FaultRecovery: A ampliação da biblioteca de tolerância a falhas

Cleiton Gonçalves de Almeida

Orientador: Prof. Me. Kleber Kruger
Campus Coxim
Universidade Federal de Mato Grosso do Sul

cleitonufms@hotmail.com

13 de Setembro de 2016

Estrutura da Apresentação

Introdução

- Os sistemas embarcados (*embedded computers* ou *embedded systems*) correspondem a maior classe de computadores e abrangem uma grande faixa de aplicações e desempenhos;
- Com a expansão da computação ubíqua, os sistemas embarcados estão cada vez mais presentes no cotidiano das pessoas.
- É importante que esses sistemas tolerem falhas, pois defeitos nessas aplicações podem causar transtornos e prejuízos.

Justificativa

- Falhas podem ser causadas por:
  - *Bugs* de software;
  - Questões ambientais;
    - Interferências eletromagnéticas;
    - Pulsos transitórios causados por prótons, íons pesadores e elétrons; Que podem causar um *bit-flip*
  - Problemas intrínsecos;
    - Desgaste (envelhecimento) dos componentes de *hardware*;
    - Problemas de fabricação;

Objetivo

- Objetivo geral: Ampliar o injetor de falhas (FaultInjector) e a biblioteca de recuperação de falhas (FaultRecovery) desenvolvidos por Kruger em sua dissertação de mestrado.

Tolerância a falhas

- Tolerância a falhas é a propriedade que permite a um sistema continuar funcionando adequadamente, mesmo que num nível reduzido, após a manifestação de falhas em alguns de seus componentes.

Injeção de Falhas

- A injeção de falhas é um processo importante para validar e verificar a confiabilidade de um sistema, seja por alteração de código, simulando uma falha de software ou a nível de pinos (Pin-level Injection) injetando falhas diretamente no hardware.
- Por que usar injetores de falhas ao invés de testar em um ambiente real?

Padrão de Projeto State

- Padrões de comportamento são aqueles que descrevem como os objetos interagem, distribuindo responsabilidades.
- O padrão State faz parte dos padrões de comportamento e permite que parte do comportamento de um objeto seja alterado conforme o estado do objeto.

Metodologia

- Utilizou-se como ponto de partida as bibliotecas de injeção de falhas e de recuperação de falhas.
- Para a ampliação das bibliotecas utilizou-se um microcontrolador *mbed*, modelo NXP LPC1768. Este módulo é composto por:
  - Núcleo ARM Cortex-M3 de 32 bits e 96 MHz de *clock*;
  - Memória *flash* com capacidade de 512 kB;
  - Memória SRAM de 64 kB (32 kB SRAM destinados ao programa de usuário, e dois blocos adicionais SRAM de 16 kB para uso interno do microcontrolador);

Biblioteca FaultInjector

- Na versão de Kruger a biblioteca FaultInjector permite simular o efeito do bit-flip em qualquer região de memória SRAM do microcontrolador mbed LPC1768, no entanto possuía duas limitações:
  - Falta de flexibilidade, por não executar em outros modelos de microcontroladores.
  - Impossibilidade de inserção de falhas na memória flash.

Biblioteca FaultInjector

- Primeira limitação:
  - Foi possível criar uma única versão que automaticamente mapeia as regiões de memória dos modelos LPC1768, 66, 65 e 64.

Biblioteca FaultInjector

- Segunda limitação:
  - O interesse era explorar a memória flash ou seja, descobrir se seria possível injetar falhas nessa região de memória, uma vez que não era possível.

Biblioteca FaultRecovery

- A biblioteca FaultRecovery foi criada para facilitar o uso de estratégias de tolerância e recuperação de falhas. Porém ela possuía algumas limitações:
  - A estrutura da versão anterior não era baseada em um padrão de projeto.
  - Utilizava funções por meio de defines (má prática).
  - Difícil modularização do código.

Biblioteca FaultRecovery

- Foi modificada para seguir o padrão *State* forçando o usuário a implementar o *firmware* como uma máquina de estados.
- Cada estado contém a sua classe de implementação, possibilitando a separação por responsabilidades.
- Possibilita a criação de pontos de recuperação de falhas.

Fluxograma da Biblioteca FaultRecovery

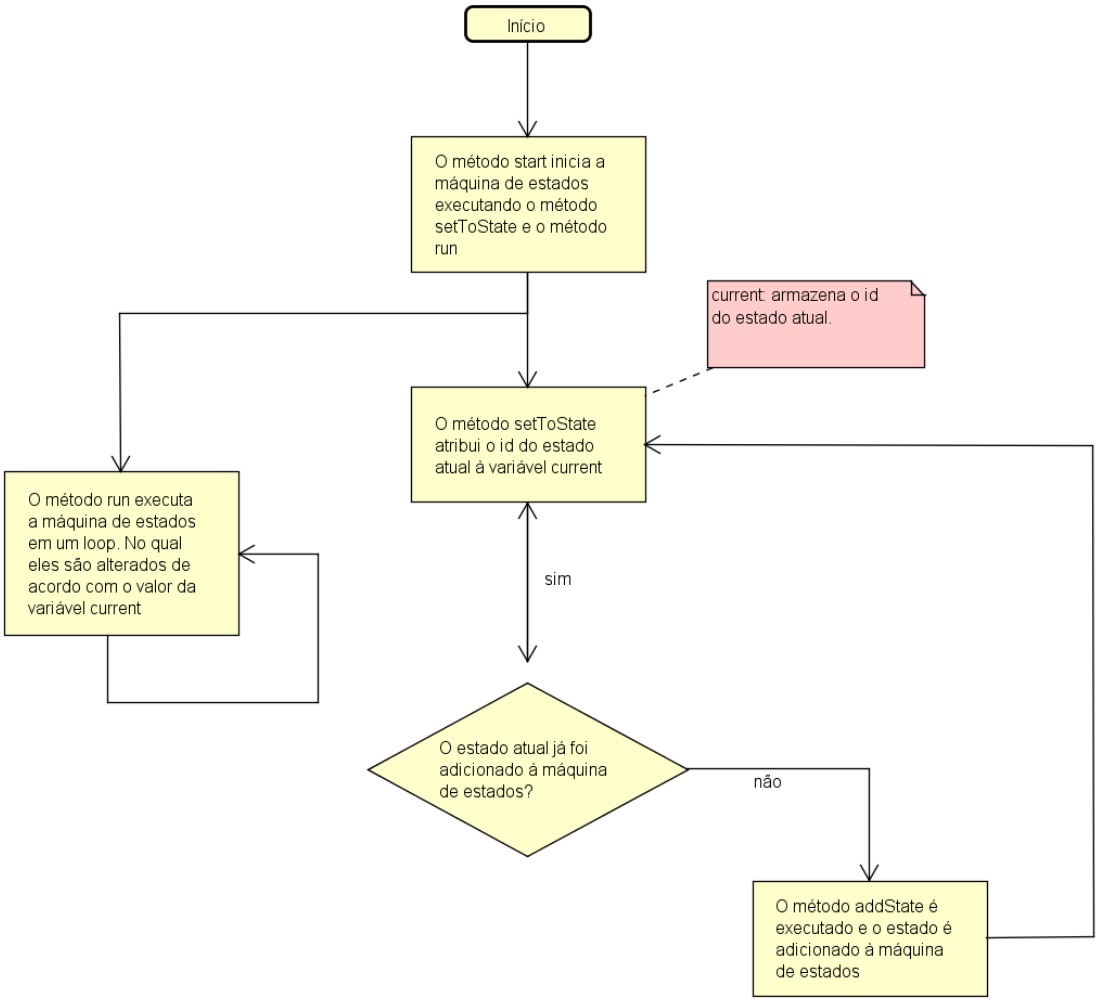
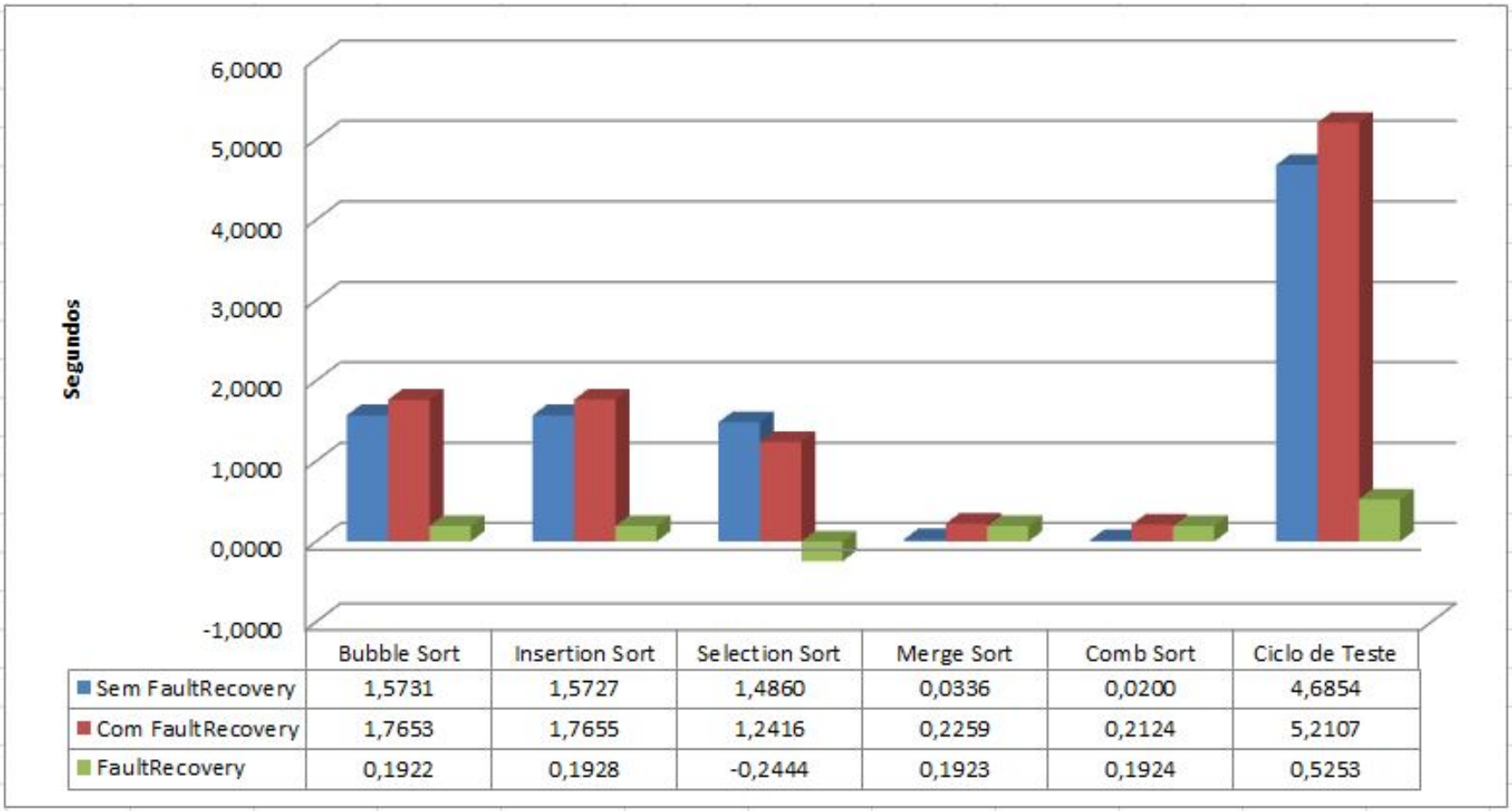


Figura: Fluxograma do funcionamento da máquina de estados da biblioteca.

Classe TData

- Redundância de dados automatizada tanto para variáveis primitivas quanto para objetos.
- O valor do objeto é armazenado em 3 cópias de segurança.
- As cópias são protegidas por um esquema de votação.
- Funciona com ponteiros e objetos de pilha.
- Mínima reescrita de código.
- Exemplo:
  - TData<tipo_da_variável> variavel(valor ou objeto).

Resultado do Desempenho da Biblioteca FaultRecovery



	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Comb Sort	Ciclo de Teste
Sem FaultRecovery	1,5731	1,5727	1,4860	0,0336	0,0200	4,6854
Com FaultRecovery	1,7653	1,7655	1,2416	0,2259	0,2124	5,2107
FaultRecovery	0,1922	0,1928	-0,2444	0,1923	0,1924	0,5253

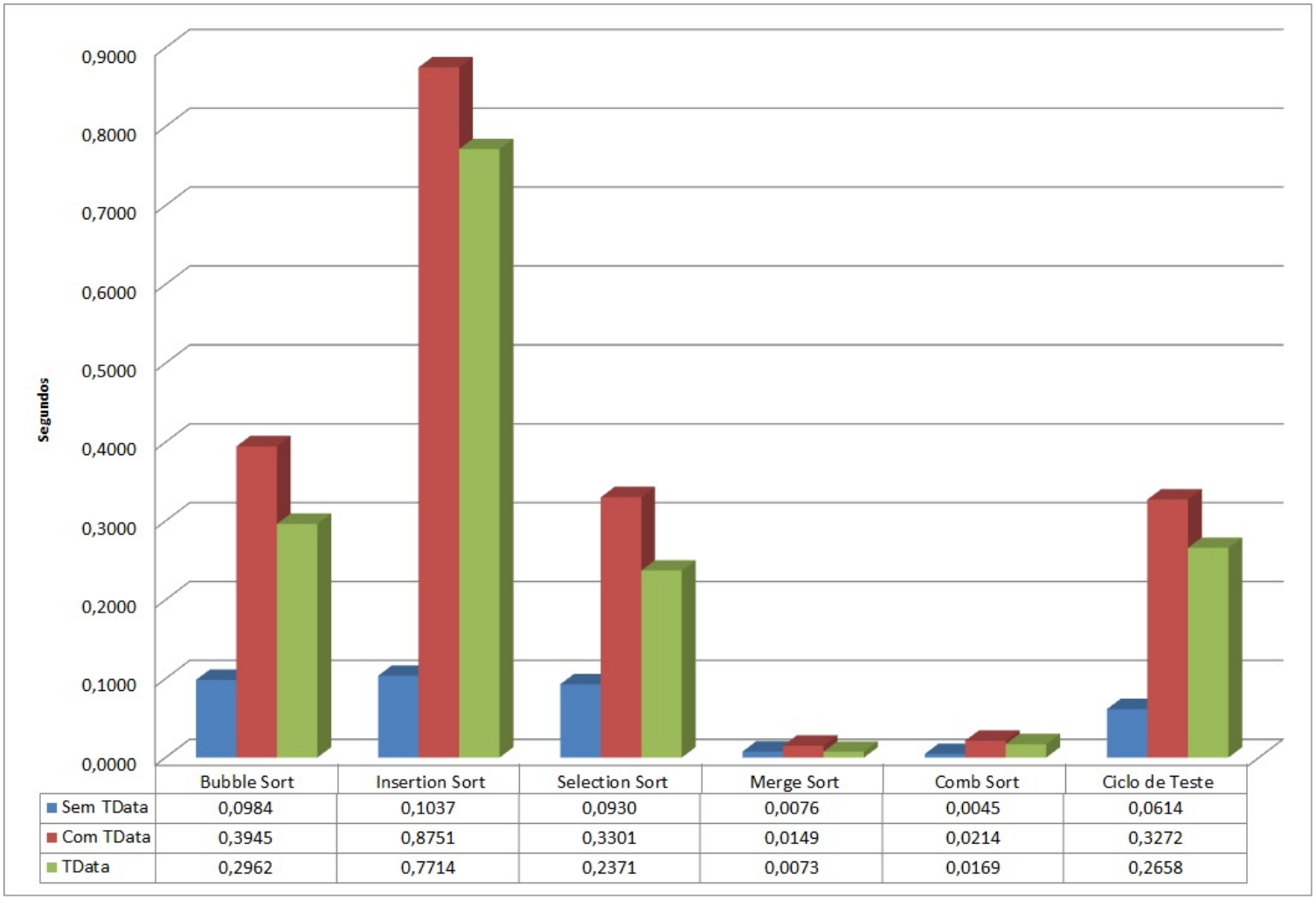
Figura: Tempo de execução dos algoritmos de ordenação sem a biblioteca e com ela.

Resultado do Desempenho da Biblioteca FaultRecovery

	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Comb Sort
Com FaultRecovery(vetor 8KB)	0,1922	0,1928	-0,2444	0,1923	0,1924
Com FaultRecovery(vetor 6KB)	0,1966	0,1964	-0,0490	0,1970	0,1970
Com FaultRecovery(vetor 4KB)	0,2029	0,2023	0,0929	0,2020	0,2027

Figura: Tempo de execução da biblioteca FaultRecovery com três vetores de tamanhos diferentes.

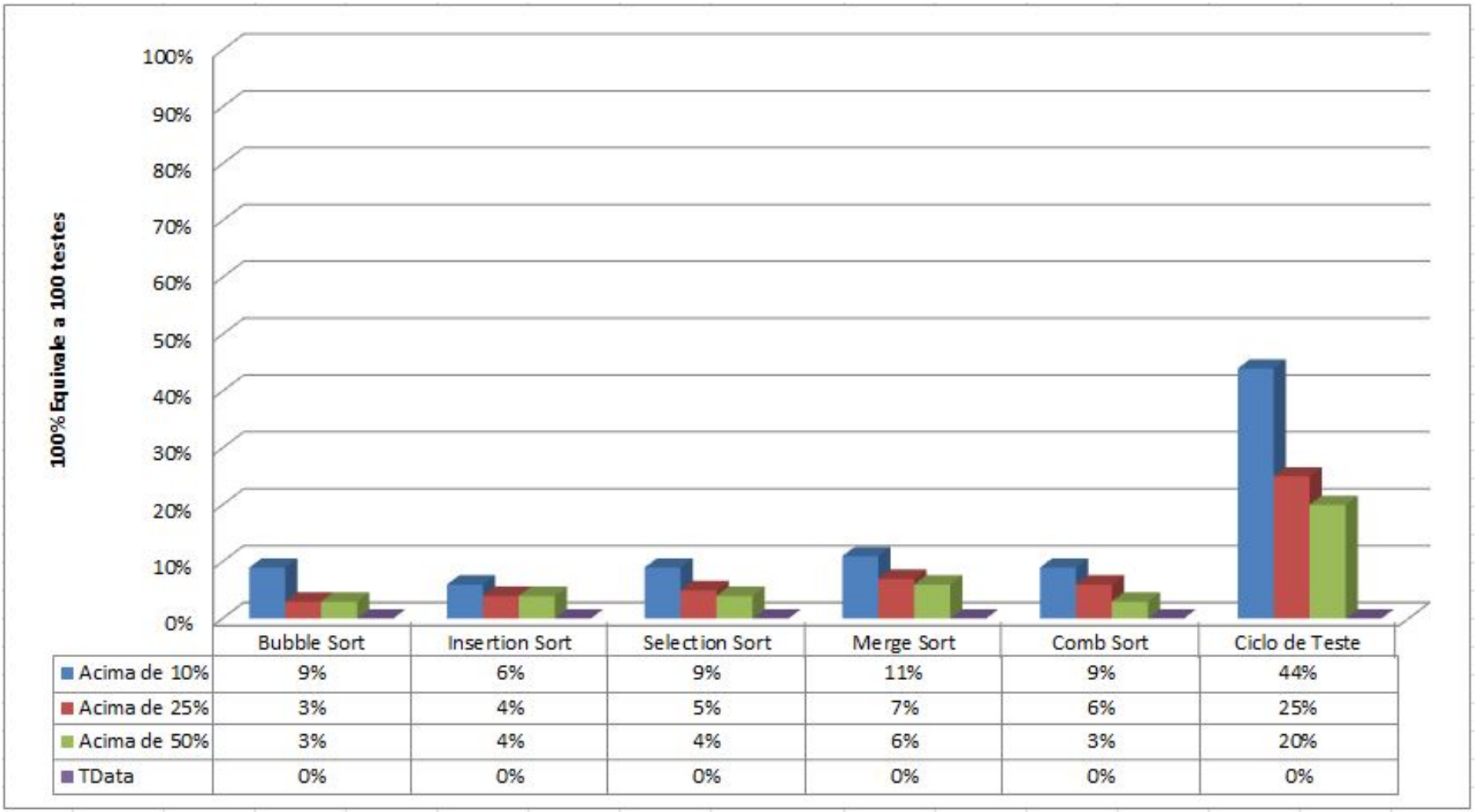
Eficiência da Classe TData



	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Comb Sort	Ciclo de Teste
Sem TData	0,0984	0,1037	0,0930	0,0076	0,0045	0,0614
Com TData	0,3945	0,8751	0,3301	0,0149	0,0214	0,3272
TData	0,2962	0,7714	0,2371	0,0073	0,0169	0,2658

Figura: O tempo de execução dos algoritmos sem a classe TData e com ela.

Eficiência da Classe TData



	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Comb Sort	Ciclo de Teste
Acima de 10%	9%	6%	9%	11%	9%	44%
Acima de 25%	3%	4%	5%	7%	6%	25%
Acima de 50%	3%	4%	4%	6%	3%	20%
TData	0%	0%	0%	0%	0%	0%

Figura: Falhas detectadas nos teste.

Recuperação de Falhas da Biblioteca FaultRecovery

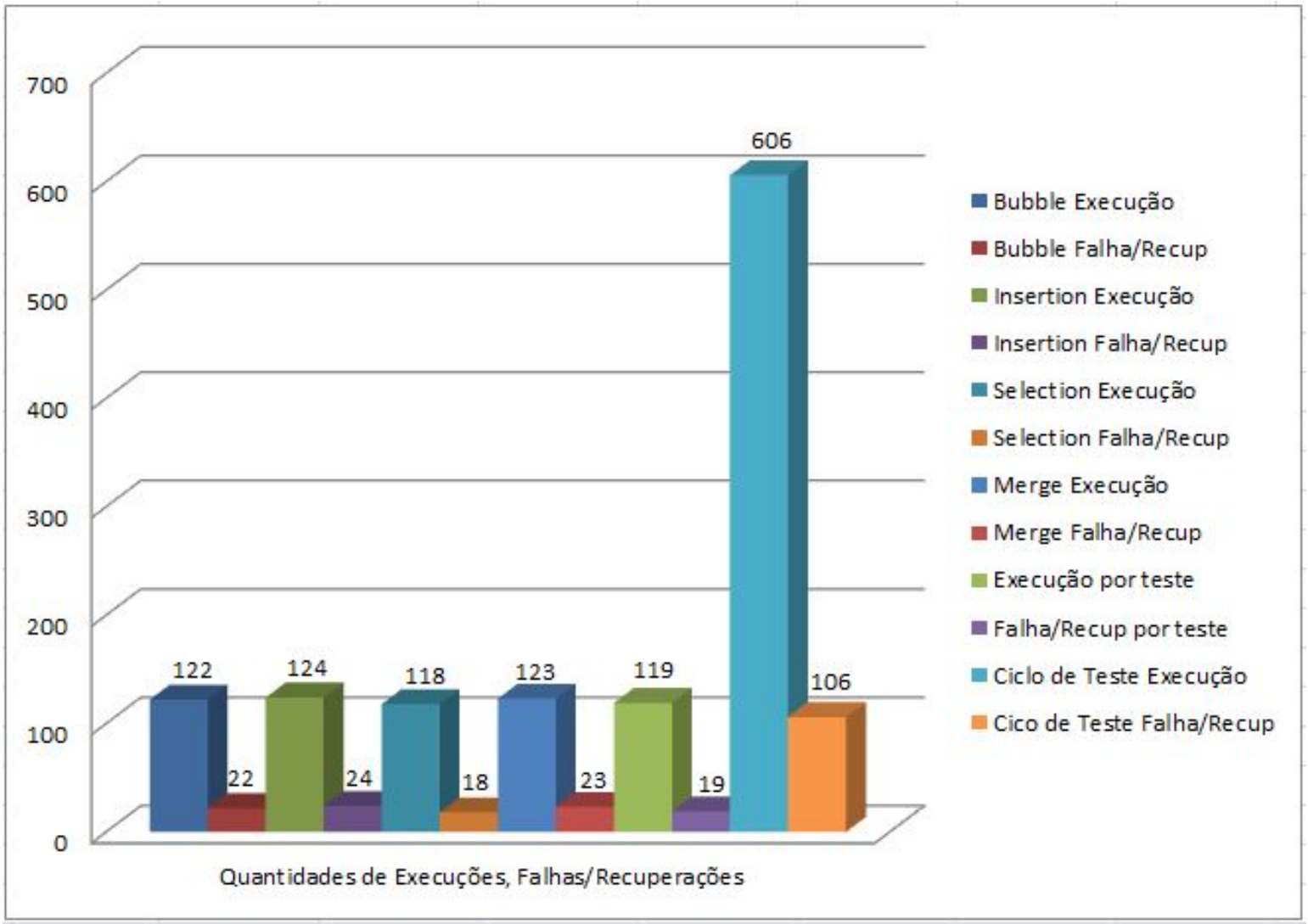


Figura: Quantidade de execuções e falhas de cada algoritmo.

Injeção de Falhas na Memória Flash

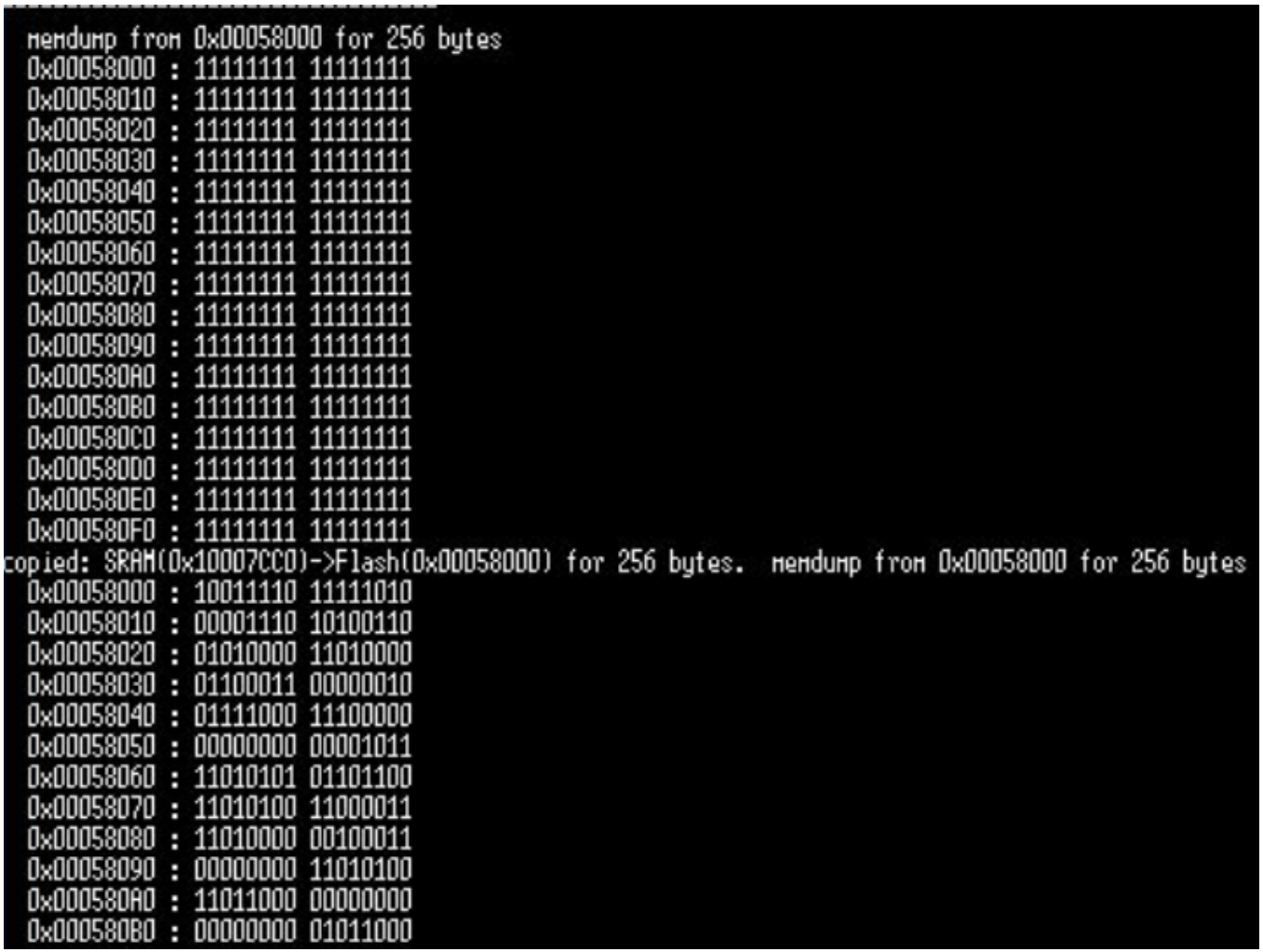


Figura: Antes e depois dos bits serem modificados.

Conclusão

- Sem a FaultRecovery:
  - O *firmware* falhou e não continuou a sequência da máquina de estados.
- Com a FaultRecovery:
  - O *firmware* falhou e continuou a sequência da máquina de estados de acordo com o ponto de recuperação configurado.
  - Aumento no tempo de execução do *firmware*. No entanto a maioria das aplicações embarcadas não tem como fator principal o tempo de execução, exceto algumas aplicações de tempo real.

Conclusão

- Sem redundância de dados:
  - Nenhuma falha foi tolerada.
  - Num total de 100 testes, 44% falharam.
- Com redundância de dados:
  - 100% das falhas foram toleradas. No entanto adiciona um custo de desempenho no tempo de processamento.
  - Num total de 100 testes, nenhum teste falhou.

Contribuições deste trabalho

- Parte da biblioteca FaultRecovery está sendo utilizada pelo Projeto de extensão Coxim robótica sediado na UFMS - campus Coxim para implementar uma máquina de estados para um carrinho seguidor de linha.

Trabalhos Futuros

- Testar o mapeamento de memória incluindo na biblioteca FaultInjector para outros modelos além do modelo *mbed* LPC1768.
- Salvar as cópias utilizadas pela classe TData, para se ter redundância de dados, em outras regiões de memória.
- Aperfeiçoar a biblioteca FaultRecovery e a classe TData para diminuir o tempo de processamento em uma aplicação embarcada.
- Explorar a injeção de falhas na memória flash, implementando um *firmware* e injetando falhas enquanto ele está em execução.

Trabalhos Futuros

Obrigado!