تاریخ دریافت مقاله: ۹۸/۰۶/۳۱
تاریخ پذیرش مقاله: ۹۹/۰۱/۲۴

بهبود کارایی توابع چندرسانه‌ای با استفاده از برنامه‌نویسی SIMD

اسدالله شاه بهرامی*
دانشیار گروه مهندسی کامپیوتر، دانشکده فنی، دانشگاه گیلان، رشت، ایران
پست الکترونیکی: shahbahrami@guilan.ac.ir

حسین امیری
دانشجوی کارشناسی ارشد گروه مهندسی کامپیوتر، دانشکده فنی، دانشگاه گیلان، رشت، ایران
پست الکترونیکی: hossein.amiri.papers@gmail.com

مریم مرادی‌فر
دانشجوی کارشناسی ارشد گروه مهندسی کامپیوتر، دانشکده فنی، دانشگاه گیلان، رشت، ایران
پست الکترونیکی: maryam.moradifar@gmail.com

چکیده:

با پیشرفت فنّاوری و تولید داده‌های چندرسانه‌ای با کیفیت، پردازش برنامه‌های مبتنی بر داده‌های چندرسانه‌ای به‌عنوان یک امر مهم تلقی می‌شود. الگوریتم‌های این حوزه پردازش‌های زیادی را بر روی داده‌های حجیم چندرسانه‌ای اعمال می‌کنند. از این رو، عملیات چندرسانه‌ای بسیار زمانبر است و بهبود کارایی آن‌ها یک چالش بزرگ در توسعه برنامه‌های چندرسانه‌ای به حساب می‌آید. یکی از فنّاوری‌هایی که در زمینه بهبود کارایی الگوریتم‌ها و توابع چندرسانه‌ای مورد استفاده قرار می‌گیرد، فنّاوری یک دستورالعمل و چند داده (SIMD) است که می‌تواند یک عمل را بر روی تعدادی داده در ثبّات‌های پردازنده به صورت برداری و همزمان انجام دهد. در این مقاله مروری مختصر بر مفاهیم چندرسانه‌ای، فنّاوری SIMD، مدل برنامه‌نویسی اینترینزیک (IPM) جهت بردارسازی صریح و بردارسازی خودکار کامپایلر (CAV) به‌عنوان راهکار بردارسازی غیرصریح صورت می‌گیرد. همچنین تعدادی از توابع چندرسانه‌ای با استفاده از IPM پیاده‌سازی شده و در مقایسه با CAV مورد ارزیابی قرار می‌گیرد. نتایج پیاده‌سازی‌ها نشان می‌دهد حداکثر افزایش کارایی تا ۲۲/۶۷ در پیاده‌سازی IPM کامپایلر ICC، برای الگوریتم مجموع قدرمطلق تفاضل‌ها نسبت به پیاده‌سازی متوالی به‌دست می‌آید. از طرفی با وجود کارایی بالاتر بردارسازی IPM نسبت به CAV، استفاده از روش بردارسازی خودکار کامپایلر راحت‌تر است و توسعه این سبک بردارسازی برای فنّاوری SIMD بیشتر مورد توجه پژوهشگران است.

واژه‌های کلیدی: پردازش موازی، چندرسانه‌ای، موازی‌سازی سطح داده، یک دستورالعمل چند داده

۱- مقدمه

برنامه‌های چندرسانه‌ای و استانداردهای فشرده‌سازی

* نویسندهٔ مسئول

مـــورد اســـتفاده در آن‌هـــا مانـنـــد JPEG، JPEG2000 MPEG-2/4 و H26x به عنوان یکی از مهم‌ترین برنامه‌های حال حاضر در دسترس عموم کاربران قرار گرفته‌اند. این برنامه‌ها ویژگی‌های خاص خود از جمله، قابلیت استفاده از موازی‌سازی دانه ریز[1] و دانه درشت[2]، سازمان‌دهی مجدد داده‌ها[3]، حلقه‌های کوچک، نیاز به حافظه با پهنای باند بالا و انواع داده‌ای کوچک را دارا می‌باشند. این ویژگی‌ها باعث می‌شـــود که پردازش توابع[4] و هســـته‌های[5] چندرسانه‌ای در پردازنده‌های همه منظـــوره[6] (GPPs) با محدودیت‌هایی مواجه شـــود. برای نمونه هر پیکسل هشـــت بیتی در یک تصویر خاکستری، معمولاً دوازده بیت فضای محاسباتی نیـــاز دارد، در حالی که اندازه ثبّات‌ها بزرگ‌تر از این مقدار است [۱]. برای پردازش‌های چندرسانه‌ای، در پردازنده‌های همه منظوره قابلیت‌هایی گنجانده شده تا با استفاده از آن بتوان داده‌های بیشـــتری را در یـــک ثبّات پردازش نمود و برنامه‌های چندرسانه‌ای را به صورت کارآمدتر اجرا کرد. یکی از این قابلیت‌ها استفاده از فنّاوری یک دستورالعمل و چندین داده[7] (SIMD) است [۲].

فنّاوری SIMD می‌تواند با اجرای یک دســـتورالعمل بر روی چندین داده، که در ثبّات‌های برداری[8] ذخیره می‌شوند، کارایی برنامه‌های محاســـباتی را به نحو چشـــمگیری بالا ببرد. اســـتفاده از فنّاوری SIMD برای موازی‌ســـازی یک راهکار بهینه محســـوب می‌شـــود، چرا که یک جزء اصلی پردازنده‌ها است و سربار کمتری نسبت به بقیه روش‌های موازی‌سازی مانند اجرای چندین دستورالعمل چندین داده[9] دارد. از این گذشـــته مصرف انـــرژی کمتری را نیز در پی خواهد داشـــت و می‌تواند با دیگر روش‌های موازی‌سازی به صورت ترکیبی اســـتفاده شود. با در نظر گرفتن اهمیت استفاده از این فنّاوری شرکت‌های سخت‌افزاری در هر نسل از این فنّاوری، قابلیت‌های جدیدی را در آن می‌گنجانند [۳].

شـــرکت اینتل به‌عنوان یکی از بزرگ‌ترین شرکت‌های تولید کننده پردازنده در ســـال ۲۰۱۱ با معرفی بسط بردار پیشـــرفته[10] (AVX) گام بزرگی در ارتقاء سرعت و کارایی پردازنده‌ها برداشت. در AVX اندازه ثبّات‌ها دو برابر نسل قبل یعنی ۲۵۶ بیت گردید و دســـتورهای جدیدی نیز برای بهبـــود کارایی در آن گنجانده شـــد. برای اســـتفاده از این قابلیت‌ها، راهکارهای برنامه‌نویســـی متعـــددی مانند مدل برنامه‌نویسی اینترینزیک[11] (IPM) و بردارسازی خودکار کامپایلر[12] (CAV) وجود دارد که IPM به‌عنوان بردارسازی صریـــح[13] و CAV به‌عنـــوان بردارســـازی غیرصریح قابل اســـتفاده است. هرچند استفاده از IPM در مقایسه با دیگر مدل‌های برنامه‌نویســـی قابل استفاده در زبان C، سخت‌تر به نظر می‌رســـد اما زمانـــی که کارایی برنامـــه مهم‌تر از راحتی برنامه‌نویسی باشـــد این مدل می‌تواند به بالاترین میزان ممکن از فنّاوری SIMD گنجانده شـــده بهره ببرد و باعث بهبود ســـرعت اجرای برنامه‌ها گردد. از این گذشته این کتابخانه به‌عنوان پیش‌فرض در کامپایلرهای مهم مانند کامپایلـــر C اینتل[14] (ICC)، مجموعه کامپایلر گنو[15] (GCC) و ماشین مجازی ســـطح پایین[16] (LLVM) گنجانده شده است [۴، ۵]. پژوهش‌های مرتبـــط در زمینه بهبود کارایی توابع مختلف چندرسانه‌ای صورت گرفته است. یکی از توابع پر استفاده و مهم ترانهاده کردن ماتریس است که بردارسازی آن با AVX برای اعداد اعشـــاری و ابعاد ماتریس کوچک با اســـتفاده از دستورهای مرتب سازی انجام شده است [۹]. در این مقاله ترانهاده ماتریس با مجموعه دستورهای جدید جمع‌آوری‌کننـــده[17] که در AVX2 معرفی شـــده برای اعداد صحیح و اعشـــاری به صورت صریح با IPM بردارسازی

1- Fine-grained Parallelism
2- Coarse-grained
3- Data Reorganization
4- Kernels
۵- منظور از هســـته، تابعی از یک برنامه چندرســـانه‌ای است که بیشترین زمان اجرای آن برنامه را به خود اختصاص داده است. در این مقاله عنوان تابع برای هسته استفاده می‌شود.
6- General Purpose Processors (GPPs)
7- Single Instruction Multiple Data (SIMD)
8- Vector Registers
9- Multiple Instruction Multiple Data (MIMD)
10- Advanced Vector Extensions (AVX)
11- Intrinsic Programming Model (IPM)
12- Compiler's Automatic Vectorization (CAV)
13- Explicit
14- Intel C Compiler (ICC)
15- GNU Compiler Collection (GCC)
16- Low Level Virtual Machine (LLVM)
17- Gather

شده است. تابع پرکاربرد دیگر فیلتر پاسخ ضربه محدود[18] است که در [۱۱] با سری دستورات بسط یک دستورالعمل و چندین داده جریانی[19] و به وسیله زبان ماشین بردارسازی شده است. در این پژوهش فیلتر پاسخ ضربه محدود به روش دیگری با استفاده از IPM به دستورات AVX2 نگاشت شده است. همچنین بردارسازی توابع دیگر چندرسانه‌ای مورد توجه پژوهشگران واقع شده که به صورت مفصل در بخش کارهای مرتبط تشریح می‌شود. با در نظر گرفتن این موضوع که راهکارهای مختلفی جهت بردارسازی برنامه‌ها وجود دارد، جهت رفع نیازهای خاص حوزه چندرسانه‌ای کدامیک از این راهکارها می‌توانند الف) با هزینه بردارسازی کمتر و ب) میزان کارایی بیشتر استفاده شوند؟

در این مقاله مروری مختصر بر مفاهیم پایه محیط چندرسانه‌ای و فنّاوری یک دستورالعمل و چند داده صورت می‌گیرد. علاوه بر این تعدادی از توابع چندرسانه‌ای با استفاده از راهکارهای مختلف بردارسازی، شامل مدل برنامه‌نویسی اینترینزیک (IPM) جهت بردارسازی صریح و بردارسازی خودکار کامپایلر (CAV) جهت بردارسازی غیرصریح، به همراه چند نمونه از فنّاوری‌های SIMD شرکت اینتل از جمله SSE4 و AVX2 بهبود داده شده‌اند. ارزیابی بر روی یک هسته از پردازنده نسل ششم اینتل با استفاده از کامپایلرهای GCC، ICC و LLVM صورت گرفته است. نتایج پیاده‌سازی‌ها نشان داد که میزان کارایی فنّاوری IPM-AVX2 نسبت به IPM-SSE4 برای ماتریس‌های با اندازه کوچک به دلیل تأثیر کمتر گلوگاه حافظه‌ای[20] و ناچیز بودن فقدان حافظه نهان به مراتب بالاتر از ماتریس‌های با اندازه بزرگ است. بیشترین افزایش کارایی تا ۲۲/۶۷ برای الگوریتم مجموع قدرمطلق تفاضل‌ها نسبت به پیادهسازی متوالی در پیاده سازی IPM کامپایلر ICC برای اندازه ماتریس ۵۱۲×۵۱۲ به دست آمده است. همچنین شاهد رفتارهای متفاوت کامپایلرها در برخورد با هر تابع بودیم. کامپایلرهای ICC و GCC می‌توانند با استفاده از دستورهای با کاربرد خاص[21] (SPIs) نسبت به LLVM بردارسازی مؤثرتری داشته باشند.

ادامه مقاله به صورت زیر سازمان‌دهی شده است. در بخش دو مفاهیم اولیه شامل برنامه‌های چندرسانه‌ای، توابع چندرسانه‌ای، مفاهیم پایه SIMD، بسط بردار پیشرفته و مدل‌های برنامه‌نویسی SIMD بیان می‌شوند. در بخش سه کارهای مرتبط گردآوری و بحث شده‌اند. تعدادی از بردارسازی‌های صریح با استفاده از مدل برنامه‌نویسی اینترینزیک (IPM) در بخش چهار بیان گردیده است. در بخش پنج محیط ارزیابی و نتایج آزمایشگاهی به تصویر کشیده شده است. در نهایت مقاله با نتیجه‌گیری در بخش شش به اتمام می‌رسد.

۲- مفاهیم اولیه

در این قسمت مفاهیم پایه‌ای محیط‌های چندرسانه‌ای و فنّاوری SIMD به‌طور مختصر معرفی می‌شوند.

۲-۱ برنامه‌های چندرسانه‌ای

برنامه‌های کاربردی چندرسانه‌ای[22] به‌عنوان یکی از برجسته‌ترین قسمت‌های پر کار یک سیستم رایانه‌ای محسوب می‌شوند که در بسیاری از سیستم‌ها از جمله رایانه‌های شخصی، گوشی‌های تلفن همراه و غیره مورد استفاده قرار می‌گیرند. گستردگی و اهمیت موضوع باعث شده تا از پردازش‌های مؤثر[23] و کارای برنامه‌های چندرسانه‌ای به عنوان یک زمینه پر چالش در حوزه پردازش رسانه‌ها یاد شود. برای حوزه خاص[24] مانند تصاویر ماهواره‌ای، هواشناسی و غیره پردازنده‌های خاص تدارک دیده شده‌اند اما دسترسی همه افراد به این‌گونه پردازنده‌ها مهیا نبوده و مهم‌تر این‌که پردازنده برای یک حوزه معین به صورتی تغییر داده شده که ممکن است یک پردازش ساده از حوزه دیگر را به صورت بهینه

18- Finite Impulse Response (FIR)
19- Streaming SIMD Extensions 2 (SSE2)
20- Memory Bottleneck
21- Special Purpose Instructions (SPIs)
22- Multimedia Applications (MMAs)
23- Efficient
24- Domain-specific

انجام ندهد. از این رو بر روی سیستم‌هایی که در دسترس عموم قرار می‌گیرد پردازنده‌های همه منظوره قرار می‌گیرد تا با استفاده از امکان برنامه‌ریزی[۲۵] بهبود کارایی[۲۶] محقق گردد و برای کاربردهای چندرسانه‌ای نیز امکاناتی مانند واحدهای SIMD در آن‌ها تعبیه شده باشد. دستورالعمل‌های موجود در فنّاوری‌های مختلف SIMD برای کاربردهای گوناگون در زمینه‌هایی از جمله رفع گلوگاه محاسباتی برنامه‌ها، سریع‌تر نمودن برنامه‌های چندرسانه‌ای، بهبود الگوریتم‌های حساس به محاسبه و غیره به کار می‌رود [٦].

برخی از ویژگی‌های برنامه‌های چندرسانه‌ای عبارتند از:

• قابلیت استفاده از موازی‌سازی دانه ریز و دانه درشت، معمولاً برنامه‌های چندرسانه‌ای عملیات یکسانی بر روی داده‌های متفاوت انجام می‌دهند و عموماً وظایف بر روی مقادیر مستقل صورت می‌پذیرند. از آنجایی که این عملیات از یکدیگر مستقل هستند زمینه را برای استفاده از SIMD و موازی‌سازی در سطح ریسه[۲۷] فراهم می‌کنند.

• سازمان‌دهی مجدد داده‌ها: یک برنامه چندرسانه‌ای مکرراً نیاز به تغییر چینش داده‌ها[۲۸] دارد که دستورالعمل‌های متنوعی در مجموعه دستورهای SIMD جهت تغییر چیدمان داده‌های درون ثبّات‌های برداری تعبیه شده است.

•حلقه‌های کوچک: برنامه‌ها و استانداردهای چندرسانه‌ای معمولاً از توابع مختلف تشکیل می‌گردند که اکثر توابع دارای حلقه‌های برنامه‌نویسی کوچک با تکرار زیاد هستند که بیشترین زمان اجرای توابع در این حلقه‌ها صرف می‌شود.

• نیاز به حافظه با پهنای باند بالا: این برنامه‌ها معمولاً بر روی حجم زیادی از داده‌ها عملیات محاسباتی مشابه و یکسان انجام می‌دهند. که نقل و انتقالات از حافظه به واحدهای پردازش و از پردازنده به حافظه می‌تواند بسیار پرهزینه باشد.

25- Programmability
26- Performance
27- Thread Level Parallelism (TLP)
28- Data Reordering

جدول ۱: برخی از استانداردهای فشرده‌سازی حوزه چندرسانه‌ای

کاربرد	استاندارد
فشرده سازی تصویر	JPEG, JPEG2000, BPG, PNG
فشرده سازی ویدئو	MPEG-(1,2,3,4), VP(8, 9), H26(1, 2, 3, 4/AVC, 5/HEVC)
فشرده سازی صدا	MP3, AAC, FLAC, AMR

• انواع داده‌ای کوچک: داده‌های برنامه‌های چندرسانه‌ای اغلب داده‌های صحیح هشت و شانزده بیتی هستند که به صورت میانگین ٤٠٪ از داده‌های هشت بیتی، ٥١٪ از داده‌های شانزده بیتی و ٩٪ از داده‌های ٣٢ بیتی استفاده می‌کنند. علاوه بر این، برنامه‌های حوزه چندرسانه‌ای معمولاً عملیات محاسباتی بر روی اعداد صحیح انجام می‌دهند و محاسبات اعشاری برای برخی از برنامه‌های خاص این حوزه نیاز است. به دلیل اهمیت این موضوع اغلب تعداد واحدهای محاسبات و منطق[۲۹] (ALU) برای اعداد صحیح به مراتب بیشتر از تعداد واحدهای محاسباتی اعداد اعشاری است.

چندین استاندارد جهت فشرده‌سازی تصویر، ویدئو و صدا در جدول (١) گردآوری شده‌اند. همچنین برخی از برنامه‌های کاربردی حوزه چندرسانه‌ای نیز در جدول (٢) گردآوری شده‌اند که برای ویرایش حرفه‌ای تصاویر، ویدئو، ساخت انیمیشن و غیره استفاده می‌شوند.

۲-۲ توابع چندرسانه‌ای

برخی از توابع پرکاربرد در استانداردهای چندرسانه‌ای و برنامه‌های کاربردی این حوزه در جدول ٣ گردآوری شده‌اند که در ادامه توضیح مختصری برای هرکدام ارائه می‌گردد [٨-١٠].

عملیات ماتریسی[۳۰] یکی از پر کاربردترین عملیات در پردازش سیگنال، تصویر، ویدئو، صدا، تبدیل سه بعدی[۳۱] و گرافیک رایانه‌ای به حساب می‌آیند. سه نمونه از عملیات مبنایی بر روی ماتریس‌ها جمع ماتریس[۳۲]، ترانهاده

29- Arithmetic Logic Unit (ALU)
30- Matrix Operations
31- 3D Rendering
32- Matrix Addition

جدول ۲: برخی از برنامه‌های کاربردی حوزه چندرسانه‌ای

کاربرد	نام برنامه
ویرایش تصویر	Adobe Photo Shop, GIMP, AVS Photo Editor
ویرایش ویدئو	Adobe Premiere, Corel Video Studio, Speed Video Splitter
ویرایش صدا	Adobe Audition, FL Studio, Ashampoo Music Studio
ساخت انیمیشن	Adobe Falsh, 3D MAX, Maya
پخش کننده	JetAudio, Kmplayer, Real Player, QuickTime Player, Com Player, VLC

ماتریس[۳۳] و ضرب ماتریس[۳۴] است که هر کدام الگوریتم مخصوص به خود را دارند. برای نمونه، الگوریتم‌های متنوعی برای ضرب ماتریس معرفی شده‌اند. یکی از آن‌ها الگوریتم ضرب ماتریس در ترانهاده ماتریس است که برای بهبود دسترسی‌ها به حافظه و رفع نیاز الگوریتم پایه برای خواندن از نشانی‌های ناپیوسته موجود در حافظه به وجود آمده است.

فیلتر پاسخ ضربه محدود[۳۵] یکی از مهم‌ترین الگوریتم‌ها در حوزه پردازش چندرسانه‌ای است. محاسبات این فیلتر بر مبنای مفهوم پیچش انجام می‌شود که ضرایب را در مقادیر خروجی ضرب می‌کند و حاصل جمع را در یک عنصر خروجی ذخیره می‌کند [۱۱، ۱۲]

تخمین حرکت دو تصویر یکی از اصلی‌ترین مفاهیم حوزه پردازش ویدئو است که در فشرده‌سازی ویدئو، تثبیت ویدئو، تراز کردن تصویر و تبدیل افزایشی نرخ قاب تصاویر کاربرد دارد. یکی از گسترده‌ترین کاربردهای تخمین حرکت در فشرده‌سازی ویدئو است که بر اساس کاهش افزونگی زمانی در تصاویر مشابه استوار است. چرا که شدت و رنگ در مسیر حرکت تصویر افزونگی ایجاد می‌کند و تخمین یا محاسبه آن می‌تواند به فشرده‌سازی هر چه بیشتر تصویر کمک کند. یکی از روش‌های ساده اما پرکاربرد این است که بلوک‌های قاب قبلی با قاب کنونی مقایسه شوند و بهترین بردار حرکت برای هر بلوک را تخمین بزنند. مجموع قدرمطلق تفاضل‌ها[۳۶] گسترده‌ترین راهکار مورد استفاده برای انجام این عملیات است که تفاوت پیکسل‌های متناظر موجود در قاب را به دست می‌آورد و با مقایسه آن کمترین میزان خروجی را به دست می‌آورد که بیانگر بهترین بردار حرکت است [۱۳]. راهکار دیگر مبتنی بر بلوک، مجموع مربعات تفاضل‌ها[۳۷] است که به طریقی مشابه می‌تواند جهت تخمین حرکت مورد استفاده قرار گیرد. با توجه به حجم بالای محاسبات برای پیدا کردن دو بلوک با شباهت بیشتر هر دو راهکار بسیار زمان‌بر هستند [۱۴].

۲-۳ مفاهیم پایه و روند توسعه SIMD

فنّاوری SIMD به این صورت است که در آن یک دستورالعمل مقادیر مختلف داده را به صورت برداری در ورودی دریافت می‌کند و پس از انجام دستورالعمل آن را به صورت برداری[۳۸] در خروجی قرار می‌دهد و با استفاده از افزایش پهنای محاسبات سعی بر افزایش سرعت برنامه دارد. این فنّاوری در مقابل مدل قدیمی‌تر که با نام یک دستورالعمل یک داده[۳۹] شناخته می‌شود به وجود آمده است. پردازنده‌هایی که امروزه مورد استفاده قرار می‌گیرند و حتی پردازنده‌های موجود در گوشی‌های تلفن همراه نیز به نحوی از فنّاوری SIMD استفاده می‌کنند و دستورالعمل‌های مخصوص به خود را دارند.

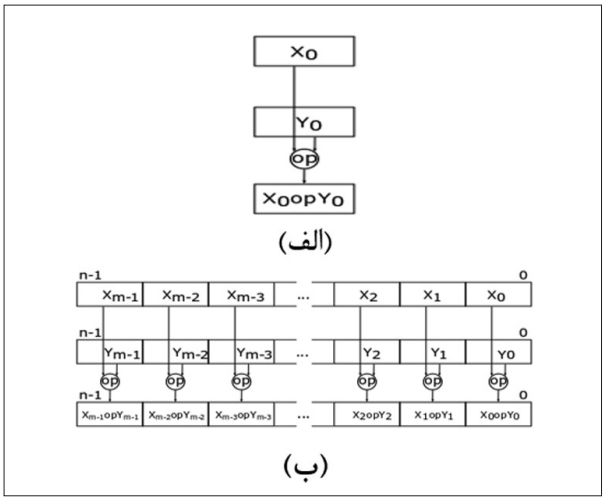
شکل ۱ (الف) نحوه انجام یک عملیات ساده با استفاده از مدل SISD را نشان می‌دهد، به این صورت که یک مقدار موجود در یک ثبّات با یک مقدار موجود در ثبّات دیگر در عملیات شرکت می‌کنند و نتیجه در ثبّات مقصد ذخیره می‌گردد. در مقایسه با آن شکل ۱ (ب) مدل SIMD را به تصویر می‌کشد. همان‌طور که مشاهده می‌شود در این مدل m مقدار یا m قلم داده‌ای در ثبّات‌های مبدأ n بیتی وجود

33- Matrix Transposition
34- Matrix Multiplication
35- Finite Impulse Response (FIR)

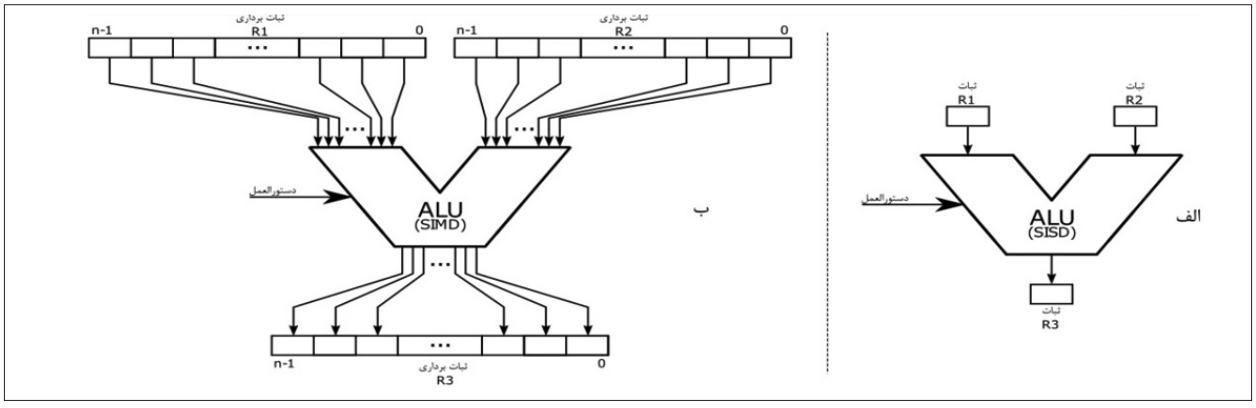
36-Sum of Absolute Differences (SAD)
37-Sum of Squared Differences (SSD)
38-Vector
39-Single Instruction Single Data (SISD)

جدول ۳: برخی از توابع پرکاربرد حوزه چندرسانه‌ای

توابع چندرسانه‌ای	کاربرد	تعداد حلقه
عملیات ماتریسی	پردازش تصویر، ویدئو و صدا رندرینگ سه بعدیگرافیک کامپیوتری	۲-۳
فیلتر پاسخ ضربه محدود	انواع فیلترها در تصویر، ویدئو و صدا مانند فیلتر کردن صدا در استاندارد AMR. شناسایی الگو در پردازش سیگنال رقمی	۲-۳
تخمین حرکت	فشرده‌سازی ویدئو در استانداردهایی مانند MPEG و HEVC	۵



شکل ۱: انجام یک عمل ساده با استفاده از (الف) یک دستورالعمل یک داده(SISD) (ب) یک دستورالعمل چند داده (SIMD)



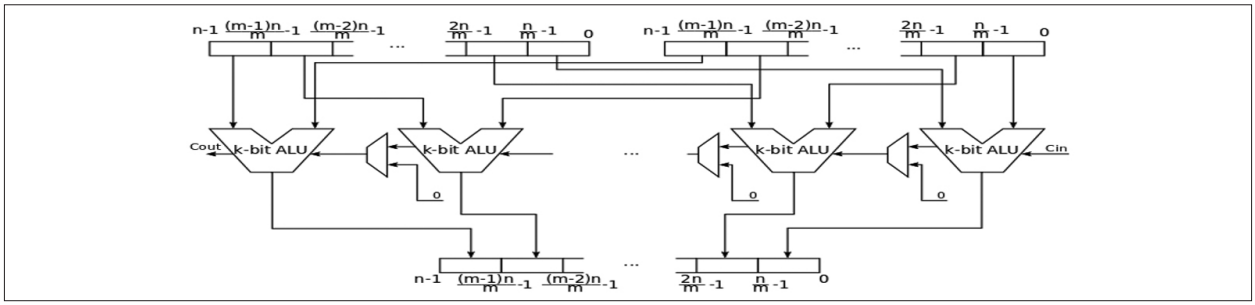
شکل ۲: مقایسه معماری (الف) SISD-ALU و (ب) SIMD-ALU

دارد و با انجام عملیات به صورت همزمان m بخش نتایج در ثبّات مقصد n بیتی که دارای m بخش اســت قرار داده می‌شود. یک معماری کلی از SIMD-ALU در مقایسه با یک معماری SISD-ALU در شــکل ۲ نمایش داده شده است. در شکل ۲ (الف) مقادیر مورد محاســبه در ثبّات‌های R2، R1 و R3 نگهــداری می‌شــوند. این ثبّات‌هــا در معماری SISD به‌عنوان ثبّات‌های متوالی نام‌گذاری می‌شــوند که توانایی ذخیره‌سازی یک مقدار داده را دارند.

شکل ۲ (ب) یک معماری ساده از SIMD را نشان می‌دهد. از یک ســو، ثبّات‌های برداری پهنای بیتی بسیار بیشتری دارنــد. می‌توان چنین در نظر گرفت که یک کلمه به تعدادی زیرکلمه[40] شکسته می‌شــود تا عملیات لازم بر روی آن‌ها انجام شود و سپس زیرکلمات در کنار یکدیگر قرار گیرند تا نتیجه را تشــکیل دهند. هر کــدام از زیرکلمات می‌توانند به مثابه یک مقدار در معماری SISD تلقی شــوند. از سوی دیگر، به علت حضور مقادیــر مختلف در یک ثبّات SIMD دسترسی‌های مکرر به حافظه کمتر می‌شود. از آنجایی که دسترسی به حافظه یک نیاز مهم اما پرهزینه است، SIMD می‌تواند باعث کاهش هزینه‌های سربار حافظه‌ای گردد چرا که با اســتفاده از یک دســتور SIMD مربوط به خواندن یا نوشتن حافظه می‌توان چندین زیرکلمه را به‌طور همزمان در یک ثبّات SIMD یا ثبّات برداری ذخیره کرد [۲].

شکل ۳ یک معماری کلی SIMD-ALU با ثبّات‌های n بیتی که قابلیت بخش‌بندی به m بخش را دارند نشــان می‌دهد. همان‌طور که در شــکل مشــاهده می‌گردد ســاختار ALU n بیتــی طوری طراحی و پیاده‌ســازی می‌گردد که قابلیت

40-Sub-words



شکل ۳: یک معماری کلی SIMD-ALU با ثبّات‌های n بیتی قابل بخش‌بندی به زیربخش‌های کوچک‌تر

انجام عملیات محاسباتی و منطقی بر روی تعداد بخش‌های مختلفی از ثبّات‌ها بر حسب نوع دستورهای SIMD را دارد که این عمل در شکل با انتقال بیت نقلی یا صفر از ALU،k بیتی $\left(\frac{n}{m} = k\ bit\right)$ به مرحله بعد نشان داده شده است. با استفاده از این معماری در پردازنده‌های با کاربری عمومی موازی‌سازی در سطح داده استخراج می‌شود تا از طرفی مفهوم پردازنده همه منظوره حفظ شود و از طرف دیگر پردازنده از مزایای SIMD بهره‌مند گردد.

در ابتدای روند توسعه SIMD، بسط چندرسانه‌ای[41] یک راهکار قوی برای بالا بردن سرعت برنامه‌های چندرسانه‌ای و برنامه‌های حوزه ارتباطات که محاسبات زیاد بر روی قطعه داده‌ای کوچک انجام می‌دادند محسوب می‌شد. فنّاوری MMX می‌توانست محاسبات SIMD را بر روی داده‌های صحیح از نوع بسته‌های[42] ۸، ۱۶، ۳۲ و ٦٤ بیتی از اعداد صحیح انجام دهد. انواع داده‌ای در ثبّات‌های ٦٤ بیتی ذخیره می‌شدند و می‌توانست هشت بسته ۸ بیتی، چهار بسته ۱٦ بیتی، دو بسته ۳۲ بیتی و یک عدد ٦٤ بیتی از اعداد صحیح را در خود نگه دارد [۱۵]. فنّاوری MMX به شدت مورد استقبال برنامه‌نویسان و پژوهشگران قرار گرفت اما مشکلاتی از قبیل یکی بودن ثبّات‌های مورد استفاده MMX و واحد ممیز شناور[43]، عدم پشتیبانی از نوع داده اعشاری و غیره باعث تولید فنّاوری بعدی یعنی SSE[44] گردید.

فنّاوری SS E برای داده‌های از نوع اعشاری به وجود آمد و می‌توانست عملیات زیادی بر روی بسته‌های ۳۲ بیتی انجام دهد. ثبّات‌ها به صورت مجزا در نظر گرفته شدند و می‌توانستند به صورت هم‌زمان با واحد ممیز شناور مورد استفاده قرار گیرند. اندازه این ثبّات‌ها ۱۲۸ بیتی است که می‌توانند هم‌زمان چهار بسته ۳۲ بیتی اعشاری را در خود نگه دارند. با توجه به عدم پشتیبانی از داده‌های نوع صحیح با اندازه ثبّات‌های ۱۲۸ بیتی و تنها داشتن یک نوع اعشاری که دقت کمی را در کاربردهای مهندسی و غیره فراهم می‌کند فنّاوری بعد یعنی SSE2 بر روی همین ثبّات‌ها معرفی شد. فنّاوری SSE2 توانایی انجام محاسبات بر روی انواع داده‌های مورد نیاز برنامه‌ها را دارا است و می‌تواند علاوه بر انواع داده‌ای SSE بسته‌های ٦٤ بیتی از نوع اعشاری بسته‌های ۸، ۱٦، ۳۲ و ٦٤ بیتی از نوع صحیح را استفاده کند. این فنّاوری دستورالعمل‌هایی برای محاسبات بر روی اعداد صحیح به صورت برداری را فراهم آورده تا کاستی‌های فنّاوری‌های قبلی رفع گردد. با توجه به وجود بسته‌های ٦٤ بیتی از نوع اعشاری، SSE2 را می‌توان علاوه بر کاربردهای چندرسانه‌ای و برنامه‌های حوزه ارتباطات در برنامه‌های محاسباتی با دقت بالا، برنامه‌های حوزه مهندسی و غیره نیز استفاده کرد. با فراگیر شدن استفاده از این فنّاوری در حوزه‌های مختلف و نیاز به دستورالعمل‌های جدید برای رفع گلوگاه‌های محاسباتی فنّاوری SSE3 بر روی همین ثبّات‌ها معرفی گردید.

فنّاوری SSE3 برنامه‌نویس را قادر می‌سازد تا محاسبات افقی[45] بر روی ثبّات‌ها انجام دهد تا نیازی به

41- Multimedia Extensions (MMX)
42- Pack
43- Floating Point Unit (FPU)
44- Streaming SIMD Extensions (SSE)

45- Horizontal Operations

درهم‌سازی[46] ثبّات‌ها برای انجام یک عمل در ثبّات خاص نباشد. همچنین دستورهایی اضافه گردید تا بتوان بدون این‌که نیاز به دستورالعمل‌های اضافه‌تری باشد، یک مقدار را در خانه‌های مختلف ثبّات کپی کرد. برای پوشش عملیات بیشتر بر روی انواع داده‌ای مختلف فنّاوری SSSE3[47] بر روی همین ثبّات‌ها معرفی شد. فنّاوری SSSE3 می‌تواند با انجام یک دستورالعمل، عملیات افقی را روی انواع داده‌های از نوع صحیح انجام دهد و در یک دستورالعمل توابع خاص ریاضی مانند قدر مطلق را انجام دهد و حتی با دریافت یک متغیر سوم، علاوه بر ثبّات‌ها، الحاقی دلخواه از ثبّات‌ها را به وجود آورد. این فنّاوری با استفاده از یک متغیر تعیین‌کننده نقطه شروعی بود برای تولید فنّاوری SSE4 که بر روی عملیات هندسی و مانند آن تمرکز دارد. فنّاوری SSE4 با افزودن محاسبات پیچیده ریاضی در یک دستورالعمل باعث افزایش سرعت محاسبات می‌شود. اغلب دستورها یک متغیر سوم را در نظر می‌گیرند که این متغیر تعیین کننده نحوه انجام محاسبات است. برای مثال با این فنّاوری می‌توان گردش حول یک محور خاص را پیاده‌سازی نمود، بدون این‌که نیاز به انجام درهم‌سازی‌های پیچیده‌ای باشد. توابع چندکاره ریاضی مانند گرد کردن[48] حول یک مقدار خاص، کف‌گیری[49] و غیره با این فنّاوری راحت‌تر و در تعداد چرخهٔ کمتری انجام می‌شوند. محاسبات پیچیده اغلب نیاز به پهنای محاسبات بیشتری دارند، از این رو این فنّاوری علاوه بر قوی‌تر نمودن مدل‌های قبلی نیاز به تغییر و ارتقاء به فنّاوری‌های بعدی را داشت که با معرفی فرضیاتی به نام SSE5 توسط شرکت AMD و در نهایت عرضه فنّاوری AVX با پهنای محاسباتی دو برابر یعنی ۲۵۶ بیت و بعد از آن فنّاوری AVX-512 با پهنای محاسباتی چهار برابر یعنی ۵۱۲ بیت فرصت‌های بیشتری برای افزایش کارایی به وسیله فنّاوری یک دستورالعمل و چندین داده به وجود آمد [۳].

46- Shuffling
47- Supplemental Streaming SIMD Extensions 3 (SSSE3)
48- Round
49- Floor

۲-٤ بسط بردار پیشرفته

فنّاوری بسط بردار پیشرفته با تمرکز بر روی اعداد اعشاری طراحی نرم‌افزارهای پیشرفته مهندسی را با کارایی بهتری انجام می‌دهد و موازی‌سازی در سطح ریسه با درجه متغیر را با کارایی بهتری ارائه می‌دهد [۱٦]. تأخیرها و بازده[50] دستورهای به کار رفته در AVX تقریباً به اندازه فنّاوری‌های پیشین است با این تفاوت که پهنای محاسبات دو برابر گردیده است. از این رو می‌توان انتظار داشت که در بسیاری از موارد AVX تقریباً دو برابر سریع‌تر از SSE4 عمل خواهد کرد. فنّاوری AVX2 نیز با تمرکز بر روی داده‌هایی از نوع عدد صحیح مورد استفاده در اغلب برنامه‌های چندرسانه‌ای، سعی بر بالا بردن کارایی و ارتقای فنّاوری‌های پیشین دارد. یکی از بهبودهایی که بسط بردار پیشرفته به وجود آورد استفاده از حداقل سه عملوند در دستورهای زبان ماشین است. با این قابلیت می‌توان دقیقاً c=a+b را شبیه‌سازی نمود. بدین صورت که عملوند اول به‌عنوان مقصد و محل نگهداری نتیجه در نظر گرفته می‌شود و دیگر عملوندها به‌عنوان عملوندهای شرکت کننده در عملیات محسوب می‌شوند [۱۷، ۱۸]. دستورهای موجود در بسط بردار پیشرفته برای افزایش کارایی و همچنین رفع نیازهای با مصرف پایین انرژی طراحی شده‌اند [۱۹، ۲۰].

۲-۵ مدل‌های برنامه‌نویسی SIMD

پس از به وجود آمدن بسط‌های فنّاوری یک دستورالعمل و چندین داده موجود در پردازنده‌ها، مدل‌های متنوع برنامه‌نویسی جهت به کار بردن فنّاوری‌ها برای اهداف خاص به وجود آمدند [٤]. مدل‌های برنامه‌نویسی SIMD را می‌توان در دو گروه مدل‌های صریح[51] و غیرصریح[52] تقسیم‌بندی کرد. مدل‌های صریح به آن‌هایی گفته می‌شود که برنامه‌نویس به صراحت از عملیات برداری استفاده می‌کند تا با استفاده از فنّاوری گنجانده شده در پردازنده کارایی را

50- Throughput
51- Explicit
52- Implicit

بالا ببرد. به‌عبارت دیگر تبدیل کدها و الگوریتم‌های سریال به عملیات برداری و نحوه نگاشت به فضای موازی‌سازی بر عهده برنامه‌نویس است. برای مثال مدل برنامه‌نویسی اینترینزیک[۵۳] (IPM) توابع متعددی برای پیاده‌سازی عملیات برداری در اختیار برنامه‌نویس قرار می‌دهد.

مدل‌های غیرصریح یا ضمنی به آن‌هایی گفته می‌شود که برنامه‌نویس برای بالا بردن کارایی به صراحت از عملیات برداری استفاده نمی‌کند، در عوض با به کار بردن قابلیت‌های نرم‌افزاری اقدام به استخراج فنّاوری SIMD می‌نماید. برای نمونه قابلیت بردارسازی خودکار کامپایلرها[۵۴] (CAV) و ابزار OpenMP از دستورالعمل‌های SIMD در زمان کامپایل و یا اجرا برای بالا بردن کارایی بهره می‌برند.

۲-۵-۱ مدل برنامه‌نویسی اینترینزیک

مدل برنامه‌نویسی اینترینزیک (IPM) به‌عنوان یک مرجع اصلی جهت استفاده از فنّاوری SIMD موجود در پردازنده‌ها استفاده می‌شود. توابع به کار رفته در این مدل به این صورت هستند که در برنامه عیناً وجود خواهند داشت، به جای این‌که از این توابع به‌عنوان یک ارجاع یاد شود و به آن‌ها پیوندی ایجاد شود. مجموعه دستورالعمل‌های مورد استفاده در پردازنده‌های اینتل در کتابخانه‌های مختلفی گردآوری شده‌اند که هر کتابخانه شامل توابع مورد استفاده در یک بسط و بسط‌های ماقبل آن است. برای مثال xmmintrin.h شامل کلیه توابع مورد نیاز MMX و SSE است و immintrin.h دربردارنده توابع مورد نیاز جهت نگاشت به دستورهای AVX2 است. در بسیاری از کامپایلرها کتابخانه پرکاربرد x86intrin.h قابل استفاده است. این کتابخانه همه موارد مورد نیاز برای برنامه‌نویسی فنّاوری یک دستورالعمل و چندین داده برای معماری x86 اینتل را در بر دارد و علاوه بر توابع SIMD، امکانات اضافه‌تری مانند خواندن شمارنده تکانهٔ پردازنده نیز وجود دارد. در این مقاله از کتابخانه x86intrin برای برنامه‌نویسی صریح SIMD استفاده شده است و از آن به‌عنوان مدل برنامه‌نویسی اینترینزیک یا IPM یاد می‌شود [۲۱].

هرچند استفاده از IPM می‌تواند راحت‌تر از دستورهای اسمبلی مورد استفاده قرار گیرد و با دانش بالای برنامه‌نویسی کارایی به شکل قابل توجهی بالا خواهد رفت اما استفاده از این مدل برنامه‌نویسی مشکلاتی از این قبیل رقم خواهد زد [٤]:

- قابلیت حمل برنامه برای بسط‌های متفاوت فنّاوری SIMD و سازگاری کدهای نوشته شده با یکدیگر، نگهداری و به روز رسانی کدها
- سختی برنامه‌نویسی با توابع اینترینزیک در مقایسه با دیگر راهکارها و بهینه‌سازی‌های ناکارآمد کامپایلر در برخی از پیاده‌سازی‌ها

مشکلات بیان شده و نیازمندی دنیای نرم‌افزار باعث شده تا روش‌های جایگزینی برای استفاده از فنّاوری یک دستورالعمل و چندین داده ، مانند بردارسازی خودکار در کامپایلر محبوبیت نسبی کسب نماید که در ادامه به نحوه استفاده از این قابلیت کامپایلرها خواهیم پرداخت.

۲-۵-۲- قابلیت‌های بردارسازی موجود در کامپایلرها

کامپایلرها برای ترجمهٔ کدهای نوشته شده در زبان سطح بالا به کدهای ماشین، استفاده می‌شوند. برای یک برنامه خاص، دنبالهٔ مختلف زبان ماشین می‌تواند به کار رود تا خروجی مورد نظر به دست آید. از آنجایی که برای هر پردازنده کدهای مخصوص به آن پردازنده در نظر گرفته شده است و تأخیر[۵۵] دستورها با یکدیگر متفاوت است، کارایی یک برنامه ترجمه شده به کدهای ماشین مختلف می‌تواند به صورت قابل ملاحظه‌ای متفاوت باشد. بنابراین کامپایلر مورد استفاده برای ایجاد کدهای ماشین کارآمدتر یک عامل مهم کارایی به حساب می‌آید. علاوه بر این، تکنیک‌های بهینه‌سازی کامپایلر برای تولید کدهای ماشین بهینه بسیار مهم و اساسی است.

بهینه‌سازی حلقه جهت کارآمدتر نمودن عملیات درون حلقه به انواع مختلفی از جمله، بازکردن حلقه[۵۶]، ترکیب

53- Intrinsic Programming Model (IPM)
54- Compiler's Automatic Vectorization (CAV)
55- Latency
56- Loop Unrolling

حلقه[۵۷]، تبـادل یا جابجایی حلقه[۵۸]، وارون‌سـازی حلقه[۵۹]، لولـه‌ای کردن نرم‌افزار[۶۰]، موازی‌سـازی و بردارسـازی تقسـیم می‌شـود. لوله‌ای کردن نرم‌افزار باعث می‌شـود که کدهای ماشـین مربوط به طبقـات[۶۱] مختلف واحدهای عملیاتی[۶۲] پردازنـده به صورتی که تأخیـر واحدها را به حداقل برسـاند، در یک دنباله قرار گیرند. باز کردن حلقه به این معناسـت که عملیات مربوط به چندین دور حلقه در یک چرخش حلقه گنجانده شـود و شمارنده حلقه متناسب با آن تغییر کند. برای موازی‌سازی، عملیات درون حلقه به نحوی تقسیم می‌شوند که بتوانند هم‌زمان در پردازنده‌های مختلف یا هسـته‌های مختلف یک پردازنده انجام شوند و در نهایت با پیوسـتن نتایج، خروجی کل عملیات به دست آید. بردارسـازی به این صورت اسـت که با اسـتفاده از دسـتورالعمل‌های SIMD، اعمال درون حلقه انجام شـود. برای این منظور دو گذر به نام‌های بردارسـازی در سطح حلقه[۶۳] و موازی‌سـازی در سـطح کلمهٔ مافوق[۶۴] به‌عنوان راهکارهـای اصلـی بردارسـازی مـورد اسـتفاده قرار می‌گیرند. گذر SLP یک نمونه محدود شـده موازی‌سازی در سـطح دستورالعمل[۶۵] اسـت که راه حل کم هزینه‌تر و کم مصرف‌تری محسـوب می‌شـود. گذر SLP برای ایجاد گره جدیدی از عبارت‌ها، دسترسـی‌های مجاور در حافظه را شناسـایی می‌کند، سپس آن‌ها را با توجه به طول کلمه مافـوق ادغام می‌کنـد و در نهایت با دسـتورهای SIMD جایگزین می‌کنـد [۲۲]. تمام این روش‌هـا تنها در کدهای ماشین ایجاد شده تغییر ایجاد می‌کنند و مقدار خروجی و الگوریتم به هیچ عنوان نباید تغییر کند.

۳- کارهای مرتبط

شباهت ذخیره‌سازی و محاسـباتی برنامه‌های حوزه چندرسـانه‌ای با مفهوم و سـاختار ماتریس‌ها باعث شده تا عملیات ماتریسـی و الگوریتم‌های بـه کار رفته در این زمینه برای پیاده‌سـازی بسـیاری از پردازش‌های حوزه چندرسـانه‌ای مورد اسـتفاده قرار گیرنـد. زمانبر بودن عملیات ماتریسی و نیاز به افزایش کارایی باعث جلب توجه بسیاری از پژوهشـگران برای افزایش کارایی آن‌ها شده اسـت. از آنجایی که معمولاً عملیات ماتریسـی به گونه‌ای هستند که محاسـبات می‌توانند مستقل از هم انجام شوند و همچنیـن یـک یا چند دسـتورالعمل به صـورت مکرر بر روی داده‌هـا اعمال گردد جهت افزایـش کارایی آن‌ها می‌توان از مفهوم موازی‌سازی در سطح داده منفعت برد. در الگوریتـم ترانهاده کردن ماتریس مـکان داده‌ها تغییر می‌کند، محاسبات ریاضی بر روی مقادیر ورودی صورت نمی‌گیـرد و از دسـتورهای مرتب‌سـازی داده‌هـا مانند درهم‌سازی[۶۶] و جایگشت کردن[۶۷] استفاده می‌شود. Zekri [۹] ترانهاده‌کردن ماتریس را با اسـتفاده از فنّاوری SIMD بیان نموده اسـت، که برای این منظور از دستورالعمل‌های بسط بردار پیشرفته استفاده می‌شود.

دیگر عملیات پرکاربرد ماتریسـی ضـرب ماتریس‌ها اسـت. ضرب ماتریس به‌عنـوان یک الگوریتـم پرهزینه محاسباتی که قابلیت بالایی جهت موازی‌سازی دارد مورد توجه بسیاری قرار گرفته است. Hassan و همکارانش [۵] مفاهیم مربوط به بسـط بردار پیشـرفته و کاربرد آن در ضرب دو ماتریس را به صورت مفصل بیان نموده‌اند.

پیچـش[۶۸] یک عملیات سـاده ریاضی اسـت که برای بسـیاری از عملگرهای پردازش تصویر از جمله فیلترهای رفـع نوفه در حوزه مکان کاربرد دارد. از این‌رو به منظور بهبود کارایی و افزایش سرعت پردازش الگوریتم‌های مبتنی بـر مفهوم پیچش راه‌حل‌های نرم‌افزاری و سـخت‌افزاری متعددی ارائه گردیده اسـت. یکی از راهکارهای موجود به منظور بهبود کارایی الگوریتم‌های مبتنی بر مفهوم پیچش بهره‌بـرداری از فنّاوری OpenMP اسـت، که Cadena و

57- Loop Fusion
58- Loop Interchange
59- Loop Inversion
60- Software Pipelining
61- Stages
62- Functional Units
63- Loop Level Vectorization (LLV)
64- Super-word Level Parallelism (SLP)
65- Instruction Level Parallelism (ILP)
66- Shuffle
67- Permute
68- Convolution

همکارانش [۲۳] اجرای کلاسیک و بهبود یافته فیلتر گوسی را با اســتفاده از این فنّــاوری ارائه دادنــد. نتایج تجربی به‌دســت آمده نشان می‌دهد نســخه بهبود یافته الگوریتم تأثیر بسزایی بر روی پیچیدگی زمانی الگوریتم می‌گذارد. Kim و همکارانش [۲٤] نیز چند نمونه روش بهینه‌ســازی با استفاده از بردارسازی توسط دستورالعمل‌های فنّاوری SIMD و دسترســی به حافظه تراز شده به روی یک هسته از پردازنــده Intel x86 به منظور بهبود کارایی عملگرهای پردازش تصویر با استفاده از پیچش یک بعدی ارائه دادند.

بــرای بهبــود کارایــی فیلتر پاســخ ضربــه محدود، Shahbahrami و همکارانش [۱۱] از زبان برنامه‌نویسی C و زبان‌ماشین برای بردارسازی حلقه بیرونی، حلقه درونی و هر دو حلقه، به صورت مجزا اســتفاده کردند. علاوه بر ایــن بدون اســتفاده از دســتورهای SIMD، در کد برنامه بهینه‌سازی‌های باز کردن حلقه و لوله‌ای کردن نرم‌افزاری به صورت دســتی انجام شده اســت. این بهینه‌سازی‌ها و بردارســازی‌ها باعث افزایش کارایی فیلتر پاســخ ضربه محدود شــده اما به دلیــل گلوگاه‌های حافظــه‌ای افزایش سرعت برای اندازه‌های مختلف داده‌ها متفاوت بوده است.

از راهکارهــای مــورد اســتفاده بــرای اندازه‌گیــری میزان شــباهت میان بلوک‌های تصویر یا ویدئو، استفاده از الگوریتم‌هــای مجمــوع قدرمطلق تفاضل‌هــا و مجموع مربعات تفاضل‌ها است. Shahbahrami و همکارانش [۱٤] دســتورهای خاص موجود در فنّاوری SIMD را که قابلیت استفاده در الگوریتم‌های اندازه‌گیری شباهت دارند بررسی نمودنــد. در این مدل بیان شــده که تنها کاربرد دســتور خــاص مجموع قدرمطلق تفاضل‌ها برای تخمین حرکت در استانداردهایی مانند MPEG1/2/4 و H.263/4 است. علاوه بر محاسبه مجموع قدرمطلق تفاضل‌ها، نتایج پیاده‌سازی مربوط به مجموع مربعات تفاضل‌ها نیز تشریح و ارزیابی شده است. افزایش سرعت‌های به دست آمده برای مجموع قدرمطلق تفاضل‌ها تا حداکثر ۲٤ برابر و برای مجموع مربع تفاضل‌ها حداکثر ۸ برابر حالت متوالی گزارش شده است.

خلاصه برخی از کارهای مرتبط با توابع انتخابی تحقیق حاضر به همراه نقاط قوت و ضعف آن‌ها در جدول ٤ بیان شده است. همان‌طور که نشــان داده شده، این پژوهش‌ها علاوه بر طرح مسئله‌های مهم و یافتن جواب آن‌ها خلأهایی همچنان وجود دارد.

به‌عنوان مثال، اغلب کدهای اجرایی در سطح بهینه سازی O2 ارزیابی شده اند که باعث می‌شود بهترین شرایط برای ترجمه کدهای حالت متوالی به زبان ماشین مهیا نباشد. لذا، مقایسه کد بهبود داده شده با بهترین اجرای حالت متوالی بــرای ارزیابی دقیق‌تر حائز اهمیت اســت. از طرفی دیگر، اغلب کارهای پیشــین از یک کامپایلر استفاده کرده‌اند که باعث می‌شــود ارزیابی صورت گرفته کاستی محسوسی در حوزه کامپایلرها و بردارســازی خودکار داشته باشد. علاوه بر موارد ذکر شــده، پردازنده‌ها در سال‌های اخیر پیشرفت های قابل ملاحظه‌ای داشته‌اند که نیاز است توابع انتخابی با به‌روزترین فنّاوری SIMD همه جا حاضر، یعنی AVX2 پیاده‌ســازی و ارزیابی شــوند. بــا در نظر گرفتن نقــاط ضعف کارهای مرتبط، روش‌های پیشــنهادی برای بردارســازی توابع انتخابی در ادامه مورد بحث و بررسی قرار می‌گیرند.

۴- روش پیشنهادی بردارسازی

بــرای بهبود کارایی برخی از توابع چندرســانه‌ای از جملــه جمع ماتریــس (ADD)، ترانهــاده ماتریس (TRA)، ضــرب ماتریــس (MUL)، فیلتر پاســخ ضربــه محدود (FIR)، مجمــوع قدرمطلق تفاضل‌ها (SAD) و مجموع مربع تفاضل‌ها (SSD) پیاده‌ســازی‌های مختلفی با اســتفاده از فنّاوری SIMD انجام شــده اســت که جدول ٥ نام توابع پیاده‌سازی شده و فنّاوری‌های مختلف SIMD که با انواع داده‌ای صحیح (int) و اعشــاری (flloat) اســتفاده شده است را نشــان می‌دهد. در برنامه‌هایی که در این قسمت ارائه شــده‌اند، توابع اینترینزیک ۱۲۸ بیتی با mm و ۲٥٦ بیتی با mm ۲٥٦ شروع می‌شوند. تمامی پیاده‌سازی‌هایی

جدول ۴: خلاصه برخی از کارهای مرتبط پیشین

تابع	مرجع	نقاط قوت	نقات ضعف
ماتریس ترانهاده	Zekri [۹]	روش جدیدی برای ترانهاده‌کردن ماتریس حاوی اعداد اعشاری با دقت معمولی با بهره‌گیری از تکنولوژی AVX و SSE مطرح شده است و نتایج با استفاده از کامپایلر GCC ارزیابی شده است. حداکثر افزایش کارایی ۲/۸۳ برابر نسبت به اجرای سریال برنامه ترانهاده به دست آمده و در ترکیب این تابع با جمع ماتریس (A+BT) افزایش کارایی ۳/۱۹ به دست آمده است.	مقایسه در سطح بهینه‌سازی O۲ صورت گرفته که باعث می‌شود بسیاری از بهینه‌سازی‌ها در حالت سریال در نظر گرفته نشود و زمان خروجی سریال بهترین زمان نباشد. همچنین ترانهاده با استفاده از دستورات درهم‌سازی پیاده‌سازی شده و بلوک‌های ۴×۴ برای ترانهاده انتخاب شده است در حالی که بردارهای ۲۵۶ بیتی این قابلیت را دارند که بلوک‌های ۸×۸ را در یک مرحله ترانهاده کنند. علاوه بر این ارزیابی نتایج برای داده‌های نوع صحیح صورت نگرفته است.
ضرب ماتریس	Hassan [۵]	ضرب ماتریس حاوی اعداد اعشاری با دقت معمولی با استفاده از زبان ماشین (inline assembly) و IPM با بهره‌گیری از دستورالعمل‌های AVX در کامپایلرهای ICC و MSVC++ پیاده سازی شده است. نتایج نشان می‌دهد که کارایی IPM نسبت به زبان ماشین بیش از ۲ برابر است.	مقایسه بین ضرب ماتریس حاوی اعداد صحیح با استفاده از AVX۲ و همچنین مقایسه بردارهای ۱۲۸ بیتی SSE۴ صورت نگرفته است. علاوه بر این کامپایلرهایی مانند GCC و LLVM که کاربرد وسیعی در توسعه کامپایلرها دارند ارزیابی نشده‌اند.
فیلتر پاسخ ضربه محدود	Shahbahrami [۱۱]	برنامه FIR با استفاده از زبان C و دستورات زبان ماشین MMX به روش‌های مختلفی از جمله بردارسازی حلقه بیرونی، حلقه درونی و هر دوحلقه انجام شده است و نتایج در کامپایلر GCC ارزیابی شده است. علاوه بر این کدهای زبان ماشین و برنامه C به صورت واضح در مقاله گنجانده شده است و از تصاویر مناسب جهت تفهیم الگوریتم استفاده شده است.	در این مقاله از inline assembly استفاده شده که نسبت به IPM قابلیت انتقال‌پذیری کمتری دارد و همانطور که Hassan و همکاران [۵] نشان داده اند کارایی کمتری هم می‌تواند داشته باشد. علاوه بر این بردارسازی‌ها با فنّاوری قدیمی‌تر که اکنون منسوخ شده است صورت گرفته و نتایج در سطح بهینه سازی O۲ ارزیابی شده‌اند.
اندازه‌گیری شباهت	Shahbahrami [۱۴]	مطالعه جامع و مفصلی برروی دستورالعمل psadbw انجام شده که بیان می‌کند تنها حالت‌ها و برنامه‌های خاصی هستند که می‌توانند از این دستورالعمل بهره‌گیرند. به دلیل محدود بودن نوع داده‌ای دستورالعمل مذکور، دستورالعمل‌هایی برای انواع دیگر داده‌ها طراحی و ارزیابی شده است.	دستورالعمل‌های طراحی شده برای کامپیوترهای همه منظوره قابل استفاده نیستند چراکه انواع داده‌ای ضریبی از ۱۲ بیت هستند در حالی که پایه کامپیوترهای همه منظوره ضریبی از هشت بیت است. علاوه بر این نتایج با استفاده از بسط MMX ارزیابی شده اند.

جدول ۵: فنّاوری‌های استفاده شده در بردارسازی صریح برای هسته‌های چندرسانه‌ای

	SSE4 (float)	SSE4 (int)	AVX (float)	AVX2 (int)
ADD	√	√	√	√
TRA	√	√	√	√
MUL	√	√	√	√
FIR	-	√	-	√
SAD	-	√	-	√
SSD	-	√	-	√

که از بردارهای ۱۲۸ بیتی استفاده می‌کنند را با نام SSE4 خطاب می‌کنیم [۲۵، ۲۶].

٤-۱ جمع ماتریس

پیاده‌سازی IPM الگوریتم جمع ماتریس برای اعداد اعشاری و صحیح با استفاده از فنّاوری‌های SSE4 و AVX2 در شکل ٤ و ٥ به ترتیب نشان داده شده است. همان‌طور که در این شکل‌ها مشخص است بردارهایی از مکان‌های متناظر دو ماتریس A و B خوانده می‌شود و در ثبّات‌های برداری بارگذاری می‌شوند و نتیجه حاصل جمع در ثبّات مقصد ذخیره می‌گردد. سپس ثبّات برداری مقصد در نشانی تعیین شده از درایه i و j ماتریس C ذخیره می‌شود. شکل ٤ (الف) برنامه برداری جمع ماتریس با استفاده از SSE4 برای اعداد اعشاری با دقت معمولی را نشان می‌دهد که با استفاده از بردارهای ۱۲۸ بیتی محاسبات مربوطه را انجام می‌دهد. به‌عبارت دیگر چهار عدد اعشاری در یک ثبّات برداری قرار می‌گیرد. همان‌طور که در شکل مربوطه

```
1   int/float
2    __attribute__(( aligned(16))) A[N][M],
3    __attribute__(( aligned(16))) B[N][M],
4    __attribute__(( aligned(16))) C[N][M];

5       __m128 vec_r0, vec_r1, vec_r2;          (الف)
6     for(i = 0 ; i < N ; i++)
7      for(j = 0 ; j < M ; j += 4){
8      vec_r0 = _mm_load_ps(&A[i][j]);
9      vec_r1 = _mm_load_ps(&B[i][j]);
10     vec_r2 = _mm_add_ps(vec_r0, vec_r1);
11     _mm_store_ps( &C[i][j], vec_r2);
12     }
13      __m128i vec_r0, vec_r1, vec_r2;         (ب)
14    for(i = 0 ; i < N ; i++)
15     for(j = 0 ; j < M ; j += 4){
16     vec_r0 = mm_load_si128(
17     (__m128i *) &A[i][j]);
18     vec_r1 = mm_load_si128(
19     (__m128i *) &B[i][j]);
20     vec_r2 = _mm_add_epi32(vec_r0, vec_r1);
21     _mm_store_si128(
22     (__m128i *) &C[i][j], vec_r2);
23     }
```

شکل ۴: بردارسازی برنامه جمع ماتریس با استفاده از SSE۴ (الف) اعداد اعشاری با دقت معمولی (ب) اعداد صحیح

```
1   int/float
2    __attribute__(( aligned(32))) A[N][M],
3    __attribute__(( aligned(32))) B[N][M],
4    __attribute__(( aligned(32))) C[N][M];

5   __m256 vec_r0, vec_r1,                      (الف)
    vec_r2;
6     for(i = 0 ; i < N ; i++)
7      for(j = 0 ; j < M ; j += 8){
8      vec_r0 = _mm256_load_ps( &A[i][j]);
9      vec_r1 = _mm256_load_ps( &B[i][j]);
10     vec_r2 = _mm256_add_ps(vec_r0,
       vec_r1);
12     _mm256_store_ps( &C[i][j], vec_r2);
       }
13  __m256i vec_r0, vec_r1,                     (ب)
    vec_r2;
14    for(i = 0 ; i < N ; i++)
15     for(j = 0 ; j < M ; j += 8){
16     vec_r0 =
17
18    _mm256_load_si256((__m256i*)&A[i][j]);
19     vec_r1 =
20
21    _mm256_load_si256((__m256i*)&B[i][j]);
22     vec_r2 = _mm256_add_epi32(vec_r0,
23     vec_r1);
       _mm256_store_si256(
       (__m256i*)&C[i][j], vec_r2);
       }
```

شکل ۵: بردارسازی برنامه جمع ماتریس با استفاده از AVX2 (الف) اعداد اعشاری با دقت معمولی (ب) اعداد صحیح

در کدهای داخل حلقه‌ها مشخص است، چهار عدد اعشاری از ماتریس A و چهار عدد اعشاری از ماتریس B در بردارهای مربوطه قرار می‌گیرند و با دستور جمع به‌طور همزمان این چهار عدد جمع شده و در بردار حاصل قرار می‌گیرد. به دلیل استفاده از بردارهای ۱۲۸ بیتی داده‌های موجود در حافظه باید به مرزهای ۱۶ بایتی تراز گردند. به این معنی که نشانی شروع عناصر مضرب ۱۶ بوده تا دسترسی به نشانی‌های حافظه با اطمینان انجام شود.

در شکل ٤ (ب) برنامه برداری جمع ماتریس با استفاده از SSE٤ نشان داده شده که بر روی اعداد صحیح محاسبات را انجام می‌دهد. تفاوت این پیاده‌سازی با پیاده‌سازی شکل ٤ (الف) در نوع داده‌ای و توابع اینترینزیک استفاده شده است. در این پیاده‌سازی دسترسی به نشانی حافظه نیاز به تعیین نوع داده‌ای دارد که با *m128i __ مشخص شده است تا کامپایلر بداند که به چه نوع داده‌ای دسترسی انجام می‌دهد.

در شکل ٥ (الف) برنامه جمع ماتریس با استفاده از فنّاوری AVX2 برای اعداد اعشاری با دقت معمولی نشان داده شده که محاسبات را با استفاده از ثبّات‌های برداری ۲٥٦ بیتی انجام می‌دهد. مرزبندی حافظه به دلیل استفاده از ثبّات‌های برداری ۲٥٦ بیتی، به صورت ۳۲ بایتی در نظر گرفته شده است. با توجه به پردازش همزمان هشت عنصر در حلقه متغیر شمارنده حلقه با هشت جمع می‌شود (j+=8). شکل ٥ (ب) برنامه برداری جمع ماتریس با استفاده از بسط بردار پیشرفته را برای اعداد صحیح نشان می‌دهد. همان‌طور که مشاهده می‌شود در این توابع برای دسترسی به داده‌های صحیح *m256i __ قبل از نشانی حافظه، جهت تفهیم کامپایلر، اضافه می‌گردد.

٤-۲ ترانهاده ماتریس

برای ترانهاده کردن ماتریس روش بلوک‌بندی انجام می‌شود. به این صورت که ماتریس اولیه A به بلوک‌های ۸×۸ تقسیم‌بندی می‌شود. بلوک‌های انتخابی هر کدام ترانهاده می‌شوند و در مکان مناسب قرار می‌گیرند. شکل ٦ برنامه برداری ترانهاده کردن ماتریس که از دستور gather استفاده می‌کند را به تصویر کشیده است. عناصر ماتریس از نشانی‌های ناپیوسته حافظه به صورت ستونی

```
1   int index [8] =
2    {0, N, N*2, N*3, N*4, N*5, N*6, N*7};
3   __m256i vec_index =
4    _mm256_load_si256((__m256i *)
5   &index[0]);
6   __m256i vec_r0, vec_r1, vec_r2, vec_r3,
7    vec_r4, vec_r5, vec_r6, vec_r7;
8   for(i=0; i<N; i+=8)
9    for(j=0; j<M; j+=8){
10   vec_r0 = _mm256_i32gather_epi32 (
11   (int const *)&A[i][j+0], vec_index, 4);
12   …
13   vec_r7 = _mm256_i32gather_epi32 (
14   (int const *)&A[i][j+7], vec_index, 4);
15   _mm256_store_si256(
16   (__m256i *)&A_T[j+0][i], vec_r0);
17   …
18   _mm256_store_si256(
19   (__m256i *)&A_T[j+7][i], vec_r7);
     }
```

شکل ۶: بردارسازی ترانهاده کردن ماتریس با استفاده از دستورالعمل gather و فنّاوری AVX2

```
1   for(i=0; i < N; i += 8){
2    vec_r0 = _mm256_load_si256 (
3    (__m256i *) &A[i]);
4    vec_Co = _mm256_set1_epi32
5   (Coeff[0]);
6    vec_r2 = _mm256_mullo_epi32 (
7    vec_Co , vec_r0);
8    for(j=1; j < M; j++){
9    vec_r0 = _mm256_loadu_si256(
10   (__m256i *)&A[i+j]);
11   vec_Co = _mm256_set1_epi32
12  (Coeff[j]);
13   vec_r1 = _mm256_mullo_epi32(
14   vec_Co, vec_r0);
15   vec_r2 = _mm256_add_epi32(
16   vec_r1, vec_r2);
17   }
18   _mm256_store_si256(
     (__m256i *)&B[i], vec_r2);
     }
```

شکل ۷: بردارسازی فیلتر FIR با استفاده از دستورالعمل‌های AVX۲

خوانده می‌شوند و در مکان مناسب با نشانی پیوسته نوشته می‌شوند.

٤-٣ فیلتر پاسخ ضربه محدود

فیلتر پاسخ ضربه محدود بر مبنای عملیات پیچش استوار است. به این صورت که ضرایب و داده‌ها در آرایه یک بعدی ذخیره می‌شوند و با محاسبه مجموع حاصلضرب ضرایب در داده‌ها به ترتیب عناصر خروجی به‌دست می‌آید. در شکل ۷ نحوه بردارسازی فیلتر پاسخ ضربه محدود با استفاده از IPM نشان داده شده است که بردار vec_Co در بردارنده یک ضریب که در همه عناصر آن پخش شده است و بردار vec_r0 حاوی مقادیر سیگنال ورودی است. حاصلضرب برداری ضرایب و سیگنال‌های ورودی در بردار vec_r1 ذخیره می‌شود و با بردار vec_r2 جمع می‌شود تا در انتها یک بردار خروجی حاصل شود و در آرایه خروجی ذخیره گردد.

٤-٤ مجموع قدرمطلق تفاضل‌ها

در شکل ۸ برنامه برداری مجموع قدرمطلق تفاضل‌ها نشان داده شده است. برای این پیاده‌سازی از دستورالعمل با کاربرد خاص vpsadbw که در مجموعه دستورالعمل‌های AVX۲ گنجانده شده استفاده می‌شود. با این دستورالعمل محتویات دو بردار ۲۵٦ بیتی را از یکدیگر تفریق کرده و مجموع قدرمطلق چهار گروه هشت عنصری را در بردار خروجی ذخیره می‌کند. به همین دلیل در هر بار تکرار حلقه ۳۲ داده یا عنصر پردازش می‌شوند و متغیر شمارنده حلقه در هر تکرار با عدد ۳۲ جمع می‌شود (j+=32). لذا با استفاده از تابع معادل این دستور _mm256_sad_epu8 نتیجه محاسبات دو بردار ورودی در بردار خروجی ذخیره می‌شود و با جمع آن‌ها در انباره نتیجه حاصل می‌شود. در انتها کافیست عناصر صفر، چهار، هشت و دوازده از بردار خروجی با هم جمع گردند و نتیجه که همان بردار جابجایی است حاصل گردد.

٤-٥ مجموع مربعات تفاضل‌ها

در شکل ۹ برنامه برداری مجموع مربعات تفاضل‌ها با استفاده از AVX2 نشان داده شده است که بر روی اعداد صحیح ١٦ بیتی محاسبات انجام می‌دهد. برای این منظور ابتدا دو بردار حاوی پیکسل‌های تصویر را از یکدیگر تفریق می‌کند و بردار نتیجه را در خودش ضرب می‌کند تا

```
for(i=0; i<N; i++)
 for(j=0; j<M; j+=32){
 vec_r0 =
 _mm256_load_si256((__m256i*)&A[i][j]);
 vec_r1 =
 _mm256_load_si256((__m256i*)&B[i][j]);
 vec_r2 = _mm256_sad_epu8( vec_r0 ,
vec_r1);
 vec_r3 = _mm256_add_epi32(vec_r2,
vec_r3);
 }
```

شکل ۸: بردارسازی برنامه مجموع قدرمطلق تفاضل‌ها

مربع آن‌ها به دست آید و سپس حاصل‌ضرب را با برداری که به‌عنوان انباره در نظر گرفته شـــده جمع می‌کند. پس از اتمام جمع افقی به صورت متوالی انجام می‌شـــود. در این برنامه در هر بار تکرار حلقه ۱۶ داده یا عنصر از هر بردار ورودی پردازش می‌گردد که متغیر شـــمارنده حلقه در هر تکرار با عدد ۱۶ جمع می‌شود (j+=16).

همان‌طـــور کـــه در مثال‌هـــای بالا شـــرح داده شـــد، بردارســـازی با استفاده از IPM مســـتلزم یادگیری زبان ماشـــین، توابع کتابخانه‌ای، عملکرد دستورهای SIMD و نحوه اســـتفاده بهینه از آن‌ها، نوع فنّاوری SIMD موجود و نظارت دقیق بر روی متغیرهای شـــمارنده حلقه اســـت کـــه برنامه‌نویس باید بـــا جزئیات دســـتورها و معماری پردازنده نیز آشـــنایی داشـــته باشـــد. به دلیل مشکلات موجود در بردارســـازی صریح، در کامپایلرها خاصیت بردارسازی خودکار گنجانده شده است. این قابلیت‌ها را هر ســـاله توسعه و رشد می‌دهند و کار برنامه‌نویسان را راحت‌تر می‌کنند. به این صورت که کد متوالی برنامه‌های نوشـــته شده را گرفته و تبدیل به کدهای SIMD بر حسب توانمندی‌هـــای فراهم شـــده می‌کنند. به منظـــور ارزیابی کارایـــی کامپایلرهـــای GCC، ICC و LLVM در مســـئله بردارســـازی خودکار، مقایســـه آن‌ها با پیاده‌سازی‌های پیشـــنهادی IPM و یافتن ارتباط بین هزینه بردارســـازی و میزان بهبود کارایی در ادامه نتایج آزمایشـــگاهی شرح داده می‌شوند.

```
for(i=0; i<N; i++)
 for(j=0; j<M; j+=16){
 vec_r0 = _mm256_sub_epi16
 (
 _mm256_load_si256(
 (__m256i *)&A[i][j]),
 _mm256_load_si256(
 (__m256i *)&B[i][j]));
 vec_r1 =
 _mm256_mullo_epi16(
 vec_r0, vec_r0);
 vec_r2 = _mm256_add_epi16
 (
 vec_r1, vec_r2);
 }
```

شکل ۹: بردارسازی برنامه مجموع مربعات تفاضل‌ها با استفاده از مجموع دستورات SIMD فنّاوری AVX۲

۵-نتایج پیاده سازی

۵-۱ محیط ارزیابی

برای ارزیابی روش پیشنهادی، پیاده‌سازی‌های انجام شده با فنّاوری یک دستورالعمل و چندین داده از پردازنده چهارهسته‌ای نسل ششـــم اینتل با فرکانس ۲/۶ گیگاهرتز اســـتفاده شده است که می‌تواند هشت ریسه را به صورت هم‌زمان اجرا کند. سیســـتم عامل استفاده شده نسخه ۲۷ از توزیع لینوکسی فدورا اســـت. برای اجرای برنامه‌ها از کامپایلرهای GCC، ICC و LLVM اســـتفاده شـــده اســـت و بردارسازی صریح با استفاده از IPM، کتابخانه x86intrin، انجام شده اســـت. در حالت تک هسته‌ای اجرای برنامه بر روی یک هســـته که از پیش ذخیره شـــده اســـت صورت می‌گیرد تا احتمال اجرای دیگر فرآیندها در سیستم عامل از بین برود. برای گرفتن میزان تکانه‌های مصرفی هر برنامه از تابع ()rdtsc_ اســـتفاده شده است که شمارنده پردازنده را به صورت یک عدد صحیـــح ٦٤ بیتی ذخیره می‌کند. به صورت جزئی‌تر مشـــخصات محیط ارزیابی در جدول ٦ بیان شده اســـت. برای به دست آوردن افزایش کارایی هر الگوریتم به تعداد دفعات زیادی اجرا شده و کمترین میزان تکانه مصرفی برای ارزیابی به کار برده شـــده است. برای ارزیابی نتایج، حالت موازی و بهبود یافته با حالت متوالی

جدول ۶: مشخصات محیط ارزیابی و بُن‌سازهٔ پیاده‌سازی

پردازنده مرکزی	Intel Corei7- 6700HQ
پهنای ثبّات	حداکثر ۲۵۶ بیت
اندازه هر خط از حافظه نهان	۶۴ بایت
سطح اول حافظه نهان	۳۲ کیلوبایت، هشت مسیره، کمترین تأخیر ۴ تکانه، ۲×۳۲ بایت بارگذاری و ۳۲ بایت ذخیره‌سازی
سطح دوم حافظه نهان	۲۵۶ کیلوبایت، چهار مسیره، کمترین تأخیر ۱۲ تکانه
سطح سوم حافظه نهان	حداکثر ۲ مگابایت برای هر هسته، حداکثر ۱۶ مسیره، کمترین تأخیر ۴۴ تکانه
سیستم عامل	فدورا نسخه ۲۷، ۶۴ بیتی
کامپایلرها	ICC 18.0.1 GCC 7.2.1 LLVM 5.0.1
ابزارهای برنامه‌نویسی	زبان برنامه‌نویسی C کتابخانه x86intrin.h رابط OpenMP
کامپایلر مبنای ارزیابی	ICC
ذخیره شمارنده	_rdtsc();
خط فرمان کامپایلر حالت متوالی	icc -O3 -no-vec
	gcc -O3 -fno-tree-vectorize -fno-tree-slp-vectorize
	clang -O3 -fno-vectorize -fno-slp-vectorize

اجرا شـــده در کامپایلر ICC مقایسه می‌شود. ذکر این نکته مهم اســـت که کامپایلر ICC در حالت متوالی کارایی بسیار خوبی نسبت به دیگر کامپایلرها نشان می‌دهد.

۵-۲ تشریح و ارزیابی نتایج پیاده‌سازی

در این بخش پیاده‌سازی‌های انجام شده با استفاده از بردارســـازی صریـــح، IPM-SSE4 و IPM-AVX2، همچنین بردارسازی غیرصریح، CAV-AVX2، برای حالت تک‌هسته‌ای با استفاده از کامپایلرهای استفاده شده، GCC، ICC و LLVM، ارزیابی می‌شوند. جدول ۷ مخفف‌سازی‌های استفاده شده در جـــدول ۸ جهـــت ارائه نتایج را نشـــان می‌دهد. به دلیل این‌که همه پیاده‌ســـازی‌ها برای اعـــداد صحیح و تعدادی از آن‌ها برای اعداد اعشـــاری انجام شـــده است، زمانی که نوع داده‌ای مطرح نشـــود منظور عدد صحیح است. اعداد صحیح استفاده شـــده ۸، ۱۶ و ۳۲ بیتی هستند که پیش‌تر برای هر کدام از الگوریتم‌ها بیان گردید. از آنجایی که نحوه استخراج فنّاوری AVX2 برای همهٔ ابزارها بررسی گردیده اســـت، تنها زمانی که نیاز باشد به آن اشاره می‌کنیم و در صورتی که تأکید بر نوع بسط استفاده شده نشود منظور AVX2 برای اعداد صحیح است.

جـــدول ۸ میـــزان بهبـــود کارایـــی تعـــدادی از توابع چندرســـانه‌ای را بـــا اســـتفاده از مـــدل برنامه‌نویســـی اینترینزیک (IPM) و بردارســـازی خودکار کامپایلر (CAV) کـــه از فنّاوری‌هایSSE4 (۱۲۸ بیتـــی) وAVX2 (۲۵۶ بیتی) اســـتفاده می‌کنند، بر روی یک هســـته از پردازنده، نسبت به پیاده‌ســـازی متوالی نشـــان می‌دهـــد. همان‌طور که در جدول قابل مشـــاهده اســـت کارایـــی IPM-AVX2 به دلیل پردازش تعداد عناصر داده‌ای بیشتر در یک ثبّات نسبت به IPM-SSE4 بیشـــتر است، چرا که طول ثبّات‌های AVX2 دو برابر SSE4 اســـت، اما لزوماً دو برابر شـــدن طول ثبّات‌ها به معنی دو برابر شـــدن کارایی نیست، چرا که خواندن و نوشتن در حافظه یکی از چالش‌های موجود است و چنانچه در جدول نشان داده شده است، این میزان کارایی و بهبود برای اندازه تصویر کوچک‌تر (۵۱۲×۵۱۲) به مراتب بالاتر از انـــدازه تصویـــر بزرگ‌تر (۱۰۲٤×۱۰۲٤) اســـت. نتیجه دیگری که از جدول قابل اســـتخراج اســـت، این اســـت که، به‌طور کلی کارایی مدل برنامه‌نویســـی اینترینزیک یا IPM نسبت به بردارسازی خودکار کامپایلر یا CAV بالاتر است. از طرف دیگر تمامی کامپایلرها توانایی بردارسازی توابع، برای مثال تابع TRA را ندارند. کامپایلرها از دســـتورهای زبان ماشـــین فنّاوری یک دســـتورالعمل و چندین داده در تولید کدها استفاده می‌کنند. حتی کامپایلرهای ICC و GCC در بردارسازی تابع مجموع قدرمطلق تفاضل‌ها، از دستور خاص تهیه شده در معماری پردازنده استفاده می‌کنند، در حالی که کامپایلر LLVM، در تولید کد تابع مجموع قدرمطلق تفاضل‌ها از دســـتورهای معمولی SIMD استفاده می‌کند. به همین دلیل اســـت که کارایی کامپایلرهـــای ICC و GCC به مراتب بیشـــتر از LLVM اســـت. به‌طور کلی کامپایلرها

جدول ۷: واژه‌های مخفف شده برای پیاده‌سازی‌ها

IPM-SSE4	بردارسازی‌های صریح که با استفاده از مدل برنامه‌نویسی اینترینزیک (IPM) انجام شده و از دستورهای SSE4 برای انجام عملیات بهره برده است.
IPM و IPM-AVX2	بردارسازی‌های صریح که با استفاده از مدل برنامه‌نویسی اینترینزیک (IPM) انجام شده و از دستورهای AVX2 برای انجام عملیات بهره برده است.
CAV و CAV-AVX2	بردارسازی‌های غیرصریح که با استفاده از بردارسازی خودکار کامپایلر (CAV) انجام شده و از دستورهای AVX2 برای انجام عملیات بهره برده است.
ADD و ADD-float	ADD : پیاده‌سازی انجام شده برای برنامه جمع ماتریس، حاوی اعداد صحیح ۳۲ بیتی است و ADD-float برای اعداد اعشاری با دقت معمولی است.
TRA و TRA-float	TRA : پیاده‌سازی انجام شده برای برنامه ترانهاده ماتریس، حاوی اعداد صحیح ۳۲ بیتی است و TRA-float برای اعداد اعشاری با دقت معمولی است.
MUL و MUL-float	MUL : پیاده‌سازی انجام شده برای برنامه ضرب ماتریس در ترانهاده ماتریس، حاوی اعداد صحیح ۳۲ بیتی است و MUL-float برای اعداد اعشاری با دقت معمولی است. اگر نیاز به تفکیک برنامه ضرب ماتریس در ماتریس ترانهاده و ضرب ماتریس در ماتریس باشد به ترتیب از ABT و AB استفاده می‌شود.
FIR	پیاده‌سازی انجام شده برای برنامه فیلتر پاسخ ضربه محدود بر روی آرایه‌ای حاوی اعداد صحیح ۳۲ بیتی است.
SAD	پیاده‌سازی انجام شده برای برنامه مجموع قدر مطلق تفاضل‌ها بر روی ماتریس حاوی اعداد صحیح ۸ بیتی است.
SSD	پیاده‌سازی انجام شده برای برنامهٔ مجموع مربعات تفاضل‌ها بر روی ماتریس حاوی اعداد صحیح ۱۶ بیتی است.

رفتارهای متفاوتی در برخورد با هر تابع نشـــان می‌دهند. اگرچـــه کارایی IPM تا حدودی بالاتر از CAV اســـت، ولی به دلیل مشـــکلات موجود در IPM، از قبیل عدم سازگاری و قابـــل حمل بودن، وقت گیر و خطا پذیـــر بودن و این‌که برنامه‌نویس باید تمامی جزئیات دستورها را بداند، توسعه و گســـترش قابلیت‌های CAV و استفاده از آن بیشتر مورد توجه است.

نتیجه‌گیری

امـــروزه برنامه‌های چندرســـانه‌ای به صـــورت قابل توجهـــی در کاربردهـــای عمومی محبوبیـــت یافته‌اند. این برنامه‌ها ویژگی‌های خاص خود مانند انواع داده‌ای کوچک و موازی‌ســـازی دانه‌ریز و دانه‌درشـــت را دارا می‌باشند. از طرفـــی در پردازنده‌هـــای همـــه منظوره، فنّـــاوری یک دستورالعمل و چندین داده گنجانده شده است که با فراهم نمودن ثبّات‌های ۱۲۸ و ۲۵۶ بیتی بتوان داده‌های بیشتری در یک ثبّات بارگذاری کرد و عملیات یکسانی بر روی آن‌ها اعمال نمود. در این مقاله تعدادی از الگوریتم‌های پر کاربرد حوزه چندرســـانه‌ای انتخاب و پیاده‌ســـازی گردید. برای پیاده‌ســـازی از زبان C و مدل برنامه‌نویســـی اینترینزیک برای بردارســـازی صریح اســـتفاده گردید. همچنین برای بردارســـازی غیرصریح از قابلیت بردارســـازی خودکار کامپایلر استفاده گردید. نتایج پیاده‌سازی‌ها بر روی پردازنده Intel® Core™ i7-6700HQ با استفاده از کامپایلرهای ICC، GCC و LLVM نشان داد کارایی فنّاوری IPM-AVX2 نسبت به IPM-SSE4 بیشـــتر اســـت و میزان افزایـــش کارایی در بردارسازی‌های انجام شـــده ارتباط زیادی به نوع داده‌ای به کار گرفته شـــده دارد. هرچه اندازه نـــوع داده و اندازه آرایه کوچک‌تر باشد میزان کارایی بهبود داده شده بیشتر خواهـــد بود، چرا که نوع داده‌ای با انـــدازه بزرگ‌تر، تعداد عناصر موجود در یک بردار کمتر را به دنبال دارد و به تبع آن تعداد مسیرهای موازی کمتر، افزایش کارایی کمتری را رقم می‌زند. همچنین آرایه بزرگ‌تـــر احتمال رخداد فقدان حافظه نهان بیشتر می‌شود و زمان زیادی برای مهیا کردن داده مورد نیاز در حافظه نهان صرف می‌شـــود. از طرفی دو برابر شدن طول بردار به‌طور قطعی افزایش کارایی دو

جدول ۸: بهبود کارایی توابع چندرسانهای با استفاده از برنامه‌نویسی موازی IPM و CAV که از فنّاوری‌های SSE۴ و AVX۲ بر روی یک هسته از پردازنده استفاده می کنند نسبت به پیاده سازی‌های متوالی.

تابع	پیاده‌سازی	۵۱۲×۵۱۲			۱۰۲۴×۱۰۲۴		
		ICC	GCC	LLVM	ICC	GCC	LLVM
ADD	IPM-SSE4	۱,۷۵	۱,۷۴	۱,۷۳	۱,۰۵	۱,۰۶	۱,۰۵
	IPM-AVX2	**۲,۰۳**	**۲,۰۱**	**۲,۰۱**	**۱,۰۸**	**۱,۰۸**	**۱,۰۸**
	CAV-AVX2	۲,۰۴	۲,۰۳	۲,۰۱	۱,۳۷	۱,۰۷	۱,۰۸
ADD-float	IPM-SSE4	۱,۹۴	۱,۸۷	۱,۸۸	۱,۱۴	۱,۱۴	۱,۱۴
	IPM-AVX2	۲,۱۹	۲,۱۷	۲,۱۷	۱,۱۷	۱,۱۷	۱,۱۷
	CAV-AVX2	**۲,۱۹**	**۲,۱۸**	**۲,۱۸**	**۱,۵**	**۱,۱۶**	**۱,۱۷**
TRA	IPM-SSE4	۱,۸۱	۱,۸۱	۱,۸۲	۲,۸	۲,۶۸	۲,۷۷
	IPM-AVX2	**۳,۵۵**	**۳,۵۵**	**۳,۵۴**	**۳,۵۷**	**۳,۶۹**	**۳,۶۵**
	CAV-AVX2	۰,۹۹	۰,۴۵	۰,۴۵	۱,۱۱	۰,۷۶	۰,۷۶
TRA-float	IPM-SSE4	۱,۷۶	۱,۷۶	۱,۷۶	۲,۸۵	۲,۸۴	۲,۷۴
	IPM-AVX2	**۳,۴۶**	**۳,۴۶**	**۳,۴۵**	**۳,۷۴**	**۳,۶۸**	**۳,۷۳**
	CAV-AVX2	۱,۰۴	۰,۴۴	۰,۴۴	۰,۹۷	۰,۷۷	۰,۸
MUL	IPM-SSE4	۱,۹۲	۱,۹	۲,۳۷	۱,۹۱	۱,۸۴	۲,۲۵
	IPM-AVX2	**۳,۳۳**	**۳,۲۳**	**۳,۶۹**	**۲,۸۸**	**۲,۹۶**	**۳,۱۹**
	CAV-AVX2	۳,۷۴	۳,۲۱	۳,۷۳	۳,۶۲	۳	۳,۲۷
MUL-float	IPM-SSE4	۱,۲۹	۱,۳۲	۱,۳۳	۱,۱۳	۱,۱۴	۱,۱۵
	IPM-AVX2	**۲,۵۹**	**۲,۹**	**۳,۰۲**	**۲,۳**	**۲,۴۱**	**۲,۴۶**
	CAV-AVX2	۴,۲۶	۲,۷۶	۳,۸۳	۳,۷۳	۲,۳۶	۳,۳۲
FIR	IPM-SSE4	۳,۳۱	۳,۵۴	۳,۵۲	۳,۰۸	۳,۴۴	۳,۴
	IPM-AVX2	**۶,۴۲**	**۶,۶۱**	**۶,۵**	**۵,۵۶**	**۵,۷**	**۵,۵۹**
	CAV-AVX2	۱,۳۸	۳,۶۶	۱,۰۱	۱,۳۷	۳,۴۶	۱,۰۱
SAD	IPM-SSE4	**۱۶,۸۵**	**۱۶,۵۷**	**۱۸,۹۷**	**۱۶,۶۵**	**۱۶,۳**	**۱۵,۶**
	IPM-AVX2	۲۲,۶۷	۲۱,۷۳	۲۱,۸۶	۲۱,۱۴	۲۰,۹۷	۲۰,۸۴
	CAV-AVX2	۲۱,۶۱	۲۲,۴۷	۹,۵۵	۲۰,۷	۲۰,۹۸	۹,۵۹
SSD	IPM-SSE4	۵,۳۷	۵,۳۴	۵,۱۷	۴,۶	۴,۷۶	۴,۹۳
	IPM-AVX2	۷,۰۱	۶,۹۸	۷,۰۶	۶,۵۳	۶,۶۴	۶,۶۴
	CAV-AVX2	۳,۸۷	۳,۷۲	۴,۱۶	۳,۷۸	۳,۶۱	۴,۲۱

برابر نمی‌دهد و در بسیاری از موارد کمتر از ۱/۵ برابر کارایی را افزایش می‌دهد. کامپایلرهای ICC و GCC در اکثر پیاده‌سازی‌ها عملکرد بهتری از خود نشان می‌دهند و کامپایلر LLVM کارایی قابل ملاحظه‌ای ندارد. اگرچه کارایی مدل برنامه‌نویسی اینترینزیک تا حدودی بالاتر از بردارسازی خودکار کامپایلر است، اما به دلیل سختی و مشکلات موجود در مدل برنامه‌نویسی اینترینزیک و راحت‌تر بودن بردارسازی خودکار کامپایلر، توسعه و گسترش قابلیت‌های بردارسازی خودکار و استفاده از آن بیشتر مورد توجه است.

مراجع

1. A. Shahbahrami, B. Juurlink, and S. Vassiliadis, “Matrix register file and extended subwords,” in Proceedings of the 2nd conference on Computing frontiers, 2005, vol. 5, no. 1, p. 171.
2. A. Shahbahrami, “Avoiding conversion and rearrangement overhead in SIMD architectures,” Ph.D. dissertation, Computer Engineering Laboratory, Delft University of Technology, Delft, Netherlands, 2008.
3. Intel Corporation, “Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 1: Basic Architecture,” vol. 1, no. 253665–060US, 2016.
4. A. Pohl, B. Cosenza, M. A. Mesa, C. C. Chi, and B. Juurlink, “An evaluation of current SIMD programming models for C++,” in Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing, 2016, pp. 1–8.
5. S. A. Hassan, A. M. Hemeida, and M. M. M. Mahmoud, “Performance Evaluation of Matrix-Matrix Multiplications Using Intel’s Advanced Vector Extensions (AVX),” Microprocess. Microsystem, vol. 47, pp. 369–374, Nov. 2016.
6. A. Shahbahrami and B. Juurlink, “SIMD Architectural Enhancements to Improve the Performance of the 2D Discrete Wavelet Transform,” in 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009, pp. 497–504.
7. A. Shahbahrami, B. Juurlink, and S. Vassiliadis, “A Comparison Between Processor Architectures for Multimedia Application,” in Proceedings of the 15th Annual Workshop on Circuits, Systems and Signal Processing, 2004, pp. 138–152.
8. J. Choi, J. J. Dongarra, and D. W. Walker, “Parallel Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers,” Parallel Computing, vol. 21, no. 9, pp. 1387–1405, 1995.
9. A. S. Zekri, “Enhancing the Matrix Transpose Operation Using Intel Avx Instruction Set Extension,” Internationl Journal Comput. Sci. Inf. Technol., vol. 6, no. 3, pp. 67–78, 2014.
10. S. Chatterjee and S. Sen, “Cache-efficient matrix transposition,” in Proceedings Sixth International Symposium on High-Performance Computer Architecture, 2000, pp. 195–205.
11. A. Shahbahrami, B. Juurlink and S. Vassiliadis, “Efficient vectorization of the FIR filter,” in Proceedings of the 16th Annual Workshop on Circuits Systems and Signal Processing (ProRISC), 2005, pp. 432-437
12. H. Amiri, and A. Shahbahrami, “High performance implementation of 2-D convolution using AVX2,” 19th International Symposium on Computer Architecture and Digital Systems, pp. 1-4, 2017.
13. R. C. Gonzalez and R. E. Woods, Digital Image Processing, Second. Prentice-Hall, 2002.
14. A. Shahbahrami, B. Juurlink and S. Vassiliadis, “Limitations of special-purpose instructions for similarity measurements in media SIMD extensions,” in Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, 2006, pp. 293-303
15. A. Peleg and U. Weiser, “MMX Technology Extension to the Intel Architecture,” IEEE Micro, vol. 16, no. 4, pp. 42–50, 1996.
16. P. Gepner, V. Gamayunov, and D. L. Fraser, “Early performance evaluation of AVX for HPC,” Procedia Comput. Sci., vol. 4, pp. 452–460, 2011.
17. Intel Corporation, “Intel Architecture Instruction Set Extensions Programming Reference,” no. 319433–012A, 2012.
18. Oracle Corporation, “x86 Assembly Language Reference Manual,” Oracle, no. 817–5477–11, 2010.
19. N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, “Intel AVX: New Frontiers in Performance Improvements and Energy Efficiency,” White Paper, Intel Corporation, pp. 1–9, 2008.
20. G. Lento, “Optimizing Performance with Intel Advanced Vector Extensions,” White Paper, Intel Corporation, 2014.
21. Intel Corporation, “Intel Intrinsics Guide.” [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/. [Accessed: 29-Jan-2017].
22. S. Larsen, “Exploiting Superword Level Parallelism with Multimedia Instruction Sets,” Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2000.
23. L. Cadena, A. Zotin, and F. Cadena, “Enhancement of Medical Image using Spatial Optimized Filters and OpenMP Technology,” International Multi Conference of Engineers and Computer Scientists, 2018.
24. D. Kim, V. W. Lee, and Y. K. Chen, “Image processing on multicore X86 architectures,” IEEE Signal Process Mag, vol. 27, no. 2, pp. 97-107, 2010.
25. Amiri Hossein, “Intel SIMD Technologies,” github, 2017. [Online]. Available: https://github.com/sinusee/Intel-SIMD-Technologies. [Accessed: 18-Mar-2017].
26. M. Moradifar, A. Shahbahrami, M. Nematpour, and H. Amiri, “Performance Improvement of Multimedia Kernels Using Data- and Thread- Level Parallelism on CPU Platform,” International Congress on High-Performance Computing and Big Data Analysis, pp. 459-467, 2019.