Journal of Electronical & Cyber Defence
Vol. 10, No. 1,Spring 2022, Serial No. 37

An Approach to Dependability Enhancement of Cache Memories

M. Ghazvini*, M.J. Shahnavazi, B. Ghavami

*Associate Professor, Shahid Bahonar University of Kerman, Kerman, Iran
(Received: 30/01/2021, Accepted: 11/12/2021)

ABSTRACT

The modern processors that consist of large caches are very vulnerable to transient errors. Due to the importance of this problem, coding methods are adopted for protection against errors. The cost of reliability should be incurred to optimize the use of energy and area. This paper proposes an approach to dependability enhancement in unprotected caches of processors. For this purpose, the low-cost tag error mitigation mechanism is adopted to analyze the reliability of caches in modern processors. Benefiting from the tag Hamming distance increase technique, the proposed approach has a much lower overhead and decreases the false hit rate to zero.

Keywords: Hamming distance, reliability, false hit, cache memory, error.

*Corresponding Author Email: mghazvini@gmail.com

نشریه علمی "پدافند الکترونیکی و سایبری"
سال دهم، شماره ۱، بهار ۱۴۰۱، ص ۱۰-۱

علمی-پژوهشی

ارائه روشی جهت افزایش اتکاپذیری حافظه‌ی نهان

محمدجواد شهنوازی[۱]، مهدیه قزوینی[۲*]، بهنام قوامی[۳]
۱- دانشجوی کارشناسی ارشد، ۲ و ۳- دانشیار دانشگاه شهید باهنر، کرمان، ، ایران
(دریافت: ۱۳۹۹/۱۱/۱۱، پذیرش: ۱۴۰۰/۰۹/۲۰)

چکیده

پردازنده‌های مدرن که شامل حافظه‌ی نهان بزرگ است، نسبت به خطاهای گذرا آسیب‌پذیری بالایی دارد. اهمیت این موضوع باعث شده است تا از روش‌های کدگذاری برای محافظت در برابر خطا استفاده شود. هزینه‌ی قابلیت اطمینان باید به نحوی تأمین گردد تا استفاده‌ی بهینه از انرژی و ناحیه را در پی داشته باشد. در این مقاله روشی برای افزایش اتکاپذیری حافظه‌ی نهان محافظت نشده در پردازنده‌ها ارائه شده است. در این مقاله قابلیت اطمینان حافظه‌ی نهان در پردازنده‌های مدرن با استفاده از مکانیسم کاهش خطای برچسب کم‌هزینه مورد مطالعه قرار می‌گیرد. روش پیشنهادی از تکنیک افزایش فاصله‌ی همینگ در برچسب بهره‌برداری می‌کند. علاوه‌بر کاهش False Hit به صفر ، دارای سربار بسیار پایین نیز هست.

کلیدواژه‌ها: فاصله‌ی همینگ، اتکاپذیری، False Hit، حافظه‌ی نهان، اشکال

۱- مقدمه

یک واحد پردازشگر گرافیکی (GPU) شامل تعداد زیادی هسته است که به‌صورت کلی برای پردازش تصاویر به کار می‌رود. با توجه به پیشرفت تکنولوژی امکان استفاده از واحد پردازش گرافیکی در انواع وسیع‌تری از دامنه‌های محاسباتی امکان‌پذیر شده است. در راستای بهبود عملکرد و کاهش سربار انرژی ناشی از دسترسی به داده‌های حافظه‌ی اصلی، تمامی پردازنده‌های مدرن مانند واحد پردازش مرکزی (CPU) و GPU، چندین سطح از ساختارهای حافظه‌ی نهان را دارند که اگر به‌تازگی از داده‌ای استفاده شده باشد آن داده در حافظه‌ی نهان وجود دارد[1] و [2].

پیشرفت تکنولوژی و کوچک‌تر شدن قطعات الکترونیکی باعث شده است تا این قطعات نسبت به خطاهای گذرا حساس شوند. از طرفی، کاهش اندازه‌ی ساخت و کاهش سطوح ولتاژ، ساختارهای حافظه را نسبت به خطاهای گذرای سخت‌افزاری آسیب‌پذیر کرده است [3][4]. برای محافظت در برابر خطاهای گذرا، حافظه‌ی نهان، به‌طور معمول از مکانیزم‌های تشخیص/تصحیح کد مانند ECC/Parity استفاده می‌شود [5]. زمانی که خطوط حافظه‌ی نهان برای افزایش قابلیت اطمینان با بیت‌های ECC/Parity افزایش پیدا می‌کند به هنگام دسترسی به حافظه نیازمند فرآیند تشخیص/کدگذاری هستند. اگرچه طرح‌های محافظت در برابر خطا باعث تحت تأثیر قرار گرفتن مساحت و انرژی می‌شود، ولی با توجه به ضرورت قابلیت اطمینان برای حافظه‌ی نهان داده، استفاده از این روش ضروری است.

در صورت بروز خطا درون بیت‌های برچسب ممکن است در یک انطباق نادرست با برچسب مرجع باعث خواندن اطلاعات نادرست از حافظه‌نهان شود، که به آن Hit False می‌گویند. رویداد Hit False به وجود آمده باعث خروجی نادرست برنامه می‌شود. بنابراین اگر فاصله‌ی همینگ (تعداد تفاوت در بیت‌های متناظر دو رشته n بیتی) بیت‌های برچسب‌های درون یک مجموعه را افزایش دهیم، احتمال خروجی نادرست برنامه را کاهش داده‌ایم. به‌عبارتی‌دیگر، برچسب‌های ورودی در یک مجموعه در صورتی که با فاصله‌ی همینگ کمتر از ۲ از هم جدا شوند، در صورت رخ دادن یک خطای تک‌بیتی غیرایمن هستند [1].

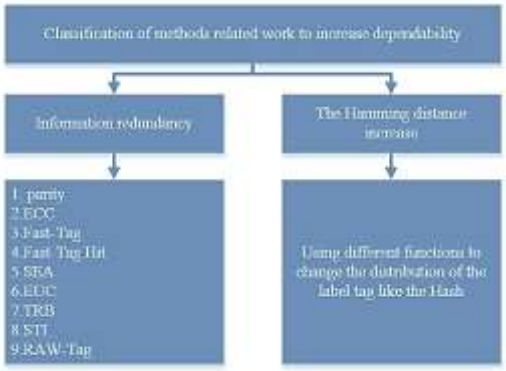
عمده‌ی کارهای انجام شده برای جلوگیری از خطا درون حافظه‌ی نهان در زمینه‌ی کدگذاری است. در این مقاله با استفاده از تعامل ساده‌ی تابع درهم‌ساز بر روی بیت‌های ثابت برچسب و تعریف یک ماشین حالت، فاصله‌ی همینگ از مقدار ۲ بیشتر را با سربار کم برای حافظه‌ی نهان L1 به وجود آمده است. روش ارائه‌شده علاوه‌بر بهبود Hit False سربار اجرایی پایینی نیز دارد.

ادامه‌ی این مقاله بدین شکل سازمان‌دهی شده است: در بخش دوم کارهای پیشین مرور خواهد شد. در بخش سوم مفاهیم و در بخش چهارم روش پیشنهادی بیان می‌شود. در بخش پنجم نیز ارزیابی‌های روش پیشنهادی انجام می‌شود.

۲- کارهای پیشین

در چند دهه‌ی اخیر کارهای زیادی برای افزایش تحمل‌پذیری حافظه‌ها در مقابل خطاهای گذرا انجام شده است. ما این روش‌ها را به دو دسته‌ی افزونگی اطلاعات و افزایش فاصله‌ی همینگ تقسیم‌بندی کرده‌ایم (شکل ۱).

* رایانامه نویسنده مسئول: mghazvini@gmail.com



شکل (۱): دسته‌بندی کارهای پیشین

۲-۱- افزونگی اطلاعات

افزونگی اطلاعات نیاز به نگهداری اطلاعات اضافی دارد. علاوه براین باعث ایجاد سربار هم، از لحاظ پردازش و مساحت می‌شود. ساده‌ترین روش افزونگی اطلاعات استفاده از کدهای تشخیص/تصحیح خطا مانند Parity/ECC است[6]. به‌طور کلی بیت توازن دارای فضای کم ولی با قابلیت اطمینان خیلی کم است؛ اگرچه کد ECC می‌تواند قابلیت اطمینان بیشتری به وجود آورد، اما توان و فضای بیشتری را طلب می‌کند.

در حال حاضر در حافظه‌ی نهان سطح اول کد ECC اصلاً استفاده نمی‌شود. روش Fast-Tag [7] تکنیکی برای اجرای مؤثر Parity/ECC برای برچسب حافظه‌ی نهان است. در حافظه‌ی نهان انجمن گروهی برای هر way نیاز به یک چک‌کننده‌ی بیت توازن یا ECC است. در روش Fast-Tag فقط برای آدرس ورودی بیت توازن یا کد ECC محاسبه‌شده و با برچسب‌های حافظه‌ی نهان بررسی می‌شود. به عبارتی چک‌کنندها از wayها حذف می‌شوند که باعث بهبود پیچیدگی مدار می‌شود. مزیت اصلی Fast-Tag کاهش پیچیدگی مدار مورد استفاده برای محافظت از حافظه‌ی نهان است (کاهش مساحت، انرژی و تأخیر را در پی دارد).

نویسندگان [8] با استفاده از Fast-Tag و فاصله‌ی همینگ تکنیک Hit Fast-Tag را برای حافظه‌ی نهان سطح دوم و سوم معرفی می‌کنند. روش Hit Fast-Tag برای خواندن برچسب از حافظه‌ی نهان ابتدا ECC برچسب را ایجاد می‌کند و با فاصله‌ی همینگ برچسب‌های موجود در مجموعه مقایسه می‌شود. اگر فاصله‌ی همینگ صفر باشد hit\miss داریم در غیر این صورت خطا رخ‌داده است سپس از ECC استفاده می‌شود.

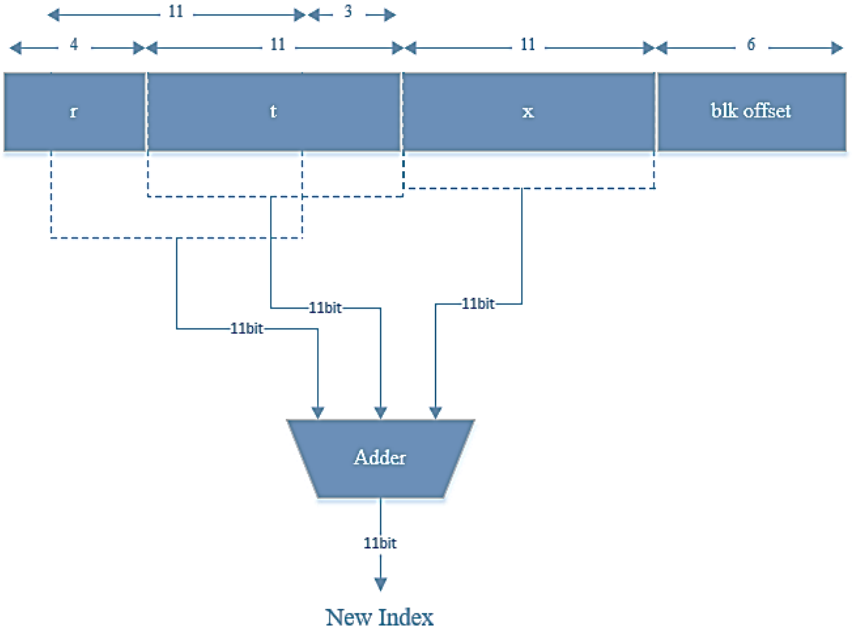
[9] با ارائه‌ی SEA[1] فضای اختصاصی برای ECC را حذف و آن را در فضای داده‌ی حافظه‌ی نهان قرار می‌دهد. مجموعه‌های حافظه‌ی نهان گروه‌بندی و به‌صورت جداگانه مدیریت می‌شود تا کارایی ECC بیت را افزایش دهد. این روش در (LLC) Last-cache Level پیاده می‌شود؛ زیرا عملکرد سیستم نسبت به ظرفیت LLC حساس نیست. اگر تمام ECCها وارد فضای LLC بشوند مقدار زیادی از فضای داده‌ی حافظه اشغال و به طور قابل توجهی باعث کاهش عملکرد سیستم می‌شود. داده‌هایی که تغییر نکرده‌اند را با استفاده از روش دیگری مانند بیت توازن و داده‌هایی که تغییر کرده‌اند و از آن‌ها در سیستم نسخه‌ی پشتیبان وجود ندارد را با ECC حفاظت می‌کنیم تا تعداد ECCهای مورد نیاز کاهش پیدا کند.

مهم‌ترین مسئله استفاده از ECC سربار است که باعث می‌شود در حافظه‌ی نهان سطح اول قابل استفاده نباشد. حامد فربه و همکاران [10] با معماری Cache ECC-United (EUC) که کارایی ECCها در حافظه‌ی نهان سطح اول را بهبود می‌بخشد معرفی کردند. معماری EUC محافظت از یک کلمه را به چندین کلمه افزایش می‌دهد. علاوه بر این دسترسی به خطوط حافظه‌نهان در یک مجموعه به‌صورت موازی انجام می‌شود، درحالی‌که تنها یک خط داده ارسال خواهد شد.

روش دیگر افزونگی اطلاعات، نگهداری کپی‌های اضافی از برچسب درون حافظه‌ی نهان است. درصورتی‌که در حافظه‌ی نهان باسیاست پس‌نویسی، اگر برچسب تحت خطای نرم تغییر کند، در حافظه‌ی نهان پایین‌تر به اشتباه نوشته می‌شود. بنابراین با توجه به اهمیت حیاتی آن برای صحت دسترسی به حافظه‌ی نهان، آرایه‌ی برچسب، اعتبار بسیار بالایی را در برابر خطاهای نرم دارا است. ونگ شوآی و همکاران [11] با بهره‌برداری از آدرس‌های هم‌جوار یک بافر تکرار برچسب (TRB) که نسخه‌هایی از برچسب‌هایی را که اغلب بازدید شده‌اند ضبط و نگهداری می‌کند و در صورت نیاز آن‌ها ارائه می‌دهد تا قابلیت اطمینان آرایه‌ی برچسب در حافظه‌ی نهان L1 را افزایش دهد. یکی از مسائل کلیدی در طراحی TRB نحوه‌ی شناسایی برچسب اصلی با تکراری است. برای این منظور طراحی اشاره‌گر اتخاذ شده است. قسمت اشاره‌گر در طرف ردیف اصلی برچسب قرار می‌گیرد. هر

[1] Smart ECC Allocation

ورودی برچسب در آرایه‌ی برچسب اصلی با یک اشاره‌گر به محل کپی آن در TRB مرتبط می‌شود. TRB تنها شامل کپی برچسب است. یک بیت اضافی (بیت کپی) برای هر ورودی برچسب اصلی اضافه شده تا نشان دهد که آیا یک کپی در TRB وجود دارد یا نه. این طرح، برچسب را تا ۵۵% بازیابی می‌کند و دارای مساحت بالای ۱۷۵% و توان ۳۷% است. برای کاهش مساحت و توان طرح TRB-PTR را ارائه کردند[11].



شکل (۲): سخت‌افزار تابع درهم‌ساز [16].

با استفاده از بیت‌های مشابه (STI)[1] می‌توان قابلیت اطمینان حافظه‌ی نهان را افزایش داد. نتایج تجربی نشان می‌دهد که بسیار محتمل است که بیت‌های یکسان در مجموعه‌های مجاور حافظه‌ی نهان وجود داشته باشد [12]. بدین منظور می‌توان برای بهبود قابلیت اطمینان آرایه‌ی برچسب در برابر خطاها از بیت‌های مشابه در مجموعه‌های مجاور استفاده نمود[12]. با استفاده از هم‌جواری مکانی، بیت‌های مشابه درون مجموعه‌های مجاور یافت می‌شود. وقتی عمل واکشی داده‌ها از حافظه‌ی اصلی انجام می‌شود، باید بررسی شود که خطوط مجاور حافظه‌ی نهان همان برچسب را دارد یا نه. این اطلاعات اضافه به‌عنوان بیت‌های اضافه ذخیره می‌شود. طرح STI دارای مساحت، انرژی و سربار اجرایی کم و با پوشش حفاظتی خطا 97.7% به‌طور متوسط است.

تکرارهای ذاتی برای آرایه‌ی برچسب قابلیت اطمینان کافی ندارد؛ زیرا درصد زیادی از برچسب‌ها بدون تکرار باقی می‌ماند. RAW-Tag[2] [13] مشابه طرح STI از برچسب‌های مشابه درون مجموعه‌های مجاور استفاده می‌کند. برای تضمین برچسب‌های dirty ، RAW-Tag با بررسی مجموعه‌های مجاور بالایی و پایینی اگر برچسب مشابه را یافت نکند یک عملیات پیش‌واکشی و write-back زودهنگام برای عملیات معمولی حافظه‌ی نهان انجام می‌دهد تا یک کپی برای برچسب dirty به درون حافظه‌ی نهان آورده شود.

۲-۲- افزایش فاصله‌ی همینگ

یک اشکال گذرا زمانی می‌تواند باعث Hit False شود که فاصله‌ی همینگ برچسب‌های درون یک مجموعه کمتر از ۲ باشد. یکی از راه‌های جلوگیری از رویداد Hit False افزایش فاصله‌ی همینگ برچسب‌های ورودی یک مجموعه با برهم زدن توزیع فاصله‌ی همینگ برچسب در کل حافظه‌ی نهان است[1]. یکی از توابعی که این امکان را به ما می‌دهد تا توزیع برچسب در حافظه‌ی نهان را تغییر دهیم توابع درهم‌ساز است. اگر تابع مناسبی انتخاب شود منجر به افزایش فاصله‌ی همینگ و بهبود عملکرد می‌شود. توابع درهم‌ساز به‌طور گسترده در معماری کامپیوتر برای به حداقل رساندن تضادهای منجر به miss در حافظه‌های نهان [14] یا برای بهبود دسترسی به حافظه‌های چندبانکی [15] استفاده می‌شود. همچنین توابع درهم ساز یکی از پُرکاربردترین بلوک‌های مخابراتی در طراحی پروتکل‌های مخابرات دیجیتال است [16]. بسته به موقعیت، یک نوع از توابع درهم‌ساز می‌تواند با توجه به ویژگی‌های الگوهای دسترسی، مناسب‌تر باشد. برای مثال تابع درهم‌ساز ارائه‌شده در مقاله[17] برای محاسبه مجموعه باعث کاهش تضادهای منجر به miss

[1] Same tag information
[2] Replicating in Altered Cache Ways

می‌شود(شکل ۲). نویسندگان [1] با استفاده از تابع درهم‌ساز که مشتق‌شده از چند جمله‌ای‌های گالیوس فیلد است، هر دو آدرسی که به یک مجموعه ارجاع داده شده‌اند را با فاصله‌ی همینگ ۲ یا بیشتر از هم جدا می‌کند. استفاده از توابع درهم‌ساز موجب افزایش زمان دسترسی به میزان کم، اما در عوض هزینه‌ی پیاده‌سازی کم و باعث کاهش Hit False با فاکتور ۱۰ می‌شود. اتکاپذیری در اینجا به این صورت است که در صورت رخ دادن اشکال فقط با یک miss روبه‌رو خواهیم شد و خطا/شکست رخ نمی‌دهد (جدول ۱).

جدول (۱): بررسی روش‌ها

دسته بندی	روش	مزایا	معایب	مراجع
افزونگی اطلاعات	کدهای تشخیص و تصحیح خطا	انعطاف‌پذیری بالا	کاهش عملکرد، افزایش مساحت و توان	[6,7,8,9,10,22]
	بافر کپی برچسب	امکان بازیابی برچسب دچار اشکال شده تا ۵۵%	مساحت و توان زیاد	[11]
	استفاده از بیت‌های مشابه	سربار اجرایی کم و پوشش حفاظتی خطا تا ۹۷%	نیاز به اصلاحات معماری دارد	[12,13]
افزایش فاصله‌ی همینگ	توابع درهم‌ساز	بهبود عملکرد و کاهش Hit False با فاکتور ۱۰	افزایش زمان دسترسی به میزان کم	[1,14,15,17]

۳- مفاهیم

۳-۱- رویداد Hit False

اگر یک آرایه‌ی برچسب حافظه SRAM محافظت‌نشده داشته باشیم، خطا در ورودی برچسب می‌تواند منجر به Hit False یا Miss False بشود[1].

فرض کنید که در یک ورودی برچسب محافظت‌نشده خطایی رخ می‌دهد و باعث تغییر مقدار T[i][j] (i مشخص‌کننده‌ی مجموعه و j نشان دهنده way) از TA1 به TA2 می‌شود.

- ✓ فرض کنید برچسب SRAM یک آدرس جدید با مجموعه i و مقدار TA1 دریافت می‌کند. در حالتی که خطا وجود ندارد باید Hit رخ دهد؛ اما به دلیل وجود خطای گذرا نتیجه Hit نمی‌شود. این رویداد Miss False نامیده می‌شود.
- ✓ فرض کنید برچسب SRAM یک آدرس جدید با مجموعه i و مقدار TA2 (دارای یک بیت فاصله‌همینگ با TA1) دریافت می‌کند. در این سناریو، آدرس دریافت‌شده با مقدار نادرست TA2 مطابقت داده می‌شود و Hit رخ می‌دهد. این وضعیت Hit False نامیده می‌شود. رویداد Hit False تحت دو وضعیت رخ می‌دهد:

- زمانی که آدرس جدید با مقدار TA2 داریم که درون برچسب SRAM وجود ندارد اما مقدار TA1 تحت خطا به مقدارTA2 تبدیل شده است. این وضعیت را Hit False تحت یک miss درست می‌نامیم.
- زمانی که آدرس جدید با مقدار TA2 داریم که درون برچسب SRAM وجود دارد و اینکه مقدار TA1 تحت خطا به مقدارTA2 تبدیل شده است. در صورتی که TA2 اشتباه انتخاب شود رویداد Hit False تحت یک Hit درست رخ داده است.

۳-۲- محاسبه‌ی رویداد Hit False

رویداد Hit False و Miss False زمانی که یک خطای تک‌بیتی درون خطوط حافظه‌ی نهان اتفاق می‌افتد، رخ می‌دهد. منبع [1] راهکاری برای اندازه‌گیری Rate Hit False ارائه کرده و نیازی به تزریق اشکال و یا اجرای چند بار برنامه نیست و فقط با یک‌بار اجرای برنامه معیارها اندازه‌گیری می‌شود. در ابتدا نحوه‌ی محاسبه rate Hit False را بیان می‌کنیم. Hit False زمانی رخ می‌دهد که رویداد Miss cache یا یک رویداد Hit cache داشته باشیم. برای شمارش hit falseهای تحت hit cache یا miss cache مفاهیم SetEntryFalseHits_Hit و SetEntryFalseHits_Miss را برای هر مجموعه تعریف می‌کنیم. اگر دسترسی به حافظه‌ی نهان منجر به hit شود، فاصله‌ی همینگ برچسب آدرس جدید با تمامی برچسب‌های معتبر مجموعه مقایسه می‌کنیم، در صورتی که فاصله‌همینگ ۱ باشد SetEntryFalseHits_Hit را یک واحد افزایش می‌دهیم. همین طور، دسترسی به حافظه‌ی نهان منجر به miss بشود، و فاصله‌ی همینگ برچسب آدرس جدید با برچسب‌های معتبر مجموعه برابر ۱ باشد SetEntryFalseHits_Miss را یک واحد افزایش می‌دهیم.

در پایان بعد از اجرای برنامه برای هر مجموعه hit False را تخمین می‌زنیم. شمارنده‌ی هر مجموعه را بر تعداد بیت‌های مجموعه تقسیم می‌کنیم (رابطه‌ی ۱ و ۲) و تحت عنوان EstimatedFalseHits برای هر مجموعه نگهداری می‌کنیم.

$$EstimatedFalseHits\_Miss_i = \frac{SetEntryFalseHits\_miss_i}{\text{Associativity} \times \text{TagBitWidth}} \quad (۱)$$

$$EstimatedFalseHits\_Hit_i = \frac{SetEntryFalseHits\_Hit_i}{\text{Associativity} \times \text{TagBitWidth}} \quad (۲)$$

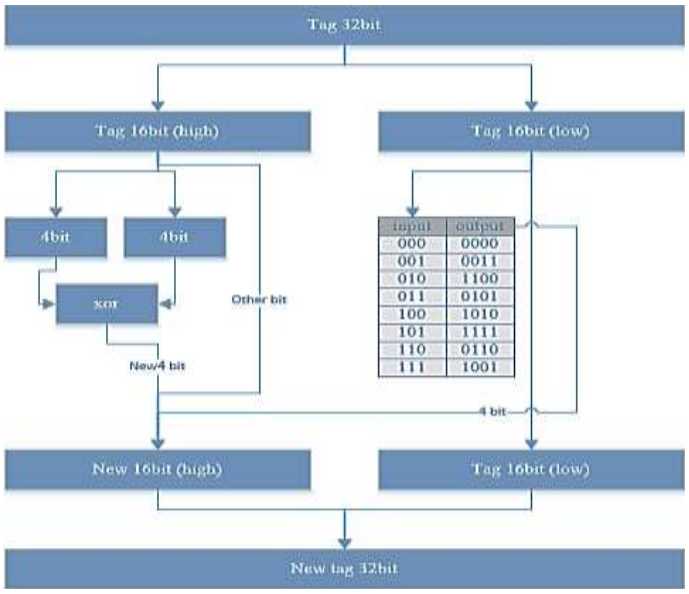
برای برآورد کردن نرخ hit False درون حافظه‌ی نهان، hit under hits false estimated (برای miss under hits false) به‌صورت جداگانه محاسبه می‌کنیم) برای همه‌ی مجموعه‌ها را محاسبه می‌کنیم و به تعداد دسترسی‌های حافظه‌ی نهان تقسیم می‌کنیم (رابطه‌ی ۳ و ۴).

$$FalseHitRate_{Miss} = \frac{\sum_{i=1}^{\#sets} EstimatedFalseHits_{Miss_i}}{Accesses} \quad (۳)$$

$$FalseHitRate\_Hit = \frac{\sum_{i=1}^{\#sets} EstimatedFalseHits\_Hit_i}{Accesses} \quad (۴)$$

۴- روش پیشنهادی

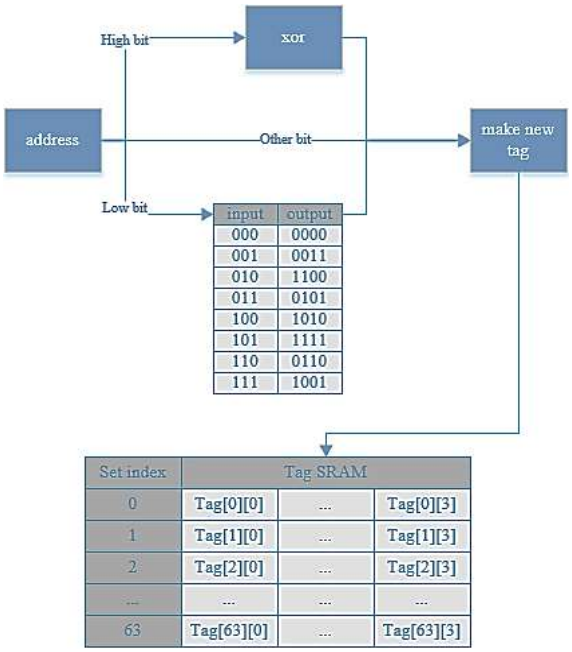
روش ارائه‌شده، که آن را XOR and Machine State (SM_X) می‌نامیم، با افزایش فاصله‌ی همینگ بین برچسب‌های ورودی یک مجموعه‌ی حافظه‌ی نهان باعث جلوگیری از رویداد hit False می‌شود. شکل ۳ نمای کلی روش پیشنهادی را نشان می‌دهد. روش SM_X از دو بخش جداگانه که درنهایت باهم ادغام می‌شوند تشکیل‌شده است. بخش اول با استفاده از حالت‌های ازپیش‌تعریف‌شده آماده شده، بخش دوم با استفاده از بیت‌های پرارزش برچسب به وجود می‌آید.



شکل (۳): نمای کلی

بخش اول، SM_X با استفاده از بیت‌های کم‌ارزش برچسب یکی از حالت‌های جدول شکل ۳ را انتخاب می‌کند که هر کدام از حالت‌ها فاصله‌ی همینگ دو نسبت به دیگر حالت‌ها دارد. با توجه به اینکه نتایج تجربی نشان می‌دهد که بیت‌های پرارزش برچسب به‌ندرت تغییر می‌کند [11] بنابراین SM_X در بخش دوم تغییراتی درون بیت‌های پرارزش ایجاد می‌کند تا فاصله‌ی همینگ بیشتری داشته باشیم. در بخش دوم، دو قسمت ۴بیتی (با توجه به بیت‌های حالت‌های ازپیش‌تعریف‌شده) از بیت‌های پرارزش را انتخاب و از برچسب برمی‌داریم تا قسمت اول را با بخش اول SM_X جایگزین و قسمت دوم را از نتیجه‌ی xor دو قسمت بیت‌های پرارزش پر کنیم و در کنار بقیه‌ی بیت‌های برچسب قرار دهیم. هر دو عمل به‌صورت موازی اجرا می‌شوند تا به سربار تأخیر کمتر تحمیل شود (شکل ۴).

در واقع SM_X قسمتی از بیت‌های برچسب را تغییر می‌دهد تا فاصله‌ی همینگ برچسب‌های درون یک مجموعه با هم بیشتر از دو باشد. SM_X در بخش اول فاصله‌ی همینگ دو و در بخش دوم روش تضمین فاصله‌ی همینگ بیشتر از دو را به وجود می‌آوریم. شایان به ذکر است که با استفاده از روش SM_X می‌توانیم FaسسسlseFa hit را به صفر برسانیم. هر برچسب ورودی با بقیه‌ی برچسب‌های ورودی مجموعه حداقل فاصله‌ی همینگ دو را دارا است. بدین ترتیب از به وجود آمدن خطا یا خروجی نادرست برنامه جلوگیری می‌شود که افزایش اتکاپذیری را در پی دارد.



شکل (۴): اجرای موازی دو بخش SM_X

۵- ارزیابی

در این بخش، نتایج تجربی استفاده از اعمال روش پیشنهادی SM_X درون حافظه‌ی نهان L1 در AMD GPU Southern- island را نشان می‌دهیم. نرخ به‌دست آمده Hit False و Cache Hit از روش SM_X را در کنار روش Hash-base [1] به نمایش می‌گذاریم. علاوه‌براین (۵) Accuracy، و (۶) Specificity را تعریف می‌کنیم تا دقت روش پیشنهادی را نسبت به حالت عادی بسنجیم (جدول ۲ و ۳).

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (۵)$$

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (۶)$$

جدول(۲): معیار Accuracy

Benchmark	normal	Hash-base	SM_X
DCT	۹۹/۸۶	۹۹/۹۹	۱۰۰
EigenValue	۹۹/۷	۹۹/۹۹	۱۰۰
Histogram	۹۹/۳۳	۹۹/۹	۱۰۰
MatrixMultiplication	۹۹/۹۱	۹۹/۹۹	۱۰۰
MatrixTranspose	۹۹/۹۱	۱۰۰	۹۹/۹۷
MersenneTwister	۹۹/۸۷	۹۹/۹۸	۱۰۰
PerfixSum	۹۹/۷۵	۹۹/۹۹	۱۰۰
RadixSort	۹۹/۹۲	۱۰۰	۱۰۰
Reduction	۹۹/۷	۹۹/۹۹	۱۰۰
ScanLargeArrays	۹۹/۸۳	۹۹/۹۹	۱۰۰
SobelFilter	۹۹/۷۵	۹۹/۹۵	۱۰۰
URNG	۹۹/۶۳	۹۹/۹۱	۱۰۰

جدول(۳): معیار Specificity

Benchmark	normal	Hash-base	SM_X
DCT	۹۹/۷۵	۹۹/۹۹	۱۰۰
EigenValue	۱۰۰	۱۰۰	۱۰۰
Histogram	۹۹/۶۶	۹۹/۹	۱۰۰
MatrixMultiplication	۹۹/۹۵	۱۰۰	۱۰۰
MatrixTranspose	۹۹/۸۹	۱۰۰	۹۹/۹۸
MersenneTwister	۹۹/۸۷	۱۰۰	۱۰۰
PerfixSum	۹۹/۸۸	۱۰۰	۱۰۰
RadixSort	۹۹/۹۶	۱۰۰	۱۰۰
Reduction	۹۹/۷۹	۱۰۰	۱۰۰
ScanLargeArrays	۹۹/۸۲	۹۹/۹۸	۱۰۰
SobelFilter	۹۹/۸۵	۹۹/۹۹	۱۰۰
URNG	۹۹/۶۳	۹۹/۹۱	۱۰۰

الف) تنظیمات و محیط شبیه‌سازی

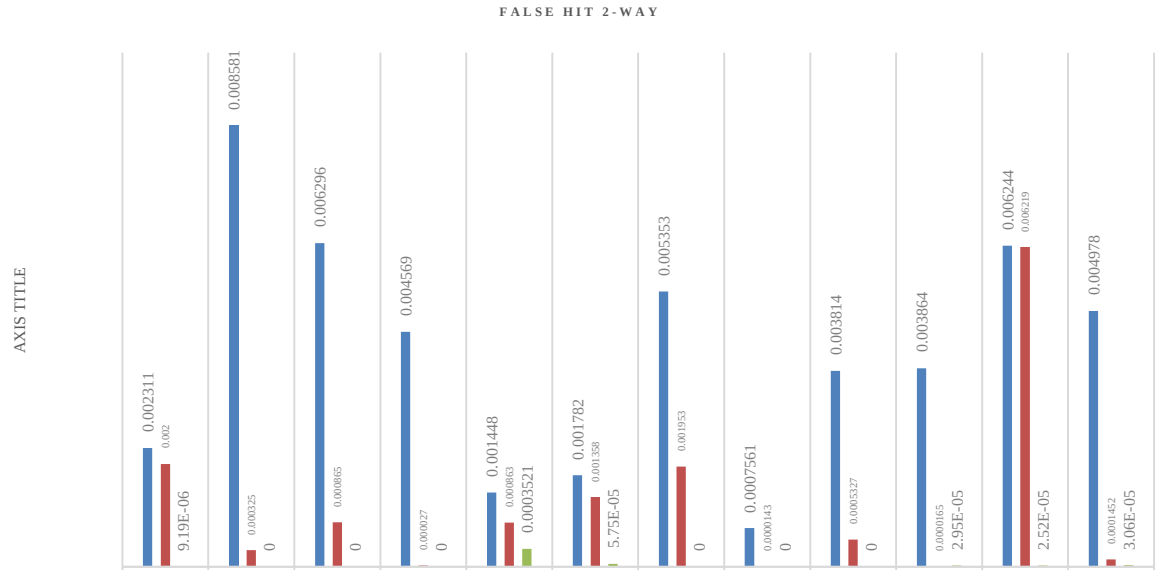
یک GPU معمولی شامل چندین خوشه GPC[1] و شامل چندین SM[2] به‌ازای هر GPC است. هر کدام از SMها می‌تواند به‌عنوان یک پردازنده مستقل دیده شود. به طور خاص، هر SM دارای حافظه‌ی نهان داده و دستورالعمل L1، scheduler و واحد dispatch است. حافظه‌ی نهان سطح دوم برای ارتباط بین SMها و یک حافظه Global برای دسترسی همه SM ها وجود دارد. تعداد قابل توجهی برچسب SRAM درون ساختار GPU وجود دارد[18][19] که باید در برابر خطاهای گذرا محافظت شوند.

[1] GPU Processing Clusters
[2] Streaming Multiprocessors

Multi2sim[17] یک شبیه‌ساز CPUها و GPUها است که برای آزمایش و اعتبارسنجی سخت‌افزار جدید طراحی شده قبل از ساخته شدن استفاده می‌شود. [19] از مجموعه بنچمارک‌های ارائه شده برای Multi2sim از مجموعه m2s-bench-amdapp-2.5 برای ارزیابی استفاده می‌کنیم.

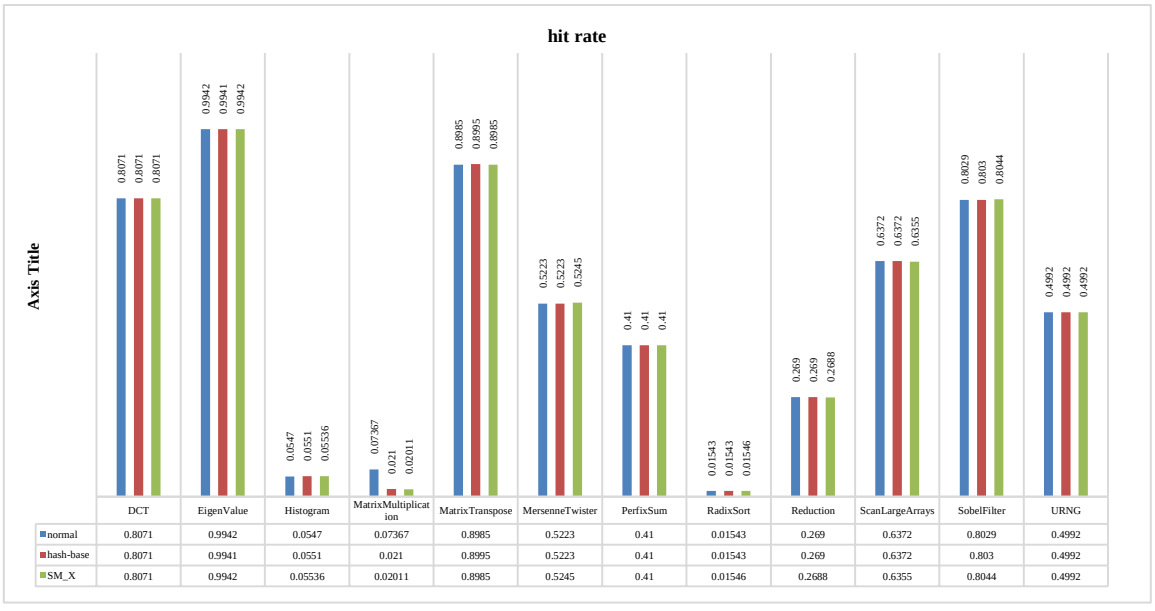
ب) ارزیابی ساختار برچسب SRAM حافظه‌نهان

برای سه حافظه‌ی نهان 2-way، 4-way و 8-way با set=64 و size=64 Block، آزمایش را انجام دادیم. در ابتدا نتایج حافظه‌ی نهان 2-way با ظرفیت 8KB را ارائه می‌دهیم. نمودار(۲) نشان‌دهنده‌ی نرخ Hit False برای سه حالت نرمال، Hash-base [1] و SM_X است. هر سه طرح دارای نرخ Hit False کمتر از 0.1% هستند و همان‌طور که از نمودار بر می‌آید SM_X بیشتر به صفر نزدیک است. تأثیر هر روش بر روی نرخ Hit درون نمودار(۳) نمایش داده شده است. نرخ Hit برای تمامی حالات خیلی نزدیک یا مشابه هم هستند.



	DCT	EigenValue	Histogram	MatrixMultiplication	MatrixTranspose	MersenneTwister	PerfixSum	RadixSort	Reduction	ScanLargeArrays	SobelFilter	URNG
normal	0.002311	0.008581	0.006296	0.004569	0.001448	0.001782	0.005353	0.0007561	0.003814	0.003864	0.006244	0.004978
hash-base	0.002	0.000325	0.000865	0.000027	0.000863	0.001358	0.001953	0.0000143	0.0005327	0.0000165	0.006219	0.0001452
SM_X	9.19E-06	0	0	0	0.0003521	5.75E-05	0	0	0	2.95E-05	2.52E-05	3.06E-05

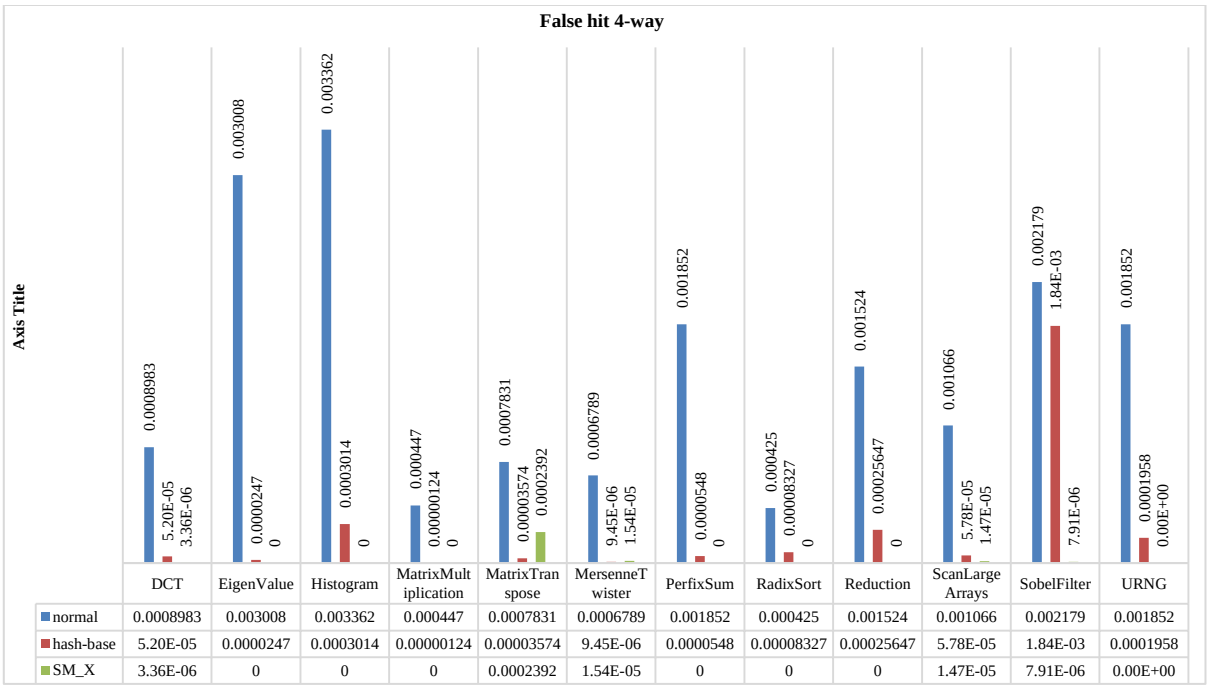
نمودار(۲): حافظه‌ی نهان 2-way با ظرفیت 8KB



	DCT	EigenValue	Histogram	MatrixMultiplication	MatrixTranspose	MersenneTwister	PerfixSum	RadixSort	Reduction	ScanLargeArrays	SobelFilter	URNG
normal	0.8071	0.9942	0.0547	0.07367	0.8985	0.5223	0.41	0.01543	0.269	0.6372	0.8029	0.4992
hash-base	0.8071	0.9941	0.0551	0.021	0.8995	0.5223	0.41	0.01543	0.269	0.6372	0.803	0.4992
SM_X	0.8071	0.9942	0.05536	0.02011	0.8985	0.5245	0.41	0.01546	0.2688	0.6355	0.8044	0.4992

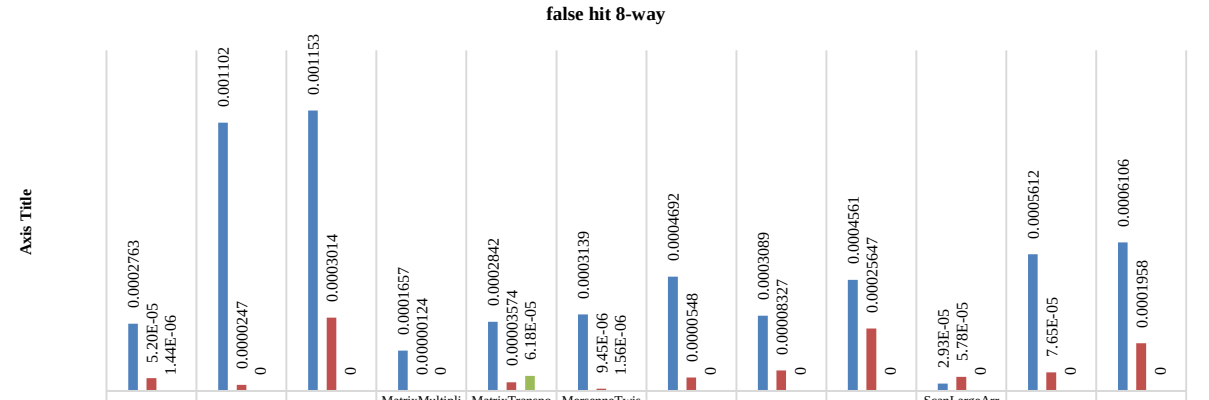
نمودار(۳): Rate Hit

در آزمایش دوم، از حافظه‌نهان 4-way با ظرفیت 16KB استفاده می‌کنیم تا تأثیر روش SM_X بر روی افزایش سایز حافظه‌ی نهان را بررسی کنیم. نمودار ۴ نرخ Hit False برای هر سه طرح بر روی حافظه‌ی نهان 4-way را نشان می‌دهد. با توجه به نمودار(۴) متوجه خواهیم شد که افزایش سایز حافظه‌ی نهان به خودی خود باعث کاهش hit False می‌شود. همانند نمودار ۲ برای هر سه حالت نرخ Hit False کمتر از ۰/۱% است.



	DCT	EigenValue	Histogram	MatrixMultiplication	MatrixTranspose	MersenneTwister	PerfixSum	RadixSort	Reduction	ScanLargeArrays	SobelFilter	URNG
normal	0.0008983	0.003008	0.003362	0.000447	0.0007831	0.0006789	0.001852	0.000425	0.001524	0.001066	0.002179	0.001852
hash-base	5.20E-05	0.0000247	0.0003014	0.00000124	0.00003574	9.45E-06	0.0000548	0.00008327	0.00025647	5.78E-05	1.84E-03	0.0001958
SM_X	3.36E-06	0	0	0	0.0002392	1.54E-05	0	0	0	1.47E-05	7.91E-06	0.00E+00

نمودار(۴): حافظه‌ی نهان 4-way با ظرفیت 16KB



	DCT	EigenValue	Histogram	MatrixMultiplication	MatrixTranspose	MersenneTwister	PerfixSum	RadixSort	Reduction	ScanLargeArrays	SobelFilter	URNG
normal	0.0002763	0.001102	0.001153	0.0001657	0.0002842	0.0003139	0.0004692	0.0003089	0.0004561	2.93E-05	0.0005612	0.0006106
hash-base	5.20E-05	0.0000247	0.0003014	0.00000124	0.00003574	9.45E-06	0.0000548	0.00008327	0.00025647	5.78E-05	7.65E-05	0.0001958
SM_X	1.44E-06	0	0	0	6.18E-05	1.56E-06	0	0	0	0	0	0

نمودار(۵): حافظه‌ی نهان 8-way با ظرفیت 24KB

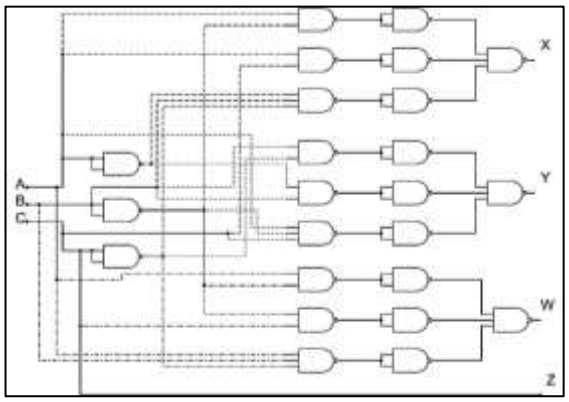
در آزمایش سوم، حافظه‌ی نهان 8-way با ظرفیت 24KB را در نظر می‌گیریم. از آنچه که نمودار ۵ نشان می‌دهد بر می‌آید که افزایش way در حالت عادی باعث کاهش نرخ Hit False می‌شود. در اینجا مشاهده می‌کنیم تأثیر افزایش way بر روش SM_X باعث شده تا نرخ False Hit صفر یا خیلی نزدیک به صفر را ارائه دهد.

پ)عملکرد و بیت‌های موردنیاز SM_X

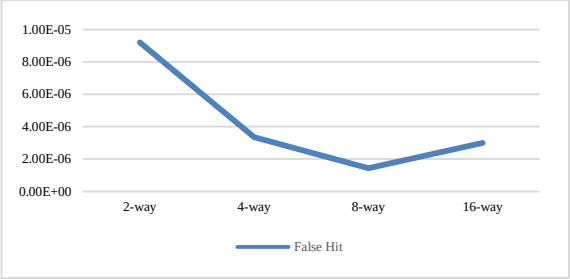
هرچه تعداد بیت‌های استفاده‌شده برای SM_X بیشتر باشد امکان تعریف حالت‌های بیشتر مهیا می‌شود. اگر تعداد wayهای حافظه‌ی نهان کمتر از تعداد حالت‌های تعریف شده باشد عملکرد مطلوبی را مشاهده خواهیم کرد، اما اگر تعداد wayهای حافظه‌ی نهان بیشتر از حالت‌ها باشد عملکرد SM_X کاهش پیدا می‌کند

(نمودار ۶). استفاده از حالت‌های بیشتر در گرو اشغال کردن تعداد بیت بیشتر از تگ است. همین‌طور هرچه اندازه‌ی حافظه‌ی نهان بیشتر شود، هم‌جواری مکانی درون یک مجموعه کمتر دیده می‌شود. البته در صورتی که تعداد Way ثابت و تعداد مجموعه‌ها افزایش پیدا کند.

تأخیر به وجود آمده از تبدیل برچسب با روش ما، به‌اندازه‌ی یک xor و یک مدار ترکیبی است که با هم به‌صورت موازی انجام می‌شود تا تأخیر به حداقل برسد. ما با استفاده از گیت nand مدار ترکیبی را پیاده‌سازی کردیم، تأخیر به‌وجودآمده به‌اندازه‌ی ۴ سطح گیت nand است (شکل ۵). با بهره‌گیری از ابزار McPAT[21] مساحت و توان مصرفی را بررسی و دیدیم که طرح پیشنهادی تأثیری بر مساحت و توان پردازنده ندارد.



شکل (۵): مدار ترکیبی برای پیاده‌سازی حالت‌های ازپیش‌تعریف‌شده



نمودار (۶): روند Hit False با توجه به تعداد Way

۶- نتیجه‌گیری

بخش اعظمی از پردازنده‌های گرافیکی شامل حافظه‌ها است. حافظه‌ی نهان نسبت به سایر بخش‌ها بیشتر در معرض خطای گذرا است. به‌طور معمول حافظه‌ی نهان با استفاده از کدهای Parity/ECC محافظت می‌شود که از ECC به‌خاطر سربار مساحت و کارایی درون حافظه‌ی نهان L1 استفاده نمی‌شود، Parity فقط قادر به تشخیص یک خطای تک‌بیتی است. در این مقاله ما با استفاده از روش پیشنهادی، فاصله‌ی همینگ برچسب‌های درون یک مجموعه را افزایش می‌دهیم. به‌نحوی که فاصله‌ی همینگ همواره از مقدار دو بیشتر باشد؛ بنابراین از رویداد Hit False که منجر به خروجی اشتباه برنامه می‌شود جلوگیری کنیم.

۷- منابع

[1] Lotfi, N. Saxena, R. Bramley, P. Racunas and P. Shirvani, "Low Overhead Tag Error Mitigation for GPU Architectures," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg City, 2018, pp. 314-321.

[2] H. Wen and W. Zhang, "Heterogeneous Cache Hierarchy Management for Integrated CPU-GPU Architecture," *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, 2019, pp. 1-6, doi: 10.1109/HPEC.2019.8916239.

[3] R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," in *Proceedings of International Electron Devices Meeting*, 2002.

[4] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, 2003.

[5] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai and S. W. Keckler, "Optimizing Software-Directed Instruction Replication for GPU Error Detection," *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA, 2018, pp. 842-854, doi: 10.1109/SC.2018.00070.

[6] R. W. Hamming, "Error detecting and error correcting codes," The Bell System Technical Journal, vol. 29, no. 2, pp. 147–160, April 1950.

[7] P. Reviriego, S. Pontarelli, M. Ottavi and J. A. Maestro, "FastTag: A Technique to Protect Cache Tags Against Soft Errors," in IEEE Transactions on Device and Materials Reliability, vol. 14, no. 3, pp. 935-937, Sept. 2014.

[8] A. Gendler, A. Bramnik, A. Szapiro and Y. Sazeides, "Don't Correct the Tags in a Cache, Just Check Their Hamming Distance from the Lookup Tag," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 571-582, doi: 10.1109/HPCA.2018.00055.

[9] J. Hong and S. Kim, "Smart ECC Allocation Cache Utilizing Cache Data Space," in

IEEE Transactions on Computers, vol. 66, no. 2, pp. 368-374, 1 Feb. 2017.

[10] H. Farbeh, L. Delshadtehrani, H. Kim and S. Kim, "ECC-United Cache: Maximizing Efficiency of Error Detection/Correction Codes in Associative Cache Memories," in *IEEE Transactions on Computers*, doi: 10.1109/TC.2020.2994067.

[11] S. Wang, J. Hu and S. G. Ziavras, "Replicating Tag Entries for Reliability Enhancement in Cache Tag Arrays," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no. 4, pp. 643-654, April 2012.

[12] J. Hong, J. Kim and S. Kim, "Exploiting Same Tag Bits to Improve the Reliability of the Cache Memories," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 2, pp. 254-265, Feb. 2015.

[13] H. Farbeh, F. Mozafari, M. Zabihi and S. G. Miremadi, "RAW-Tag: Replicating in Altered Cache Ways for Correcting Multiple-Bit Errors in Tag Array," in *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 651-664, 1 July-Aug. 2019.

[14] Antonio González, Mateo Valero, Nigel Topham, and Joan M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. In Proceedings of the 11th international conference on Supercomputing (ICS '97). Association for Computing Machinery, New York, NY, USA, 76–83. 1997. DOI:https://doi.org/10.1145/263580.263599

[15] Zhao Zhang, Zhichun Zhu and Xiaodong Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000, Monterey, CA, USA, 2000, pp. 32-41.

[16] Ghazi Maghribi, Saeed, Alemi, Hadi. A new way to identify the blind of the initial state of the synchronous hash after the channel encoder. Electronic and Cyber Defense, 1400; 9 (1): 19-27.

[17] M. Kharbutli, Y. Solihin and Jaejin Lee, "Eliminating conflict misses using prime number-based cache indexing," in IEEE Transactions on Computers, vol. 54, no. 5, pp. 573-586, May 2005.

[18] R. Ubal, B. Jang, P. Mistry, D. Schaa and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, 2012, pp. 335-344.

[19] Y. Arafa, A. A. Badawy, G. Chennupati, N. Santhi and S. Eidenbenz, "PPT-GPU: Scalable GPU Performance Modeling," in IEEE Computer Architecture Letters, vol. 18, no. 1, pp. 55-58, 1 Jan.-June 2019, doi: 10.1109/LCA.2019.2904497.

[20] https://github.com/Multi2Sim/

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009, pp. 469-480

[22] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," in IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, pp. 397-404, Sept. 2005, doi: 10.1109/TDMR.2005.856487.