدادی چندپردازنده‌ی جریانی<sup>۳</sup> است که از طریق یک شبکه‌ی ارتباطی با هم در ارتباطند. 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، چندپردازنده‌های جریانی کاملاً همسان هستند. حافظه‌ی سراسری تنها حافظه‌ای است که می‌توان از سمت حافظه‌ی اصلی میزبان (پردازنده‌ی مرکزی) به آن داده رد و بدل کرد. حافظه‌ی نهان سطح دو مشترک بین چندپردازنده‌های جریانی است. واحد گیگاترد زمانبندی سطح یک است و بلوک‌هایی از نخ‌ها را جهت اجرا بین چندپردازنده‌های جریانی توزیع می‌کند. واسط میزبان وظیفه‌ی برقراری ارتباط و هماهنگی بین میزبان و دستگاه را به عهده دارد.

به لحاظ پردازشی، مهم‌ترین بخش این معماری چندپردازنده‌های جریانی هستند که هسته‌های پردازشی را در خود جای داده‌اند. هر چندپردازنده‌ی جریانی شامل تعدادی هسته بوده که پردازنده‌ی جریانی<sup>۴</sup> نامیده می‌شوند و به صورت یک دستورالعمل چندین داده<sup>۵</sup> بکار گرفته شده و نخ‌ها را به طور موازی اجرا می‌کنند. علاوه بر این، در یک چندپردازنده‌ی جریانی واحدهای محاسبات با دقت مضاعف<sup>۶</sup>، واحد تابعی خاص<sup>۷</sup> برای توابع غیرجبری، واحد تنسورفلو<sup>۸</sup> برای محاسبات مربوط به یادگیری عمیق<sup>۹</sup> و واحدهای بارگذاری و ذخیره‌ی داده‌ها<sup>۱۰</sup> وجود دارند. یکی از مهم‌ترین اجزای درونی یک چندپردازنده‌ی جریانی

اجرای یک برنامه روی آن‌ها باید ابتدا داده‌ها منتقل شوند و پس از اجرای برنامه نیز نتایج به حافظه‌ی اصلی برگردانده شوند. این رفت و برگشت داده‌ها تاثیر مستقیمی روی کارایی 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 [15,16] 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.

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

## 1-2- معماری پردازنده‌ی گرافیکی

معماری پردازنده‌های گرافیکی انویدیا که در شکل (1) نشان داده شده است، شامل یک حافظه‌ی سراسری، یک حافظه‌ی نهان سطح دو، یک زمانبند سطح اول به نام گیگاترد<sup>۱</sup>، یک واسط

<sup>2</sup> Host Interface

<sup>3</sup> Streaming Multiprocessor (SM)

<sup>4</sup> Streaming Processor (SP)

<sup>5</sup> Single Instruction Multiple Data (SIMD)

<sup>6</sup> Double Precision Unit (DPU)

<sup>7</sup> Special Function Unit (SFP)

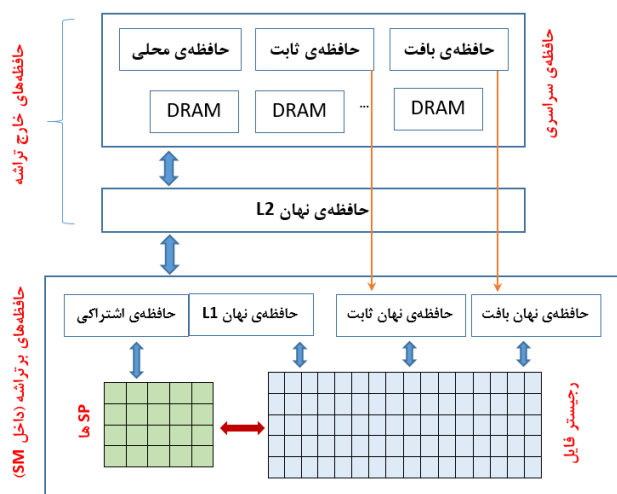
<sup>8</sup> Tensor Processing Unit (TPU)

<sup>9</sup> Deep Learning

<sup>10</sup> Load/Store (LD/ST)

<sup>1</sup> GigaThread

یک و بافت یکپارچه هستند [19]. از معماری پاسکال به بعد حافظه‌ی اشتراکی و حافظه‌ی نهان سطح یک یکپارچه شده و قابل پیکربندی هستند [20-22]. حافظه‌ی 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 حافظه‌ی سراسری که فقط برای خواندن کش می‌شود، حافظه‌ی محلی هم برای خواندن و هم برای نوشتن کش می‌شود [24]. نخ‌هایی که روی یک چندپردازنده‌ی جریانی اجرا می‌شوند از طریق منابع آن می‌توانند با هم همگام شوند [25].

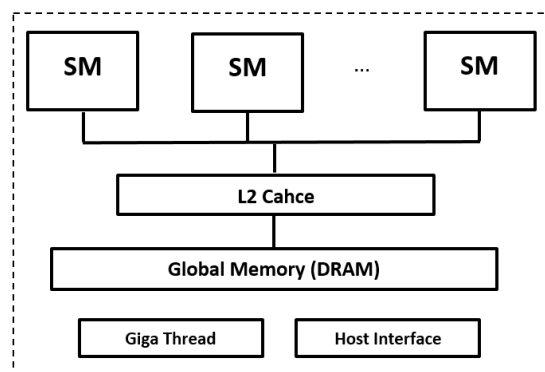


شکل (2): سلسله‌مراتب حافظه‌ی پردازنده‌های گرافیکی انویدیا

## 2-2- مدل برنامه‌نویسی کودا

یک برنامه‌ی 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 برنامه‌ی کودا شامل اجرای یک یا چند کرنل چندنخی است که به صورت داده‌موازی<sup>9</sup> روی پردازنده‌ی گرافیکی اجرا می‌شوند. محاسبات به وسیله‌ی سلسله-مراتبی از نخ‌ها انجام می‌شوند. در واقع کرنل یک برنامه‌ی ترتیبی است که براساس آن تعداد زیادی نخ ایجاد می‌شود. این تعداد

واحد زمانبند وارپ<sup>1</sup> است که زمانبند سطح دوم در پردازنده‌ی گرافیکی است.



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

یکی از مهم‌ترین جنبه‌های معماری پردازنده‌ی گرافیکی، سلسله‌مراتب حافظه‌ی آن است که در شکل (2) نشان داده شده است. درک این سلسله‌مراتب و اندازه‌های واحدهای حافظه در بهینه‌سازی کدها تاثیر مهمی دارند. حافظه‌های محلی<sup>2</sup>، بافت<sup>3</sup> و ثابت<sup>4</sup> با حافظه‌ی سراسری یکپارچه هستند. سطح دسترسی حافظه‌ی محلی در سطح نخ‌ها است و در صورتی توسط یک نخ استفاده می‌شود که ثابت به اندازه‌ی کافی در دسترس نخ نباشد. این حافظه‌ها خارج تراشه<sup>5</sup> بوده و تاخیر دسترسی آن‌ها زیاد است. چندپردازنده‌های جریانی شامل حافظه‌ی اشتراکی، حافظه‌های نهان دستورالعمل، حافظه‌ی نهان سطح یک، حافظه‌ی نهان بافت<sup>6</sup> و حافظه‌ی نهان ثابت<sup>7</sup> و تعداد زیادی ثابت هستند که حافظه‌های بر تراشه<sup>8</sup> محسوب می‌شوند و سرعت دسترسی بالایی دارند. حافظه‌ی اشتراکی در معماری‌های فرمی و کپلر با حافظه‌ی نهان سطح یک یکپارچه است [17,18]. ولی در معماری مکسول حافظه‌ی اشتراکی به عنوان یک واحد مستقل در چندپردازنده‌ی جریانی قرار گرفته است و حافظه‌های نهان سطح

<sup>1</sup> Warp Scheduler  
<sup>2</sup> Local Memory  
<sup>3</sup> Texture Memory  
<sup>4</sup> Constant Memory  
<sup>5</sup> Off-chip  
<sup>6</sup> Texture Cache memory  
<sup>7</sup> Constant Cache Memory  
<sup>8</sup> On-chip

<sup>9</sup> Data-parallel

پردازنده‌ی 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<sup>4</sup> است [26].

### 3- مرور کارهای گذشته

در بخش 1 مدل‌های کارآیی که برای پردازنده‌ی گرافیکی ارائه شده‌اند را به دو دسته‌ی تحلیلی و مبتنی بر 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 رابطه‌ی بین داده‌ها استخراج کنند. این عمل از طریق داده‌های از قبل شناخته شده انجام می‌شود و مرحله‌ی 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 7 تا 9 درصد دست یافتند. وو و همکاران [29] مدلی براساس یادگیری ماشین برای پیش‌بینی توان و کارآیی پردازنده‌ی گرافیکی ارائه کردند. آن‌ها مجموعه‌ای از کرنل‌ها را روی پردازنده‌های گرافیکی مختلف اجرا کرده و داده‌ها را با استفاده از شمارنده‌ی کارآیی<sup>5</sup> جمع‌آوری کردند. آن‌ها کرنل‌ها را براساس ویژگی‌های آن‌ها و با استفاده از الگوریتم خوشه‌بندی K- میانگین<sup>6</sup> 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.

از طریق پیکربندی کرنل تعیین می‌شود. هنگام راه‌اندازی کرنل (از طریق برنامه‌ی میزبان)، همه‌ی نخ‌ها در قالب یک گرید و در قالب چندین بلوک سازماندهی می‌شوند که پیکربندی کرنل را ایجاد می‌کنند. این پیکربندی به صورت  $\langle\langle n_b, n_t \rangle\rangle$  مشخص می‌شود.  $n_b$  اندازه‌ی گرید (تعداد بلوک‌ها) و  $n_t$  اندازه‌ی بلوک (تعداد نخ‌ها در هر بلوک) است. اندازه‌ی 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. معمولاً این اعداد برحسب اندازه‌ی داده‌های ورودی یا اندازه‌ی مسئله تعیین می‌شوند و نقش مهمی در موازی‌سازی دارند. نخ‌های یک بلوک روی یک چندپردازنده‌ی جریان‌ی اجرا می‌شوند و می‌توانند با هم همگام شده و روی داده‌های مشترک کار کنند. مدل برنامه‌نویسی کودا برای موازی‌سازی از نگاهت 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 پردازنده‌ی 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 به‌طور همزمان کدگشایی<sup>1</sup>، صادر<sup>2</sup> و اجرا می‌شوند. هر 32 نخ عضو یک وارپ به‌طور همزمان در حال اجرای یک دستورالعمل مشابه بوده و ثبات شمارنده‌ی برنامه‌ی مشترک دارند. ارتباط بین میزبان و دستگاه از طریق متن کودا<sup>3</sup> میسر می‌شود. با فراخوانی اولین تابع کتابخانه‌ای از سمت میزبان، این متن ایجاد می‌شود [24]. به دلیل متمایز بودن حافظه‌ی اصلی از حافظه‌ی سراسری پردازنده‌ی گرافیکی گسسته، برای راه‌اندازی یک کرنل، ابتدا داده‌ها باید از حافظه‌ی اصلی میزبان به حافظه‌ی سراسری

<sup>4</sup> Single Program Multiple Data (SPMD)

<sup>5</sup> Performance Counter

<sup>6</sup> K-means

<sup>1</sup> Decode

<sup>2</sup> Issue

<sup>3</sup> CUDA Context

جدول (3): مرور مدل‌های کارآیی مبتنی بر یادگیری ماشین

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

آماریس و همکاران [32] مدل تحلیلی که در [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 (3) به‌طور 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 مسئله بکارگرفته نشده‌اند و در بخش بعدی این روش‌ها را امتحان کنیم.

#### 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 آموزش یافته است که برای

در برخی از پژوهش‌ها روش‌های یادگیری ماشین برای محاسبه-ی 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 محاسبه‌ی 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 محاسبه‌ی 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 [12] برای تخمین زمان انتقال داده‌ها بین حافظه‌ی اصلی میزبان و حافظه‌ی 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 محاسبه‌ی پارامتر  $\lambda$  از روش‌های یادگیری ماشین استفاده شده است.

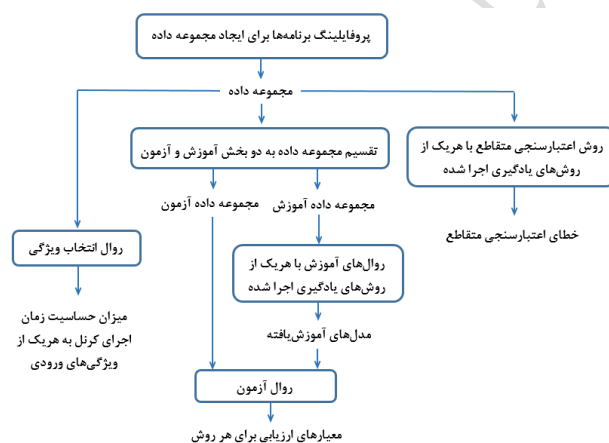
پژوهش فقط به برنامه‌های منظم می‌پردازیم. علاوه بر این، ما برای کرنل‌های حافظه‌گرا و محاسبه‌گرا مجموعه داده‌های مجزایی ایجاد کردیم. نتایج نشان دادند که در این حالت عمومی‌سازی زمان اجرای کرنل توسط روش‌های یادگیری ماشین بهتر انجام شده است.

روال آزمون ارسال می‌شوند. همچنین در این مرحله از اعتبارسنجی متقاطع K-دسته<sup>1</sup> نیز استفاده می‌شود. در روال آزمون مدل‌های آموزش یافته با استفاده از مجموعه داده آزمون ارزیابی می‌شوند و خطای هر مدل گزارش می‌شود. در ادامه هر یک از این مراحل به تفصیل تشریح می‌شوند.

جدول (4): معیارهای استخراج شده به عنوان ویژگی‌های ورودی.

معیار	توضیحات
$n$	اندازه‌ی ورودی
$s$	اندازه‌ی مسئله
$\alpha$	شدت محاسباتی <sup>2</sup>
$n_b$	تعداد بلوک‌های کرنل (اندازه‌ی گرید)
$n_t$	تعداد نخ‌ها در یک بلوک (اندازه‌ی بلوک)
$w$	تعداد وارپ‌ها در یک بلوک
$cp_x$	پیچیدگی محاسباتی کرنل
$l$	تعداد کل دستورالعمل‌های PTX
$l_c$	تعداد دستورالعمل‌های محاسباتی
$l_{mr}$	تعداد دستورالعمل‌های خواندن از حافظه‌ی سراسری
$l_{mw}$	تعداد دستورالعمل‌های نوشتن در حافظه‌ی سراسری
$l_{shr}$	تعداد دستورالعمل‌های خواندن از حافظه‌ی اشتراکی
$l_{shw}$	تعداد دستورالعمل‌های نوشتن در حافظه‌ی اشتراکی
$l_s$	تعداد دستورالعمل‌های همگام‌سازی
$l_{dpu}$	تعداد دستورالعمل‌های اعشاری دقت مضاعف
$l_{sfu}$	تعداد دستورالعمل‌های واحد تابعی خاص
$dpc$	درجه‌ی وابستگی داده‌ای
$ShMem$	تعداد کلمات حافظه‌ی اشتراکی مورد نیاز یک بلوک
$n_{reg}$	تعداد رجیسترهای مورد نیاز یک نخ

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 ترجمه‌ی شبه‌اسمبلی از کد کودا است که برای ایجاد آن نیازی به اجرای برنامه نیست و با استفاده از گزینه‌ی ترجمه‌ی ptx- از طریق مترجم NVCC به دست می‌آید. در جدول (4) این معیارها لیست شده‌اند. بنابراین مجموعه داده شامل 16 ویژگی ورودی و یک برچسب خروجی است. برچسب خروجی زمان اجرای کرنل است که با



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

#### 4-1- ایجاد مجموعه داده

برای ایجاد مجموعه داده از محک‌های دنیای واقعی و مجموعه مثال‌های انویدیا استفاده کردیم که در بخش بعدی معرفی می‌شوند. برای این منظور فقط برنامه‌های منظم بکار گرفته شده‌اند. چراکه زمان اجرای کرنل‌های کودا به شدت به دسترسی‌های حافظه وابسته است. در برنامه‌های منظم الگوی دسترسی به حافظه تقریباً مشخص است و بنابراین احتمال پیگیری موفق داده‌ها در سلسله‌مراتب حافظه بیشتر است. در عوض، در برنامه‌های نامنظم احتمال موفقیت یا خطای حافظه‌ی نهان چندان قابل پیش‌بینی نیست. از طرفی زمان دسترسی به حافظه‌ی سراسری و حافظه‌های نهان تفاوت بسیار محسوسی دارند و بنابراین پراکندگی زمان اجراهای ثبت شده برای این برنامه‌ها بسیار متنوع هستند و این مسئله باعث می‌شود که در مرحله‌ی ایجاد مجموعه داده، ممکن است داده‌های بد تولید شوند. به همین دلیل در این

<sup>1</sup> Arithmetic Intensity

<sup>1</sup> K-fold Cross Validation



پروفایلینگ اجرای کرنل اندازه‌گیری می‌شود. این زمان برحسب میلی‌ثانیه ثبت می‌شود.

هفت ویژگی اول در جدول (4) با استفاده از تحلیل کدهای کودا محاسبه می‌شوند.  $n$  اندازه‌ی ورودی است و فرض بر این است که عددی مشخص است. ویژگی  $cpx$  پیچیدگی محاسباتی (مرتبه‌ی اجرا) الگوریتم کرنل است که معمولاً به صورت مجانبی برحسب اندازه‌ی ورودی تعیین می‌شود. با توجه به مشخص بودن اندازه‌ی ورودی این ویژگی نیز به صورت یک عدد محاسبه می‌شود.  $s$  اندازه‌ی مسئله است و براساس ماهیت مسئله و الگوریتم آن تعیین می‌شود. مثلاً در مورد مسئله‌ی ضرب ماتریس‌ها با ابعاد  $n \times n$  اندازه‌ی مسئله  $n^2$  و پیچیدگی آن  $n^3$  است.  $\alpha$  پارامتر شدت محاسباتی است که در راهنمای برنامه‌نویسی انوی‌دیا [42] معرفی شده است و از تقسیم تعداد دستورالعمل‌هایی که از حافظه‌ی برتراشه (دستورالعمل‌های محاسباتی) استفاده می‌کنند، بر تعداد دستوراتی که از حافظه‌ی خارج تراشه (دستورالعمل‌های حافظه) استفاده می‌کنند، به دست می‌آید. در محاسبه‌ی این پارامتر، دستورالعمل همگام‌سازی درون بلاکی (`__syncthreads`) جزء دستورالعمل‌های برتراشه محسوب می‌شود. چراکه این عملیات با استفاده از حافظه‌ی اشتراکی انجام می‌شود. این پارامتر با تحلیل کدهای کودا و با شمارش تعداد دستورالعمل‌های کرنل استخراج می‌شود. صرف‌نظر از اینکه این دستورات چند بار اجرا می‌شوند. پارامترهای  $n_b$  و  $n_t$  و  $w$  به ترتیب اندازه‌ی گرید، اندازه‌ی بلوک و تعداد وارپ‌ها در یک بلوک هستند و با توجه به پیکربندی کرنل تعیین می‌شوند.

هشت ویژگی بعدی در جدول (4) با استفاده از تحلیل کدهای PTX به دست می‌آیند. معیارهای  $l$ ،  $l_c$ ،  $l_{mr}$ ،  $l_{mw}$ ،  $l_{shr}$  و  $l_{shw}$  تعداد دستوراتی از کدهای PTX هستند که اجرا می‌شوند. این پارامترها با تعداد دستورات متفاوت است. مواردی نظیر حلقه-

های تکرار باعث می‌شوند که برخی از دستورات بیش از یک بار اجرا شوند. باید توجه شود که در اینجا فرض ما بر این است که تعداد تکرار حلقه‌ها در کرنل‌ها به مقدار داده‌ی ورودی بستگی ندارند. چراکه در این حالت پیش‌بینی دقیق تعداد تکرارهای حلقه به صورت ایستا غیرممکن است و اساساً پیش‌بینی ایستای زمان اجرا نیز با خطای بسیار زیادی همراه خواهد بود. بنابراین ما کرنل‌هایی را انتخاب کردیم که تعداد تکرار حلقه‌ها به اندازه‌ی ورودی یا اندازه‌ی مسئله 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 حافظه‌ی برتراشه ذخیره می‌شوند.  $l_{mr}$  و  $l_{mw}$  به ترتیب تعداد دستورالعمل‌های خواندن و نوشتن در حافظه‌ی سراسری هستند. باید توجه شود که خیلی مهم است که این دو دستورالعمل تفکیک شوند. چون عملیات نوشتن در حافظه‌ی 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 حافظه‌ی اشتراکی هستند. ما از طریق ریزمحک‌زنی تاخیر خواندن و نوشتن در حافظه‌ی اشتراکی را روی دستگاه 940MX که معماری مکسول است، امتحان کردیم. تاخیر خواندن از حافظه‌ی 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 معادله‌ی (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 نتیجه‌ی یک دستورالعمل، به عنوان ورودی در دستورالعمل دیگری استفاده شود. وابستگی داده‌ای یک گلوگاه

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.

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

جدول (5): نحوه‌ی محاسبه معیار  $dpc$  در برنامه‌ی NN.

Inst#	Instructions	U	$U[i] - i$
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 (4) یعنی معیارهای  $n_{reg}$  و  $ShMem$  با کمک مترجم NVCC محاسبه می‌شوند. برای این منظور از گزینه‌ی ترجمه‌ی `--ptxas-options=-v` استفاده می‌شود. هر کلمه در حافظه را معادل 4 بایت در نظر می‌گیریم که معادل اندازه‌ی یک عدد اعشاری است. اهمیت این دو معیار به دلیل تأثیر در زمانبندی سطح یک در گیگاترد است. هرچقدر ثبات‌های تخصیصی و یا میزان حافظه‌ی اشتراکی مورد استفاده یک بلوک بیشتر باشد، آنگاه تعداد بلوک‌هایی توسط گیگاترد به‌صورت همزمان به یک چندپردازنده‌ی جریان‌ی تخصیص می‌یابد، کاهش می‌یابد. لذا اجرای همروند بلوک‌ها در یک چندپردازنده‌ی جریان‌ی کاهش می‌یابد. این مسئله به دلیل محدودیت منابع درون یک چندپردازنده‌ی جریان‌ی است. بنابراین این معیارها در زمان

مهم برای کارآیی برنامه‌های کودا است. چراکه باعث می‌شود اجرای موازی دستورالعمل‌ها در واحدهای تابعی<sup>1</sup> درون SM‌ها به تأخیر بیفتند. به این ترتیب که اگر دستورالعمل  $j$ ام به دستورالعمل  $i$ ام ( $i < j$ ) وابستگی داده‌ای داشته باشد، آنگاه صدور<sup>2</sup> دستورالعمل  $j$ ام تا کامل شدن اجرای دستورالعمل  $i$ ام به تأخیر می‌افتد [1]. همچنین اگر بین دو دستورالعمل متوالی وابستگی داده‌ای وجود داشته باشد، صدور دوگانه<sup>3</sup> دستورالعمل‌ها توسط زمانبند وارپ انجام نمی‌گیرد. برای محاسبه‌ی  $dpc$  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 اندازه‌ی  $\frac{1}{U[i]-i}$  افزوده می‌شود. در جدول (5) کدهای PTX برنامه‌ی 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 معادله‌ی (2) محاسبه می‌شود. در این معادله  $k$  تعداد خطوط برنامه‌ی PTX است و در جدول (5) این عدد برابر 29 است. در واقع از مجموع مقادیر میانگین گرفته شده است.

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 نتیجه‌ی دستورالعمل  $i$ ام زودتر مورد استفاده قرار می‌گیرد و بنابراین موازی‌سازی در واحدهای تابعی کمتر است.

<sup>1</sup> Functional Units  
<sup>2</sup> Issue  
<sup>3</sup> Dual-issue

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<sup>2</sup> است و با استفاده از جعبه ابزار متلب پیاده‌سازی شده است. از روش‌های انتخاب ویژگی برای کاهش تعداد ویژگی‌ها و در برخی موارد بهبود کارایی مدل هم استفاده می‌شود [46,45]. اما در اینجا هدف کاهش تعداد ویژگی نیست.

#### 5- ارزیابی مدل پیشنهادی

آزمایشاتی برای تعیین میزان کارایی 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.

جدول (6): مشخصات دستگاه استفاده شده در آزمایشات.

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

#### 5-1- روش‌شناسی آزمایشات

مشخصات دستگاه استفاده شده در آزمایشات در جدول (6) 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

اجرای کرنل تاثیرگذار هستند و به همین دلیل به‌عنوان ویژگی‌های ورودی مجموعه داده در نظر گرفته شده‌اند.

پس از ایجاد مجموعه داده، با استفاده از روش حداکثر-حداقل<sup>1</sup> عملیات نرمال‌سازی روی مجموعه داده اجرا شده است و ویژگی‌های ورودی به بازه‌ی صفر تا یک نگاشت شده‌اند. سپس به‌صورت 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.

#### 4-2- 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 به مجموعه داده‌های آموزش و آزمون) اجرا شده است.

#### 4-3- انتخاب ویژگی

در جدول (4) 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 هر یک از ویژگی‌های ورودی مجموعه داده چقدر است. این اطلاعات

<sup>2</sup> Minimum Redundancy Maximum Relevance (MRMR)

<sup>1</sup> Max-Min

جدول (8): ویژگی‌های استخراج شده برای کرنل‌های محاسبه‌گرا.

ویژگی	Hotspot	MM	Gussian,Fan1	Gussian,Fan2
$\alpha$	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
$n_{req}$	34	22	12	11

جدول (9): ویژگی‌های استخراج شده برای کرنل‌های حافظه‌گرا

ویژگی	NN	Srad,Kernel1	Srad,Kernel2	VecAdd
$\alpha$	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	0
$dpc$	0/5738	0/5717	0/5852	0/5943
$ShMem$	0	6144	5120	0
$n_{req}$	9	19	17	10

## 5-2- نتایج آزمایشات

در این بخش نتایج حاصل از اجرای روش‌های یادگیری ماشین روی مجموعه‌های داده ارائه می‌شوند. مجموعه‌های داده براساس ویژگی‌های ایستای کرنل‌ها ایجاد شده‌اند. برخی از ویژگی‌ها مختص هر کرنل بوده و برای هر کرنل فقط یکبار محاسبه می‌شوند. در جدول (8) مقادیر محاسبه‌شده برای کرنل‌های محاسبه‌گرا و در جدول (9) مقادیر محاسبه‌شده برای کرنل‌های حافظه‌گرا ارائه شده‌اند. مقدار ویژگی  $cpx$  در مجموعه داده کرنل‌های حافظه‌گرا برای همه‌ی نمونه‌ها برابر یک است. لذا این ویژگی از این مجموعه داده حذف شده است.

سه روش ماشین یادگیری افراطی (ELM)، شبکه‌های عصبی پرسپترون چندلایه (MLP) و ماشین بردار پشتیبان رگرسیونی (SVMR) برای پیش‌بینی زمان اجرا پیاده‌سازی شده‌اند. خطای

16/1 با استفاده از کودای 9 اجرا شده‌اند. برای پیاده‌سازی روال‌های یادگیری ماشین از محیط متلب 2021 در سیستم عامل ویندوز 10 استفاده شده است.

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. علاوه‌براین، برای هر یک از روش‌های یادگیری ماشین، مدت زمان روال‌های آموزش و آزمون نیز گزارش خواهند شد.

جدول (7): محک‌های استفاده شده برای ساخت مجموعه‌های داده.

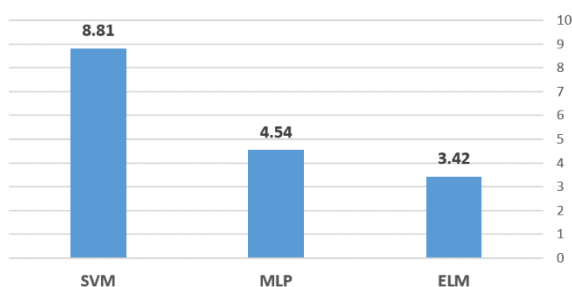
محک	برنامه	کرنل	C/M <sup>1</sup>
Rodinia 3.1	SRAD v2	srad_cuda_1	M
Rodinia 3.1	SRAD v2	srad_cuda_2	M
Rodinia 3.1	NN	euclid	M
CUDA Samples	Vec Add	addKernel	M
Rodinia 3.1	Hotspot	calculate_temp	C
CUDA Samples	Matrix Multiply	matrixMulCUDA	C
Rodinia 3.1	Gaussian	Fan1	C
Rodinia 3.1	Gaussian	Fan2	C

برای ایجاد مجموعه‌های داده از محک‌های دنیای واقعی و معتبر استفاده شده است که فهرست آن‌ها در جدول (7) 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 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 (4) محاسبه شده‌اند و پس از اجرای کرنل و محاسبه‌ی زمان اجرای آن یک سطر از مجموعه داده به دست می‌آید.

<sup>1</sup> Computation-bound/Memory-bound

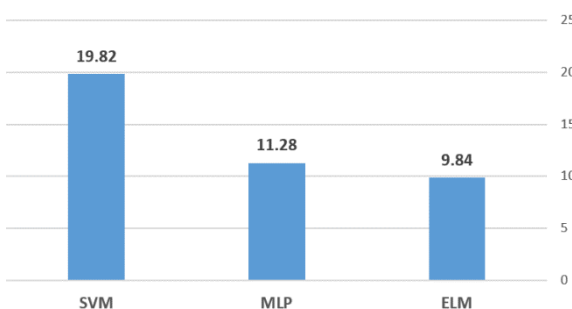
3/64 3/83	10/32 9/84	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	ELM
0/006 0/089	11/76 11/28	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	MLP
0/36 0/16	21/4 19/82	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	SVMR

حداکثر درصد خطای آزمون روی مجموعه داده محاسبه‌گرا



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

میانگین درصد خطای آزمون روی مجموعه داده حافظه‌گرا



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

پیش‌بینی این روش‌ها روی دو مجموعه داده محاسبه‌گرا و حافظه‌گرا به ترتیب در جداول (10) و (11) 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<sup>1</sup> 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.

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. علاوه‌بر 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 (12) مدت زمان اجرای این دو مرحله در هر سه روش روی مجموعه داده محاسبه‌گرا گزارش شده‌اند و نشان می‌دهد روش ELM سریع‌ترین روش است.

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

روش	درصد خطا		معیار ارزیابی	روش
	حداکثر	میانگین		
ELM	3/42	0/97	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	ELM
MLP	5/18	0/007	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	MLP
SVMR	10/78	0/016	خطای اعتبارسنجی 10-دسته‌ای خطای آزمون	SVMR

جدول (12): 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

### 3-5 - تاثیر جداسازی مجموعه‌های داده

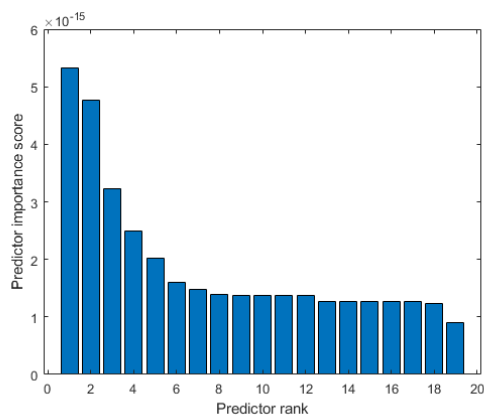
در آزمایش بعدی مجموعه‌های داده محاسبه‌گرا و حافظه‌گرا را با هم ادغام کردیم و سه روش یادگیری ماشین را روی این مجموعه داده اجرا کردیم. در واقع دو مجموعه داده آموزش را

جدول (11): دقت روش‌های یادگیری روی مجموعه داده حافظه‌گرا.

روش	درصد خطا		معیار ارزیابی	روش
	حداکثر	میانگین		

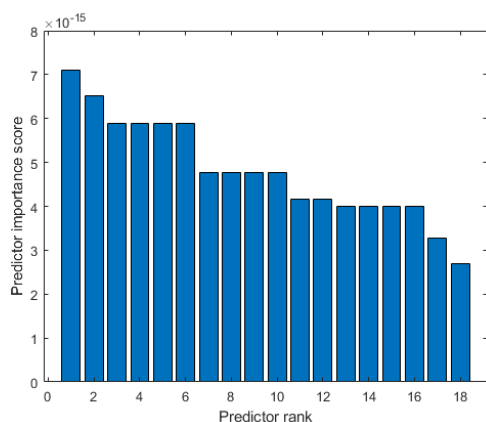
<sup>1</sup> Activation Function

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 (14) میزان حساسیت هر ویژگی گزارش شده است. ویژگی  $cpx$  به دلیل داشتن مقادیر ثابت از مجموعه داده حافظه‌گرا حذف شده است.



idx = 5 15 4 7 6 16 1 11 14 12 18 13 19 17 3 8 9 10 2

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



idx = 4 15 5 16 2 6 7 18 8 3 9 1 11 17 12 13 14 10

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

## 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

با هم 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 (13) گزارش شده‌اند.

جدول (13): دقت روش‌های یادگیری روی مجموعه داده ادغام شده.

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

جدول (14): 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
$\alpha$	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

## 4-5- تاثیرگذاری هریک از ویژگی‌های ورودی

برای ارزیابی میزان تاثیر هریک 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 (14) ارائه شده‌اند. خروجی روال انتخاب ویژگی به صورت یک نمودار ارائه شده است که در آن ویژگی‌ها به ترتیب

دستورالعمل‌های حافظه نیز بیشتر است. در عوض ما ویژگی‌های  $l_c, l_{mr}, l_{mw}, l_{shr}, l_{shu}, l_{DPU}$  و  $l_{SFU}$  را معرفی کردیم که براساس کدهای PTX به دست می‌آیند و هر یک تخمینی از تعداد اجرای دستورالعمل‌ها هستند. محاسبه‌ی این ویژگی‌ها با شمارش تعداد دستورالعمل‌ها انجام نشده و براساس 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. در واقع 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 نویتی گردشی است که وابستگی زیادی به تراکنش‌های حافظه دارد. زمانبندی بلوک‌ها روی یک چندپردازنده‌ی جریانی و اجرای همروند چندین بلوک روی یک چندپردازنده‌ی 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 (13) نشان

محاسبه‌گرا بهتر از کرنل‌های حافظه‌گرا بوده است. دلیل این مسئله این است که اساساً عمومی‌سازی کارایی 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 حافظه‌ی کش به‌طور 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 کارایی 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. باتوجه به اینکه روش ELM براساس شبکه‌های عصبی پیش‌رو<sup>1</sup> ایجاد شده است و تنها یک لایه‌ی مخفی دارد، سرعت بسیار خوبی در روال‌های آموزش و آزمون ارائه کرده است.

در این تحقیق چندین معیار ایستا برای کرنل‌های کودا ارائه شده‌اند که مهم‌ترین نوآوری این تحقیق محسوب می‌شود. پارامتر  $\alpha$  ویژگی چندان موثری برای پیش‌بینی زمان اجرای کرنل نیست. چراکه این ویژگی براساس شمارش دستورالعمل‌های محاسباتی و حافظه در کدهای کودا محاسبه می‌شود و دستورالعمل‌های محاسباتی همگی یکسان در نظر گرفته شده‌اند. درحالی‌که 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

<sup>1</sup> Feed-forwarding Neural Network

بنابراین تراکنش‌های حافظه‌ی این وارپ‌ها تقریباً هم‌زمان رخ می‌دهند. به همین دلیل در کرنل‌های حافظه‌گرا که تراکنش‌های حافظه‌ی بیشتری دارند، تنظیم تعداد بلوک‌ها اهمیت بیشتری نسبت به تعداد نخ‌ها دارد و افزایش تعداد بلوک‌ها می‌تواند کارایی 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 اندازه‌ی بلوک اهمیت بیشتری دارد. نتایج ارائه شده در جدول (14) این نکات را تایید می‌کنند.

مزیت مدل پیشنهادی در مقایسه با مدل‌های تحلیلی این است که استفاده‌کنندگان از پرداختن به جزئیات سطح پائین 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. نقطه‌ی ضعف مدل پیشنهادی این است که برای هر دستگاه باید به‌طور جداگانه ساخته شود و قابلیت‌حمل 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. مثلاً از معماری ولتا به بعد مفهوم زمانبندی نخ‌های مستقل<sup>2</sup> توسعه‌یافته است و این تغییر باعث می‌شود که بسیاری از مدل‌های تحلیلی که براساس 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.

دادیم که در صورتیکه دو مجموعه داده محاسبه‌گرا و حافظه‌گرا باهم 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 کارایی کرنل کمک کرده است.

در جدول (3) 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 نکته‌ی بسیار مهمی که باید به آن توجه شود این است که در [32] برای ساختن مجموعه داده هم‌زمان از ویژگی‌های ایستا و پویای کرنل استفاده شده است. نویسندگان در این پژوهش معیارهایی مانند اشغال<sup>1</sup> را نیز به‌عنوان ویژگی‌های ورودی در مجموعه داده لحاظ کردند. این عمل با هدف ایجاد مدل کارایی در تناقض است. چراکه معیارهای پویا در اثر اجرای کرنل و انجام پروفایلینگ به دست می‌آیند. مدل کارایی 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 (14) 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 رابطه‌ی مستقیمی با زمانبندی در پردازنده‌ی گرافیکی دارد. در صورتیکه 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،

<sup>1</sup> Independent Thread Scheduling

<sup>1</sup> Occupancy



## 7- نتیجه گیری و کارهای آینده

در این تحقیق یک مدل کارآیی مبتنی بر یادگیری ماشین برای پیش بینی زمان اجرای کرنل های کودا ارائه شده است. در طرح پیشنهادی دو مدل جداگانه برای کرنل های محاسبه گرا 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 می شود. برای تعیین این مقادیر نیازی به اجرای برنامه ها نیست. برای ساختن مجموعه داده از چهار کرنل محاسبه گرا و چهار کرنل حافظه گرا استفاده کردیم. با اجرای این کرنل ها و انجام عملیات پروفایلینگ، زمان اجرای کرنل ها را اندازه گیری کرده و به عنوان برچسب خروجی در مجموعه داده ثبت کردیم. هر رکورد از مجموعه داده حاصل اجرای یک کرنل کودا است. ما مجموعه های داده جداگانه ای برای کرنل های محاسبه گرا و حافظه گرا ایجاد کردیم. سپس سه روش یادگیری ماشین ELM، MLP و SVM را روی این مجموعه های داده اجرا و نتایج آزمایشات را گزارش کرده ایم. نتایج نشان دادند که روش ELM زمان اجرای کرنل های محاسبه گرا را با حداکثر خطای 3/42 و زمان اجرای کرنل های حافظه گرا را با حداکثر خطای 9/84 درصد پیش بینی می کند. برای اعتبارسنجی فرآیندهای آموزش و آزمون ما اعتبارسنجی متقاطع 10-دسته ای را روی هر سه روش اجرا و نتایج را گزارش کردیم. همچنین برای اینکه میزان تاثیر هر یک از ویژگی های 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 کرده ایم.

به عنوان کارهای آینده پیشنهاد می کنیم که سایر روش های یادگیری ماشین مثل یادگیری عمیق را روی این مجموعه های داده اجرا کنیم. همچنین پیشنهاد می کنیم که مجموعه داده ای از معیارهای پویا ایجاد کرده و با استفاده از روش های انتخاب

ویژگی میزان تاثیر این ویژگی ها را روی زمان اجرای کرنل های

کودا تعیین کنیم.

تعارض منافع: نویسندگان اعلام می کنند که هیچ تعارض

منافعی ندارند.

## مراجع

- [1] NVIDIA, C.U.D.A. "CUDA C Programming Guide, Version 10.1.", NVIDIA Corp, 2019.
- [2] Liu, Siting, et al. "Fastgr: Global routing on cpu-gpu with heterogeneous task graph scheduler." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 7, pp 2317-2330, 2022, doi: 10.1109/TCAD.2022.3217668.
- [3] Zhang, Feng, et al. "Performance evaluation and analysis of sparse matrix and graph kernels on heterogeneous processors.", CCF Transactions on High Performance Computing, vol. 1, pp. 131-143, 2019, doi: 10.1007/s42514-019-00008-6.
- [4] Schoonhoven, Richard, Ben van Werkhoven, and K. Joost Batenburg. "Benchmarking optimization algorithms for auto-tuning GPU kernels." IEEE Transactions on Evolutionary Computation, vol. 27, no. 3, 2022, doi: 10.1109/TEVC.2022.3210654.
- [5] Mustafa, Dheya. "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] Lemeire, Jan, Jan G. Cornelis, and Elias Konstantinidis. "Analysis of the analytical performance models for GPUs and extracting the underlying Pipeline model." Journal of Parallel and Distributed Computing vol. 173, pp. 32-47, 2023, doi: 10.1016/j.jpdc.2022.11.002.
- [7] Lattuada, Marco, et al. "Performance prediction of deep learning applications training in GPU as a service systems." Cluster Computing, vol. 25, no. 2, pp. 1-24, 2022, doi: 10.1007/s10586-021-03428-8.
- [8] Weber, Nicolas, and Michael Goesele. "MATOG: array layout auto-tuning for CUDA." ACM Transactions on Architecture and Code Optimization (TACO), vol. 14, no. 3, pp. 1-26, 2017, doi: 10.1145/3106341
- [9] Al-Mouhamed, Mayez A., Ayaz H. Khan, and Nazeeruddin 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] Madougou, Souley, et al. "The landscape of GPGPU performance modeling tools." Parallel Computing, vol. 56, pp. 18-33, 2016, doi: 10.1016/j.parco.2016.04.002.
- [11] Reuther, Albert, et al. "Survey and benchmarking of machine learning accelerators." In high performance extreme computing conference (HPEC). IEEE, 2019, doi: 10.1109/HPEC.2019.8916327.
- [12] Riahi, Ali, Abdorreza Savadi, and Mahmoud 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] Peddie, Jon. "What is a GPU?" In The History of the GPU-Steps to Invention. Cham: Springer International Publishing, pp. 333-345, 2023, doi: 10.1007/978-3-031-10968-3\_7.

- Conference on High Performance Computing (HiPC), IEEE, 2015, doi: 10.1109/HiPC.2015.34.
- [34] Yun, Juwon, 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] Rummelg, Toomas, et al. "High-level hardware feature extraction for GPU performance prediction of stencils.", In Proceedings of the 13th Annual Workshop on General Purpose Processing Using Graphics Processing Unit, 2020, doi: 10.1145/3366428.3380769.
- [36] Bouzidi, Halima, et al. "Performance prediction for convolutional neural networks on edge gpus.", in Proceedings of the 18th ACM International Conference on Computing Frontiers, 2021. doi: 10.1145/3457388.3458666.
- [37] O'neal, Kenneth, et al. "GPU performance estimation using software rasterization and machine learning." ACM Transactions on Embedded Computing Systems (TECS), vol. 16, no. 5, pp. 1-21, 2017, doi: 10.1145/3126557.
- [38] Benatia, Akrem, et al. "Sparse matrix format selection with multiclass SVM for SpMV on GPU.", in 45th International Conference on Parallel Processing (ICPP), IEEE, 2016, doi: 10.1109/ICPP.2016.64.
- [39] Dao, Thanh Tuan, et al. "A performance model for GPUs with caches." IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 7, pp. 1800-1813, 2014, doi: 10.1109/TPDS.2014.2333526.
- [40] Lehnert, Christoph, et al. "Performance prediction and ranking of spmv kernels on gpu architectures.", in Euro-Par 2016: Parallel Processing: 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings 22. Springer International Publishing, 2016, doi: 10.1007/978-3-319-43659-3\_7.
- [41] Gianniti, Eugenio, Li Zhang, and Danilo Ardagna. "Performance prediction of gpu-based deep learning applications." in 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE, 2018, doi: 10.1109/CAHPC.2018.8645908.
- [42] NVIDIA, C. U. D. A. "CUDA C Programming Guide, Version 7.0.", NVIDIA Corp, 2015.
- [43] Hall, MarkA. "Practical machine learning tools and techniques." United State: Morgan Kaufman, 2011.
- [44] Carvalho, Pablo, et al. "Using machine learning techniques to analyze the performance of concurrent kernel execution on gpus." Future Generation Computer Systems, vol. 113, pp. 528-540, 2020, doi: 10.1016/j.future.2020.07.038.
- [45] عرفان صابری، الناز رادمند، جمشید پیرگری و علی کرمانی، "استراتژی خرید و فروش در بازار بورس ایران با استفاده از مدل های یادگیری ماشین همراه با انتخاب ویژگی به کمک الگوریتم جستجوی فاخته"، مجله محاسبات نرم، 1402، 1144، 252793.252793.1144، doi: 10.22052/scj.2023.252793.1144
- [46] ویسی هادی، قاید شرف، حمیدرضا ابراهیمی و مرتضی بهبود، "کارایی الگوریتم های یادگیری ماشین در تشخیص بیماری های قلبی با بهینه سازی داده ها و ویژگی ها" مجله محاسبات نرم، دوره 8، شماره 1، صفحات 70-85، doi: 10.22052/8.1.70.1398.85
- [47] Shekofteh, S. K., Noori, H., Naghibzadeh, M., Yazdi, H. S., & Fröning, H., "Metric selection for gpu kernel classification.", ACM Transactions on Architecture and Code Optimization (TACO), vol. 15, no. 4, pp. 1-27, 2019, doi: g/10.1145/3295690.
- [14] Wang, Jialun, et al. "D-Cubicle: boosting data transfer dynamically for large-scale analytical queries in single-GPU systems." Frontiers of Computer Science, vol. 17, no. 4, 2023, doi: 10.1007/s11704-022-2160-z
- [15] NVIDIA, C. U. D. A. "CUDA C++ Programming Guide, Version 11.7.", NVIDIA Corp, 2022.
- [16] NVIDIA, C. U. D. A. "CUDA C++ Programming Guide, Version 12.1.", NVIDIA Corp, 2023.
- [17] NVIDIA, "Fermi Whitepaper.", NVIDIA Corp, 2009.
- [18] NVIDIA, "GTX 680 Whitepaper.", NVIDIA Corp, 2012.
- [19] NVIDIA, "980: Featuring Maxwell, the most advanced GPU ever made.", NVIDIA Corp, 2014.
- [20] NVIDIA, "Whitepaper NVIDIA GeForce GTX 1080 Gaming Perfected", NVIDIA Corp, 2016
- [21] NVIDIA, "NVIDIA Turing GPU architecture: Graphics reinvented", NVIDIA Corp, 2018
- [22] NVIDIA, "Tesla V100 GPU Architecture: The world's most advanced datacenter GPU.", NVIDIA Corp, 2017.
- [23] Glaskowsky, Peter N. "NVIDIA's Fermi: the first complete GPU computing architecture.", White paper 18, 2009.
- [24] NVIDIA, C. U. D. A. "CUDA C++ Programming Guide, Version 11.7.", NVIDIA Corp, 2022.
- [25] محمد پویا اکبرپور، کیهان خام فروش، وفا میهمی. "روشی برای بهبود الگوریتم بهینه سازی اجتماع ذرات با استفاده از CUDA بر روی پردازنده گرافیکی"، مجله محاسبات نرم، دوره 8، شماره 2، صفحات 2-21، doi: 10.22052/8.2.2.1398
- [26] Ghike, Swapnil, et al. "Directive-based compilers for GPUs" in Languages and Compilers for Parallel Computing: 27th International Workshop, LCPC 2014, Hillsboro, OR, USA, September 15-17, 2014, Revised Selected Papers 27. Springer International Publishing, 2015, doi: 10.1007/978-3-319-17473-0\_2.
- [27] Lopez-Novoa, Unai, Alexander Mendiburu, and José Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing.", IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 1, pp. 272-281, 2015, doi: 10.1109/TPDS.2014.2308216.
- [28] O'Neal, Kenneth, et al., "HALWPE: Hardware-Assisted Light Weight Performance Estimation for GPUs.", in Proceedings of the 54th Annual Design Automation Conference, ACM, 2017, doi: 10.1145/3061639.306225.
- [29] Wu, Gene, et al., "GPGPU performance and power estimation using machine learning.", in 21st International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2015, doi: 10.1109/HPCA.2015.7056063.
- [30] Huang, Jen-Cheng, et al., "GPUMech: GPU performance modeling technique based on interval analysis.", in 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE, 2014, doi: 10.1109/MICRO.2014.59.
- [31] Resios, Andreas, and V. Holdermans. "GPU performance prediction using parametrized models." Utrecht University, 2011.
- [32] Amaris, Marcos, et al., "A comparison of GPU execution time prediction using machine learning and analytical modeling.", in 15th International Symposium on Network Computing and Applications (NCA), IEEE, 2016, doi: 10.1109/NCA.2016.7778637.
- [33] Amaris, Marcos, et al., "A simple bsp-based model to predict execution time in gpu applications.", in 22nd International