RTEMS/SPARC 환경에서 소프트웨어 기반의 태스크 수준 결함 감내 기법

김범식, 양회석
아주대학교 전자공학과
asd326541@ajou.ac.kr, hyang@ajou.ac.kr

A Software-Based Task-Level Fault-Tolerance Technique on RTEMS/SPARC

Beomsik Kim, Hoeseok Yang
Department of Electrical and Computer Engineering, Ajou University

요 약

본 논문에서는 RTEMS/SPARC 환경에서 소프트웨어 기반의 결함 감내 기법을 통해 태스크 수준에서 메모리에 발생하는 결함을 감내하는 기법을 구현하고 이를 보고한다. 일반적으로 고칠 수 없는 결함은 소프트웨어적으로 처리하기 어려우므로 watchdog을 이용한 시스템 재부팅으로 감내한다. 이 논문에서는 SPARC 기반 프로세서에 이식된 RTEMS(Real-Time Executive for Multiprocessor Systems) 운영체제의 RM(Rate-Monotonic) 스케줄러를 수정하고, 결함 주입, 결함 검출, 결함 정정 기법을 구현하여 태스크 수준에서 SRAM 메모리의 결함을 감내할 수 있음을 확인하였다. 또한, 비용분석을 위하여 구현 기법에 따른 평균 연산 시간을 측정하고, 이를 해석하였다.

Ⅰ. 서 론

임베디드 시스템에 사용되는 마이크로프로세서에 트랜지스터의 집적도가 높아지고, 동작 전압이 낮아지면서 소프트에러(soft-error)의 발생 가능성이 커지고 있다. 소프트에러란 반도체에 저장된 비트값이 일시적으로 0에서 1로 또는 1에서 0으로 바뀌는 현상을 말하며, 우주 방사선(cosmic rays), 태양열과 같은 외부 환경이나 내부적인 요인(alpha particle) 등에 의해 발생한다.

높은 신뢰성이 요구되는 항공, 위성, 자동차, 의료기기 등의 시스템에서 소프트에러를 감내하지 못한다면 심각한 결과를 초래할 수 있다. 소프트에러를 감내하기 위해 하드웨어를 이중화, 삼중화하고, 소프트웨어적으로 중복 연산을 수행하는 등의 다양한 보호 기법이 적용되고 있으며, 이에는 추가적인 비용이 요구된다.

태스크 수준에서 소프트에러를 감내하기 위한 대부분의 연구는 이론적인 모델링 위주로 수행되고 있으며 실제 하드웨어와 실시간운영체제 환경에서 직접 구현하고, 검증하는 실용적인 사례는 거의 없다. 또한, 고칠 수 없는 결함은 소프트웨어적으로 유연하게 처리하기보다 watchdog을 이용한 시스템 재부팅으로 감내하는 것이 일반적이다 [3].

[1]에서는 소프트에러 발생확률이 낮고, 주기적인 제어 태스크가 소프트에러를 어느 정도 감내할 수 있으므로 모든 태스크에 보호 기법을 적용하는 것은 비효율적이라고 보고하고 있으며, (m, k)-제약을 이용하여 제어 성능의 큰 저하 없이 시스템 이용률을 줄일 수 있는 기법을 제안했다. [2]에서는 ESA(European Space Agency)와 [1]의 기법을 RTEMS(Real-Time Executive for Multiprocessor Systems) 실시간 운영체제의 RM(Rate-Monotonic) 스케줄러를 수정하여 FTS(Fault Tolerant Scheduler)를 구현하고자 했지만 가장 중요하고 실질적인 결함 주입, 결함 검출, 결함 정정 기법을 구현으로 검증하지 않았다.

본 논문에서는 [1, 2]를 기반으로 RTEMS/SPARC 환경에서 RM 스케줄러를 수정하고, 태스크 수준에서 결함 주입, 검출, 감내하는 기법을 구현하여 SRAM 메모리의 결함을 감내할 수 있음을 확인하고, 결함 검출, 감내 기법에 따른 평균 연산 시간을 측정하고, 분석했다.

Ⅱ. 본 론

GR712RC의 FTMCTRL 메모리 컨트롤러는 SRAM에 EDAC(Error Detection and Correction)의 적용 여부를 설정할 수 있다. 32-비트 word와 7-비트 체크섬을 이용하고, 1-비트 결함의 정정, 2-비트 이상 결함의 감지가 가능하며, MCFG3 레지스터의 TCB에 체크섬이 저장된다 [4]. 결함 종류에 따라서 AHB status register의 NE(New Error), CE(Correctable Error) 비트가 설정되며, 정정 불가능한 결함일 경우 AMBA ERROR 인터럽트가 발생하고, data_access_exception trap(tt=0x09)이 생성되며 프로세서가 정지된다 (해당 trap의 핸들러는 구현되어 있지 않다). 소프트웨어적으로 결함을 처리하기 위해선 시스템이 종료하지 않아야 하므로 RTEMS의 set_vector() API(Application Programming Interface)를 이용하여 trap 핸들러를 추가하고, 인라인 어셈블리 코드로 trap 핸들러에서 복귀하도록 하였다 [3].

Ⅱ.a FTS [1]

태스크는 결함 검출, 정정 기법의 적용 여부에 따라서 세 가지 버전으로 구분된다. 태스크 τ_B(Basic)는 연산만 수행하고, 태스크 τ_D(Detection)는 연산 후 결함을 검출하고, 태스크 τ_C(Correction)는 연산 후 결함을 정정하며, 실행시간은 $\tau_B < \tau_D < \tau_C$ 순이다.

(m, k)-제약은 주기적인 태스크의 연속적인 k번 실행 중에서 m번을 결함 없이 실행해야 하고, (m, k)-패턴은 0과 1로 이루어지며 어떤 버전의 태스크를 실행할지의 계획을 나타낸다. 예를 들면, (3, 5)-제약에서 패턴이 [0, 1, 0, 1, 1]일 때 정적기법은 0에서 τ_B, 1에서 τ_C를 실행하며, τ_C를 3번 실행했으므로 (3, 5)-제약을 만족한다. 동적기법은 0에서 τ_D, 1에서 τ_C를 실행하며, τ_D에서 결함이 검출되지 않으면 τ_C를 실행한 것과 같게 취급하여 m번 중 1번을 만족한 것이고, 패턴에 S를 넣어 뒤로 미룬다. (한 번 미뤄진 패턴: [S, 0, 1, 0, 1], 두 번 미뤄진 패턴: [S, S, 0, 1, 0]) 즉 동적기법은 (m, k)-제약을 위배하지 않으면서 패턴에 따른 실행을 최대한 늦춰 오버헤드가 큰 τ_C의 실행을 최소화하여 시스템 이용률과 전력 소모량을

줄이게 된다.

[2]의 소스코드가 LEON3FT 기반의 GR712RC 개발 보드, RTEMS-5.0, RCC(RTEMS Cross Compiler)-1.3-rc6 환경에서 구동될 수 있도록 RTEMS 커널 일부를 수정하고, 알고리즘의 오류를 수정한 후 결함 주입, 검출, 감내 기법을 추가로 구현하였다.

Ⅱ.b 결함 주입

결함을 주입할 32-비트 변수의 체크섬에서 0-비트, 1-비트, 2-비트의 값을 수정하여 MCFG3 TCB에 기록함으로써 감내할 수 있는 결함과 감내할 수 없는 결함을 선택적으로 SRAM에 주입할 수 있다 [3]. 감내할 수 없는 결함이 주입된 변수에 접근할 때 인터럽트가 발생하는데, 캐시 메모리의 영향 없이 SRAM 메모리에 직접 접근하도록 인라인 어셈블리 코드를 작성하였다.

Ⅱ.c 결함 검출

결함 검출은 하드웨어 기능의 여부로 나눌 수 있다. 하드웨어 기능이 있는 경우, GR712RC의 AHB status register를 이용하여 결함의 종류를 검출할 수 있다. 결함이 발생하면 NE 비트가 1로 설정되는데, 이때 CE 비트가 1이면 감내 가능한 결함이고, 0인 경우에는 감내 불가능한 결함이다 [4]. 하드웨어의 기능이 없는 경우 태스크 이중실행(double execution) 이후 연산 결과를 비교하여 결함 발생 여부를 확인할 수 있다.

Ⅱ.d 결함 정정

결함 정정은 재실행(re-execution)과 삼중실행(triple execution) 두 가지 방법을 사용하여 구현하였다. 재실행 기법은 결함 검출 기법을 이용하여 결함이 검출되면 연산을 재실행한 후 여러 결과 중 다수결(voter) 결과를 활용하여 결함을 정정한다. 삼중실행의 경우, 처음부터 τ_B를 세 번 수행하고, 그 결과를 다수결하여 결함이 없는 연산 결과를 얻는 방법이다.

Ⅲ. 실험 결과

다음은 그림 1에 도시되고, 표 1에 요약된 여러 태스크 버전에 대한 설명이다. τ_B: Stanford baby benchmark suite만을 연산한다. τ_{D1}: 1개의 태스크에서 벤치마크 연산 이후 AHB status register를 읽어서 결함 발생 여부를 확인한다. $\tau_{D2,1}$: 1개의 태스크에서 벤치마크 연산을 2번하고, 연산 결과를 비교하여 결함 발생 여부를 확인한다. $\tau_{D2,2}$: 2개의 태스크에서 벤치마크 연산을 1번씩하고, 연산 결과를 비교하여 결함 발생 여부를 확인한다. τ_{C1}: τ_{D1}을 실행하고, 결함이 검출되었다면 τ_{D1}을 재실행하고, 결함이 검출되지 않으면 종료한다. $\tau_{C2,1}$: $\tau_{D2,1}$을 실행하고, 결함이 검출되었다면 벤치마크 연산을 1번 더 수행하고, 다수결을 이용하여 결함이 없는 연산 결과를 얻는다. $\tau_{C2,2}$: $\tau_{D2,2}$를 실행하고, 결함이 검출되었다면 벤치마크 연산을 1번 더 수행하고, 다수결을 이용하여 결함이 없는 연산 결과를 얻는다. $\tau_{C3,1}$: 1개의 태스크에서 벤치마크 연산을 3번하고, 다수결을 이용하여 결함이 없는 연산 결과를 얻는다. $\tau_{C3,2}$: 3개의 태스크에서 벤치마크 연산을 1번씩하고, 다수결을 이용하여 결함이 없는 연산 결과를 얻는다. 이러한 여러 버전에 대하여 태스크를 수행하고 평균 연산 시간을 구하였다.

결함을 감지하기 위한 하드웨어의 도움이 있는 경우(τ_{D1}, τ_{C1}), 하드웨어의 도움이 없는 경우에 비해 평균 연산 시간이 매우 적게 필요한 것을 확인할 수 있다. 하드웨어의 도움이 없는 경우, 벤치마크를 중복으로 연산하여 결함을 검출, 정정하는데, 태스크 한 개에서 연산할 때 태스크를 여러 개 생성하여 연산하는 경우보다 태스크 생성의 오버헤드가 적기 때문에 평균 연산 시간이 적게 필요한 것을 확인할 수 있다.

태스크 버전		결함	평균 연산 시간 (s)
Basic	τ_B	-	1.3975
Detection	τ_{D1}	x	1.4835
		o	1.5141
	$\tau_{D2,1}$	x	2.8377
		o	2.8294
	$\tau_{D2,2}$	x	2.9466
		o	2.9530
Correction	τ_{C1}	x	1.4866
		o	2.9280
	$\tau_{C2,1}$	x	2.8288
		o	4.3064
	$\tau_{C2,2}$	x	2.9413
		o	4.5311
	$\tau_{C3,1}$	x	4.2607
		o	4.2692
	$\tau_{C3,2}$	x	4.4810
		o	4.4947

표 1. 기법에 따른 평균 연산 시간

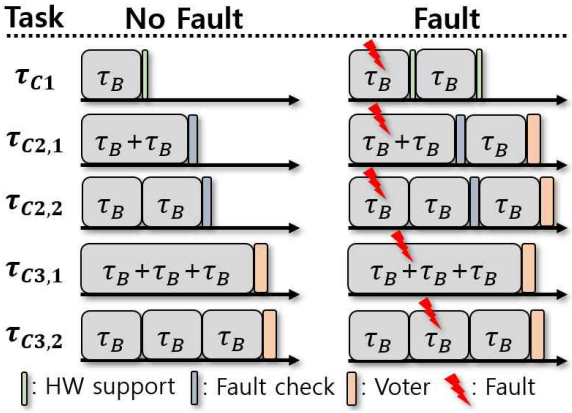

그림 1. 결함 여부에 따른 결함 정정 태스크

Ⅳ. 결 론

본 논문에서는 RTEMS/SPARC 환경에서 소프트웨어 기반의 결함 감내 기법을 통해 태스크 수준에서 메모리에 발생하는 결함을 감내하는 기법을 구현하고 이를 보고하였다. 향후 구현한 기법을 이용하여 인공위성의 운용환경에서 소프트에러와 프로세서의 수명 신뢰성을 동시에 고려하는 하이브리드 기법을 고안할 계획이다.

ACKNOWLEDGMENT

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터지원사업의 연구결과로 수행되었음 (IITP-2020-2018-0-01424)

참 고 문 헌

[1] K.-H. Chen et al. Compensate or Ignore? Meeting Control Robustness Requirements through Adaptive Soft-Error Handling, In LCTES, 2016.

[2] https://devel.rtems.org/wiki/SOCIS/2017

[3] Handling of External Memory EDAC Errors in LEON/GRLIB Systems (https://www.gaisler.com/doc/antn/GRLIB-AN-0004.pdf)

[4] GR712RC User's Manual