Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: www.jscit.nit.ac.ir
Volume 7, Number 1, Spring & Summer 2018, pp. 1-12
Received: 2018/02/14; Revised: 2018/09/05; Accepted: 2018/10/16

An Enhanced Method for Detecting Control Flow Errors Caused by Soft Errors in the Processors Running the Programs

Seyedeh Afifeh Alavi[1], Mojtaba Valinataj[2]*, Mojtaba Mansoori[3]
1- Dept. of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Babol, Iran.
2*- Corresponding Author: Dept. of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Shariati Ave., Babol, Iran.
3- Dept. of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Babol, Iran.
[1] afifeh.alavi@yahoo.com, [2*]m.valinataj@nit.ac.ir, [3]mansoori@nit.ac.ir

Corresponding author address: Mojtaba Valinataj, Faculty of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Babol, Iran, Post Code : 47148 – 71167.

Abstract- The processing systems utilized in the satellites or nuclear reactors are highly susceptible to produce wrong results because of the existence of different radiations. In this paper, a new method is proposed for enhancing the program executions on the fault-prone processors in extreme environments. The main goal is the detection of control flow errors caused by soft errors that maybe produced by transient faults occurred in the underlying hardware. This method detects the errors by recognizing the deviations from the proper program flow execution. The proposed method operates by allocating two specific signatures and at most three control instructions to each basic block inside a program. This way, many control flow errors are detected. The experimental results based on the simulation of the proposed method together with the previous methods show that the proposed method detects the control flow errors better than the previous methods with respect to three main parameters including fault coverage, performance overhead and memory overhead.

Keywords- Transient Errors, Error Detection, SIHFT, Control Flow Checking, Data Flow Checking.

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: www.jscit.nit.ac.ir
جلد ۷، شماره ۱، فصل بهار و تابستان ۱۳۹۷، صفحه ۱-۱۲
دریافت: ۱۳۹۶/۱۱/۲۵، بازنگری: ۱۳۹۷/۰۶/۱۴، پذیرش: ۱۳۹۷/۰۷/۲۴

روشی برای بهبود تشخیص نرم‌افزاری خطاهای کنترلیِ مبتنی بر خطاهای گذرا در پردازنده‌ها حین اجرای برنامه‌ها

سیده عفیفه علوی[۱]، مجتبی ولی‌نتاج*[۲]، مجتبی منصوری[۳]
۱- دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی نوشیروانی بابل، بابل، ایران.
*۲- دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی نوشیروانی بابل، بابل، ایران.
۳- دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی نوشیروانی بابل، بابل، ایران.
[1]afifeh.alavi@yahoo.com, [2*]m.valinataj@nit.ac.ir, [3]mansoori@nit.ac.ir

* نشانی نویسنده مسؤول: مجتبی ولی‌نتاج، بابل، خیابان شریعتی، دانشگاه صنعتی نوشیروانی بابل، دانشکده مهندسی برق و کامپیوتر، کد پستی: ۷۱۱۶۷-۴۷۱۴۸

چکیده- سیستم‌های پردازشی در کاربردهایی مانند ماهواره‌ها، فضاپیماها و رآکتورهای هسته‌ای به علت وجود انواع اشعه، بسیار مستعد تولید خروجی‌های نادرست هستند. در این کاربردها استفاده از تجهیزات الکترونیکی مقاوم هزینه زیادی را تحمیل می‌کند. یک راه برای کاهش هزینه، استفاده از نرم‌افزارهای مقاوم یا بهبودیافته بر روی تجهیزات رایج مانند پردازنده‌های عام است. در این مقاله، روشی جدید برای بهبود اجرای انواع کدها بر روی پردازنده‌ها به منظور تشخیص خطاهای گذرا که در بستر سخت‌افزاری رخ می‌دهد، ارائه می‌گردد. این روش جزء روش‌هایی است که با شناسایی خطاهای کنترلی یا همان تغییر در روند اجرای کد برنامه، خطاهای گذرا را تشخیص می‌دهند. روش پیشنهادی مبتنی بر تخصیص امضاها و استفاده از متغیرهای محاسبه شونده حین اجرا بوده و با افزودن دو امضا و حداکثر سه دستورالعمل کنترلی به هر بلوک پایه بسیاری از خطاهای کنترلی را تشخیص می‌دهد. نتایج شبیه‌سازی و پیاده‌سازی روش پیشنهادی به همراه روش‌های پیشین نشان می‌دهد که روش پیشنهادی با توجه به سه پارامتر میزان پوشش خطا، سربار کارایی و سربار حافظه، بهتر از روش‌های قبلی شناسایی خطاهای کنترلی را انجام می‌دهد.

واژه‌های کلیدی: خطاهای گذرا، تشخیص خطا، تحمل‌پذیری اِشکال سخت‌افزاری با پیاده‌سازی نرم‌افزاری (SIHFT)، وارسی جریان کنترلی (CFC)، وارسی جریان داده‌ای (DFC).

۱- مقدمه

در سیستم‌های پردازش‌گر امروزی احتمال رخداد خطاهای گذرا به علت تنوع کاربردها و محیط‌های استفاده گوناگون و وجود عوامل محیطی متنوع افزایش یافته است. این ویژگی خصوصاً برای سیستم‌هایی که در تأسیسات هسته‌ای یا ماهواره‌ها مورد استفاده قرار می‌گیرند، به علت وجود انواع اشعه و ذرات باردار پرانرژی مانند آلفا و بتا بسیار آشکار است. یکی از راه‌های مقاوم سازی این سیستم‌ها تشخیص خطاهای گذرا و سپس انجام عملی متناسب با خطای رخداده است. برای تشخیص خطاهای گذرا می‌توان از روش‌های سخت‌افزاری یا نرم‌افزاری استفاده نمود. روش‌های سخت-افزاری دارای پوشش خطای بالاتری هستند، ولی هزینه نسبتاً زیادی را به سیستم تحمیل می‌کنند [۱]. استفاده از قطعات مقاوم در برابر تشعشعات نیز از روش‌های معمول سخت‌افزاری است که به دلیل کاهش کارایی، عدم امکان استفاده از قطعات مرسوم و افزایش هزینه و وزن، خیلی مطلوب نیست [۱]. روش‌های نرم‌افزاری، پوشش

خطای کمتر و تأخیر اجرای بیشتری دارند، اما به هزینه کمتری احتیاج دارند. علاوه بر این، روش‌های نرم‌افزاری به سخت‌افزار پیچیده‌ای نیاز ندارند و مستقل از نوع پردازنده، قابل اعمال بر روی انواع تجهیزات و پردازنده‌های رایج (COTS[1]) هستند که باعث کاهش هزینه و انعطاف‌پذیری بالای این روش‌ها شده است [۲].

روش‌های نرم‌افزاری مورد بحث در این مقاله برای تشخیص خطاهای گذرا، روش‌های مبتنی بر تحمل‌پذیری اشکال‌های سخت‌افزاری اما با پیاده‌سازی نرم‌افزاری و در سطح کدهای برنامه هستند که به آن‌ها SIHFT[2] گفته می‌شود. در یکی از مجموعه روش‌های مبتنی بر SIHFT، تشخیص خطا با افزودن یک سری کدهای کنترلی به برنامه‌ها و قبول سربارهای حافظه و کارایی انجام می‌شود. این روش‌ها خود به دو دسته تقسیم می‌شوند. در دسته اول، شناسایی خطاهای مؤثر بر جریان داده (DFE)[3] (مانند تغییر در مقدار ثبات‌ها و متغیرها) با وارسی جریان داده‌ای (DFC)[4] و در دسته دوم، شناسایی خطاهای مؤثر بر جریان کنترل (CFE)[5] که تغییر دهنده روند اجرای برنامه هستند، با وارسی جریان کنترلی (CFC)[6] انجام می‌شود. به طور تجربی نشان داده شده است که ۳۳ تا ۷۷ درصد از اشکال‌های گذرا منجر به خطاهای از نوع CFE و بقیه منجر به خطاهای از نوع DFE می‌شوند [۳]. بنابراین، روش‌های نرم‌افزاری خصوصاً روش‌های مبتنی بر وارسی جریان کنترلی می‌توانند در کاهش هزینه تشخیص خطاها مؤثر باشند [۴]. یک سری روش‌های ترکیبی مانند روش‌های ارائه شده در مراجع [۵،۶] نیز وجود دارند که با استفاده همزمان از روش‌های نرم‌افزاری و سخت‌افزاری سعی در تشخیص خطاهای گذرا دارند. به عنوان نمونه در مرجع [۶]، وارسی جریان کنترلی هم با تغییر در کدهای برنامه و هم با استفاده از یک وارسی کننده سخت‌افزاری انجام شده است.

روش‌های مبتنی بر SIHFT با توجه به مستقل بودن از نوع بستر پردازشی، قابلیت اجرا بر روی انواع پردازنده‌های عام را دارند [۷]. از طرف دیگر، این پردازنده‌ها لازم نیست که دارای ویژگی تحمل‌پذیری اشکال باشند. بنابراین، این برنامه‌ها دارای انعطاف‌پذیری مناسب و قابلیت حمل خواهند بود. با این حال، سربار این روش‌ها هنوز بالاست و نیاز به روش‌هایی با هزینه کمتر وجود دارد. در این مقاله، با توجه به روش‌های موجود مبتنی بر وارسی جریان کنترلی، روشی جدید برای تشخیص خطاهای مؤثر بر جریان کنترلی یا روند اجرا ارائه می‌گردد که با افزودن یک سری کدهای کنترلی خاص در سطح کدهای برنامه همراه با امضا[7]های لازم، به پوشش خطایی بیش از روش‌های پیشین می‌رسد. علاوه بر این، روش پیشنهادی نسبت به روش‌های پیشین به سربارهای تأخیر و حافظه کمتری احتیاج دارد.

بخش‌های بعدی مقاله به صورت زیر سازماندهی شده‌اند؛ در بخش دوم یک پیش‌زمینه از مفاهیم مورد نیاز ارائه شده و در بخش سوم، تحقیقات مرتبط مورد بررسی قرار می‌گیرند. در ادامه در بخش چهارم، روش پیشنهادی برای تشخیص خطاهای گذرا که از نوع CFC است، شرح داده می‌شود. در بخش پنجم، نتایج پیاده‌سازی و شبیه‌سازی روش پیشنهادی به همراه روش‌های مرتبط قبلی از نظر میزان پوشش خطا و سربارهای کارایی و حافظه ارائه و ارزیابی شده و در بخش ششم، نتیجه‌گیری نهایی مقاله ارائه می‌گردد.

۲- مفاهیم پایه

با توجه به این که کدام بخش از سیستم پردازشی اجرا کننده یک برنامه به علت رخداد یک خطای گذرا دچار تغییر شده است، می‌توان آن را در دو گروه DFE و CFE قرار داد [۳]. اگر در اثر خطای گذرا قسمتی از ساختمان داده پردازنده مانند متغیرها و ثبات‌ها یا حافظه داده تغییر کند، خطای جریان داده‌ای (DFE) رخ داده است. این نوع خطا در پردازنده ممکن است به محاسبه نتیجه نادرست منجر شود، اما در جریان برنامه تغییری ایجاد نمی‌کند. اما اگر اثر خطای گذرا بر روی ساختمان داده پردازنده یا حافظه‌های داده و دستورالعمل به گونه‌ای باشد که اجرای عادی یک برنامه را تحت تأثیر قرار دهد، خطای جریان کنترلی (CFE) رخ داده است و اثر منفی آن بیش از DFE است [۵]. این نوع خطا می‌تواند باعث انحراف در روند اجرایی برنامه شود یا حتی یک حلقه بی‌نهایت در بخشی از برنامه و حتی در یک دستورالعمل ایجاد کند. اگر بخواهیم از روشی پایه مانند دونسخه‌سازی و مقایسه[8] برای تشخیص این نوع خطاهای گذرا استفاده کنیم، حداقل ٪۱۰۰ سربار هم در تأخیر و هم در حافظه مصرفی به سیستم تحمیل می‌شود. بنابراین، از روش‌های DFC و CFC برای تشخیص این نوع خطاها استفاده می‌شود. اما روش‌های مبتنی بر SIHFT معمولاً با هدف شناسایی تنها یکی از این دو نوع خطا (DFE یا CFE) پیاده‌سازی می‌شوند [۷].

روش‌های CFC معمولاً با تجزیه و تحلیل کد و روند اجرای برنامه، آن را به تعدادی بلوک پایه (BB[9]) تقسیم نموده و جریان برنامه را مانند یک گراف و حرکت بین رؤوس این گراف دنبال می‌کنند [۸]. در شکل ۱ کد الگوریتم مرتب‌سازی حبابی (BS[10]) و گراف جریان کنترلی آن به همراه بلوک‌های پایه نشان داده شده است. هر بلوک پایه مجموعه‌ای از بیشترین دستورالعمل‌های پشت‌سرهم است که تنها دستورالعمل انتهایی آن ممکن است از نوع پرش شرطی[11] یا

غیر شرطی[۱۲] یا فراخوانی[۱۳] تابع باشد. در واقع، دستورالعمل‌های پرش، بلوک‌های پایه را از هم جدا می‌کنند. همچنین، هیچ دستورالعملی در بلوک پایه غیر از اولین دستور نمی‌تواند مقصد یک پرش باشد [۹].

در روش‌های CFC، تعدادی دستورالعمل افزونه و امضا به هر بلوک پایه برنامه تخصیص داده می‌شود. این روش‌ها اکثراً بر قانون پایش امضا بنا نهاده شده‌اند. در این روش‌ها، شمایی از جریان اجرای برنامه استخراج شده و امضاهایی که نمایش دهنده جریان درست اجرای برنامه هستند، قبل از اجرای برنامه به آن نسبت داده می‌شوند. در زمان اجرا، جریان برنامه توسط این امضاها کنترل می‌گردد تا روند اجرا در نقاط صحیح به بلوک‌های برنامه وارد و از آن‌ها خارج گردد. تفاوت روش‌های مختلف CFC در نحوه محاسبه امضاها و چگونگی بررسی آن‌ها است.

خطاهایی که در این روش‌ها باید مورد بررسی قرار گیرند به سه دسته کلی زیر تقسیم می‌شوند [۸،۱۰،۱۱]:

۱- پرش‌های غیرمجاز رخ داده در داخل یک بلوک پایه
۲- پرش‌های غیرمجاز رخ داده بین دو بلوک پایه
۳- پرش‌های غیرمجاز رخ داده از یک بلوک پایه به فضای استفاده نشده از حافظه

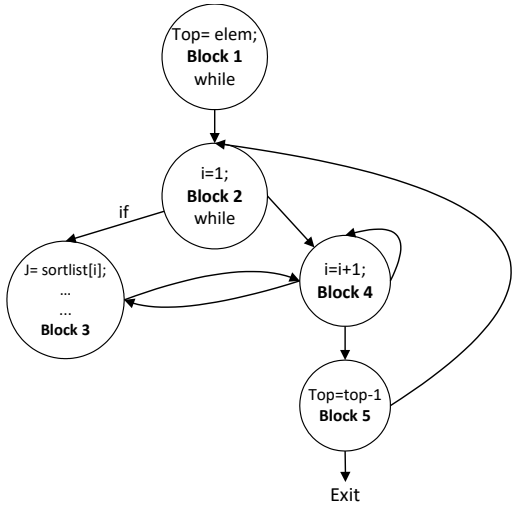
این پرش‌های غیرمجاز منجر به خطاهای از نوع CFE می‌شوند که می‌توان آن‌ها را به شش نوع مختلف به شرح زیر تقسیم نمود [۱۱]:

۱- نوع اول: خطای تولید شـــده به علت پرش غیرمجاز از انتهای یک بلوک پایه به ابتدای بلوک پایه دیگر
۲- نوع دوم: خطای تولید شده به علت پرش مجاز اما نادرست از انتهای یک بلوک پایه به ابتدای بلوک پایه دیگر
۳- نوع سوم: خطای تولید شده به علت پرش از انتهای یک بلوک پایه به درون بلوک پایه دیگر
۴- نوع چهارم: خطای تولید شده به علت پرش از درون یک بلوک پایه به درون بلوک پایه دیگر
۵- نوع پنجم: خطای تولید شـــده به علت پرش از نقطه‌ای از یک بلوک پایه به نقطه دیگری از همان بلوک پایه
۶- نوع شـــشـم: خطای تولید شـده به علت پرش از هر نقطه‌ای از یک بلوک پایه به فضـــای اســـتفاده نشـــده حافظه یا بیرون از بلوک‌های پایه

این خطاها حتی با تغییر یک بیت از حافظه داده یا حافظه دستور پردازنده ممکن است رخ دهند. روش‌های CFC بایستی قابلیت تشخیص حداقل بعضی از انواع خطاهای ذکر شده را داشته باشند.

```
1     top = elem;
      While(top > 1){
2       i = 1;
        While(i < top){
          if(sortlist[i] > sortlist[i+1]){
3           j = sortlist[i];
            Sortlist[i] = sortlist[i+1];
            Sortlist[i+1] = j;
          }
4         i = i+1;
        }
5       top = top-1;
      }
```

(الف)



(ب)

شکل ۱: الف) کد الگوریتم مرتب‌سازی حبابی به همراه شماره بلوک‌های پایه، ب) گراف جریان کنترلی معادل [۹].

۳- تحقیقات مرتبط

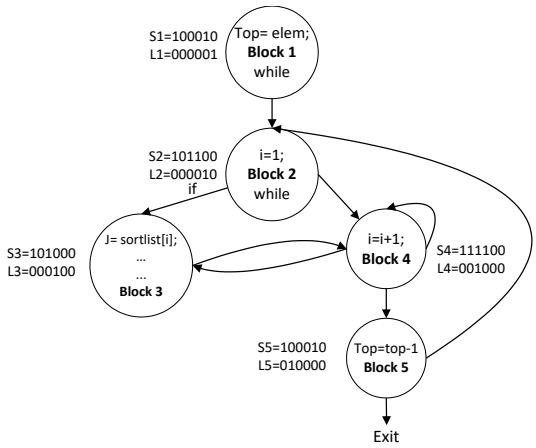
تاکنون روش‌های زیادی مبتنی بر CFC برای تشخیص نرم‌افزاری خطاهای گذرا ارائه شده‌اند. اما در این بخش، مهمترین روش‌ها بررسی خواهند شد. اولین روش درخور توجه روش ECCA[۱۴] [۱۰] است که در آن پس از تقسیم برنامه به بلوک‌های پایه، به هر بلوک پایه یک شناسه (BID[۱۵]) بزرگتر از دو که عددی اول است، تخصیص داده می‌شود. در زمان اجرای برنامه، یک متغیر سراسری که عددی صحیح است، هنگام ورود و خروج به هر بلوک پایه به‌روزرسانی می‌شود. در واقع، به هر بلوک پایه دو خط کد نیز اضافه می‌گردد. در این روش که از عملیات ضرب و تقسیم برای محاسبات لازم استفاده می‌شود، خطاهای منفرد CFE از نوع اول، سوم و چهارم قابل تشخیص است؛ اما این روش در تشخیص بقیه انواع خطاها ناتوان است.

روش بعدی با نام CFCSS[۱۶] [۸] از تخصیص و مقایسه امضاها استفاده می‌کند. در این روش، در زمان کامپایل به هر بلوک پایه

امضای اختصاصی باینری S_i با طول مشخص داده می‌شود. سپس، حین اجرای برنامه با استفاده از یک امضای دیگر به نام d_i و محاسبه متغیر سراسری G (با مقدار اولیه S_0) از روی آن با عملگر XOR که در هر بلوک پایه محاسبه می‌گردد، مقداری به‌دست می‌آید که باید با امضای اختصاصی S_i آن بلوک پایه برابر باشد. در غیر این صورت، خطا اعلام می‌شود. این روش بیشتر خطاهای منفرد CFE از نوع اول، سوم و چهارم را تشخیص می‌دهد. با این حال، این روش توانایی تشخیص خطاهای نوع چهارمی را که به خاطر پرش اشتباه از درون یک بلوک پایه به ابتدای یکی از پیش‌رو[۱۷]های همان بلوک (یعنی بلوک‌هایی که احتمالاً پس از اجرای بلوک فعلی اجرا خواهند شد) رخ داده است، ندارد.

روش RSCFC[۱۸] [۹،۱۲] که بهبود یافته آن در مرجع [۱۳] ارائه شده از تخصیص و محاسبه امضا برای هر بلوک پایه با توجه به ارتباط میان بلوک‌های پایه استفاده می‌کند. در شکل ۲، گراف جریان کنترلی الگوریتم مرتب‌سازی حبابی که قبلاً در شکل ۱ نشان داده شده بود، همراه با امضاها و شماره بلوک‌های تخصیص یافته به بلوک‌های پایه نشان داده شده است. در این شکل، Li شماره بلوک پایه i ام و Si امضای تخصیص یافته به آن است که در زمان اجرا دوباره محاسبه می‌شود. بیت پرارزش‌تر Si هنگام تخصیص اولیه برابر '۱' بوده و بقیه مکان‌های بیتی آن نشان دهنده بلوک‌هایی است که پیش‌رو بلوک پایه i ام محسوب می‌شوند. طول Li و Si برحسب بیت یک واحد بیشتر از تعداد کل بلوک‌های پایه برنامه یا همان تعداد رؤوس گراف جریان کنترلی است. علاوه بر امضاهای بین بلوکی[۱۹]، از یک امضای درون بلوکی[۲۰] نیز استفاده می‌شود. در این روش، چند دستور در ابتدا و چند دستور در انتهای هر بلوک پایه قرار داده می‌شود تا بررسی‌ها برای تشخیص خطا حین اجرا را انجام دهند. پوشش خطای این روش نسبت به روش‌های قبلی بیشتر است اما به علت استفاده از تعدادی بیشتر دستورالعمل کنترلی، سربارهای حافظه و کارایی بیشتری تحمیل می‌کند.

روش درخور توجه بعدی CFCCB[۲۱] [۱۴] نام دارد که در آن بر اساس دسته‌بندی بلوک‌های پایه، به آن‌ها امضا تعلق می‌گیرد. در حالت ساده، تعداد پیش‌روهای بلوک‌های پایه به عنوان مبنایی برای دسته‌بندی آن‌ها است که اگر یکی بود، بلوک پایه از دسته شماره یک و در غیر این صورت از دسته شماره دو محسوب می‌شود. با این حال، به خاطر اشتراک میان پیش‌روهای دو بلوک پایه در بعضی از مواقع، این دسته‌بندی کامل نیست. به همین دلیل در این روش، برای این موارد از مفهوم گراف ناسازگاری[۲۲] که از روی گراف اولیه به‌دست می‌آید استفاده می‌شود.



شکل ۲: گراف جریان کنترلی الگوریتم مرتب‌سازی حبابی شکل ۱ همراه با امضاهای تخصیص یافته به بلوک‌های پایه [۱۲].

یکی از بهترین روش‌های ارائه شده تاکنون SCFC[۲۳] [۱۱] نام دارد که تکمیل شده روش بیان شده در مرجع [۱۵] است. در این روش همانند بسیای از روش‌های قبلی از تخصیص امضا و شناسه به بلوک‌های پایه استفاده می‌شود. در روش SCFC برای تشخیص خطاهای از نوع CFE، به هر بلوک پایه چهار دستورالعمل کنترلی اضافه می‌شود. اولین دستور که به ابتدای بلوک پایه افزوده می‌شود *control* نام دارد و ورود درست یا نادرست اجرا به بلوک پایه را با مقایسه شناسه از قبل محاسبه شده با شناسه واقعی آن بلوک تعیین می‌کند. دستورالعمل دوم به نام *check* تعیین کننده درستی یا نادرستی بلوک پایه فعلی به عنوان مقصد بلوک پایه قبلی است. این دستور به همراه دستورالعمل سوم به نام *update* در وسط بلوک‌های پایه قرار می‌گیرد و به همراه آن برای تشخیص بعضی از پرش‌های غیرمجاز درون بلوکی و بین بلوکی استفاده می‌شود. دستورالعمل چهارم با نام *exit* در انتهای هر بلوک پایه قرار داده می‌شود و برای به‌روز رسانی شناسه حین اجرا با مقدار شناسه واقعی بلوک فعلی به‌کار می‌رود. این روش، خطاهای منفرد CFE از نوع اول و درصدی از انواع سوم، چهارم و پنجم را تشخیص می‌دهد، به همین دلیل دارای پوشش خطای بیشتری نسبت به روش‌های قبلی است.

در مرجع [۱۶] روشی با تمرکز بر مدیریت پرش‌های غیر شرطی غیر مستقیم و فراخوانی توابع ارائه گردیده که در سطح کدهای اسمبلی، بدون استفاده از تخصیص امضا به بلوک‌های پایه، تغییرات لازم را به برنامه اعمال می‌کند. در مرجع [۱۷] روشی پیشنهاد شده که با دادن امضاهای توالی پرش‌ها مشابه روش‌های CFC قبلی عمل می‌کند. تفاوت این روش با روش‌های قبلی به خوبی توضیح داده نشده و ایراد آن درجه پوشش خطای آن (حدود ٪۹۰) است که نسبت به بعضی از روش‌های قبلی کمتر است. روش پیشنهادی در مرجع [۱۸] با تفاوت قائل شدن میان انواع بلوک‌های پایه عمل

نموده و به بهبود کارایی نسبت به روش CFCSS [۸] رسیده است. در این مرجع مقایسه فقط با روش CFCSS و تنها از نظر سربار کارایی انجام شده است که نشان دهنده بهبود این پارامتر نسبت به روش CFCSS است.

در مرجع [۱۹] ابتدا روش‌های اصلی نرم‌افزاری مبتنی بر CFC مورد بررسی مجدد قرار گرفته و سپس روشی با نام RASM[۲۴] معرفی شده است که در آن با توجه به روش‌های قبلی، ملزومات کلی این نوع روش‌ها رعایت شده و امضاها و کدهای کنترلی مناسب در سطح زبان اسمبلی انتخاب و اعمال می‌شود. این روش توانسته به پوشش خطایی حدود ٪۹۸ برسد. در مرجع [۲۰] روشی سخت‌افزاری برای وارسی جریان کنترلی از دیدگاه مبارزه با حملات و نفوذهای غیرمجاز برای تغییر حافظه ارائه شده است. این روش، هم بر روی FPGA[۲۵] و هم به صورت ASIC[۲۶] طراحی شده و مورد ارزیابی قرار گرفته است.

۴- روش پیشنهادی

در این بخش روشی جدید به نام OSCFC[۲۷] ارائه می‌شود که با افزودن دو امضا و حداکثر سه دستورالعمل کنترلی به هر بلوک پایه نسبت به روش‌های مشابه قبلی بهتر عمل کرده و بسیاری از خطاهای از نوع CFE را تشخیص می‌دهد.

۴-۱- معرفی روش OSCFC

روش پیشنهادی OSCFC بر روی گراف جریان کنترلی برنامه مانند نمونه نشان داده شده در شکل ۲ که از روی بلوک‌های پایه ساخته می‌شود، عمل می‌کند. برای هر رأس V_i در این گراف، مجموعه $succ(V_i)$ به عنوان مجموعه رؤوسی که از رأس V_i قابل دسترسی هستند (بلوک‌های پیش‌رو بلوک i ام) و مجموعه $pred(V_i)$ به عنوان مجموعه رؤوسی که به رأس V_i دسترسی دارند (بلوک‌های پس‌رو بلوک i ام) در نظر گرفته می‌شوند. تخصیص امضاها در روش پیشنهادی برای وارسی جریان کنترلی مشابه روش RSCFC است (شکل ۲). امضای Li حاوی شماره بلوک i ام (یا همان رأس V_i) است. در واقع، تمام بیت‌های Li غیر از بیت شماره i ام به صفر مقداردهی می‌شوند. اما مکان‌های بیتی امضای Si با توجه به بلوک‌های پیش‌رو بلوک i ام مقدار '۰' یا '۱' می‌گیرند. به عنوان مثال در شکل ۲، به خاطر این که بلوک‌های پیش‌رو بلوک ۲ بلوک‌های ۳ و ۴ هستند، بیت‌های سوم و چهارم امضای S2 برابر با '۱' شده و بقیه بیت‌ها غیر از بیت آخر که از ابتدا '۱' شده است، برابر با '۰' می‌شوند (S2=101100). همان طور که در بخش سوم بیان شد، طول Si و Li برحسب بیت یک واحد بیشتر از تعداد کل بلوک‌های پایه برنامه است. علاوه بر این، در این روش از دو متغیر سراسری S و L با همان تعداد بیت Si و Li استفاده می‌شود که حین اجرای برنامه مقدار می‌گیرند.

در روش پیشنهادی OSCFC برای وارسی جریان کنترلی، بلوک‌های پایه به دو دسته تقسیم می‌شوند. دسته اول بلوک‌های پایه‌ای هستند که تنها شامل یک دستورالعمل غیر پرش بوده و دسته دوم بلوک‌های پایه‌ای هستند که حداقل شامل دو دستورالعمل غیر پرش هستند. برای کنترل روند اجرا، به بلوک‌های پایه دسته اول دو دستورالعمل کنترلی شامل دستور *test* در ابتدای بلوک و دستور *set* در انتهای بلوک اضافه می‌شود. دستور *test* برای شناسایی خطا و دستور *set* برای تغییر مقدار متغیر سراسری S به امضای اختصاصی Si مربوط به بلوک جاری استفاده می‌شود.

با الهام از روش‌های I2BCFC [۱۵] و SCFC [۱۱]، برای کشف خطاهای درون بلوکی در بلوک‌های پایه دسته دوم، علاوه بر دستورهای کنترلی *test* و *set* یک دستور با نام *update* در وسط این بلوک‌های پایه اضافه می‌شود. این دستور درست در وسط بلوک پایه قرار گرفته و مقدار متغیر سراسری L را تغییر می‌دهد. دستور *test* در بلوک‌های پایه دسته دوم مشابه دسته اول است؛ اما دستور *set* علاوه بر مقایسه مقادیر S و Li، مقدار L را نیز با Li مقایسه می‌کند.

جدول ۱: دستورالعمل‌های کنترلی روش پیشنهادی **OSCFC** با توجه به نوع بلوک پایه و مکان قرار گرفتن آن‌ها

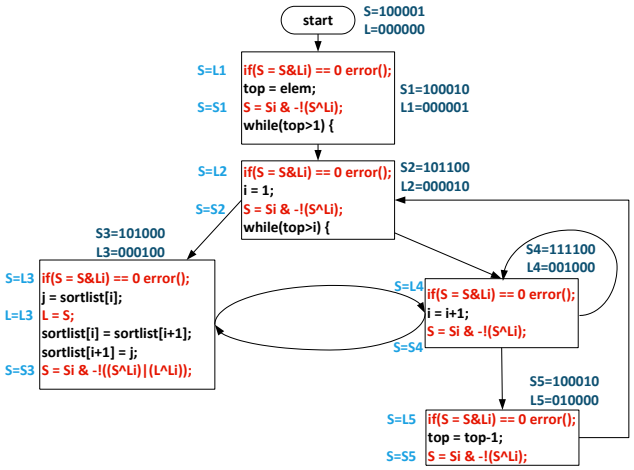
دستورالعمل‌های کنترلی	مکان و علت افزودن	بلوک‌های پایه دسته اول	بلوک‌های پایه دسته دوم
Test	ابتدای بلوک پایه، بررسی صحت پرش‌ها	If(S = S&Li) == 0 error();	If(S = S&Li) == 0 error();
Update	وسط بلوک، تغییر مقدار L برای تشخیص خطاهای درون بلوکی	—	L = S;
Set	انتهای بلوک پایه، مقداردهی S با مقدار Si مربوط به بلوک جاری	S = Si & -!(S^Li);	S = Si & -!((S^Li)|(L^Li));

جدول ۱ دستورهای کنترلی مورد استفاده در روش پیشنهادی OSCFC و محل قرار گرفتن آن‌ها را در دو نوع بلوک پایه نشان می‌دهد.

با توجه به جدول ۱، دستور *test* که در ابتدای هر بلوک پایه قرار داده می‌شود، در حالت عدم وجود خطا، Li را به متغیر سراسری S منتسب می‌کند. باید توجه داشت که مقدار اولیه S برابر با 10…01 بوده تا در شروع کار، فقط ورود به بلوک پایه ۱ مجاز باشد. بنابراین در حالت عدم وجود خطا، S&Li برابر با Li و مقداری غیر صفر شده و به S داده می‌شود. دستور *set* در انتهای هر بلوک پایه در حالت عدم وجود خطا Si را با S به‌روز رسانی می‌کند، زیرا به عنوان مثال، برای بلوک‌های پایه دسته اول با دستور *test* مقدار S با Li برابر شده و عملیات XOR میان S و Li در دستور *set* مقدار صفر را تولید کرده و (0)!- برابر با -1 یا 1…1 می‌شود که پس از AND با Si همان Si را تولید می‌کند. پس از خروج از بلوک فعلی، اگر ورود به بلوک بعدی اشتباه باشد، یعنی بلوک بعدی جزء پیش‌روهای بلوک فعلی ((Succ(Vi)) نباشد، دستور *test* در ابتدای بلوک بعدی منجر به صفر شدن S شده و خطا تشخیص داده می‌شود. البته در حالتی که در بلوک فعلی به علت رخداد یک خطا S با Li برابر نشود، اجرای دستور *set* منجر به صفر شدن S می‌گردد که پس از آن با ورود به هر بلوک دیگر، خطا به کمک دستور *test* تشخیص داده می‌شود.

روش عمل برای بلوک‌های پایه دسته دوم مشابه بلوک‌های پایه دسته اول است با این تفاوت که برای تشخیص خطاهای درون بلوکی، در وسط بلوک پایه، متغیر سراسری L با مقدار S به‌روز رسانی می‌شود. مقدار اولیه L در شروع کار برابر با 0…0 است. همچنین، در انتهای بلوک، دستور *set* برابری L و Li را نیز بررسی می‌کند که به عنوان یک شرط لازم برای به‌روز رسانی S با Si است.

به عنوان نمونه، گراف جریان کنترلی الگوریتم مرتب سازی حبابی همراه با کدهای تغییر یافته این برنامه مطابق با روش پیشنهادی OSCFC در شکل ۳ نشان داده شده است. در این شکل، خطوط قرمزرنگ دستورهای کنترلی و خطوط سیاه‌رنگ کدهای اصلی برنامه است. ضمناً، انتساب‌های بیرون کادرها در سمت چپ دستورهای کنترلی، مقادیر منتسب به متغیرهای سراسری S و L را در حالت عدم وجود خطا نشان می‌دهند. تفاوت اصلی روش پیشنهادی OSCFC با روش RSCFC [۱۲] در عدم استفاده از امضای انباشته محلی است که در روش RSCFC در هر بلوک پایه چهار دستورالعمل کنترلی صرف مقداردهی به آن می‌شود. اما تفاوت اصلی روش پیشنهادی با روش SCFC [۱۱] تفاوت قائل شدن میان انواع بلوک‌های پایه و استفاده از فقط یک دستورالعمل کنترلی در وسط بلوک‌های پایه‌ای است که شامل حداقل دو دستور غیر پرش هستند. علاوه بر موارد ذکر شده، محتویات دستورهای کنترلی روش پیشنهادی با روش‌های RSCFC و SCFC متفاوت است.



شکل ۳: گراف جریان کنترلی الگوریتم مرتب‌سازی حبابی همراه با کدهای تغییر یافته بلوک‌های پایه آن طبق روش پیشنهادی OSCFC.

۴-۲- مکانیزم کشف خطا در روش OSCFC

در این بخش، نحوه تشخیص انواع خطاهای منفرد CFE با استفاده از روش پیشنهادی بررسی می‌گردد.

۱) خطای نوع اول: پرش غیرمجاز از انتهای بلوک پایه V_i به ابتدای بلوک پایه V_j

در این حالت متغیر سراسری S به خاطر اجرای دستور *set* در انتهای بلوک V_i با مقدار Si وارد بلوک V_j می‌شود که در نتیجه با توجه به این که V_j جزء بلوک‌های پیش‌رو V_i نیست یعنی $V_j \notin succ(V_i)$، دستور *test* در ابتدای بلوک V_j مقدار S را صفر کرده و اعلام تشخیص خطا با اجرای روال مربوطه انجام می‌شود. به عنوان مثال در شکل ۳، اگر چنین پرشی از بلوک ۱ به بلوک ۳ انجام شود، محاسبات تشخیص خطا به صورت زیر انجام خواهد شد:

S = S1 = 100010; //*set* in Block V_1
S = S & L3 =100010 & 000100 = 0; //*test* in Block V_3

۲) خطای نوع سوم: پرش از انتهای بلوک پایه V_i به درون بلوک پایه V_j

در این حالت V_j چه جزء بلوک‌های پیش‌رو V_i باشد و چه نباشد، به دلیل اجرا نشدن دستور *test* در ابتدای بلوک Vj، مقدار S به‌روز نمی‌شود و همان مقدار Si را خواهد داشت. بنابراین، دستور *set* مقدار S را صفر کرده و در صورت ورود به بلوک بعدی، خطا در ابتدای آن بلوک تشخیص داده می‌شود. به عنوان مثال در شکل ۳،

اگر چنین پرشی از بلوک ۱ به بلوک ۲ انجام شود، محاسبات زیر انجام خواهد شد:

```
S = S1 = 100010;                                    //set in Block V1
S = S2 & -!(S^L2) = 101100 & -!(100010 ^ 000010)
  = 101100 & -!(100000) = 101100 & 000000 = 0;//set in Block V2
```

در بلوک‌های پایه با حداقل دو دستورالعمل هم، به دلیل مخالف صفر شدن نتیجه عملیات XOR میان S و Lj (S^Lj) در دستور *set*، مقدار S در انتهای بلوک V_j صفر شده و خطا در ابتدای بلوک پایه بعدی تشخیص داده می‌شود.

۳) خطای نوع چهارم: پرش از درون بلوک پایه V_i به درون بلوک پایه V_j

حالات مختلف این نوع خطا در شکل ۴ نمایش داده شده است. در این حالات به علت عدم اجرای دستور *set* در بلوک V_i، مقدار S حین ورود به درون بلوک V_j برابر Li بوده و با اجرای دستور *set* در انتهای بلوک V_j مقدار S صفر خواهد شد. در نتیجه در صورت ورود به بلوک بعدی، خطا در ابتدای آن بلوک تشخیص داده می‌شود. باید توجه داشت که پرش به درون بلوک V_j چه قبل از دستور *update* انجام شود و چه پس از آن و قبل از دستور *set*، در تشخیص خطا تفاوتی نخواهد داشت. به عنوان مثال در شکل ۳، اگر چنین پرشی از درون بلوک ۲ به درون بلوک ۳ و قبل از دستور *update* انجام شود، در این صورت خواهیم داشت:

```
S = L2 = 000010;                                    //test in Block V2
L = S = 000010;                                     //update in Block V3
S = S3 & -!((S^L3) | (L^L3))
  = 101000 & -!((000010 ^ 000100) | (000010 ^ 000100))
  = 101000 & -!(000110 | 000110)
  = 101000 & -!(000110) = 101000 & 000000 = 0;//set in Block V3
```

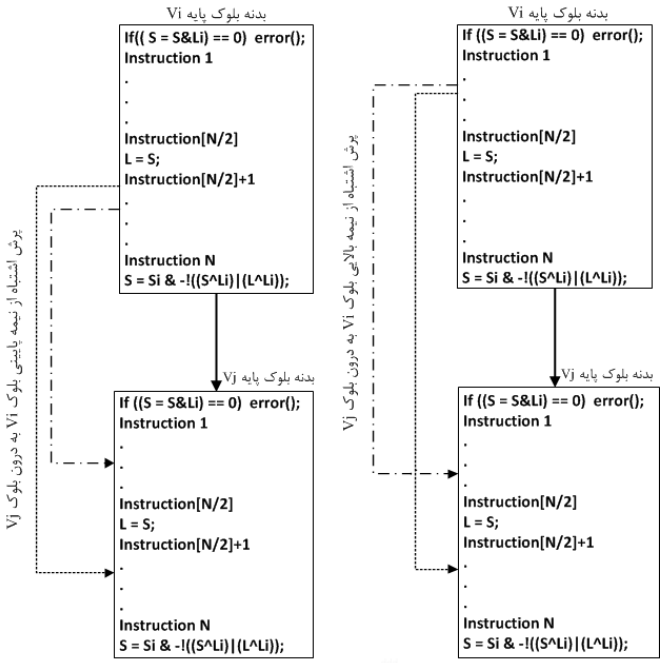
همچنین، اگر پرش از درون بلوک V_i به ابتدای بلوک V_j انجام شود، خطا به راحتی در بلوک V_j تشخیص داده می‌شود، زیرا مقدار S قبل از ورود به بلوک V_j برابر با Li است و پس از اجرای دستور *test* در ابتدای بلوک V_j مقدار آن صفر می‌شود:

```
S = S & Lj = Li & Lj = 0; //test in Block Vj
```

لازم به ذکر است که V_j چه جزء بلوک‌های پیش‌رو V_i باشد و چه نباشد، تشخیص خطای نوع چهارم قابل انجام است.

۴) خطای نوع پنجم: پرش از نقطه‌ای از بلوک پایه V_i به نقطه دیگری از همان بلوک پایه

این نوع خطا مربوط به بلوک‌های پایه‌ای است که دارای حداقل دو دستورالعمل هستند. بنابراین، دستور کنترلی *update* که مطابق جدول ۱ در وسط این بلوک‌ها برای به‌روزرسانی متغیر سراسری L قرار داده می‌شود، به تشخیص این نوع خطا کمک می‌کند. اگر خطای رخداده به نحوی باشد که روند اجرای برنامه از قبل از دستور *update* به بعد از آن منتقل شود، به دلیل اجرا نشدن این دستور، دستور *set* در انتهای بلوک مقدار صفر را به S منتسب می‌کند که باعث تشخیص خطا در ابتدای بلوک بعدی خواهد شد.



شکل ۴: حالات مختلف پرش اشتباه از درون بلوک پایه V_i به درون بلوک پایه V_j که با روش پیشنهادی قابل تشخیص است.

به عنوان مثال در شکل ۳، اگر چنین پرشی در درون بلوک ۳ قبل از دستور *update* به پس از دستور *update* انجام شود، در این صورت خواهیم داشت:

```
L = 000000;              //initial value before coming to Block V3
S = L3 = 000100;                                    //test in Block V3
S = S3 & -!((S^L3) | (L^L3))
  = 101000 & -!((000100 ^ 000100) | (000000 ^ 000100))
  = 101000 & -!(000000 | 000100)
  = 101000 & -!(000100) = 101000 & 000000 = 0;//set in Block V3
```

باید توجه داشت که با این روش بعضی از خطاهای نوع پنجم قابل شناسایی هستند، زیرا تنها پرش‌های اشتباه از قبل از دستور *update* به پس از دستور *update* تشخیص داده می‌شوند.

با توجه به مطالب بیان شده، روش پیشنهادی همانند روش SCFC [۱۱] خطاهای نوع دوم و ششم را تشخیص نمی‌دهد. همچنین، در بعضی از حالت‌ها تشخیص خطاهای نوع سوم و چهارم منوط به وارد شدن به بلوک سومی است که ممکن است به خاطر رسیدن به آخرین بلوک پایه برنامه هیچ‌گاه رخ ندهد.

۵- نتایج شبیه‌سازی و ارزیابی

در این بخش، روش پیشنهادی OSCFC به همراه پنج روش معتبر قبلی شامل ECCA [۱۰]، CFCSS [۸]، RSCFC [۹،۱۲]، CFCCB [۱۴] و SCFC [۱۱] در بستر شبیه‌سازی یکسان که همگی بر روی چند برنامه از مجموعه محک SPEC2000 پیاده‌سازی می‌گردند، از نظر میزان پوشش خطا (با استفاده از تزریق تصادفی خطا[۲۸]) و سربارهای کارایی و حافظه مورد ارزیابی قرار می‌گیرند. مشابه بعضی از مراجع مانند [۹،۱۱،۱۲]، سه برنامه محک مرتب‌سازی حبابی (BS)، مرتب‌سازی سریع (QS[۲۹]) و ضرب ماتریس ۴۰×۴۰ (MM[۳۰]) برای پیاده‌سازی روش‌های مختلف انتخاب شده است.

۵-۱- تزریق خطا

تزریق خطاهای از نوع CFE در برنامه‌های محک با استفاده از روش نرم‌افزاری تزریق خطاها و مبتنی بر دیباگر GNU انجام شده است. این دیباگر با توجه به ویژگی‌های مناسب، اجرا در بسیاری از سیستم‌های شبه یونیکس و پشتیبانی از اکثر زبان‌های برنامه‌نویسی و پردازنده‌های مختلف، مورد استفاده قرار گرفته است. با استفاده از روش نرم‌افزاری تزریق خطا، در هر اجرای برنامه یک خطا از نوع CFE به طور تصادفی به برنامه اعمال می‌گردد که با توجه به نوع مدل خطا شامل حذف پرش، افزودن پرش و یا تغییر عملوند پرش است. اجرای برنامه حاوی خطا و تعداد تکرار عملیات تزریق خطا برای هر برنامه به گونه‌ای است که تمام مکان‌های با قابلیت تغییر روند اجرای برنامه مورد آزمون قرار می‌گیرند.

تزریق خطاهای از نوع CFE در هر برنامه به صورت زیر انجام می‌شود:

۱) به طور تصادفی یکی از حالات خطای حذف پرش، افزودن پرش و یا تغییر عملوند پرش انتخاب می‌شود.

۲) دستورالعملی که قرار است خطا روی آن تزریق شود، به طور تصادفی انتخاب می‌شود. اگر خطای مورد نظر از نوع حذف پرش یا تغییر عملوند پرش باشد، دستورالعمل انتخابی باید از نوع پرش باشد.

۳) اگر دستورالعمل انتخابی درون یک حلقه قرار داشته باشد، به طور تصادفی تعیین می‌شود که خطا در تکرار چندم حلقه به آن دستورالعمل تزریق گردد.

۴) پس از اجرای برنامه و رسیدن به دستورالعمل انتخاب شده (اگر دستورالعمل انتخابی درون یک حلقه باشد، اجرای برنامه تا شماره تکرار انتخابی ادامه می‌یابد)، متناسب با نوع خطای انتخاب شده تزریق خطا به صورت زیر انجام می‌شود:

الف) حذف پرش: شمارنده برنامه[۳۱] به گونه‌ای تغییر می‌کند که به خط بعدی برود.

ب) افزودن پرش: به طور تصادفی یکی از بیت‌های شمارنده برنامه تغییر داده می‌شود.

ج) تغییر عملوند پرش: به طور تصادفی مقصد دستورالعمل پرش تغییر داده می‌شود.

۵) با ادامه اجرای برنامه، اثر خطا مورد بررسی قرار می‌گیرد. امکان دارد که سخت‌افزار یا سیستم عامل خطا را تشخیص دهند. در غیر این صورت، خروجی برنامه با خروجی برنامه بدون خطا مقایسه می‌شود.

اثر اعمال خطا بر روی یک برنامه را می‌توان به پنج حالت مختلف زیر تقسیم‌بندی نمود [۱۱،۱۲]:

۱) **خروجی درست (CR[۳۲])**: برنامه با وجود خطای تزریق شده به مشکلی برنمی‌خورد و خروجی نهایی برنامه صحیح است.

۲) **تشخیص سیستم عامل (OS[۳۳])**: خطای تزریق شده به برنامه با قابلیت‌های سیستم عامل و اعلام استثنا تشخیص داده می‌شود.

۳) **اتمام مهلت زمانی (TO[۳۴])**: خطای تزریق شده منجر به تشکیل حلقه‌ای با زمان اجرای زیاد شده و با توجه به تجاوز زمان اجرا از مقدار از قبل تعیین شده، تشخیص داده می‌شود.

۴) **تشخیص نرم‌افزاری خطا (SD[۳۵])**: خطای تزریق شده توسط روش اعمال شده روی برنامه با توجه به مکانیزم تشخیص خطا، کشف می‌شود.

۵) **خروجی نادرست (WR[۳۶])**: خطای تزریق شده بدون کشف منجر به تولید خروجی نادرست می‌شود. البته این حالت در شبیه‌سازی‌ها با توجه به مقایسه با خروجی برنامه بدون خطا، تشخیص داده می‌شود.

باید توجه داشت که جمع احتمالات رخداد حالت‌های ذکر شده در بالا غیر از حالت WR، به عنوان درجه پوشش خطای روش تحت بررسی برای برنامه اجرا شده محسوب می‌شود.

۵-۲- ارزیابی نتایج

نتایج حاصل از تزریق خطا در سه برنامه مختلف قبل و پس از اعمال روش‌های گوناگون تشخیص خطاهای کنترلی در جدول ۲ ارائه شده است. در این جدول، درصد پنج حالت مختلف رخ دهنده به تفکیک

ارائه شده و جمع درصدهای هر سطر برابر ۱۰۰ است. در ستون SD بزرگترین درصد برای هر برنامه به صورت ضخیم نشان داده شده است که با توجه به آن، برای هر سه برنامه بیشترین درصد تشخیص نرم‌افزاری خطا مربوط به روش پیشنهادی OSCFC است. علاوه بر این، در این جدول در ستون WR کوچک‌ترین درصد برای هر برنامه به صورت ضخیم نشان داده شده است که با توجه به آن، برای دو برنامه کمترین درصد عدم تشخیص خطا با خروجی نادرست مربوط به روش پیشنهادی OSCFC و در یک برنامه مربوط به روش SCFC [۱۱] بوده است. هرچه درصد SD یک روش بیشتر و درصد WR آن کمتر باشد، درصد پوشش خطای آن بیشتر خواهد بود که با توجه به این قضیه، روش پیشنهادی در وضعیتی مطلوب قرار دارد.

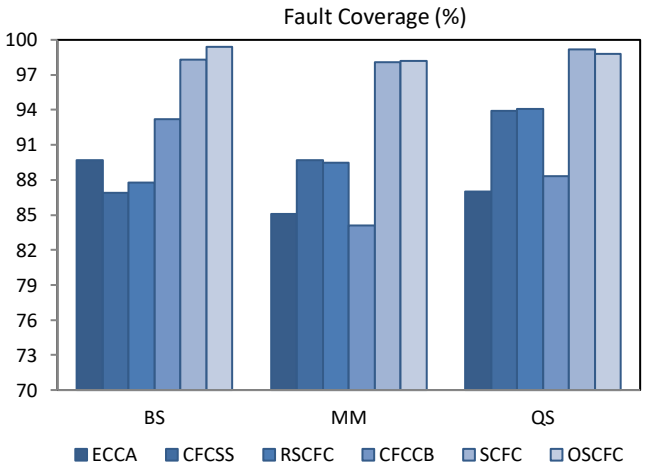
جدول ۲: نتایج حاصل از تزریق خطا در برنامه‌های محک، قبل و پس از تقویت با روش‌های گوناگون تشخیص خطا (همه اعداد به درصد هستند).

روش_برنامه	CR	OS	WR	TO	SD
BS	31.2	30.8	4	34	0
BS_ECCA	27.1	28	10.3	6.1	28.5
BS_CFCSS	18.2	31.9	13.1	10.3	26.5
BS_RSCFC	38.6	18.3	12.2	2.5	28.4
BS_CFCCB	29	21.9	6.8	1.8	40.5
BS_SCFC	40.5	17.6	1.7	1.2	39
BS_OSCFC	23.3	24.4	**0.6**	1.2	**50.5**
MM	23.4	32.3	25.7	18.6	0
MM_ECCA	11.6	39.2	14.9	6.2	28.1
MM_CFCSS	24.6	26.9	10.7	16.8	21.4
MM_RSCFC	32.8	25.2	10.5	1.6	29.9
MM_CFCCB	22	20.6	15.9	7.2	34.3
MM_SCFC	46.6	28.3	1.9	1.3	21.9
MM_OSCFC	36.6	20.4	**1.8**	1.4	**39.8**
QS	31.9	39.4	25	3.7	0
QS_ECCA	11.2	34.8	13	8.1	32.9
QS_CFCSS	31.9	22.8	6.1	5.4	33.8
QS_RSCFC	37.2	27.2	5.9	3.6	26.1
QS_CFCCB	22.2	27.5	11.7	9.6	29
QS_SCFC	45.1	26.6	**0.8**	1.8	25.7
QS_OSCFC	31.1	24.7	1.2	1.2	**41.8**

برای مقایسه درصد پوشش خطای روش‌های مختلف، جدول ۳ جمع اعداد سطرهای مربوطه در جدول ۲ غیر از ستون WR را برای روش‌های مختلف به همراه میانگین آن‌ها نشان می‌دهد. همچنین، در شکل ۵ نمودار درصد پوشش خطای روش‌های مختلف به تفکیک برنامه‌های محک نمایش داده شده است. در جدول ۳ اعداد ضخیم شده بیشترین درصد پوشش خطای بدست آمده برای یک برنامه خاص یا بهترین میانگین را نشان می‌دهند که با توجه به آن، روش پیشنهادی OSCFC دارای بیشترین میانگین پوشش خطا است.

جدول ۳: درصد پوشش خطای حاصل از اعمال روش‌های مختلف بر روی برنامه‌های محک به همراه میانگین بدست آمده برای هر روش.

روش	BS	MM	QS	میانگین پوشش خطا
ECCA	89.7	85.1	87	87.3
CFCSS	86.9	89.7	93.9	90.2
RSCFC	87.8	89.5	94.1	90.5
CFCCB	93.2	84.1	88.3	88.5
SCFC	98.3	98.1	**99.2**	98.5
OSCFC	**99.4**	**98.2**	98.8	**98.8**



شکل ۵: پوشش خطای حاصل از اعمال روش‌های مختلف بر روی سه برنامه محک.

درصد سربارهای کارایی و حافظه روش‌های تحت بررسی پس از اعمال بر روی برنامه‌های محک و اجرای آن‌ها، به ترتیب در جدول‌های ۴ و ۵ ارائه شده است. منظور از سربار کارایی همان سربار زمان اجرا است و منظور از سربار حافظه میزان افزایش حافظه مصرفی برنامه تغییر داده شده نسبت به برنامه اولیه است. در این دو جدول، مقادیر ضخیم شده کمترین سربارها را نشان می‌دهند که با توجه به آن، روش پیشنهادی در هر سه برنامه محک دارای کمترین سربار حافظه است. همچنین، جدول ۶ میانگین سربارهای کارایی و حافظه روش‌های مختلف را پس از اعمال به سه برنامه محک نشان می‌دهد. با توجه به این جدول، روش پیشنهادی OSCFC دارای کمترین سربار حافظه در میان روش‌های مختلف است، اما از نظر سربار کارایی در جایگاه دوم قرار دارد.

جدول ۴: درصد سربارهای کارایی روش‌های مختلف پس از اعمال بر روی برنامه‌های محک.

روش	BS	MM	QS
ECCA	39	36	42
CFCSS	40	68	45
RSCFC	64	48	59
CFCCB	**35**	**33**	34
SCFC	46	40	34
OSCFC	40	38	**33**

جدول ۵: درصد سربارهای حافظه روش‌های مختلف پس از اعمال بر روی برنامه‌های محک.

روش	BS	MM	QS
ECCA	48	41	46
CFCSS	45	50	55
RSCFC	72	43	65
CFCCB	41	37	45
SCFC	45	39	45
OSCFC	**40**	**36**	**44**

جدول ۶: میانگین سربارهای کارایی و حافظه روش‌های مختلف به درصد.

روش	میانگین سربار حافظه	میانگین سربار کارایی
ECCA	45	39
CFCSS	50	51
RSCFC	60	57
CFCCB	41	**34**
SCFC	43	40
OSCFC	**40**	37

برای ارزیابی و مقایسه دقیق‌تر روش پیشنهادی با روش‌های قبلی با توجه به هر سه پارامتر پوشش خطا، سربار کارایی و سربار حافظه، از فاکتور ارزیابی (EF[۳۷]) کلی طبق معادله (۱) که در مرجع [۱۱] معرفی شده است، استفاده می‌کنیم:

$$EF = \frac{\text{میانگین پوشش خطا}}{\text{سربار کارایی} \times \text{سربار حافظه}} \qquad (۱)$$

در معادله بالا، میانگین پوشش خطا به درصد بیان می‌گردد، اما سربارهای کارایی و حافظه به صورت نسبت کلی بیان می‌شوند. به عنوان مثال، با توجه به جدول ۶ که میانگین سربارهای کارایی و حافظه روش پیشنهادی به ترتیب برابر با ٪۳۷ و ٪۴۰ است، اعداد مورد استفاده در مخرج معادله (۱) به ترتیب برابر با ۱/۳۷ و ۱/۴۰ خواهند بود. در نتیجه، فاکتور ارزیابی روش پیشنهادی OSCFC با توجه به مقدار میانگین پوشش خطای آن که طبق جدول ۳ به درصد عدد ۹۸/۸ است، برابر با ۵۱/۵ خواهد شد. جدول ۷ نتایج بدست آمده طبق فاکتور ارزیابی معادله (۱) را برای پنج روش قبلی به همراه روش پیشنهادی نشان می‌دهد. با توجه به فاکتور ارزیابی، هر چه عدد بدست آمده بزرگتر باشد، نشان دهنده بهتر بودن روش تحت بررسی است. با توجه به این جدول، روش پیشنهادی OSCFC با توجه به ارزیابی همزمان سه پارامتر، بهترین روش محسوب می‌گردد.

روش پیشنهادی OSCFC در مقایسه با روش RASM [۱۹] که اخیراً معرفی شده، با توجه به نتایج عددی ارائه شده در مرجع [۱۹] وضعیت مناسبی دارد. میانگین پوشش خطای روش RASM برای سه برنامه محک استفاده شده در این مقاله برابر با ٪۹۸/۲ است که از میانگین پوشش خطای روش پیشنهادی کمتر است. همچنین، سربارهای کارایی و حافظه روش RASM برای برنامه‌های محک مشترک به ترتیب برابر با ٪۷۷/۷ و ٪۱۲/۴ است که اولی بیشتر و دومی کمتر از روش پیشنهادی است. علاوه بر این، مقدار فاکتور ارزیابی روش RASM برابر با ۴۹/۲ می‌شود که از روش پیشنهادی OSCFC کمتر است.

جدول ۷: مقادیر بدست آمده برای فاکتور ارزیابی کلی روش‌های مختلف.

روش	مقدار فاکتور ارزیابی
ECCA	43.3
CFCSS	39.8
RSCFC	36.0
CFCCB	46.9
SCFC	49.2
OSCFC	**51.5**

۶- نتیجه‌گیری

در این مقاله، یک روش جدید با نام OSCFC برای شناسایی خطاهای کنترلی رخ دهنده حین اجرای برنامه‌ها در پردازنده‌ها معرفی گردید. دلیل رخداد خطاهای کنترلی خطاهای گذرایی است که ممکن است در بخش‌های مختلف یک پردازنده رخ دهد. در روش پیشنهادی دو نوع دستورالعمل کنترلی به ابتدا و انتهای هر بلوک پایه اضافه می‌گردد. همچنین، یک دستورالعمل کنترلی برای تشخیص خطاهای درون‌بلوکی به بلوک‌های حاوی حداقل دو

دستورالعمل غیر پرش اضافه می‌شود. پس از تشخیص خطا، مدیریت وضعیت رخ داده بر عهده سطح بالاتر سیستم است. روش پیشنهادی برای تقویت برنامه‌های اجرا شونده بر روی پردازنده‌های عام در محیط‌هایی مانند خارج از جو زمین که احتمال رخداد خطاهای گذرا به علت برخورد ذرات پرانرژی زیاد است، مفیدتر خواهد بود. نتایج شبیه‌سازی، تزریق خطا و پیاده‌سازی پنج روش پیشین همراه با روش پیشنهادی نشان می‌دهد که میزان پوشش خطا و سربار حافظه روش پیشنهادی OSCFC مناسب‌تر از روش‌های قبلی بوده و سربار کارایی آن در وضعیت مطلوبی قرار دارد. علاوه بر این، با توجه به فاکتور ارزیابی کلی تعریف شده که دربرگیرنده هر سه پارامتر اصلی است، روش پیشنهادی بهترین روش محسوب می‌گردد.

مراجع

[1] P. Shirvani, N. Saxena, and E. McCluskey, "Software-implemented EDAC protection against SEUs", IEEE Trans. on Reliability, vol. 49, no. 3, pp. 273–284, 2000.

[2] A. Shrivastava, A. Rhisheekesan, R. Jeyapaul, and C.-J. Wu, "Quantitative Analysis of Control Flow Checking Mechanisms for Soft Errors", 51th ACM/EDAC/IEEE Design Automation Conf. (DAC), pp. 1–6, 2014.

[3] D. Zhu and H. Aydin, "Reliability Effects of Process and Thread Redundancy on Chip Multiprocessors", Proc. 36th Annu. IEEE/IFIP Int. Conf. Dependable Systems and Networks, pp. 212–213, 2006.

[۴] طاهره برومند نژاد، محمد عبداللهی ازگمی، شاهرخ جلیلیان، "بررسی فنون نرم‌افزاری تحمل‌پذیری خطای گذرا در نرم‌افزارهای ماهواره"، فصل‌نامه علمی پژوهشی علوم و فناوری فضایی، جلد ۵، شماره ۴، ص ۹-۱۸، زمستان ۱۳۹۱.

[5] J.R. Azambuja, F. Kastensmidt, and J. Becker, "Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors", Springer International Publishing Switzerland, 2014.

[6] M. Duricek and T. Krajcovic, "Hybrid Control-Flow Checking with On-Line Statistics", 4th Eastern European Regional Conf. on the Engineering of Computer Based Systems, pp. 122–125, 2015.

[7] S.A. Asghari, A. Abdi, H. Taheri, S. Pourmozaffari, and H. Pedram, "SEDSR: Soft error detection using software redundancy," Journal of Software Engineering and Applications, vol. 5, pp. 664–670, 2012.

[8] N. Oh, P.P. Shirvani, and E.J. McCluskey, "Control-Flow Checking by Software Signatures", IEEE Trans. on Reliability, vol. 51, no. 2, pp. 111–122, March 2002.

[9] A. Li and B. Hong, "Software implemented transient fault detection in space computer", Elsevier, Aerospace Science and Technology, vol. 11, no. 2–3, pp. 245–252, 2007.

[10] Z. Alkhalifa, V.S.S. Nair, N. Krishnamurthy, and J.A. Abraham, "Design and Evaluation of System-Level Checks for On-Line Control Flow Error Detection", IEEE Trans. on Parallel and Distributed Systems, vol. 10, no. 6, pp. 627–641, June 1999.

[11] S.A. Asghari, H. Taheri, H. Pedram, and O. Kaynak, "Software-Based Control Flow Checking Against Transient Fault in Industrial Environments", IEEE Trans. on Industrial Informatics, pp. 481–490, 2014.

[12] A. Li and B. Hong, "On-line control flow error detection using relationship signatures among basic blocks," Computers and Electrical Eng., vol. 36, pp. 132–141, 2010.

[13] Boroomandnezhad, T., Azgomi, M. A., "An efficient control-flow checking technique for the detection of soft-errors in embedded software", Computers & Electrical Engineering, vol. 39, no. 4, pp. 1320–1332, 2013.

[14] Jianli, L., Qingping, T., Jianjun, X., "A Software-Implemented Configurable Control Flow Checking Method", *3rd Int. Symp. on Parallel Architectures*, Algorithms and Programming, pp. 199–205, 2010.

[15] Asghari, S.A., Abdi, A., Taheri, H., Pedram, H., Pourmozaffari, S., "I2BCFC: An Effective Intra-Inter Block Control Flow Checking Method Against Single Event Upsets", Research Journal of Applied Sciences, Engineering and Technology, vol. 4, no. 21, pp. 4367–4379, 2012.

[16] L. Terras, Y. Teglia, M. Agoyan, R. Leveugle, "Taking into account indirect jumps or calls in continuous control-flow checking", 11th Int. Design & Test Symp. (IDT), pp. 125–130, 2016.

[17] L. Liu, L. Ci, W. Liu, C. Bin, "Control-Flow Checking Using Branch Sequence Signatures", IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 839–845, 2016.

[18] Z., Zhu, J. Callenes-Sloan, "Towards Low Overhead Control Flow Checking Using Regular Structured Control", 20th Design Automation and Test in Europe Conf. and Exhibition (DATE), pp. 826–829, 2016.

[19] J. Vankeirsbilck, N. Penneman, H. Hallez, J. Boydens, "Random Additive Signature Monitoring for Control Flow Error Detection", IEEE Trans. on Reliasbility, vol. 66, no. 4, Dec. 2017.

[20] S. Das, W. Zhang, Y. Liu, "A Fine-Grained Control Flow Integrity Approach Against Runtime Memory Attacks for Embedded Systems", IEEE Trans. on VLSI Systems, vol. 24, no. 11, pp. 3193–3207, Nov. 2016.

زیرنویس‌ها:

[1] Commercial Off-The Shelf

[2] Software-Implemented Hardware Fault-Tolerance

[3] Data Flow Errors

[4] Data Flow Checking

[5] Control Flow Errors

[6] Control Flow Checking

[7] Signature

[8] Duplication and comparison

[9] Basic Block

[10] Bubble Sort

[11] Branch

[12] Jump

[13] Call

[14] Enhanced Control flow Checking using Assertion

[15] Block Identifier

[16] Control Flow Checking by Software Signature

[17] Successor

[18] Relationship Signatures for Control Flow Checking

[19] Inter-block

[20] Intra-block

[21] Control Flow Checking method based on Classifying Basic blocks

[22] Conflict

[23] Software-based Control Flow Checking

[24] Random Additive Signature Monitoring

[25] Field Programmable Gate Array

[26] Application Specific Integrated Circuit

[27] Optimized Software-based Control Flow Checking
[28] Random error injection
[29] Quick Sort
[30] Matrix Multiplication
[31] Program Counter
[32] Correct Result
[33] Operating System
[34] Time-Out
[35] Software Detection
[36] Wrong Result
[37] Evaluation Factor