تاریخ دریافت مقاله: ۹۶/۰۸/۳۰
تاریخ پذیرش مقاله: ۹۷/۰۳/۰۷

تسریع اجرای الگوریتم تولید خودکار الگوی آزمون مبتنی بر رویکرد ارضاپذیری بولی با استفاده از پردازنده‌های گرافیکی

محمد مهدی ابوالحسنی فروغی
کارشناس ارشد دانشکده مهندسی برق و کامپیوتر – دانشگاه شیراز – شیراز – ایران
پست الکترونیکی: foroughi1991@cse.shirazu.ac.ir

محسن راجی*
استادیار دانشکده مهندسی برق و کامپیوتر – دانشگاه شیراز – شیراز – ایران
پست الکترونیکی: mraji@shirazu.ac.ir

چکیده

با توجه به بزرگ شدن اندازه مدارها، روش‌های تولید خودکار الگوی آزمون برای تراشه‌های بسیار مجتمع امروزی، کارآیی خود را از دو منظر درصد شناسایی اشکال (پوشش اشکال) و سرعت اجرا از دست داده‌اند. دسته‌ای از این روش‌ها، الگوریتم‌های تولید الگوی آزمون مبتنی بر رویکرد ارضاپذیری بولی هستند که در آن‌ها، تابع منطقی مدار ابتدا به تعدادی ماکسترم تبدیل شده و سپس عبارت بولی به‌دست آمده اصطلاحاً حل شده و بردارهای آزمون به‌دست می‌آیند. این الگوریتم‌ها پوشش اشکال بالایی از خود نشان می‌دهند اما زمان اجرای زیاد آن‌ها باعث شده تا اقبال کمتری به این دسته از الگوریتم‌ها وجود داشته باشد. در این مقاله، با بهره‌گیری از محاسبات موازی، سرعت اجرای یک الگوریتم تولید الگوری آزمون برای مدارهای دیجیتال مبتنی بر ارضاپذیری بولی بهبود داده شده است. به این منظور، بخشی از فرآیند تولید الگوی آزمون به‌صورت موازی و بر روی پردازنده گرافیکی (با استفاده از کتابخانه کودا) انجام می‌شود. نتایج به‌دست آمده نشان می‌دهد روش پیشنهادی به میزان بیش از ٤ برابر سریع‌تر از زمانی است که بر روی یک پردازنده چندهسته‌ای اجرا می‌شود و به‌طور میانگین بیش از ۹ برابر سریع‌تر از الگوریتم پیشنهادی در [۸] است که سعی نموده زمان اجرای الگوریتم‌های مبتنی بر ارضاپذیری بولی را بهبود دهند.

واژه‌های کلیدی: تولید خودکار الگوی آزمون، بردار آزمون، پوشش اشکال، محاسبات موازی، پردازنده گرافیکی.

۱- مقدمه

امروزه مدارهای مجتمع در کاربردهای بسیار حساسی مانند سیستم کنترل خودرو، وسایل پزشکی یا وسایل کنترلی هواپیما استفاده می‌شوند. در نتیجه، اطمینان از عملکرد صحیح تراشه تولید شده که به کمک فرآیند آزمون انجام می‌پذیرد، بسیار حیاتی می‌باشد. در این فرآیند، ورودی‌هایی که بردارهای آزمون[1] یا الگوهای آزمون[2] نامیده می‌شوند، به مدارِ تحتِ آزمون داده می‌شود. سپس

1- Test vectors
2- Test patterns

* نویسندهٔ مسئول

خروجی مدار سالم با خروجی مدار تحت آزمون مقایسه می‌شود. اگر یک یا چند خروجی، جوابی متفاوت با خروجی مورد انتظار داشته باشد، تراشه به‌عنوان یک تراشه معیوب رد می‌شود. استفاده از تمام حالات ممکن ورودی‌ها در مجموعه بردار آزمون باعث می‌شود که فرآیند آزمون بیش از حد زمانبر و پر هزینه گردد. بنابراین، بایستی زیرمجموعه‌ای از حالات ممکن ورودی‌ها به‌عنوان مجموعه بردار آزمون تعیین شوند که بیشترین قابلیت در شناسایی مدار معیوب را داشته باشد. الگوریتم‌هایی که این بردارها را تولید می‌کنند، الگوریتم‌های تولید خودکار الگوی آزمون (ATPG) نامیده می‌شوند [۱].

اغلب الگوریتم‌های کلاسیک ATPG برپایه الگوریتم D هستند که توسط Roth در سال ۱۹۶۶ ارائه شد [۱]. این الگوریتم‌ها به‌طور مستقیم از اطلاعات ساختاری مدار بهره می‌برند. اما با پیشرفت فناوری و افزایش تعداد ترانزیستورها و دروازه‌ها در مدارهای امروزی، کارایی ATPGهای کلاسیک روز به روز محدودتر می‌شوند. بردارهای آزمون برای تعداد بسیار زیادی از اشکال‌ها همچنان تولید می‌شود اما تعداد اشکال‌هایی که برای آن‌ها الگوی آزمون تولید نمی‌شود نیز به‌صورت قابل توجهی در حال افزایش است. به این ترتیب، نسبت تعداد اشکال‌هایی که به کمک یک مجموعه از بردارهای آزمون شناسایی شده و پوشش داده می‌شوند به کل تعداد اشکال‌های بالقوه در مدار کاهش می‌یابد. به این نسبت، پوشش اشکال[3] گفته می‌شود. با کاهش پوشش اشکال، کیفیت آزمون مدارها که توسط ATPG تولید می‌شود کاهش می یابد.

یک راه حل مناسب برای دستیابی به پوشش اشکال بسیار بالا، به کار گرفتن روش ارضاپذیری بولی[4] یا SAT است[۲]. برخلاف ATPGهای کلاسیک، الگوریتم‌های SAT بر مبنای ساختار مدار کار نمی‌کنند بلکه بر روی فرمول بولی[5] یا فرم نرمال کانونی[6] (CNF) از مدار متناظر مثلا Z کار می‌کند[۳] . مسئلهٔ SAT به این گونه تعریف می‌شود که آیا انتسابی به نام a وجود دارد به گونه‌ای که فرمول بولی Z(a) برابر با ۱ شود یا خیر[۱] . به این ترتیب، مسئله یافتن الگوی آزمون به یک فرمول بولی تبدیل شده که جواب آن در صورت ۱ شدن، برداری از ورودی‌هاست به نحوی که اشکالی که در یکی از دروازه‌های مدار در نظر گرفته شده است، در حداقل یکی از خروجی‌های مدار مشاهده شود. با وجود پوشش اشکال بالا، زمان حل مسئله SAT، بسیار بالا بوده و همین امر باعث شده که برای تولید الگوی آزمون، به‌صورت گسترده مورد استفاده قرار نگیرند.

اخیرا برای تسریع برخی الگوریتم‌ها به کمک کامپیوتر از محاسبات موازی بر روی پردازنده‌های گرافیکی استفاده می‌شود[۷][۶][۵][٤]. در این رویکرد، از قابلیت اجرای همزمان بخش های مختلف یک الگوریتم یا اجرای یک بخش یکسان از الگوریتم بر روی تعداد زیادی داده ورودی به‌طور همزمان استفاده می‌شود و بر روی امکانات موجود بر روی پردازنده‌های گرافیکی به اجرا درمی‌آیند[٤]. به این ترتیب، سرعت اجرای الگوریتم‌ها نسبت به حالت غیرموازی افزایش می یابد.

در این مقاله، با بهره‌گیری از محاسبات موازی بر روی پردازنده‌های گرافیکی، سرعت اجرای الگوریتم تولید الگوی آزمون مبتنی بر رویکرد ارضاپذیری بولی را افزایش می‌دهیم. به این صورت که ابتدا تولید الگوی آزمون به‌صورت یک مسئله ارضاپذیری بولی تبدیل می‌شود و سپس بخشی از حل آن مسئله، بر روی پردازنده گرافیکی انجام می‌شود. به این ترتیب، مسئله به مسائل کوچک‌تر شکسته می‌شود و الگوریتم بر روی داده‌های با تعداد زیاد به‌صورت همزمان شروع به کار می‌کند. قسمت سریال روش پیشنهادی بر روی پردازنده ۵ هسته ای[7] و قسمت موازی آن بر روی کارت گرافیک جی تی ایکس[8] ۷۸۰ به اجرا درآمده است. از روش پیشنهادی برای تولید الگوی آزمون برای مدارهای محک ISCA'85 [۱۹]استفاده شده

3- Fault coverage
4- Boolean satisfiability
5- Boolean formula
6- Conjunctive Normal Form

7- core i5 4550
8- GTX780

است. نتایج به‌دست آمده نشان می‌دهد روش پیشنهادی به میزان بیش از ٤ برابر سریع‌تر از زمانی است که بر روی یک پردازنده چندهسته‌ای اجرا می‌شود و به‌طور میانگین بیش از ۹ برابر سریع‌تر از الگوریتم پیشنهادی در [۸] است که سعی نموده زمان اجرای الگوریتم‌های مبتنی بر ارضاپذیری بولی را بهبود دهند.

ادامه این مقاله به این صورت سازماندهی شده است: در بخش ۲ مروری بر کارهای مشابه خواهیم داشت و نقاط ضعف و قوت به اختصار بیان می‌شود. در بخش ۳ با مفهوم ارضاپذیری بولی آشنا و در زیربخش آن یکی از الگوریتم‌های متداول برای حل مسائل آن بررسی می‌شود. در قسمت ٤ با کودا آشنا می‌شویم و در زیر بخش آن با ساختار پردازنده گرافیکی و همچنین نحوه اجرای کد بر روی پردازنده گرافیکی آشنا می شویم. در قسمت ٥ معماری و الگوریتم پیشنهادی برای ساخت الگوی آزمون با استفاده از پردازنده گرافیکی ارائه شده است. در قسمت ٦ نتایج اجرا آورده شده است همچنین با نتایج اجرای شبیه‌ساز خطای SAT Compress مقایسه زمانی انجام گرفته است. در قسمت ۷ نتیجه‌گیری این مقاله آورده شده است.

۲- مروری بر کارهای گذشته

مسئلهٔ SAT اولین مسئله‌ای بود که NP کامل بودن آن از نظر زمانی توسط کوک در سال ۱۹۷۱ اثبات شد[۲] یک ابزار ATPG بر پایه SAT به نام PASSAT، برای مدل‌های اشکال در محیط‌های صنعتی ارائه شده است.[۳] اولین نتایج PASSAT بسیار امیدوار کننده بود برای مثال بسیاری از اشکال‌هایی که با روش‌های قدیم به سختی شناسایی می‌شدند یا قابل شناسایی نبودند توسط این الگوریتم شناسایی شدند. نتایج نشان دادند که این الگوریتم پوشش خطای بالایی دارند اما PASSAT به‌عنوان یکی از پیشرفته‌ترین روش‌های ATPG، نقاط ضعفی دارد که باعث عدم کارایی آن در مسائل صنعتی می‌شود. نقاط ضعف اصلی شامل موارد زیر است:

۱) زمان اجرا – الگوریتم‌های SAT نتایج امیدوار کننده‌ای در شناسایی اشکال‌هایی که به سختی قابل کشف هستند دارند اما سربار زیادی در شناسایی اشکال‌های آسان دارند که قابل قبول نیست.

۲) فشردگی الگوی آزمون – فرآیند ATPG در محیط‌های صنعتی بخشی از یک جریان بزرگ‌تر می‌باشد. برای به‌دست آوردن یک الگوی آزمون کوچک، آزمون‌های تولید شده توسط روش‌هایی به نام فشرده‌سازی آزمون پردازش و از نظر طول و تعداد بهینه می‌شوند. به‌طور معمول ATPGهای بر پایه SAT تعداد زیادی الگوی آزمون تولید می‌کنند که معمولا اندازه مجموعه آزمون به‌صورت قابل توجه افزایش می‌یابد که قابل قبول نیست زیرا به‌طور مستقیم هزینه آزمون را افزایش می‌دهد.

۳) اشکال‌های مدل تاخیر – فرآیند آزمون در محیطهای صنعتی معمولا برای بیش از یک مدل اشکال انجام می‌شود. اغلب ATPG بر پایه SAT برای مدل اشکال چسبندگی (SAFM)[9] توسعه می‌یابند. این محدودیت برای PASSAT نیز وجود دارد و این روش، سایر مدل‌های اشکال را در نظر نگرفته است.

٤) اشکال‌های تاخیر کوچک[10]– شناسایی اشکال‌های تاخیر کوچک به یک مسئلهٔ بزرگ در ساخت تبدیل شده است. شناسایی چنین اشکال‌هایی نیازمند پردازش اطلاعات زمانی بلند ترین مسیر مدار است و از آنجایی که SAT یک مسئلهٔ بولی است نه یک مسئلهٔ بهینه‌سازی، الگوریتم‌های قدیمی ATPG بر پایه SAT مناسب چنین مشکلاتی نیستند و باید به گونه‌ای اطلاعات ساختاری را ذخیره کرد.

جهت افزایش فشردگی الگوهای آزمون تولید شده شبیه ساز خطای SAT Compress در سال ۲۰۱۰ ارائه شد [۸] که اجرا در آن به‌صورت ردیفی انجام می‌شود و زمان اجرای نن با بزرگ شدن مدار بیشتر می‌شود. شبیه ساز خطای TI-GUAN برای خطاهای مقاومتی و اشکال‌های تاخیر کوچک

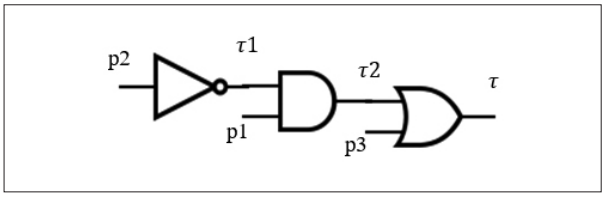
9- Single stuck at fault model
10- Small delay defects

به‌صورت چند نخی تنها با ٤ نخ پردازشـــی در سال ۲۰۱۲ ارائه شد[۹] . اما هنوز مشکلاتی در حوزه ATPGهای مبتنی بر SAT باقی مانده اســـت. مهم‌ترین این مشکلات که باعث شـــده است تا هنوز این رویکرد در صنعت با اقبال چندانی رو به رو نشـــود، مسئله زمان اجرای بسیار زیاد آن است که به دلیل NP کامل بودن نوع الگوریتم است [۳].

۳- ارضاپذیری بولی

برخلاف روش‌های ATPG که بر روی ساختار مدار در سطح دروازه کار می‌کنند، روش‌های ارضاپذیری بولی بر روی فرمول‌های بولی کار می‌کنند. روش‌های بر پایه SAT را بر اســـاس داده‌ای کـــه روی آن کار می‌کنند می‌توان به دو دســـته تقســـیم کرد: روش‌هایی که از گراف استنباط[11] اســـتفاده می‌کنند و روش‌هایی که مـــدار را به فرم CNF[12] تبدیل می‌کنند. گراف اســـتنباط، گراف مستقیم جهت‌داری است که عملکرد مدار را به شکل گراف نشان می‌دهد. مزیت این روش تلفیق مزایای ساختاری مدار با اطلاعات مربوط به عملکرد مدار اســـت، گرچه تشکیل گراف برای مدارهای بزرگ به صرفه نیســـت. در فرم CNF، عملکرد منطقی مدار را به‌عنـــوان فرمول بولی نمایش می‌دهد. یکی از دلایلی که باعث قـــوی بودن این روش می‌شـــود تحلیل قوی تناقض اســـت. اما در عوض اطلاعات ســـاختاری مدار را از دست می‌دهیم.

تبدیل یک مدار دیجیتال به شکل CNF یک بخش اساسی در یک ATPG مبتنی بر ارضاپذیری بولی است. فرمول CNF با m متغیـــر، ترکیب عطفی[13] n گزاره اســـت که هر گزاره ترکیب فصلـــی[14] تعدادی لیترال[15] اســـت. لیترال‌ها فازهای مختلف یک متغیر هستند. یک CNF زمانی ارضاپذیر است که بتوانیـــم مقادیری از متغیرهای مـــدار بیابیم به نحوی که تمام گزاره‌هـــای آن ارضا (معادل با مقدار منطقی یک) شده باشند. بنابراین در تبدیل مدار به فرم CNF باید به هر



شکل۱ - مثالی برای تولید CNF مدار

سیگنال در مدار یک متغیر نسبت داده شود. سپس برای هر دروازه، CNF آن با توجه به عملکرد آن دروازه تعیین شود. CNF کل مـــدار Θ هم از ترکیب عطفی CNF دروازه‌های آن تشکیل می‌شود [٤]:

$$\Theta_c = \prod_{i=1}^{n} \Theta_{g_i} \qquad (۱)$$

در فرمول ۱ CNF مدار برابر اســـت با ترکیب عطفی CNF دروازه‌های درون مدار. برای تولید CNF روش‌های متفاوتی وجود دارد. اســـتخراج فرمول از روی گراف [٦]، الگوریتم جکسون-شـــریدان [۱۱] و تبدیل ســـتین[۱۲] از جمله این روش‌هاســـت. به دلیـــل پایین‌تر بـــودن پیچیدگی زمانی و پیچیدگی پیاده‌سازی از الگوریتم ستین استفاده شده است. مزیت این الگوریتم‌ها به‌طور کلـــی کاهش فضای ورودی مســـئله اســـت که رابطه خطی با تعـــداد ورودی‌های مدار دارد. ایـــن روش مدار را بـــه دو بخش اتمیک و غیر اتمیک تقســـیم می‌کند و برای قســـمت‌های غیر اتمیک زیرفرمول Θ یک متغیر کمکی ماننـــد q_θ در نظر می‌گیرد به‌طوری که $\theta \leftrightarrow q_\theta$ (بـــه این معنی که q_θ و θ معادل هســـتند) برقرار است. ســـپس از قوانین دمورگان برای حذف ↔ استفاده می‌شـــود و در نهایـــت فرمولی که شـــامل ترکیب فصلی لیترال‌ها اســـت به‌دست می‌آید. برای مثال در مدار شکل ۱ CNF به کمک روابط (۱) تا (۳) به‌دست می‌آید.

مدار بالا OR دو بخش ورودی p۳ و خروجی AND دو عبارت دیگر است. با توجه به توضیحات بالا ورودی‌های اصلی متغیرهای اتمیک هستند و سایر عبارت‌ها غیر اتمیک هســـتند بنابراین برای هر قسمت غیراتمیک مانند فرمول ۲ یک متغیر کمکی در نظر می‌گیریم. برای p2~ متغیر کمکی τ1 و بـــرای and دو عبـــارت p1 و p2~ متغیـــر کمکی τ2

11- Implication Graph
12- Conjunctive Normal Form
13- Conjunction
14- Disjunction
15- Literal

همچنین برای کل عبارت τ در نظر می‌گیریم.

$$\tau = \underbrace{(p_1 \wedge \overbrace{\neg p_2}^{\tau_1})}_{\tau_2} \vee p_3 \qquad (۲)$$

$$\begin{aligned} &(q_{\tau_1} \leftrightarrow \neg p_2) \wedge \\ &(q_{\tau_2} \leftrightarrow p_1 \wedge q_{\tau_1}) \wedge \\ &(q_{\tau} \leftrightarrow q_{\tau_2} \vee p_3) \wedge \\ &q_{\tau} \end{aligned} \qquad (۳)$$

سپس معادل بودن متغیرهای کمکی و عبارت‌های غیراتمیک را به‌صورت روابط دو طرفه می‌نویسیم مانند فرمول ۳. با اعمال قوانین دمورگان و حذف روابط دوطرفه فرمول نهایی ٤ به‌دست می‌آید:

$$\begin{aligned} &(q_{\tau_1} \vee p_2) \wedge (\neg q_{\tau_1} \vee \neg p_2) \wedge \\ &(\neg q_{\tau_2} \vee p_1) \wedge (\neg q_{\tau_2} \vee q_{\tau_1}) \wedge (q_{\tau_2} \vee \neg p_1 \vee \neg q_{\tau_1}) \wedge \\ &(q_{\tau} \vee \neg q_{\tau_2}) \wedge (q_{\tau} \vee \neg p_3) \wedge (\neg q_{\tau} \vee q_{\tau_2} \vee p_3) \wedge \\ &q_{\tau} \end{aligned} \qquad (۴)$$

برای استفاده از رویکرد ارضاپذیری بولی به منظور تولید الگوی آزمون، باید مدار سالم با مدار اشکال‌دار را از قسمت وقوع اشکال تا خروجی اصلی، XOR کنیم. سپس از طریق مراحلی که بیان شد، CNF کلی را به‌دست آوریم. به این ترتیب با حل فرمول به‌دست آمده الگوی آزمون به‌دست خواهد آمد؛ به این معنی که با یافتن جواب‌های فرمول به نحوی که منجر به یک شدن خروجی XOR شود (که نشان‌دهنده متفاوت شدن جواب مدار سالم و اشکال‌دار است)، الگوهای آزمون به‌دست می‌آیند. بنابراین CNF مدار شامل XOR، مدار سالم و مدار اشکال‌دار را به‌دست آورده و به این ترتیب، CNF برای مدار کشف کننده اشکال F (Θ_{test}^{F}) برابر خواهد شد با:

$$\Theta_{test}^{F} = \Theta_C \wedge \Theta_F \wedge \Theta_{XOR} \qquad (۵)$$

که در آن Θ_C، Θ_F و Θ_{XOR} به ترتیب CNF مربوط به مدار سالم C، مدار اشکال دار F و مدار XOR خواهد بود.

جواب‌های فرمول ٥ الگوهای آزمونی که اشکال F را شناسایی می‌کنند نمایش می‌دهند. اگر این فرمول جوابی نداشته باشد، اشکال F قابل شناسایی نیست.[۱۳]

در قدم بعدی باید به دنبال روشی کارآمد برای حل این فرمول و دستیابی به مقادیر هر متغیر باشیم. روش‌های متعددی برای حل این فرمول ارائه شده است که رائج‌ترین آن روش DPLL[16] است [١٤] که رویکردهای خلاقانهٔ بسیار زیادی برای بهبود آن ارائه شده است.

۳-۱- الگوریتم DPLL

الگوریتم DPLL به‌صورت تکرارشونده فرمول ورودی را با استفاده از قوانین زیر به گونه‌ای تغییر می‌دهد که فرمول جدید خالی از گزاره شود که به معنی ارضا شدن فرمول است. قوانین به شرح زیر هستند:

• انتشار واحد[17]: اگر گزاره‌ای با یک لیترال در فرمول ظاهر شد تمام گزاره‌هایی که آن لیترال را دارند حذف می‌کنیم و تمام لیترال‌های متمم آن را از گزاره‌ها حذف می‌کنیم. حذف یک لیترال به معنای این است که مقدار آن لیترال در حل فعلی فرمول، مشخص شده است.

• لیترال خالص[18]: اگر در فرمول یک لیترال تنها با یک فاز مثبت یا منفی خود ظاهر شد، تمام گزاره‌هایی که آن لیترال را دارند حذف می‌کنیم.

• تحلیل[19]: پس از اعمال قوانین بالا یک متغیر انتخاب می‌کنیم و فرمول را ساده می‌کنیم، در صورت برخورد با تناقض که به‌صورت ظهور یک متغیر با دو مقدار است، عقب‌گرد می‌کنیم و مقداردهی قبلی خود را تصحیح می‌کنیم برای مثال اگر مقدار اولیه یک در نظر گرفتیم آن را به صفر تصحیح می‌کنیم و برعکس.

همان‌طور که در مرحله آخر مشخص است، در بدترین حالت برای مسئله‌ای با n متغیر، مجبور به بررسی 2^n حالت هستیم. رویکردهای خلاقانه بسیار زیادی برای کاهش زمان اجرای این الگوریتم ارائه شده است [۱۵]. به دلیل این‌که هدف اصلی ما پیاده‌سازی بر روی پردازنده‌های گرافیکی است و نیاز اولیه این بُن‌سازه[20] استقلال داده‌ها است، رویکرد خلاقانه‌ای را انتخاب کنیم که نیازمند کمترین وابستگی ممکن در داده‌ها باشد زیرا در غیر این صورت مناسب برای موازی‌سازی نخواهد بود. برای این منظور

16- Davis Putnam Logemann Loveland
17- Unit propagation
18- Pure literal
19- Resolution
20- Platform

از رویکرد خلاقانه یادگیری گزاره[21] [۱۶] استفاده کرده‌ایم.

۳-۲- یادگیری گزاره

در جریان حل یک رابطه CNF، گزاره‌های متناقض بسیار زیادی در زمان جستجوی جواب تولید می‌شوند. در حل CNF مربوط به یک مدار دیجیتال، گزاره متناقض ناشی از مقداردهی‌های نامناسب به مقادیر درون فرمول است. هر زمان که به تناقضی رسیدیم، آن تناقض تحلیل شده و گزاره متناقض تولید می‌شود که به آن یادگیری می‌گویند. گزاره‌های متناقض اضافه شده به مسئله کمک بسیار زیادی در انتخاب متغیر و نوع مقداردهی آن می‌کند و در نهایت، باعث کمتر شدن تعداد عقبگردها می‌شود. برای جزئیات بیشتر درباره روش یادگیری گزاره می‌توانید به [۱۶] مراجعه کنید.

۴- کودا

کودا (CUDA) که مخفف عبارت معماری دستگاه محاسبه یکپارچه[22] است یک بُن‌سازهٔ پردازش موازی و مدل برنامه‌نویسی است که توسط شرکت NVidia ارائه شده است. کودا به تولیدکنندگان نرم‌افزار اجازه می‌دهد تا از یک پردازنده گرافیکی[23] (GPU) که از CUDA پشتیبانی می‌کند، برای هدف پردازش عمومی استفاده کنند؛ رویکردی که پردازش گرافیکی همه منظوره[24] نامیده می‌شود. کودا به توسعه‌دهندگان امکان دسترسی مستقیم به حافظه و مجموعه دستورالعمل‌های GPU را می‌دهد. بُن‌سازهی کودا برای کار با زبان‌های برنامه‌نویسی مانند C و C++ و فرترن[25] طراحی شده‌است. این دسترسی باعث می‌شود تا استفاده از منابع GPU برای برنامه‌نویسان آسان‌تر شود، برخلاف راه کارهای دیگر مانند DIRECT3D و OpenGL که نیاز به توانایی حرفه‌ای در برنامه نویسی گرافیک داشتند.

21- Clause Learning
22- Compute Unified Device Architecture
23- Graphical Processing Unit
24- GPGPU
25- Fortran

۵- روند اجرای برنامه با استفاده از GPU

ساختار کلی برنامه که با استفاده از GPU اجرا می‌شود در شکل ۲ آورده شده است. کد برنامه به دو بخش تقسیم می‌شود: بخش میزبان[26] که روی پردازنده اصلی به اجرا درمی‌آید و بخش دستگاه[27] که روی GPU اجرا می‌شود. قسمت‌هایی از برنامه را که قابلیت اجرای موازی دارند و داده‌های آن کمترین وابستگی به هم را دارند می‌توان بر روی دستگاه به اجرا در آورد. برای این منظور، ابتدا باید تنظیمات تعداد نخ و بلوک پردازشی مشخص شوند. سپس، باید تمام اطلاعات مورد نیاز برای اجرا بر روی پردازنده گرافیکی به حافظه اصلی GPU کپی شود. پس از آن، اجرای برنامه به پردازنده گرافیکی منتقل می‌شود و پردازنده اصلی تا پایان اجرای قسمت کد دستگاه منتظر می‌ماند. در آخر، نتایج به میزبان منتقل می‌شود و میزبان می‌تواند ادامه کد خود را اجرا کند. باید توجه داشت اگر از این معماری برای پردازش استفاده می‌کنیم باید الگوریتم ما یا داده حجیمی داشته باشد یا این که محاسبات زیادی در الگوریتم انجام شود؛ در غیر این صورت، استفاده از این معماری به دلیل هزینه زمانی تبادل داده‌ای بین بُن‌سازهٔ میزبان و دستگاه، دیگر به صرفه نخواهد بود. برای اطلاعات بیشتر می‌توانید به [۱۷] مراجعه کنید.

معماری یک پردازنده گرافیکی در شکل ۳ نشان داده شده است. وقتی اجرا به‌دستگاه منتقل شد، کدی که در قسمت هسته نوشته شده است، برای هر ریسه کپی می‌شود. سپس هر ریسه اطلاعات مورد نیاز خود را از حافظه اصلی برداشته و بر روی حافظه محلی خود قرار داده و استفاده می‌کند. به این صورت، دستور یکسانی توسط ریسه‌های بسیار زیاد به‌صورت همزمان بر روی داده‌های متفاوت اجرا می‌شود.

۵-۱- پیاده‌سازی پیشنهادی برای ATPG

شکل ٤ معماری پیشنهادی برای ATPG را نشان می‌دهد. همان‌طور که مشاهده می‌شود، معماری پیشنهادی

26- Host
27- Device

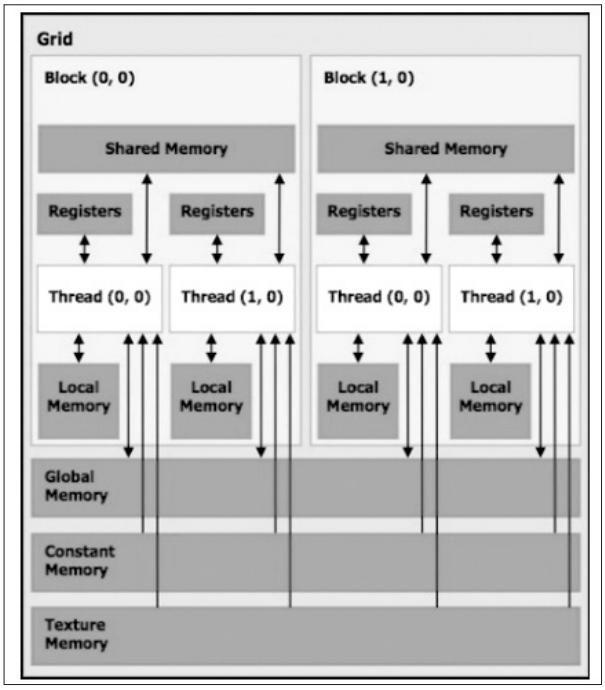
1	#include Here // CUDA kernel here
2	int main(int argc, char* argv[])
3	{
4	...Host(CPU) code Here
5	//Copy data from host(CPU)to device(GPU)
6	// Thread Configuration
7	// Launch the kernel
8	// Copy Data back to host(CPU)
9	...Other Host(CPU) code Here
10	}

شکل ۲ : ساختار کلی برنامه ATPG برای بهره‌گیری از GPU

از دو بخش میزبان و دستگاه تشکیل شده است.

در قسمت میزبان توصیفی (نت‌لیست) از مدار تحت آزمون خوانده شده و گرافی از آن به‌دست می‌آید. از مجموعه مدل‌های اشکال سطح منطق، متداول‌ترین مدل یعنی مدل اشکال چسبیده به صفر و چسبیده به یک را در نظر گرفته‌ایم. سپس، در نقاط مختلف مدار اشکال‌های مختلف را تزریق می‌کنیم. منظور از تزریق اشکال این است که نقطه‌ای را در مدار انتخاب می‌کنیم و یک اشکال چسبیده به یک یا چسبیده به صفر را به ترتیب با اتصال به منطق یک و یا اتصال به منطق صفر به آن اعمال می‌کنیم. در صورتی که با این تغییر، خروجی مدار نسبت به خروجی مدار سالم (بدون این تغییر) متفاوت بود، اشکال تزریق شده قابل کشف تلقی می‌شود و در غیر این صورت قابل شناسایی و کشف نخواهد بود [۱۸]. به این ترتیب، در مرحله بعد مدار سالم و مدار اشکال‌دار را XOR می‌کنیم. سپس مدار حاصل را به فرم CNF در می‌آوریم. این کار را برای سایر اشکال‌ها تکرار می‌کنیم و مجموعه‌ای از CNFها تشکیل می‌دهیم. سپس برای یافتن جواب یک CNF به سمت پردازنده گرافیکی ارسال می‌کنیم. در قسمت کد دستگاه، یک کرنل الگوریتم DPLL به همراه شهود یادگیری گزاره پیاده‌سازی شده است. همچنین برای انتخاب متغیر از ٤ روش مختلف استفاده شده است.

در قسمت دستگاه، گزاره‌های فرمول اصلی بین



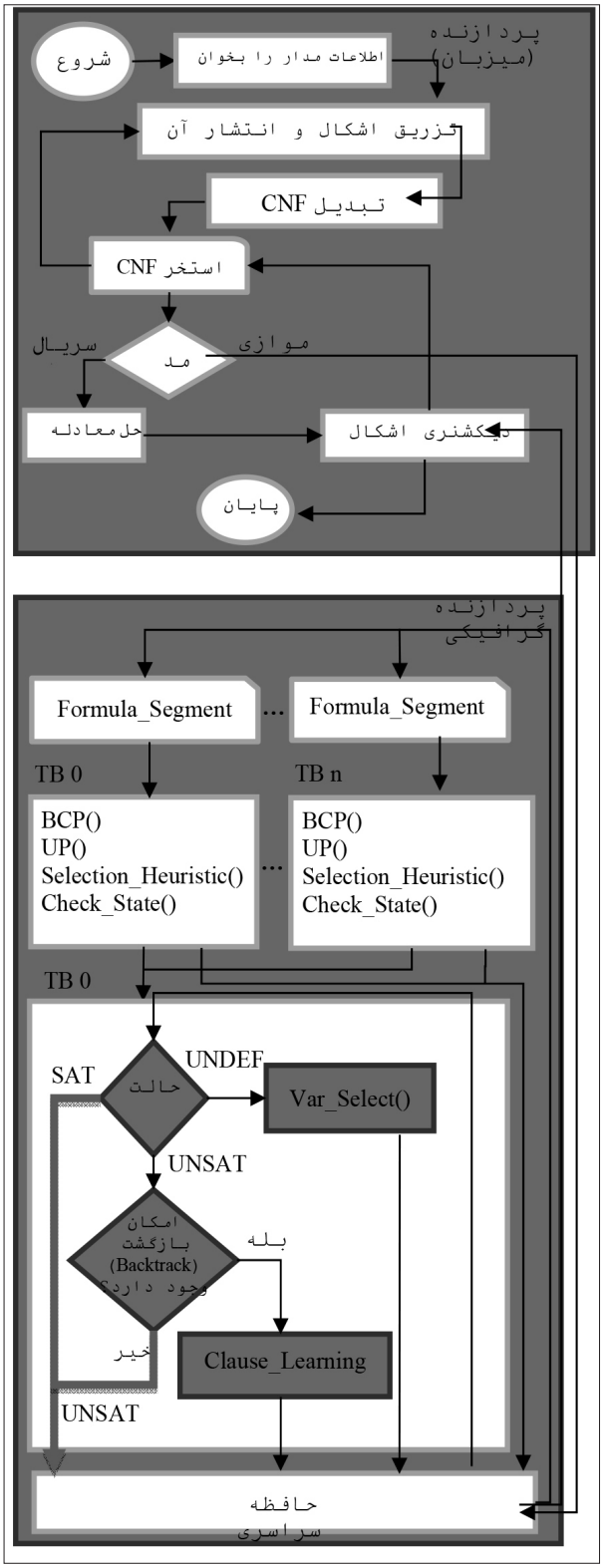
شکل ۳: معماری یک پردازنده گرافیکی [۲۰]

بلوک‌های پردازشی تقسیم می‌شود. این گزاره‌های تقسیم شده به‌صورت مجزا با استفاده از الگوریتم DPLL حل شده و وضعیت هر زیر بخش از فرمول اصلی را ذخیره می‌کنند. سپس، اجرا به بلوک اول منتقل شده و وضعیت کلی فرمول بر اساس وضعیت هر زیربخش حل شده در بلوک‌های مختلف، برآورد می‌گردد. اگر حاصل تمام گزاره‌های درون فرمول برابر منطق ۱ شود، به دلیل ترکیب عطفی بودن فرمول، کل فرمول برابر یک شده یا اصطلاحا ارضا می‌شود. در این مرحله مقادیر انتسابی به متغیرها که درون حافظه اصلی ذخیره شده است و باعث ارضا شدن فرمول شده است به میزبان منتقل می‌شود و یک فرهنگ لغات اشکال تشکیل داده می‌شود که در آن به ازای هر اشکال، مقادیر ورودی که منجر به شناسایی اشکال شده است، درج می‌شود. اما اگر وضعیت فرمول همچنان نامشخص باشد یعنی هنوز گزاره‌هایی وجود دارند که هنوز مقدار ۱ ندارند که منجر به نامشخص شدن جواب کل فرمول می‌شوند، متغیر دیگری توسط یکی از روش‌های انتخاب متغیر، انتخاب شده سپس مقداردهی می‌شود و دوباره فرمول تقسیم شده و بین بلوک‌های پردازشی GPU

پخش می‌شـــود و دوباره الگوریتـــم DPLL روی آن انجام می‌شـــود. در صورت ارضا نشـــدن فرمول، که در حالتی کـــه مقدار کلـــی فرمول برابر ۱ نمی‌شـــود و حالتی که یک متغیـــر همزمان دو مقدار متفـــاوت دارد اتفاق می‌افتد، باید به دنبال مراحل عقب‌گرد باشـــیم. یعنی اگر برای متغیری هنوز مقادیری موجود باشـــد که انتساب نداده‌ایم، به عقب برگشـــته و مقداردهی آن را به مقدار انتســـاب داده نشده تصحیح می‌کنیم و الگوریتم DPLL را دوباره اجرا می‌کنیم. در هنگام عقبگرد با بررسی فرمول و گزاره‌هایی که پس از چندین مرحله ساده شده‌اند و باعث ایجاد تناقض می‌شوند، با استفاده از تکنیک یادگیری گزاره به فرمول اصلی اضافه می‌شوند که در مراحل بعدی به‌صورت جلوتر از موعد از رسیدن به تناقض جلوگیری خواهد کرد. همچنین می‌تواند در انتخـــاب بهترین مرحله عقبگرد کمک کند. در هر مرحله انتســـاب متغیر، وضعیت جدیـــدی از فرمـــول در حافظه اصلی ذخیره می‌شـــود که می‌توان در زمان عقبگرد به آن وضعیت از فرمول و انتساب‌ها بازگشت نمود. اما اگر حالت عقبگردی موجود نباشد یعنی تمام حالت‌های قبلی فرمول بررسی شده و از حافظه اصلی خارج شده‌اند، تمام مقادیر ممکن به متغیرها انتســـاب داده شده است که به این معنی است فرمول ارضا شدنی نیست و جوابی در فرهنگ لغات اشکال ثبت نمی‌شود. این اشکال که برای آن هیچ انتسابی در فرمول وجود ندارد اصطلاحاً اشکال غیرقابل[۲۸] آزمون نامیده می‌شود. این روال تا زمانی تکرار می‌شود که تمام CNFهای تولید شده برای اشکالهای مختلف حل شوند و نتیجه ارضا شدن آنها در فرهنگ لغات اشکال ذخیره شود.

۶- نتایج آزمایش‌ها

قســـمت سریال روش پیشـــنهادی بر روی پردازنده ۵ هســـته‌ای[۲۹] و قســـمت موازی آن بر روی کارت گرافیک جی تی ایکس ۷۸۰[۳۰] به اجرا درآمده اســـت. برنامه بسیار



شکل ۴ : معماری پیشنهادی برای ATPG

وابســـته به محاسبات است؛ بنابراین مقدار حافظه اصلی[۳۱] تاثیر خاصی بر اجرای برنامه نخواهد داشت، هرچند حافظه

28-Untestable fault
29-core i5 4550
30-GTX780

31-Global Memory

جدول ۱ : زمان اجرای تولید الگوی آزمون برای روش‌های سری، موازی (پیشنهادی) و SATC. [۸] و میزان تسریع

	(#Gates,#In, #Out)	# Faults	Fault Coverage	Serial (sec.)	Parallel (sec.)	SAT Compress (sec.)	Speedup Serial/	Speedup /SATComp.
C432	(233,36,7)	524	99.2	13.99	2.73	14.46	5.1	5.2
C499	(638,41,32)	758	98.9	150.90	40.26	453.32	3.7	11.2
C880	(433,60,26)	942	100.0	39.89	7.60	162.4	5.2	21.4
C1355	(629,41,33)	1566	99.4	412.71	66.58	439.06	6.2	6.6
C1908	(425,33,25)	1879	99.4	487.90	84.19	883.75	5.8	10.5
C2670	(872,157,64)	2747	95.7	1132.57	225.46	2959	5.0	13.1
C3450	(901,50,22)	3428	96.0	12118.96	5492.25	7514.31	2.2	1.4
C7552	(2171,207,108)	7550	98.2	6766.52	1565.27	11195.8	4.3	7.2
AVG.			98.3				4.6	9.5

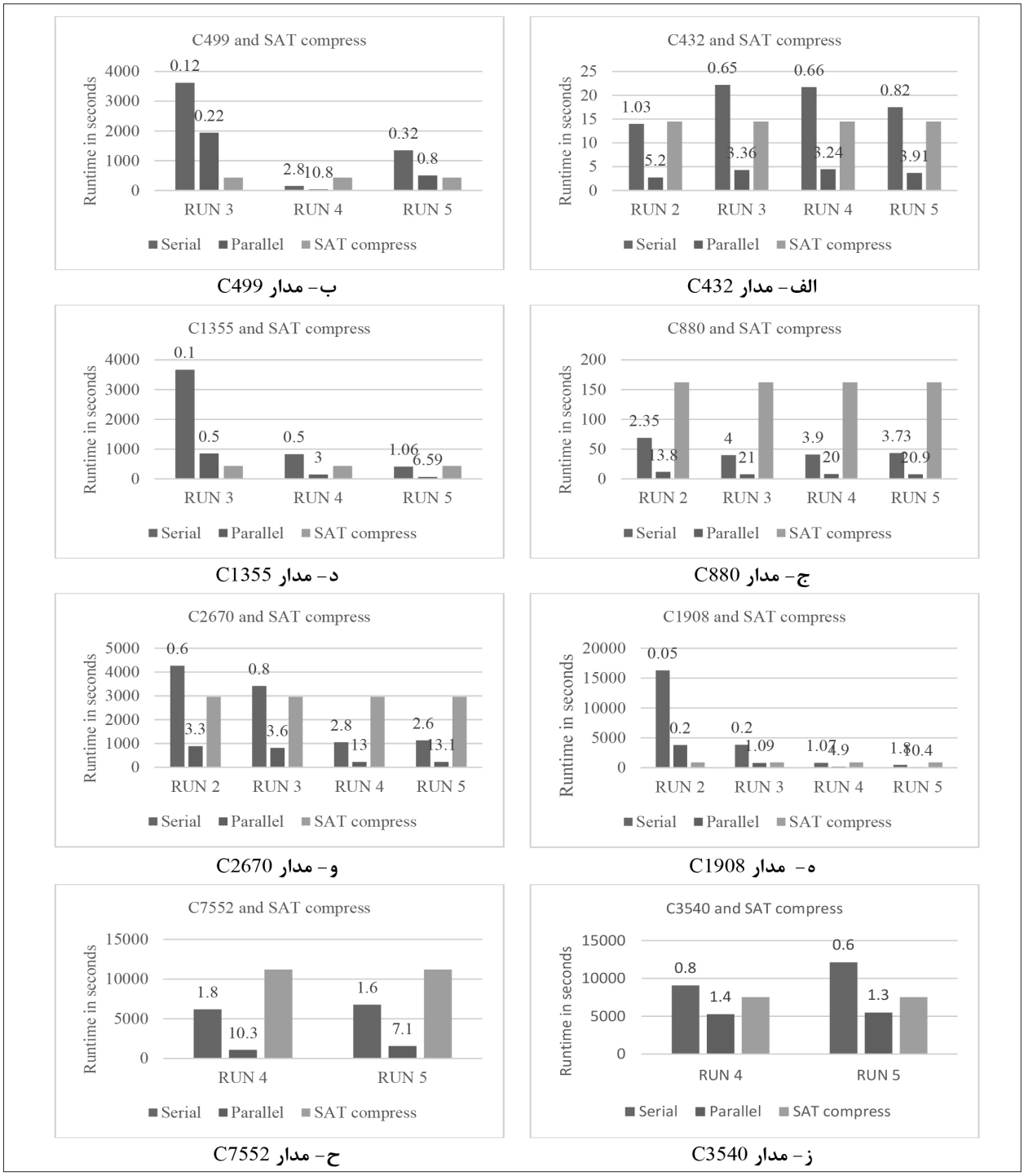
اصلی کامپیوتر ۸ گیگابایت و همچنین حافظه اصلی کارت گرافیک ٤ گیگابایت در نظر گرفته شده است. این روش بر روی مدارهای محک مختلف از مجموعه ISCAS'85 [۱۹] اعمال شده است.

به منظور مقایسه تولید الگوی آزمون در سه حالت مختلف در نظر گرفته می‌شود: ۱) ردیفی (Serial) که به حالتی اشاره می کند که الگوریتم تولید الگوی آزمون تماماً به‌صورت ردیفی و روی پردازنده ۵ هسته ای اجرا شده است، ۲) موازی (Parallel) که روش پیشنهادی در این مقاله است، ۳) روش SAT compress [۸].

جدول ۱ زمان اجرای الگوریتم مبتنی بر ارضاپذیری بولی در سه حالت در نظر گرفته شده و همچنین تسریع به‌دست آمده از روش پیشنهادی نسبت به دو روش دیگر را نشان می‌دهد. در این جدول، دو ستون سمت چپ بیانگر نام مدار و مشخصات آن (تعداد دروازه‌ها، تعداد ورودی‌ها، تعداد خروجی‌ها) می‌باشد. دو ستون بعدی، تعداد اشکال‌های تزریق شده و درصد پوشش اشکال (که برای هر سه روش یکسان است) گزارش شده است. سه ستون بعدی، به ترتیب، زمان اجرای الگوریتم تولید الگوی آزمون در حالات، ردیفی، روش پیشنهادی و روش SAT compress را در واحد ثانیه بیان می‌کند. در ستون هشتم، میزان تسریع به‌دست آمده از روش پیشنهادی نسبت به روش سری و در ستون نهم، میزان تسریع به‌دست آمده نسبت به روش SAT compress گزارش شده است. برای محاسبه این میزان تسریع، نسبت زمان اجرای الگوریتم در روش مدنظر به زمان اجرای الگوریتم با روش پیشنهادی در نظر گرفته می‌شود. بهترین بهبود سرعت نسبت به روش SAT compress بیش از ۲۱ برابر در مدار C880 و به‌طور میانگین نسبت به این روش، بیش از ۹ برابر بهبود سرعت داشتیم.

شکل۵ زمان اجرای الگوریتم تولید الگوی آزمون در روش های مختلف برای هر مدار با رویه‌های مختلف انتخاب متغیر آورده شده است. منظور از انتخاب متغیر آن است که هنگام ارضای یک گزاره، متغیرها بایستی به تدریج مقداردهی شوند و با توجه به این مقداردهی، حل مسئله ارضای گزاره در یک مسیر مشخصی ادامه پیدا می کند. به این منظور، انتخاب متغیر با راهبردهای مختلفی قابل انجام هستند. در بعضی اجراها به دلیل بیش از اندازه طولانی شدن زمان اجرا با یک روش انتخاب متغیر، نتایج آن روش نشان داده نشده است. بر روی هر ستون میزان بهبود سرعت نسبت به سرعت الگوریتم SAT compress نوشته شده است.

در شکل ۵-ج مربوط به مدار C499 زمان اجرا نسبت به اندازه مدار بسیار بالاست. دلیل این اتفاق بزرگ شدن اندازه گزاره‌های تولید شده در فرآیند تبدیل مدار به CNF است. به همین دلیل در هر گزاره تعداد زیادی متغیر برای



شکل ۵: مقایسه زمان اجرای الگوریتم تولید الگوی آزمون با لحاظ روش‌های مختلف انتخاب متغیر

انتخاب وجود دارد که روش انتخاب متغیر بسیار در زمان اجرا اثر دارد، به ایـن صورت که در یک گزاره متغیرهای زیادی برای انتخاب و مقدار دهی وجود دارد اما با توجه به بزرگ بودن فضای مسئله (ناشی از تعداد زیاد متغیرهایی که می توانند مقداردهی شـوند) احتمال رسیدن به جواب پایین بوده و فرآیند حل با تعداد بسیار زیادی عقبگرد برای هر متغیر روبرو خواهد شـد و آن گزاره به سختی ارضا شـده و از کل فرمول حذف می‌شود. در نتیجه این فرآیند، حل مسـئله به میزان قابل توجهی طولانی خواهد شـد. به همین دلیل، اگر اندازه گزاره کوچک‌تر باشد گزاره راحت‌تر

از فرمول اصلی حذف می‌شـــود. بنابراین، باید نحوه تبدیل مدار به CNF را بهینه کرد تا تعداد کمی متغیر در هر گزاره وجود داشته باشد. این کار را می‌توان با معرفی متغیرهای کمکی انجام داد و گزاره‌های بزرگ را به گزاره‌های کوچک شکســـت که در نتیجه طول هر گزاره کوچک‌تر ولی تعداد گزاره‌ها بیشـــتر می‌شـــود. انجام این کار به‌عنوان یکی از کارهای آینده این مقاله قابل طرح می‌باشد.

۷- نتیجه‌گیری و کارهای آینده

با وجود پوشش اشکال بالا در الگوریتم‌های مبتنی بر ارضاپذیری بولی، اجرای این الگوریتم‌ها روی پردازنده‌های ســـنتی بسیار زمانبر می‌شـــود که نقطه ضعف اصلی این الگوریتم‌ها تلقی می‌شود. در این مقاله، اجرای یک الگوریتم تولید خودکار الگـــوی آزمون مبتنی بر ارضا پذیری بولی تســـریع شده است. به این منظور، از توان پردازش موازی که در پردازنده‌های گرافیکی وجود دارد، استفاده نموده ایم. در این روش، در مقایســـه با یکی از الگوریتم‌های معروف مبتنی بر ارضاپذیری بولی، بیش از ۲۱ برابر ســـریع‌تر و به‌طور میانگین بیش از ۹ برابر بهبود سرعت داشتیم.

کارهـــای مختلفی را می‌تـــوان در ادامه ایـــن کار مطرح نمود. می توان از یک ســـو، بـــا روش های دیگری که برای حل ارضای گزاره های بولی تا کنون پیشـــنهاد شده است، اجرای این الگوریتم‌ها را ســـرعت داد و از ســـوی دیگر، با اعمال بهینه‌ســـازی در کد و اضافه کردن سایر روش‌های خلاقانه در انتخاب متغیرها، امکان افزایش تسریع بیشتر را فراهم نمود.

مراجع

[1] J. P. Roth, “Diagnosis of automata failures: a calculus and a method,” IBM Res Dev, pp. 278-281, 1966.

[2] S. A. Cook, “The complexity of theorem proving procedures,” ACM symposium on theory of computing, pp. 151-158, 1971.

[3] J. Shi, G. Fey, R. Drechsler, A. Glowatz, and F. Hapke, “PASSAT:efficient SAT-based test pattern generation,” Proceedings of the IEEE annual symposium on VLSI, pp. 187-196, 2005.

[4] W. Pan, F. Zheng, Y. Zhao, W. T. Zhu and J. Jing, “An Efficient Elliptic Curve Cryptography Signature Server with GPU Acceleration,” in *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 111-122, Jan. 2017

[5] Shijie Lia, Yong Doua, Xin Niua, Qi Lv, Qiang Wang, “A fast and memory saved GPU acceleration algorithm of convolutional neural networks for target detection”, Neurocomputing, Volume 230, pp. 48-59, 2017.

[6]James Sweet, David H. Richter, Douglas Thain, “GPU acceleration of Eulerian–Lagrangian particle-laden turbulent flow simulations”, International Journal of Multiphase Flow, pp. 437-445, 2018

[7] Martin Lukac, Georgiy Krylov, “Study of GPU Acceleration in Genetic Algorithms for Quantum Circuit Synthesis”, Multiple-Valued Logic (ISMVL), 2017 IEEE 47th International Symposium on, pp.1-6

[8] J. Balcarek, P. Fiser, and J. Schmidt, “Test patterns compression technique based on a dedicated SAT-based ATPG,” In Digital System Design: Architectures, Methods and Tools (DSD) 13th Euromicro Conference, pp. 805-808, 2009

[9] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. Reddy and B. Becker, “Thread parallel integrated test pattern generator utilizing satisfiability analysis,” Int J Parallel Progr, pp. 185-202, 2012.

[10] H. Marijn , M. Järvisalo and A. Biere, “Efficient CNF simplification based on binary implication graphs,” International Conference on Theory and Applications of Satisfiability Testing, pp. 201-215, 2011.

[11] P. Jackson and D. Sheridan, “Clause form conversions for boolean circuits,” International Conference on Theory and Applications of Satisfiability Testing, Berlin Heidelberg, 2004.

[12] G. Tseitin, On the complexity of derivation in propositional calculus, Berlin Heidelberg: Springer , 1983.

[13]L. T, “Test pattern generation using boolean satisfiability,” IEEE Transactions on Computer-Aided Design, pp. 4-15, 1992.

[14] M. Davis and H. Putnam, “A computing procedure for quantification theory,” Journal of the ACM (JACM), Vol. 7, No. 3, pp. 201-215, 1960.

[15] S. Eggersglüß and R. Drechsler, High Quality Test Pattern Generation and Boolean Satisfiability, Berlin: Springer Science & Business Media, 2012.

[16] P. Beame, H. Kautz and A. Sabharwal, “Towards understanding and harnessing the potential of clause learning,” Journal of Artificial Intelligence Research, Vol. 22, N0. 1, pp. 319-351, 2004.

[17] J. Sanders and E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional, 2010.

[18] M. Bushnell و V. Agrawal, Essentials of electronic testing for digital,memory and mixed-signal VLSI circuits, Boston, USA: Kluwer, 2000.

[19] M. C. Hansen, H. Yalcin and J. P. Hayes, “Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering,” IEEE Design & Test , Vol. 16, No. 3, pp. 72-80, 1999.

[20] M. Harris, Optimizing Parallel Reduction in CUDA, NVIDIAWhitepaper, http://www.nvidia.com/object/cuda/sample/dat/a%20-%20parallel.html