C-Lock: 멀티코어 임베디드 시스템을 위한 에너지와 성능 효율적 공유 데이터 동기화 기법

*이상형, 전민제, 이병훈, 정의영
연세대학교 전기전자공학부
e-mail : {ideallsh, minje.jun, bh2}@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

C-Lock: Energy- and Performance-Efficient Data Synchronization for Multicore Embedded System

*Sang Hyong Lee, Minje Jun, Byunghoon Lee, and Eui-Young Chung
School of Electrical and Electronic Engineering
Yonsei University

Abstract

We propose an energy- and performance-efficient data synchronization method for multicore embedded systems, called *C-Lock*. C-Lock can save system energy by gating the clocks of cores when the data they attempt to access are already preempted by another cores. In the experiment, we evaluate our C-Lock against traditional lock and transactional memory, and the result shows that C-Lock provides the best energy-delay product.

I. 서론

최근 멀티코어(multicore)는 범용 시스템뿐 아니라 임베디드 시스템(embedded system)에도 널리 적용되기 시작하고 있다. 하지만 Amdahl의 법칙에 따르면 프로그램의 병렬화 정도에 의해 성능 향상의 제약이 발생한다. 이에 멀티코어 환경에서 효율적인 병렬화를 위한 공유 데이터간의 동기화 기법으로서 기존의 Lock 기반 기술의 제약을 극복할 수 있는 Transactional Memory(TM)에 관련된 연구들이 최근 활성화되고 있는 추세이다[1,2]. 하지만 투기적 수행(speculative execution)을 통해 성능 향상을 꾀하는 TM의 기본 원리상 불필요한 에너지 소모가 필수적이다. 이는 최고의 성능을 요구하는 범용 시스템과는 달리 효율적인 전력 관리가 필수적인 임베디드 시스템의 경우 TM 기법보다 효율적인 에너지 관리가 가능한 공유 데이터 동기화 기법이 필요하다.

본 논문에서는 반복 실행 가능성이 없어 불필요한 에너지 소모가 없는 lock의 장점과, 단순히 식별자가 아닌 실제 접근하는 데이터에 대한 충돌을 감지할 수 있는 TM의 장점, 즉 두 기법의 장점들을 살릴 수 있는 임베디드 시스템을 위한 멀티코어상 공유 데이터 동기화 기법인 C-Lock을 제안하고자 한다.

II. 관련연구

2.1 Lock

대표적인 데이터 동기화 기술로써 lock, spin lock, semaphore, mutex 등이 현재까지도 널리 사용되고 있다. 에너지 소모를 줄이기 위해 lock시 코어가 sleep상태로 진입하는 기법 등의 연구가 있다.

Lock의 경우 id로 코드영역에 대한 권한을 획득/반환하는 방식을 취한다. 이 경우 다른 데이터에 대한 처리일지라도 다른 코어가 동시에 동일 코드에 대한 접근이 차단되기 때문에 멀티코어 환경에서 병렬성이 저하되는 문제점이 있다.

2.2 Transactional Memory

Lock기법의 병렬성 제약 문제를 극복하기 위한 lock-free 기법의 대표적인 기술로, 투기적 수행을 통해 성능 향상을 꾀하며 원자성(atomicity)을 보장함으로써 데이터의 일관성을 보장한다. 초기의 소프트웨어(software, SW)적 구현 기법의 저성능 문제로, 최근에는 cache coherency protocol 등을 활용한 하드웨어(hardware, HW) 구현 기법들이 연구되고 있으며, 임

베디드 시스템용으로는 대표적으로 EmbeddedTM[3] 등이 있다.

TM의 경우 기본적으로 투기적 수행을 기본 전제로 하고 있기 때문에, 공유 데이터에 동시 접근시, 충돌을 감지하여 abort 발생시 자동으로 TM적용 수행영역의 시작부분으로 roll-back이 되어 재실행이 일어나는 구조로써 부가적인 에너지 소모가 불가피하다. 즉, 성능을 높이기 위해 에너지 소모를 희생하고 있다.

Ⅲ. 본론

3.1 C-Lock - Overview

본 논문에서 제안하는 C-Lock은 멀티코어 환경에서 공유 데이터 접근 권한 요청시 권한 획득 여부에 따라 해당 코어로 연결된 clock을 인가/차단해 줌으로써 에너지 소모를 최적화하기 위한 공유 데이터간 동기화 기법이다. 이 C-Lock 기법을 구현하기 위해서 시스템 구성 방법의 변화와 추가적인 HW 로직, 그리고 이를 제어하기 위한 SW 제어 방법 변경이 필요하다.

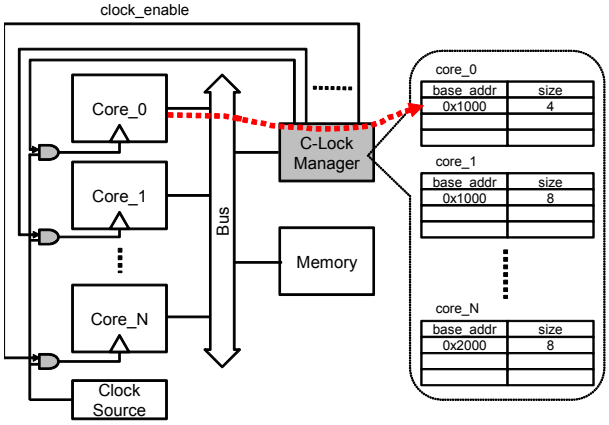

그림 1 C-Lock 시스템 구성도

3.2 C-Lock - System Architecture

그림 1은 C-Lock이 적용된 시스템 구성도이다. 멀티코어 환경에서 각 코어에 연결되는 clock 부분에 Enable 신호를 추가하고, 각 코어의 공유 데이터 접근 권한 요청시 해당 공유 데이터의 선점 여부에 의해 Enable 신호가 생성되어 각 코어에 연결되는 clock 신호를 제어할 수 있도록 시스템을 구성하였다. 이를 통해 그림 2와 같이 필요시에만 코어로 연결된 clock을 인가할 수 있게 함으로써 불필요한 에너지 소모를 방지할 수 있게 한다.

3.3 C-Lock - Software Control

표 1은 두 공유 데이터 X, Y에 접근하는 Update() 라는 예제 함수에서 각 기법별로 데이터 동기화 기법을 사용하기 위한 코드 작성 예시를 나타내고 있다. lock의 경우 해당 코드 영역별로 고유한 lock_id로 lock을 요청/회수한다. 반면 TM의 경우, SW에서는 해당 코어별로 atomic section의 시작과 끝만을 알려주면 된다. 실제 접근하는 데이터의 충돌 판별은 HW에 의해 자동으로 수행되며, 충돌이 발생하여 abort가 되면 자동으로 해당 section의 처음 부분부터 다시 수행을 시도하는 방식이다.

표 3. 기법별 software 코딩 예제

Lock	Update() {
	acquire(lock_id);
	/*modify shared data X,Y*/
	release(lock_id);
	}

TM	Update() {
	begin_atomic(core_id);
	/*modify shared data X,Y*/
	end_atomic(core_id);
	}
	
	or
	
	Update() {
	atomic(core_id) {
	/*modify shared data X,Y*/
	}
	}

C-Lock	Update() {
	add(&X,sizeof(X));
	add(&Y,sizeof(Y));
	begin_c-lock(core_id);
	/*modify shared data X,Y*/
	end_c-lock(core_id);
	}

반면 C-Lock의 경우 표 1의 C-Lock행과 같이 공유 데이터별로 접근 주소 및 크기를 등록한 이후 C-Lock의 시작과 끝을 알려준다. C-Lock의 경우 lock 및 TM에 비해 부가적인 코드가 필요하지만, 이를 통해 단순히 고유 id가 아닌 접근하는 데이터들의 범위로써 권한 획득 유무를 판단하여 실질적인 충돌을 판별할 수 있기 위해서이다. 이를 통해 동일 코드영역이라도 처리하는 데이터가 다르다면 각 코어에서 동시에 진입이 가능하게 되어 병렬성을 향상시킬 수 있게 한다.

3.4 C-Lock - Hardware Implementation

C-Lock 기법을 지원하기 위한 C-Lock Manger는 각 코어별로 base address와 access size를 등록할 수 있는 레지스터 및 제어에 필요한 레지스터 등을 각 코어에 제공한다. 한 코어에 의해 base address와 size가 등록되면 history 추적이 가능하도록 global counter 값과 함께 해당 코어용 pool에 추가되고, begin_c-lock() 명령에 의해 다른 pool들에 등록된

표 4. 4개 코어에서의 공유 데이터 접근 시나리오 예제 (회색 음영은 clock이 차단된 구간)

	core0		core1		core2		core3	
Time	operation	en	operation	en	operation	en	operation	en
T0		1	add(@0x100,4) add(@0x104,4) begin_c-lock(1)	1		1		1
T1		1		1		1	add(@0x100,4) begin_c-lock(3)	0
T2		1	add(@0x108,4) begin_c-lock(1)	1		1		0
T3	add(@0x10a,1) begin_c-lock(0)	0		1		1		0
T4		0		1	add(@0x108,4) begin_c-lock(2)	0		0
T5		1	end_c-lock(1)	1		0		0
T6	end_c-lock(0)	1		1		0		0
T7		1	end_c-lock(1)	1		1		1
T8		1		1	end_c-lock(2)	1		1
T9		1		1		1	end_c-lock(3)	1

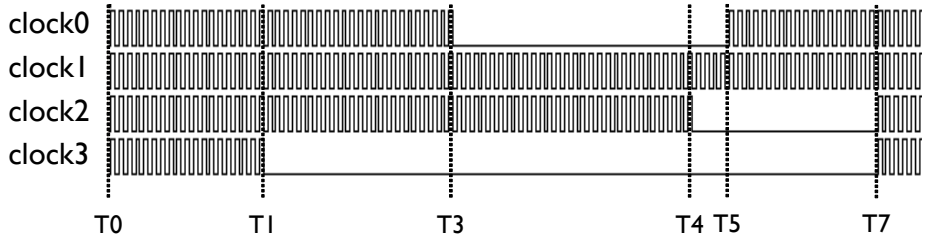

그림 2 C-Lock 적용시 코어별 클럭 동작 예시

item들과의 비교 수행을 통해 충돌 여부를 확인하게 된다. 충돌이 없을 경우 해당 코어에 연결된 clock이 계속 인가되어 동작을 지속하며, 충돌시에는 clock이 차단된다. 이 때 차단되었던 clock은 선점했던 코어가 해당 item을 end_c-lock()으로 해제시, 자동으로 나머지 item들간의 충돌여부를 다시 판단하여, 등록 순서상 가장 오래 전에 차단되었던 코어 쪽에 clock을 다시 인가하도록 구현되었다.

3.5 C-Lock - Operation Scenario

논의된 C-Lock 기법이 적용되었을 경우 동작하는 시나리오를 살펴본다. 표 2와 같이 4개의 멀티코어환경에서 공유 데이터들에 대해 접한다고 가정했을 때, C-Lock 결과에 의해 그림 2와 같이 해당 자원이 이미 선점된 경우 해당 코어에 인가되는 clock이 차단되고, 다른 코어에 의해 선점된 자원이 반환되면서 해당 코어로 다시 clock이 인가되면서 동작하게 된다.

표 2에서 회색 음영으로 나타난 셀들과 그림 2에서의 clock이 차단된 구간은, 권한 요청시 이미 다른 코어에 의해 해당 공유 데이터가 선점되었기 때문에 해당 코어로 인가되는 clock이 차단된 구간을 나타내고 있다. 예를 들어 T3에서 core0가 0x10a번지의 크기 1인 부분에 대한 사용 권한 요청시, 이미 0x108~0x10b 부분이 core1에 의해 T2부터까지 선점되고 있기 때문에 권한을 획득하지 못하게 되고, 따라서 core0로 인가되는 clock이 차단되고 있다. 한편, core1이 사용완료 후 T5에서 반환할 때, core0에 연결된 clock이 다시 인가되기 시작하는 것을 확인할 수 있다.

즉, C-Lock을 통해 공유 데이터간 동기화를 적용할 경우, 그림 2와 같이 데이터 충돌이 일어나는 구간동안 clock을 차단시킴으로써 적극적으로 동적 전력 소모를 방지할 수 있게 된다.

Ⅳ. 실험 환경 및 결과

4.1 C-Lock on MPARM

MPARM[4]은 SystemC 기반으로 작성된 멀티코어 환경을 지원하며, testbench를 수행한 cycle 정보 및 bus 관련 지표, 각 컴포넌트별 소모 에너지 등의 다양한 정보를 수집할 수 있는 cycle accurate한 상위수준 모델 simulator 환경이다. 이 기존의 MPARM환경에 C-Lock 기능을 구현하여 실험환경을 구축하였다.

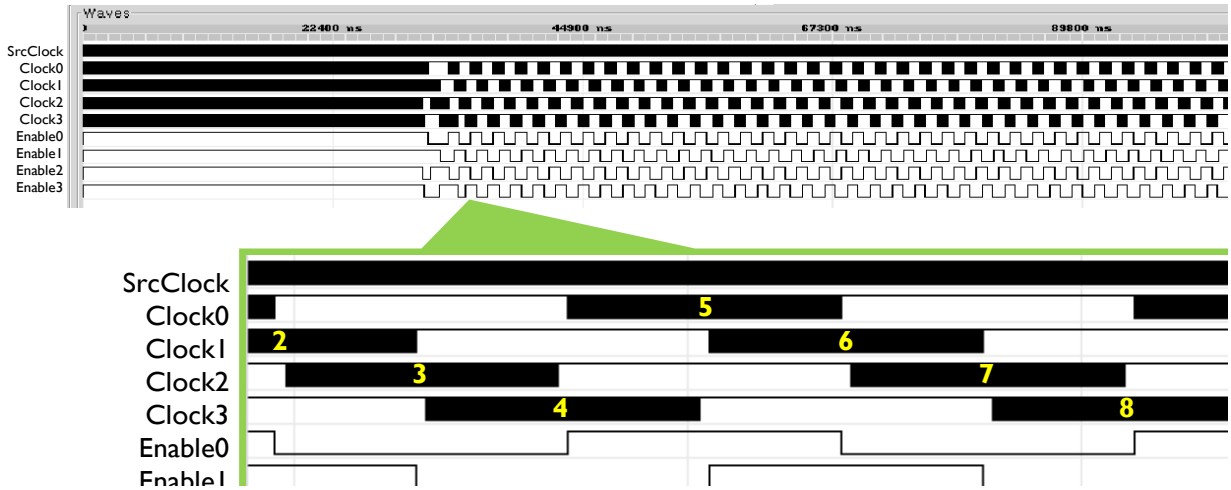

그림 3 clock/enable 신호 @counter w/ C-Lock

4.2 Evaluation – counter benchmark

counter 벤치마크는 동일한 공유 데이터 영역에 대해 4개의 멀티코어가 0에서 255까지 동시에 카운트를 수행한다. 즉, 공유 데이터에 대한 충돌이 빈번하게 발생하는 counter 벤치마크에 C-Lock을 적용한 결과, 그림 3과 같이 각 코어에 연결된 clock이 enable 신호에 의해 제어되며 동작하는 것을 확인할 수 있었다.

표 5. counter 벤치마크 적용 결과

policy	total cycles	energy [pJ]	EDP [pJ·s]	EDP (normalized to lock)
C-Lock	56,873	15,071,175.76	4,285.71	38%
Lock	75,223	29,619,430.60	11,140.31	100%
EmbeddedTM	99,086	26,339,860.13	13,049.56	117%

파란색 – best, 빨간색 – worst

단순히 성능만이 아닌 소모 에너지까지 고려하여 평가하여 임베디드 시스템에 적합한 평가를 할 수 있도록 Energy-Delay Product(EDP)를 성능지표로 삼았다. 기존 기술인 lock 및 EmbeddedTM과 비교 평가 결과, 표 3와 같이 EDP 측면에서 C-Lock 기법을 사용할 경우 lock 기법 대비 62%, EmbeddedTM 기법 대비 79%의 효율성 향상을 확인할 수 있었다. 대조군인 TM(Embedded TM)의 경우 에너지 효율성을 위해 roll-back시 random back off 하는 동안 전력을 차단하는 등의 기법을 통해 lock기법 대비 에너지 소모는 적었으나, 투기적 수행으로 인한 수행 성능 저하를 가져왔으며, 결국 EDP가 lock보다 17%의 증가하는 결과를 보인다.

V. 결론 및 향후 연구 방향

C-Lock은 성능뿐 아니라 전력소모에 대한 고려도 필요한 임베디드 시스템에 있어서 성능과 에너지, 양측면에 모두 부합하는 효율적인 공유 데이터 동기화 기법이다. EDP로 평가한 결과 counter 벤치마크의 경우 C-Lock이 최적의 기법임을 확인하였다.

향후 다양한 벤치마크에 적용하여 C-Lock의 효과를 확인할 것이며, 실질적인 데이터 의존성에 따른 동작을 할 수 있도록 C-Lock 기능을 개선할 계획이다.

Acknowledgement: 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2010-0025423, No. 2010-0026822).

참고문헌

[1] Maurice Herlihy and J. E. B. Moss. Transactional memory. architectural support for lock-free data structures. In Conference Proceedings - Annual Symposium on Computer Architecture, pages 289 - 300, 1993.

[2] H. Grahn. Transactional memory. Journal of Parallel and Distributed Computing, 70(10):993 - 1008, 2010.

[3] C. Ferri, S. Wood, T. Moreshet, R. Iris Bahar, and M. Herlihy. Embedded-TM: Energy and complexity-effective hardware transactional memory for embedded multicore systems. Journal of Parallel and Distributed Computing, 70(10):1042 - 1052, 2010.

[4] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. Mparm: Exploring the multiprocessor soc design space with systemc. Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, 41(2):169 - 182, 2005.