ON-LINE KONTROLA TOKU PROGRAMU VO VNORENÝCH SYSTÉMOCH

Jaroslav Abaffy
Počítačové systémy a siete, prvý ročník, denné štúdium
Školiteľ: doc. Ing. Tibor Krajčovič, PhD.

Fakulta informatiky a informačných technológií STU v Bratislave
Ilkovičova 3, 842 16 Bratislava 4

abaffy@fiit.stuba.sk

Abstrakt. Článok sa zaoberá prehľadom existujúcich riešení kontroly toku programu vo vnorených systémoch. Slúži ako zhrnutie doterajších poznatkov získaných v rámci doktorandského štúdia a v závere predkladá možné ďalšie smerovanie výskumu v problémovej oblasti.

Kľúčové slová. Občasné poruchy, Spoľahlivosť, Doba odozvy, Vnorený systém, Tok programu, On-line kontrola.

1 Úvod

Medzi najdôležitejšie požiadavky na vnorený systém patria požiadavky na prácu v reálnom čase, nízka energetická náročnosť, veľmi často aj požiadavky na veľkosť a tvar výsledného systému a v neposlednom rade cena výsledného produktu [1]. Pri vnorených systémoch je ale najpotrebnejšia vlastnosť ich odolnosť voči poruchám a zabezpečenie spoľahlivosti, pričom treba nesmierne dbať aj na ostatné spomenuté požiadavky.

V rámci dizertačnej práce sa budeme zaoberať metódami na zvýšenie spoľahlivosti týchto systémov, avšak s veľkým dôrazom na zachovanie ostatných požiadaviek kladených na vnorený systém, čo predstavuje netriviálny problém z toho hľadiska, že tieto vlastnosti sú si navzájom konkurujúce a zlepšenie jednej je vždy na úkor ostatných. Pri návrhu vnorených systémov sa preto musí brať do úvahy miera kompromisov, ktorú je možné akceptovať.

Ukázalo sa, že občasné a prechodné poruchy tvoria nezanedbateľnú časť porúch vo vnorených systémoch (až do výšky 70% [3]). Zistenie ich vplyvu však vyžaduje, aby sa systém nachádzal vo svojom pracovnom prostredí a kontrola sa vykonávala súbežne s behom aplikačného programu. Dodatočné kontrolné procesy nesmú znížiť výpočtový výkon systému natoľko, aby nebola dodržaná doba odozvy.

2 Občasné poruchy

Kozmické žiarenie, elektromagnetický šum, radiácia a ďalšie vplyvy prostredia spôsobujú zmeny informácií v systémoch, či už v pamätiach, polovodičoch alebo na prenosovom kanáli. Fyzické poškodenie je pamäte alebo kanálu je veľmi zriedkavé, oveľa častejšie dochádza k dočasnej poruche konkrétnej informácie. Pri zistení takejto chyby je možné opraviť dáta správnymi údajmi a prípadne napraviť aj spôsobenú chybu v behu programu, avšak najväčším problémom je práve detekcia takýchto porúch a určenie správnej metódy nápravy.

Špecialitou týchto porúch je, že spôsobujú poruchu aj inak odladeného systému pri zanedbaní vplyvov reálneho prostredia. Často je v laboratórnych podmienkach nemožné nasimulovať pracovné prostredie, do ktorého bude systém vnorený, a táto skutočnosť býva neraz podceňovaná.

Paradoxom je, že pokrok v zlepšovaní výrobných technológií znižuje odolnosť systémov na tieto javy. To, či bude mať dopadajúca častica vplyv na súčiastku závisí predovšetkým od jej relatívnej energie k danému polovodiču, ale aj od smeru a uhlu dopadu. Logicky, čím väčšia častica a väčšou silou narazí, tým skôr dôjde k preklopeniu stavu pamäťovej bunky, pričom je dôležitý aj smer. Naopak, čím je napríklad pamäťová bunka väčšia, tým menší vplyv bude táto častica mať. Taktiež čím väčšou silou je stav tejto bunky udržiavaný, tým viac je odolná voči poruchám [2].

Parameter	Late 1980's	1992	1995	1998	2001	2004
Supply voltage (V)						
High performance	5	5/3.3	3.3/2.5	2.5/1.8	1.5	1.2
Low power	-	3.3/2.5	2.5/1.5	1.5/1.2	1.0	1.0
Lithog. resolution (µm)	1.25	0.8	0.5	0.35	0.25	0.18
Channel length (µm)	0.9	0.6/0.45	0.35/0.25	0.2/0.15	0.1	0.07
Gate oxide thickness (nm)	23	15/12	9/7	6/5	3.5	2.5
Relative density	1.0	2.5	6.3	12.8	25	48
Relative speed						
High performance	1.0	1.4/2.0	2.7/3.4	4.2/5.1	7.2	9.6
Low power	-	1.0/1.6	2.0/2.4	3.2/3.5	4.5	7.2

Tabuľka 1: Vývoj RAM pamätí [11]

Za primárny zdroj prechodných porúch sa považuje kozmické žiarenie. Napriek tomu, že takmer žiadne častice kozmického žiarenia nedosiahnu zemský povrch, spôsobia v atmosfére vznik sekundárnych častíc, vysoko energetických neutrónov, ktoré sa považujú za príčinu 95% prechodných porúch. Aj keď neutróny samé o sebe nie sú elektricky nabité, napriek tomu spôsobujú zmeny v elektrických obvodoch. Práve vďaka tomu, že neutrón nenesie žiadny náboj, je vyššia pravdepodobnosť ako pri protóne, že pri náraze do jadra atómu s ním vytvorí nové ťažšie jadro a dôjde k nukleárnej reakcii, pri ktorej sú uvoľnené alfa častice [10]. Našťastie nie všetky neutróny zasiahnu jadro atómu nejakej súčiastky a aj v prípade nárazu často nemajú dostatočnú energiu alebo potrebný smer, takže pravdepodobnosť chyby sa v moderných RAM pamätiach pohybuje na úrovni jedného chybného bitu na 1 GB za mesiac [11].

3 Metódy kontroly toku programu

Na zvýšenie odolnosti systémov voči poruchám existuje niekoľko metód. Patrí medzi ne napríklad zvýšenie redundancie, a to buď hardvérová alebo softvérová redundancia. Typickým príkladom hardvérovej redundancie sú RAID polia, ECC pamäte alebo hardvérový triplex obvodov. Pri softvérovej redundancii dochádza k zvýšeniu spoľahlivosti replikáciou údajov, keď každý údaj je uložený viackrát, alebo pomocou diverzity algoritmov výpočtu. V tomto prípade sa sa k tomu istému výsledku dopracujeme viackrát, avšak vždy iným algoritmom. Obrovskou výhodou tohto riešenia je, že odhaľuje nielen chyby v hardvéri, ale taktiež programátorské a algoritmické chyby. Nevýhodou je však zložitá implementácia pri komplexných programoch a taktiež zvýšenie pamäťových a procesorových nárokov [7]. My sa budeme zaoberať metódami založenými na kontrole toku programu.

3.1 Enhanced Control-Flow Checking Using Assertions (ECCA)

Táto metóda je založená na vkladaní kontrolných značiek na začiatok a koniec programových blokov. Každému bloku je pridelený blokový identifikátor BID, ktorý by mal spĺňať dve podmienky. BID by malo byť prvočíslo väčšie ako 2 a zároveň musí byť jednoznačné pre každý programový blok. Bloku je potom priradená ďalšia premenná NEXT, ktorá obsahuje informácie o tom, ktoré ďalšie bloky môžu nasledovať po tomto bloku. Ďalšou premennou je id obsahujúce BID aktuálne vykonávaného bloku.

Každý blok je uzatvorený medzi dve kontrolné priradenia, pomocou ktorých sme schopní identifikovať chyby toku programu. Algoritmus tejto metódy by sa dal popísať v skratke takto:

1. Rozdeľ program na programové bloky.
2. Každému bloku priraď unikátne prvočíslo BID
3. Zostroj graf toku programu
4. Každému bloku priraď celočíselnú premennú NEXT obsahujúcu súčin BID všetkých blokov, ktoré môžu byť nasledovníkmi daného bloku.
5. Na začiatok bloku vlož prvé priradenie SET

$$id \leftarrow \frac{BID}{\overline{\overline{(id \bmod BID)} \cdot (id \bmod 2)}}$$

6. Na koniec bloku vlož priradenie TEST.

$$id \leftarrow NEXT + \overline{\overline{(id - BID)}}$$

Na začiatku bloku sa do premennej id nastaví hodnota aktuálneho bloku, na konci bloku sa do tejto premennej uloží hodnota NEXT aktuálneho bloku, a teda kombinácia všetkých možných nasledovníkov. Indikátorom nesprávneho toku programu je, že pri priradení SET dôjde k deleniu nulou, a to v prípade, že tento blok nie je nasledovníkom predchádzajúceho (id mod BID != 0) alebo id je párne číslo. BID sú prideľované tak, aby to boli prvočísla väčšie ako 2, a teda hodnota NEXT musí byť vždy nepárna. V priradení TEST sa nastavuje id na hodnotu NEXT, pričom ak id aktuálneho bloku je nerovné BID (id – BID == 0), tak je do tejto premennej uložené párne číslo a priradenie SET to vyhodnotí ako chybu [8].

3.2 Control-flow Checking by Software Signatures (CFCSS)

Na obdobnom princípe funguje aj CFCSS. Každému bloku priradí unikátnu značku, v tomto prípade nemusí ísť o prvočíslo, pričom ku bloku priradí aj rozdiel medzi značkou zdrojového a cieľového bloku. Kým ECCA na konci bloku určuje bloky, kam je možné skočiť, CFCSS problém rieši z opačného konca. CFCSS má na začiatku bloku kontrolu, či z daného zdrojového bloku je možné skočiť.

Algoritmus kontroly je potom jednoduchý. V globálnej premennej G sa nachádza značka aktuálneho bloku, d je rozdiel medzi zdrojovou a cieľovou značkou vypočítaný ešte v čase kompilácie. Počas behu programu sa urobí XOR medzi aktuálnou značkou v G a vypočítaným rozdielom medzi značkami d. V prípade správneho behu programu by výsledkom mala byť značka aktuálneho bloku. Ak tak nie je, je detekovaná chyba.

Výhodou tohto algoritmu oproti ECCA je, že má podstatne nižšiu výpočtovú aj pamäťovú náročnosť, pretože ECCA používa prvočísla a na kontrolu viacnásobné delenie, pričom CFCSS si vystačí s jedným vykonaním jednoduchej operácie XOR.

Problémom je, že ak je do bloku možné skočiť z viacerých zdrojových blokov, je potrebné do každého z týchto zdrojových blokov pridať opravnú značku D. V prípade, že zdrojové bloky zdieľajú viaceré cieľové bloky, dochádza k takzvanému aliasingu. Ak máme napríklad graf s hranami E={1,4; 1,5; 2,5; 2,6; 3,5; 3,6}, nepovolený skok z bloku 1 do bloku 6 nebude touto metódou detekovaný [6]

3.3 Control-flow Checking by Software Signatures – Interrupt count (CFCSS-IR)

Ďalšou metódou na zvýšenie spoľahlivosti je metóda založená na CFCSS, ktorá pridáva navyše počítadlo inštrukcií v bloku. V prípade, že dôjde k skoku v rámci jedného bloku alebo do stredu iného, počítadlo inštrukcií bude mať inú hodnotu než očakávanú [5].

3.4 Yet Another Control-Flow Checking using Assertions (YACCA)

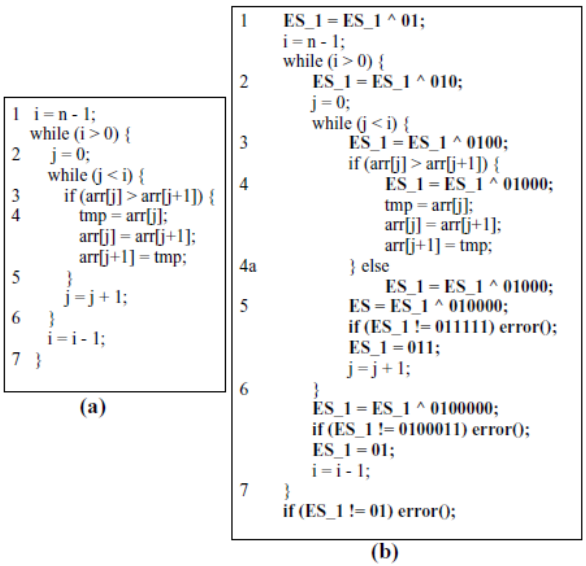
Je ďalšou metódou založenou na CFCSS s jediným rozdielom, že miesto ukladania rozdielu medzi jednotlivými značkami kontroluje zakaždým rozdiel značiek

code = (code & !(Bj ^ Bk)) ^ (Bj & !(Bj ^ Bk) ^ Bi)

V premennej code je uložená značka aktuálne vykonávaného bloku. Ak dôjde k skoku do bloku Bi, ktorý má predchodcov Bj a Bk, skontrolujú sa podľa vzorca tieto značky a ak nedošlo k chybe, tak do premennej CODE sa uloží značka Bi. Ak tak nie je, indikuje to chybu. Výhodou je zvýšenie odolnosti voči aliasingu známeho z CFCSS, nevýhodou zvýšenie výpočtovej zložitosti [9].

3.5 Assertions for Control Flow Checking (ACFC)

Táto metóda nepoužíva vzťahy medzi blokmi, ale každej funkcii programu prideľuje premennú nazvanú stav vykonávania (execution status). Lepšie ako z popisu algoritmu sa princíp pochopí z nasledujúceho obrázku popisujúceho zabezpečenie Bubble Sortu. Ide o to, že pri každom ďalšom vnorení sa do novej vetvy pridáva do kontrolného slova ES parita daného bloku pomocou XOR. jednotka. Ak sa z vetvy vychádza, tak sa kontroluje hodnota kontrolného slova, v prípade slučiek sa následne aj nastaví na hodnotu, ktorú malo na začiatku slučky.



Obrázok 1: Algoritmus vkladania značiek v prípade metódy ACFC

Myšlienka je to vcelku jednoduchá a je založená na princípe, že koľko úrovní sme sa vnorili, toľko sa musíme aj vynoriť. V prípade skoku mimo bude tým pádom hodnota kontrolného slova iná než očakávaná [13].

3.6 Kontrolný proces

Jednou z metód SIHFT (Software-Implemented Hardware Fault Tolerance) je aj implementácia watchdog časovača pomocou softvéru. Táto metóda je však vhodná len pre systémy zvládajúce

multitasking [15]. Miesto hardvérovej implementácie obvodu kontrolujúceho beh programu sa použije ďalší kontrolný proces, ktorý kontrolovaný proces neustále informuje o svojom behu. Kontrolný proces obsahuje graf toku údajov kontrolovaného procesu. Kontrolovaný proces posiela pri každom vnorení sa do programového bloku kontrolnému procesu informáciu, do ktorého sa vnoril. V prípade, že tento skok nie je povolený, kontrolný proces to vyhodnotí ako chybu toku programu.

3.7 Ďalšie metódy

Počas analýzy existujúcich prístupov sme sa stretli ešte aj s niekoľkými ďalšími metódami, ktoré fungujú na podobných princípoch ako predchádzajúce, väčšinou vychádzajúc z CFCSS. Čo sme si všimli, tak v popise takmer všetkých metódach (aj tých vyššie spomenutých) je uvedené porovnanie s ECCA a CFCSS, pričom ako východzia je zvolená CFCSS kvôli nižšej pamäťovej a výkonnostnej náročnosti. Do tejto sú potom pridané rôzne drobné zlepšenia a samozrejme vymyslený nový inovatívne znejúci názov. Obligátnym je nakoniec porovnanie s ECCA, voči ktorému vyjde konkrétna metóda ako jednoznačne výhodnejšia z hľadiska výkonnosti a samozrejme sa ukáže aj nárast pokrytia chýb v porovnaní s CFCSS. Pre úplnosť spomenieme ešte niektoré z metód, ktorými sme sa doteraz zaoberali.

- Control Flow Checking using Branch Trace Exceptions for PowerPC processors [14]. Táto metóda využíva špecialne inštrukcie debugovania obsiahnuté v inštrukčnej sade procesorov rodiny PowerPC. Bližšie sme sa ňou nezaoberali, pretože je závislá na konkrétnom hardvéri. Veľmi zaujímavá by bola pri analýze procesorov vhodných pre hard RT systém.
- Software-Based Error Detection Technique Using Encoded Signatures [16] je ďalšou metódou založenou na CFCSS, pričom ponúka algoritmus prideľovania značiek. V skratke, bloku sa pridelí číslo X, jeho nasledovníkom 2X a 2X+1. Ak má len jedného, je mu pridelené číslo 2X a 2X+1 je uložené do čiernej listiny. Ak nemá nasledovníkov, sú tam vložené čísla 2X aj 2X+1. Týmto spôsobom sú zabezpečené lepšie vzdialenosti medzi jednotlivými značkami a aj vzťahy medzi nimi.
- Control-Flow Checking Using Branch Instructions [4], táto metóda pochádza od autorov predchádzajúcej metódy. V tomto prípade použili kontrolu programových blokov zabezpečenie inštrukcií skokov s tým, že navyše pridali do skokových inštrukcií kontrolnú paritu.

4 Záver

V doterajšej práci sme sa zamerali predovšetkým na existujúce softvérové riešenia problému, nazývané tiež SIHFT (Software-Implemented Hardware-Fault Tolerance), aj napriek tomu, že hardvérové riešenie sprevádza síce často lepšia výkonnosť a lepšie dodržanie požadovaných maximálnych oneskorení [12]. Na druhej strane však spôsobuje niekoľkonásobné zvýšenie nákladov v porovnaní so softvérovým riešením.

V ďalšej fáze výskumu sa zameriame predovšetkým na možnosti ukladania a obnovy stavu procesu. Počas prvej fázy analýzy sme zistili, že existuje nespočetné množstvo viac, či menej, úspešných metód kontroly toku programu. Všetky tieto metódy sa zameriavajú na detekciu chýb v toku programu, ale neriešia problém vysporiadania sa s danou chybou. Typicky sa v takomto prípade proces reštartne. Tento prístup je veľmi výhodný pri malých procesoch, pri ktorých celková doba trvania behu programu nie je oveľa dlhšia ako samotné spustenie programu. Ak však uvažujeme procesy, ktoré bežia dlhší čas a opätovné spustenie by znamenalo stratu časti potrebných údajov, je vhodné zamyslieť sa nad možnosťami ukladania stavu procesu a následného reštartu z bodu obnovy.

Ako výsledok dizertačnej práce predpokladáme implementáciu vlastnej alebo niektorej zo spomínaných metód kontroly toku programu s vlastnými vylepšeniami obohatenú o možnosť obnovy stavu procesu, ako aj metodiku určovania únosnej miery kontroly toku procesu a ukladania stavu procesu. Zameriame sa na implementáciu týchto metód na viacprocesorových systémoch, ktoré postupne prenikajú aj do oblastí vnorených systémov [17].

Poďakovanie

Táto práca vznikla za podpory grantu číslo 1/0649/09 Vedeckej grantovej agentúry VEGA.

Zdroje

[1] Laplante, P. A. : Real-time Systems Design and Analysis. IEEE Press, New York, 339pp., 1993

[2] Mukkherjee, S. : Architecture Design for Soft Errors, Elsevier, 23pp., 2008

[3] Czech, E. W. , Siewiorek, D. : Effects of Transient Gate-level Faults on Program Behavior, International Symposium on Fault-Tolerant Computing, 236pp., 1990

[4] Jafari-Nodoushan, M., Miremadi, S. G., Ejlali, A. : Control-Flow Checking Using Branch Instructions, 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, vol 1., 66pp., 2008

[5] Wu, Y., Gu, G., Huang, S., and Ni, J. : Control Flow Checking Algorithm using Soft-based Intra-/Inter-block Assigned-Signature. In Proceedings of the Second international Multi-Symposiums on Computer and Computational Sciences, IEEE Computer Society, Washington, DC, 412pp., 2007

[6] Oh, N., Shirvani, P.P., McCluskey, E.J. : Control flow checking by software signatures, IEEE Transactions on Center for Reliable Computing Technical Report, Volume 51, Issue 1, 111pp., 2002

[7] Oh, N., Mitra, S., McCluskey, E. J.: ED4I: Error Detection by Diverse Data and Duplicated Instructions, IEEE Transactions on Computers, vol. 51, no. 2, 180pp, 2002

[8] Alkhalifa, Z., Nair, V. S. S., Krishnamurthy, N. , Abraham, J. A.: Design and evaluation of system-level checks for on-line control flow error detection, IEEE Transactions On Parallel and Distributed Systems, Vol. 10, No. 6, 1999

[9] Goloubeva, O., Rebaudengo, M., Sonza Reorda, M., Violante, M.: Soft-Error Detection Using Control Flow Assertions, IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 581pp., 2003

[10] Normand, E.: Single event upset at ground level, IEEE Transactions on Nuclear Science, Volume 43, Issue 6, 2742pp., 1996

[11] Johnston, A. H.: Scaling and Technology Issues for Soft Error Rates, In 4th Annual Research Conf. on Reliability, 2000

[12] Bernardi, P., Bolzani, L. M. V., Rebaudengo, M., Reorda, M. S., Vargas, F. L., Violante, M.: A New Hybrid Fault Detection Technique for Systems-on-a-Chip, IEEE Transactions on Computers, vol. 55, no. 2, pp. 185pp., 2006

[13] Venkatasubramanian, R., John P. Hayes, J.P. , Murray, B.T. : Low-Cost On-Line Fault Detection Using Control Flow Assertions, 9th IEEE International On-Line Testing Symposium, 137pp., 2003

[14] Fazeli, M., Farivar, R., Miremadi, S. G.: A Software-Based Concurrent Error Detection Technique for PowerPC Processor-based Embedded Systems, 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05), 266pp., 2005

[15] Majzik, I., Pataricza A.: Control Flow Checking in Multitasking Systems. Periodica Polytechnica Ser. Electrical Engineering, Vol. 39, No. 1, Technical University of Budapest, 27pp., 1995

[16] Sedaghat, Y., Miremadi, G. S., Fazeli, M.: A Software-Based Error Detection Technique Using Encoded Signatures, 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06), 389pp., 2006

[17] Abaffy, J.: Performance comparison of OS schedulers on symmetric multiprocessors, In Bieliková, M., ed.,: Proceedings in Informatics and Information Technologies Student Research Conference 2009, Vydavateľstvo STU, Bratislava, 277pp., 2009