محاسبه تقریبی عملیات تقسیم در داخل شبکه به کمک زبان برنامه‌نویسی پی‌فور

دانیال خراسانی‌زاده
دانشجوی کارشناسی‌ارشد، دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف، تهران
(رایانامه: danial.k77@sharif.edu)

محمدحسین فرحناکیان
دانشجوی دکتری، دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف ، تهران
(رایانامه: mh.farahnakiyan@ce.sharif.edu)

مهدی دولتی
استادیار، دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف، تهران
(رایانامه: mdolati@sharif.edu)

رسول جلیلی
دانشیار، دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف، تهران
(رایانامه: jalili@sharif.edu)

چکیده

لایه داده برنامه‌پذیر اجازه می‌دهد مسیر اصلی پردازش بسته‌ها یک منطق سفارشی‌سازی‌شده را برای افزایش امنیت یا پایش ترافیک در نرخ خط اجرا کند. بااین‌حال، محدودیت‌های زبان برنامه‌نویسی پی‌فور و سخت‌افزارهای میزبان آن، پیاده‌سازی برخی از توابع همچون تقسیم را دشوار کرده‌اند. روش‌های موجود برای عبور از این مانع از جدول‌های جست‌وجو، شبیه‌سازی اعداد نقطه‌شناور وابسته به سخت‌افزار، یا تقریب‌های نقطه‌ثابت با بازه دقت خاص‌منظوره استفاده کرده‌اند که دقت و تعمیم‌پذیری را محدود می‌کند. در این مطالعه، یک الگوریتم تقسیم نقطه‌ثابت بدون استفاده از منابع ذخیره‌سازی و مستقل از معماری سخت‌افزاری ارائه شده که برای لایه داده برنامه‌پذیر مبتنی بر پی‌فور طراحی شده است. این راهکار با هنجارسازی عملیات تقسیم بازه مقادیر قابل پشتیبانی را تا حد قابل‌توجهی گسترش داده، با تقریب خطی محاسبات نقطه‌ثابت استفاده از منابع سخت‌افزاری را کاهش داده، و با بهبود تقریب به کمک روش نیوتن–رافسون امکان تنظیم مصالحه بین دقت و سربار محاسباتی را فراهم کرده است. الگوریتم مذکور با استفاده از زبان پی‌فور پیاده‌سازی شده و با استفاده از مدل رفتاری بی‌ام‌وی۲ مورد ارزیابی قرار گرفته است. ارزیابی ما نشان می‌دهد روش پیشنهادی بدون سربار حافظه و تأثیر 1540 کیلوبیت بر ثانیه‌ای قابل چشمپوشی بر گذردهی، بالاترین سطح دقت را در مقایسه با روش‌های موجود ارائه می‌دهد. با ارائه یک روش تقسیم قابل‌تعمیم و کارآمد، این پژوهش دامنه محاسبات عددی که به صورت مستقیم در لایه داده قابل‌انجام هستند را گسترش داده است.

کلیدواژه‌ها

شبکه‌های کامپیوتری، محاسبات نقطه‌ثابت، ریاضیات محاسباتی، محاسبات درون‌شبکه، لایه داده برنامه‌پذیر، زبان برنامه‌نویسی پی‌فور

۱. مقدمه

شبکه‌های برنامه‌پذیر می‌توانند بسته‌ها را مطابق یک منطق سفارشی‌سازی‌شده درون مسیرهای اصلی[۱] پردازش بسته با نرخ خط[۲] پردازش کنند [۱]. زبان پی‌فور[۳] [۲] طراحی این شبکه‌ها را تسهیل کرده است. این سیر تکاملی، زمینه‌ساز دسته بزرگی از مطالعات پژوهشی در زمینه *پردازش درون‌شبکه*[۴] شده که براساس آن، همه یا بخشی از عملیات محاسباتی و منطقی که پیش‌ازاین توسط کارساز[۵]ها انجام می‌شد به سخت‌افزار لایه داده محول شود. انتقال چنین وظایفی به لایه داده مزایای متعددی از جمله کاهش چشمگیر تاخیر، سربار کمتر بر روی میزبان[۶]ها، و امکان اتخاذ تصمیم‌های بلادرنگ در درون ساختار شبکه را به ارمغان خواهد آورد [۳].

علی‌رغم این مزایا، محاسبات در لایه داده چالش‌هایی نیز به همراه دارد. سوئیچ[۷]های برنامه‌پذیر تنها از برخی توابع محاسباتی مربوط به اعداد صحیح و تعداد دستورات محدود در مسیر اصلی پشتیبانی می‌کنند و بسیاری از محاسبات مربوط به اعداد حقیقی در این دستگاه‌ها در دسترس نیست. به‌عنوان‌مثال، استفاده از انواع کسری، تغییر مکان دودویی[۸] با مقدار متغیر، و محاسبات اعداد نقطه‌شناور[۹]، در لایه داده اغلب ممکن نیست. در همین جهت، مطالعات پیشین رویکردهای متعددی برای گسترش مجموعه عملیات قابل‌انجام در لایه داده ارائه کرده‌اند. برخی از این راهکارها از جدول‌های جست‌وجو[۱۰] و بهینه‌سازی سخت‌افزار بستر اجرای دستورات استفاده کرده‌اند. دسته دیگری از روش‌ها نیز از تقریب اعداد به کمک بسط‌های چندجمله‌ای در قالب اعداد نقطه‌ثابت[۱۱] برای انجام محاسبات استفاده کرده‌اند؛ اما، تمامی این روش‌ها یا سربار ذخیره‌سازی قابل‌توجهی به سخت‌افزار لایه داده تحمیل می‌کنند، یا از دقت محاسباتی کافی برخوردار نیستند،

[۱] Pipeline
[۲] Line-rate
[۳] Programming protocol-independent packet processors (P4)
[۴] In-network computing
[۵] Server
[۶] Host
[۷] Switch
[۸] Binary shift
[۹] Floating-point
[۱۰] Lookup table
[۱۱] Fixed-point

هفتمین کنفرانس ملی انفورماتیک ایران، ۸ و ۹ بهمن ۱۴۰۴

تهران، خیابان شهید لواسانی، جنب برج کوه نور، پلاک ۷۰

و یا به ویژگی‌های خاص در معماری سخت‌افزار وابسته‌اند که منجر به محدودیت تعمیم‌پذیری آن‌ها به عموم سخت‌افزارهای لایه داده برنامه‌پذیر خواهد شد.

یکی از چالش‌های کلیدی این حوزه، امکان انجام عملیات تقسیم در لایه داده است [۴]. چرا که عملیات تقسیم سنگ بنای بسیاری از وظایف کلیدی در حوزه محاسبات درون‌شبکه از جمله هنجارسازی[۱۲] و مقیاس‌بندی[۱۳] در تحلیل‌های آماری، محاسبه احتمال و گرادیان در یادگیری ماشین، و طیف گسترده‌ای از وظایف دیگر است. باوجود ضرورت ارائه یک روش مستقل از معماری سخت‌افزار، بدون اتکا به جدول‌های جست‌وجو، مبتنی بر اعداد نقطه‌ثابت، و با مصرف بهینه حافظه، هیچ‌یک از راهکارهای موجود رویکرد مناسبی برای حل این مشکل ارائه نکرده‌اند. برای برطرف کردن خلأ موجود، این مطالعه راهکاری نوین ارائه کرده که عملیات تقسیم نقطه‌ثابت را آگاه از تمامی قیدهای ذکر شده در لایه داده برنامه‌پذیر مبتنی‌بر پی‌فور پیاده‌سازی می‌کند.

راهکار پیشنهادی با ترکیب تقریب خطی[۱۴] و روش نیوتن–رافسون[۱۵]، کاستی‌های موجود در سایر راهکارها را برطرف کرده است. استفاده از تقریب خطی امکان استخراج تابع معکوس بر روی یک بازه مقیاس‌پذیر را فراهم کرده که تنها با استفاده از ضرب و جمع‌های نقطه‌ثابت انجام خواهد شد. روش نیوتن–رافسون باتوجه‌به همگرایی درجه دوم خود، برای بهبود تخمین اولیه استفاده شده است که با هر دفعه تکرار آن کاهش قابل‌پیش‌بینی‌ای در خطا ایجاد خواهد شد. ترکیب این دو جزء، امکان برقراری مصالحه‌ای بین دقت و مصرف منبع را نیز فراهم می‌آورد.

برخلاف راهکارهای مبتنی بر جدول و وابسته به معماری‌های خاص سخت‌افزاری، الگوریتم پیشنهادی تنها نیازمند عملیات محاسباتی قابل تعریف در زبان پی‌فور است. به‌منظور امکان‌سنجی استفاده از راهکار پیشنهادی در لایه داده برنامه‌پذیر، الگوریتم ارائه شده با استفاده از زبان برنامه‌نویسی پی‌فور توسعه داده شده و برای ارزیابی در مدل رفتاری بی‌اِم‌وی۲[۱۶] [۵] پیاده‌سازی و اجرا شده است.

ارزیابی‌های صورت گرفته نشان می‌دهند که باوجود افزایش سربار محاسباتی به میزان محدود، راهکار پیشنهادی با حذف کامل مصرف منابع ذخیره‌سازی سوئیچ، به دقتی یکسان با بهترین روش‌های موجود دست یافته است. به صورت خاص، رویکرد پیشنهادی تنها با سربار میانگین 1540 کیلوبیت بر ثانیه بر گذردهی، توانسته 853 بایت میانگین مصرف حافظه را کاهش دهد و خطای محاسبه را به میزان بیش از دو مرتبه بزرگی در مقایسه با روش‌های پایه کمتر کند. به‌طورکلی، دستاوردهای کلیدی این پژوهش عبارت‌اند از:

- مرور نظام‌مند و پیاده‌سازی راهکارهای موجود برای انجام عملیات تقسیم در لایهٔ دادهٔ برنامه‌پذیر.
- ارائهٔ یک الگوریتم تقسیم نقطه‌ثابت مستقل از معماری سخت‌افزاری و بدون نیاز به مصرف منابع ذخیره‌سازی سخت‌افزار لایه داده.
- ترکیب هنجارسازی تقسیم، تقریب خطی، و روش نیوتون برای نخستین بار در بستر زبان برنامه‌نویسی پی‌فور به‌منظور انجام عملیات تقسیم نقطه‌ثابت.
- پیاده‌سازی کامل الگوریتم پیشنهادی در محیط شبیه‌سازی بی‌اِم‌وی۲ و ارزیابی تجربی آن در مقایسه با راهکارهای پیشین.
- انتشار پیاده‌سازی روش پیشنهادی به‌صورت متن‌باز، با هدف تضمین قابلیت بازتولید نتایج و تسهیل پژوهش‌های آتی[۱۷].

سایر بخش‌های این مقاله به این ترتیب تنظیم شده‌اند. بخش ۲ راهکارهای محاسبات با استفاده از زبان برنامه‌نویسی پی‌فور را مرور می‌کند. بخش ۳ راهکار پیشنهادی را شرح می‌دهد. جزئیات پیاده‌سازی الگوریتم ارائه شده با استفاده از زبان برنامه‌نویسی پی‌فور در بخش ۴ ارائه شده است. بخش ۵ به ارزیابی و تحلیل نتایج اختصاص دارد و در نهایت بخش ۶ این پژوهش را جمع‌بندی می‌کند.

۲. پیشینه پژوهش

ظهور و توسعه لایه داده برنامه‌پذیر و معماری پیزا[۱۸] [۶]، بستری مناسب برای انتقال برخی از توابع محاسباتی به داخل شبکه را فراهم آورده است. این بستر در بسیاری از حوزه‌های پژوهشی همچون اندازه‌گیری شبکه، امنیت، و یادگیری ماشین [۱]، به‌عنوان یک ابزار حائز اهمیت برای بهبود سرعت محاسبات و تقسیم بار پردازشی مطرح شده است. علاوه بر این، توسعه زبان‌های برنامه‌نویسی خاص‌منظوره همچون پی‌فور، قابلیت نگاشت منطق سطح بالای تطبیق–اقدام[۱۹] را درون مسیرهای اصلی‌ای فراهم کرده که با نرخ خط عمل می‌کنند؛ اما از طرفی، سخت‌افزارهای میزبان این زبان‌ها به دلیل نیاز به تضمین گذردهی بالا، محاسبات با زمان متغیر را محدود کرده‌اند. [۷]. همچنین، زبان پی‌فور از محاسبات عدد صحیح با طول ثابت پشتیبانی می‌کند. درنتیجه توابعی از جمله معکوس، لگاریتم، و توان باید از طریق محاسبات تکراری ساده‌تر با تعداد تکرار ثابت پیاده‌سازی شوند. به همین منظور، چندین راهکار در جهت گسترش سوئیچ‌های برنامه‌پذیر برای انجام تقریبی از محاسبات اعشاری ارائه شده‌اند. در ادامه، این راهکارها را در دو دستهٔ شبیه‌سازی محاسبات نقطه‌شناور و محاسبات نقطه‌ثابت بررسی خواهیم کرد.

۲-۱. محاسبات نقطه‌شناور

راهکارهای متنوعی در جهت پیاده‌سازی یک شبیه‌سازی تقریبی از محاسبات نقطه‌شناور در لایه داده برنامه‌پذیر ارائه شده‌اند. یکی از این راهکارها، استفاده از جداول جست‌وجو برای محاسبه تقریبی مقادیر است. در این روش، مقادیر تقریبی حاصل از عملیات محاسباتی در جدول جست‌وجوی سوئیچ ذخیره خواهند شد. به‌عنوان‌مثال، راهکار پیشنهادی سویی و همکاران [۸]، یک روش جدول‌محور است که عملیات نقطه‌شناور را با ذخیره نتایج تقریبی بازه‌های عددی مختلف

[۱۲] Normalization
[۱۳] Scaling
[۱۴] Linear approximation
[۱۵] Newton-Raphson method
[۱۶] Behavioral Model version 2 (BMv2)
[۱۷] پیاده‌سازی ارائه‌شده پس از پذیرش مقاله به‌صورت عمومی منتشر خواهد شد.
[۱۸] Protocol-independent switch architecture (PISA)
[۱۹] Match-Action

در جدول‌های بزرگ انجام می‌دهد و امکان تقریب این مقادیر با دقت بالا را فراهم خواهد کرد. این جدول‌های بزرگ برای حافظه محدود سخت‌افزار لایه داده مناسب نیستند؛ به‌همین‌علت، جدول‌های مذکور در این راهکار، به چندین جدول جست‌وجوی کوچک‌تر تجزیه خواهند شد و در جداول جست‌وجوی سوئیچ‌ها ذخیره خواهند شد.

ژوزه و همکاران [۹]، در جهت انجام محاسبات نقطه‌شناور، از نمایش نیم‌دقت[۲۰] استفاده کرده‌اند. در این راهکار، زیرعبارت‌هایی با دامنه محدود به جداول جست‌وجو نگاشت خواهند شد که تقریبی از محاسبات را برای ما فراهم خواهند کرد. این روش‌ها نیاز به حافظه را تا حد قابل‌قبولی کاهش می‌دهند و تقریب به‌نسبت دقیقی از اعداد ارائه می‌کنند؛ اما، اتکای به جداول جست‌وجو بهینگی آن‌ها را از نظر حافظه مصرفی تحت‌تأثیر قرار خواهد داد.

برخی از روش‌ها برای نمایش و محاسبات اعداد نقطه‌شناور، نمایش‌های جدید و نوآورانه ارائه کرده‌اند. یوان و همکاران [۱۰] نمونه‌ای از این روش‌ها هستند که با ارائه یک نمایش جدید از اعداد نقطه‌شناور که پردازش پایه[۲۱] و نما[۲۲] را از هم جدا می‌کند، امکان انجام محاسبات بر روی این اعداد را در سوئیچ‌های مبتنی بر معماری پیزا فراهم کرده‌اند. هرچند که برای حفظ گذردهی[۲۳] سوئیچ‌ها در صورت استفاده از این نمایش، بایستی تغییرات سخت‌افزاری در آن‌ها اعمال شود.

به‌طورکلی، این دسته از راهکارها امکان استفاده و انجام محاسبات بر روی اعداد نقطه‌شناور را در لایه داده برنامه‌پذیر فراهم می‌کنند؛ اما به‌شدت به جدول‌های جست‌وجو و یا تغییرات و بهینه‌سازی‌های سخت‌افزاری خاص وابسته هستند. در نتیجه هیچ‌یک از آن‌ها سازوکار مناسبی برای پیاده‌سازی عملیات تقسیم در چارچوب زبان برنامه‌نویسی پی‌فور ارائه نخواهند کرد.

۲-۲. محاسبات نقطه‌ثابت

دسته دیگری از راهکارهای محاسباتی در لایه داده، از محاسبات و اعداد نقطه‌ثابت استفاده می‌کنند. این محاسبات، با توانایی‌ها و امکانات ارائه شده توسط زبان پی‌فور و سخت‌افزار میزبان آن تطابق بیشتری دارند. مثال‌های بارز این دسته از روش‌ها، راهکارهای پیشنهادی پاتل و همکاران [۴] و سادا و همکاران [۱۱] هستند. در روش ارائه شده در [۴]، اعداد به سیستم نقطه‌ثابت نگاشت می‌شوند و به جای نمونه‌برداری از توابع محاسباتی و ذخیره آن‌ها در جدول‌های سوئیچ، از بسط تیلور [۱۲] برای تقریب زدن آن‌ها استفاده خواهد شد. این روش با کاهش محاسبات مورد نیاز در لایه داده، به مصرف حافظه ثابت و زمان خطی برای انجام محاسبات نیاز خواهد داشت. این روش بهبود چشمگیری به نسبت روش‌های مبتنی بر اعداد نقطه‌شناور دارد؛ اما به علت استفاده از ثبّات‌ها، حافظه سوئیچ که یک منبع ارزشمند در عملکرد این دستگاه‌های سخت‌افزاری است را مشغول می‌کند.

جدول ۱ مقایسه‌ای از کارهای پیشین را از نظر معیارهای زیر ارائه می‌دهد: (۱) نقطه‌ثابت یا نقطه‌شناور بودن محاسبات، (۲) امکان تنظیم دقت متغیر، (۳) استفاده از جدول‌های جست‌وجو، (۴) استفاده از ثبات‌ها، (۵) استقلال از معماری سخت‌افزار، و (۶) نیاز به معین کردن بازه تقریب از پیش تعیین شده. در مجموع، هیچ‌یک از راهکارهای موجود موفق به کاهش میزان حافظه موردنیاز و درعین‌حال، ارائه یک تقریب با دقت بالا از اعداد و محاسبات در لایه داده نشده‌اند. این مطالعه با کاهش مصرف منابع، افزایش دقت و فراهم کردن مصالحه بین دقت و مصرف منابع به چالش‌های موجود پاسخ داده و کاستی‌های مطالعات پیشین را برطرف کرده است.

۳. راهکار پیشنهادی

در این بخش، راهکار پیشنهادی برای تقسیم نقطه‌ثابت بدون استفاده از منابع ذخیره‌سازی سوئیچ ارائه می‌شود. این راهکار منطبق بر زبان پی‌فور است و امکان مصالحه بین دقت و مصرف منابع را فراهم می‌کند. راهکار پیشنهادی در چهار مرحله کار می‌کند: ابتدا مقدار مقسوم‌علیه بِهَنجار می‌شود. سپس یک تقریب اولیه خطی محاسبه شده که در ادامه با استفاده از روش نیوتن–رافسون بهبود می‌یابد. در نهایت، با تغییر مکان دودویی، عبارت از حالت هنجارسازی شده خارج شده و با ضرب در مقسوم، عملیات تقسیم تکمیل می‌شود.

۳-۱. معادله‌نویسی مسئله

هدف غائی از طراحی این الگوریتم، محاسبه مقدار

$$q = \frac{a}{b} \qquad (۱)$$

است که در معادله (۱)، مقادیر a و $b \neq 0$ ، اعدادی هستند که در لایه داده مبتنی بر پی‌فور به صورت نقطه‌ثابت کدگذاری شده‌اند.

۳-۲. کدگذاری نقطه‌ثابت

یک عدد نقطه‌ثابت معمولا به صورت $Q_{I.F}$ نمایش داده می‌شود که در آن I بیانگر تعداد بیت[۲۴]های صحیح و F نشان دهنده تعداد بیت‌های اعشاری است. برای تبدیل یک عدد صحیح و یا یک عدد نقطه‌شناور به یک عدد نقطه‌ثابت در پایه دو از معادله (۲)، و برای تبدیل یک عدد نقطه‌ثابت به یک عدد نقطه‌شناور از معادله (۳) استفاده خواهد شد.

$$n_{Q_{I.F}} = \lfloor 2^F n \rfloor \qquad (۲)$$

$$n = \frac{n_{Q_{I.F}}}{2^F} \qquad (۳)$$

۳-۳. الگوریتم عملیات تقسیم

همانطور که پیش‌تر نیز گفته شد، عملیات تقسیم در راهکار پیشنهادی در چهار گام (۱) هنجارسازی (۲) حدس خطی (۳) بهبود حدس، و (۴) خروج از حالت بِهَنجار و انجام ضرب در تابع معکوس انجام شده است. در ادامه روش انجام و پایه‌های ریاضی هر یک از مراحل بررسی خواهد شد.

۳-۳-۱ هنجارسازی اعداد

یکی از محدودیت‌های اصلی تمامی روش‌های محاسبات تقریبی، دقت بالای آن‌ها حول یک نقطه خاص و دقت پایین در سایر نقاط است.

---

[۲۰] Half precision
[۲۱] Mantissa
[۲۲] Exponent
[۲۳] Throughput
[۲۴] Bit

جدول ۱: مروری بر راهکارهای موجود

راهکار	نوع محاسبات	تنظیم دقت	مستقل از جدول	مستقل از ثبات	مستقل از معماری	عدم تعیین بازه تقریب
یوان و همکاران [۱۰]	نقطه‌شناور	✗	✓	✓	✗	✓
سنکاران و همکاران [۱۳]	نقطه‌شناور	✗	✗	✓	✓	✓
ژوزه و همکاران [۹]	نقطه‌شناور	✗	✗	✓	✓	✓
سویی و همکاران [۸]	نقطه‌شناور	✗	✗	✓	✓	✓
پاتل و همکاران [۴]	نقطه‌ثابت	✓	✓	✗	✓	✗
راهکار پیشنهادی	**نقطه‌ثابت**	✓	✓	✓	✓	✓

به دلیل این محدودیت، استفاده صرف از هر یک از روش‌های تقریبی منجر به خطای زیاد در اکثر نقاط خواهد شد و صرفاً می‌توان با دانش به دامنه خاصی از اعداد که مناسب یک نیازمندی خاص هستند، تنها حول همان دامنه از تقریب استفاده کرد. باتوجه‌به اینکه این اتفاق منجر به محدودیت در محاسبات خواهد شد، نیاز به روشی وجود دارد که بتوان بوسیله آن تابع مورد نظر را در دامنه کوچک‌تری از اعداد تقریب زده و سپس با استفاده از روابط ریاضی، تقریب در این دامنه کوچک را به پاسخ نهایی تبدیل کرد. برای انجام این کار می‌توان از یک فرایند هنجارسازی مطابق رابطه (۴) استفاده کرد.

$$\forall b \in \mathbb{R} \setminus \{0\}\ \exists k \in \mathbb{Z},\ n \in \mathbb{R}: \\ \big[b = 2^k n \ \wedge\ \big((1 \le n < 2)\ \vee\ (-2 < n \le -1)\big)\big]. \tag{۴}$$

باتوجه‌به رابطه (۴)، برای هنجارسازی مخرج تابع تقسیم می‌توان از طریق معادله (۵) اقدام کرد.

$$\frac{a}{b} = a \times \frac{1}{2^k n} = a \times 2^{-k} \times \frac{1}{n} \tag{۵}$$

رویکرد مذکور، این امکان را فراهم خواهد کرد که پس از تقریب زدن مقدار تقسیم $\frac{1}{n}$، با استفاده از عملیات تغییر مکان دودویی تاثیر ضرب در 2^{-k} را بر آن اعمال کرده و در پایان، مقدار نهایی محاسبه شده برای مخرج را در a ضرب کرده تا مقدار نهایی عبارت $\frac{a}{b}$ حاصل شود. باتوجه‌به آنچه که شرح داده شد، مسئله‌ای که برای انجام عملیات تقسیم نیاز به حل آن داریم، یافتن یک تقریب خوب برای مقدار $\frac{1}{n}$، در شرایطی که $1 \le n < 2$ برقرار باشد است. (در ادامه برای سادگی، اعداد منفی در نظر گرفته نمی‌شوند اما روش پیشنهادی قابل تعمیم به اعداد منفی نیز خواهد بود.)

۳-۳-۲ تقریب خطی

حدس‌های چندجمله‌ای و به طور خاص حدس‌های خطی، یکی از روش‌های ساده و کارآمد برای تخمین مقادیر توابع غیرخطی در بازه‌های کوچک هستند. به طور کلی، الگوریتم‌های کمینه‌ـ‌بیشینه تخمین تلاش می‌کنند که در بازه‌ای محدود، حداکثر میزان خطای یک تقریب چندجمله‌ای برای یک تابع خاص را کمینه کنند. همچنین، روش‌های ریاضیاتی مانند بسط تیلور و چندجمله‌ای‌های چبیشف نیز، برای تخمین توابع به صورت چندجمله‌ای استفاده می‌شوند. با محاسبه یک تخمین خطی برای تابع $\frac{1}{n}$ در بازه $[1,2]$، به دو عبارت $1.42082 - 0.5n$ و $1.40812 - 0.47664n$ خواهیم رسید که عبارت اول، بیشینه خطا در بازه را کمینه کرده و عبارت دوم، مجموع مربعات خطاها را کمینه خواهد کرد. باتوجه‌به اینکه ضرایب این عبارت‌ها مقدار پیچیده‌ای داشته و تاثیر میزان کاهش خطای آن‌ها در ادامه ناچیز خواهد بود، عبارت (۶) را به عنوان یک تقریب خطی با خطای نسبتاً کم برای تابع $\frac{1}{n}$ انتخاب می‌کنیم.

$$\frac{1}{n} \approx 1.5 - 0.5n \tag{۶}$$

۳-۳-۳ بهبود تقریب بوسیله روش نیوتون‌ـ‌رافسون

در تحلیل عددی، روش نیوتون‌ـ‌رافسون راهکاری مناسب برای پیدا کردن تقریبی ریشه‌های یک تابع حقیقی است. این روش با یک حدس x_0 از ریشه تابع شروع کرده و اگر این حدس به ریشه نزدیک بوده و تابع شروط خاصی را برآورده کند، با تکرار رابطهٔ (۷)، به تقریب دقیق‌تری از ریشه دست خواهد یافت.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{۷}$$

در این رابطه، مقدار x_{n+1} تقریبی دقیق‌تر از x_n است. با استفاده از روش مذکور و برای تقریب زدن مقدار $\frac{1}{n}$ می‌توان از ریشه تابع $f(x) = \frac{1}{x} - n$ استفاده کرد. برای محاسبه این تقریب پس از تولید حدس اولیه، به کمک مقدار به‌دست‌آمده و رابطه ساده شده (۸)، تقریب محاسبه شده بهبود خواهد یافت.

$$x_{t+1} = x_t(2 - n x_t) \tag{۸}$$

۳-۳-۴ خروج از شکل هنجارسازی شده و ضرب

با تبدیل مقدار مقسوم‌علیه به یک قالب هنجارسازی شده، مقدار آن در این قالب تخمین زده خواهد شد. سپس برای تکمیل عملیات تقسیم، باید این قالب هنجارسازی شده با ضرب در مقدار 2^{-k} که در مراحل قبلی محاسبه شده بود، از قالب بِهَنجار خارج شود. با انجام این کار، مراحل محاسبه معکوس مقسوم‌علیه تکمیل خواهد شد و در نتیجه می‌توان با ضرب مقسوم در این مقدار محاسبه شده، به مقدار نهایی تقسیم دست یافت.

۳-۴. مصالحه بین دقت و مصرف منابع

با توجه به اینکه هر یک از وظایف درون‌شبکه دارای الزامات متفاوتی از نظر سرعت اجرا، میزان مصرف منابع، و میزان دقت خروجی هستند، امکان تنظیم پیراسنجه[۲۵] های فرایند بهبود حدس، می‌تواند نقش مهمی در بهینه‌سازی این وظایف ایفا کند. به این معنا که کاربر قادر است با تغییر تعداد تکرارهای مرحله بهبود حدس متناسب با ویژگی‌ها و نیازمندی‌های موردنظر خود، تعادلی میان دقت نهایی و میزان منابع

[۲۵] Parameter

مصرفی برقرار سازد. این رویکرد می‌تواند به‌ویژه در سامانه‌هایی که محدودیت‌های محاسباتی دارند یا در فرنامه[۲۶]هایی که سرعت پاسخ‌گویی نسبت به دقت اولویت بیشتری دارد، منجر به بهبود کارایی کلی سیستم و بهره‌برداری بهینه از منابع موجود شود.

۴. پیاده‌سازی در پی‌فور

در این بخش، روش نگاشت الگوریتم ارائه شده در بخش ۳، با استفاده از مسیر اصلی مبتنی بر زبان برنامه‌نویسی پی‌فور شرح داده شده است. برای پیاده‌سازی روش پیشنهادی با استفاده از زبان برنامه‌نویسی پی‌فور و به‌منظور سهولت در توسعه‌پذیری، از یک برنامه نوشته شده به زبان برنامه‌نویسی پایتون[۲۷] [۱۴] برای تولید برنامه پی‌فور استفاده شده است. این برنامه با گرفتن عواملی مانند دقت اعشار در ساختار نقطه‌ثابت به‌عنوان ورودی، برنامه پی‌فور متناظر با این عوامل را تولید خواهد کرد. در ادامه، پیاده‌سازی هر یک از گام‌های روش پیشنهادی و همچنین برخی از اعمال پشتیبان که در روند اجرای الگوریتم موردنیاز خواهند بود، با استفاده از زبان برنامه‌نویسی پی‌فور مورد بررسی قرار گرفته است.

۴-۱. پروتکل طراحی شده برای استفاده از روش پیشنهادی

برای پیاده‌سازی راهکار پیشنهادی پروتکلی طراحی شده است تا با گرفتن دو عدد، مقدار حاصل تقسیم آن‌ها بر یکدیگر را برای ارزیابی به سمت لایه کنترل[۲۸] ارسال کند. این پروتکل به عنوان یک سرآیند[۲۹] بر روی سرآیند اترنت [۳۰] قرار می‌گیرد. خصیصه‌های این سرآیند شخصی‌سازی شده عبارتند از: (۱) نوع درخواست که بسته به مبدأ و مقصد بسته (ارسال به سمت لایه داده یا لایه کنترل) می‌تواند مقدار صفر یا یک داشته باشد. (۲) مقدار عدد مقسوم که اندازه آن با توجه به دقت تعریف شده اعداد نقطه‌ثابت در نظر گرفته خواهد شد. (۳) مقدار عدد مقسوم‌علیه. (۴) مقدار حاصل تقسیم. (۵) میزان زمان مورد نیاز برای انجام محاسبات مربوط به عملیات تقسیم در لایه داده. برنامه پی‌فور مربوط به پروتکل طراحی شده، در قطعه کد ۱ ارائه شده است.

```
typedef bit<32> number_req_t;
header calc_t {
    bit<8> type;
    number_req_t a;
    number_req_t b;
    number_req_t result;
    bit<48> time_delta;
}
```

کد ۱: پروتکل طراحی شده برای ارزیابی روش‌های تقسیم در لایه داده برای دقت ۱۶ بیت صحیح و ۱۶ بیت اعشاری در پی‌فور

۴-۲. ضرب نقطه‌ثابت

عملیات ضرب در ساختار نقطه‌ثابت، در ابتدا دقیقاً مشابه ضرب دو عدد صحیح انجام خواهد شد؛ اما ازآنجاکه با انجام عمل ضرب، ارزش مکانی مقادیر حاصل‌ضرب به‌هم می‌ریزد، در پایان بایستی با استفاده از عمل تغییر مکان دودویی و با تغییر مکان حاصل به راست (به اندازه دقت اعشار نقطه‌ثابت)، به جواب نهایی ضرب رسید. نحوه پیاده‌سازی این فرایند در قطعه کد ۲ قابل‌مشاهده است.

```
typedef int<32> fixed;
...
action fixed_mul(fixed a, fixed b){
    meta.temp_A = (fixed)(((int<64>)a * (int<64>)b) >> 16);
}
```

کد ۲: ضرب نقطه‌ثابت با دقت ۱۶ بیت صحیح و ۱۶ بیت اعشاری در پی‌فور (اعداد ابتدا به اندازه بزرگ‌تری برده می‌شوند تا از سرریز در هنگام ضرب جلوگیری شود)

۴-۳. هنجارسازی عدد

زبان برنامه‌نویسی پی‌فور، محدودیت‌های خاصی داشته و از حلقه‌های تکرار پشتیبانی نخواهد کرد. به همین خاطر عملیات هنجارسازی که در حالت عادی می‌توان آن‌را با یک حلقه پیاده‌سازی کرد، باید با تکرار خطوط کد به اندازه تعداد دفعات مورد نیاز پیاده‌سازی شود. برای هنجارسازی یک عدد در پی‌فور پس از دریافت آن، به تعداد دفعات برابر با تعداد بیت‌های صحیح آن عدد در ساختار نقطه‌ثابت، ابتدا بررسی خواهد شد که عدد بزرگ‌تر از دو باشد. در صورت برقرار بودن این شرط، با استفاده از تغییر مکان دودویی به راست، آن را تقسیم بر دو کرده و مقدار متغیر تعداد دفعات تغییر مکان به راست را یک عدد افزایش خواهیم داد. پس از این، به تعداد دفعات برابر با تعداد بیت‌های اعشاری عدد در ساختار نقطه‌ثابت، ابتدا بررسی خواهد شد که عدد کوچک‌تر از یک باشد. در صورت برقرار بودن این شرط، با استفاده از تغییر مکان دودویی به چپ آن را دو برابر کرده و مقدار متغیر دفعات تغییر مکان به راست را یک عدد کاهش خواهیم داد. در نهایت، عدد خروجی برابر با مقدار n، و تعداد تغییر مکان‌های به راست، برابر با مقدار k خواهد بود. پیاده‌سازی این فرایند با استفاده از زبان پی‌فور، در قطعه کد ۳ آمده است.

```
action normalize(fixed b) {
    meta.norm_val = b;
    meta.shift_amount = 0;
    if (meta.norm_val >= FIXED_TWO) {
        meta.norm_val = meta.norm_val >> 1;
        meta.shift_amount = meta.shift_amount + 1;
    }
    ...
    if (meta.norm_val < FIXED_ONE) {
        meta.norm_val = meta.norm_val << 1;
        meta.shift_amount = meta.shift_amount - 1;
    }
}
```

کد ۳: هنجارسازی عدد نقطه‌ثابت با دقت ۱۶ بیت صحیح و ۱۶ بیت اعشاری در پی‌فور

۴-۴. پیاده‌سازی روش نیوتون–رافسون

برای پیاده‌سازی روش نیوتون–رافسون ابتدا حدس خطی را محاسبه کرده، و سپس با انجام عملیات شرح داده شده در بخش ۳-۳-۳، و با استفاده از عملیات ضرب نقطه‌ثابت، دقت خطای حدس را کاهش خواهیم داد. مسیر انجام این عملیات با زبان پی‌فور، در قطعه کد ۴ ارائه شده است.

---
[۲۶] Scenario
[۲۷] Python
[۲۸] Control-plane
[۲۹] Header
[۳۰] Ethernet

```
meta.temp_B = (FIXED_ONE + (FIXED_ONE >> 1)) - (meta.norm_val >> 1);
fixed_mul(meta.norm_val, meta.temp_B);
fixed_mul(meta.temp_B, (FIXED_TWO - meta.temp_A));
meta.temp_B = meta.temp_A;
fixed_mul(meta.norm_val, meta.temp_B);
fixed_mul(meta.temp_B, (FIXED_TWO - meta.temp_A));
```

کد ۴: پیاده‌سازی حدس خطی و دو تکرار از روش نیوتون–رافسون در پی‌فور

۴-۵. پیاده‌سازی تقسیم بر توان ۲

پیاده‌سازی مراحل خارج کردن عدد از شکل بِهَنجار و تقسیم بر عددی که توانی از ۲ است، با توجه به عدم دسترسی به حلقه‌های تکرار، با تغییر مکان به راست به اندازه حداکثر میزان تغییر مکان به راست ممکن قابل انجام است. این مقدار برابر با عدد بیشتر بین دو مقدار دقت اعشاری و دقت صحیح در دستگاه نقطه‌ثابت خواهد بود. باتوجه‌به مثبت یا منفی بودن مقدار تعداد تغییر مکان‌های به راست، حاصل را به راست یا چپ تغییر مکان داده و از مقدار تغییر مکان‌ها کم کرده و یا به آن اضافه خواهیم کرد تا مقدار صفر حاصل شود. پس از انجام این عملیات، عدد حاصل، مقدار تقسیم عدد اولیه بر توان ۲ مورد نظر خواهد بود. پیاده‌سازی انجام این فرایند در قطعه کد ۵ قابل‌مشاهده است.

```
if (meta.shift_amount > 0) {
    meta.temp_A = meta.temp_A >> 1;
    meta.shift_amount = meta.shift_amount - 1;
} else if (meta.shift_amount < 0) {
    meta.temp_A = meta.temp_A << 1;
    meta.shift_amount = meta.shift_amount + 1;
}
```

کد ۵: پیاده‌سازی تغییر مکان دودویی برای تقسیم بر توان ۲ در پی‌فور

۵. نتایج و ارزیابی

در این بخش، ارزیابی الگوریتم پیشنهادی برای محاسبه تقسیم نقطه‌ثابت در لایه داده برنامه‌پذیر از دیدگاه دقت، گذردهی، میانگین زمان محاسبه، و حافظه مصرفی مورد بررسی و تحلیل قرار گرفته است. هدف از این ارزیابی (۱) سنجش کمّی دقت عددی عملیات تقسیم در پیکربندی‌های مختلف، (۲) بررسی مزایا و معایب استفاده از این الگوریتم در مقایسه با سایر روش‌ها، و (۳) امکان‌سنجی پیاده‌سازی این الگوریتم با استفاده از زبان برنامه‌نویسی پی‌فور و آگاه از محدودیت‌های لایه داده برنامه‌پذیر است.

۱-۵. برپایش ارزیابی

ارزیابی با استفاده از مدل رفتاری بی‌اِم‌وی۲، یک پیاده‌سازی مرجع و نرم‌افزاری از معماری پیزا انجام شده است. یکی از پیکربندی‌های بی‌اِم‌وی۲، ابزار سیمپل–سوئیچ[۳۱] است که یک سوئیچ مجازی با قابلیت پشتیبانی از جدول‌های تطبیق–اقدام و پردازش انعطاف‌پذیر سرآیندها را ارائه خواهد داد. برای انجام آزمایش‌های این پژوهش از ابزار سیمپل–سوئیچ نسخه 1.15.0-68f4a978 مترجم پی‌فور-p4c-bm2-ss [۱۵] با نسخه 1.2.5.8 (SHA: 2265f8045 BUILD: Release) بر روی یک ماشین مجازی مجهز به سیستم‌عامل اوبونتو[۳۲] نسخه ۲۴.۰۴ و ۶ هسته پردازنده از نوع 13th Gen Intel® Core™ i7-13620H و ۸ گیگابایت حافظه دسترسی تصادفی استفاده شد.

در ارزیابی روش پیشنهادی، از چهار دقت نقطه‌ثابت که هر یک ۱۶ بیت صحیح و به ترتیب ۸، ۱۶، ۲۴، و ۳۲ بیت اعشاری دارند استفاده شده است. برای ارزیابی بر روی هر یک از دقت‌های انتخابی، دو آزمایش انجام شده که در آزمایش اول مقادیر دو عدد a و b از بازه $[1, 100]$ انتخاب شده و در آزمایش دوم مقدار a برابر با یک، و مقدار b در بازه $[1, 15000]$ در نظر گرفته شده است. در هر آزمایش، پس از انتخاب دو عدد مقسوم و مقسوم‌علیه، این دو عدد در لایه کنترل به دقت نقطه‌ثابت مورد نظر تبدیل شده و به سمت سوئیچ ارسال می‌شوند. سوئیچ پس از انجام عمل تقسیم بر روی این دو عدد، مقدار حاصل را به همراه زمان انجام عملیات به سرآیند محاسبه اضافه کرده و برای لایه کنترل ارسال خواهد کرد. لایه کنترل پس از دریافت پاسخ، آن را به نمایش نقطه‌شناور تبدیل کرده و خطای پاسخ لایه داده را نسبت به تقسیم نقطه‌شناور محاسبه خواهد کرد.

۲-۵. معیارهای ارزیابی

ارزیابی راهکار پیشنهادی در مقایسه با روش‌های پایه بر اساس چهار معیار (۱) میانگین زمان محاسبه، (۲) گذردهی، (۳) میانگین قدر مطلق خطا، و (۴) میزان حافظه مصرفی صورت گرفته که هر کدام در ادامه شرح داده شده‌اند.

میانگین زمان محاسبه: در ارزیابی انجام شده، مدت زمانی که بسته در مسیر اصلی ورودی یک سوئیچ برنامه‌پذیر برای انجام محاسبات سپری می‌کند تحت عنوان زمان محاسبه در نظر گرفته شده است که از رابطهٔ (۹) و در واحد میکروثانیه قابل محاسبه است.

$$delay(\mu s) = t_d - t_a \qquad (۹)$$

در رابطهٔ (۹)، t_a، بیانگر زمان ورود به مسیر اصلی ورودی و t_d، نشان‌دهنده زمان خروج از مسیر اصلی ورودی است. همچنین، با استخراج و محاسبه میانگین حسابی این مقدار به‌ازای تمامی بسته‌ها، مقدار میانگین زمان محاسبه به‌دست خواهد آمد.

گذردهی: گذردهی یک سیستم تحت لایه داده برنامه‌پذیر بر اساس تعداد بسته‌هایی که در واحد زمان از یک مسیر مشخص عبور خواهند کرد قابل سنجش است. این معیار به کمک رابطهٔ (۱۰) محاسبه خواهد شد.

$$throughput(kb/s) = \frac{packets\ processed}{time} \qquad (۱۰)$$

میانگین قدر مطلق خطا: به منظور سنجش دقت عملیات تقسیم صورت گرفته در لایه داده، مقادیر محاسبه شده و بازگردانده شده به لایه کنترل، با مقادیر به‌دست آمده از دستورات زبان برنامه‌نویسی پایتون (که در قید محدودیت‌های زبان برنامه‌نویسی پی‌فور نیستند)، مبنی بر رابطهٔ (۱۱) محاسبه خواهد شد.

$$\Delta = |q_{\text{fp}} - q_{\text{ref}}| \qquad (۱۱)$$

در رابطه (۱۱)، q_{fp}، بیانگر دقت به‌دست‌آمده از عملیات تقسیم

[۳۱] Simple-switch

[۳۲] Ubuntu

نقطه‌ثابت با زبان پی‌فور و q_{ref}، نشان‌دهنده دقت محاسبه شده با زبان پایتون خواهد بود.

میزان حافظه مصرفی: در روش‌های مبتنی بر جدول جست‌وجو که به‌عنوان روش پایه اننتخاب شده‌اند، تعداد جدول‌ها، خانه‌های هر جدول، و اندازه داده‌های ذخیره شده در هر خانه، نشان‌دهنده میزان حافظه مورد نیاز برای آن روش خواهند بود.

۵-۳. روش‌های پایه

روش پیشنهادی در این مطالعه، در برابر روش‌های پایهٔ زیر ارزیابی خواهد شد.

۵-۳-۱ جدول جست‌وجو

```
action get_reciprocal_value(fixed reciprocal_frac_part) {
    meta.temp_A = reciprocal_frac_part;
}
table reciprocal_lut_table {
    key ={ meta.lut_index : exact; }
    actions = { get_reciprocal_value; NoAction; }
    default_action = NoAction;
    size = 1024;
}
...
apply {
    normalize(b);
    meta.lut_index = (fixed)((meta.norm_val - FIXED_ONE) >> 6);
    if(meta.lut_index > 1023){
        meta.lut_index = 1023;
    }
    reciprocal_lut_table.apply();
    ...
}
```

کد ۶: پیاده‌سازی یک جدول جست‌وجو با اندازه ۱۰۲۴ برای اعداد نقطه‌ثابت با دقت ۱۶ بیت صحیح و ۱۶ بیت اعشاری

در این رویکرد، بازهٔ مورد نظر به بخش‌های مساوی تقسیم شده و مقدار تابع در نقطهٔ نمایندهٔ هر بخش محاسبه و در جدول ذخیره می‌شود. هنگام اجرا نیز با یافتن نزدیک‌ترین مقدار جدول به ورودی، مقدار متناظر تابع به‌عنوان تقریب ارائه می‌گردد. به بیان دیگر، با داشتن مجموعه‌ای از نقاط $T = \{(x_1, x_2, \ldots, x_n\}, x_i < x_{i+1}$ که مقادیر تابع در آن‌ها از پیش محاسبه شده است، مقدار تقریبی تابع برای ورودی دلخواه با توجه به عبارت (۱۲) محاسبه خواهد شد. استفاده از روش جدول جست‌وجو می‌تواند به تنهایی و یا به همراه یک جدول جست‌وجوی کوچک جایگزین حدس خطی در روش بهبود نیوتون–رافسون استفاده شود.

$$\hat{f}(x) = f(x_i), x_i = \min_{t \in T} |x - t| \qquad (۱۲)$$

یکی از ساختارهای اصلی در زبان پی‌فور، جدول‌های تطبیق–اقدام هستند که در زمان اجرا با بررسی کلیدهای ازپیش‌تعریف‌شده، در صورت وجود ورودی متناظر اقدام مربوطه را اجرا خواهند کرد. این جدول‌ها امکان انجام تقریب عددی مبتنی بر جست‌وجوی جدول را به این صورت فراهم ساخته که پس از محاسبه مقدار هنجارسازی‌شده عدد، با یافتن نزدیک‌ترین مقدار در جدول جست‌وجو، مقدار تقریبی تابع برای ورودی مورد نظر را مشخص خواهند کرد. در پیاده‌سازی پیشنهادی، برای ساخت جدول جست‌وجو، مطابق شبه کد ۱ اقدام خواهد شد.

الگوریتم ۱ فرایند ساخت جدول جست‌وجو

for $i = 1$ to N **do**
  $x \leftarrow 1.0 + \frac{i}{N}$
  $L[i] \leftarrow \frac{1.0}{x}$
end for

سپس در زمان اجرا، جدول تولید شده را بر روی سوئیچ بارگذاری شده و برای تقریب از آن استفاده خواهد شد. به منظور یافتن نزدیک مقدار به عدد مورد نظر، باتوجه‌به روش تولید اعداد جدول که پیش‌تر به آن اشاره شد، ابتدا یک را از مقدار هنجارسازی شده کاسته و سپس با تغییر مکان آن به سمت راست، عدد شاخص درون جدول را خواهیم یافت. مقدار مورد نیاز برای تغییر مکان، حاصل رابطه (۱۳) است.

$$A = bits_f - \lfloor \log_2(T) \rfloor \qquad (۱۳)$$

در رابطه (۱۳)، A نشان‌دهنده مقدار تغییر مکان مورد نیاز، $bits_f$، دقت بیت‌های اعشاری، و T، اندازه جدول جست‌وجو است. در ادامه، دستورات لازم برای پیاده‌سازی جدول جست‌وجو با استفاده از یک جدول تطبیق–اقدام در پی‌فور در قطعه کد ۶ آورده شده است.

۵-۳-۲ تقریب با استفاده از بسط تیلور

در تحلیل ریاضی، بسط تیلور یک تابع حول یک نقطه، یک جمع نامتناهی از جملات است که از محاسبه مشتق‌های تابع در آن نقطه محاسبه شده است. برای اکثر توابع، جمع سری نامتناهی تیلور و مقدار تابع، در نقطه مورد محاسبه برابر است؛ به همین خاطر، می‌توان با نوشتن تعداد محدودی از جملات بسط تیلور، مقدار تابع در یک نقطه خاص و نقاط اطراف آن نقطه را تقریب زد. بسط تیلور تابع $\frac{1}{n}$ حول نقطه $n = 1$ از رابطه (۱۴) قابل محاسبه است.

$$\frac{1}{n} = 1 - (x-1) + (x-1)^2 - (x-1)^3 + \cdots \qquad (۱۴)$$

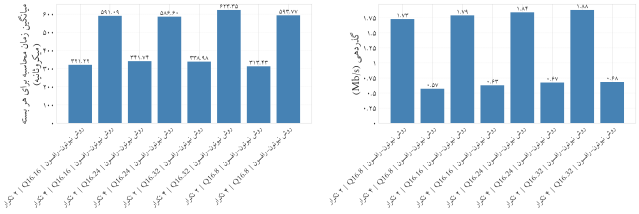
با قطع کردن جملات این بسط در یک نقطه خاص، می‌توان به تقریبی از تابع $\frac{1}{n}$ در نقاط نزدیک به عدد یک رسید. برای پیاده‌سازی ساده‌تر بسط تیلور در پی‌فور، می‌توان از روش هورنر[۳۳] [۱۶] کمک گرفت. در این روش، محاسبه مقدار یک چندجمله‌ای درجه n، از طریق محاسبه n ضرب و n جمع صورت می‌گیرد. صورت ریاضی روش هورنر در معادله (۱۵) و پیاده‌سازی این روش در پی‌فور به کمک قطعه کد ۷ قابل انجام است.

$$\begin{aligned} &a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + \cdots + x\,(a_{n-1} + x a_n)\cdots))) \\ &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \end{aligned} \qquad (15)$$

```
meta.temp_B = meta.norm_val - FIXED_ONE;
meta.temp_A = FIXED_ONE;
fixed_mul(meta.temp_A, meta.temp_B);
meta.temp_A = FIXED_ONE - meta.temp_A;
fixed_mul(meta.temp_A, meta.temp_B);
```

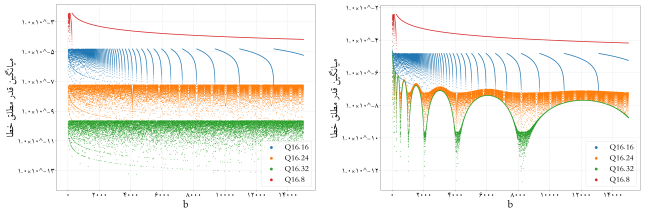
کد ۷: پیاده‌سازی بسط تیلور با درجه ۲ در پی‌فور

[۳۳] Horner's method



(آ) گذردهی (ب) میانگین زمان محاسبه

شکل ۱: مقایسه الگوریتم پیشنهادی با روش‌های پایه



(آ) دو تکرار الگوریتم نیوتن (ب) چهار تکرار الگوریتم نیوتن

شکل ۲: دقت الگوریتم پیشنهادی در پیکربندی‌های مختلف

۵-۴. تحلیل نتایج

با انجام آزمایش‌های شرح داده شده در بخش‌های پیشین، نتایج کلی ارزیابی انجام شده، از قرار زیر هستند. نتایج مقایسه الگوریتم پیشنهادی در مقابل روش‌های پایه در جدول ۲ ارائه شده است. این جدول، مقدار معیارهای سنجش عملکرد را به‌ازای هر روش مشخص کرده است. روش پیشنهادی موفق شده در مقایسه با روش‌های با دقت محاسباتی مشابه، مصرف منابع ذخیره‌سازی را به میزان حداقل 512 بایت کاهش دهد. علاوه‌براین، در مقایسه با روش‌های با مصرف حافظه مشابه، الگوریتم پیشنهادی توانسته مقدار میانگین خطا را از 5.52×10^{-2} به 1.34×10^{-4} کاهش دهد. علی‌رغم بهبودهای به‌دست‌آمده، الگوریتم پیش‌نهادی با تبدیل سربار محاسباتی به گام‌های پردازشی، در مقایسه با روش‌های پایه متحمل افزایش میانگین زمان محاسبات به میزان حداکثر 303 میکروثانیه و کاهش گذردهی به میزان حداکثر 1928 کیلوبیت بر ثانیه شده است.

نتایج نشان می‌دهند که روش پیشنهادی میزان دقت عملیات تقسیم را به اندازه قوی‌ترین روش‌های مبتنی بر جدول بهبود داده است. نکته کلیدی در الگوریتم ارائه شده، امکان تنظیم دقت مورد نظر است که این امکان را می‌دهد با کاهش تعداد دفعات تکرار روش نیوتن، سربار محاسباتی را در کنار دقت نهایی کاهش دهد. شکل‌های ۱آ و ۱ب به‌خوبی نشان‌دهنده این مصالحه هستند؛ با کاهش دقت مورد انتظار، میزان محاسبات انجام شده در سوئیچ نیز کاهش یافته و درنتیجه، تاخیر و گذردهی نیز بهبود خواهند یافت.

ازآنجاکه دقت عملیات تقسیم به روش مورد استفاده برای تقریب مقدار اولیه وابسته است، بررسی میزان دقت به‌دست‌آمده از اجرای این عملیات در لایه داده نیز بسیار حائز اهمیت است. ازاین‌رو، شکل‌های ۲آ و ۲ب به بررسی این موضوع اختصاص دارند. در این نمودارها، میانگین قدر مطلق خطا برای محاسبه مقدار $\frac{1}{b}$ در لایه داده و به ازای بازه مشخص شده، با تعداد بیت‌های متغیر و تعداد دفعات تکرار روش نیوتن مجزا برای عدد نقطه‌ثابت نهایی بررسی شده است.

نتایج ارائه شده در مجموع بیانگر چند نکته کلیدی هستند. اولاً، فرایند هنجارسازی عملیات تقسیم، بازه قابل پوشش با دقت بالا را برای روش‌های مستقل از جدول‌های جست‌وجو به طور چشم‌گیری افزایش خواهند داد؛ ثانیاً، استفاده از تقریب خطی برای حدس اولیه اعداد نقطه‌ثابت در محیط‌های دارای محدودیت از نظر محاسباتی، امکان دستیابی به تاخیر و سربار پردازشی قابل تحمل را فراهم خواهد کرد؛ در نهایت، روش نیوتن برای بهبود دقت اعداد، امکان ایجاد یک مصالحه قابل تنظیم بین میزان مصرف منابع محاسباتی و میزان خطای نهایی را به برنامه‌نویس ارائه خواهد کرد.

۶. نتیجه‌گیری

لایه داده برنامه پذیر بستری قدرتمند برای اجرای محاسبات به صورت مستقیم در داخل شبکه است؛ بااین‌حال، محدودیت‌های محاسباتی این لایه باعث محدودیت کاربردهای قابل اجرا در این لایه می‌شود. یکی از محدودیت‌های کلیدی لایه داده، عملیات تقسیم است. روش‌های موجود نیز، به علت اتکا به منابع ذخیره‌سازی، شبیه‌سازی اعداد نقطه‌شناور وابسته به سخت‌افزار، یا انجام بهینه‌سازی‌های خاص‌منظوره، به خوبی این مشکل را مرتفع نکرده‌اند. این مطالعه، یک الگوریتم تقسیم نقطه‌ثابت بدون استفاده از منابع ذخیره‌سازی در لایه داده و مستقل از معماری‌های سخت‌افزار ارائه کرده است. این راهکار، با ترکیب هنجارسازی عملیات تقسیم نقطه‌ثابت، تقریب خطی، و روش نیوتن–رافسون، امکان محاسبه تابع تقسیم را تنها با استفاده از عملیات حسابی مورد پشتیبانی توسط سخت‌افزارهای میزبان زبان برنامه‌نویسی پی‌فور فراهم کرده است. این الگوریتم، از طریق کنترل مصالحه بین دقت و سربار محاسباتی، قابلیت سازگاری با کاربردها و بارهای کاری متنوع در لایه داده را خواهد داشت. پیاده‌سازی انجام شده در زبان پی‌فور و ارزیابی مبتنی بر بی‌اِم‌وی۲، نشان‌دهنده عدم مصرف حافظه، دقت بالا، و امکان مصالحه بین سربار محاسباتی و میزان خطا هستند. کاهش سربار محاسباتی، پیاده‌سازی سایر توابع ریاضی مانند لگاریتم و ارزیابی با استفاده از سوئیچ‌های سخت‌افزاری از مسیرهای پژوهشی آتی هستند.

مراجع

[1] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Computing Surveys (CSUR)*, vol.54, no.4, pp.1–36, 2021.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol.44, no.3, pp.87–95, 2014.

[3] S. Kianpisheh and T. Taleb, "A survey on in-network computing: Programmable data plane and technology specific applica-

روش	میانگین زمان محاسبه (میکروثانیه)	گذردهی (کیلوبیت بر ثانیه)	میانگین قدر مطلق خطا	حافظه مصرفی (بایت)
بسط تیلور با درجه ۴	358.115	1770.62	5.52×10^{-2}	0
جدول جست‌وجوی ۸ بیتی	287.747	2558.14	1.34×10^{-4}	2048
جدول جست‌وجوی ۶ بیتی و ۲ تکرار روش نیوتن	311.140	2184.35	1.34×10^{-4}	512
۴ تکرار روش نیوتن (راهکار پیشنهادی)	591.093	630.55	1.34×10^{-4}	0

جدول ۲: گزارش عملکرد راهکار پیشنهادی در مقایسه با روش‌های پایه

[13] G. C. Sankaran, J. Chung, and R. Kettimuthu, "Leveraging in-network computing and programmable switches for streaming analysis of scientific data," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pp.293–297, IEEE, 2021.

[14] G. Van Rossum *et al.*, "Python programming language.," in *USENIX annual technical conference*, vol.41, pp.1–36, Santa Clara, CA, 2007.

[15] P4 Language Consortium, "p4c: P4_16 reference compiler," https://github.com/p4lang/p4c. Accessed: December 14, 2025.

[16] D. E. Knuth, "Evaluation of polynomials by computer," *Communications of the ACM*, vol.5, no.12, pp.595–599, 1962.

tions," *IEEE Communications Surveys & Tutorials*, vol.25, no.1, pp.701–761, 2022.

[4] S. Patel, R. Atsatsang, K. M. Tichauer, M. H. Wang, J. B. Kowalkowski, and N. Sultana, "In-network fractional calculations using p4 for scientific computing workloads," in *Proceedings of the 5th International Workshop on P4 in Europe*, pp.33–38, 2022.

[5] P4 Language Consortium, "The reference p4 software switch," https://github.com/p4lang/behavioral-model. Accessed: December 14, 2025.

[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol.43, no.4, pp.99–110, 2013.

[7] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-network machine learning using programmable network devices: A survey," *IEEE Communications Surveys & Tutorials*, vol.26, no.2, pp.1171–1200, 2023.

[8] P. Cui, H. Pan, Z. Li, J. Wu, S. Zhang, X. Yang, H. Guan, and G. Xie, "Netfc: Enabling accurate floating-point arithmetic on programmable switches," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pp.1–11, IEEE, 2021.

[9] M. Jose, K. Lazri, J. François, and O. Festor, "Inrec: In-network real number computation," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp.358–366, IEEE, 2021.

[10] Y. Yuan, O. Alama, J. Fei, J. Nelson, D. R. Ports, A. Sapio, M. Canini, and N. S. Kim, "Unlocking the power of inline {Floating-Point} operations on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp.683–700, 2022.

[11] M. F. Sada, J. J. Graham, M. Tatineni, D. Mishin, T. A. DeFanti, and F. WÃžrthwein, "Real-time in-network machine learning on p4-programmable fpga smartnics with fixed-point arithmetic and taylor," *arXiv preprint arXiv:2507.00428*, 2025.

[12] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol.16, no.2, pp.146–160, 1976.