دوره پانزدهم، شماره پاییز و زمستان ۱۴۰۱

مجله فناوری اطلاعات در طراحی مهندسی

Information Technology in Engineering Design

http://ited.sinaweb.net

LEAM: الگوریتم ضرب تقریبی با خطای کم

سعیده جهانشاهی[۱] امیر صباغ ملاحسینی*[۲] آزاده سادات عمرانی زرندی [۳]

(۱) گروه مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان، ایران

(۲) گروه مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان، ایران*

(۳) گروه مهندسی کامپیوتر، دانشگاه شهید باهنر کرمان، کرمان، ایران

(تاریخ دریافت :۱۴۰۲/۰۳/۰۷ تاریخ پذیرش: ۱۴۰۲/۰۴/۲۷)

چکیده

ضرب اعداد اعشار ممیز شناور (FP) یکی از پرهزینه ترین پردازش‌های پردازنده از نظر زمان و انرژی است و از طرف دیگر کاربرد زیادی نیز در الگوریتم‌های مختلف دارد. با توجه به تحمل پذیری خطا در بسیاری از الگوریتم‌های امروزی تقریبی نمودن ضرب یکی از روش‌های افزایش کارایی آن است. اما با این وجود در برخی از روش‌های ضرب تقریبی خطا به میزان قابل توجهی افزایش می‌یابد. در این مقاله یک الگوریتم ضرب تقریبی جدید به نام "ضرب تقریبی با خطای کم" (LEAM) معرفی شده که همزمان به دنبال افزایش کارایی و حفظ خطا در سطح قابل قبول است. رویکرد پیشنهادی این مقاله توانسته در مقایسه با روش RMAC خطا را به میزان ۸۹٪ کاهش داده در حالیکه زمان اجرای LEAM و RMAC تفاوت چندانی با هم ندارند. نتایج نشان داده است که LEAM حداکثر به میزان ۳ درصد سریعتر و در برخی موارد حداکثر به همین میزان کندتر از RMAC می‌باشد. علاوه بر این LEAM و RMAC در این مقاله به صورت کاملا نرم افزاری پیاده سازی شده اند و جهت کاهش هزینه‌های پیاده سازی نرم افزاری، دستورات AVX-512 در پردازنده‌های با قابلیت یک دستور بر روی چندین داده (SIMD) بکار گرفته شده است.

کلمات کلیدی: ضرب تقریبی، یک دستور بر روی چندین داده (SIMD)، محاسبات تقریبی، برداری سازی

*عهده‌دار مکاتبات:

صباغ ملاحسینی، امیر

نشانی: **بخش مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان، ایران**

پست الکترونیکی: sabbagh@iauk.ac.ir

۱- مقدمه

در حال حاضر، مصرف انرژی یکی از مهم‌ترین نگرانی‌ها حتی در سیستم‌های محاسباتی بزرگ است، زیرا هزینه انرژی و سرمایش مراکز داده بیشتر از هزینه دستگاه‌های دیگر است [۱]. از سوی دیگر، الگوهای محاسباتی جدید مانند اینترنت اشیا[۱] و هوش محاسباتی مرزی[۲] نیاز به سیستم‌های محاسباتی سریع و با کارایی انرژی بالا جهت پردازش به صورت بلادرنگ دارند [۲, ۳]. منابع اصلی مصرف انرژی در سیستم‌های محاسباتی شامل محاسبات عددی و جابجایی داده بین سلسله مراتب حافظه هستند [۳]. برخی از برنامه‌های کاربردی مانند پردازش چندرسانه‌ای/پردازش سیگنال، استخراج داده و یادگیری ماشین، نیازی به دقت بالا و نتایج دقیق ندارند [۴]. با این حال، برخی دیگر از برنامه‌های کاربردی محاسباتی مانند سیستم‌های پردازش گراف، حداقل در بخشی از محاسبات خود نیاز به دقت بالایی دارند [۵–۷]. در این رابطه، محاسبات تقریبی به عنوان رویکردی برای ارائه محاسبات عددی کارآمد، پژوهش‌های گسترده‌ای را به خصوص در حوزه هوش مصنوعی و شبکه‌های عصبی طی سال‌های گذشته به خود اختصاص داده است [۸]. مقالات اخیر نشان داده اند که استفاده از محاسبات تقریبی امکان توسعه و پیاده سازی شبکه‌های عصبی عمیق را با کاهش استفاده از حافظه و پیچیدگی محاسبات هموار و امکان پذیر می‌کند [۹]. در حوزه محاسبات تقریبی برخی از تحقیقات همچون [۱۰–۱۶] تلاش دارند تا مصرف انرژی واحد پردازشی را کاهش دهند و برخی دیگر همچون [۲, ۵–۷, ۱۷] تلاش دارند تا فرمت حافظه و سیستم عددی جدیدی را برای کاهش پهنای باند مصرفی معرفی کنند. تحقیقات نشان می‌دهد در حدود ۵۰ درصد از انرژی مصرفی در سیستم‌های تعبیه شده مربوط به حافظه و محاسبات بر روی اعداد اعشار ممیز شناور[۳] است [۱۸]. برای اثبات این مسئله مقاله [۱۸] برنامه‌های کاربردی دارای حجم بالای محاسبات بر روی اعداد FP را بر روی میکروکنترلر PULPino [۱۹] اجرا کرد. نتایج نشان داد که ۳۰ درصد از انرژی مصرفی مربوط به محاسبات بر روی FP است و ۲۰ درصد دیگر مربوط به انتقال اعداد FP از حافظه به ثبات و بالعکس می‌باشد. در بیشتر برنامه‌ها، ضرب، یکی از عملیات‌های فراوان و با مصرف انرژی بالا است و بسیاری از برنامه‌های تحمل پذیر خطا مانند یادگیری ماشین و ضرب ماتریس، نیاز به تعداد قابل توجهی ضرب دارند. بنابراین، کاهش هزینه این عملیات لز طریق تقریبی نمودن آن می‌تواند عملکرد بسیاری از برنامه‌ها را بهبود بخشد. از طرف دیگر همانطور که نتایج مقاله [۱۸] نشان می‌دهد ضرب اعداد FP از نظر کارایی بسیار کند بوده و مصرف انرژی زیادی دارد. تکنیک‌های مختلف ضرب تقریبی را می‌توان در ۳ سطح مختلف دسته بندی نمود [۲۰]. سطح الگوریتم مانند [۱۰, ۱۳–۱۶, ۲۱] شامل تکنیک‌هایی است که تلاش می‌کند در الگوریتم ضرب تغییراتی دهند تا سریعتر و کارآمدتر شود. تکنیک‌های در این سطح تنها امکان پیاده سازی نرم افزاری دارند. در تکنیک‌های سطح معماری تلاش دارند تا با ایجاد تغییراتی در معماری و مراحل مختلف سخت افزاری ضرب باینری عملکرد آن را بهبود بخشند مانند [۱۶, ۲۲–۲۴] و در تکنیک‌های سطح مدار مانند [۱۶, ۲۲, ۲۳] در سطح بسیار پایین طراحی مدار تغییراتی جهت بهبود عملیات ضرب ایجاد می‌کنند. در بسیاری از تحقیقات تکنیک‌های سطح مختلف ترکیب می‌شوند. پژوهش‌ها در حوزه ضرب تقریبی را می‌توان به

[۱] internet of things (IOT)

[۲] edge intelligence

[۳] floating point number (FP)

دو دسته کلی تقسیم نمود: برخی بر روی ساخت الگوریتم‌های جدید جهت ضرب تقریبی تمرکز دارند و برخی دیگر تلاش کرده اند سیستمی قابل پیکربندی در زمان اجرا برای تنظیم دقت در طول ضرب تقریبی معرفی کنند، زیرا کنترل میزان خطا در نتیجه‌ی نهایی یک مسئله در محاسبات تقریبی مهم است. به عنوان مثال، [۲۵] سعی کرد تأثیر ضرب تقریبی بر مصرف انرژی و دقت نتیجه نهایی در شبکه‌های عصبی کانولوشنال (CNN) را تحلیل کند. نتیجه این تحقیق نشان داد که ضرب تقریبی به انرژی کمتری نیاز دارد و دقت نتیجه نهایی آن، با استفاده از اعداد اعشاری با دقت منفرد (SP)، برابر است.

در این تحقیق قصد داریم تا یک الگوریتم ضرب تقریبی جدید برای اعداد FP به نام ضرب تقریبی با خطای کم[۴] را معرفی کنیم. الگوریتم LEAM به صورت کاملا نرم افزاری برخلاف روش‌های گذشته پیاده سازی شده است تا هزینه پیاده سازی آن کاهش یافته و قابلیت اجرا بر روی پردازنده‌های همه منظوره امروزی را به راحتی داشته باشد. این الگوریتم از تکنیک خطی کردن که یک تکنیک در سطح الگوریتم می‌باشد، برای تقریب استفاده کرده و تلاش کرده است تا به حفظ خطا در سطح قابل قبول امکان استفاده از این الگوریتم را در قسمت‌های مختلف یک برنامه بدون نگرانی از افزایش بی رویه خطا فراهم کند. با توجه به اینکه بیشتر تحقیقات در حوزه محاسبات تقریبی بر روی پیاده سازی سخت افزاری روش پیشنهادی خود متمرکز بوده اند و بر روی تکنیک‌های سخت افزاری جهت بهبود دقت و کارایی و مصرف انرژی تکیه داشته اند، در حالیکه یکی از اهداف این تحقیق معرفی یک الگوریتم ضرب تقریبی با قابلیت پیاده سازی بر روی پردازنده‌های همه منظوره کنونی است. در نتیجه انتخاب یک الگوریتم از تحقیقات گذشته جهت مقایسه و ارزیابی LEAM که از یک طرف قابلیت پیاده سازی بر روی نرم افزار را بدون از دست دادن هیچکدام از امکانات و ویژگی‌ها را داشته و از طرف دیگر یک تحقیق برجسته در این حوزه باشد، یکی از مهمترین چالش‌های این مقاله بوده است. در این میان الگوریتم ضرب کننده اعداد اعشار ممیز شناور با قابلیت پیکره بندی در زمان اجرا برای محاسبات تقریبی[۵] [۱۰] یکی از مقالات موفق در حوزه ضرب تقریبی خطی است که مورد ارزیابی و مقایسه در مقالات متعددی همچون [۱۴, ۱۵] بوده و امکان پیاده سازی نرم افزاری به همراه تمامی قابلیت‌های آن وجود دارد. بنابراین این تحقیق این مقاله را به عنوان مقاله هدف جهت ارزیابی LEAM انتخاب کرده است. با توجه به پیاده سازی سخت افزاری این روش در مقاله [۱۰]، در این تحقیق پیاده سازی نرم افزاری از RMAC نیز ارائه شده و مورد بررسی قرار گرفته است. با توجه به اینکه پیاده سازی نرم افزاری ضرب تقریبی سربار قابل توجهی را نسبت به پیاده سازی سخت افزاری دارد، از این رو این مقاله جهت کاهش این سربار از تکنیک یک دستورالعمل بر روی چندین داده[۶] بر روی پردازنده‌های اینتل با قابلیت AVX-512 استفاده نموده است. پردازنده‌های اینتل AVX-512 دارای ثبات‌های ۵۱۲ بیتی بوده و امکان انجام همزمان محاسبات بر روی ۸ داده ۶۴ بیتی و یا ۱۶ داده ۳۲ را فراهم می‌کنند. و به دلیل آنکه این تکنیک موازی سازی کاملا در ثبات‌ها و به صورت سخت افزاری انجام می‌شود می‌تواند سرعت را افزایش داده و سربار ناشی از پیاده سازی نرم افزاری را به صورت قابل توجهی از بین ببرد. LEAM همچون RMAC و سایر روش‌های خطی، ضرب مانتیس‌ها را با جمع جایگزین می‌کند اما برخلاف RMAC با حفظ محدوده مانتیس در ضرب دقیق تلاش می‌کند تا خطا را در سطح قابل قبولی حفظ کرده تا علاوه

[۴] Low Error Approximate Multiplier(LEAM)

[۵] runtime configurable floating point multiplier for approximate computing (RMAC)

[۶] single instruction multiple data (SIMD)

بر افزایش عملکرد، خطا نیز به طور افسارگسیخته ای افزایش نیابد و بتوان محاسبات بیشتری را نسبت به RMAC به صورت تقریبی با استفاده از LEAM انجام داد.

بقیه مقاله به شرح زیر است: بخش ۲ مروری دارد بر کارهای مرتبط در حوزه محاسبات تقریبی به روش خطی و بخش ۳ به طور خلاصه استاندارد IEEE 754 برای اعداد اعشار ممیز شناور و روش ضرب با استفاده از این استاندارد را مرور می‌کند. بخش ۴ LEAM را معرفی کرده و جزئیات پیاده سازی نرم افزاری و برداری سازی شده آن را به همراه روش RMAC بر روی پردازنده‌های اینتل توضیح می‌دهد. بخش ۵ ارزیابی پیاده سازی نرم افزاری LEAM در مقایسه با RMAC را ارائه می‌دهد و در نهایت بخش ۶ به نتیجه‌گیری مقاله می‌پردازد.

۲- کارهای مرتبط

روش‌های مختلفی برای تقریب وجود دارد. یکی از این روش‌ها تقریب خطی است. این روش که از تکنیک‌های در سطح الگوریتم است، تلاش می‌کند تا تابع غیر خطی ضرب را با استفاده از جمع به یک تابع خطی تبدیل کنند. یکی از روش‌های خطی استفاده از لگاریتم است. دلیل استفاده از این روش تبدیل ضرب به جمع و شیفت در لگاریتم می‌باشد. به دلیل آنکه ورودی‌ها باید به لگاریتم تبدیل شوند، در نتیجه این تبدیل ذاتا تقریبی است و همواره مقداری خطا به نتیجه نهایی اضافه می‌شود. این روش برای اولین بار توسط Mitchel در سال ۱۹۶۲ [۲۶] معرفی گردید. یکی از مشکلات روش Mitchel خطای تقریب بالای آن بود زیرا عملیات ضرب نیز میزانی خطا را به الگوریتم Mitchel اضافه می‌کند. بنابراین تحقیقات در این حوزه عمدتا متمرکز بر روی ترکیب استفاده از مدارهای تقریبی و روش‌های تقریب برای کاهش خطای محاسبه لگاریتم ورودی‌ها می‌باشند. در [۱۳] یک الگوریتم تکرار شونده معرفی گردید. در این مقاله با محاسبه خطای ضرب روش Mitchel پی بردند که این خطا ساختار مشابه با ضرب اولیه دارد. در نتیجه می‌توانند با ایجاد یک الگوریتم ضرب تکرار شونده از ساختار Mitchel برای محاسبه این ضرب نیز استفاده کرده و با تکرار این الگوریتم می‌توان دقت را به اندازه قابل قبولی افزایش داد. این مقاله همچنین روش Mitchel را با تکنیک کوتاه کردن ترکیب نمود. Liu و همکاران در [۲۷] ضرب‌های تقریبی لگاریتمی تکرار شونده و غیر تکرار شونده را به دقت بررسی کردند و توانستند یک مجموعه مدارهای جمع تقریبی و دقیق جدیدی را جایگزین کنند که دقت به اندازه ۱۸ درصد و توان مصرفی به اندازه ۳۷ درصد بهبود یابد. در [۱۶] یک الگوریتم ضرب تقریبی لگاریتمی و بهره ور در مصرف انرژی را برای کاربرد شبکه‌های عصبی معرفی کرده است. در این مقاله برای محاسبه لگاریتم ورودی‌ها آن‌ها را به نزدیکترین توان از ۲ گرد می‌کند که حجم جدول جستجو کاهش یابد. آن‌ها توانستند در حدود ۸ درصد دقت را افزایش داده و در کاربردهای مختلف به طور میانگین به اندازه ۲۱/۸۵ درصد مصرف انرژی را کاهش دهند. در برخی دیگر از کارها همچون Gao و همکاران [۴] از روش‌های لگاریتمی را جهت شناسایی محاسبات با قابلیت تقریبی استفاده نمودند. آن‌ها در مقاله خود سیستم لگاریتم را بر روی گراف جریان داده یک برنامه اعمال کردند تا محاسبات با قابلیت تحمل خطا را شناسایی کرده و محاسبات قسمت‌های شناسایی شده را به صورت تقریبی انجام دهند. آن‌ها توانستند نسبت به روش‌های استاتیک هم دقت را افزایش داده و هم در مصرف انرژی صرفه جویی کنند. در حالیکه روش‌های ضرب تقریبی لگاریتمی در حقیقت ضرب را به

یک ضرب بر روی اعداد اعشار ممیز ثابت تبدیل کرده و در نتیجه به دلیل کاهش دقت خطا در این روش‌ها افزایش می‌یابد اما تحقیقات محدودتری برای ضرب اعداد اعشار ممیز شناور معرفی شده اند. در ضرب تقریبی اعداد اعشار ممیز شناور با توجه به اینکه ماهیت ورودی‌ها برخلاف روش‌های لگاریتمی به صورت FP باقی می‌ماند خطا نیز کمتر خواهد بود. در روش‌های خطی برای ضرب اعداد اعشار ممیز شناور تلاش شده است تا پر هزینه ترین عملیات یعنی ضرب مانتیس‌ها را خطی کرده و به جمع تبدیل کنند. مانند RMAC [۱۰] که ضرب دو مانتیس را با جمع جایگزین نمود و توانست سرعت را تا ۳/۱ برابر افزایش و انرژی را به اندازه ۱/۸ برابر کاهش دهد. RMAC جهت کنترل خطا از N بیت با ارزش مانتیس تقریبی حاصل استفاده می‌کند و در صورتیکه خطا از حد آستانه بیشتر باشد ضرب را یک بار دیگر به صورت دقیق محاسبه می‌کند. نتایج RMAC نسبت به مقاله پیشین یعنی ضرب کننده ممیز شناور قابل پیکره بندی برای محاسبات با کارایی بالای انرژی[۷] [۲۱] بسیار بهبود داشت. گرچه CFPU یک روش خطی نیست اما در این روش نیز با استفاده از ۳ تکنیک مختلف تقریب مانتیس، انتخاب انطباقی و تنظیم کننده، ضرب پر هزینه مانتیس‌ها را حذف می‌کند و به جای آن مانتیسی که کمترین خطا را تولید می‌کند انتخاب می‌کند. در CFPU نیز جهت کنترل خطا از N بیت با ارزش مانتیس حذف شده استفاده می‌کند و در صورتیکه حذف این مانتیس منجر به بیشتر شدن خطا از حد آستانه شود ضرب دقیق انجام می‌گیرد. نرخ خطای CFPU زیاد و بین صفر و ۵۰ درصد بوده و مصرف انرژی با افزایش دقت مورد نیاز افزایش می‌یابد. پس از RMAC در سال ۲۰۱۹ الگوریتمی به نام ApproxLP [۱۴] معرفی شده که از یک تکنیک تقریب خطی تکرار شونده برای جایگزینی ضرب مانتیس‌ها با جمع استفاده می‌کرد. در این روش در حقیقت ضرب مانتیس‌ها با جمع‌های وزن دار جایگزین می‌شود تا میزان خطا افزایش چشم گیری نداشته باشد. این مقاله دامنه مربع [1,2)×[1,2) ورودی را به زیر دامنه‌های مختلف تقسیم می‌کند و به ازای هر زیر دامنه تابع خطی مخصوص آن زیر دامنه تعریف می‌شود. پارامترهای ورودی ضرب با عبور از توابع خطی مختلف نتیجه نهایی را تولید می‌کنند. در این الگوریتم به راحتی می‌تواند دقت و مصرف انرژی را با توجه به برنامه‌های کاربردی مختلف تغییر داده و تنظیم کرد. ApproxLP توانست ۴/۵ برابر در مصرف انرژی صرفه جویی کند. اما مشکل این الگوریتم تاخیر زیاد ناشی از وجود دستورات شرطی است که با افزایش تعداد زیر دامنه‌ها نیز به صورت نمایی افزایش یافته و تاثیر زیادی بر روی کارایی دارد [۱۵]. به همین دلیل گروه نویسندگان مقاله ApproxLP الگوریتم دیگری را در سال ۲۰۲۰ [۱۵] پیشنهاد دادند. برای کاهش تعداد دستورات مقایسه، Chen و همکاران دامنه ورودی را به زیر دامنه‌های مساوی تقسیم می‌کردند. در هر سطح یک دامنه به ۴ دامنه مساوی تقسیم می‌شود. در نتیجه دستورات مقایسه و در نتیجه تاخیر ناشی از آن‌ها در این الگوریتم حذف می‌شود. در این الگوریتم نیز همچون ApproxLP می‌توان دقت را در زمان اجرا با تقسیم بیشتر دامنه افزایش داد.

۳- استاندارد IEEE 754 برای نمایش اعداد اعشاری با ممیز شناور

IEEE 754 استانداردی جهت نمایش اعداد اعشار ممیز شناور در سیستم‌های کامپیوتری امروزی می‌باشد. هر عدد اعشاری ممیز شناور IEEE 754 شامل سه بخش است: ۱) نما، یک عدد صحیح برای تعیین محدوده پویا، ۲) مانتیس، یک عدد اعشاری مثبت بین صفر و یک، و ۳) یک بیت برای نشان‌دادن علامت مانتیس. به عبارت دیگر، نما نشان‌دهنده بزرگی یا

[۷] configurable floating point multiplier for energy-efficient computing (CFPU)

کوچکی و مانتیس نشان‌دهنده دقت یک عدد اعشاری FP است. بنابراین، افزایش/کاهش بیت‌های نما برای افزایش/کاهش محدوده اعداد قابل نمایش و افزایش/کاهش بیت‌های مانتیس برای افزایش/کاهش دقت اعداد FP مورد استفاده قرار می‌گیرد. علاوه بر این، تمام اعداد اعشاری ممیز شناور نرمال شده شامل یک عدد مخفی یک هستند که در حافظه ذخیره نمی‌شود، اما در محاسبات به مانتیس اضافه می‌شود. عدد ثابتی به نام بایاس به نما اضافه می‌شود تا قبل از ذخیره سازی در حافظه آن را به یک عدد صحیح مثبت تبدیل کند. طول نما و مانتیس و مقدار بایاس بسته به نوع اعداد اعشاری ممیز شناور در IEEE 754 متفاوت است. این استاندارد انواع مختلفی برای FP تعریف می‌کند، اما دو نوع آن‌ها از همه متداول‌تر است و تمام پردازنده‌ها از آن پشتیبانی می‌کنند: اعداد اعشار ممیز شناور با دقت منفرد[۸] و اعداد اعشار ممیز شناور با دقت مضاعف[۹]. SP و DP به ترتیب ۴ و ۸ بایت در حافظه می‌گیرند. جدول ۱ تعداد بیت‌های مانتیس و نما، مقدار بایاس و بزرگترین عدد قابل نمایش برای هر نوع را نشان داده است. به علاوه یک بیت نیز برای علامت مانتیس ذخیره شده است. هر چند که فرمت نیمه دقیق[۱۰] استاندارد IEEE در برخی ازGPU ها وجود دارد، اما هیچ گزینه‌ای برای استفاده از داده‌های FP با دقت کمتر و محدوده پویا برابر SP و DP که منجر به کاهش طول داده نیز می‌شود در پردازنده‌های کنونی وجود ندارد. بنابراین، عدم انعطاف‌پذیری، بزرگترین مشکل استاندارد IEEE است، به‌ویژه برای برنامه‌هایی که محاسبات تقریبی می‌تواند مورد استفاده قرار گیرد ([۱۷]). یک مشکل دیگر، احتمال بروز سرریز و underflow هنگام تبدیل از DP به SP است.

جدول ۱: استاندارد IEEE برای اعداد FP

نوع داده	نما	بایاس	مانتیس	بزرگترین عدد قابل نمایش
DP	۱۱	۱۰۲۳	۵۲	$۲^{۱۰۲۴}\times(^{۵۳}-۲-۱)$
SP	۸	۱۲۷	۲۳	$۲^{۱۲۸}\times(^{۲۴}-۲-۱)$

ضرب در IEEE 754 از دو بخش تشکیل شده است: ابتدا نمای اعداد با هم جمع می‌شود و سپس (مانتیس + ۱) هر دو عدد در هم ضرب می‌شوند. شکل ۱ نشان می‌دهد که چگونه دو عدد FP در IEEE 754 ضرب می‌شوند. نمای هر دو ورودی شامل بایاس است؛ بنابراین، نتیجه جمع یک بایاس اضافی دارد که باید حذف شود. همچنین مانتیس نتیجه نهایی هم باید به‌طور معمول بین ۱ و ۲ نرمال شود.

[۸] single point floating point number (SP)

[۹] double point floating point number (DP)

[۱۰] half precision floating point number

شکل ۱: استاندارد IEEE 754 برای اعداد FP و تبدیل آن به عدد اعشار متناظر [۱۷]

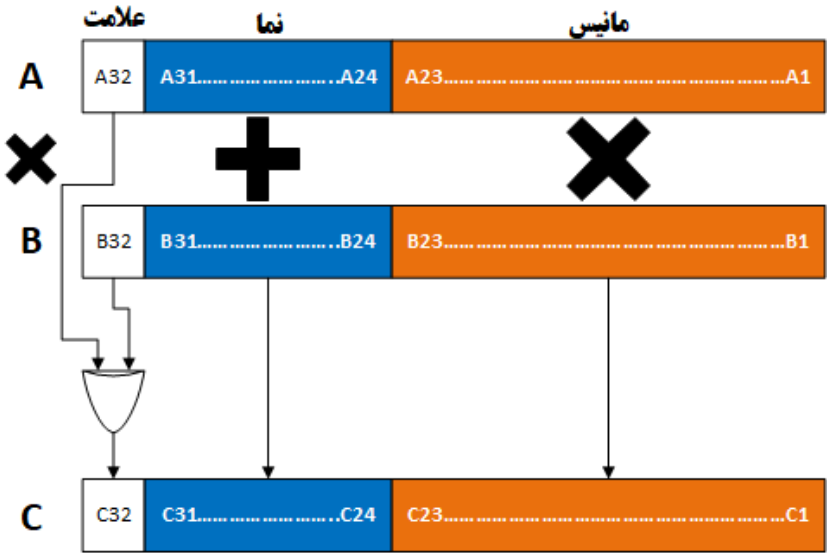
۴- روش پیشنهادی

RMAC با جایگزینی عملیات ضرب با عملیات جمع، هزینه ضرب را کاهش داده است [۱۰]. هرچند که بسیاری از تحقیقات در این حوزه پیاده سازی سخت افزاری را پیشنهاد داده اند. عملیات ضرب در مانتیس‌ها از مهمترین عملیات‌ها در ضرب FP است. رابطه (۱) ضرب دو عدد FP را بر اساس استاندارد IEEE-754 نشان می‌دهد. در این ضرب، نماها با یکدیگر جمع شده و عملیات ضرب روی مانتیس‌ها انجام می‌شود (شکل ۲). بنابراین، این عملیات ضرب هزینه بالایی دارد و به همین دلیل، RMAC عملیات ضرب را با جمع جایگزین نموده است تا کارایی را افزایش دهد. رابطه (۲) نحوه ضرب در RMAC را نشان داده است. در ضرب مانتیس‌ها به روش RMAC، هیچ رقم نقلی ایجاد نمی‌شود زیرا محدوده هر دو مانتیس ثابت می‌ماند. اما در جمع مانتیس‌های دو عدد FP همانطور که در شکل ۳ نشان داده شده است، ممکن است بیت نقلی تولید شود که RMAC آن را به نمای نتیجه اضافه می‌کند [۱۰].

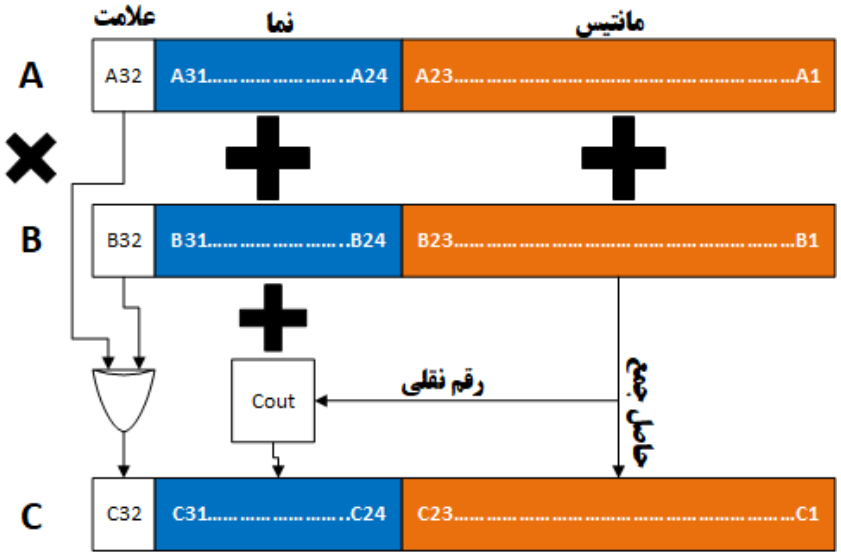
$$\begin{cases} a = -1^{S_a} \times 1.m_a \times 2^{e_a} \\ b = -1^{S_b} \times 1.m_b \times 2^{e_b} \end{cases} \Longrightarrow c = -1^{S_a \otimes S_b} \times (1 + m_a) \times (1 + m_b) \times 2^{e_a + e_b} \quad (۱)$$

$$\xrightarrow{\text{RMAC}} c_{approx} = -1^{S_a \otimes S_b} \times (1 + \mathrm{m_a}) + (1 + \mathrm{m_b}) \times 2^{e_a + e_b} \quad (۲)$$

جایگزینی ضرب با جمع در RMAC باعث افزایش خطا به طور قابل توجهی شده است و می‌تواند منجر به کاهش قابلیت استفاده از این روش شود. با وجود آنکه الگوریتم‌های تحمل پذیر خطای زیادی همچون الگوریتم‌های یادگیری ماشین وجود دارند اما افزایش خطای خروجی منجر به کاهش قابل توجه کیفیت خروجی می‌گردد. بنابراین، کنترل مقدار خطا در محاسبات تقریبی اهمیت بسزایی دارد. در این مقاله یک الگوریتم جدید ضرب تقریبی به نام LEAM معرفی شده است که هدف آن کاهش خطای روش RMAC در عین حفظ کارایی و سرعت روش RMAC می‌باشد.



شکل ۲: ضرب معمولی در استاندارد IEEE 754



شکل ۳: ضرب تقریبی RMAC

۴-۱- روش LEAM

از بخش ۳ می‌دانیم که مانتیس‌ها همواره بین ۱ و ۲ هستند هر چند که بخش صحیح ۱ عدد هیچگاه در حافظه ذخیره نمی‌شود. مطابق رابطه (۳)، هنگامی که دو مانتیس با یکدیگر ضرب می‌شوند، محدوده نتیجه قبل از نرمال سازی بین ۱ و ۴ است. اما محدوده مانتیس پس از ضرب RMAC به دلیل آنکه در این روش مانتیس‌ها با هم جمع می‌شوند، بین ۲ و ۴ قرار می‌گیرد (رابطه (۴)). این مشکل در RMAC، یعنی حذف محدوده بین صفر و یک، باعث افزایش خطا می‌شود، به ویژه در برنامه‌هایی که چندین ضرب متوالی نیاز دارند.

$$0 \le m < 1 \Longrightarrow 1 \le 1 + m < 2 \xrightarrow{\text{normal mult}} 1 \le (1 + m_a)(1 + m_b) < 4 \quad (۳)$$

$$0 \le m < 1 \Longrightarrow 1 \le 1 + m < 2 \xrightarrow{\text{RMAC}} 2 \le (1 + m_a) + (1 + m_b) < 4 \quad (۴)$$

LEAM از ایده RMAC استفاده کرده و ضرب دو مانتیس را با جمع جایگزین می‌کند اما همزمان محدوده مانتیس نتیجه را بین ۱ و ۴ حفظ می‌کند. برای این منظور، ابتدا ضرب دو مانتیس به صورت رابطه (۵) توزیع می‌شود:

$$(1 + m_a)(1 + m_b) = 1 + m_a + m_b + m_a m_b \quad (۵)$$

تنها ضرب هزینه‌بر در رابطه (۵) ضرب دو عدد m_a و m_b است. نتیجه‌ی ضرب این دو عدد همیشه بین ۰ و ۱ باقی می‌ماند و کوچکتر از هر دو عدد است چرا که m_a و m_b بین ۰ و ۱ هستند. فرض کنید مانتیس عدد b کوچکتر از مانتیس عدد a باشد. بنابراین، می‌توانیم ضرب دو مانتیس را با کمترین مانتیس، که m_b است، تخمین بزنیم:

$$\begin{aligned} m_a m_b < m_a, m_b \Longrightarrow m_a m_b \approx \min(m_a, m_b) \Longrightarrow 1 + m_a + m_b + m_a m_b \\ \approx 1 + m_a + m_b + \min(m_a, m_b) \xrightarrow{\text{if } \min(m_a, m_b) = m_b} = 1 + m_a + 2m_b \end{aligned} \quad (۶)$$

محدوده مانتیس در رابطه (۶) به همان اندازه ضرب معمولی است، زیرا:

$$\begin{cases} 0 \le m_a < 1 \\ 0 \le 2m_b < 2 \end{cases} \Longrightarrow 0 \le m_a + 2m_b < 3 \Longrightarrow 1 \le 1 + m_a + 2m_b < 4 \quad (۷)$$

مانندRMAC ، رقم نقلی حاصل از جمع مانتیس‌های رابطه (۶) باید به نمای نتیجه در LEAM اضافه شوند.

۴-۲- جزئیات پیاده‌سازی LEAM

با وجود آنکه RMAC [۱۰] بر روی سخت افزار پیاده سازی شده است اما در این مقاله جهت مقایسه خطا و زمان اجرای LEAM و RMAC، هر دوی این الگوریتم‌ها را بر روی نرم افزار پیاده‌سازی شده است. در این پیاده‌سازی از تکنیک SIMD برای کاهش سربار ناشی از پیاده سازی نرم افزاری بهره گرفته شده است. پردازنده‌های SIMD اینتل دارای معماری و ثبات‌هایی با اندازه‌های مختلف جهت بردارسازی می‌باشند. پردازنده‌های AVX512 دارای ثبات‌های ۵۱۲ بیتی [۲۸]، AVX/AVX2 دارای ثبات‌های ۲۵۶ بیتی و SSE دارای ثبات‌های ۱۲۸ بیتی [۲۹] می‌باشند. علاوه بر این، اینتل توابع intrinsicرا برای معماری‌های مختلف معرفی کرده است تا بردارسازی را بر روی پردازنده‌های خود را ساده‌تر کند [۳۰]. در این مقاله، LEAM و RMAC بر روی پردازنده‌های AVX512 اینتل پیاده سازی شده اند بنابراین تمامی شبه کدها از توابه AVX512 intrinsic بهره گرفته اند.

در ابتدا، علامت اعداد اعشار در LEAM در نظر گرفته نمی‌شود. بنابراین نمای عدد اعشار FP با بزرگترین مانتیس به چپ شیفت داده می‌شود و سپس یکی به مانتیس این عدد اضافه می‌شود. این عملیات $(1+m_a)$ در رابطه (۶) را محاسبه می‌کند. برای دو برابر کردن عدد FP با کمترین مانتیس، این عدد فقط یک بیت به چپ شیفت داده می‌شود. شکل ۴ و شکل ۵ این عملیات برای اعداد ورودی SP نشان می‌دهد. پس از آن، دو عدد FP با هم جمع می‌شوند. در این مرحله، بیت‌های مانتیس و

نما با هم جمع می‌شوند و رقم نقلی که از جمع مانتیس‌ها تولید می‌شود به صورت خودکار به نمای حاصل اضافه می‌شود. نمای حاصل دارای یک بایاس اضافه است که باید از آن کم شود.

شکل ۴: عملیات LEAM بر روی عدد SP با مانتیس بزرگتر

شکل ۵: عملیات LEAM بر روی عدد SP با مانتیس کوچکتر

طبق رابطه (۷)، اگر مانتیس حاصل‌ضرب کمتر از ۲ باشد، نیازی به نرمال‌کردن آن نیست. در غیر این صورت، اگر بخش صحیح مانتیس ۲ یا ۳ باشد، تنها با شیفت به سمت راست آن را نرمال می‌کنیم. اگر بیت ۲۴ در SP یا بیت ۵۳ در DP حاصل‌ضرب یک باشد و بیت متناظر در FP با کمترین مانتیس صفر باشد، نیازی به نرمال‌کردن حاصل‌ضرب نیست. در غیر این صورت، باید حاصل‌ضرب را نرمال کنیم. شکل ۶، شبه کد نحوه نرمال‌کردن حاصل‌ضرب با استفاده از دستورات AVX-512را نشان می‌دهد.

۵- ارزیابی تجربی LEAM

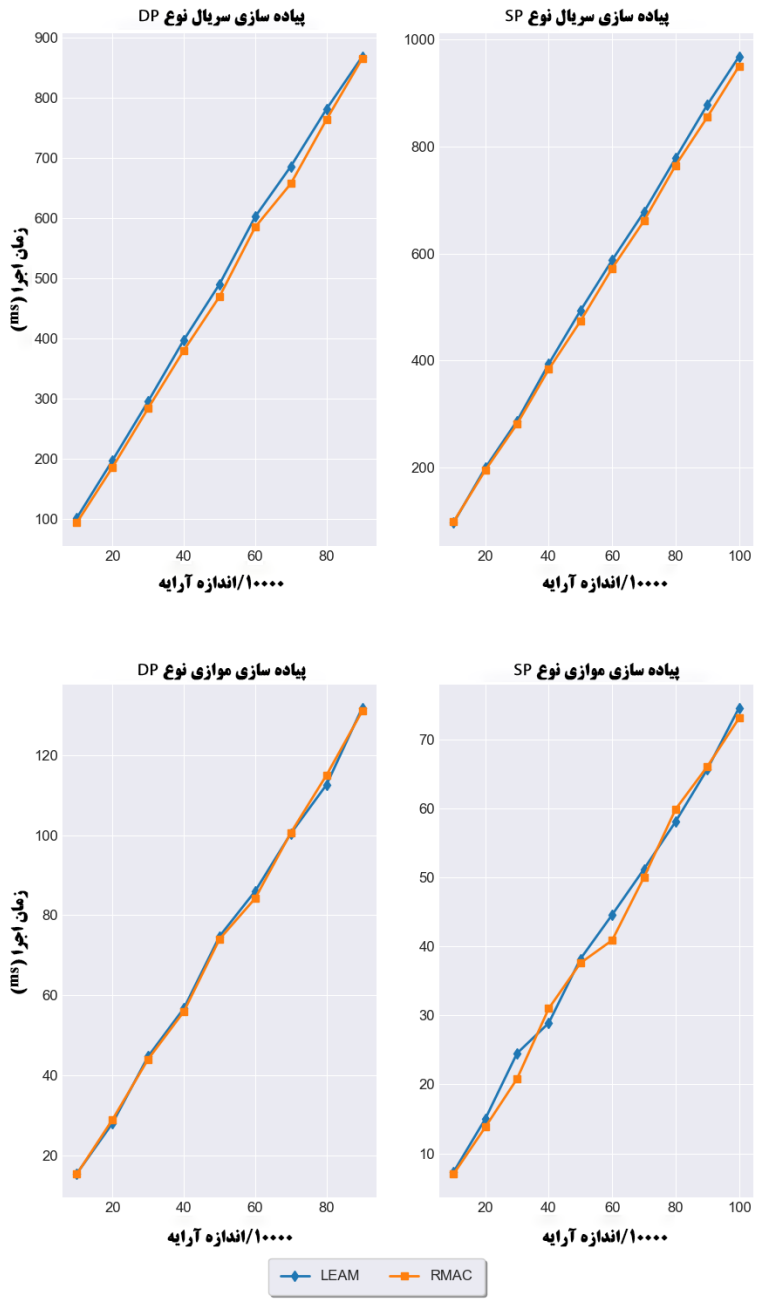
کلیه پیاده سازی‌های این مقاله بر روی یک سرور با مدل Intel Xeon platinum 8168 و نسخه GCC ۱۰,۲ انجام شده است. اندازه کش‌های L1 ، L2 و L3 در این سیستم به ترتیب ۱,۵ مگابایت، ۲۴ مگابایت و ۳۳ مگابایت است. همانطور که می‌دانیم، میزان خطا در ضرب اعداد تقریبی زمانی افزایش می‌یابد که تعداد ضرب‌های پیاپی افزایش یابد. بنابراین، ما تابع ضرب داخلی را به عنوان تابع نمونه جهت ارزیابی الگوریتم انتخاب کرده ایم تا تحلیلی مناسب از نظر خطا داشته باشیم. علاوه بر این، ما نسخه SIMD از RMAC را با استفاده از همان سیستم پیاده سازی کرده ایم تا زمان اجرا و خطای LEAMو RMAC را مقایسه کنیم. اندازه‌های مختلفی برای آرایه‌ها در تابع ضرب داخلی انتخاب شده است تا زمان اجرا روی مجموعه داده‌های مختلف ارزیابی شود. تحلیل بر روی هر دو نوع داده‌های ممیز شناور DP و SP، انجام می‌شود.

```
    //c is the result and min is the number with the smallest mantissa
    //check if the result need normalization or not
    __m512i c_most_sig_man = _mm512_and_si512(c, NORMALIZE_AVX512);
    __m512i min_most_sig_man = _mm512_and_si512(min, NORMALIZE_AVX512);
    __m512i is_normalize = _mm512_or_si512(min_most_sig_man,
_mm512_xor_si512(c_most_sig_man, NORMALIZE_AVX512));

    //if the corresponding bit in mask_is_normalize is 0, the result does not need
normalizing
    __mmask16 mask_is_normalize = _mm512_cmpeq_epi32_mask(is_normalize,
NORMALIZE_F_AVX512);

    //when we need normalizing just shift c to the right
    __m512i c_normalize = _mm512_srli_epi32(c, 1);


    c = _mm512_mask_blend_epi32(mask16, c, c_normalize);
```

شکل ۶: شبه کد نرمال سازی در AVX512. **مقدار ثابت** NORMALIZE_AVX512 **در** SP **برابر** 0x00800000 **و در** DP **برابر** 0x0010000000000000 **است. همچنین نوع متغیر** mask_is_normalize **در** DP، __mmask8 **است.**

در ابتدا، زمان اجرای LEAM و RMAC در ضرب داخلی بر روی اعداد DP و SP به صورت سریال و برداری شده با هم مقایسه شده است. همانطور که در

شکل ۷ مشاهده می‌شود پیاده سازی SIMD هر دو الگوریتم ۱۰ برابر سریعتر از پیاده سازی سریال است. این نتایج تاثیر تکنیک SIMD بر روی تسریع الگوریتم‌ها را به خوبی نشان می‌دهد. در برخی از حالات LEAM به اندازه ۲ تا ۳ درصد سریعتر از RMAC می‌باشد و در برخی از حالات دیگر RMAC به همین اندازه از LEAM سریعتر است. این نتایج نشان می‌دهد که به طور میانگین LEAM و RMAC از نظر سرعت تفاوت چندانی با هم ندارند. علت کندی LEAM در برخی از حالات نیاز به چک کردن نتیجه نهایی برای نرمال بودن یا نبودن است زیرا محدوده مانتیس در LEAM بین ۱ و ۴ است و در صورتیکه این محدوده بین ۱ و ۲ باشد نیاز به نرمال سازی وجود ندارد و در غیر این صورت نتیجه نهایی می‌بایست نرمال سازی شود. در حالیکه RMAC با نگه داشتن مانتیس بین ۲ و ۴ همواره نیاز به نرمال سازی نتیجه نهایی دارد. به عبارت دیگر RMAC نیاز به بررسی محدوده مانتیس نتیجه نهایی ندارد.



شکل ۷: مقایسه زمان اجرای LEAM و RMAC

در این مقـالـه میـانگین مربع خطاها[۱۱] برای تحلیل خطا بکار رفته اســـت. جدول ۲ و جدول ۳ میزان خطا برای LEAM و RMAC را برای داده‌های مختلف نشـان می‌دهد. همانطور که مشاهده می‌شود میزان خطا تقریبا ثابت بوده و دقت LEAM ۸۹ درصد بالاتر از RMAC می‌باشد. در حالیکه نتایج نشان می‌دهد سرعت LEAM و RMAC تفاوت چندانی با یکدیگر ندارد اما LEAM توانسـته خطای RMAC را به میزان قابل توجهی کاهش دهد. در حقیقت RMAC محدوده مانتیس بیت ۱ و ۴ را به محدوده کوچکتر ۲ و ۴ نگاشـت می‌کند که این مسئله بر روی دقت تاثیر قابل توجهی دارد و حتی می‌تواند برای

[۱۱] Mean square error (MSE)

ضـرب‌های متوالی همچون ضـرب ماتریس‌ها به صـورت قابل توجهی منجر به افزایش دقت شـود. در این شرایط RMAC مجبور به استفاده از سیستمی جهت کنترل مداوم خطا در زمان اجرا می‌باشد و همین سیستم نیز سربار اضافه ای را نیز تحمیل می‌کند. اما نیاز به اسـتفاده از این سـیسـتم برای LEAM با توجه به خطای بسیار پایین آن کمتر است. در نتیجه از LEAM می‌توان برای کاربردهای متنوعی بدون نگرانی از افزایش نمایی خطا بهره برد. از سـوی دیگر همانطور که RMAC نیز نشان داده است، LEAM به دلیل جایگزینی ضرب مانتیس‌ها با جمع انرژی کمتری در مقایسه با ضرب معمولی مصرف می‌کند.

جدول ۲: مقایسه خطای LEAM و RMAC در DP

تعداد عناصر/۱۰۰۰	LEAM	RMAC
۱۰۰	۱۲۸/۹	۱۱۷۵/۶۴
۲۰۰	۱۲۷/۷۷	۱۱۶۷/۳
۳۰۰	۱۲۸/۰۹	۱۱۶۸/۶۳
۴۰۰	۱۲۷/۸۸	۱۱۶۵/۱۹
۵۰۰	۱۲۸/۱۴	۱۱۶۶/۸۲
۶۰۰	۱۲۸/۹	۱۱۷۱/۲۷
۷۰۰	۱۲۸/۰۵	۱۱۶۷/۶۴
۸۰۰	۱۲۸/۴۱	۱۱۶۹/۶۷
۹۰۰	۱۲۸/۰۳	۱۱۶۷/۵۹

جدول ۳: مقایسه خطای LEAM و RMAC در SP

تعداد عناصر/۱۰۰۰	LEAM	RMAC
۱۰۰	۱۲۸/۱۶	۱۱۶۸/۷۴
۲۰۰	۱۲۷/۹۶	۱۱۶۷/۵۶
۳۰۰	۱۲۸/۶۴	۱۱۷۲/۷۷
۴۰۰	۱۲۸/۱۱	۱۱۶۷/۵۹
۵۰۰	۱۲۸/۲۵	۱۱۶۹/۱۱
۶۰۰	۱۲۸/۵۰	۱۱۷۰/۷۲
۷۰۰	۱۲۸/۷۸	۱۱۷۱/۹۶
۸۰۰	۱۲۸/۶۲	۱۱۷۰/۷۳
۹۰۰	۱۲۸/۳۳	۱۱۷۰/۷۸
۱۰۰۰	۱۲۸/۵۲	۱۱۷۱/۵۰

۶- نتیجه گیری

این مقاله الگوریتم جدیدی با نام LEAM برای ضرب تقریبی معرفی کرد که با استفاده از دستورات SIMD در نرم‌افزار پیاده‌سازی شده است. الگوریتم جدید می‌تواند همزمان با حفظ محدوده‌ی مانتیس‌ها مانند ضرب معمولی در استاندارد IEEE 754، ضرب مانتیس‌ها که زمان و انرژی زیادی مصرف می‌کند، حذف کرده و آن‌ها را با جمع جایگزین کند. در مقایسه با روش RMAC، LEAMدارای خطای ۸۹٪ کمتری از RMAC است. همچنین RMAC از تکنیکی در زمان اجرا جهت کنترل خطاهای بالا استفاده می‌کند و زمانی که خطا بیشتر از حد آستانه‌ای شود از ضرب معمول استفاده می‌کند. اما با توجه به این که LEAMدارای خطای به مقدار قابل توجه کمتری نسبت به RMAC است، می‌تواند در بیشتر شرایط مورد استفاده قرار گیرد. همچنین تجزیه و تحلیل زمان اجرایی نشان داد که سرعت LEAM در مواردی در حدود ۲ درصد بهتر و در موارد دیگری به همین میزان کمتر از RMAC است.

۷- مراجع

[1] Sampson, A., et al., *EnerJ: Approximate data types for safe and general low-power computation.* ACM SIGPLAN Notices, 2011. **46**(6): p. 164-174.

[2] Anderson, A., S. Muralidharan, and D. Gregg, *Efficient multibyte floating point data formats using vectorization.* IEEE Transactions on Computers, 2017. **66**(12): p. 2081-2096.

[3] Flegar, G., et al., *Floatx: Ac++ library for customized floating-point arithmetic.* ACM Transactions on Mathematical Software (TOMS), 2019. **45**(4): p. 1-23.

[4] Gao, M. and G. Qu, *Estimate and recompute: A novel paradigm for approximate computing on data flow graphs.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018. **39**(2): p. 335-345.

[5] Grützmacher, T. and H. Anzt. A modular precision format for decoupling arithmetic format and storage format. in Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers 24. 2019. Springer.

[6] Grützmacher, T., et al., *Acceleration of PageRank with customized precision based on mantissa segmentation.* ACM Transactions on Parallel Computing (TOPC), 2020. **7**(1): p. 1-19.

[7] Grützmacher, T., et al., *A customized precision format based on mantissa segmentation for accelerating sparse linear algebra.* Concurrency and Computation: Practice and Experience, 2020. **32**(15): p. e5418.

[8] Ghabeli, H., et al., *Variable latency carry speculative adders with input-based dynamic configuration.* Computers & Electrical Engineering, 2021. **93**: p. 107247.

[9] Wang, E ,.et al., Deep neural network approximation for custom hardware: Where we've been, where we're going. ACM Computing Surveys (CSUR), 2019. **52**(2): p. 1-39.

[10] Imani, M., et al. RMAC: Runtime configurable floating point multiplier for approximate computing .in Proceedings of the international symposium on low power electronics and design. 2018.

[11] Imani, M., et al. Acam: Approximate computing based on adaptive associative memory with online learning. in Proceedings of the 2016 International Symposium on Low Power Electronics and Design. 2016.

[12] Imani, M., et al. CANNA: Neural network acceleration using configurable approximation on GPGPU. in 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC). 2018. IEEE.

[13] Ahmed, S.E., S. Kadam, and M. Srinivas. An iterative logarithmic multiplier with improved precision. in 2016 IEEE 23nd symposium on computer arithmetic (ARITH). 2016. IEEE.

[14] Imani, M., et al. ApproxLP: Approximate multiplication with linearization and iterative error control .in Proceedings of the 56th Annual Design Automation Conference 2019. 2019.

[15] Chen, C., et al. Optimally approximated and unbiased floating-point multiplier with runtime configurability. in Proceedings of the 39th international conference on computer-aided design. 2020.

[16] Ansari, M.S., B.F. Cockburn, and J. Han, *An improved logarithmic multiplier for energy-efficient neural computing.* IEEE Transactions on Computers, 2020. **70**(4): p. 614-625.

[17] Jahanshahi, S., A.S. Molahosseini, and A.A.E. Zarandi, *uLog: a software-based approximate logarithmic number system for computations on SIMD processors.* The Journal of Supercomputing, 2023. **79**(2): p. 1750-1783.

[18] Tagliavini, G., et al. A transprecision floating-point platform for ultra-low power computing. in ۲۰۱۸ Design, Automation & Test in Europe Conference & Exhibition (DATE). 2018. IEEE.

[19] Gautschi, M., et al., *Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2017. **25**(10): p. 2700-2713.

[20] Wu, Y., et al., A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits. arXiv preprint arXiv:2301.12181, 2023.

[21] Imani, M., D. Peroni, and T. Rosing. CFPU: Configurable floating point multiplier for energy-efficient computing. in Proceedings of the 54th Annual Design Automation Conference 2017. 2017.

[22] Wang, M., et al. An optimized compression strategy for compressor-based approximate multiplier. in 2020 IEEE International Symposium on Circuits and Systems (ISCAS). 2020. IEEE.

[23] Van Toan, N. and J.-G. Lee, FPGA-based multi-level approximate multipliers for high-performance error-resilient applications. IEEE Access, 2020. **8**: p. 25481-25497.

[24] Xiao, W., C. Zhuo, and W. Qian .OPACT: optimization of approximate compressor tree for approximate multiplier. in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2022. IEEE.

[25] Kim, M.S., et al., *The effects of approximate multiplication on convolutional neural networks.* IEEE Transactions on Emerging Topics in Computing, 2021. **10**(2): p. 904-916.

[26] Mitchell, J.N., *Computer multiplication and division using binary logarithms.* IRE Transactions on Electronic Computers, 1962(4): p. 512-517.

[27] Liu, C., J. Han, and F. Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. in 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2014. IEEE.

[28] Intel. *Intel AVX512-FP16 Architecture Specification* ۲۰۲۱ .[cited 2022 17 Agust]; Available from: https://software.intel.com/content/www/us/en/develop/download/intel-avx512-fp16-architecture-specification.html.

[29] Lomont, C. *Introduction to Intel Advanced Vector Extensions*. 2011 [cited 2022 17 Agust]; Available from: https://software.intel.com/content/dam/develop/external/us/en/documents/intro-to-intel-avx-183287.pdf.

[30] Intel. *Intel Intrinsics Guide*. 2021 [cited 2022 17 Agust]; Available from: https://software.intel.com/sites/landingpage/IntrinsicsGuide./