Técnica Híbrida Não Intrusiva para Detectar SEEs em Microprocessadores

Maurício Altieri, José Rodrigo Azambuja, Fernanda Lima Kastensmidt

Instituto de Informática – Universidade Federal do Rio Grande do Sul

Contato: mascarpato@inf.ufrgs.br, jrfazambuja@gmail.com, fglima@inf.ufrgs.br

INTRODUÇÃO

Os recentes avanços na indústria de semicondutores possibilitaram um aumento exponencial de desempenho dos microprocessadores, mas também os deixaram mais sensíveis à interferência da radiação. Portanto, aplicações de alta confiança, como espaciais, necessitam de técnicas de tolerância a falhas que possam evitar ou recuperar o sistema após a ocorrência de um erro, sendo elas baseadas em redundância de software ou de hardware.

TÉCNICA PROPOSTA

O programa é dividido em **BB**s (blocos básicos). Um grafo orientado é gerado onde cada *BB* é um nodo e cada transição entre *BBs* é uma aresta. Um registrador **S** (não usado no programa) é reservado para a técnica, o qual será atualizado durante a execução sempre que o programa entrar ou sair de um *BB*. Para cada *BB*, 3 assinaturas são geradas: **NIS**, valor de *S* ao entrar, **NES**, ao sair, e **NS**, dentro do *BB* (resultado do *XOR* de todas suas instruções). Os *BBs* são divididos em 2 tipos:

- **A**, quando possui mais de 1 pai e pelo menos um deles possui mais de 1 filho;
- **X**, quando possui só 1 pai ou é o único filho de seus pais.

No caso dos *BBs* tipo *X*, a determinação do *NIS* é simples, ele será igual ao *NES* do seu pai. Já com os tipo *A*, é necessário mascarar um ou mais bits da assinatura antes, de modo que o *NES* de todos seus pais sejam aceitos, como mostra a Fig. 1.

Original	Protegido
beq r1,r2, 6	1: beq r1, r2, 6
add r2, r3, 1 ***Tipo X***	**2: xor S, constante** 3: add r2, r3, 1 **4: store S** **5: xor S, constante**
add r2, r3, 4 ***Tipo A***	**6: and S, constante** **7: xor S, constante** 8: add r2, r3, 4 **9: store S** **10: xor S, constante**
jmp end	11: jmp end

Fig. 3: Código protegido

RESULTADOS

Mais de 40 mil falhas foram injetadas, 1 por execução, em cada uma das 2 aplicações, e, em ambas, ocorreram em torno de 5 mil erros no fluxo de controle. Como mostra a Tab. 1, a técnica proposta obteve **100% de detecção.**

Tab. 1: Resultado da Injeção de Falhas

Aplicação	Falhas Injetadas	Erros Ocorridos	Falhas Detectadas
Multiplicação de Matrizes	41.000	5.021	5.021
Bubble Sort	47.000	5.146	5.146

Tab. 2: Aumentos gerados pela técnica proposta

Aplicação	Tempo de Execução	Tamanho do Código
Multiplicação de Matrizes	10,7%	79,1%
Bubble Sort	20%	25%

OBJETIVO

Técnicas baseadas em software possuem um baixo custo, porém não são capazes de oferecer 100% de detecção de erros no fluxo de controle do programa. O presente trabalho, então, propõe uma nova técnica híbrida baseada em software e em um módulo de hardware não intrusivo para detectar *SEEs* (*Single Event Effects*), fundamentada em 2 técnicas apresentadas em [1] e [2].

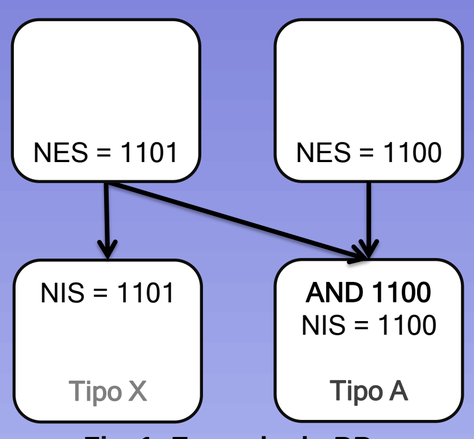

Fig. 1: Exemplo de *BBs*

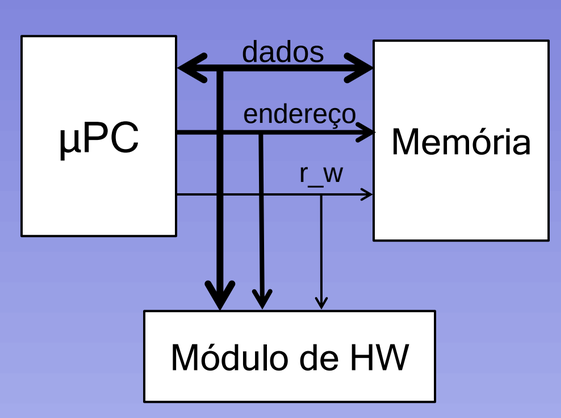

Fig. 2: Módulo de HW conectado

IMPLEMENTAÇÃO

Um módulo de hardware é conectado junto ao barramento entre o µPC e a memória, como na Fig. 2. O módulo possui um registrador, o qual possui o seu valor zerado no início de cada *BB* para, então, armazenar o *XOR* de todas instruções executadas pelo µPC. No fim, o módulo compara o *NS* do *BB* com o resultado dos *XORs* e, se forem diferentes, é porque algum erro ocorreu.

Na Fig. 3 há um exemplo de código protegido. O módulo zera o seu registrador sempre que um *XOR* no *S* é realizado, e verifica se houve algum erro através de um *STORE* do *S*, comparando o valor enviado à memória com o que ele calculou. A ***constante*** é o resultado do *XOR* entre *NIS* e o *NS*, pois, sabendo que *S* entrará no BB com o valor do *NIS*, depois dessa instrução ele ficará com o *NS*. O módulo também possui um contador interno, incrementado a cada clock e zerado no início de cada *BB*, para detectar loops.

VALIDAÇÃO

Para verificar a eficiência da técnica, foi realizada uma simulação de injeção de falhas em um microprocessador *MIPS* através do *ModelSim*. Duas aplicações foram testadas: uma multiplicação de matrizes 6x6 e um *bubblesort* de 10 elementos. Tanto a proteção dos códigos quanto a injeção de falhas foram realizadas automaticamente por 2 ferramentas desenvolvidas em Java.

CONCLUSÃO

Como desejado, a técnica proposta foi bem sucedida em garantir total proteção ao fluxo de controle. Além disso, ela apresentou um aumento consideravelmente pequeno no tempo de execução das aplicações. Outro ponto positivo foi que, ao adicionar o módulo de HW na arquitetura do *MIPS*, o aumento observado na área ocupada por ele foi de apenas 11%.

REFERÊNCIAS

[1] R. Vemu, J. Abraham, “CEDA: Control-flow error detection through assertions,” in Proc. of IEEE Int. On-Line Testing Symposium, 2006.

[2] J. R. Azambuja, A. Lapolli, L. Rosa, F. L. Kastensmidt, “Detecting SEEs in Microprocessors Through a Non-Intrusive Hybrid Technique,” in IEEE Transactions on Nuclear Science, Vol. PP, Issue: 99, 2011.