Em busca de soluções em nível de sistema para tecnologias não confiáveis

Carlos A. L. Lisboa, Luigi Carro

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{calisboa, carro}@inf.ufrgs.br

Abstract. *New technologies, providing faster and smaller devices, will lead to embedded devices with billions of components. However, transient pulses caused by radiation will last longer than the clock cycle of the circuits, thereby precluding the use of several current mitigation techniques, such as those based on time redundancy. Traditional hardware level techniques using space redundancy usually impose heavy area and power penalties, unacceptable for embedded applications. As an alternative to cope with this challenging scenario, this paper proposes the use of application specific, algorithm level mitigation techniques, able to detect errors caused by transient faults with reduced performance overhead. Examples of such techniques for two applications widely used in the embedded systems world, matrix multiplication and sorting, are presented and compared with other techniques in order to confirm the potentiality of the proposed approach, and a path towards the generalization of the proposed approach is suggested.*

Resumo. *Novas tecnologias, que oferecem dispositivos mais rápidos e menores, viabilizarão a produção de sistemas embarcados com bilhões de componentes. Entretanto, pulsos transitórios provocados por radiação irão ter duração maior do que o ciclo de relógio dos circuitos, descartando em decorrência disto o uso de diversas técnicas de mitigação de erros atuais, tais como aquelas baseadas em redundância temporal. Técnicas tradicionais em nível de hardware e que usam redundância espacial normalmente impõem pesadas penalidades em termos de área e potência, características inaceitáveis para aplicações embarcadas. Como uma alternativa para enfrentar tal cenário desafiador, este trabalho propõe o uso de técnicas de mitigação em nível de algoritmo, específicas para cada aplicação, capazes de detectar erros causados por falhas transitórias com sobrecarga de desempenho reduzida. Exemplos de tais técnicas para duas aplicações amplamente utilizadas em sistemas embarcados, multiplicação de matrizes e classificação, são apresentadas e comparadas com outras técnicas, para confirmar a potencialidade da abordagem proposta, e um caminho para generalização da aplicação da abordagem proposta é sugerido.*

1. Introdução

Os dispositivos mais rápidos oferecidos por novas tecnologias permitem a construção de circuitos com tempos de ciclo cada vez mais curtos, enquanto a duração de pulsos transitórios devidos à radiação não varia na mesma proporção. Isto vai levar à

ocorrência de pulsos transientes que durarão mais do que o tempo de ciclo dos circuitos, impossibilitando o uso de diversas técnicas de mitigação de erros temporários de baixo custo, principalmente aquelas baseadas em redundância temporal, tais como [Anghel 2000, Mitra 2005].

O uso de redundância espacial como a solução preferencial para fazer frente a erros temporários neste novo cenário poderia ser uma alternativa possível. Entre estas, a redundância modular tripla (TMR) [Johnson 1994] é uma boa candidata, porque mesmo se um dos módulos é afetado por um transitório de longa duração, os outros dois módulos estarão trabalhando adequadamente em paralelo, assegurando resultados corretos. TMR tem um ponto fraco: falhas que afetem o votador podem levar a saídas com erro, e para evitar este problema o uso de um votador analógico foi proposto em [Schüler 2005, Lisboa 2005]. Entretanto, uma outra fraqueza da técnica TMR é que as penalidades que ela impõe, em termos de área e energia, são superiores a 200%, o que torna inviável seu uso em sistemas onde energia é um recurso escasso, tais como SoCs para sistemas embarcados.

Neste trabalho, propomos o uso de abordagens em nível de sistema para fazer frente aos desafios impostos pelos pulsos transitórios de longa duração (TLDs) previstos como um dos principais problemas para sistemas fabricados usando tecnologias futuras. Na Seção 2 é explicado o raciocínio que nos levou à previsão de TLDs. Na Seção 3, técnicas propostas para a mitigação de erros causados por falhas transientes são revisadas e as razões pelas quais algumas delas não terão mais eficácia neste novo cenário são discutidas. A Seção 4 propõe o uso de técnicas específicas para cada aplicação, implementadas em nível de algoritmo com possível intervenção do compilador, como uma abordagem possível para detectar erros causados por TLDs. Dois estudos de caso nos quais esta abordagem é usada são apresentados como exemplos e um caminho possível em direção à generalização para outros algoritmos da abordagem adotada é proposto. A Seção 5 resume as conclusões e indica trabalhos futuros nesta pesquisa.

2. Pulsos transitórios de longa duração (TLDs)

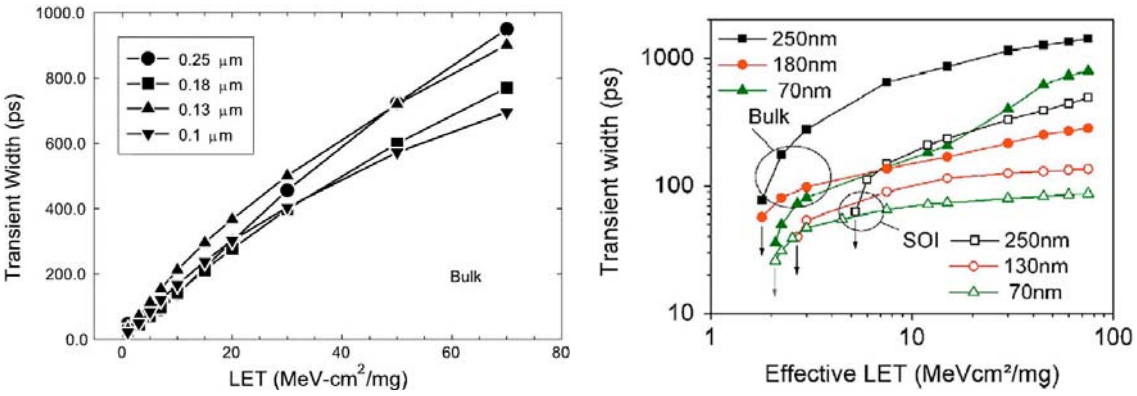
A variação da duração dos pulsos transitórios entre diversas tecnologias foi correlacionada com o tempo de ciclo dos circuitos em Lisboa 2007, que mostrou que para as futuras tecnologias, tais como a de 32 nm, pode-se prever que até mesmo partículas com valores de transferência linear de energia (LET) modestos irão produzir transitórios que durarão mais do que o tempo de ciclo previsto para os circuitos. Aquele estudo é resumidamente revisado e então complementado aqui, com a inclusão de dados mais recentes.

No estudo sobre a produção e propagação de SETs em lógica de alta velocidade apresentado em Dodd 2004, a variação da largura do pulso transitório de acordo com a tecnologia foi prevista para quatro diferentes tecnologias (0,25 μm, 0,18 μm, 0,13 μm e 0,1 μm) e os resultados obtidos para partículas com LET variando até 70 MeV-cm^2/mg, para processos com substrato (*bulk)*, são mostrados na Figura 1(a).

Além do fato já esperado de que a largura do pulso aumenta com o aumento da transferência de energia linear (LET) da partícula, este gráfico revela outras informações importantes. Primeiramente, verifica-se que, para partículas de baixa

energia, a largura do pulso transitório é quase a mesma para todas as quatro tecnologias incluídas no estudo, ou seja, apesar da maior velocidade dos novos dispositivos, para partículas com aquele nível de energia a duração dos transitórios não varia de forma significativa de uma tecnologia para outra.

Além disso, analisando o lado direito da Figura 1(a) e extraindo os valores de largura de pulso do gráfico, pode-se verificar que a variação máxima na largura dos pulsos transitórios (para partículas com LET de 70 MeV-cm^2/mg) é de 948 ps (para a tecnologia de 0,25 μm) para 694 ps (para a tecnologia de 0,1 μm), ou seja, um fator máximo de redução de 0,73. Para o mesmo valor de LET, a variação da largura do pulso transitório entre as tecnologias de 0,18 μm e 0,13 μm é de 902 ps para 769 ps, com um fator de redução de apenas 0,85. Em contraste, para as mesmas tecnologias de 0,18 μm e 0,13 μm, o fator de redução do atraso de propagação de um inversor entre estas tecnologias, medido através de simulação, é de aproximadamente 0,33 (ver Tabela 2). Estes dados mostram claramente que a redução do atraso de propagação entre tecnologias é muito mais agressiva do que a da largura do pulso transitório provocado por radiação e, portanto, é muito provável que uma falha transitória poderá durar mais do que um ciclo de relógio, como será mostrado adiante.



(a) **(b)**

Figura 1. Variação da largura de pulsos transitórios entre diferentes tecnologias

Mais recentemente, em Ferlet-Cavrois 2006, a largura do pulso transitório que se propaga, para dispositivos fabricados usando processos com substrato e com silício sobre isolante (SOI), em diferentes tecnologias, foi medida através de simulação usando uma cadeia de inversores em série, com resultados semelhantes, mostrados na Figura 1(b). Este trabalho analisou somente três tecnologias (250 nm, 180 nm e 70 nm) e também não fez correlação com a variação do tempo de ciclo nem explorou as conseqüências desta descoberta.

Tabela 1. Larguras de transitórios prevista (ps)

Tecnologia (nm)	180[(1)]	130[(1)]	100[(1)]	70[(2)]
10 MeV-cm^2/mg	140	210	168	170
20 MeV-cm^2/mg	277	369	300	240

(1) dados de Dodd 2004 (2) dados de Ferlet-Cavrois 2006

Alguns valores de largura de transitórios foram extraídos manualmente das Figuras 1(a) e 1(b) e reunidos na Tabela 1, que mostra as larguras de transitórios previstas para as tecnologias de 180 nm, 130 nm e 100 nm estudadas em Dodd 2004 e para a tecnologia de 70 nm estudada em Ferlet-Cavrois 2006, para partículas com LET de 10 e 20 MeV-cm^2/mg, respectivamente.

Para comparar a variação das larguras de transientes mostradas na Tabela 1 com aquela do tempo de ciclo dos circuitos através de diferentes tecnologias, medimos o atraso de propagação para diversas cadeias de inversores em série, em quatro tecnologias distintas. As medições foram feitas usando a ferramenta HSPICE [Synopsis URL] e os parâmetros para todas as tecnologias foram obtidos do sítio na Web de Predictive Technology Model [PTM URL]. O número de inversores em cada cadeia, bem como os atrasos de propagação medidos na simulação para cada tecnologia estão mostrados na Tabela 2.

Tabela 2. Variação do tempo de propagação entre diferentes tecnologias (ps)

Tecnologia (nm)	180	130	90	32
1 inversor	49,85	16,78	13,26	10,14
cadeia de 4 inversores	202,65	63,81	48,93	33,74
cadeia de 6 inversores	304,55	95,14	72,66	49,02
cadeia de 8 inversores	406,45	126,45	96,39	64,30
cadeia de 10 inversores	508,35	157,75	120,15	79,58

Os resultados na Tabela 2 mostram que o atraso de propagação é reduzido em aproximadamente 3,2 vezes da tecnologia 0,18 μm para a tecnologia 0,13 μm e de 5 a 6,4 vezes, dependendo do números de inversores na cadeia, entre as tecnologias 0,18 μm e 32 nm. Estes fatores de redução são muito maiores do que a redução máxima prevista em [7, 8] para a largura dos pulsos transitórios entre tecnologias diferentes, analisada acima, que é de apenas 1,37 vezes.

Na Figura 2, as larguras previstas para pulsos transitórios para as quatro tecnologias mostradas na Tabela 1 foram combinadas com o gráfico de linhas que mostra a redução do atraso de propagação entre tecnologias. As duas barras verticais mais à esquerda mostram a largura do pulso transitório prevista para partículas com LET de 20 MeV-cm^2/mg, para as tecnologias 0,18 μm e 0,13 μm.

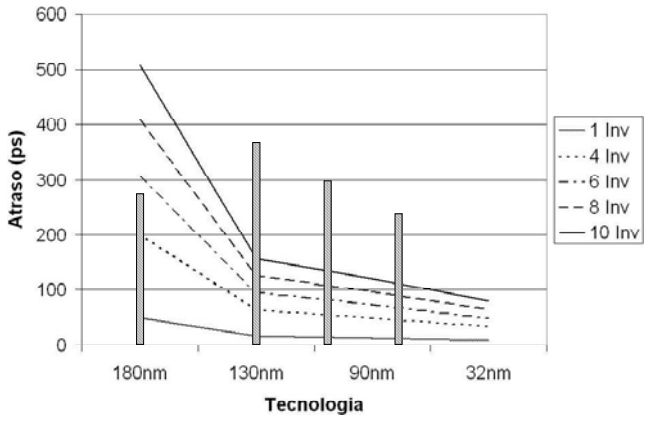


Figura 2. Variação do atraso de propagação x largura de pulsos transitórios entre diferentes tecnologias

A terceira e a quarta barras verticais correspondem às tecnologias 0,1 μm e 70 nm, para as quais não medimos os atrasos de propagação. O posicionamento horizontal destas duas barras no gráfico foi feito por interpolação, considerando que a variação do atraso da tecnologia de 0,13 μm para a de 32 nm é aproximadamente linear.

A duração de transitórios devidos a partículas com LET de até 20 MeV-cm^2/mg, mostrada na Figura 2, é suficiente para demonstrar o que pretendemos – nas tecnologias futuras os transitórios irão durar mais do que o ciclo de relógio esperado; entretanto, como se pode ver pelas Figuras 1(a) e 1(b), podem ocorrer transitórios de duração bem maior (acima de 1000 ps) para partículas com energia mais elevada.

As conclusões tiradas da Figura 2 se aplicam somente a dispositivos fabricados em processos com substrato, e são mencionadas somente de forma implícita em Dodd 2004, quando os autores afirmam que, para dispositivos com silício sobre isolante, "a variação do tamanho dos dispositivos claramente induz um decréscimo das distribuições tanto da carga coletada quanto da largura do transitório", enquanto "se considerarmos dispositivos com substrato, não existem uma tendência óbvia de variação".

O fato do fator de variação da largura do pulso entre tecnologias ser muito menor do que o fator de aceleração da velocidade entre diferentes gerações de tecnologias aponta para a necessidade um novo paradigma a ser considerado no projeto de sistemas tolerantes a falhas para tecnologias futuras: pulsos transientes durando mais de um ciclo de relógio dos circuitos. Entretanto, como é mostrado na próxima seção, a maioria das técnicas existentes para mitigação de erros temporários não é capaz de suportar tais TLDs.

3. Revisão de técnicas de mitigação de erros temporários

Muitas técnicas diferentes para mitigar erros temporários foram propostas nos últimos anos. Elas podem ser grosseiramente classificadas como segue: técnicas baseadas em hardware (usando redundância temporal, redundância espacial e circuitos verificadores ou I-IPs), técnicas baseadas em software (usando duplicação ou verificação de assinaturas) e técnicas híbridas.

Técnicas baseadas em hardware usando redundância temporal fazem a verificação das saídas do circuito através da comparação de seus valores em instantes diferentes separados por um intervalo de tempo fixo. Na medida em que a duração dos pulsos transitórios aumenta, a duração deste intervalo, que é uma penalidade imposta a cada ciclo de operação, implicará em sobrecargas de desempenho intoleráveis. Exemplos destas técnicas são [Anghel 2000, Mitra 2005]. Também em [Austin 2004] o mesmo conceito é usado para verificar as saídas de um circuito e calibrar a taxa de erros temporários através do ajuste dinâmico da voltagem, visando reduzir o consumo. A aplicação desta técnica também será afetada de forma negativa pela ocorrência de TLDs.

O grupo de técnicas baseadas em redundância espacial é o que mais provavelmente oferecerá proteção, mesmo na presença de TLDs, porque, considerando o modelo de falha única (que ainda é o modelo dominante [Rossi 2005]), somente uma das cópias do circuito seria afetada pelo TLD, enquanto a(s) outra(s) forneceria(m) resultados corretos. No caso da duplicação com comparação [Lima 2003], isto permitiria a detecção do erro causado pelo TLD. No caso da redundância modular

tripla, além de detectar o erro o circuito seria capaz de escolher o resultado correto, descartando o resultado errado causado pelo TLD. Uma outra abordagem que se baseia no modelo de falhas simples, na qual o caminho crítico de circuitos combinacionais é robustecido através da duplicação de portas lógicas e erros temporários são detectados através da comparação das saídas duplicadas, é proposta em [Nieuwland 2006]. Entretanto, as penalidades em acréscimo de área e, principalmente, em aumento de consumo impostas por estas soluções são uma grande preocupação, principalmente para sistemas embarcados.

O terceiro grupo de técnicas baseadas em hardware se apoia no uso de circuitos verificadores ou IPs de infra-estrutura (I-IPs) para verificar os resultados produzidos pelo circuito a ser protegido. Caso os resultados calculados pelo circuito verificador ou pelo I-IP sejam diferentes daqueles produzidos pelo circuito principal, eles ativam um indicador de erro que dispara um processo de recomputação ou, como em [Austin 1999], usam o valor calculado pelo verificador, considerando que este está sempre correto (o que é uma suposição arriscada). Estas abordagens usualmente implicam em uma sobrecarga de desempenho elevada, já que os resultados precisam ser calculados duas vezes, e também em sobrecarga de área, devido à adição do verificador ou I-IP. Mesmo em soluções onde parte da verificação é feita em paralelo com o circuito principal, tais como em [Lisboa 2006], a sobrecarga de desempenho ainda é significativa.

Técnicas baseadas em software que duplicam o código e as variáveis e comparam os resultados para verificar se houve erros são muito caras, tanto em ocupação de memória quanto em tempo de execução. As técnicas baseadas em assinaturas requerem a modificação do software para incluir instruções para processamento e verificação das assinaturas em todos o blocos básicos, impondo penalidades em codificação e desempenho.

Técnicas híbridas procuram combinar as melhores características das técnicas baseadas em hardware e software, com o objetivo de obter o melhor balanceamento entre sobrecargas de área e de desempenho.

Devido às limitações das técnicas de mitigação discutidas acima, nosso trabalho propõe o uso de uma abordagem diferente, na qual a verificação e a recuperação de erros é feita em nível de algoritmo, como é discutido na seção seguinte. Como será mostrado, as principais vantagens da técnica aqui proposta são a sobrecarga de desempenho reduzida, com dissipação adicional de potência mínima, e boa adequação ao tamanho do problema.

4. Abordagens específicas para cada aplicação para detectar TLDs

A abordagem proposta neste trabalho é baseada nas seguintes considerações sobre a natureza dos erros transitórios provocados por radiação e os processos de detecção e correção associados:

- Se por um lado os efeitos do impacto de uma partícula contra um circuito podem ser muito prejudiciais, e portanto não podem ser ignorados, a freqüência de tais eventos é muito baixa, e nem todos os impactos de partículas provocam um erro; por exemplo, até bem pouco tempo (tecnologias até 100 nm), as taxas de erros temporários para circuitos lógicos

costumavam ser desprezíveis, em comparação com a taxa de falhas em dispositivos de memória, e para um sistema em um só circuito integrado (SoC) que usasse memórias com uma taxa de erros de 10.000 FIT/Mbit a taxa de erros do sistema seria de aproximadamente um erro por semana [Heijmen 2002].

- Devido à importância da detecção de erros, o mecanismo de detecção, seja ele implementado em software ou hardware, precisa verificar continuamente os resultados gerados pelo circuito a ser protegido. Isto significa que o circuito de detecção será executado vários milhões de vezes até que um erro temporário seja detectado e, portanto, deve ser o mais leve possível, em termos de sobrecarga de área, desempenho e consumo de potência.
- Em contraste, o mecanismo de recuperação de erro será ativado somente quando um erro for detectado, o que acontece muito raramente, e portanto as sobrecargas de desempenho e de potência dinâmica introduzidas por este mecanismo não são a preocupação principal. No entanto, ainda é importante que as sobrecargas de área e potência impostas por este mecanismo sejam minimizadas, já que estamos voltados para aplicações em sistemas embarcados.

Considerando os fatos acima, nossa proposta é trabalhar em nível de algoritmo, usando esquemas de verificação específicos para cada aplicação, com pequena sobrecarga, para detectar a ocorrência de erros temporários. Embora os mecanismos de detecção de erros propostos nos estudos de caso na seção 4 sejam implementados em software, nossa abordagem não exclui o uso de mecanismos de detecção baseados em hardware, como será comentado mais adiante.

Quanto à sobrecarga aceitável, em nossa abordagem ela é claramente dependente da aplicação. No entanto, nosso objetivo é definir mecanismos de verificação que permitam a verificação da correção dos resultados produzidos por um determinado algoritmo em tempo significativamente menor do que aquele necessário para executar o algoritmo completamente mais uma vez. Se assim não fosse, uma solução direta e mais simples seria executar o algoritmo duas vezes e comparar os resultados, uma tarefa que pode inclusive ser automatizada, como mostrado em [Cheynet 2000]. Como se pode ver nas próximas seções, para os dois estudos de caso apresentados como exemplos este objetivo foi atingido com uma folga razoável para as aplicações propostas, e acreditamos que o caminho indicado na subseção 4.3 irá levar a resultados semelhantes para outras aplicações.

4.1. Verificação de um algoritmo de classificação: o uso de invariantes de laço

Como um primeiro exemplo de aplicação da abordagem aqui proposta, usaremos uma aplicação de classificação para destacar os conceitos básicos de nossa técnica. Entre os muitos algoritmos de classificação diferentes disponíveis na literatura, usaremos o *quicksort* [Knuth 1973], que faz a classificação através do particionamento do conjunto de registros e troca de posições. Este algoritmo é mais adequado para grandes conjuntos de registros e usualmente é capaz de ordenar um conjunto de n registros em tempo proporcional a $O(n \log n)$; no entanto, em algumas situações específicas, tal como

quando o conjunto já está previamente ordenado quando o algoritmo inicia, pode levar um tempo de até $O(n^2)$ [Knuth 1973].

De acordo com o objetivo por nós definido, o esquema de detecção de erro precisa ser tal que o tempo necessário para verificar os resultados seja significativamente menor do que aquele necessário para uma recomputação completa dos resultados. Neste caso específico, existe um esquema de detecção muito direto, que demanda apenas um tempo $O(n)$: comparar as chaves dos registros ordenados, na ordem final fornecida pelo algoritmo de classificação, para verificar se eles estão efetivamente ordenados.

De forma a confirmar que o mecanismo de detecção é realmente mais rápido do que o algoritmo de classificação, executamos o algoritmo *quicksort* proposto em [Knuth 1973] e também o mecanismo de detecção sugerido aqui e comparamos os resultados, em termos do número de comparações e trocas necessárias para ordenar conjuntos com diferentes quantidades de registros com chaves geradas aleatoriamente, como mostrado na Tabela 3. O algoritmo de verificação usa somente n-1 comparações para verificar o vetor ordenado.

Tabela 3. Custo de execução dos algoritmos: classificação vs. verificação

n	Comparações	Trocas	Total Classificação	Total Verificação	Sobrecarga (%)
100	653	340	993	99	9,97
200	2.154	968	3.122	199	6,37
500	7.104	2.924	10.028	499	4,98
1.000	17.546	8.051	25.597	999	3,90

Embora o custo total de execução do *quicksort* varie, para a mesma quantidade de registros a ordenar, de acordo com a posição inicial dos registros no conjunto, analisando os números obtidos pode-se confirmar que a sobrecarga imposta pelo mecanismo de verificação é muito menor do que o custo de execução do processo de classificação completo, e também que a percentagem desta sobrecarga diminui à medida em que o número de registros a ordenar cresce, o que também é uma característica do custo de execução do algoritmo *quicksort*.

A generalização do procedimento mostrado aqui para outros exemplos será discutida na seção 4.3.

4.2. Verificação de um algoritmo para multiplicação de matrizes: otimizando a técnica de Freivalds

Nosso segundo estudo de caso se aplica à verificação de multiplicação de matrizes, um algoritmo que é freqüentemente utilizado como parte de aplicações embarcadas, tais como filtros de uso geral, mecanismos de controle, compressão de imagens, filtros de imagens e compressão de voz, entre outras. O número de operações necessárias para multiplicar duas matrizes quadradas $n \times n$ é $O(n^3)$. De acordo com a regra fundamental de nossa abordagem aqui proposta, precisamos agora encontrar um processo de verificação que precise de menos operações para ser executado. Infelizmente, para algoritmos de multiplicação de matrizes e outros o processo de verificação não é tão intuitivo como para o estudo de caso de classificação.

Uma técnica para verificação rápida da correção de algoritmos para multiplicação de matrizes é apresentada em [Motwani 1995]. Ela é creditada a Rúsiņš Freivalds, que provou que máquinas probabilísticas são capazes de executar algumas computações específicas mais rapidamente que as determinísticas e que elas podem computar aproximações de uma função em uma fração do tempo necessário para computar a mesma função de forma determinística [Freivalds 1977].

Em suma, a técnica de Freivalds propõe o uso de multiplicação de matrizes por vetores a fim de reduzir o tempo de computação quando se está verificando os resultados produzidos por um determinado algoritmo de multiplicação de matrizes, da seguinte maneira: dadas duas matrizes, A e B, de dimensões *n x n* e a matriz C, de mesmas dimensões, que é o produto de A por B que foi calculado usando o algoritmo a ser testado, são executados os seguintes cálculos:

1. Cria-se aleatoriamente um vetor *r* no qual os valores dos elementos são unicamente 0 ou 1.
2. Calcula-se Cr = C × r
3. Calcula-se ABr = A × (B x r)

Freivalds provou que, sempre que A × B ≠ C, a probabilidade de Cr ser igual a ABr é menor ou igual a ½. A demonstração é apresentada em [Motwani 1995]. Além disso, se os passos 1 a 3, acima, são executados independentemente *k* vezes (com diferentes valores do vetor *r*), a probabilidade se torna $\leq ½^k$. Usando essa técnica, a verificação do resultado pode ser feita em menos tempo do que a multiplicação original, já que a multiplicação de uma matriz por um vetor é executada com $O(n^2)$ operações, o que poderia torná-la uma boa candidata para ser usada como o mecanismo de verificação para multiplicação de matrizes, de acordo com abordagem aqui proposta. Entretanto, como a técnica de Freivalds é estatística, não há garantia de que os erros sempre serão detectados.

Em um trabalho anterior Lisboa 2007, mostramos que a probabilidade de detectar um elemento errado na matriz C usando a técnica de Freivalds é aproximadamente ½ porque os elementos do vetor *r* gerados aleatoriamente têm a mesma probabilidade ½ de ser 0 ou 1, e supondo que o elemento de C que tem um valor incorreto é C_{ij}, no cálculo de Cr (passo 2 acima) este elemento é multiplicado por um único elemento r_k do vetor, sendo assim cancelado durante a geração de Cr (se r_k é igual a zero) ou não (quando r_k é igual a 1) com probabilidades iguais de ½.

Considerando que os elementos do vetor *r* podem ser escolhidos aleatoriamente, propusemos então executar a computação com um segundo vetor, r_c, no qual cada elemento é o complemento binário dos valores em *r*. De acordo com a técnica original de Freivalds, isto levaria à detecção de um erro em C com uma probabilidade de ¼. os elementos de C que fossem cancelados na primeira computação não seriam cancelados na segunda e vice versa. Por isso, se C_{ij} tivesse um valor incorreto, teríamos necessariamente A × (B × r) ≠ C × r ou A × (B × r_c) ≠ C × r_c e a probabilidade de detectar um erro em um único elemento de C seria igual a 1, isto é, se o valor incorreto fosse mascarado no cálculo de ABr/Cr, ele não seria mascarado quando ABr_c/Cr_c fossem calculados e vice versa. Com esta extensão, produzimos um algoritmo capaz de

detectar todos os erros nos quais um único elemento de C está errado, com apenas duas execuções da técnica de Freivalds.

Indo mais a fundo na exploração da extensão da técnica de Freivalds proposta em Lisboa 2007, argumentamos que, como a técnica é válida para qualquer vetor *r* selecionado aleatoriamente, também é válida para um vetor específico $r_1 = \{1, 1, ..., 1\}$ e seu vetor complementar $r_0 = \{0, 0, ..., 0\}$, e neste caso temos:

$$C \times r_1 = \{\Sigma^n_{j=1} C_{1j}, ..., \Sigma^n_{j=1} C_{nj}\} \quad (1)$$

$$A \times (B \times r_1) = \{\Sigma^n_{j=1} (\Sigma^n_{k=1} A_{1k} . B_{kj}), ..., \Sigma^n_{j=1} (\Sigma^n_{k=1} A_{nk} . B_{kj})\} \quad (2)$$

e

$$C \times r_0 = 0 \quad (3)$$

$$A \times (B \times r_0) = 0 \quad (4)$$

A partir das expressões (3) e (4) acima, concluímos que, para o vetor r_0 dado, a condição $C \times r_0 \neq A \times (B \times r_0)$ é sempre falsa (pois os dois produtos são sempre iguais a zero) e, portanto, o teste da condição composta $A \times (B \times r_1) \neq C \times r_1$ ou $A \times (B \times r_0) \neq C \times r_0$ pode ser simplificado para $A \times (B \times r_1) \neq C \times r_1$, reduzindo significativamente o custo do processo de verificação, pois o cálculo das expressões (3) e (4) não é mais necessário. Além disso, no cálculo de (1) e (2) não há mais necessidade de multiplicar pelos elementos de r_1, já que estes são todos iguais a 1.

A partir de (1) e (2), também concluímos que, como no processo de multiplicação calculamos o valor de cada elemento da matriz C como sendo $C_{ij} = \Sigma^n_{k=1} A_{ik} . B_{kj}$, se um dos elementos C_{ij} tem um valor incorreto, a condição $A \times (B \times r_1) \neq C \times r_1$ será verdadeira e o erro sempre será detectado.

As conclusões acima foram confirmadas através da execução de diversas séries de inserção de falhas com o uso de simulações em Matlab [Matlab URL] e concluímos que esta técnica otimizada oferece um método capaz de detectar todos erros em um único elemento ocorridos na operação de multiplicação de matrizes, com pequena sobrecarga, razão pela qual foi escolhida como nosso algoritmo de verificação para a multiplicação de matrizes no espírito da abordagem aqui proposta. A Tabela 4 mostra o número de operações (adições e multiplicações) necessárias para multiplicar e verificar matrizes com diferentes dimensões, conforme descrito em Lisboa 2007, e tais números confirmam que a sobrecarga de desempenho fica bem abaixo do limite de 100% que nós mesmos impusemos para a abordagem proposta.

Tabela 4. Comparação de tempos de execução (número de operações)

Tamanho da matriz (*n*)	3	8	16	32
Multiplicação	45	960	7.936	64.512
Verificação	27	232	976	4.000
Sobrecarga	60%	24%	12%	6%

4.3. Generalização formal da abordagem: uso de invariantes de programa para detectar TLDs

Como mostrado nos estudos de caso anteriores, a seleção de um método de verificação para um determinado algoritmo nem sempre á uma tarefa fácil. Além disso, uma vez

encontrado um mecanismo de verificação adequado, ainda que seja significativamente mais rápido do que o algoritmo a ser verificado, como pode-se ter certeza de que este é o mecanismo mais eficiente? Idealmente, seria melhor dispor de uma ferramenta automatizada que, dado o algoritmo a ser protegido, pudesse fornecer o melhor mecanismo de verificação para a aplicação dada. Entretanto, tanto quanto saibamos, não existe nenhuma ferramenta como esta disponível atualmente.

Também é importante destacar que os dois esquemas de verificação propostos em nossos estudos de caso verificam os resultados produzidos pelo algoritmo somente depois deles terem sido executados completamente. Isto significa que, no caso de um erro ser detectado durante o processo de verificação, o estado do programa precisa ser restaurado para o mesmo estado em que ele se encontrava no início da execução do algoritmo e este precisa ser executado por completo novamente, de forma a obter um novo resultado. Com isto em mente, uma outra característica desejável do mecanismo de verificação seria a capacidade de detectar um erro assim que ele ocorre, reduzindo desta forma o tempo de recomputação no caso de erros. No entanto, como já mencionado, apesar desta característica ser desejável ela não é obrigatória, já que a freqüência de recomputação em virtude de erros temporários provocados por radiação é muito baixa.

Na busca por uma solução para o problema da determinação de um algoritmo de verificação adequado, e tomando emprestados alguns conhecimentos da área de computação formal, acreditamos que a identificação de invariantes de programas ou invariantes de laços pode ser uma boa alternativa para encontrar o melhor mecanismo de verificação a ser usado para uma determinada aplicação.

Uma invariante de laço expressa uma relação entre os valores das variáveis manipuladas no corpo do laço que é preservada pela execução correta do mesmo. Em outras palavras, se a relação expressa pela invariante do laço é verdadeira antes que corpo do laço seja executado e o laço termina, ela também será verdadeira após o laço ter sido terminado, não importando quantas vezes o corpo do laço seja repetido. Existem diversas invariantes para um determinado laço, mas uma invariante útil é aquela que expressa uma relação entre as variáveis que é preservada pelo laço, mesmo que os valores das variáveis propriamente ditas possam mudar [Huth 2001]. Analogamente, uma invariante de um programa em uma determinada posição é uma asserção que é verdadeira para qualquer estado do programa em que ele atinge aquela posição, e a asserção é dita indutiva em uma determinada posição de um programa se ela é verdadeira na primeira vez em que aquela posição é atingida pelo fluxo de execução do programa e é preservada todas as vezes em que o fluxo de controle retorna para aquela posição. Asserções indutivas tradicionalmente têm sido usadas para provar que os programas estão corretos [Sankaranarayanan 2004].

Dadas as definições acima, pode-se ver que o que fizemos de maneira intuitiva em nosso estudo de caso de classificação foi verificar a seguinte invariante, na posição do programa na qual o algoritmo termina, depois de classificar os n registros em ordem ascendente de suas chaves:

$$\text{chave}_i \leq \text{chave}_{i+1},\ 1 \leq i \leq n\text{-}1$$

De maneira similar, no estudo de caso de multiplicação de matrizes, a relação

$$A \times (B \times r_1) = C \times r_1$$

é uma invariante do algoritmo na posição em que a multiplicação está terminada. Nos dois casos, quando a relação não se mantém, isto é, as invariantes são falsas, é uma indicação de que o algoritmo não foi executado corretamente.

Como já mencionado, em uma determinada posição de um programa pode-se ter mais de uma invariante, mas nem todas elas são úteis para fins de verificação. Portanto, além da capacidade de identificar as invariantes possíveis em um determinado ponto do programa, também é preciso saber quais dentre elas são úteis. Um outro aspecto fundamental na seleção da melhor invariante é a existência de um mecanismo de baixo custo para verificá-la. Por exemplo, o programa Matlab mostrado a seguir, adaptado de [Huth 2001], calcula o fatorial de um valor armazenado na variável *x*:

```
y=1;
z=0;
while z ~= x,
   z = z + 1;
   y = y * z;
end
```

Considerando que 0! = 1 e que, quando a execução do laço termina temos z = x e y contém o fatorial de x calculado, a invariante do laço neste exemplo pode facilmente ser identificada como:

$$y = z!$$

Entretanto, esta invariante não é adequada para uso na verificação de acordo com a abordagem aqui proposta, porque seria necessário calcular novamente o valor de z! para poder compará-lo com y e confirmar a correção do resultado.

Como se pode ver, a generalização de um método para determinar o melhor mecanismo de verificação para um dado algoritmo não é uma tarefa simples. Embora existam ferramentas para inferir de forma automática invariantes de laço [Ernst 2001, Daikon URL], o uso de uma ferramenta nem sempre assegura o sucesso e, mesmo após descobrir a invariante, a obtenção do verificador não é uma tarefa simples. Esta questão, portanto, demandará investigações adicionais e é um dos principais passos futuros em nosso trabalho de pesquisa.

4.4. Considerações sobre a implementação em hardware de mecanismos de verificação

O uso de hardware para implementar mecanismos de verificação de algoritmos tais como aqueles propostos neste trabalho é uma alternativa a ser mais explorada.

Entretanto, é preciso ter em mente que, enquanto a implementação em software requer somente a adição de uma porção extra de código ao software de aplicação, a implementação em hardware requer acesso ao projeto do hardware, o qual, mesmo em aplicações embarcadas, pode nem sempre estar disponível. Como um exemplo, em sistemas que usam processadores disponíveis comercialmente (COTs) não é possível incorporar em hardware os recursos de controle e processamento necessários para implementar um esquema de verificação tal como aquele proposto para o algoritmo de classificação na subseção 4.1. Por outro lado, para um SoC no qual é utilizado hardware

dedicado para uma determinada operação e o projetista tem acesso à descrição do hardware dos circuitos, é relativamente fácil adicionar os circuitos necessários para implementar esquemas de verificação tais como aquele proposto para a operação de multiplicação de matrizes.

Portanto, a escolha entre mecanismos de verificação implementados em hardware ou software é uma decisão que muito provavelmente será tomada pelo projetista caso a caso, e também será objeto de pesquisa adicional em nossos trabalhos futuros.

5. Conclusões e trabalhos futuros

Através da análise da evolução entre diferentes tecnologias da largura de pulsos transitórios causados por radiação e do atraso de propagação dos circuitos, este trabalho mostrou que futuras tecnologias estarão sujeitas a pulsos transitórios que durarão mais do que o ciclo de relógio dos circuitos, aqui denominados TLDs, e que isto vai impossibilitar a aplicação de diversas técnicas de mitigação de erros temporários tradicionais.

O uso de mecanismos de verificação em nível de algoritmo, específicos para cada aplicação, foi proposto como uma forma de enfrentar este novo desafio e dois estudos de caso, com os mecanismos de detecção de erros correspondentes, foram apresentados e discutidos, como exemplos de aplicação da abordagem proposta.

Foi proposta a generalização formal da técnica, através do uso de invariantes de programas para verificar os resultados produzidos por algoritmos, e foram indicadas diretrizes para pesquisa futura como sendo:

- Estudo de invariantes de programas e invariantes de laços, com o objetivo de definir uma metodologia genérica a ser usada na seleção do melhor mecanismo de verificação a ser adotado para uma determinada aplicação.
- Estudo dos benefícios e desvantagens de implementar mecanismos que permitam a verificação parcial dos resultados durante a execução dos algoritmos, considerando que, se por um lado o custo de executar novamente todo o algoritmo quando for detectado um erro é alto, por outro a freqüência de detecção de erros temporários é muito baixa.
- Definição de um conjunto de regras a serem seguidas durante o processo de escolha entre implementação em hardware ou em software dos mecanismos de verificação.

Referências

Anghel, L. and Nicolaidis, M., “Cost Reduction and Evaluation of a Temporary Faults Detecting Technique”, in Proceedings of the Design, Automation and Test in Europe Conference and Exhibits – DATE 2000, IEEE Computer Society, Paris, France, March 27-30, 2000, pp 591-598.

Austin, T., “DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design”. In *MICRO32 - Proceedings of the 32nd ACM/IEEE International*

Symposium on Microarchitecture, pages 196-207, Los Alamitos, CA, November, 1999.

Austin, T., Blaauw, D., Mudge, T., and Flautner, K., "Making typical silicon matter with Razor", IEEE Computer, Vol. 37, No. 3, IEEE Computer Society, Los Alamitos, March 2004, pp 57-65.

Cheynet, P., Nicolescu, B., Velazco, R., Rebaudengo, M., Sonza Reorda, M., and Violante, M., "Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors", *IEEE Transactions on Nuclear Science*, Vol. 47, No. 6 (part 3), December 2000, pp. 2231-2236.

Daikon: invariant detector tool, available at http://pag.csail.mit.edu/daikon, last accessed April 15, 2007.

Dodd, P. E. et al., "Production and propagation of Single-Event Transients in High-Speed Digital Logic ICs", IEEE Transactions on Nuclear Science, Vol 51, No 6, Part 2, IEEE Computer Society, Los Alamitos, CA, December 2004, pp 3278-3284.

Ernst, M. D., Cockrell, J., and Griswold, W. G., "Dynamically Discovering Likely Program Invariants to Support Program Evolution", IEEE Transactions on Software Engineering, Vol. 27, No. 2, IEEE Computer Society, New York, NY, February 2001, pp 99-123.

Ferlet-Cavrois, V. et all., "Statistical Analysis of the Charge Collected in SOI and Bulk Devices Under Heavy Ion and Proton Irradiation—Implications for Digital SETs", IEEE Transactions on Nuclear Science, Vol 53, No 6, Part 1, IEEE Computer Society, Los Alamitos, CA, December 2006, pp 3242-3252.

Freivalds, R. "Probabilistic machines can use less running time", in *Proceedings of IFIP Congress 77*, B. Gilchrist, editor, North-Holland, Toronto, August 1977, pp 839-842.

Heijmen, T., "Radiation Induced Soft Errors in Digital Circuits: A Literature survey". Philips Electronics National Laboratory, Netherlands, Report 2002/828, August, 2002.

Huth, M. R. A. and Ryan, M. D., *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge University Press, Cambridge, UK, 2001.

Johnson, B. W., *Design and Analysis of Fault Tolerant Digital Systems: Solutions Manual*, Addison-Wesley Publishing Company, Reading, MA, October 1994.

Knuth, D. E., *The Art of Computer Programming: Volume 3 / Sorting and Searching*, Addison-Wesley Publishing Company, Reading, MA, 1973.

Lima, F., Carro, L. and Reis, R., "Techniques for Reconfigurable Logic Applications: Designing Fault Tolerant Systems into SRAM-based FPGAs", in *Proceedings of the International Design Automation Conference, DAC 2003*, pp. 650-655, ACM, New York, 2003.

Lisboa, C. A. and Carro, L., "System Level Approaches for Mitigation of Long Duration Transient Faults in Future Technologies", to be published in Proceedings of the 18th Symposium on Integrated Circuits and Systems Design – ETS 2007, May 20- 24, 2007.

Lisboa, C. A. L., Carro, L., Sonza Reorda, M., and Violante, M. "Online Hardening of Programs against SEUs and SETs", in *Proceedings of the 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems - DFT 2006*, IEEE Computer Society, Los Alamitos, CA, October 2006, pp. 280-288.

Lisboa, C. A., Schüler, E., and Carro, L., "Going Beyond TMR for Protection Against Multiple Faults", in Proceedings of the 18th Symposium on Integrated Circuits and Systems Design – SBCCI 2005, Sociedade Brasileira de Computação, Florianópolis, September 4-7 2005, pp. 80-85.

Matlab web page: http://www.mathworks.com/products/matlab. Last visited April 15, 2007.

Mitra, S., Seifert, N., Zhang, M., Shi, Q., and Kim, K. S., "Robust system design with built-in soft-error resilience", in Computer, Vol. 38, No. 2, February 2005, pp. 43-52.

Motwani, R., and Raghavan, P., *Randomized Algorithms*, Cambridge University Press, New York, 1995.

Nieuwland, A. Jasarevic, S. and Jerin, G., "Combinational Logic Soft Error Analysis and Protection", in Proceedings of the 12th IEEE International On-Line Test Symposium – IOLTS 2006, IEEE Computer Society, Los Alamitos, CA, July 2006, pp. 99-104.

Predictive Technology Model web site, last visited in April, 15 2007: http://www.eas.asu.edu/~ptm

Rossi, D., Omaña, M., Toma, F. and Metra, C., "Multiple Transient Faults in Logic: An Issue for Next Generation ICs ?", in Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2005), IEEE Computer Society, Los Alamitos, CA, October 2005, pp. 352-360.

Sankaranarayanan, S., Sipma, H. B., and Manna, Z. "Non-linear Loop Invariant Generation using Gröbner Bases", in Proceedings of the 31st ACM SIGPLAN-SIGACT Annual Symposium on Principles of Programming Languages, ACM Press, New York, NY, January 2004, pp 318-329.

Schüler, E., and Carro, L., "Reliable Circuits Design Using Analog Components", in Proceedings of the 11th Annual IEEE International Mixed-Signals Testing Workshop – IMSTW 2005, Volume 1, IEEE Computer Society, Cannes, June 27-29, 2005, pp 166-170.

Synopsis web site: http://www.synopsys.com/products/mixedsignal/hspice/hspice.html. Last visited in November 2006.

Agradecimento

Os autores agradecem a Lorenzo Petroli, Bolsista IC e aluno do curso de Engenharia de Computação da UFRGS, pela colaboração no preparo deste manuscrito.