مقاله پژوهشی

بهبود تحمل‌پذیری تأخیر پوشه ثبات در پردازنده‌های گرافیکی به کمک بازتولید مقادیر میانی

راحیل براتی، سیدمحمد صدرالساداتی و حمید سربازی آزاد

چکیده: پوشه ثبات بزرگ در پردازنده‌های گرافیکی با بهبود موازات سطح نخ، باعث کاهش دسترسی به حافظه می‌شود. قبلاً برای افزایش ظرفیت پوشه ثبات با سربار توان و مساحت قابل قبول، روش LTRF ارائه شده است. معماری پوشه ثبات LTRF دوسطحی است که از یک حافظه نهان ثبات و یک پوشه ثبات اصلی استفاده می‌کند. ثبات‌های کلاف‌ها قبل از اجرای یک کلاف به حافظه نهان ثبات پیش‌واکشی می‌شوند. برای پیش‌واکشی ثبات‌ها، گراف کنترل جریان برنامه در سطح مترجم به زیرگراف‌هایی به نام بازه‌ثبات تقسیم می‌شود. یکی از سربارهای روش LTRF انجام عمل پیش‌واکشی ثبات و تحمیل بیکاری کلاف در طول مدت پیش‌واکشی است که کاهش تعداد بازه‌ثبات به میزان چشم‌گیری این سربار را کاهش می‌دهد. اما تعداد ثبات قابل استفاده در هر بازه‌ثبات محدود است و افزایش این تعداد در بازه‌ثبات منجر به افزایش ترافیک پیش‌واکشی و ظرفیت حافظه نهان می‌گردد که راه حل مناسبی برای کاهش تعداد بازه‌ثبات‌ها نیست. در این پژوهش به کمک بازتولید مقادیر میانی در زمان ترجمه سعی در کاهش تعداد ثبات‌های مورد نیاز در هر بازه‌ثبات داریم. نتایج شبیه‌سازی نشان می‌دهند که روش پیشنهادی ما، میزان تحمل‌پذیری تأخیر دسترسی به پوشه ثبات در روش LTRF را به میزان ۲۹ درصد بهبود می‌بخشد. همچنین با به کارگیری یک پوشه ثبات سلول‌های حافظه DWM، معماری پیشنهادی قادر است که کارایی پردازنده گرافیکی مجهز به LTRF را به طور میانگین ۱۸ درصد (حدود ۳۰ درصد نسبت به معماری پردازنده گرافیکی پایه) افزایش دهد و این در حالی است که مقادیر انرژی و توان مصرفی به میزان ۳۸ و ۱۵ درصد کاهش می‌یابد.

کلیدواژه: **پردازنده‌های گرافیکی، پوشه ثبات، بازتولید مقادیر، واحدهای اجرایی.**

۱- مقدمه

طی سال‌های اخیر، با محدودشدن میزان موازات در سطح دستور[1]، استفاده از چند هسته روی تراشه به عنوان راهکاری برای افزایش کارایی پردازنده‌ها مورد توجه قرار گرفته است. به همین خاطر، رویکرد طراحان سیستم‌های کامپیوتری به پردازش تعداد زیادی نخ به صورت موازی یا هم‌روند معطوف شده است. در همین راستا از معماری پردازنده‌های گرافیکی[2] که به طور سنتی برای پردازش تعداد زیادی نقطه[3] در محاسبات گرافیکی طراحی شده بودند، به عنوان گزینه مناسبی برای محاسبات عام‌منظوره نام برده می‌شود.

ویژگی معماری پردازنده‌های گرافیکی این است که با آماده نگه داشتن تعداد بسیار زیادی نخ برای اجرا، سعی در پنهان‌سازی تأخیر دسترسی به حافظه دارند [۱] تا [۴]. بنابراین اگر برنامه به گونه‌ای باشد که به اندازه کافی نخ فعال در حین اجرا داشته باشد، هنگام رخداد دسترسی به حافظه، فرایند اجرا به یک نخ دیگر منتقل می‌شود تا تأخیر دسترسی به حافظه به کمک زمان اجرای سایر نخ‌ها پنهان شود. اما در تحقیقات گذشته نشان داده شده که تعداد نخ‌های آماده اجرا برای بارهای کاری عام‌منظوره در پردازنده گرافیکی به اندازه‌ای نیست که بتواند تأخیر دسترسی به حافظه را به صورت مناسب پنهان کند. در این راستا، عوامل متعددی از قبیل تعداد مجاز بلوک نخ، ظرفیت حافظه مشترک و ظرفیت پوشه ثبات منجر به کاهش تعداد نخ‌های آماده اجرا می‌شوند [۱] و [۵] تا [۸].

مهم‌ترین عامل محدودکننده تعداد نخ‌های آماده اجرا، ظرفیت محدود پوشه ثبات است [۹] تا [۱۱]. از آنجا که هر نخ باید ثبات‌های اختصاصی داشته باشد تا فرایند تعویض محتوا به سرعت انجام گیرد، پردازنده‌های گرافیکی به تعداد ثبات‌های بسیار زیادی نیاز دارند [۹] تا [۱۲]. متأسفانه به دلیل مساحت زیاد و توان مصرفی چشم‌گیر پوشه ثبات که از سلول‌های SRAM ساخته شده است، ظرفیت کنونی آن بسیار کمتر از ظرفیت مورد انتظار برای جلوگیری از محدودشدن آن برای تعداد نخ‌های آماده اجرا است.

افزایش حجم حافظه‌های روی تراشه از جمله افزایش ظرفیت پوشه ثبات با هدف بهبود کارایی، بدون در نظر گرفتن محدودیت‌های مساحت و توان مصرفی امکان‌پذیر نیست [۱۳]. پژوهش‌های زیادی به ارائه راهکارهایی با رویکرد کاهش توان مصرفی پوشه ثبات پرداخته‌اند [۱۳] تا [۱۶]. از جمله راهکارهای پیشنهادشده می‌توان به اضافه‌کردن یک حافظه نهان به ساختار پوشه ثبات و یکپارچه‌سازی حافظه‌های روی تراشه و اختصاص آن به هر بخش به شکل ایستا یا پویا اشاره کرد [۱۳]. در سال ۲۰۱۹ روشی به نام CORF ارائه شد که با کمک تکنیک‌های زمان ترجمه به ادغام چند دسترسی به پوشه ثبات می‌پردازد. این روش با کاهش دفعات دسترسی فیزیکی به پوشه ثبات، انرژی مصرفی را کاهش می‌دهد و با ادغام چندین خواندن، فشار روی پوشه ثبات و احتمال تداخل بین بانک‌ها در پوشه ثبات را کم می‌کند [۱۶]. پژوهش دیگری با نام

این مقاله در تاریخ ۱۲ اسفند ماه ۱۳۹۹ دریافت و در تاریخ ۲۱ آذر ماه ۱۴۰۰ بازنگری شد.

راحیل براتی (نویسنده مسئول)، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران، (email: rahil.barati@ipm.ir).

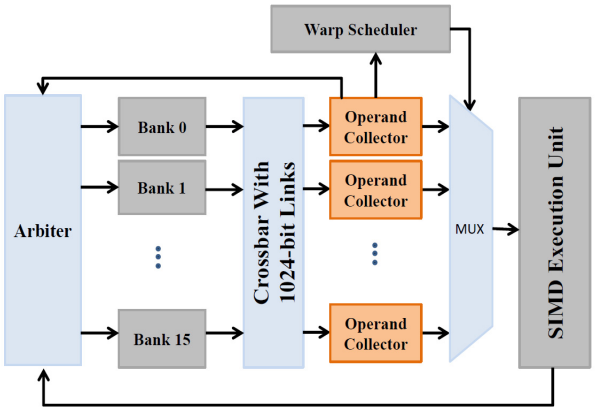
محمد صدرالساداتی، پژوهشگاه دانش‌های بنیادی، تهران، ایران، (email: sadrosadati@ipm.ir).

حمید سربازی آزاد، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف تهران، ایران، (email: azad@sharif.edu).

1. Instruction Level Parallelism

2. Graphics Processing Unit

3. Pixel



شکل ۱: معماری پایه پوشه ثبات در پردازنده‌های گرافیکی [۱۹].

BOW با ایجاد تغییراتی در واحد جمع‌آوری عملوندها[1] و اضافه‌کردن بافر، از خواندن و نوشتن‌های غیر ضروری بر پوشه ثبات جلوگیری می‌کند. در روش BOW بهینه‌سازی در زمان ترجمه، در تصمیم‌گیری برای نوشتن یک عملوند در پوشه ثبات یا صرفاً نوشتن آن در بافر مؤثر خواهد بود [۱۴]. در سال‌های اخیر، توجه به تکنیک‌های زمان ترجمه برای استفاده حداکثری از ظرفیت پوشه ثبات و یا تحمل‌پذیر ساختن سربارها، مورد توجه بیشتری قرار گرفته که از این میان می‌توان به روش RegMutex اشاره کرد [۱۵]. در این روش در زمان ترجمه، پوشه ثبات به دو بخش مجموعه ثبات‌های پایه و مجموعه ثبات‌های گسترده تقسیم می‌شود. در زمان اجرا، ثبات‌های متعلق به مجموعه ثبات‌های گسترده میان کلاف‌ها به طور مشترک استفاده می‌شوند [۱۵]. البته روش‌های اشاره‌شده در بالا به موضوع سربار مساحت برای افزایش ظرفیت پوشه ثبات در کنار توان مصرفی به خوبی نپرداخته‌اند و بهبود اشاره‌شده در توان مصرفی پوشه ثبات‌های پیشنهادی در بهترین حالت حدود ۵۰ درصد گزارش شده است که می‌تواند برای طراحی یک پوشه ثبات با ظرفیت ۱/۵ تا ۲ برابری مورد استفاده قرار گیرد. با توجه به تحلیل‌های صورت‌گرفته (تأثیر افزایش ظرفیت پوشه ثبات بر کارایی در شکل ۳ دیده می‌شود که این مقایسه با معماری پایه پردازنده گرافیکی صورت گرفته است)، این میزان افزایش در حجم پوشه ثبات کافی نبوده و نیاز به پوشه ثبات بسیار بزرگ‌تر همچنان وجود دارد.

در سال ۲۰۱۷ یک پژوهش مدعی شد که با کمک تکنیک‌های زمان ترجمه می‌توان پوشه ثبات بسیار کوچک‌تری را جایگزین پوشه ثبات فعلی نمود و پوشه ثبات اصلی را به خارج از تراشه منتقل کرد [۱۷]. از آنجایی که در روش ارائه‌شده، بخشی از پوشه ثبات به خارج از تراشه منتقل می‌شود و طراحی پوشه ثبات خارج از تراشه با محدودیت‌های توان و مساحت کمتری روبه‌رو است، امکان افزایش ظرفیت آن وجود دارد. از طرفی، تأثیرگذاری و کارایی این روش طراحی، به مشخصات بارهای کاری وابسته است. اگر بازه زندگی ثبات‌های مورد استفاده بارهای کاری طولانی باشد، نیاز به انتقال مقادیر ثبات بین بخش درون تراشه و بخش برون تراشه از پوشه ثبات بالا رفته و اثربخشی روش، کاهش می‌یابد. بنابراین اثربخشی این روش نیز عمومیت نداشته و به نوع بار کاری وابسته است.

روش LTRF[2] [۱۸] برای افزایش ظرفیت پوشه ثبات با سربار توان و مساحت قابل قبول ارائه شده است. معماری پوشه ثبات LTRF از یک حافظه نهان ثبات و یک پوشه ثبات اصلی استفاده می‌کند. ثبات‌های کلاف‌ها قبل از اجرای یک کلاف از پوشه ثبات اصلی به حافظه نهان ثبات پیش‌واکشی می‌شوند و تأخیر پیش‌واکشی ثبات به کمک زمان اجرای سایر کلاف‌ها پنهان می‌شود. برای پیش‌واکشی ثبات‌ها، گراف کنترل جریان برنامه در سطح مترجم به زیرگراف‌هایی به نام بازه‌ثبات تقسیم می‌شود.

یکی از سربارهای روش LTRF بیکاری کلاف در طول مدت پیش‌واکشی است که با کاهش تعداد بازه‌ثبات‌ها می‌توان این سربار را کاهش داد. هنگام تقسیم گراف کنترل جریان برنامه به بازه‌ثبات‌ها، ۲ محدودیت اصلی در نظر گرفته می‌شود: ۱) این که هر بازه‌ثبات، زیرگرافی است که تنها یک ورودی دارد و ۲) این که بیشینه تعداد ثبات در هر بازه‌ثبات محدود است. بررسی‌ها نشان می‌دهند که عامل محدودکننده اصلی برای بزرگ‌ترشدن بازه‌ثبات و کمترشدن تعداد بازه‌ثبات‌ها، اغلب عامل دوم است. محدودیت ترافیک پیش‌واکشی و حجم حافظه نهان، اجازه افزایش تعداد ثبات‌های مجاز در بازه‌ثبات‌ها را نمی‌دهد.

این پژوهش با هدف کاهش سربارهای پیش‌واکشی و بیکاری کلاف‌ها در طول زمان پیش‌واکشی با استفاده از بازتولید مقادیر میانی ارائه شده است. در بخش اول با استفاده از یک روش زمان ترجمه، به محاسبه مجدد ثبات‌ها با توجه به مقادیر اولیه آنها می‌پردازیم. بازتولید این مقادیر با توجه به بازه زندگی هر ثبات و وجود عملوندهای مورد نیاز بازتولید صورت می‌گیرد. بازتولید مقادیر میانی در روش LTRF باعث بزرگ‌ترشدن اندازه بازه‌ثبات، بیشترشدن تعداد دستورات قابل اجرا در یک بازه‌ثبات و در نتیجه کاهش سربار پیش‌واکشی و نیز کاهش سربار بیکاری کلاف در طول مدت پیش‌واکشی می‌شود. مزیت این روش آن است که بدون ایجاد تغییر در تعداد مجاز ثبات‌ها و بدون افزایش ترافیک پیش‌واکشی و تغییر در ظرفیت حافظه نهان، می‌توان بازه‌ثبات‌های بزرگ‌تری داشت. بهره‌گیری از این روش در معماری LTRF باعث بهبود عملکرد روش LTRF و افزایش میزان تحمل‌پذیری تأخیر دسترسی به پوشه ثبات آن گردیده است. نتایج شبیه‌سازی ما نشان می‌دهد که تحمل‌پذیری تأخیر دسترسی به پوشه ثبات روش LTRF به میزان ۲۹ درصد با به کارگیری روش پیشنهادی بهبود می‌یابد.

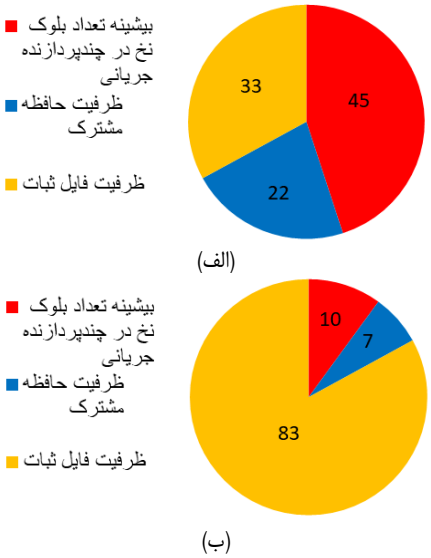
در ادامه از نتایج روش پیشنهادی (تحمل‌پذیری تأخیر پوشه ثبات) استفاده می‌کنیم تا کارایی و انرژی پردازنده گرافیکی را بهبود دهیم. آزمایش‌های ما با پیکربندی‌های گوناگون برای تعداد کلاف و تعداد بازه‌ثبات متفاوت، نشان می‌دهند که طراحی صورت‌گرفته در بخش قبل، با پیکربندی ۴ کلاف فعال و حداکثر ۱۲ ثبات در هر بازه‌ثبات، تحمل تأخیر پوشه ثبات تا ۶ برابر ظرفیت معماری پردازنده گرافیکی پایه را دارد. نتایج شبیه‌سازی با پیکربندی جدید، بهبود ۳۸ درصدی انرژی پردازنده گرافیکی و بهبود ۱۸ درصدی کارایی را به همراه دارد.

۱-۱ معماری پوشه ثبات

شکل ۱ معماری متداول پوشه ثبات در چند پردازنده جریانی پردازنده‌های گرافیکی را نشان می‌دهد [۱۹]. ظرفیت پوشه ثبات در پردازنده‌های گرافیکی در حدود چندین مگابایت است که برای اجرای هم‌زمان تعداد فراوان نخ‌ها ضروری است. به عنوان مثال، ظرفیت پوشه ثبات در پردازنده گرافیکی مدل GP ۱۰۰ در حدود ۱۴ مگابایت است. هر پوشه ثبات متشکل از تعدادی بانک می‌باشد و به منظور فراهم‌کردن پهنای باند مناسب، تعداد بانک بیشتری در پوشه ثبات پردازنده‌های

1. Operand Collector

۲. منظور از LTRF در این مقاله، نسخه نهایی ارائه‌شده در [۱۸] است که در آنجا به آن به عنوان LTRF+ اشاره شده است.



شکل ۲: مقایسه تأثیر عوامل گوناگون در محدودسازی موازات سطح نخ، (الف) نسل فرمی و (ب) نسل پاسکال.

گرافیکی در نظر گرفته می‌شود تا امکان دسترسی موازی برای تعداد بالای نخ‌ها را ایجاد کند. هنگامی که پردازنده گرافیکی یک دستور برای اجرا صادر می‌کند، واحد جمع‌آوری عملوندها، ثبات‌های مورد نیاز این دستور را طی چندین سیکل از پوشه ثبات می‌خواند. دسترسی واحدهای جمع‌آوری عملوند به بانک‌های ثبات از طریق یک شبکه میان‌ارتباطی تقاطع[1] صورت می‌گیرد که مدیریت دسترسی به این شبکه میان‌ارتباطی بر عهده یک داور مرکزی است.

۲-۱ موازات سطح نخ

عوامل متعددی وجود دارد که تعداد نخ‌های قابل تخصیص را به هر چند پردازنده جریانی، محدود می‌کنند که شامل موارد زیر هستند:

۱) بیشینه تعداد بلوک نخ در چند پردازنده جریانی
۲) ظرفیت حافظه مشترک
۳) ظرفیت پوشه ثبات

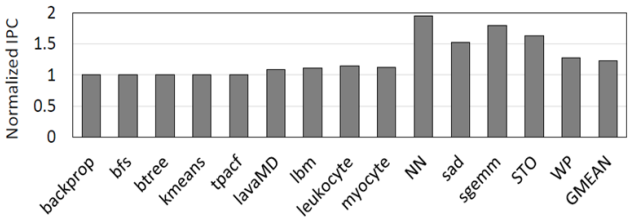
تعداد بیشینه ثباتی که مترجم پردازنده گرافیکی در کد اسمبلی به کار می‌گیرد در نسل‌های متوالی افزایش چشم‌گیری داشته است، به طوری که از ۶۴ ثبات به ازای هر نخ در نسل فرمی به عدد ۲۵۶ ثبات به ازای هر نخ در نسل ولتا رسیده‌ایم.

شکل ۲ تأثیر عوامل گوناگون را در محدودکردن موازات سطح نخ در دو نسل پردازنده گرافیکی، فرمی (سال ۲۰۰۹) و پاسکال (سال ۲۰۱۶) نشان می‌دهد. همان طور که مشاهده می‌شود در نسل‌های کنونی، ظرفیت پوشه ثبات سهم ۸۳ درصدی در محدودکردن موازات سطح نخ دارد و در نتیجه، مهم‌ترین عامل محدودکردن موازات سطح نخ ظرفیت پوشه ثبات است. معماری LTRF برای غلبه بر این مشکل طراحی شده که این معماری، امکان ساخت پوشه ثبات‌های بزرگ و با توان مصرفی و مساحت کم را ایجاد می‌کند، اما معایبی دارد که در بخش بعد به آن می‌پردازیم.

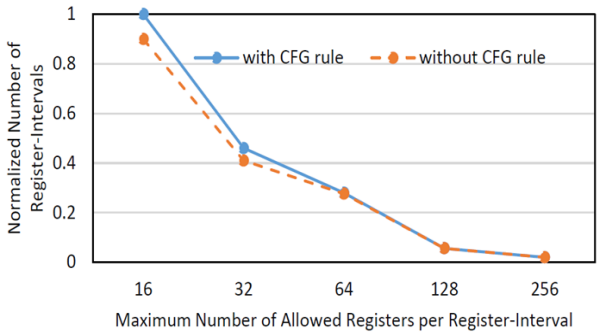
۲- انگیزه

پوشه ثبات بزرگ می‌تواند در بهبود کارایی پردازنده‌های گرافیکی مؤثر باشد. شکل ۳ کارایی پردازنده گرافیکی را بعد از افزایش ظرفیت پوشه

1. Crossbar



شکل ۳: تأثیر افزایش ظرفیت پوشه ثبات روی کارایی پردازنده‌های گرافیکی در حالت ایده‌آل نسبت به معماری پایه پردازنده گرافیکی.



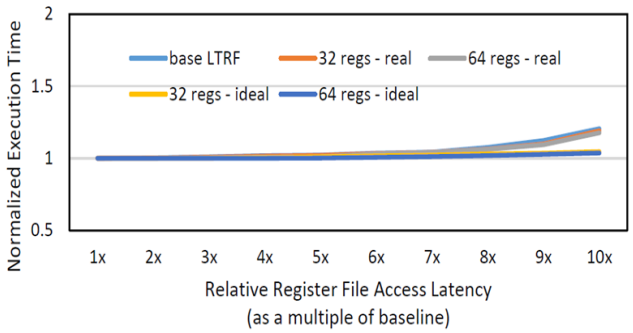
شکل ۴: تأثیر افزایش تعداد مجاز ثبات‌ها در بازه‌ثبات بر میانگین تعداد بازه‌ثبات‌ها.

ثبات از ۲۵۶ کیلوبایت به ۱٫۵ مگابایت نشان می‌دهد. در واقعیت افزایش ظرفیت پوشه ثبات منجر به افزایش تأخیر دسترسی به پوشه ثبات می‌شود که می‌تواند کارایی را کاهش دهد. اما در این آزمایش در حالت ایده‌آل فرض شده که تأخیر دسترسی افزایش نیافته است. روش LTRF به عنوان یک پوشه ثبات سلسله‌مراتبی با قابلیت تحمل تأخیر دسترسی ارائه گردیده است.

سازوکار کلیدی در روش LTRF پیش‌واکشی نزدیک به ایده‌آل ثبات‌ها است. در این سازوکار، گراف کنترل جریان برنامه به تعدادی بازه‌ثبات تقسیم می‌گردد و مجموعه ثبات‌های هر بازه‌ثبات در آغاز آن به حافظه نهان پوشه ثبات آورده می‌شوند. در نتیجه، کلاف هنگام اجرای یک بازه‌ثبات، با تأخیر کم دسترسی به حافظه نهان پوشه ثبات به جای تأخیر زیاد دسترسی به پوشه ثبات اصلی مواجه می‌شود. معماری LTRF مسیر را برای ارائه انواع روش‌های بهینه‌سازی توان و مساحت در پوشه ثبات پردازنده‌های گرافیکی هموار می‌کند و با قابل تحمل ساختن اثر منفی پوشه ثبات کندتر بر کارایی، فضای طراحی بزرگ‌تری در اختیار معماران پردازنده‌های گرافیکی قرار می‌دهد تا مصالحه بین توان، مساحت و تأخیر را با آزادی بیشتری انجام دهند.

هر بازه‌ثبات یک زیرگراف از گراف کنترل جریان برنامه است که دو ویژگی مهم دارد: ۱) تنها یک نقطه ورودی کنترل جریان دارد و ۲) تعداد ثبات‌های داخل آن از بیشینه تعداد ثبات مشخص‌شده برای هر کلاف در حافظه نهان پوشه ثبات کمتر یا مساوی است. در این بخش ابتدا بررسی می‌شود که کدام یک از ویژگی‌های بازه‌ثبات، در محدودکردن بازه‌ثبات تأثیر بیشتری می‌گذارد. برای این منظور، تعداد مجاز ثبات‌ها در یک بازه‌ثبات از ۱۶ تا ۲۵۶ افزایش داده شده و میانگین تعداد بازه‌ثبات‌ها برای هر حالت بین بارهای کاری گوناگون محاسبه گردیده است.

در شکل ۴، تأثیر افزایش تعداد مجاز ثبات‌ها در یک بازه‌ثبات بر تعداد بازه‌ثبات‌ها، در دو حالت بدون قانونِ CFG (خط‌چین نارنجی) و با قانونِ CFG (ممتد آبی) مقایسه شده‌اند. در این شکل نتایج هنجارسازی شده به تعداد پایه بازه‌ثبات گزارش شده است. همان طور که مشاهده می‌شود، افزایش تعداد ثبات‌ها از ۱۶ به ۲۵۶، میانگین تعداد بازه‌ثبات را به میزان ۹۸ درصد کاهش داده است. کاهش تعداد بازه‌ثبات‌ها در این آزمایش



شکل ۵: مقایسه تأثیر افزایش تعداد مجاز ثبات در بازه‌ثبات روش LTRF بر میزان پنهان‌سازی تأخیر دسترسی به پوشه ثبات در دو حالت ایده‌آل و با سربار واقعی.

نشان‌دهنده کاهش محدودیت ساخته‌شدن بازه‌ثبات و امکان ترکیب‌شدن بیشتر بازه‌ثبات‌ها با یکدیگر است.

سپس آزمایش با حذف اثر ویژگی کنترل جریانی تکرار شده (بدون قانونِ CFG) و میانگین تعداد بازه‌ثبات‌ها برای مقادیر گوناگون ثبات‌ها محاسبه گردیده و نتایج به تعداد پایه بازه‌ثبات، هنجارسازی شده است. مشاهده می‌شود که حذف قانون شماره ۱ تأثیر چندانی در کاهش تعداد بازه‌ثبات‌ها ندارد و تعداد بازه‌ثبات‌ها با حالت قبل تقریباً یکسان است. همچنین با افزایش تعداد مجاز ثبات‌ها در بازه‌ثبات، تأثیر حذف قانون در سطح گراف کنترل جریان کاهش می‌یابد. نتیجه می‌گیریم که تعداد محدود ثبات‌ها در بازه‌ثبات، نقشی کلیدی را در تعیین تعداد بازه‌ثبات‌ها ایفا می‌کند.

کاهش تعداد بازه‌ثبات‌ها (یا افزایش اندازه بازه‌ثبات‌ها) می‌تواند مزایای زیر را به ارمغان بیاورد:

- تعداد دستورالعمل‌های ایستا و پویا را در بازه‌ثبات افزایش می‌دهد که منجر به افزایش قدرت مکانیزم LTRF در پنهان‌سازی تأخیر پیش‌واکشی ثبات‌ها می‌گردد.
- سربار انرژی و کارایی پیش‌واکشی ثبات‌ها را به دلیل کاهش تعداد بازه‌ثبات‌ها، کاهش می‌دهد.

اما کاهش تعداد بازه‌ثبات‌ها از طریق افزایش تعداد ثبات‌های مجاز در بازه‌ثبات راهکار مناسبی نیست. زیرا افزایش تعداد ثبات‌ها منجر به افزایش اندازه حافظه نهان پوشه ثبات و افزایش تأخیر پیش‌واکشی ثبات‌ها می‌گردد که در نهایت می‌تواند سود حاصل از کاهش تعداد بازه‌ثبات را از بین ببرد. برای نشان‌دادن بیشتر این موضوع آزمایشی را طرح کردیم. در این آزمایش بیشینه تعداد مجاز ثبات در بازه‌ثبات را از ۱۶ به ۶۴ افزایش دادیم و کارایی روش LTRF را در پنهان‌سازی تأخیر دسترسی به پوشه ثبات در دو حالت بررسی نمودیم. در حالت اول سربار افزایش تعداد ثبات‌ها را در نظر می‌گیریم و در حالت دوم فرض می‌شود که سربار پیش‌واکشی با افزایش تعداد ثبات تغییری نمی‌کند. شکل ۵ نتایج هنجارشده نسبت به معماری پایه پردازنده گرافیکی را گزارش می‌کند. همان طور که مشاهده می‌شود با در نظر گرفتن سربار افزایش تعداد ثبات‌ها، کارایی روش LTRF با افزایش تعداد ثبات‌های بازه‌ثبات تغییر چندانی نمی‌کند. در حالی که اگر تعداد ثبات‌ها را بدون سربار افزایش دهیم می‌تواند کارایی روش LTRF در پنهان‌سازی تأخیر دسترسی به پوشه ثبات را به میزان قابل توجهی افزایش دهد. بنابراین کاهش تعداد بازه‌ثبات‌ها باید با راهی متفاوت از افزایش تعداد مجاز ثبات در بازه‌ثبات صورت گیرد.

در این پژوهش تصمیم داریم که با کاهش تعداد ثبات‌های مصرفی در بازه‌ثبات به کمک روش بازتولید مقادیر میانی، تعداد بازه‌ثبات‌ها را بدون افزایش بیشینه تعداد مجاز ثبات، کاهش دهیم. پژوهش‌های گذشته نشان می‌دهند که واحدهای اجرایی در پردازنده‌های گرافیکی به میزان قابل توجهی بیکاری دارند که می‌توان از پهنای باند پردازشی به هدر رفته برای بازتولید مقادیر میانی بهره برد.

۳- روش پیشنهادی

در روش پیشنهادی در الگوریتم ساخت بازه‌ثبات‌ها علاوه بر شرط تعداد ثبات مجاز در هر بازه‌ثبات، شروط دیگری نیز بررسی می‌شوند.

- اول امکان محاسبه مجدد یک ثبات از طریق مقادیر پایه آن بررسی می‌گردد که این شرط با توجه به بازه زندگی هر ثبات و وجود عملوندهای مورد نیاز آن بررسی می‌شود. بازه زندگی هر ثبات از ابتدای تعریف‌شدن آن یعنی دستورالعمل نوشته‌شدن در ثبات آغاز می‌گردد و تا آخرین باری که بدون تغییر خوانده می‌شود، ادامه می‌یابد. وجود عملوندها را این گونه تعریف می‌کنیم: قطع‌نشدن بازه زندگی عملوندها از محل تعریف‌شدن ثبات مورد نظر تا آخرین باری که استفاده شده است.
- دوم شرط کمترشدن مجموع تعداد ثبات از محدودیت تعیین‌شده (N) در صورت محاسبه مجدد، مورد بررسی قرار می‌گیرد. بدیهی است که در صورت کمترنشدن تعداد ثبات از مقدار N عمل محاسبه مجدد تأثیری در اندازه بازه‌ثبات نخواهد داشت.

در صورت برقراری این شروط در زمان ترجمه، کد محاسبه مجدد ثبات در کد اصلی جایگذاری می‌شود. به این ترتیب می‌توان بازه‌ثبات بزرگ‌تری ساخت و نیز از پیش‌واکشی غیر ضروری به حافظه نهان ثبات جلوگیری کرد. عمل محاسبه مجدد یک ثبات از طریق مقادیر اولیه آن در ادامه این مقاله، بازتولید مقادیر میانی نامیده شده است.

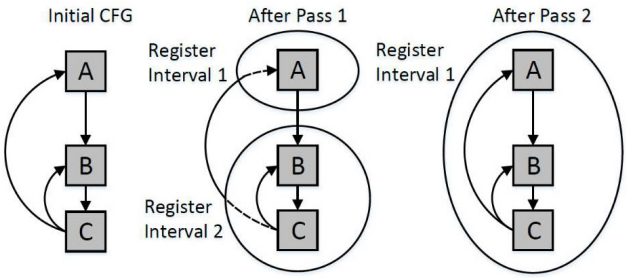
در ادامه در بخش ۳-۱ مفاهیم تئوری ساخت بازه‌ثبات در روش LTRF توضیح داده می‌شود و در بخش ۳-۲ شبه‌کد دومرحله‌ای روش پیشنهادی و توضیحات مربوط به آن آمده است.

۳-۱ نحوه ساخت بازه‌ثبات‌ها

به علت وقفه ایجادشده در ابتدای زیرگراف پیش‌واکشی، دستورات مربوط به پیش‌واکشی ثبات تأخیر زیادی دارند که این تأخیر می‌تواند منجر به کاهش کارایی پردازنده گرافیکی گردد و بنابراین استفاده از دستورالعمل‌های پیش‌واکشی باید بسیار محدود باشند. در روش LTRF گراف کنترل جریان برنامه‌های پردازنده گرافیکی به تعدادی بازه‌ثبات تقسیم می‌شود. بازه‌ثبات، زیرگرافی از گراف کنترل جریان با دو شرط است: ۱) تنها یک ورودی کنترل جریان دارد و ۲) تعداد ثبات‌های داخل آن از بیشینه تعداد ثبات مشخص‌شده برای هر کلاف در حافظه نهان پوشه ثبات، کمتر یا مساوی است. مهم‌ترین تفاوت بازه‌ثبات با ساختارهای مشابهی که در کارهای گذشته ارائه شده است این است که در بازه‌ثبات ساختارهای پیچیده کنترل جریان (مانند پرش‌های برگشتی[1]) مجاز بوده و این ساختارها منجر به اتمام بازه‌ثبات نمی‌شوند. به عبارت دیگر، بازه‌ثبات محدودیت‌های بسیار کمتری نسبت به ساختارهای گذشته دارد.

کاهش محدودیت‌ها در ساخت بازه‌ثبات ۲ سود مهم به ارمغان می‌آورد: ۱) افزایش تعداد دستورالعمل‌های ایستا در بازه‌ثبات که منجر به کاهش تعداد دستورالعمل‌های پیش‌واکشی می‌گردد و ۲) مجازبودن پرش‌های برگشت در بازه‌ثبات باعث می‌شود تا حلقه‌ها در بازه‌ثبات جای گرفته و

1. Backward Branches



شکل ۶ ساخت بازه‌ثبات برای حلقه‌های تودرتوی ساده [۱۸].

تعداد دستورالعمل‌های پویا در هر بازه‌ثبات افزایش یابد. افزایش تعداد دستورالعمل‌های پویا، زمان اجرای هر بازه‌ثبات را افزایش می‌دهد که از اثرات این موضوع، افزایش قدرت پنهان‌سازی تأخیر پیش‌واکشی ثبات‌ها است که با بهره‌گیری از آن، تأخیر دسترسی به پوشه ثبات بهتر تحمل می‌گردد.

در روش LTRF و نیز روش پیشنهادی، از الگوریتم ساخت بازه‌ها برای ساخت بازه‌ثبات‌ها استفاده شده است. الگوریتم ساخت بازه‌ها گراف کنترل جریان برنامه را به تعدادی زیرگراف جدا از هم تقسیم می‌کند به طوری که هر زیرگراف فقط دارای یک ورودی باشد. از الگوریتم تحلیل بازه به عنوان الگوریتم پایه در بیشتر الگوریتم‌های بهینه‌سازی در سطح مترجم استفاده شده است. به عنوان مثال برای تشخیص حلقه‌ها در برنامه و سنجش کاهش‌پذیری گراف کنترل جریان از الگوریتم تحلیل بازه استفاده می‌شود. تنها محدودیتی که در الگوریتم بازه‌ثبات نسبت به الگوریتم تحلیل بازه اضافه شده است، محدودبودن تعداد ثبات‌های مصرفی در هر بازه است. این محدودیت برای جایگیری ثبات‌های هر بازه در حافظه نهان پوشه ثبات اعمال شده است. بنابراین بازه‌ثبات‌ها می‌توانند از بازه‌ها در الگوریتم تحلیل بازه کوچک‌تر باشند، زیرا ممکن است به دلیل محدودیت تعداد ثبات، یک بازه به تعدادی بازه‌ثبات تقسیم گردد.

الگوریتم ساخت بازه‌ثبات‌ها به صورت یک الگوریتم چندمرحله‌ای ارائه شده است. در مرحله اول، الگوریتم می‌کوشد تا بازه‌ثبات‌ها را با ترکیب هرچه بیشتر بلوک‌های پایه در گراف کنترل جریان بسازد. بنابراین الگوریتم، اولین بازه‌ثبات را با قراردادن بلوک پایه ورودی می‌سازد و در یک روال تکراری می‌کوشد تا بلوک‌های بعدی را به بازه‌ثبات اضافه کند. یک بلوک پایه مناسب برای قرارگیری در بازه‌ثبات، باید دو ویژگی مهم داشته باشد: ۱) مسیر ورودی بلوک پایه تنها از طریق بازه‌ثبات مورد نظر باشد و ۲) تعداد ثبات در مجموعه ثبات‌های این بلوک پایه و بازه‌ثبات مورد نظر از بیشینه تعداد مجاز ثبات در بازه‌ثبات بیشتر نباشد. الگوریتم در صورتی که نتواند هیچ بلوک پایه مناسبی پیدا کند، متوقف می‌شود. پس از اتمام ساخت اولین بازه‌ثبات، الگوریتم به تمام بلوک‌های پایه که یال ورودی از این بازه‌ثبات دارند، یک بازه‌ثبات جدید اختصاص می‌دهد. اگر تعداد ثبات‌های یک بلوک پایه از بیشینه تعداد مجاز ثبات در بازه‌ثبات بیشتر باشد، آن بلوک پایه به دو یا چند بازه‌ثبات شکسته می‌شود. همچنین بلوک‌های پایه نیز با صدازدن تابع‌ها شکسته شده و هر تابع به عنوان یک بازه‌ثبات جداگانه معرفی می‌گردد. مرحله اول الگوریتم زمانی به اتمام می‌رسد که تمام بلوک‌های پایه به عنوان بازه‌ثبات معرفی شده باشند. در واقع با اتمام مرحله اول الگوریتم، گراف کنترل جریان به گراف کنترل جریان بازه‌ثبات تبدیل می‌شود که در آن گره‌ها به جای بلوک‌های پایه، بازه‌ثبات هستند.

در مرحله دوم با ترکیب بازه‌ثبات‌ها، گراف کنترل جریان بازه‌ثبات کاهش می‌یابد. این مرحله مشابه مرحله اول کار می‌کند با این تفاوت که امکان شکستن بازه‌ثبات‌ها نیست. دو بازه‌ثبات در صورت برقراری این دو شرط با هم ترکیب می‌شوند: ۱) یکی از این بازه‌ثبات‌ها تنها از طریق دیگری قابل دسترسی باشد (یال ورودی آن تنها از دیگری باشد) و ۲) مجموعه ثبات‌های این دو بازه‌ثبات در فضای حافظه نهان پوشه ثبات جا گیرد. مرحله دوم آن قدر تکرار می‌شود تا گراف کنترل جریان بازه‌ثبات دیگر کاهش‌پذیر نباشد (هیچ کدام از بازه‌ثبات‌ها قابل ترکیب‌شدن نباشند).

همان طور که در الگوریتم ساخت بازه‌ثبات توضیح داده شده است، یک بلوک پایه تنها زمانی می‌تواند با یک بازه‌ثبات دیگر ترکیب شود که یال ورودی این بلوک پایه فقط از این بازه‌ثبات باشد. در نتیجه، یال‌های برگشتی (پرش‌های برگشتی) و حلقه‌ها همواره بازه‌ثبات جدید ایجاد می‌کنند. این ویژگی کلیدی در الگوریتم ساخت بازه‌ثبات‌ها، از آنها زیرگراف‌های مناسبی برای پیش‌واکشی معرفی می‌کند. در هر حلقه، به احتمال زیاد یک بازه‌ثبات برای کل حلقه (شامل‌شدن حلقه‌های داخلی) در نظر گرفته می‌شود تا برای کل حلقه تنها یک دستور پیش‌واکشی وجود داشته باشد.

شکل ۶ حلقه‌های تودرتو را نشان می‌دهد. فرض می‌کنیم که تمام ثبات‌های این گراف در حافظه نهان پوشه ثبات جای گیرند. در مرحله اول الگوریتم، بلوک پایه "A" به عنوان بازه‌ثبات اول معرفی می‌گردد. بلوک پایه "B" نمی‌تواند به بازه‌ثبات اول ملحق شود زیرا یکی از ورودی‌های آن از بلوک پایه "C" است. بنابراین بلوک پایه "B" به عنوان بازه‌ثبات دو معرفی می‌گردد. بلوک پایه "C" می‌تواند به بازه‌ثبات دو ملحق شود زیرا تنها ورودی آن از بازه‌ثبات دو است. بنابراین کل حلقه داخلی به عنوان یک بازه‌ثبات معرفی می‌شود. در این مرحله، کار مرحله اول به انتها می‌رسد. در مرحله دوم الگوریتم ساخت بازه‌ثبات‌ها، بازه‌ثبات اول به بازه‌ثبات دوم ملحق می‌گردد زیرا تنها ورودی آن از بازه‌ثبات دو است. بنابراین کل حلقه‌های تودرتو در انتها به عنوان یک بازه‌ثبات معرفی می‌شود. همان طور که قبلاً نیز اشاره شد، این ویژگی منجر به افزایش تعداد دستورالعمل‌های پویا در هر بازه‌ثبات می‌گردد.

پیاده‌سازی مرحله اول و دوم الگوریتم ساخت بازه‌ثبات‌ها به زبان CPP در آدرس [۲۰] قرار دارد. زمان اجرای الگوریتم ساخت بازه‌ثبات برای بارهای کاری گوناگون، بسته به اندازه و پیچیدگی گراف کنترل جریان متفاوت است. در بین ۳۵ بار کاری آزمون‌شده، زمان اجرای پیاده‌سازی CPP الگوریتم ساخت بازه‌ثبات [۱۹] به طور میانگین حدود ۲ دقیقه و به طور بیشینه حدود ۷ دقیقه است. البته این هزینه برای هر بار کاری تنها یک بار پرداخت می‌شود.

۳-۲ نحوه ساخت بازه‌ثبات آگاه از بازتولید مقدار میانی

در این بخش، تغییرات الگوریتم ساخت بازه‌ثبات با اعمال روش بازتولید مقادیر بررسی می‌شود. توضیحات در این الگوریتم به رنگ آبی و فراخوانی‌های مرتبط با روش بازتولید به رنگ قرمز مشخص شده‌اند.

در شروع، الگوریتم ساخت بازه‌ثبات آگاه از بازتولید نیز مانند الگوریتم روش پایه، اولین بازه‌ثبات را با قراردادن بلوک پایه ورودی می‌سازد (خط ۸ در مرحله اول الگوریتم) و در ادامه تلاش می‌کند تا بلوک‌های بعدی را به بازه‌ثبات اضافه کند. یک بلوک پایه باید دو ویژگی مهم داشته باشد: ۱) مسیر ورودی بلوک پایه تنها از طریق بازه‌ثبات مورد نظر باشد و ۲) تعداد ثبات در مجموعه ثبات‌های این بلوک پایه و بازه‌ثبات مورد نظر از بیشینه تعداد مجاز ثبات در بازه‌ثبات بیشتر نباشد. بنابراین هنگام افزودن یک بلوک پایه به بازه‌ثبات قبلی، طبق الگوریتم ساخت بازه‌ثبات، تعداد ثبات موجود در این ترکیب جدید با عدد ثابت N مقایسه می‌گردد. در صورت

```
Algorithm One. Register-Interval Formation: Pass One
Input: Aplication Control Graph (CFG)
Output: Register-Interval CFG
1.  Initialize
2.  for each basic block: BB do
3.      BB.input list ← empty() //List of all register cache at the
        beginning of BB
4.      BB.register-interval ← Unknown
5.  end for
6.  Working-Set ← empty()
7.  entry_blockregister-interval ← new register-interval() //Each
    CFG has an entry basic block
8.  Working-Setinsert (entry_block)
9.  While (!Working-Set empty()) do
10.     BB ← a basic block from Working-Set
11.     TRAVERSE (BB)
12.     i ← BB.register-interval
13.     While ( ∃ Basic block h for which h.register-
        interval==Unknown & all of h predecessors belong to i &
        union (output_list of all h predecessors).size().size() ≤ N) do
        //N is maximum number of registers allowed in the register
        interval
14.         h.register-interval ← i
15.         h.input_list ← union (output_list of all h predecessors)
16.         TRAVERSE (h)
17.     end while
18.     for each S ∈ insuccessors() do
19.         if (S register-interval==Unknown) then
20.             S.register-interval ← new register-interval()
21.             S.input_list ← empty()
22.             Working-Set.insert (S)
23.         end if
24.     end for
25. end while
26. procedure TRAVERSE (BB)
27.     register_list ← BB.input_list
28.     for each instruction in BB do
29.         update register_list
30.         if ((register_list.size() > N) &
            reproduction(N,BB)==false)) then
31.             cut BB and introduce a new basic block: BB1
32.             BB1.register-interval ← new register-interval()
33.             BB1.input_list ← empty()
34.             Working-Set.insert (BB1)
35.             BBoutput_list ← register_list //List of all registers in
                the register file cache at the end of BB
36.             exit
37.         end if
38.     end for
39. end procedure
```

شکل ۷: مرحله اول الگوریتم ساخت بازه‌ثبات.

کوچک‌تر یا مساوی‌بودن طبق روال سابق بازه‌ثبات‌ها ساخته می‌شوند.

شبه‌کد مرحله اول الگوریتم ساخت بازه‌ثبات‌های آگاه از بازتولید مقادیر میانی در شکل ۷ آورده شده است. در این الگوریتم، در حالت بزرگ‌تربودن تعداد ثبات از عدد N تابع بازتولید فراخوانده شده و در این تابع به دو سؤال پاسخ داده می‌شود. اول آن که آیا مقادیری وجود دارند که قابل بازتولید باشند؟ و سؤال دوم این که در صورت بازتولید این مقادیر تعداد ثبات موجود در بازه‌ثبات جدید از عدد ثابت N بزرگ‌تر است یا خیر؟ در صورت بزرگ‌تربودن نسبت به عدد N بازتولید انجام نمی‌پذیرد و ساخت بازه‌ثبات با روال سابق ادامه می‌یابد. در حالتی که بتوان با روش بازتولید به شرط کوچک‌تر و مساوی N دست یافت، بازتولید انجام گرفته و بلوک پایه با موفقیت به بازه‌ثبات اضافه می‌گردد.

بازتولید مقادیر میانی با توجه به بازه زندگی هر ثبات و وجود عملوندهای مورد نیاز آن صورت می‌گیرد. بازه زندگی هر ثبات از ابتدای تعریف‌شدن آن یعنی دستورالعمل نوشته‌شدن در ثبات آغاز می‌گردد و تا آخرین باری که بدون تغییر خوانده می‌شود ادامه می‌یابد. وجود عملوندها

```
Algorithm Two. Register-Interval Formation: Pass Two
Input: Aplication Control Graph (CFG)
Output: Reduced Register-Interval CFG
1.  Initialize
2.  for each register-interval: i do
3.      i.register-interval ← Unknown
4.  end for
5.  Working-Set ← empty()
6.  entry_register-interval.next_level_register-interval ← new
    next_level_register-interval()
7.  Working-Set.insert (entry_register-interval)
8.  While (!Working-Set empty()) do
9.      i ← a register-interval from Working-Set
10.     ii ← I,next_level_register-interval
11.     ii.register_list ← i.register_list
12.     While ( ∃ register-interval h for which h.next_level_register-
        interval==Unknown & all of h predecessors belong to i &
        union (register_list of all h predecessors).size() ≤ N) do //N is
        maximum number of registers allowed in the register interval
13.         if ((union (ii.register_list & h.register_list) ≤ N) ||
            (reproduction(N,h)==true)) then
14.             h.next_level_register-interval ← ii
15.             ii.register_list ← union (ii.register_list & h.register_list)
16.         end if
17.     end while
18.     for each S ∈ insuccessors() do
19.         if (S.next_register-interval==Unknown) then
20.             S.next_level_register-interval ← new
                next_level_register-interval()
21.             Working-Set.insert (S)
22.         end if
23.     end for
24. end while
```

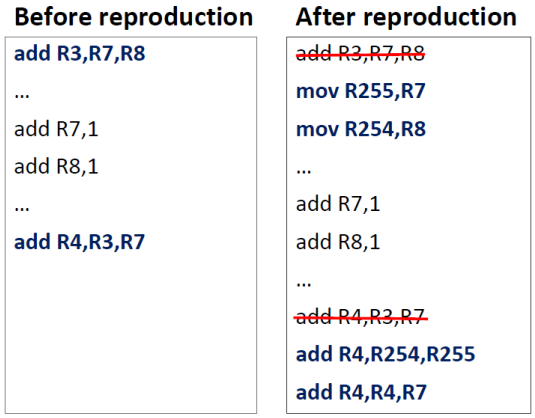
شکل ۸: مرحله دوم الگوریتم ساخت بازه‌ثبات.

را این گونه تعریف می‌کنیم: قطع‌نشدن بازه زندگی عملوندها از محل تعریف‌شدن ثبات مورد نظر تا آخرین باری که استفاده شده است. بازه زندگی هر ثبات نسبت به بازه‌ثبات جاری در یکی از این دو حالت زیر قرار می‌گیرد:

- در حالت نخست، شروع بازه زندگی ثبات و پایان آن در بازه‌ثبات جاری قرار دارد. در این حالت بازتولید این ثبات در صورت وجود عملوندهای آن بدون مشکل قابل انجام است.
- در حالت دوم، ثبات در همین بازه‌ثبات جاری تعریف شده است اما پایان بازه زندگی آن در بازه‌ثبات جاری قرار ندارد. در صورت وجود عملوندهای مورد نیاز در این حالت، تلاش می‌کنیم بازتولید را برای این بازه‌ثبات و بازه‌ثبات‌های بعدی که در آنها از این مقدار استفاده شده است، انجام دهیم. اگر در حداقل یکی از بازه‌ثبات‌های بعدی امکان بازتولید مقدار وجود نداشته باشد، بازتولید مقدار در کل بازه‌ثبات‌ها صورت نخواهد گرفت.

شبه‌کد مرحله دوم از الگوریتم ساخت بازه‌ثبات با بازتولید مقادیر میانی در شکل ۸ آمده است.

از مزایای روش بازتولید مقادیر میانی، کاهش ثبات‌هایی است که پیش‌واکشی می‌شوند. در صورتی که یک ثبات در بازه‌ثبات جاری تعریف شده باشد، برای آن عمل پیش‌واکشی صورت نمی‌گیرد اما همچنان حذف آن از طریق بازتولید مقادیر میانی به بهبود روش LTRF کمک می‌کند. در روش LTRF در صورتی که یک ثبات در بازه‌ثباتی تعریف شده باشد، در شمارش ثبات‌های مورد نیاز شمرده می‌شود. به این معنی که با وجودی که نیاز به پیش‌واکشی در این حالت وجود ندارد اما در بررسی شرط محدودیت ثبات‌ها تأثیرگذار است و بنابراین روش بازتولید از مزیت بزرگ‌ترشدن اندازه بازه‌ثبات و بیشترشدن تعداد دستورات قابل اجرا در یک



شکل ۹: نحوه بازتولید مقادیر میانی در صورت تغییر مقادیر اولیه در روش پیشنهادی.

بازه‌ثبات و در نتیجه کاهش احتمال وقفه، بهره می‌برد.

مقادیر مورد نیاز برای بازتولید ممکن است در دسترس نباشند، به این معنی که در طول بازه زندگی مقدار تولیدشده، عملوندها تغییر کرده‌اند. این چالش فرصت بازتولید مقادیر میانی را کاهش می‌دهد. آزمایش‌های ما نشان می‌دهند که این چالش به میزان حدود ۶۰ درصد فرصت بازتولید را از بین می‌برد، به این معنی که در ۶۰ درصد حالت‌ها حداقل یکی از عملوندها تغییر کرده است. برای افزایش فرصت بازتولید، برخی از عملوندها را در حافظه‌ای سریع نگهداری می‌کنیم. به این منظور، حافظه‌ای کوچک در کنار حافظه نهان پوشه ثبات اضافه کرده و نام آن را پوشه ثبات عملوند می‌گذاریم. پوشه ثبات عملوند برای هر کلاف فعال در روش LTRF+ چهار ثبات ۱۰۲۴ بیتی در نظر می‌گیرد. این ثبات‌ها به ۴ ثبات انتهایی معماری مجموعه دستورالعمل‌ها تخصیص می‌یابد. نتایج تحلیل‌های ما روی ۳۵ بار کاری نشان می‌دهد که کاهش ۴ ثبات از معماری مجموعه دستورالعمل‌ها تأثیری در کاهش کارایی ندارد چون حداکثر ۱۸۰ ثبات از ۲۵۶ ثبات معماری در بارهای کاری استفاده شده است. روش پیشنهادی، عملوندهای مورد نیاز را قبل از تغییر به این ثبات‌ها منتقل می‌کند و موقع بازتولید از آن ثبات‌ها می‌خواند. اگر در زمانی که عملوندی در این چهار ثبات قرار دارد، کلاف دچار وقفه طولانی شود به ناچار باید این ثبات‌ها را بازنویسی کرد. برای جلوگیری از این رخداد، این روش تنها برای زمانی استفاده می‌شود که دستورهایی که می‌توانند باعث ایجاد وقفه در اجرای کلاف شوند (مانند دستورهای حافظه‌ای و پیش‌واکشی ثبات‌ها) در بازه مقداردهی این ثبات‌ها و استفاده از آنها نباشد. لذا در زمان رخداد وقفه طولانی این ثبات‌ها بازنویسی نمی‌شوند. با این روش فرصت بازتولید مقادیر میانی از ۴۰ درصد به ۷۵ درصد می‌رسد. لازم به توضیح است که موقع شمارش ثبات‌ها برای پیش‌واکشی ثبات، ثبات‌های شماره ۲۵۲ تا ۲۵۵ محاسبه نمی‌شوند. شکل ۹ با مثالی ساده نحوه بازتولید مقادیر میانی را در صورت تغییر مقادیر اولیه نشان می‌دهد.

شکل ۱۰ به طور خلاصه الگوریتم بازتولید ارائه‌شده را نشان می‌دهد. این الگوریتم بیشینه تعداد مجاز ثبات در بازه‌ثبات و بلوک پایه یا بازه‌ثبات فعلی را به عنوان ورودی می‌گردد و تلاش می‌کند با بازتولید مقادیر میانی، تعداد ثبات‌های آن را کاهش دهد و در نهایت موفقیت و عدم موفقیت را اعلام کند. الگوریتم پیشنهادی، زمان ترجمه را به طور میانگین ۱۰ دقیقه و به طور بیشینه به میزان ۲۵ دقیقه افزایش می‌دهد. البته این هزینه برای هر بار کاری تنها یک بار پرداخت می‌شود. این مقادیر برای روش LTRF به طور میانگین ۲ دقیقه و به طور بیشینه ۷ دقیقه است.

۴- متدولوژی ارزیابی

برای ارزیابی روش پیشنهادی از شبیه‌ساز GPGPU-Sim [۲۱] استفاده شده که شبیه‌سازی با یک تناوب دقت است و با تمرکز بر بارهای کاری عام‌منظوره طراحی گردیده است. نحوه ارزیابی به این صورت بوده که ۳۵ بار کاری متعلق به مجموعه برنامه‌های محک CUDA SDK، Rodinia [۲۲] و Parboil [۲۳] را به دو دسته حساس به ثبات و غیر حساس به ثبات دسته‌بندی کردیم. در بارهای کاری حساس به ثبات، پوشه ثبات عامل محدودکننده موازات سطح نخ است. سپس به صورت تصادفی ۹ بار کاری از دسته حساس به ثبات و ۵ بار کاری از دسته غیر حساس به ثبات انتخاب کرده‌ایم. در ادامه کارایی روش پیشنهادی را در مقایسه با روش LTRF، پوشه ثبات دوسطحی و معماری پایه پردازنده‌های گرافیکی می‌سنجیم.

۵- سربار روش پیشنهادی

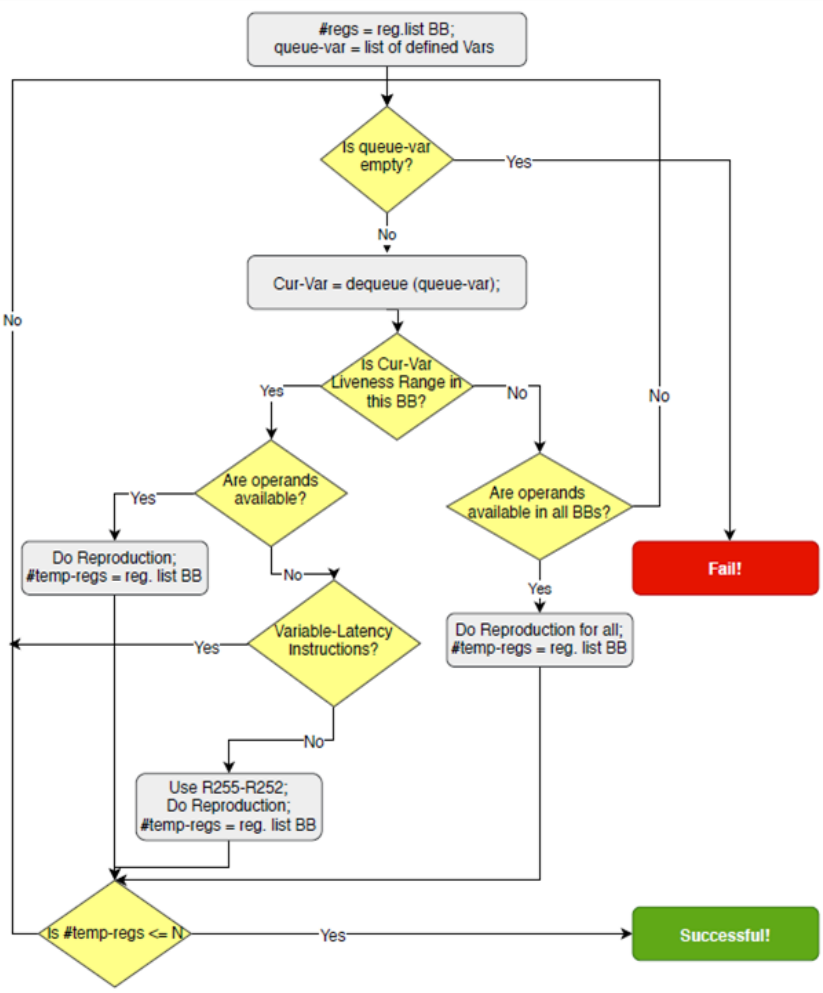
از آنجایی که روش پیشنهادی این پژوهش، تعداد دستورالعمل‌های اجراشده را افزایش می‌دهد، در این قسمت سربار کارایی و انرژی روش پیشنهادی را می‌سنجیم. سربار کارایی در روش پیشنهادی به دو دلیل محتمل است: ۱) به دلیل افزایش تعداد دستورالعمل‌ها، نرخ فقدان حافظه نهان دستورالعمل افزایش می‌یابد و ۲) افزایش تعداد دستورالعمل‌ها می‌تواند منجر به افزایش زمان اجرای بارهای کاری گردد. معیار سنجش کارایی، زمان اجرای برنامه‌های کاربردی است، زیرا معیار IPC به دلیل تغییر تعداد دستورالعمل‌ها نمی‌تواند به درستی نشانگر میزان سربار روش پیشنهادی باشد.

شکل ۱۱ زمان اجرای بارهای گوناگون کاری برای دو طراحی LTRF با و بدون بازتولید مقادیر میانی را مقایسه می‌کند. در این تحلیل از مزایای بازتولید مقادیر میانی در افزایش تحمل‌پذیری تأخیر صرف نظر شده است. معماری پایه بدون جایگزینی پوشه ثبات بزرگ‌تر و کندتر با همان اندازه ۲۵۶ کیلوبایت در نظر گرفته شده و تأثیر سربار کارایی مستقل از تحمل‌پذیری تأخیر گزارش شده و نتایج به زمان اجرای بارهای کاری در معماری پایه پردازنده‌های گرافیکی هنجارسازی شده است. توجه شود که در این آزمایش تأخیر دسترسی به پوشه ثبات اصلی در هر دو طراحی، برابر با تأخیر پوشه ثبات در معماری پایه است. مشاهده می‌شود که روش پیشنهادی زمان اجرای بارهای کاری را برای معماری LTRF به میزان ۰/۵ درصد افزایش می‌دهد که قابل چشم‌پوشی است.

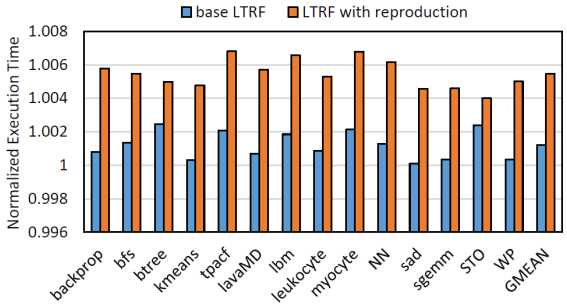
شکل ۱۲ انرژی مصرفی دو طراحی LTRF با و بدون بازتولید مقادیر میانی را مقایسه می‌کند. نتایج به انرژی مصرفی معماری پایه پردازنده گرافیکی هنجارسازی شده است. هر دو طراحی LTRF انرژی مصرفی را کاهش می‌دهد که دلیل آن کاهش توان مصرفی پوشه ثبات است. اما بهبود انرژی در روش LTRF با بازتولید مقادیر میانی (روش پیشنهادی) به میزان ۴/۳ درصد بیشتر از بهبود انرژی مصرفی در روش LTRF پایه است. این بهبود بیشتر به دلیل کاهش تعداد پیش‌واکشی ثبات و کاهش دسترسی‌ها به پوشه ثبات اصلی رخ می‌دهد.

۶- تحمل‌پذیری تأخیر دسترسی به پوشه ثبات اصلی

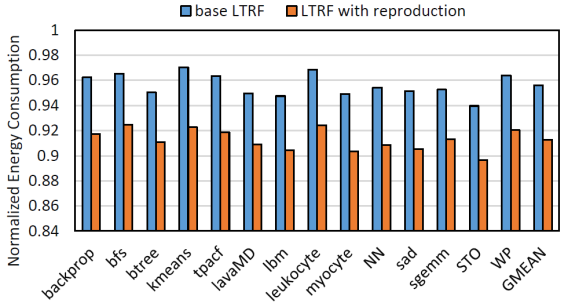
برای مقایسه قدرت تحمل‌کردن تأخیر دسترسی به پوشه ثبات اصلی در طراحی‌های گوناگون، از سنجه معرفی‌شده در مقاله [۱۸] LTRF به نام بیشینه تأخیر قابل تحمل دسترسی به پوشه ثبات اصلی استفاده می‌کنیم. این سنجه در واقع بیان‌کننده تأخیر نسبی، در مقایسه با تأخیر



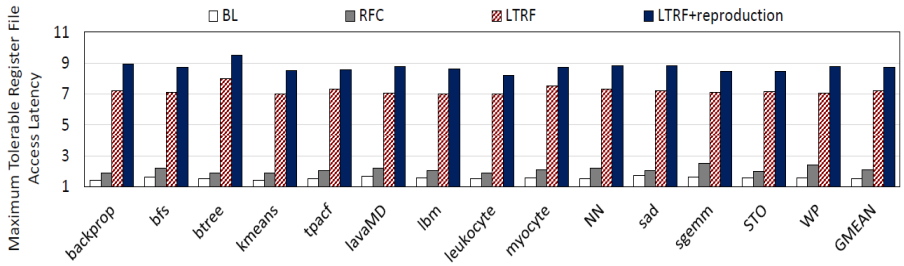
شکل ۱۰: مکانیزم پیشنهادی برای بازتولید مقادیر.



شکل ۱۱: تحلیل سربار کارایی روش پیشنهادی بدون افزایش ظرفیت پوشه ثبات.



شکل ۱۲: تحلیل انرژی مصرفی روش پیشنهادی بدون افزایش ظرفیت پوشه ثبات.

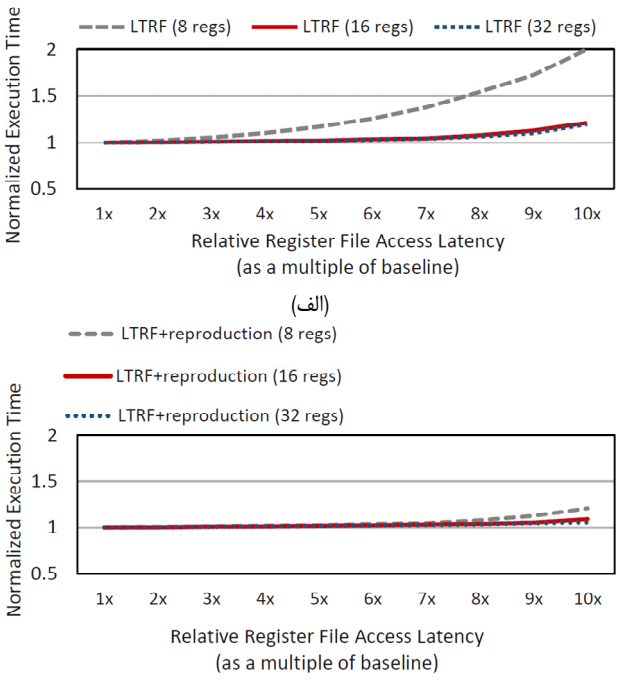


شکل ۱۳: تأخیر قابل تحمل دسترسی به پوشه ثبات با روش پیشنهادی.

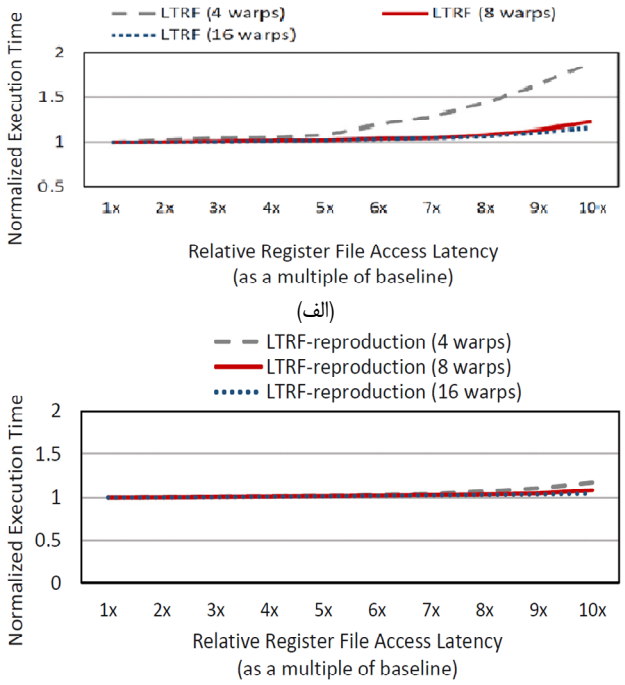
دسترسی پایه پوشه ثبات است که حداکثر ۵٪ کارایی را کاهش می‌دهد. شکل ۱۳ نتایج این سنجه را برای چهار طراحی زیر مقایسه می‌کند:

- طراحی BL: این طراحی پایه پردازنده‌های گرافیکی است که در جدول ۱ مشخصات آن گزارش شده است. تحمل‌پذیری تأخیر دسترسی به پوشه ثبات در این طراحی، تنها از طریق موازات سطح ریسمان تا حدودی ممکن می‌گردد.
- طراحی RFC[1]: این طراحی از پوشه ثبات دوسطحی بهره می‌برد که قابلیت آن را در تحمل‌کردن تأخیر دسترسی به پوشه ثبات

1. Register File Caching



شکل ۱۴: تأثیر تعداد ثبات‌های مجاز در بازه‌ثبات بر اثربخشی روش LTRF، (الف) بدون بازتولید مقادیر میانی و (ب) با بازتولید مقادیر میانی.



شکل ۱۵: تأثیر تعداد کلاف‌های فعال روی اثربخشی روش LTRF، (الف) بدون بازتولید مقادیر میانی و (ب) با بازتولید مقادیر میانی.

جدول ۱: فراسنج‌های شبیه‌سازی پردازنده گرافیکی.

تعداد چند پردازنده جریانی	۲۴
فرکانس سیکل ساعت	۱۱۳۷ مگاهرتز
زمان‌بند کلاف	دوسطحی [۲۵]
تعداد کلاف‌ها در هر چند پردازنده جریانی	۲۴
ظرفیت پوشه ثبات در هر چند پردازنده جریانی	۲۵۶ کیلوبایت (۶۵۵۳۶ ثبات)
ظرفیت حافظه نهان پوشه ثبات در هر چند پردازنده جریانی	۱۶ کیلوبایت (۴۰۹۶ ثبات)
ظرفیت حافظه مشترک در هر چند پردازنده جریانی	۶۴ کیلوبایت
ظرفیت حافظه نهان سطح اول در هر چند پردازنده جریانی	۱۶ کیلوبایت
ظرفیت حافظه نهان سطح دوم	۲ مگابایت
تعداد کلاف‌های فعال در هر چند پردازنده جریانی	۸
بیشینه مجاز تعداد ثبات در هر بازه‌ثبات	۱۶

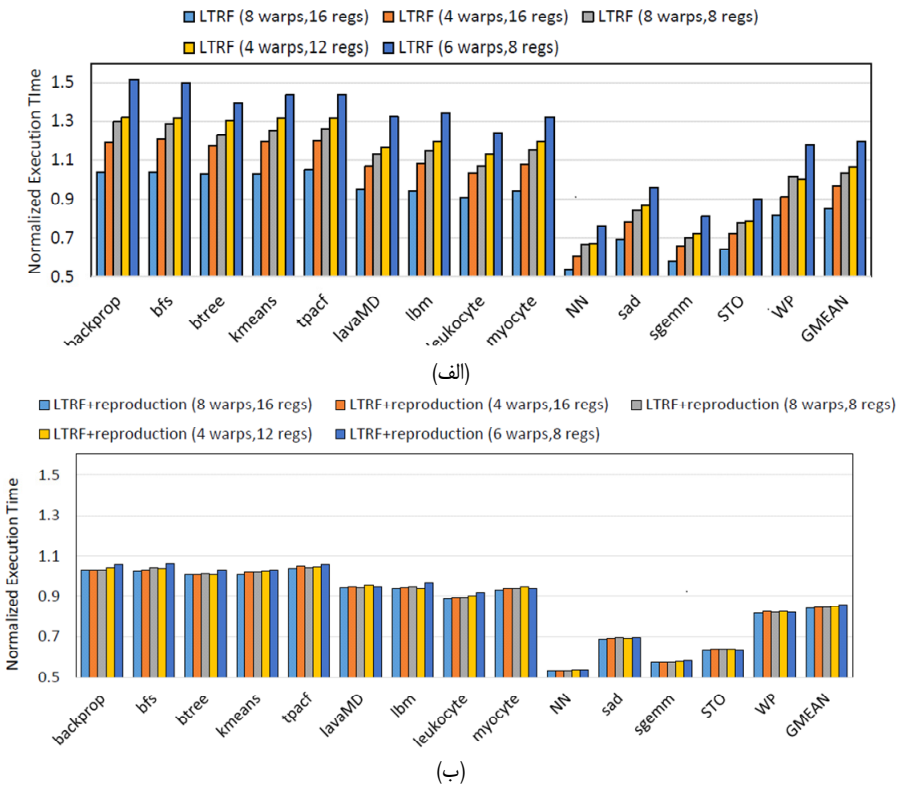
افزایش می‌دهد.

- طراحی LTRF: که معماری پیشنهادی ما برای بهبود آن ارائه شده است.
- طراحی LTRF + reproduction: در این طراحی روش LTRF روی معماری پایه‌ای پیاده‌سازی شده که از بازتولید مقادیر میانی بهره می‌برد.

با بررسی نتایج این آزمایش مشاهده می‌شود که مقدار سنجه بیشینه تأخیر قابل تحمل دسترسی به پوشه ثبات اصلی برای روش LTRF به کمک بازتولید مقادیر میانی به میزان حدود ۲۹ درصد بهبود می‌یابد. این بهبود به دلیل کاهش تعداد بازه‌ثبات‌ها رخ می‌دهد. سپس حساسیت روش پیشنهادی به ظرفیت حافظه نهان پوشه ثبات، مورد بررسی قرار گرفت. به این منظور، ارزیابی قدرت تحمل‌پذیری تأخیر برای روش‌های LTRF با و بدون استفاده از بازتولید مقادیر میانی برای تعداد متفاوت بیشینه ثبات مجاز در بازه‌ثبات و تعداد گوناگون کلاف‌های فعال تکرار شده است.

نمودارهای الف و ب در شکل ۱۴ به ترتیب تأثیر تغییر تعداد ثبات‌های مجاز در بازه‌ثبات را برای روش LTRF بدون/ با بازتولید مقادیر میانی نشان می‌دهند. نتایج به معماری پایه پردازنده گرافیکی هنجار شده‌اند. تعداد ثبات‌های مجاز در هر بازه‌ثبات، از ۸ تا ۳۲ افزایش داده شده است. مشاهده می‌شود که کارایی روش LTRF با استفاده از بازتولید مقادیر و تعداد ۸ ثبات، مشابه کارایی روش LTRF بدون استفاده از بازتولید مقادیر و تعداد ۱۶ ثبات است. به این معنی که با کمک روش LTRF و بازتولید مقادیر می‌توان ظرفیت حافظه نهان را نصف کرد و کارایی همان روش LTRF پایه را به دست آورد.

نمودارهای الف و ب در شکل ۱۵ به ترتیب تأثیر تغییر تعداد کلاف‌های فعال را برای روش LTRF بدون و با استفاده از مکانیزم بازتولید مقادیر نشان می‌دهند. نتایج به معماری پایه پردازنده گرافیکی هنجار شده‌اند و تعداد کلاف‌های فعال در این آزمایش از ۴ تا ۱۶ افزایش یافته است. همان طور که مشاهده می‌شود روش LTRF با استفاده از بازتولید مقادیر میانی و ۴ کلاف فعال، نتایج مشابهی با روش LTRF و ۸ کلاف فعال می‌دهد. در نتیجه به کمک بازتولید مقادیر می‌توان تعداد کلاف‌های فعال و به تبع آن ظرفیت حافظه نهان پوشه ثبات را نصف کرد.



شکل ۱۶: کارایی روش LTRF، (الف) بدون بازتولید مقادیر میانی و (ب) با بازتولید مقادیر میانی برای پیکربندی‌های گوناگون.

۷- افزایش ظرفیت پوشه ثبات با توجه به بهبود تحمل‌پذیری تأخیر آن

در این بخش از امکاناتی که روش پیشنهادی در اختیار طراحان پردازنده گرافیکی می‌گذارد استفاده می‌کنیم تا کارایی و انرژی پردازنده گرافیکی را بهبود دهیم. در قدم اول از حافظه DWM[1] برای ساخت پوشه ثبات اصلی استفاده می‌کنیم. سلول‌های DWM دارای مصرف توان پایین، خاصیت غیر فراری و تأخیر دسترسی قابل مقایسه با SRAM هستند [۲۴] و بنابراین استفاده از آن در سامانه‌های گوناگون، حافظه روی تراشه و ثانویه صورت گرفته است.

چگالی بالا و مصرف توان پایین سلول حافظه DWM به ما امکان افزایش ظرفیت پوشه ثبات اصلی را با مصرف توان و مساحت روی تراشه قابل قبول می‌دهد، اما در عوض، تأخیر دسترسی به پوشه ثبات اصلی را در حدود ۶ برابر افزایش می‌دهد. هر سلول حافظه DWM قابلیت ذخیره‌سازی یک ثبات ۳۲ بیتی را دارد. با بررسی ۳۵ برنامه پردازنده گرافیکی مشاهده می‌شود که ظرفیت پوشه ثبات اصلی باید حداکثر ۱۵۳۶ کیلوبایت باشد. بر این اساس به کمک سلول حافظه DWM ظرفیت پوشه ثبات اصلی را ۶ برابر افزایش می‌دهیم. حال باید تأخیر دسترسی به پوشه ثبات اصلی را به کمک روش پیشنهادی (LTRF بهبودیافته به کمک بازتولید مقادیر میانی) تحمل کنیم. در این مسیر کمترین میزان ظرفیت حافظه نهان پوشه ثبات را از طریق تنظیم فراسنج‌های روش LTRF تعداد کلاف‌های فعال و تعداد ثبات‌ها در بازه‌ثبات، به دست می‌آوریم با این قید که میزان افت کارایی برای بارهای کاری که افزایش ظرفیت پوشه ثبات، کمکی به بهبود موازات سطح نخ نمی‌کند حداکثر ۵ درصد باشد.

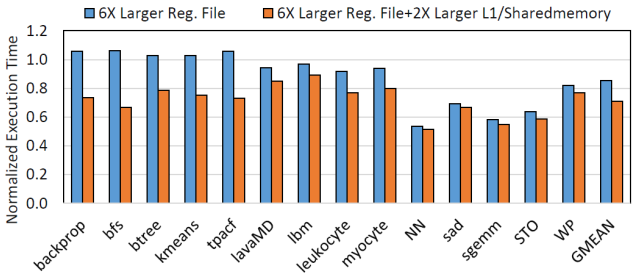
۷-۱ تحلیل کارایی

نمودارهای الف و ب در شکل ۱۶ به ترتیب کارایی پردازنده گرافیکی را با پیکربندی‌های گوناگون روش LTRF بدون استفاده و با استفاده از روش بازتولید مقادیر میانی نشان می‌دهند. در این نمودارها زمان اجرای بارهای کاری مختلف با پیکربندی‌های گوناگون در مقایسه با معماری پایه پردازنده گرافیکی گزارش شده و بنابراین مقدار کمتر، بیانگر پیکربندی با کارایی بالاتر است.

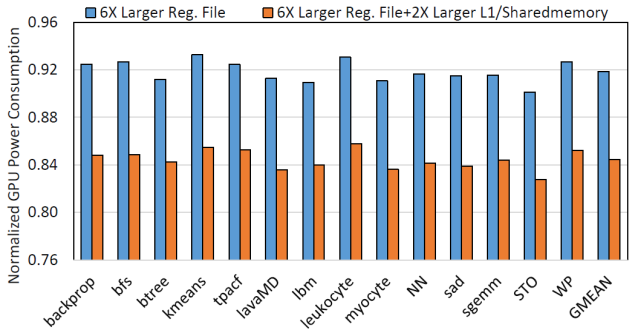
مشاهده می‌شود که اگر از روش بازتولید مقادیر میانی در کنار روش LTRF بهره ببریم، پیکربندی ۴ کلاف فعال و حداکثر ۱۲ ثبات در هر بازه‌ثبات برای تحمل‌کردن تأخیر دسترسی به پوشه ثبات اصلی ساخته‌شده با سلول حافظه DWM کافی است.

بنابراین ظرفیت حافظه نهان پوشه ثبات مورد نیاز ۶ کیلوبایت و ظرفیت پوشه ثبات عملوند ۲ کیلوبایت خواهد بود. همچنین مساحت روی تراشه پوشه ثبات ارائه‌شده (به همراه حافظه نهان پوشه ثبات، پوشه ثبات عملوند و سایر مدارهای جانبی روش LTRF) در حدود ۷۰ درصد کمتر از مساحت روی تراشه پوشه ثبات معماری پایه است. با استفاده از ابزار GPUWattch مشاهده می‌شود که فضایی در حدود ۷ میلی‌متر مربع در هر چند پردازنده جریانی آزاد می‌شود که با استفاده از آن می‌توانیم ظرفیت حافظه یکپارچه اختصاص‌یافته به حافظه نهان سطح اول و حافظه مشترک را از ۶۴ کیلوبایت به ۱۲۸ کیلوبایت برسانیم. مطابق شکل ۱۷ در نتیجه به کارگیری پوشه ثبات ۶ برابر بزرگ‌تر و با حافظه نهان سطح اول و مشترک دو برابر بزرگ‌تر، کارایی معماری پیشنهادی نسبت به معماری LTRF با پوشه ثبات ۶ برابر بزرگ‌تر تا ۸۶ درصد (به طور میانگین در حدود ۱۸ درصد) بهبود می‌یابد. این بهبود کارایی نسبت به معماری پایه حدود ۳۰ درصد می‌باشد.

1. Domain Wall Memory



شکل ۱۷: نتیجه افزایش دوبرابری حافظه نهان سطح اول و حافظه مشترک در بهبود کارایی.



شکل ۱۸: نتایج توان مصرفی پردازنده گرافیکی با به کارگیری معماری LTRF و بازتولید مقادیر.

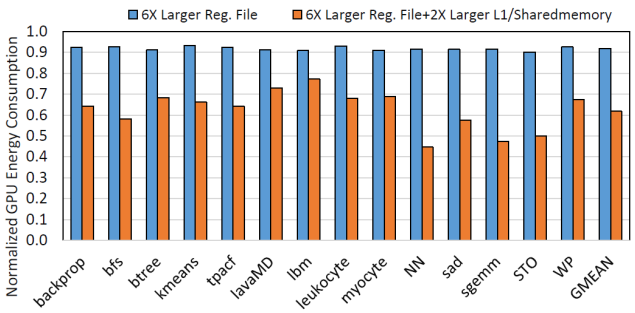
۷-۲ انرژی و توان مصرفی

توان مصرفی پردازنده گرافیکی با به کارگیری معماری پیشنهادی (LTRF و بازتولید مقادیر میانی) برای بارهای کاری گوناگون اندازه‌گیری شد. شکل ۱۸ نتایج هنجارشده مصرف توان برای بارهای کاری گوناگون را نسبت به معماری پایه پردازنده گرافیکی نشان می‌دهد. ابتدا مشاهده می‌شود که معماری پردازنده گرافیکی با پوشه ثبات ۶ برابر بزرگ‌تر به طور میانگین ۸ درصد توان مصرفی پردازنده گرافیکی را کاهش داده است. این بهبود ناشی از به کارگیری معماری LTRF و کاهش دسترسی‌ها به پوشه ثبات اصلی و سلول‌های DWM است. توجه کنید که بهبود ۸ درصدی با در نظر گرفتن تمام سربارهای روش پیشنهادی (LTRF و بازتولید مقادیر میانی و اجرای دستورالعمل‌های بیشتر) به دست آمده است. در ادامه با افزایش دوبرابری ظرفیت حافظه نهان سطح اول و حافظه مشترک، علی‌رغم این که توان مصرفی این حافظه حدود دو برابر افزایش می‌یابد، به دلیل بهبود نرخ برخورد در حافظه نهان، توان شبکه روی تراشه، حافظه نهان سطح دوم و حافظه برون تراشه کاهش می‌یابد که این مهم منجر به بهبود توان مصرفی با میانگین ۱۵ درصد می‌شود.

شکل ۱۹ نتایج هنجارشده مصرف انرژی روش پیشنهادی را نسبت به معماری پایه پردازنده گرافیکی گزارش می‌کند. همان طور که مشاهده می‌شود انرژی پردازنده گرافیکی در حدود ۳۸ درصد بهبود می‌یابد.

۸- نتیجه‌گیری

روش LTRF با تحمل‌پذیر ساختن تأخیر دسترسی به پوشه ثبات‌های بزرگ، امکان طراحی پوشه ثبات‌های بزرگ با مصرف توان کم را ایجاد می‌کند. یکی از سربارهای روش LTRF انجام عمل پیش‌واکشی ثبات‌ها و تحمیل بیکاری کلاف در طول مدت پیش‌واکشی است که کاهش تعداد بازه‌ثبات‌ها به میزان چشم‌گیری، این سربار را کاهش می‌دهد. مشاهده‌های ما نشان می‌دهند که محدودیت تعداد ثبات مورد استفاده در یک بازه‌ثبات، مهم‌ترین دلیل کاهش اندازه بازه‌ثبات و در نتیجه افزایش تعداد بازه‌ثبات‌ها



شکل ۱۹: نتایج انرژی مصرفی پردازنده گرافیکی با معماری LTRF و بازتولید مقدار میانی.

است. از طرفی افزایش تعداد ثبات قابل استفاده در بازه‌ثبات منجر به افزایش ترافیک پیش‌واکشی و ظرفیت حافظه نهان می‌گردد که راه حل مناسبی برای کاهش تعداد بازه‌ثبات‌ها نیست.

در این پژوهش به کمک بازتولید مقادیر میانی در زمان ترجمه کد، تعداد ثبات‌های مورد نیاز در هر بازه‌ثبات را کاهش دادیم. بنابراین بدون ایجاد تغییر در تعداد مجاز ثبات‌ها می‌توان بازه‌ثبات‌های بزرگ‌تری داشت. نتایج شبیه‌سازی ما نشان داد که روش ما تحمل‌پذیری تأخیر پوشه ثبات در LTRF را به میزان ۲۹ درصد افزایش می‌دهد.

در ادامه، روش پیشنهادی برای طراحی یک پوشه ثبات بزرگ‌تر با به کارگیری حافظه DWM به کار گرفته شد. نتایج شبیه‌سازی نشان داد که کارایی پردازنده گرافیکی در حالت میانگین به میزان ۱۸ درصد افزایش و انرژی و توان مصرفی به میزان ۳۸ و ۱۵ درصد کاهش یافته است.

مراجع

[1] A. Sethia and S. Mahlke, "Equalizer: dynamic tuning of gpu resources for efficient execution," in *Proc. of the IEEE/ACM 47th Annual Int. Symp. on Microarchitecture,* pp. 647-658, Cambridge, UK, 13-17 Dec. 2014.

[2] T. D. Han and T. S. Abdelrahman, "hiCUDA: high-level GPGPU programming," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 78-90, Jan. 2011.

[3] NVIDIA Corporation. *CUDA Programming Guide*, V4.0.

[4] NVIDIA Corporation. *CUDA Toolkit, 2012. Version 4.2*, http://developer.nvidia.com/cuda/cuda-downloads. Sep. 2012.

[5] J. Lee, N. Lakshminarayana, H. Kim, and R. Vuduc, "Many-thread aware prefetching mechanisms for GPGPU applications," in *Proc. IEEE/ACM of the 43th Annual Int. Symp. on Microarchitecture*, pp. 213-224, Atlanta, GA, USA, 4-8 Dec. 2010.

[6] A. Jog, O. Kayiran, A. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated scheduling and prefetching for GPGPUs," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 332-343, Jun. 2013.

[7] A. Jog, O. Kayiran, N. Chidambaram, A. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 395-406, Mar. 2013.

[8] A. Sethia, G. Dasika, M. Samadi, and S. Mahlke, "APOGEE: adaptive prefetching on GPUs for energy efficiency," in *Proc. of the IEEE 22nd Int. Conf. on Parallel Architectures and Compilation Techniques*, pp. 73-82, Edinburgh, UK, 7-11 Sept. 2013.

[9] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, "GPU register file virtualization," in *Proc. IEEE/ACM of the 48th Annual Int. Symp. on Microarchitecture*, pp. 420-432, Waikiki, HI, USA, 5-9 Dec. 2015.

[10] M. Abdel-Majeed and M. Annavaram, "Warped register file: a power efficient register file for GPGPUs," in *Proc. IEEE 19th Int. Symp. on High Performance Computer Architecture*, pp. 412-423, Shenzhen, China, 23-27, Feb. 2013.

[11] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: enabling power efficient GPUs through register compression," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 502-514, Jun. 2015.

[12] C. Hsiao, S. Chu, and C. Hsieh, "An adaptive thread scheduling mechanism with low-power register file for mobile GPUs," *IEEE Trans. on Multimedia*, vol. 16, no. 1, pp. 60-67, Sept. 2014.
[13] N. Jing, J. Wang, F. Fan, W. Yu, L. Jiang, C. Li, and X. Liang, "Cache-emulated registerfile: an integrated on-chip memory architecture for high performance gpgpus," in *Proc. IEEE/ACM of the 49th Annual Int. Symp. on Microarchitecture,* 12 pp., Taipei, Taiwan, 10-15?, Oct. 2016.
[14] H. Asghari Esfeden, A. A. Abdolrashidi, S. Rahman, D. Wong, and N. Abu-Ghazaleh, "BOW: breathing operand windows to exploit bypassing in GPUs," in *Proc. IEEE/ACM of the 53th Annual Int. Symp. on Microarchitecture*, pp. 996-1008, Athens, Greece, 17-21 Oct. 2020.
[15] F. Khorasani, H. A. Esfeden, A. Farmahini-Farahani, N. Jayasena, and V. Sarkar, "Regmutex: inter-warp gpu register time-sharing," in *Proc. of the 45th Annual Int. Symp. on Computer Architecture,* pp. 816-828, Providence, RI, USA, 13-17 Apr. 2018.
[16] H. Asghari Esfeden, F. Khorasani, H. Jeon, D. Wong, and N. Abu-Ghazaleh, "CORF: coalescing operand register file for GPUs," in *Proc. of the 24th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 701-714, Boston, MA, USA, 14-17 Oct. 2019.
[17] J. Kloosterman, *et al*., "Regless: just-in-time operand staging for gpus," in *Proc. IEEE/ACM of the 50th Annual Int. Symp. on Microarchitecture,* pp. 151-164, Boston, MA, USA, 14-17, Oct. 2017.
[18] M. Sadrosadati, A. Mirhosseini, S. B. Ehsani, H. Sarbazi-Azad, M. Drumond, B. Falsafi, R. Ausavarungnirun, and O. Mutlu, "Ltrf: enabling high-capacity register files for gpus via hardware/software cooperative register prefetching," in *Proc. of the 23rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems,* pp. 489-502, Williamsburg, VA, USA, 24-18 Mar. 2018.
[19] J. E. Lindholm, M. Y. Siu, S. S. Moy, S. Liu, and J. R. Nickolls, *Simulating Multiported Memories Using Lower Port Count Memories*, US Patent 7,339,592, 2008.
[20] *LTRF Register-Interval-Algorithm*, https://github.com/CMU-SAFARI/Register-Interval, 2018.
[21] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," *ISPASS.* in *Proc. IEEE Int. Symp. on Performance Analysis of Systems and Software,*, pp. 163-174, Boston, MA, USA, 26-28 Apr. 2009.
[22] S. Che, *et al.*, "Rodinia: a benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. on Workload Characterization*, pp. 44-54, Austin, TX, USA, 4-6 Oct. 2009.
[23] J. A. Stratton, *et al.*, "Parboil: a revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, p. 27, Mar. 2012.
[24] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "Stag: spintronic-tape architecture for GPGPU cache hierarchies," in *Proc. IEEE Int. Symp. on Computer Architecture,* pp. 253-264, Minneapolis, MN, USA, 14-18 Jun. 2014.
[25] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving gpu performance via large warps and two-level warp scheduling," in *Proc. IEEE/ACM of the 44th Annual Int. Symp. on Microarchitecture,* pp. 308-317, Porto Alegre, Brazil, 3-7, Dec. 2011.

راحیل براتی در حال حاضر به عنوان دستیار پژوهشی در پژوهشگاه دانش‌های بنیادی فعالیت می‌کند. او مدرک کارشناسی و کارشناسی ارشد خود را به ترتیب در سال‌های ۱۳۹۵ و ۱۳۹۸ از دانشگاه صنعتی شریف اخذ نموده است. موضوعات مرتبط با سیستم‌های چندهسته‌ای، پردازنده‌های گرافیگی و سیستم‌های حافظه از زمینه‌های تحقیقاتی مورد علاقه او است.

سید محمد صدرالساداتی در حال حاضر پژوهشگر پسادکترا در پژوهشگاه دانش‌های بنیادی است. او مدرک کارشناسی، کارشناسی ارشد و دکترای خود را در رشته‌ی مهندسی کامپیوتر به ترتیب در سال‌های ۱۳۹۱، ۱۳۹۳ و ۱۳۹۸ از دانشگاه صنعتی شریف کسب کرد. از حوزه‌های تحقیقاتی مورد علاقه وی می‌توان به معماری سیستم‌های چندهسته‌ای و زیادهسته‌ای، سیستم‌های حافظه و پردازش داخل حافظه اشاره کرد. وی در حوزه‌های تحقیقاتی خود مقالات متنوعی در همایش‌ها و ژورنال‌های معتبر بین‌المللی به چاپ رسانده است. در نتیجه تحقیقات عمیق وی در بهبود کارآمدی مصرف انرژی در پردازنده‌های گرافیکی، او در سال ۱۳۹۹ به عنوان برگزیده اول جشنواره جوان خوارزمی در بخش پژوهش‌های بنیادی انتخاب شد.

حمید سربازی آزاد مدرک کارشناسی خود را در رشته مهندسی کامپیوتراز دانشگاه شهید بهشتی در سال ۱۹۹۲ اخذ نمود، وی مدرک کارشناسی ارشد و دکتری خود را در رشته مهندسی کامپیوتر به ترتیب در سال‌های ۱۹۹۴ و ۲۰۰۲ از دانشگاه‌های صنعتی شریف و گلاسکو انگلستان کسب کرد. او هم‌اکنون استاد تمام دانشگاه صنعتی شریف و استاد پژوهشگر در پژوهشگاه دانش‌های بنیادی است. از علایق پژوهشی وی می‌توان به معماری کامپیوتر، معماری حافظه، شبکه و سیستم‌های روی تراشه، سیستم‌های موازی و توزیع شده، مدلسازی و ارزیابی کارایی و سیستم‌های ذخیره‌سازی اشاره کرد. وی دارای بیش از ۳۰۰ مقاله منتشر شده در کنفرانس‌ها و مجلات معتبر بین‌المللی است. او همچنین، موفق به کسب جایزه بین‌المللی خوارزمی در سال ۲۰۰۶ و جایزه دانشمند جوان TWAS در سال ۲۰۰۷ شده و در سال‌های ۲۰۰۴، ۲۰۰۷، ۲۰۰۸، ۲۰۱۰ و ۲۰۱۳ به عنوان پژوهشگر برتر دانشگاه صنعتی شریف انتخاب شده است..