Tratamento de Ponteiros Incorretos armazenados em Memórias Aproximadas

João Fabrício Filho[1,2], **Isaías B. Felzmann**[2], **Lucas F. Wanner**[2]

[1]Universidade Tecnológica Federal do Paraná – Câmpus Campo Mourão
Via Rosalina M Santos, 1233 – 87301-899 – Campo Mourão – PR – Brasil

[2]Instituto de Computação – Universidade Estadual de Campinas
Av Albert Einstein, 1251 – 13083-852 – Campinas – SP – Brasil

joaof@utfpr.edu.br, {isaias.felzmann,lucas}@ic.unicamp.br

Resumo. *Memórias aproximadas podem causar erros em dados críticos da aplicação, o que ocasiona a interrupção prematura do fluxo de execução sem produzir saídas. O objetivo deste trabalho é propor formas de tratar ponteiros incorretos armazenados em memórias aproximadas para diminuir a quebra de aplicações, e assim melhorar os resultados para essas execuções. Nosso trabalho propõe duas formas de tratar dados de endereços fora dos limites da memória: mascarar os bits para adequá-los aos limites permitidos e descartar as operações com esses valores. Nossos resultados mostram redução das quebras de aplicação em até 15,88% com as técnicas propostas.*

1. Introdução

Tradicionalmente, para obter alto desempenho em recursos computacionais, há maior utilização de recursos e o consequente maior consumo de energia. Contudo, no contexto da computação aproximada, é possível aumentar desempenho e diminuir consumo de energia ao mesmo tempo, ao custo de uma margem de imprecisão nos resultados [Kugler 2015]. O estudo desse paradigma foi intensificado na era do fenômeno *dark silicon* [Esmaeilzadeh *et al.* 2012].

Diversas técnicas de *hardware* oferecem a oportunidade de relaxar a precisão em proveito da economia de energia e maior desempenho, alterando a arquitetura, o circuito ou parâmetros de funcionamento [Mittal 2013]. O ajuste dinâmico de tensão é uma técnica que permite o funcionamento de circuitos em faixas de maior eficiência energética [De *et al.* 2017].

Em elementos de memória, o ajuste de tensão pode causar ruídos nas operações de leitura e escrita, o que expõe os dados armazenados a certa probabilidade de erro. Dessa forma, existe uma relação entre a probabilidade de erro à qual os dados são expostos e a dissipação de energia ocasionada [Wang and Calhoun 2011]. Os dados afetados pelos erros muitas vezes representam informações críticas da aplicação, e podem causar quebras, com término prematuro do fluxo de execução [Felzmann *et al.* 2018b]. Ao terminar prematuramente uma execução, dados prévios podem ser perdidos antes da escrita das saídas, e a qualidade final dos resultados pode ficar comprometida. Assim, é importante encontrar formas de diminuir a quebra de execuções para gerar resultados de maior qualidade. Ao utilizar memórias aproximadas, grande parte das interrupções

prematuras de execução é causada por tentativas de acesso a endereços inválidos, que foram lidos ou armazenados incorretamente [Felzmann *et al.* 2018b].

Este trabalho propõe formas de tratamento a tentativas de acesso a endereços que estejam fora dos limites permitidos, dentro do contexto da utilização de memórias que exponham os dados armazenados a certa probabilidade de erro.

Na memória não há informação sobre os tipos de dados armazenados, assim não há como distinguir ponteiros armazenados de outros tipos de dados. Desse modo, nossa proposta é tratar os endereços inválidos durante as operações de leitura e escrita na memória e não emitir sinais de falha de segmentação. Analisamos duas formas de tratamento de endereços, uma trunca os dados em valores dentro dos limites da memória e outra descarta as operações que envolvem endereços inválidos.

Nossos resultados apresentam que truncar os dados de endereços reduz as quebras das aplicações em 7,92%, enquanto ignorar as operações reduz em 8,91% na probabilidade de erro 1E-4. Esperamos que nosso trabalho contribua com o aumento da qualidade dos resultados em aplicações que executem em ambientes que utilizam dados aproximados, de forma que haja ganho energético e de desempenho.

2. Proposta

Nosso modelo arquitetural é baseado em um processador para dispositivos embarcados, que ajusta a tensão da memória de dados por meio de um controle disponível para a aplicação, que parametriza a probabilidade de erros.

Esse controle é desativado ao início de cada execução, para que a leitura das entradas da aplicação seja feita de forma precisa. A aproximação é ativada somente na fase de processamento/convergência da aplicação e desativada novamente para a escrita das saídas. Isso é feito para que as fases de I/O do programa não sejam afetadas, já que essas fases são mais sensíveis a erros.

A quebra de uma execução ocorre quando seu fluxo é interrompido prematuramente. No contexto do modelo proposto, a escrita das saídas não é exposta a erros e está no final do fluxo de execução do programa. Dessa forma, com a quebra não se produz uma saída para a execução. Há três tipos de quebras que podem ocorrer durante a execução: (1) quebra de fluxo de dados, que ocorre quando se tenta buscar um dado em um endereço fora dos limites permitidos; (2) quebra de fluxo de controle, que ocorre quando há tentativa de salto do fluxo de execução para um endereço inválido; e (3) quebra por tempo, quando um resultado válido não é obtido em tempo hábil.

Tanto as quebras de fluxo de dados quanto as de fluxo de controle são causadas diretamente por operações com endereços incorretos, alterados pelos ruídos de escrita ou leitura na memória aproximada. Quebras por tempo também podem ser causadas indiretamente por endereços inválidos, uma vez que algum ponteiro de controle de laço de repetição pode ser perdido, causando *loop* infinito.

Nossa proposta é tratar as operações com os valores inválidos de ponteiros no momento em que o fluxo de execução tenta ler ou escrever nesses endereços. O sinal de falha de segmentação causa imediatamente a quebra da aplicação e deve ser evitado. Contudo, sem os valores corretos dos endereços, o fluxo de execução pode se perder ao ler ou escrever dados errados. Assim, nossa primeira proposta é evitar a falha

de segmentação, descartar as operações de leitura e escrita em endereços inválidos e continuar com o restante da computação.

Temos que considerar também que o endereço inválido foi alterado pelos ruídos da memória aproximada, que causam inversão em algum *bit* da palavra de dados. Para estar fora dos limites permitidos, um valor de endereço pode estar com a inversão de algum *bit* de maior significância do que o limite máximo da memória. Dessa forma, nossa segunda proposta é aplicar uma máscara para corrigir esses endereços, de forma a ignorar os *bits* mais significativos do que o valor limite da memória. Isso realiza uma operação de módulo do valor do endereço em relação ao tamanho máximo que ele possa ter.

3. Avaliação Experimental

3.1. Configuração

Utilizamos um simulador com aproximações baseadas na representação ADeLe [Felzmann *et al.* 2018a], na qual todas as operações de memória são substituídas por modelos de *software* que expõem a palavra de dados a uma inversão de *bit*, de acordo com uma probabilidade uniforme. Cada aplicação foi avaliada com 40 probabilidades em intervalos exponenciais entre 1E-4 e 1E-9. Em cada probabilidade as aplicações foram executadas 25 vezes. A espera para ocorrer quebra por tempo foi parametrizada como 2 vezes o tempo de execução acurado.

Além de aplicações de processamento de sinais, típicas do contexto de computação aproximada, selecionamos aplicações com uso intensivo de CPU (*CPU-bound*) e de memória (*memory-bound*), dois tipos que utilizam o armazenamento em memória de formas diferentes. As aplicações foram selecionadas dentre os benchmarks AxBench [Yazdanbakhsh *et al.* 2017], MiBench [Guthaus *et al.* 2001], Microbench[1] e CBench[2], como mostra a Tabela 1.

Tabela 1. Aplicações selecionadas para a avaliação experimental.

Aplicação	Tipo	Origem	Aplicação	Tipo	Origem
nbody	*CPU-bound*	Microbench	qsort	*Memory-bound*	MiBench
mandelbrot	*CPU-bound*	Microbench	fft	Processamento de sinal	MiBench
bzip2	*Memory-bound*	Cbench	jpeg	Processamento de sinal	AxBench
bunzip2	*Memory-bound*	Cbench			

3.2. Resultados

A Figura 1 apresenta o percentual de quebras para todas as execuções das aplicações selecionadas nas probabilidades de erro em que houve mais de 5% de quebras. A *baseline* apresentada se refere ao uso de memória aproximada sem qualquer proteção aos dados de endereços. *Descartar* se refere à proposta de ignorar as operações com endereços inválidos, enquanto *Truncar* se refere à máscara que tenta corrigir os valores de endereços dentro dos limites permitidos.

As quebras de fluxo de dados quase foram eliminadas nas duas formas de tratamento, com somente uma quebra em cada probabilidade a partir de 4,12E-5 nas duas técnicas. Contudo, tanto quebras de tempo quanto de controle aumentaram com as formas

[1]http://benchmarksgame-team.pages.debian.net/benchmarksgame/

[2]http://ctuning.org/cbench

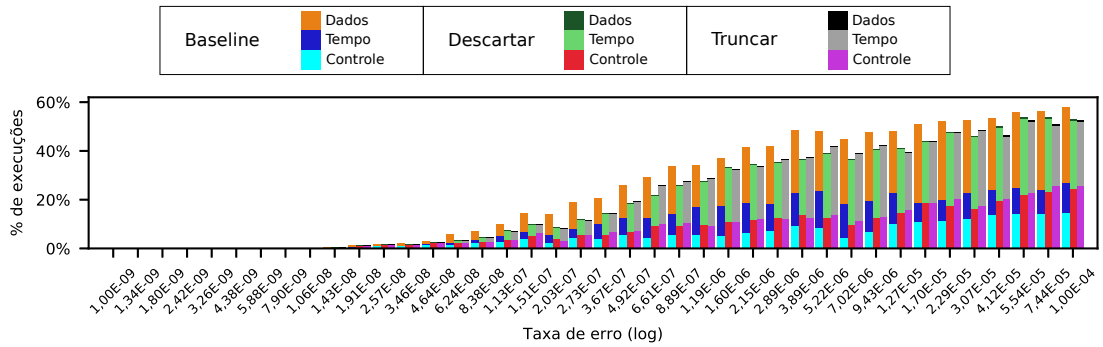

Figura 1. Percentual de cada tipo de quebra das execuções nos experimentos.

de tratamento propostas, o que mostra que os valores incorretos podem influenciar os outros dois tipos de quebra no restante da computação. Porém, prevalece a melhoria no total de quebras, que diminui 7,92% com *truncar* e 8,91% com *descartar* relativo ao *baseline* na maior probabilidade avaliada. No número global de execuções, *truncar* e *descartar* diminuem, respectivamente, 15,88% e 15,22% as quebras em relação ao *baseline*.

4. Considerações Finais

Este trabalho explorou o uso de memórias aproximadas e mostrou por meio da avaliação experimental que alterar a forma de tratar dados de endereços pode reduzir o número de quebras no fluxo de execução das aplicações. As duas estratégias propostas reduziram o número total de quebras no fluxo de execução, mostrando resultados promissores. Contudo, é necessário um estudo mais aprofundado para aferir o aumento da qualidade dos resultados, que é específica de cada aplicação. Além disso, sugerimos como trabalhos futuros a proposta de novas técnicas para proteger outros dados essenciais das aplicações no uso de memórias aproximadas, que diminuam também quebras de controle e de tempo.

Referências

De *et al.* (2017). Near Threshold Voltage (NTV) computing: Computing in the dark silicon era. *IEEE D&T*.

Esmaeilzadeh *et al.* (2012). Dark silicon and the end of multicore scaling. *IEEE Micro*.

Felzmann *et al.* (2018a). ADeLe: Rapid Architectural Simulation for Approximate Hardware. In *SBAC-PAD*.

Felzmann *et al.* (2018b). Impacto de memórias aproximadas na eficiência energética. In *WSCAD*.

Guthaus *et al.* (2001). MiBench: A free, commercially representative embedded benchmark suite. In *IEEE WWC*.

Kugler (2015). Is "good enough" computing good enough? *Commun. of the ACM*.

Mittal (2013). A Survey of Techniques for Approximate Computing. *JCIM*.

Wang and Calhoun (2011). Minimum supply voltage and yield estimation for large SRAMs under parametric variations. *IEEE Transactions on VLSI Systems*.

Yazdanbakhsh *et al.* (2017). AxBench: A Multiplatform Benchmark Suite for Approximate Computing. *IEEE D&T*.