Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: jscit.nit.ac.ir
DOI: 10.22034/jscit.2025.536245.2146
Volume 14, Number 4, Winter 2026, pp. 10-22
Received: 07/28/2025, Revised: 10/27/2025, Accepted: 11/01/2025

Predicting the Execution Time of Fused Kernels Using Machine Learning Methods

Ali Riahi[1*], Hamid Khorshidian Mianaee [2] and Abdorreza Savadi[3]
1*- Department of Electrical and Computer Engineering, University of Science and Technology of Mazandaran, Behshahr, Iran
2,3- Computer Engineering Department, Ferdowsi University of Mashhad, Mashhad, Iran.
[1*]riahi@mazust.ac.ir, [2]hamid.khorshidianmianaee@alumni.um.ac.ir, and [3]savadi@um.ac.ir

Corresponding author's address: Ali Riahi, Department of Electrical and Computer Engineering, University of Science and Technology of Mazandaran, Behshahr, Iran.

Abstract– Kernel fusion is one of the common techniques used to optimize the performance of CUDA programs. In this technique, two kernels are combined into a single one. There are three main fusion approaches, and depending on the nature of the problem and the algorithm employed in each program, one approach may yield better performance than the others. Programmers typically determine the appropriate fusion method through trial and error. Developing a performance model capable of predicting the execution time of fused kernels can eliminate this costly and time-consuming process. In this paper, we propose a performance prediction model that estimates the execution time of fused kernels created using each of the three fusion techniques, based on features extracted from the original kernels. This enables programmers to make more informed decisions when selecting the optimal fusion method. The proposed model is developed using machine learning techniques. To construct the dataset, we used eight programs from the NVIDIA CUDA Sample suite and the Rodinia benchmark. Kernels were fused pairwise using the three fusion techniques, and features were extracted from each pair to form the dataset. Nine machine learning algorithms were implemented and evaluated using k-fold cross-validation. Among them, the Random Forest algorithm achieved the best performance and was used to build the final prediction model. Experimental results demonstrate that the proposed model predicts the execution time of fused kernels with an average error of less than 5%. By leveraging machine learning, the proposed approach significantly simplifies the kernel fusion method selection process for programmers.

Keywords– GPU, CUDA, Tuning, Kernel Fusion, Performance Model, Machine Learning.

مجله علمی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
شناسه دیجیتال: 10.22034/jscit.2025.536245.2146
جلد ۱۴، شماره ۴، زمستان ۱۴۰۴، صفحه ۱۰-۲۲
دریافت: ۱۴۰۴/۰۵/۰۶، بازنگری: ۱۴۰۴/۰۸/۰۵، پذیرش: ۱۴۰۴/۰۸/۱۰

پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی‌شده با استفاده از روش‌های یادگیری ماشین

علی ریاحی[1*]، حمید خورشیدیان میانائی[2]، عبدالرضا سوادی[3]
*۱- گروه مهندسی کامپیوتر، دانشکده مهندسی برق و کامپیوتر، دانشگاه علم و فناوری مازندران، بهشهر، ایران.
۲ و۳- گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه فردوسی مشهد، مشهد، ایران.
[1*] riahi@mazust.ac.ir, [2] hamid.khorshidianmianaee@alumni.um.ac.ir, [3] savadi@um.ac.ir

* نشانی نویسنده مسئول: علی ریاحی گروه مهندسی کامپیوتر، دانشکده مهندسی برق و کامپیوتر، دانشگاه علم و فناوری مازندران، بهشهر، ایران.

چکیده- هم‌جوشی کرنل‌ها یکی از تکنیک‌های رایج در تنظیم کارآیی برنامه‌های کودا به‌شمار می‌رود. در این تکنیک دو کرنل با هم ترکیب شده و یک کرنل ایجاد می‌کنند. سه روش برای هم‌جوشی وجود دارد که بسته به ماهیت مسئله و الگوریتم استفاده شده در هر برنامه، یکی از این روش‌ها می‌تواند کارآمدتر باشد. برنامه‌نویسان معمولاً ازطریق روش آزمون و خطا، یکی از این روش‌ها را انتخاب می‌کنند. وجود یک مدل کارآیی که زمان اجرای کرنل‌های هم‌جوشی‌شده را پیش‌بینی کند، می‌تواند از آزمون و خطا توسط برنامه‌نویس جلوگیری کند. در این مقاله ما یک مدل کارآیی ارائه کردیم که براساس ویژگی‌های مستخرج از کرنل‌های اولیه، زمان اجرای کرنل‌های هم‌جوشی‌شده با هریک از این سه تکنیک را پیش‌بینی می‌کند. بنابراین برنامه‌نویس با آگاهی بهتری می‌تواند بهترین روش هم‌جوشی را انتخاب کند. مدل پیشنهادی ما براساس روش‌های یادگیری ماشین ایجاد شده است. مجموعه داده‌های مدل با استفاده از هشت برنامه از مجموعه مثال‌های NVIDIA و محک رودینا ساخته شده‌اند. کرنل‌ها دو به دو با هم و با استفاده از سه تکنیک هم‌جوشی شدند. از هریک ویژگی‌هایی استخراج شده و مجموعه داده آماده شده است. نه روش یادگیری ماشین پیاده‌سازی شده و با استفاده از روش اعتبارسنجی k-دسته‌ای مورد ارزیابی قرار گرفتند. بهترین روش جنگل تصادفی بوده است و برای ساخت مدل زمان استفاه از این روش استفاده شده است. یافته‌های تجربی نشان می‌دهند که میانگین خطای مدل پیشنهادی در پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی‌شده کمتر از ۵ درصد بوده است. مدل پیشنهادی با بهره‌گیری از یادگیری ماشین، فرآیند انتخاب روش هم‌جوشی کرنل را برای برنامه‌نویسان تسهیل می‌کند.

واژه‌های کلیدی: پردازنده‌ی گرافیکی، کودا، تنظیم، هم‌جوشی کرنل، مدل کارآیی، یادگیری ماشین.

۱- مقدمه

واحد پردازنده‌ی گرافیکی به‌عنوان یک پردازنده‌ی همکار و یا به‌عنوان یک شتاب‌دهنده، به‌طور گسترده در محاسبات با کارایی بالا استفاده می‌شود. مدل برنامه‌نویسی کودا، روشی اختصاصی برای بهره‌برداری از پردازنده‌های گرافیکی شرکت NVIDIA به‌شمار می‌رود. در اصطلاح به پردازنده‌ی مرکزی میزبان[1] و به پردازنده‌ی گرافیکی دستگاه[2] گفته می‌شود. یک برنامه از سمت میزبان آغاز شده و تابعی که کرنل[3] نام دارد، برای دستگاه ارسال می‌شود تا به روش یک برنامه چندین داده[4] اجرا شود [۱]. استفاده‌ی بهینه از توان محاسباتی دستگاه یک چالش مهم برای برنامه‌نویسان کودا محسوب می‌شود. علی‌رغم مصرف انرژی زیاد، به‌دلیل وجود محدودیت‌هایی در استفاده از حافظه، حداکثر کارآیی مورد انتظار در اکثر موارد محقق نمی‌شود [۲]. بنابراین

الف) روش بین بلوکی	ب) روش درون بلوکی	ج) روش درون نخی
$FusedKernel\ (argList1, argList2)\ \{$ $bid = calculated\ block\ id$ $if\ (bid < b)$ $Instructions\ of\ Kernel1$ $else$ $Instructions\ of\ Kernel2$ $\}$	$FusedKernel\ (argList1, argList2)\ \{$ $tid = calculated\ thread\ id$ $if\ (tid < t)$ $Instructions\ of\ Kernel1$ $else$ $Instructions\ of\ Kernel2$ $\}$	$FusedKernel\ (argList1, argList2)$ $\{$ $Instructions\ of\ Kernel1$ $Instructions\ of\ Kernel2$ $\}$

شکل ۱: نحوه‌ی بکارگیری سه تکنیک برای هم‌جوشی دو کرنل کودا.

برنامه‌نویسان کودا از روش‌های تنظیم[۵] برای بهبود کارآیی برنامه‌ها استفاده می‌کنند. در این روش‌ها پارامترهای کرنل براساس ویژگی‌های نرم‌افزاری برنامه و همچنین براساس ویژگی‌های سخت‌افزاری دستگاه، تنظیم می‌شوند تا کارآیی بهبود یابد. در هر برنامه بسته به ماهیت و اندازه‌ی مسئله، گزینه‌های زیادی برای تنظیم ویژگی‌های کرنل وجود دارند که فضای جستجو را تشکیل می‌دهند [۳]. برنامه‌نویس موظف است در این فضای جستجو بهترین گزینه را انتخاب کند. در اکثر موارد فضای جستجو بسیار بزرگ است و بنابراین امکان بررسی تمام حالت‌های ممکن وجود ندارد. پیدا کردن تنظیم بهینه یک چالش کلیدی برای برنامه‌نویسان کودا است. برنامه‌نویسان مجبورند حالت بهینه را با روش آزمون و خطا پیدا کنند.

برنامه‌هایی که روی پردازنده‌ی گرافیکی اجرا می‌شوند، معمولا از تعدادی فراخوانی کرنل تشکیل می‌شوند. اجرای متوالی این کرنل‌ها ممکن است موجب بهره‌وری پایین و استفاده ناکارآمد از قابلیت‌های سخت‌افزاری پردازنده‌ی گرافیکی گردد. چراکه معمولا سربارهای زیادی به همراه فراخوانی کرنل‌ها وجود دارند. در این موارد تکنیک هم‌جوشی کرنل[۶] می‌تواند به بهبود کارآیی کمک کند. برطبق این تکنیک، دو کرنل باهم ترکیب شده و به‌صورت یک کرنل بزرگ‌تر فراخوانی می‌شوند. یکی از مزیت‌های هم‌جوشی کرنل این است که از سربار راه‌اندازی کرنل‌ها کاسته می‌شود. همچنین با استفاده از این تکنیک، تبادل داده بین میزبان و دستگاه کاهش می‌یابد و از داده‌ها استفاده مجدد می‌شود [۴]. مزیت دیگر این است که با حذف برخی محاسبات تکراری، موازی‌سازی در سطح دستورالعمل بهبود می‌یابد [۵]. استفاده از هم‌جوشی کرنل چالش‌هایی را ممکن است به‌همراه داشته باشد. ازجمله اینکه کرنل‌های هم‌جوشی‌شده ممکن است پیچیدگی‌های زیاد [۶] و یا انعطاف‌پذیری پایینی داشته باشند [۷]. همچنین ممکن است در الگوریتم نهایی ناسازگاری‌هایی وجود داشته باشد [۸]. علاوه‌براین با افزایش حجم استفاده از منابع، ممکن است بلوک‌های فعال کم شوند. این مسئله باعث کاهش سطح موازی‌سازی و افت اشغال می‌شود [۹]. علاوه‌براین، در شرایطی که کرنل نیازمند همگام‌سازی‌های درون بلوکی و بین بلوکی است، حفظ ترتیب صحیح اجرای دستورالعمل‌ها دشوار می‌شود [۷]. ضمن اینکه درصورت وجود وابستگی داده‌ای بین دو کرنل، ممکن است هم‌جوشی آن‌ها امکان‌پذیر نباشد و یا به سختی انجام شود [۱۰]. برای این منظور باید در کرنل هم‌جوشی‌شده نگاشت داده‌ها و محاسبات به نخ‌ها را به‌گونه‌ای تنظیم کرد که داده‌ی تولید شده توسط دستورالعمل‌های کرنل اول، در اختیار دستورالعمل‌های کرنل دوم قرار گیرد. درصورتی‌که کرنل‌ها این قابلیت را داشته باشند، می‌توان دو کرنل با وابستگی داده‌ای را نیز باهم هم‌جوشی کرد.

هم‌جوشی کرنل به سه روش درون نخی، درون بلوکی و بین بلوکی انجام می‌شود [۱۱]. این سه روش در شکل ۱ نشان داده شده‌اند. در روش درون نخی، هر نخ ابتدا دستورالعمل‌های کرنل اول و سپس دستورالعمل‌های کرنل دوم را اجرا می‌کند. در روش درون بلوکی، نخ‌های یک بلوک به دو دسته تقسیم می‌شوند. دسته‌ی اول دستورالعمل‌های کرنل اول و دسته‌ی دوم دستورالعمل‌های کرنل دوم را اجرا می‌کنند. در روش بین بلوکی هر بلوک فقط به یک کرنل اختصاص دارد و بلوک‌ها بین دو کرنل تقسیم می‌شوند. هریک از این روش‌ها مزیت‌هایی دارند و بسته به ماهیت مسئله می‌توانند بهتر از سایرین باشند. برنامه‌نویس در هنگام تنظیم باید تصمیم بگیرد که به کدامیک از این سه روش هم‌جوشی را انجام دهد. پیش‌بینی زمان اجرای هریک از کرنل‌های هم‌جوشی‌شده می‌تواند به برنامه‌نویس کمک کند تا روش هم‌جوشی را به‌درستی انتخاب کند و از آزمون و خطا پرهیز کند. هم‌جوشی کرنل‌های کودا توسط کامپایلرهای معمولی انجام نمی‌شود، چراکه صحیح نگه داشتن کد هم‌جوشی‌شده بسیار سخت است [۱۲]. با این حال، این تکنیک در کامپایلرهای مبدأ به مبدأ[۷] برای تنظیم خودکار برنامه‌های کودا تولیدشده استفاده می‌شود [۱۵-۱۲].

در این مقاله، با درنظرگرفتن کرنل‌هایی که امکان هم‌جوشی برای آن‌ها وجود دارد، ویژگی‌های مؤثر بر زمان اجرای کرنل‌های هم‌جوشی‌شده شناسایی شده است. سپس با روش‌های مبتنی‌بر

یادگیری ماشین مدلی جهت پیش‌بینی زمان اجرای آن‌ها ساخته شده است. ما با استفاده از کرنل‌های کاندید، تکنیک‌های هم‌جوشی کرنل را روی آنها اجرا کرده و در هر مورد معیارهای ارزیابی موثر را از کرنل‌های اولیه و کرنل هم‌جوشی‌شده استخراج کردیم. با استفاده از این اطلاعات مجموعه داده‌ای برای مدل یادگیری ماشین ساختیم. همچنین از روش‌های انتخاب ویژگی برای کاهش ابعاد مجموعه داده و بهبود کارآیی مدل استفاده کردیم. این مدل به برنامه‌نویس کمک می‌کند تا قبل از بکارگیری تکنیک هم‌جوشی، بهترین تکنیک را برای هم‌جوشی کرنل‌ها بیابد.

در ادامه‌ی این مقاله، در بخش ۲ پیش‌زمینه‌های لازم برای درک روش پیشنهادی ارائه می‌شود. سپس در بخش ۳ پیشینه‌ی تحقیق بررسی خواهد شد. سپس در بخش ۴ روش پیشنهادی برای پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی‌شده ارائه خواهد شد. در بخش بعدی ارزیابی مدل پیشنهادی و در بخش ۶ بحث و در انتها نتیجه‌گیری و کارهای آینده ارائه خواهند شد.

۲- پیش‌زمینه

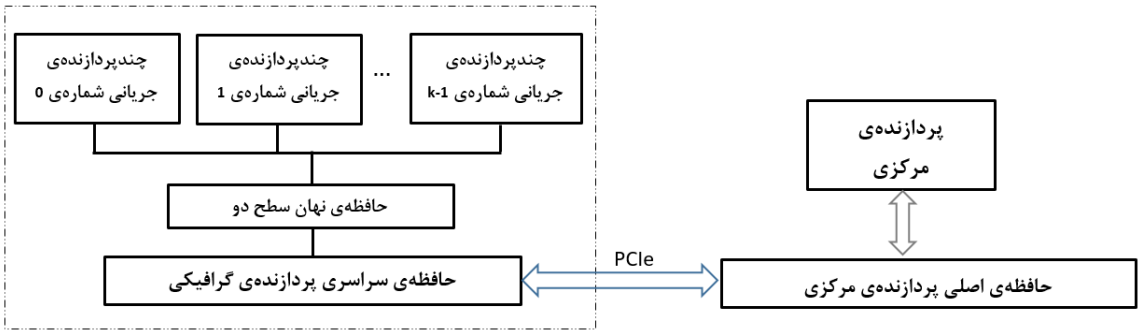
۲-۱- معماری پردازنده‌ی گرافیکی

درک ساختار و معماری پردازنده‌ی گرافیکی نقش بسیار مهمی در تنظیم برنامه‌ها ایفا می‌کند. برنامه‌نویس باید روش‌های تنظیم را براساس ویژگی‌های سخت‌افزاری و میزان منابع دردسترس استفاده کند. در حال حاضر دو نوع پردازنده‌ی گرافیکی دردسترس هستند. در پردازنده‌های گرافیکی یکپارچه[۸] فضای آدرس پردازنده‌های مرکزی و گرافیکی مشترک است. این نوع دستگاه‌ها در تجهیزات خاص مثل تجهیزات تصویربرداری در پزشکی [۱۶] و مسیریاب‌ها [۱۷] استفاده می‌شوند. اما در پردازنده‌های گرافیکی گسسته[۹] که متداول‌تر هستند، فضای آدرس پردازنده‌های مرکزی و گرافیکی مجزا هستند. بنابراین قبل از اجرای کرنل، داده‌ها باید از حافظه‌ی اصلی در سمت میزبان به حافظه‌ی سراسری در سمت دستگاه منتقل شوند. پس از اجرای کرنل نیز نتایج باید در جهت معکوس بازگردانده شوند تا در ادامه برنامه‌ی سمت میزبان مورد استفاده قرار گیرند [۱۸]. شکل ۲ ساختار پردازنده‌های گرافیکی گسسته و سلسله‌مراتب حافظه‌ی آن نشان می‌دهد. تنها حافظه‌ی پردازنده‌ی گرافیکی که با حافظه‌ی اصلی در ارتباط است، حافظه‌ی سراسری است [۱۹]. این ارتباط ازطریق خطوط PCI-Express فراهم شده است که سرعت انتقال داده‌ی آن در حد چند گیگابایت بر ثانیه است [۲۰].

همانطور که در شکل ۲ نشان داده شده است، یک حافظه‌ی سراسری و یک حافظه‌ی نهان سطح دو بین تمام هسته‌های[۱۰] پردازشی به اشتراک گذاشته شده‌اند. هسته‌های پردازشی در واحدهایی به نام چندپردازنده‌های جریانی[۱۱] سازماندهی شده‌اند. درک ساختار و میزان منابع موجود در این واحدها نقش تعیین‌کننده‌ای در پیکربندی کرنل‌های کودا ایفا می‌کند. در داخل هر چندپردازنده‌ی جریانی، علاوه بر هسته‌های پردازشی، حافظه‌ی اشتراکی، رجیسترها، حافظه‌های نهان سطح یک، بافت[۱۲] و ثابت وجود دارند. حافظه‌ی اشتراکی توسط همه‌ی نخ‌های یک بلوک به‌صورت اشتراکی مورد استفاده قرار می‌گیرد. این حافظه بسیار سریع بوده و سرعت انتقال داده در آن درحد حافظه‌ی نهان سطح یک است. در برخی از معماری‌های پردازنده‌های گرافیکی NVIDIA مثل فرمی و پاسکال، این حافظه با حافظه‌ی نهان سطح یک یکپارچه هستند [۲۲,۲۱].

۲-۲- مدل برنامه‌نویسی کودا

یک کرنل کودا توسط برنامه‌ی سمت میزبان فراخوانی می‌شود. کرنل تابعی سریال است، اما به تعداد زیاد نخ مشابه ایجاد می‌شود و هر نخ روی داده‌ی متناظر با خودش اجرا می‌شود. هنر موازی‌سازی برنامه‌نویس انتساب درست نخ‌ها به داده‌هاست. کل نخ‌های یک کرنل درقالب یک گرید شناخته می‌شوند. تعداد نخ‌ها توسط پیکربندی کرنل به‌صورت $kernelname \ll\ll n_b.n_t \gg\gg$ تعیین می‌شوند. n_b تعداد بلوک‌های یک گرید است که اندازه‌ی



شکل ۲: نمایش ساختار سلسله مراتب حافظه‌ی پردازنده‌های گرافیکی گسسته.

گرید نیز نام دارد. پارامتر n_t تعداد نخ‌های یک بلوک است که اندازه‌ی بلوک نیز نامیده می‌شود. نخ‌ها در داخل یک بلوک درقالب دسته‌های ۳۲ تایی به نام وارپ سازماندهی می‌شوند. در نسل‌های قدیمی پردازنده‌های گرافیکی NVIDIA (تا پاسکال) همه‌ی نخ‌های یک وارپ، رجیستر شمارنده‌ی برنامه‌ی مشترک داشتند و باهم اجرا می‌شدند. اما از نسل ولتا به بعد تکنیک زمانبندی نخ مستقل[۱۳] در سطح درایور دستگاه ایجاد شده است و هر نخ دارای شمارنده‌ی برنامه‌ی منحصربه‌فرد است [۲۳]. اهمیت پیکربندی کرنل به این است که تمام نخ‌های یک بلوک روی یک چندپردازنده‌ی جریانی اجرا می‌شوند. بنابراین نخ‌های درون بلوک می‌توانند با استفاده از منابع موجود در یک چندپردازنده‌ی جریانی با هم همکاری کنند. اجرای یک بلوک زمانی به اتمام می‌رسد که همه‌ی وارپ‌های موجود در آن خاتمه یابند. همزمان ممکن است چند بلوک به یک چندپردازنده‌ی جریانی تخصیص یابند، اما یک بلوک بین چندپردازنده‌های جریانی مشترک نیست [۲۴]. مدل اجرای نخ‌ها در یک وارپ روی یک چندپردازنده‌ی جریانی به‌صورت یک دستورالعمل، چندین داده[۱۴] است [۲۵]. اندازه‌ی گرید و اندازه‌ی بلوک می‌توانند به‌صورت پیکربندی‌های دوبعدی و یا سه بعدی نیز تعیین شوند. این ابعاد به برنامه‌نویس کمک می‌کنند که انتساب نخ‌ها به داده‌ها را با کیفیت بهتری انجام دهد، وگرنه در بهبود کارآیی برنامه‌ی موازی تاثیری ندارد.

۳- پیشینه‌ی تحقیق

تکنیک هم‌جوشی کرنل در سال‌های اخیر مورد توجه پژوهشگران بوده است. در این بخش مهم‌ترین پژوهش‌ها روی این تکنیک مورد بررسی قرار گرفته است.

فوزک و همکاران [۱۳] یک مدل کارآیی برای پیش‌بینی زمان اجرای یک کرنل کودا ارائه کردند. در مدل آن‌ها میزان همپوشانی محاسبات و ارتباطات حافظه محاسبه می‌شود. سپس آن‌ها این مدل کارآیی را در یک کامپایلر مبدأ به مبدأ به کار گرفتند. در این کامپایلر تکنیک هم‌جوشی کرنل به‌صورت اتوماتیک برای تنظیم برنامه‌ی خروجی کودا اعمال شده است.

وانگ [۱۱] برای رفع مشکل وابستگی داده‌ای در هم‌جوشی کرنل‌ها، ابتدا آن‌ها را با استفاده از تکنیک Strip-mining به چند کرنل تجزیه و سپس این کرنل‌ها را باهم هم‌جوشی کرده است. برای هم‌جوشی نیز دو تکنیک استفاده کرده است. این دو تکنیک شامل هم‌جوشی تکه‌های با اندازه‌های ثابت و متغیر بوده است. وانگ نشان داد که روش هم‌جوشی تکه‌های با اندازه‌ی متغیر سازگاری بهتری ارائه کرده است. همچنین ازطریق آزمایش‌ها نشان داد که با اجرای این روش، انرژی مصرفی کرنل‌ها بهبود یافته است.

قیائو و همکاران [۱۴] با بهره‌وری متعادل از منابع به‌وسیله‌ی هم‌جوشی دو یا چند کرنل به تسریع ۱/۶ برابری دست یافتند. آن‌ها امکان هم‌جوشی اتوماتیک کرنل‌ها را براساس وابستگی داده‌ای، بهره‌وری منابع و درشت‌دانگی موازی‌سازی بررسی کردند. هدف آن‌ها این بوده که با استفاده از هم‌جوشی، محلی‌سازی کرنل افزایش یابد. مزیت روش آن‌ها این است که هم‌جوشی خودکار در طول تولید کد را ممکن ساخته و قابلیت خوانایی کد کودا منتشر شده پس از هم‌جوشی را حفظ می‌کند. آن‌ها این طرح را در یک کامپایلر مبدأ به مبدأ برای برنامه‌های پردازش تصویر به‌کار گرفتند.

وو و همکاران [۴] سعی کردند که توامان از مزایای هم‌جوشی و شکافت[۱۵] کرنل استفاده کنند. در فرآیند شکافت، کرنل به چند کرنل کوچکتر شکسته می‌شود تا به‌صورت چندجریانی فراخوانی شوند. هدف آن‌ها این بود که در حین چندجریانی، محاسبات یک کرنل با ارتباطات کرنل دیگر همپوشانی و پنهان شود. آن‌ها ایده‌ی خود را روی برنامه‌ی SELECT از محک TPC-H [۲۷] اجرا کردند و بهبود ۱۳ تا ۴۱ درصدی را در کارآیی کرنل به‌دست آوردند.

تابیک و همکاران [۲۶] با هدف بهبود کارآیی زیربرنامه‌ی جبری خطی پایه[۱۶]، عملیات برداری را هم‌جوشی کردند. نتایج آن‌ها نشان می‌دهد که این عمل باعث بهبود قابل‌توجه ۹۸ درصدی در کارآیی می‌شود. این بهبود به این دلیل بوده است که با هم‌جوشی کرنل نیاز به انتقال داده‌ها کمتری است. همچنین آن‌ها تاثیر استفاده از عملیات برداری هم‌جوشی‌شده در برنامه‌ی گرادیان دو مزدوج[۱۷] را بررسی کردند و در آزمایش‌ها بهبود کارآیی بین ۹ تا ۲۲ درصد را گزارش کردند.

وهیب و همکاران [۱۰] دو چالش برای هم‌جوشی کرنل ارائه کرده‌اند. چالش اول تعیین یک روش مقیاس‌پذیر برای یافتن بازآرایش (تنظیم مجدد) بهینه و هم‌جوشی کرنل‌ها و چالش دوم استفاده دقیق از امکانات و ساختار پردازنده‌ی گرافیکی بوده است. آن‌ها برای حل این دو چالش، هم‌جوشی کرنل را به‌عنوان یک مسأله‌ی بهینه‌سازی ترکیبی استفاده کرده و یک اکتشاف تقریبی برای جستجو در فضای هم‌جوشی احتمالی ارائه نموده‌اند. همچنین آن‌ها یک مکانیسم اشتراک پردازنده‌ی گرافیکی بین دو کرنل پیشنهاد کردند که منجر به افزایش اشغال و در نتیجه کاهش زمان اجرای یک مجموعه کاری شد.

خالد و همکاران [۲۸] تکنیک FusionCL را ارائه کردند که یک مکانیزم اشتراک‌گذاری پردازنده‌ی گرافیکی بین دو کرنل

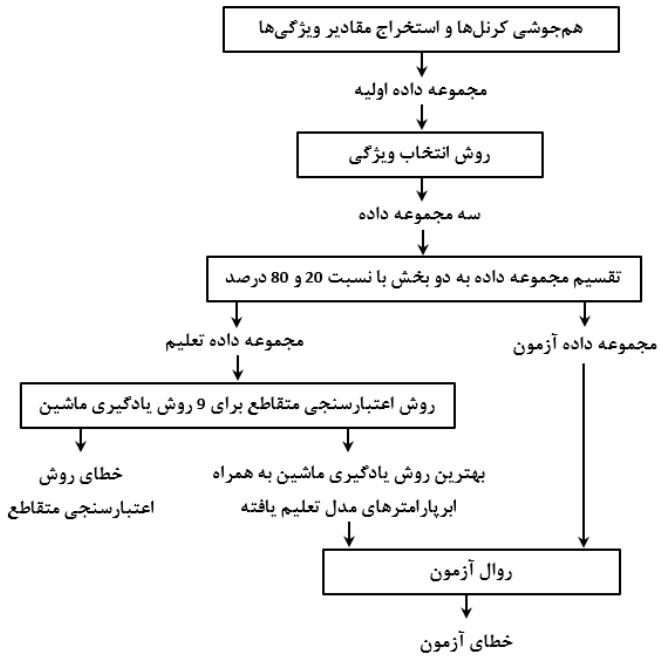
OpenCL است. FusionCL هر جفت کرنل (از مجموعه‌ی کارها) را که کاندیداهای مناسبی برای هم‌جوشی هستند را با استفاده از یک دسته‌بند مبتنی‌بر یادگیری ماشین شناسایی می‌کند. سپس از بین همه‌ی کاندیداها، یک جفت کرنل را برای هم‌جوشی انتخاب می‌کند که بتوانند بیشترین تسریع[۱۸] را ایجاد کنند. ارزیابی‌های آن‌ها نشان می‌دهند که FusionCL زمان اجرا را در مقایسه با سایر طرح‌های زمانبندی، بین ۲/۸۳ تا ۸ درصد کاهش می‌دهد.

لی و همکاران [۱۲] روش هم‌جوشی افقی را پیشنهاد کردند که مکمل روش‌های قبلی هم‌جوشی کرنل‌های کودا است. این روش نوعی هم‌جوشی درون بلوکی است که به نخ‌های عضو یک بلوک وظیفه می‌دهد که دستورالعمل‌های دو کرنل مختلف را اجرا کنند، با این تفاوت که در هم‌جوشی افقی، نخ‌ها دستورالعمل‌هایی را به‌صورت موازی اجرا می‌کنند که روی واحدهای تابعی گوناگونی اجرا شوند. به این ترتیب موازی‌سازی در سطح نخ[۱۹] افزایش می‌یابد. آن‌ها طرح خود که در یک کامپایلر مبدأ به مبدأ به نام HFUSE پیاده‌سازی کردند. در آزمایش‌های آن‌ها بهبود کارآیی بین ۲/۵ تا ۶۰/۸ درصد مشاهده شد.

در مواردی تکنیک هم‌جوشی کرنل روی برنامه‌های یادگیری ماشین اجرا شده است. سان و همکاران [۶] نشان دادند که در برنامه‌های یادگیری عمیق معمولا دو عملیات تنسوری (مثل کانولوشن) و ReLU به‌صورت متوالی فراخوانی می‌شوند. در بین این دو عملیات، انتقال داده به حافظه‌ی سراسری (برای ذخیره‌ی نتایج موقتی) باعث افت کارآیی می‌شود. بنابراین آن‌ها با هم‌جوشی این دو کرنل، کارآیی برنامه را از حیث زمان اجرا و انرژی مصرفی بهبود دادند. عطیف و همکاران [۲۹] چند کرنل را برای شبکه‌ی عصبی RBF هم‌جوشی کردند. آشاری و همکاران [۳۰] عملگرهای جبرخطی اولیه مثل ضرب ماتریس در بردار را با استفاده از تکنیک هم‌جوشی کرنل بهبود دادند. این عملگرها در کرنل‌های مرتبط با روش‌های یادگیری ماشین و کتابخانه‌های BLAS کاربرد دارند. رویکرد آن‌ها هزینه‌ی انتقال داده‌ها را به دلیل محلی‌سازی داده‌ها کاهش داده است و بستری برای تجمیع نتایج محلی فراهم کرده است. آن‌ها یک مدل تحلیلی ارائه کردند که براساس منابع موجود در دستگاه، تنظیمات نزدیک به بهینه را برای پارامترهای کرنل ارائه می‌کند. آن‌ها نشان دادند که برای نمونه‌های مختلف، سرعتی معادل ۲ تا ۶۷ برابر به‌دست آمده است. همچنین آن‌ها کارآمدی این طرح را روی یک الگوریتم یادگیری ماشین نشان دادند.

۴- طرح پیشنهادی

در این بخش روش‌های ایجاد و به‌کارگیری مدل کارآیی پیشنهادی ارائه خواهند شد. هدف ایجاد مدلی مبتنی‌بر یادگیری ماشین است که با استخراج ویژگی‌های دو کرنل اولیه، زمان اجرای کرنل‌های هم‌جوشی‌شده را پیش‌بینی کند. ما برای هریک از سه تکنیک درون نخی، درون بلوکی و بین بلوکی، سه مدل کارآیی مجزا ایجاد کردیم. خروجی هر مدل زمان اجراست که یک کمیت پیوسته است و بنابراین مسئله‌ی ما از نوع رگرسیون است. مراحل ایجاد مدل کارآیی به‌صورت گرافیکی در شکل ۳ نشان داده شده است. در ادامه‌ی این بخش، تمام مراحلی که در این شکل نمایش داده شده‌اند، تشریح می‌شوند.



شکل ۳: نمایش مراحل ایجاد مدل کارآیی پیشنهادی.



شکل ۴: نمایش مراحل به‌کارگیری مدل کارآیی برای پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی‌شده.

شکل ۴ روش به‌کارگیری مدل برای پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی‌شده را نمایش می‌دهد. برای هر دو کرنل ورودی ابتدا ویژگی‌های انتخاب شده برای هر مدل استخراج می‌شوند. همانطور که در ادامه تشریح خواهد شد، ویژگی‌های استفاده شده در سه مدل کارآیی با یکدیگر متفاوت هستند. بنابراین سه رکورد متفاوت به‌صورت بردارهای X1 و X2 و X3 ایجاد و به‌عنوان ورودی به سه مدل کارآیی داده می‌شوند. هر مدل زمان اجرای کرنل هم‌جوشی‌شده را ارائه می‌کند.

۴-۱- هم‌جوشی کرنل‌ها و استخراج مقادیر ویژگی‌ها

برای ساخت مدلی مبتنی‌بر یادگیری ماشین نیازمند ایجاد یک مجموعه داده هستیم. برای این منظور مجموعه‌ای از کرنل‌ها انتخاب شده‌اند. کرنل‌ها باتوجه به منابع سخت‌افزاری مورد استفاده در زمان اجرا، رفتارهای متفاوتی از خودشان نشان می‌دهند و عموما به دو دسته‌ی حافظه‌گرا[۲۰] و محاسبه‌گرا[۲۱] دسته‌بندی می‌شوند. کرنل‌هایی از هر دو دسته انتخاب شده‌اند، چراکه حافظه‌گرا یا محاسبه‌گرا بودن کرنل در هم‌جوشی تاثیرگذار است. معیار ما برای تعیین نوع حافظه‌گرا یا محاسبه‌گرا بودن کرنل، پژوهش شکفته و همکاران [۳۱] است. کرنل‌های استفاده شده در جدول ۱ نشان داده شده‌اند. این کرنل‌ها از مجموعه مثال‌های NVIDIA و محک رودینیا انتخاب شده‌اند.

کرنل‌های انتخاب شده را دو به دو با هم هم‌جوشی کردیم. برای هرجفت کرنل، هرسه تکنیک اجرا شده است. بنابراین تعداد هم‌جوشی‌های انجام شده برابر $3 \times \binom{8}{2} = 84$ است. برای هر کرنل

جدول ۱: کرنل‌های انتخاب شده برای ساخت مجموعه داده‌ها.

نام برنامه	نوع کرنل	تعداد خط کد	مجموعه محک
Hotspot	محاسبه‌گرا	۶۶	رودینیا
matrixMul	محاسبه‌گرا	۲۴	مثال‌های NVIDIA
Gussian	محاسبه‌گرا	۱۴	رودینیا
Srad	حافظه‌گرا	۳۷	رودینیا
vectorAdd	حافظه‌گرا	۳	مثال‌های NVIDIA
Transpose	حافظه‌گرا	۲۱	مثال‌های NVIDIA
BFS	حافظه‌گرا	۱۶	رودینیا
ScalarProd	حافظه‌گرا	۱۸	مثال‌های NVIDIA

باتوجه به اندازه‌ی داده‌ها و پیکربندی‌های مجاز، کرنل‌ها را اجرا کرده و پروفایلینگ برای تک تک آنها انجام شده است. در روش درون نخی، اندازه‌ی گرید در پیکربندی کرنل هم‌جوشی‌شده، برابر بیشینه‌ی اندازه‌ی گریدها درکرنل‌های اولیه است. اندازه‌ی بلوک نیز برابر بیشینه‌ی اندازه‌ی بلوک‌ها در کرنل‌های اولیه است. در روش درون بلوکی، اندازه‌ی گرید در پیکربندی کرنل هم‌جوشی‌شده برابر بیشینه‌ی اندازه‌ی گریدها درکرنل‌های اولیه است. اندازه‌ی بلوک نیز برابر مجموع اندازه‌ی بلوک‌ها در کرنل‌های اولیه است. البته باید توجه داشت که اندازه‌ی بلوک در کودا محدودیت‌هایی دارد، بنابراین درصورتی این نسخه ایجاد می‌شود که قابلیت هم‌جوشی برای آن وجود داشته باشد. در مدل کارآیی نیز یک فاز امکان‌سنجی برای هم‌جوشی تعبیه شده است. در روش بین بلوکی پیکربندی کرنل هم‌جوشی‌شده به این صورت است که اندازه‌ی گرید برابر مجموع اندازه‌ی گریدها درکرنل‌های اولیه بوده و اندازه‌ی بلوک نیز برابر بیشینه‌ی اندازه‌ی بلوک‌ها در کرنل‌های اولیه است. برای اطمینان از صحت به‌کارگیری تکنیک‌های هم‌جوشی کرنل، خروجی‌های کرنل‌های هم‌جوشی‌شده با خروجی‌های کرنل‌های اولیه تطبیق داده شده‌اند.

در قدم بعدی ویژگی‌های موردنظر که در ادامه معرفی می‌شوند، را از کرنل‌های اولیه و کرنل‌های هم‌جوشی شده استخراج کردیم. این ویژگی‌ها ازطریق پروفایلینگ هریک از کرنل‌ها و با ابزار nvprof به‌دست آمده است. برای ایجاد مدل با عملکردی قابل قبول، نیازمند مجموعه داده‌ای بزرگ و جامع هستیم. بنابراین این کرنل‌ها را با پیکربندی‌های مختلف راه‌اندازی و اجرا و به‌ازای هر یک اجرای کرنل‌های اولیه و هم‌جوشی‌شده، یک سطر از مجموعه داده ایجاد کردیم. تمام داده‌های مربوط به دو کرنل اولیه به‌عنوان ویژگی‌های ورودی و زمان اجراهای کرنل‌های هم‌جوشی‌شده، برچسب خروجی مجموعه داده محسوب شده‌اند. برای ثبت زمان اجرای کرنل‌ها، آن‌ها را ده بار اجرا کرده و پس از حذف نتایج پرت، از زمان‌های اجرا میانگین گرفتیم.

در ادامه داده‌ها را پیش‌پردازش کردیم. برخی داده‌ها کمی‌سازی شده و تبدیل به مقادیر عددی شدند. همچنین تعداد محدودی از

رکوردها با مقادیر ناقص و دو ویژگی با مقادیر ثابت حذف شدند. سپس نرمال‌سازی روی مجموعه داده انجام شده است.

جدول ۲: نتایج حاصل از اجرای دو روش انتخاب ویژگی.

روش هم‌جوشی	روش انتخاب ویژگی	امتیاز	تعداد ویژگی‌ها
درون نخی	حذف بازگشتی ویژگی	۰/۹۸۴	۸
	تعبیه شده	۰/۹۶۳	۱۲
درون بلوکی	حذف بازگشتی ویژگی	۰/۹۷۵	۱۲
	تعبیه شده	۰/۹۴۸	۱۷
بین بلوکی	حذف بازگشتی ویژگی	۰/۹۷	۱۱
	تعبیه شده	۰/۹۵۲	۱۲

۴-۲- اجرای روش انتخاب ویژگی

قبل از استفاده از الگوریتم‌های انتخاب ویژگی، برای کاهش ویژگی‌ها از روش فیلتر استفاه کردیم. ویژگی‌هایی که با هم همبستگی بالایی دارند، می‌توانند عملکرد کلی مدل یادگیری ماشین را کاهش دهند. بنابراین، ما تنها یکی از این ویژگی‌های هم‌بسته را در مجموعه داده حفظ کردیم. میزان همبستگی بین ویژگی‌های مجموعه داده را محاسبه کرده و از بین دو ویژگی که دارای همبستگی بیشتر از ۰/۷ داشتند، فقط یکی را در مجموعه داده حفظ کردیم.

در قدم بعدی دو روش انتخاب ویژگی حذف بازگشتی ویژگی[۲۲] و تعبیه‌شده[۲۳] را روی مجموعه داده اجرا کردیم. در روش حذف بازگشتی از الگوریتم رگرسیون جنگل تصادفی[۲۴] استفاده کردیم. بهترین نتایجی که به‌دست آمده است به‌صورت جدول ۲ است. بنابراین مجموعه داده مدل کارآیی برای روش درون نخی شامل ۸ ویژگی از هر کرنل، مجموعه داده مدل کارآیی برای روش درون بلوکی شامل ۱۲ ویژگی از هر کرنل و مجموعه داده مدل کارآیی برای روش بین بلوکی شامل ۱۱ ویژگی از هر کرنل است. تمام این ویژگی‌ها در جدول ۳ معرفی شده‌اند. شرح هریک از ویژگی‌ها را از مستندات پروفایلر NVIDIA [۳۲] استخراج کردیم. نکته‌ای که درخصوص دو ویژگی n_b و n_t باید ذکر شود این است که این دو به‌ترتیب تعداد بلوک‌ها در گرید و تعداد نخ‌ها در یک بلوک هستند. ممکن است کرنل‌ها به‌صورت دوبعدی و یا سه‌بعدی پیکربندی شوند، با این حال در مدل به‌صورت یک بعدی درنظر گرفته شده‌اند. باید توجه شود که این تکنیک در مدل‌های کارآیی مرسوم است و قبلا در مدل تحلیلی Many-BSP [۳۳] به‌کارگرفته شده است، چراکه پیکربندی‌های دوبعدی یا سه‌بعدی تنها برای نگاشت بهتر نخ‌ها به داده‌ها مورد استفاده قرار می‌گیرند و پس از تبدیل کدهای کودا به کدهای PTX، تاثیری در اجرای کرنل نخواهند داشت [۳۴].

جدول ۳: معرفی ویژگی‌های بکارگرفته شده در مجموعه داده‌ها برای هر سه مدل کارآیی.

نام ویژگی انتخاب شده	روش هم‌جوشی کرنل‌ها			توضیحات
	درون نخی	درون بلوکی	بین بلوکی	
n_b	*	*	*	اندازه‌ی گرید (تعداد بلوک‌ها)
n_t	*	*	*	اندازه‌ی بلوک (تعداد نخ‌ها در هر بلوک)
execution_time	*	*	*	زمان اجرای کرنل اولیه
stall_exec_dependency	*	*	*	درصد توقف‌های اجرای دستورالعمل‌ها که به دلیل عدم وجود ورودی (عملوند) مورد نیاز دستورالعمل رخ می‌دهند
achieved_occupancy	*	*	*	مقدار اشغال به‌دست آمده درطول اجرای کرنل است. معیار اشغال نسبتی از زمان است که هسته‌های پردازشی درحال اجرا بوده‌اند.
stall_not_selected	*			درصد توقف‌هایی است که به دلیل عدم انتخاب وارپ رخ داده است
warp_execution_efficiency	*	*		نسبت میانگین نخ‌های فعال در هر وارپ به حداکثر تعداد نخ‌های پشتیبانی‌شده در هر وارپ در یک چندپردازنده‌ی جریانی
inst_issued	*	*	*	تعداد دستورالعمل‌هایی که با هم صادر می‌شوند
stall_constant_memory_dependency		*		درصد توقف‌هایی که به دلیل شکست حافظه نهان ثابت رخ می‌دهد
shared_store_transactions		*	*	تعداد دستورالعمل‌های ذخیره‌سازی در حافظه‌ی اشتراکی
tex_cache_hit_rate		*	*	نرخ موفقیت حافظه‌ی نهان بافت
branch_efficiency		*	*	نسبت انشعابات غیر واگرا به کل انشعابات
gld_efficiency		*		نسبت گذردهی درخواست‌های حافظه‌ی سراسری به گذردهی درخواست‌های حافظه‌ی سراسری مورد نیاز
gld_transactions_per_request			*	میانگین تعداد تراکنش‌های بارگذاری حافظه‌ی سراسری انجام شده برای هر بارگذاری حافظه سراسری
gld_throughput			*	گذردهی بارگذاری حافظه‌ی سراسری

آن چیزی که در اینجا مهم است تعداد نخ‌ها و تعداد بلوک‌هاست. همچنین تعداد رکوردهای نهایی در مجموعه داده نیز ۲۴۸۱۸ سطر است.

۴-۳- اعتبارسنجی متقاطع برای نه روش یادگیری ماشین

نه روش یادگیری ماشین را برای پیش‌بینی زمان اجرای کرنل هم‌جوشی‌شده مورد آزمایش قرار دادیم. این نه روش عبارتند از: رگرسیون خطی، رگرسیون لاسو، شبکه‌ی الاستیک، جنگل تصادفی، k-نزدیکترین همسایه، درخت تصمیم، ماشین بردار پشتیبان، آدابوستینگ و گرادیان بوستینگ. برای ارزیابی بهتر از روش اعتبارسنجی متقاطع k-دسته‌ای[۲۵] استفاده کردیم. در این روش مجموعه داده به‌طور تصادفی به k زیرنمونه تقسیم می‌شود که در هر مرحله از فرایند اعتبارسنجی متقابل، k-1 زیرنمونه داده‌های تعلیم و یک زیرنمونه داده‌های آزمون را تشکیل خواهند داد. قبلا نشان داده شد که برای مسائل پیش‌بینی بهترین مقدار برای k برابر ۱۰ است [۳۵] و ما در اینجا از همین مقدار استفاده کردیم.

خروجی روال اعتبارسنجی متقاطع k-دسته‌ای، میزان خطای هریک از نه روش یادگیری ماشین است. براساس این معیار ارزیابی، بهترین روش انتخاب می‌شود و در زمان به‌کارگیری مدل، از آن استفاده می‌شود. ضمن اینکه ابرپارامترهای به‌دست آمده از بهترین روش نیز ثبت شده و در هنگام استفاده از مدل مورد استفاده قرار می‌گیرد.

۵- ارزیابی مدل پیشنهادی

برای تعیین میزان دقت مدل کارآیی آزمایش‌هایی انجام دادیم. در این بخش نتایج حاصل از آزمایش‌ها گزارش خواهند شد.

۵-۱- روش‌شناسی آزمایش‌ها

مدل‌های مبتنی‌بر یادگیری ماشین برای هر دستگاه به‌طور مجزا باید ایجاد شوند. دستگاهی که مدل روی آن ایجاد و آزمایش‌ها روی آن انجام شده‌اند، در جدول ۴ معرفی شده است. معماری این دستگاه از نوع پاسکال است. کرنل‌ها در کودای نسخه‌ی ۸ و در محیط اوبونتو ۱۶/۰۴ هم‌جوشی شده‌اند و مقادیر ویژگی‌های آن استخراج شده است. پس از آماده‌سازی مجموعه داده، روش‌های انتخاب ویژگی و روش‌های یادگیری ماشین در پایتون پیاده‌سازی شده‌اند.

چند معیار ارزیابی برای آزمایش‌ها استفاده شده است. قبل از گزارش نتایج، این معیارها را معرفی می‌کنیم. اولین معیار، منفی میانگین مربعات خطا است که برای بررسی خطای روش‌های رگرسیون در پایتون و از کتابخانه‌ی Scikit-Learn استفاده می‌شود. در آزمایش‌های ما این معیار برای ارزیابی روش اعتبارسنجی متقاطع و شناسایی بهترین روش یادگیری ماشین برای استفاده در مدل کارآیی استفاده شده است. معادله‌ی (۱) نحوه‌ی محاسبه‌ی این معیار ارزیابی را نشان می‌دهد. در این معادله، n تعداد نمونه‌های آزمایش شده، Y_i برچسب خروجی اندازه‌گیری شده برای رکورد i ام از مجموعه داده (برحسب میکروثانیه) و $\hat{Y}_i$ مقدار پیش‌بینی مدل برای این رکورد (برحسب میکروثانیه) است. معیار ارزیابی میانگین مربعات خطا (MSE[۲۶]) معادل مثبت این پارامتر است.

$$neg_mean_squared_error = -\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 = -MSE \quad (۱)$$

دیگر معیار ارزیابی، درصد خطای پیش‌بینی زمان اجرای کرنل هم‌جوشی‌شده است. برای هر رکورد از مجموعه داده آزمون، میزان خطا به‌صورت معادله‌ی (۲) محاسبه می‌شود. از بین تمام رکوردهایی که در آزمون استفاده می‌شوند، حداقل، حداکثر و میانگین خطا ارائه خواهند شد.

$$error_i\ (\%) = \frac{|Y_i - \hat{Y}_i|}{Y_i} \times 100 \quad (۲)$$

معیار ارزیابی دیگر ضریب تعیین R^2 یا تابع امتیاز رگرسیون است. این پارامتر یک معیار ارزیابی آماری است که نشان‌دهنده‌ی نسبت واریانس در متغیر وابسته است که توسط متغیرهای مستقل در یک مدل رگرسیون توضیح داده شده است. این معیار نشان می‌دهد که رگرسیون تا چه اندازه توانسته است نقاط داده‌ی واقعی

جدول ۴: مشخصات دستگاه آزمایش‌ها.

Video Card	GeForce GTX 1070
GPU Architecture	Pascal
GPU Model	GP104
CUDA Capability (CC)	6.1
# CUDA Cores	1920
SM# × Cores/SM	128 × 15
Number of 32-bit registers per SM	64 K
Shared Memory Size	48 KB
L1 Cache Size	48 KB
L2 Cache Size	2 MB
Size of Global Memory	8 GB
Maximum number of threads per block	1024
Maximum number of threads per SM	2048
CUDA Version	8
CPU	Intel Corei7
OS	Ubuntu 16.04

را تقریب بزند. مقادیر این معیار از ۰ تا ۱ متغیر است و هرچقدر به ۱ نزدیکتر باشد، نشان می‌دهد که رگرسیون تقریب بهتری از تابع هدف ارائه کرده است. مقدار این معیار با استفاده از معادله‌ی (۳) محاسبه می‌شود. در این معادله $\bar{Y}$ ازطریق معادله‌ی (۴) به‌دست می‌آید. مابقی پارامترها مشابه معادلات قبلی است.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \quad (۳)$$

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i \quad (۴)$$

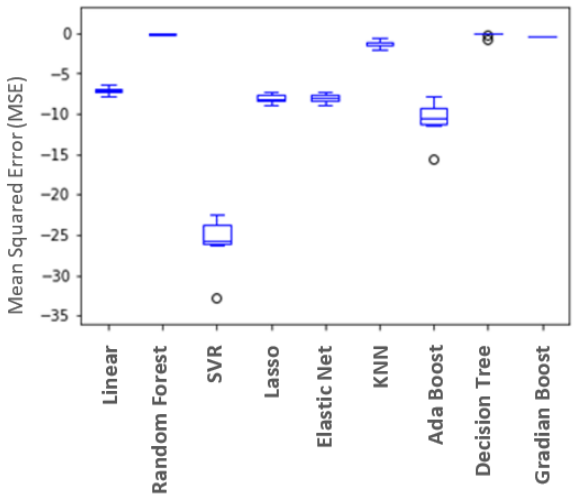
۵-۲- آزمایش اول: نتایج اعتبارسنجی متقاطع

برای نه روش یادگیری ماشین روش اعتبارسنجی متقاطع را فراخوانی کردیم. مقدار معیار ارزیابی منفی میانگین مربعات خطا برای هر روش در جدول ۵ گزارش شده است. شکل ۵ این نتایج را به‌صورت مقایسه‌ای فقط برای حالت درون نخی نشان می‌دهد.

همانطور که در نتایج مشخص است، برای هرسه تکنیک هم‌جوشی، روش جنگل تصادفی بهترین نتایج را ارائه کرده است. بنابراین برای ساخت هرسه مدل کارآیی از این روش استفاده می‌شود. در روش اعتبارسنجی متقاطع ابرپارامترهای بهترین مدل تعلیم‌یافته ذخیره می‌شوند تا برای ساخت مدل از آن استفاده شود. این مقادیر به‌صورت جدول ۶ به‌دست آمده است.

جدول ۵: منفی میانگین مربعات خطا برای نه روش یادگیری ماشین.

روش یادگیری ماشین	منفی میانگین مربعات خطا		
	درون نخی	درون بلوکی	بین بلوکی
رگرسیون خطی	-۷/۰۹۷	-۲۵/۵۷۱	-۴/۵۷۵
جنگل تصادفی	-۰/۱۱۲	-۰/۲۲۲	-۰/۰۹۴
ماشین بردار پشتیبان	-۲۵/۵۵۹	-۶۰/۴۵۹	-۲۰/۷۶۱
رگرسیون لاسو	-۸/۱۲۷	-۲۶/۲۷۱	-۵/۵۴۶
شبکه‌ی الاستیک	-۸/۰۶۴	-۲۶/۲۵۹	-۵/۱۰۳
k-نزدیکترین همسایه	-۱/۳۴۸	-۲/۱۰۳	-۰/۱۰۴
آدابوستینگ	-۱۳/۰۵۵	-۱۴/۱۲۵	-۶/۸۵۲
درخت تصمیم	-۰/۱۵	-۰/۳۷۴	-۰/۱۳۴
گرادیان بوستینگ	-۰/۴۰۱	-۰/۴۸۹	-۰/۳۱



شکل ۵: مقایسه‌ی میانگین مربعات خطا برای نه روش یادگیری ماشین برای تکنیک درون نخی

جدول ۶: مقادیر ابرپارامترهای ذخیره شده در روش جنگل تصادفی برای ساخت مدل کارآیی.

پارامتر	بهترین مقدار
n_estimate	۴۰۰
max_features	sqrt
max_depth	۱۱۰
min_samples_split	۲
min_samples_leaf	۴
bootstrap	false

۵-۳- آزمایش دوم: ارزیابی دقت مدل

پس از ایجاد مدل، باید از آن برای پیش‌بینی کرنل‌های ناشناخته استفاده شود. همان‌طور که در شکل ۳ نشان داده شد، ۲۰ درصد داده‌ها به‌عنوان مجموعه داده‌های آزمون کنار گذاشته شدند. از این مجموعه برای ارزیابی دقت مدل استفاده می‌شود. این داده‌ها در فرآیند تعلیم شرکت نداشتند و بنابراین این آزمون به‌خوبی میزان دقت مدل کارآیی را گزارش می‌کند. نتایج حاصل از روال آزمون، در جدول ۷ نشان داده شده است. در این جدول معیارهای ارزیابی که در بخش ۵-۱ معرفی شده‌اند، برای روش جنگل تصادفی گزارش شده‌اند. ابرپارامترهای این مدل تعلیم‌یافته نیز مطابق جدول ۶ است که در مرحله‌ی تعلیم به‌دست آمده است. نتایج نشان می‌دهند که به‌طور میانگین درصد خطای مدل کمتر از ۵ درصد است.

جدول ۷: نمایش دقت مدل پیشنهادی در پیش‌بینی زمان اجرای کرنل‌های هم‌جوشی شده در هریک از سه تکنیک.

تکنیک	حداقل درصد خطا	حداکثر درصد خطا	میانگین درصد خطا	MSE	R^2
درون نخی	۰/۰۱۲	۱۶/۸۷	۳/۲۴	۰/۰۳۳	۰/۹۹
درون بلوکی	۰/۰۱۳۱	۱۸/۹۹	۴/۳۱	۰/۱۶۸	۰/۹۹۷
بین بلوکی	۰/۰۰۹۴	۱۹/۰۲۵	۳/۶۰۷	۰/۰۴۹	۰/۹۹

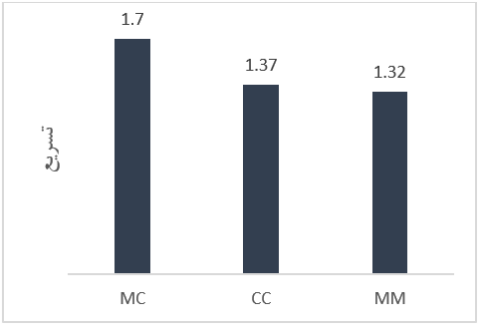
جدول ۸: نمایش پراکندگی خطای مدل در هرسه تکنیک.

تکنیک	خطای کمتر از ۵ درصد	خطای بین ۵ تا ۱۰ درصد	خطای بیش از ۱۰ درصد
درون نخی	۸۰/۲۳	۱۷/۶۸	۲/۰۹
درون بلوکی	۷۰/۲۵	۱۵/۹۴	۱۳/۸۱
بین بلوکی	۷۵/۸	۱۳/۷۹	۱۰/۴۱

حداکثر درصد خطای مدل در جدول ۷ برابر ۱۹ درصد است. در جدول ۸ میزان پراکندگی خطاها در هرسه روش نشان داده شده‌اند. نتایج این جدول نشان می‌دهند که درصد مواردی که خطای پیش‌بینی بالای ۱۰ درصد دارند، نسبتا کم هستند.

۵-۴- آزمایش سوم: ارزیابی تسریع حاصل از هم‌جوشی

کرنل‌های جدول ۱ با سه ترکیب با هم هم‌جوشی شده‌اند. هم‌جوشی دو کرنل حافظه‌گرا را به‌صورت MM، هم‌جوشی دو کرنل محاسبه‌گرا را به‌صورت CC و هم‌جوشی یک کرنل حافظه‌گرا با یک کرنل محاسبه‌گرا را با MC نمایش می‌دهیم. در شکل ۶ میانگین تسریع حاصل از هم‌جوشی هرکدام از ترکیب کرنل‌های حافظه‌گرا و محاسبه‌گرا نمایش داده شده است. این مقادیر از تقسیم مجموع زمان اجرای دو کرنل اولیه، بر زمان اجرای کرنل هم‌جوشی‌شده به‌دست آمده است. باتوجه به اینکه آزمایش‌های ما در این پژوهش روی پردازنده‌های گرافیکی گسسته انجام شده است، به هرحال چه هم‌جوشی انجام شود و یا نشود، داده‌ها باید بین حافظه‌ی اصلی سمت میزبان و حافظه‌ی سراسری سمت دستگاه رد و بدل شوند. این انتقال هم برای کرنل‌های اولیه است و هم برای کرنل‌های هم‌جوشی‌شده. همچنین اندازه‌ی داده‌های منتقل شده هم برابر هستند. نتیجه اینکه سربار انتقال داده‌ها در هردو کرنل اولیه و هم‌جوشی‌شده وجود دارد و در عملکرد کلی پیاده‌سازی‌های ما تاثیری ندارند. بنابراین در محاسبه‌ی تسریع از سربار انتقال داده‌ها صرفنظر شده است.



شکل ۶: مقایسه‌ی تسریع به‌دست آمده در هم‌جوشی کرنل‌های حافظه‌گرا و محاسبه‌گرا.

مقدار تسریع برای هم‌جوشی تمامی کرنل‌های موجود در جدول ۱ محاسبه شده و پس از اندازه‌گیری مقدار تسریع تک تک موارد، میانگین آن‌ها محاسبه و در این شکل گزارش شده است.

۶- بحث

نتایجی که در بخش قبلی گزارش شده‌اند نشان می‌دهند که مدل پیشنهادی به‌طور میانگین با خطای ۴/۳۱ درصد می‌تواند زمان اجرای کرنل‌های هم‌جوشی‌شده را پیش‌بینی نماید. در بدترین حالت این خطا بین ۱۶ تا ۱۹ درصد است. البته در جدول ۸ نشان دادیم که فراوانی بدترین حالت در آزمایش‌های ما خیلی کم بوده است و همچنین در بهترین حالت، بدترین حالت و حالت میانگین درصد خطاهای سه تکنیک هم‌جوشی یکنواخت و متناسب هستند. ازطرفی باید توجه شود که هدف از پیش‌بینی زمان اجرا، راهنمایی برنامه‌نویس در انتخاب بهترین روش برای هم‌جوشی کرنل‌هاست. بنابراین این درصد خطا در بدترین حالت قابل تحمل است. نکته‌ی دیگری که لازم است به آن توجه شود این است که اساسا تخمین زمان اجرای کرنل‌های حافظه‌گرا کمی مشکل است. تراکنش‌های حافظه با شرایط یکنواخت و قابل انتظاری سرویس‌دهی نمی‌شوند و گلوگاه‌های حافظه تخمین مدت زمان سرویس‌دهی آن را مشکل می‌کند. بنابراین اندازه‌گیری و پیش‌بینی زمان اجرای این کرنل‌ها بسیار مستعد خطاست [۳۳]. یکی از دلایل خطای حداکثر ۱۹ درصدی در مدل پیشنهادی این است که در آزمایش‌های ما از کرنل‌های حافظه‌گرا نیز استفاده شده است. به همین دلیل در برخی پژوهش‌ها [۳۶] برای کرنل‌های حافظه‌گرا و محاسبه‌گرا مدل‌های کارآیی مجزایی ساخته می‌شود.

باتوجه به نتایج گزارش‌شده در جداول ۵ و ۷ دقت مدل پیشنهادی در حالت درون بلوکی نسبت به درون نخی کاهش یافته است. دلیل این تغییرات به سیاست‌های زمانبندی در پردازنده‌ی گرافیکی برمی‌گردد. دو سطح زمانبندی در پردازنده‌ی گرافیکی وجود دارد. زمانبندی سطح یک که در گیگاترد انجام می‌شود و در آن بلوک‌ها زمانبندی می‌شوند. زمانبند سطح دو که زمانبند وارپ نام دارد، وارپ‌های درون یک بلوک را زمانبندی می‌کند. اساس زمانبندی در گیگاترد به این صورت است که براساس منابع درون

چندپردازنده‌های جریانی، بلوک‌ها را بین آن‌ها توزیع می‌کند. ممکن است هم‌زمان یک یا چند بلوک به یک چندپردازنده‌ی جریانی تخصیص یابد. ازطرفی زمانبندی در داخل هر چندپردازنده‌ی جریانی توسط زمانبند وارپ و به‌صورت ریزدانه[۲۷] انجام می‌شود. در اینجا به‌صورت نوبتی یک دستورالعمل از هر وارپ عضو یک بلوک اجرا می‌شود. بنابراین استفاده از دو روش درون بلوکی و درون نخی باعث ایجاد تغییرات زیادی در زمانبندی و در ترتیب اجرای نخ‌های کرنل هم‌جوشی شده می‌شوند. هرکدام از این دو سیاست زمانبندی در شرایطی می‌توانند باعث بهبود یا افت کارآیی شوند. بنابراین تغییرات زیاد نتایج در این دو حالت به این دلیل است.

در تمام روش‌های یادگیری ماشین دقت محصول تولید مناسب مجموعه داده‌ها و الگوریتم یادگیری ماشین است. ابتدا باید مجموعه داده با کیفیت مناسب تولید شود. برای این منظور بستر آزمایش‌ها و روش اندازه‌گیری باید به‌صورت مناسب آماده و به‌کارگیری شود. در قدم بعدی روش‌های یادگیری ماشین مختلف روی دیتاست پیاده‌سازی و اجرا می‌شوند تا ببینیم کارآیی کدام روش بهتر از بقیه است. بسته به نوع داده‌ها و روابطی که بین ویژگی‌های ورودی و برچسب خروجی وجود دارد، یکی از روشها می‌تواند از بقیه بهتر باشد. بنابراین در قدم اول کیفیت داده‌ها بسیار تعیین کننده است و در قدم دوم الگوریتم یادگیری ماشین انتخابی اهمیت دارد. در جدول ۵ نتایج نه روش یادگیری ماشین نمایش داده شده‌اند. تنها روش ماشین بردار پشتیبان بسیار نامناسب است و هشت روش دیگر نسبتا نتایج خوبی ارائه کرده‌اند. نتیجه‌گیری می‌شود که کیفیت داده‌های تولید شده بسیار مناسب است. ضمن اینکه روش جنگل تصادفی نیز به درستی انتخاب شده و کارآیی قابل قبولی ارائه کرده است.

ما از روش‌های یادگیری ماشین برای طراحی مدل کارآیی پیشنهادی استفاده کردیم. مزیت آن این است که این روش‌ها پارامترهای ناشناخته‌ی پردازنده‌ی گرافیکی را به‌خوبی عمومی‌سازی[۲۸] می‌کنند. متاسفانه در بسیاری موارد اطلاعات ناچیزی از نحوه‌ی عملکرد پردازنده‌ی گرافیکی منتشر شده است. این امر مدل‌سازی این دستگاه را بسیار مشکل کرده است. استفاده از مدل‌های مبتنی‌بر یادگیری ماشین در مقایسه با مدل‌های تحلیلی این مزیت را دارد که برنامه‌نویس را از دانستن این جزئیات بی‌نیاز می‌کند. ازطرف نقطه ضعف مدل‌های مبتنی‌بر یادگیری ماشین این است که برای هر دستگاه باید مجددا ساخته شود و به‌عبارت دیگر قابلیت‌حمل پایینی دارد. درعوض، مدل‌های تحلیلی باوجودیکه ساخت آن‌ها بسیار سخت است، اما یکبار ساخته می‌شوند و برای همه‌ی دستگاه‌ها مورد استفاده قرار می‌گیرند و قابلیت‌حمل بالایی دارند.

۷- نتیجه‌گیری و کارهای آینده

در این مقاله یک مدل کارآیی مبتنی‌بر روش‌های یادگیری ماشین ایجاد شده است. هدف از ایجاد این مدل این است که بدون اینکه دو کرنل ورودی هم‌جوشی شوند، تشخیص دهد که از بین سه تکنیک هم‌جوشی درون نخی، درون بلوکی و بین بلوکی، کدامیک برای این دو کرنل مناسب‌تر است. مدل پیشنهادی زمان اجرای کرنل‌های هم‌جوشی‌شده را تنها با اطلاعاتی که از کرنل‌های اولیه استخراج می‌کند، تخمین می‌زند. ما برای هر تکنیک هم‌جوشی، مدل جداگانه‌ای توسعه دادیم. برای پیاده‌سازی مدل کارآیی، ابتدا برای هر تکنیک، یک مجموعه داده مجزا ایجاد کردیم. سپس نه روش یادگیری ماشین را برای تخمین زمان اجرا به‌کار گرفتیم. نتایج حاصل از روال اعتبارسنجی متقاطع نشان دادند که روش جنگل تصادفی بهترین کارآیی را برای هرسه تکنیک هم‌جوشی ارائه کرده است. سپس مدل ساخته‌شده را با استفاده از داده‌های ناشناخته مورد ارزیابی قرار دادیم که نتایج نشان دادند که خطای مدل در حالت میانگین کمتر از ۵ درصد بوده است. محدودیتی که مدل پیشنهادی دارد قابلیت‌حمل پایین آن است که مربوط به ذات روش‌های مبتنی‌بر یادگیری ماشین است. درصورتیکه بخواهیم مدل را روی دیگر دستگاه‌ها پیاده‌سازی کنیم، تمام فرآیند ساخت مدل باید مجددا انجام شود.

برای کارهای آینده دو پیشنهاد ارائه می‌شود. پیشنهاد اول این است که برای ترکیب‌های مختلفی از کرنل‌های حافظه‌گرا و محاسبه‌گرا مدل‌های کارآیی مجزایی ساخته شود. به‌عبارت دیگر، باتوجه به سه تکنیک هم‌جوشی، در مجموع نه مدل کارآیی مجزا ایجاد شود و بسته به نوع دو کرنل ورودی، سه تا از این مدل‌ها فراخوانی شوند. همچنین پیشنهاد دیگر این است که از روش‌های دیگر مبتنی‌بر یادگیری ماشین مثل یادگیری عمیق و ماشین یادگیری افراطی برای پیش‌بینی زمان اجرا استفاده شود.

مراجع

[1] D. Q. Ren, "Algorithm level power efficiency optimization for CPU–GPU processing element in data intensive SIMD/SPMD computing," *Journal of Parallel and Distributed Computing*, vol. 71, no. 2, pp. 245-253, 2011, doi: 10.1016/j.jpdc.2010.10.007.

[2] Y. Oh et al., "Adaptive cooperation of prefetching and warp scheduling on GPUs," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 609-616, Apr. 2019, doi: 10.1109/TC.2018.2878671.

[3] T. L. Falch and A. C. Elster, "Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications,"

Concurrency and Computation: Practice and Experience, vol. 29, no. 8, p. e4029, 2017, doi: 10.1002/cpe.4029.

[4] H. Wu, G. Diamos, J. Wang, S. Cadambi, S. Yalamanchili, and S. Chakradhar, "Optimizing data warehousing applications for GPUs using kernel fusion/fission," *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, Shanghai, China, 2012, pp. 2433-2442, doi: 10.1109/IPDPSW.2012.300.

[5] M. Korch and T. Werner, "Exploiting limited access distance for kernel fusion across the stages of explicit one-step methods on GPUs," *2018 30th International Symposium on Computer Architecture and High Performance Computing*, Lyon, France, 2018, pp. 148-157, doi: 10.1109/CAHPC.2018.8645892.

[6] W. Sun, A. Li, S. Stuijk, and H. Corporaal, "How much can we gain from tensor kernel fusion on GPUs?," *IEEE Access*, vol. 12, pp. 126135–126144, 2024, doi: 10.1109/ACCESS.2024.3411473.

[7] P. Hijma, S. Heldens, A. Sclocco, B. Van Werkhoven, and H. E. Bal, "Optimization techniques for GPU programming," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–81, 2023.

[8] J. Filipovič et al., "Optimizing CUDA code by kernel fusion: application on BLAS," *Journal of Supercomputing*, vol. 71, pp. 3934–3957, 2015, doi: 10.1007/s11227-015-1483-z.

[9] H. Zhao et al., "Adaptive kernel fusion for improving the GPU utilization while ensuring QoS," *IEEE Transactions on Computers*, vol. 74, no. 2, pp. 386–400, Feb. 2025, doi: 10.1109/TC.2024.3477995.

[10] M. Wahib and N. Maruyama, "Scalable kernel fusion for memory-bound GPU applications," *SC 2014: Proceedings of the International Conference for High Performance Computing*, Networking, *Storage and Analysis*, New Orleans, LA, USA, 2014, pp. 191–202, doi: 10.1109/SC.2014.21.

[11] G. Wang, "Coordinate strip-mining and kernel fusion to lower power consumption on GPU," *2011 Design, Automation & Test in Europe*, *Grenoble*, France, 2011, pp. 1–4, doi: 10.1109/DATE.2011.5763317.

[12] A. Li, B. Zheng, G. Pekhimenko and F. Long, "Automatic Horizontal Fusion for GPU Kernels," *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Seoul, Korea, Republic of, 2022, pp. 14-27, doi: 10.1109/CGO53902.2022.9741270.

[13] J. Fousek, J. Filipovič, and M. Madzin, "Automatic fusions of CUDA-GPU kernels for parallel map," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, pp. 98–99, 2011, doi: 10.1145/2082156.2082183.

[14] B. Qiao et al., "Automatic kernel fusion for image processing DSLs," *in Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*, May 2018, pp. 76–85, doi: 10.1145/3207719.3207723.

[15] J. Fukuhara and M. Takimoto, "Automated kernel fusion for GPU based on code motion," *in Proceedings of the 23rd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, Jun. 2022, pp. 151–161, doi: 10.1145/3519941.3535078.

[16] N. D. Gai, "Highly efficient and accurate deep learning–based classification of MRI contrast on a CPU and GPU," *Journal of Digital Imaging*, vol. 35, no. 3, pp. 482–495, 2022, doi: 10.1007/s10278-022-00583-1.

[17] J. Lin, J. Liu, E. F. Y. Young, and M. D. F. Wong, "GAMER: GPU-Accelerated Maze Routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 583–593, Feb. 2023, doi: 10.1109/TCAD.2022.3184281.

[18] A. Riahi, A. Savadi, and M. Naghibzadeh, "Comparison of analytical and ML-based models for predicting CPU–GPU data transfer time," *Computing*, vol. 102, pp. 2099–2116, 2020, doi: 10.1007/s00607-019-00780-x.

[19] M. Fang, J. Fang, W. Zhang, H. Zhou, J. Liao, and Y. Wang, "Benchmarking the GPU memory at the warp level," Parallel Computing, vol. 71, pp. 23–41, 2018, doi: 10.1016/j.parco.2017.11.003.

[20] D.-H. Kim, "Evaluation of the performance of GPU global memory coalescing," *Evaluation*, vol. 4, no. 4, pp. 1–5, 2017.

[21] NVIDIA C. "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", NVIDIA Corp, 2009.

[22] NVIDIA C. "Whitepaper NVIDIA GeForce GTX 1080", NVIDIA Corp, 2016.

[23] NVIDIA C. "Whitepaper NVIDIA TESLA V100 GPU ARCHITECTURE", NVIDIA Corp, 2017.

[24] Huang, Jen-Cheng, et al. "GPUMech: GPU performance modeling technique based on interval analysis.", in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, *IEEE*, pp. 268-279, 2014, doi: 10.1109/MICRO.2014.59.

[25] NVIDIA C. "CUDA C Programming Guide, Version 12.9", NVIDIA Corporation, 2025.

[26] S. Tabik, G. Ortega, and E. M. Garzón, "Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study," *Journal of Supercomputing*, vol. 70, pp. 577–587, 2014, doi: 10.1007/s11227-014-1102-4.

[27] T. P. C. Benchmark and H. T. M., "Standard Specification Revision 1.3.0," *Transaction Processing Performance Council (TPC)*, 1999.

[28] Y. N. Khalid et al., "FusionCL: a machine-learning based approach for OpenCL kernel fusion to increase system performance," *Computing*, vol. 103, pp. 2171–2202, 2021, doi: 10.1007/s00607-021-00958-2.

[29] S. M. Atif et al., "Multi-Kernel Fusion for RBF Neural Networks," *Neural Processing Letters*, vol. 55, pp. 1045–1069, 2023, doi: 10.1007/s11063-022-10925-3.

[30] A. Ashari et al., "On optimizing machine learning workloads via kernel fusion," *ACM SIGPLAN Notices*, vol. 50, no. 8, pp. 173–182, 2015.

[31] S. K. Shekofteh, H. Noori, M. Naghibzadeh, H. S. Yazdi, and H. Fröning, "Metric selection for GPU kernel classification," ACM Transactions on Architecture and Code Optimization, vol. 15, no. 4, pp. 1–27, 2019, doi: 10.1145/3295690.

[32] NVIDIA C. "Profiler Release 12.9", NVIDIA, 2025.

[33] A. Riahi, A. Savadi, and M. Naghibzadeh, "Many-BSP: an analytical performance model for CUDA kernels," Computing, vol. 106, no. 5, pp. 1519–1555, 2024, doi: 10.1007/s00607-023-01255-w.

[34] NVIDIA C. "PARALLEL THREAD EXECUTION ISA v6.5", NVIDIA, 2019.

[35] M. A. Hall, *Practical Machine Learning Tools and Techniques*, 3rd ed., Morgan Kauffman, 2011.

[36] A. Riahi, A. Savadi, and M. Naghibzadeh, "Performance prediction of regular CUDA programs by machine learning methods," Soft Computing Journal, 2024, doi: 10.22052/scj.2024.252862.1145.

پاورقی‌ها:

[1] Host

[2] Device

[3] Kernel

[4] Single Program Multiple Data (SPMD)

[5] Tuning

[6] Kernel Fusion
[7] Source-to-source Compilers
[8] Integrated GPUs
[9] Discrete GPUs
[10] Core
[11] Streaming Multi-processor
[12] Texture cache
[13] Independent Thread Scheduling
[14] Single Instruction Multiple Data (SIMD)
[15] Fission
[16] Basic Linear Algebra Subprogram (BLAS)
[17] Biconjugate Gradient (BCG)
[18] Speedup
[19] Thread-level Parallelism (TLP)
[20] Memory-bound
[21] Computation-bound
[22] Recursive Feature Elimination
[23] Embedded
[24] Random Forest Regression
[25] K-fold Cross Validation
[26] Mean Squared Error (MSE)
[27] Fine-grained
[28] Generalization