情報保護學會論文誌
第 21 卷 第 3 號, 2011. 6

동기화 오버헤드를 고려한 AES-CCM의 병렬 처리*

정 용 화,[1†] 김 상 춘[2‡]
[1]고려대학교, [2]강원대학교

Considering Barrier Overhead in Parallelizing AES-CCM*

Yongwha Chung,[1†], Sangchoon Kim[2‡]
[1]Korea University, [2]Kangwon National University

요 약

본 논문에서는 현재 IEEE 802.11i에서 암호화/메시지 인증 표준으로 제안되고 있는 AES-CCM의 효율적인 병렬처리 방법을 제안한다. 특히, 데이터 종속성이 존재하는 메시지 인증 계산을 병렬처리 하기 위해서는 프로세서간 동기화가 필요한데, 멀티코어 프로세서에서는 동기화 구현을 어떻게 하였는지에 따라 매우 다양한 동기화 성능을 제공하고 있다. 본 논문에서는 AES-CCM의 계산 특성과 멀티코어 프로세서의 동기화 성능을 고려하여 전체 수행시간이 최소화될 수 있는 병렬 처리 방법을 비교 분석한다.

ABSTRACT

In this paper, we propose workload partitioning methods in parallelizing AES-CCM which is proposed as the wireless encryption and message integrity standard IEEE 802.11i. In parallelizing AES-CCM having data dependency, synchronizations among processors are required, and multi-core processors have a very large range of synchronization performance. We propose and compare the performance of various workload partitioning methods by considering both the computational characteristics of AES-CCM and the synchronization overhead.

Keywords: AES-CCM, Barrier Overhead

I. 서 론

최근 MPEG 비디오 스트림 등 대용량 멀티미디어 데이터 사용이 증가함에 따라, 이러한 대용량 데이터에 대한 정보 보호 문제가 이슈화되고 있다. 특히, 데이터의 기밀성을 보장하는 방법 외에 HMAC(Hash-based Message Authentication Code) 등과 같은 메시지 인증 기법을 추가적으로 적용할 필요가 있다. 여러 가능한 암호화 알고리즘들 중 블록 암호 표준인 AES 운용모드 중 CCM(Counter with CBC-MAC)[1]은 암/복호화 하는 수행속도가 빠르고, 한 번의 수행만으로 기밀성과 무결성을 동시에 보장할 수 있는 특성을 가지며, 무선 표준에 채택되는 등 많은 관심을 받는 방법이다. 이러한 AES-CCM은 높은 계산 요구량으로 인하여 실시간 처리를 위해서는 전용 하드웨어 칩을 사용하는 것이 현재의 일반적 추세이다. 본 논문에서는 AES-CCM의 실시간 처리를 위하여 전용 하드웨어를 사용하는 대신, 최근 그 사용 범위를 자립형(standalone) PC 뿐만 아니라 임베디드 시스템으로까지 넓혀가고 있는 범용 멀티코어 프로세서[2]를 활용하여 병렬화하는 방법을 제안한다. 이러한 방법은 적은 비용으로 성능을 높일 수 있어 효율적이다.

접수일(2010년 9월 10일), 수정일(2011년 3월 7일), 게재확정일(2011년 3월 21일)
* 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2010-0027794)
† 주저자, ychungy@korea.ac.kr
‡ 교신저자, kimsc@kangwon.ac.kr

AES-CCM의 효과적인 병렬화를 위하여 먼저 그 계산 특성을 분석해보면, 블록 데이터의 종속성이 없는 암호화 부분과 종속성이 있는 메시지 인증 부분으로 구분할 수 있다. 암호화 부분과 인증 부분은 서로 독립적이어서 동시에 처리하는 것이 가능하다. 암호화 부분은 블록 데이터 간에 종속성이 없으므로 블록 수준의 병렬 처리가 가능하다. 즉, 사용되는 코어 수가 늘어나면 실행 시간도 그에 비례하여 줄어든다. 그러나 메시지 인증 부분은 암호화 부분과 달리 블록 데이터간 종속성으로 블록 수준의 병렬처리가 불가능하고, 대신 블록 내부 계산을 병렬 처리하는 것이 필요하다.

블록 내부 계산의 병렬화는 이들을 병렬 처리하는 코어들 간에 동기화를 자주 발생시킨다. 이러한 동기화는 블록 내부 계산에 드는 병렬 실행 시간과 비교하여 상대적으로 비용이 크고 구현 방법에 따라 그 오버헤드가 상당히 상이할 뿐만 아니라, 사용하는 코어의 개수에 따라 함께 증가한다는 문제가 있다[3, 4]. 따라서 동기화 오버헤드를 고려하여, 메시지 인증 부분에 할당될 최적의 코어 수를 도출할 필요가 있다. 본 논문에서는 "태스크 수준 병렬성"을 활용하여 암호화 부분과 인증 부분의 병렬화가 동시에 수행되는 등의 다양한 부하 분산 방법들을 제안하고 그 성능을 분석한다.

II. AES-CCM 병렬 처리

AES는 DES를 대체하기 위해 제안된 블록 암호화 알고리즘으로, 하나의 블록을 암호화하기 위하여 SubBytes/ShiftRows/MixColumns/AddRoundKey로 구성된 계산 단계를 수행한다. 또한, 데이터 크기가 큰 경우 입력 데이터를 블록 단위로 구분하여 암호화하고 블록간에는 다양한 운용 모드를 정의하고 있다. 다양한 운용모드들 중에서 CCM은 하나의 key로 기밀성과 무결성을 동시에 보장할 수 있다[1]. 즉, CCM은 운용모드 중 하나인 CTR을 이용하여 데이터의 기밀성을 제공하면서, CBC-MAC을 이용하여 데이터 인증 및 무결성을 제공한다. 이러한 AES-CCM은 알려진 두 가지 표준 기술을 사용하기 때문에 신뢰성이 높다. 또한, CBC-MAC에서 인증과 암호화에 다른 키를 사용하는 것과 다르게, 하나의 키로 두 가지 목적에 모두 사용 가능한 장점이 있다. CCM을 이용할 경우, 암호화는 필요 없고 무결성 검증만 필요로 하는 데이터가 포함된 경우, 추가적인 암호문의 오버헤드 없이 검증할 수 있어 데이터를 다루기가 쉽다. 뿐만 아니라 CCM은 동일 함수를 사용하여 암호화와 인증을 모두 수행하기 때문에 비교적 작은 크기의 코드로 구현 할 수 있다. 이러한 특성의 CCM은 다른 운용모드들에 비해 융통성 있는 적용을 가능하게 해 준다.

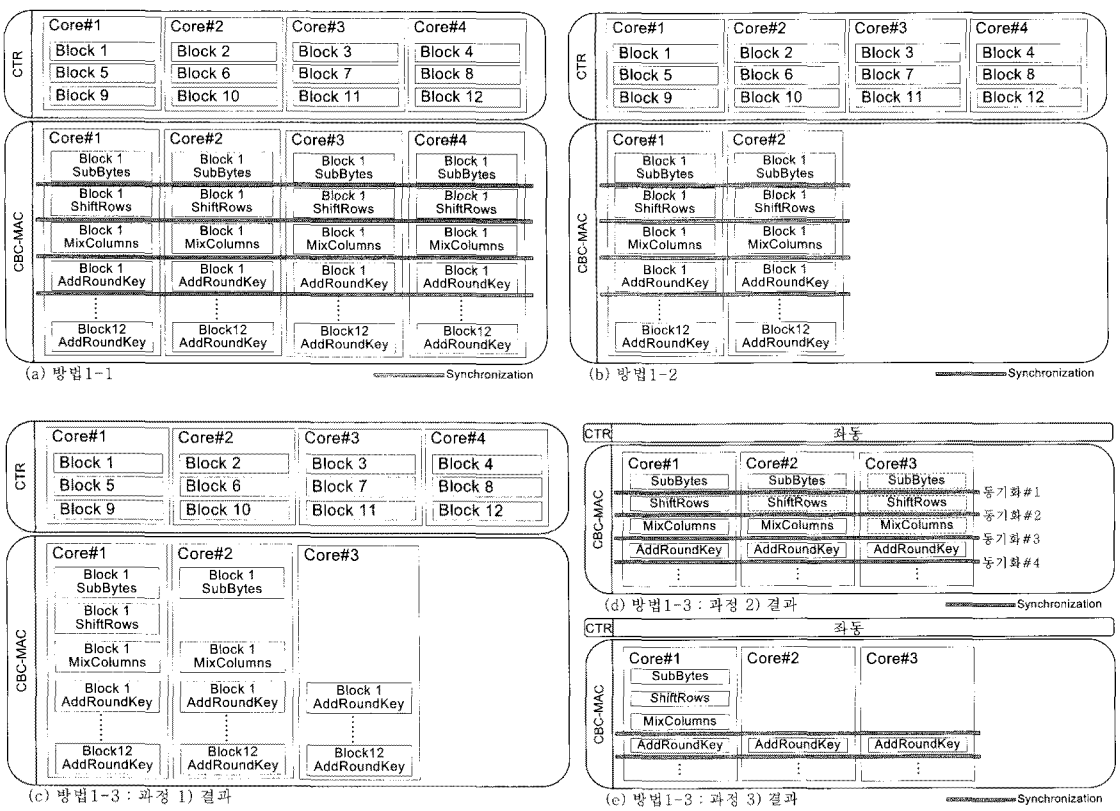
AES-CCM을 이용하여 대용량 멀티미디어 데이터의 기밀성과 무결성을 보장하고자 하는 경우에 AES-CCM의 전체 수행시간 대부분을 차지하는 메시지 암호화/인증에서 "데이터 수준 병렬성"을 활용하기 위해서는 일반적으로 블록 데이터 사이의 종속성이 없어야 한다. 그러나 블록 데이터 사이의 종속성이 없는 CTR 모드와 달리, CBC-MAC의 경우 블록간의 체이닝으로 종속성이 존재하여 블록별로 순차 처리해야 하고 정확한 블록내 계산을 위해서는 코어간 동기화를 삽입하여야 한다. 즉, [그림 1](a)와 같이 주어진 4-코어를 다 활용하여 CTR을 병렬처리하고, 블록 데이터간 종속성이 있는 CBC-MAC의 경우 블록내 계산을 주어진 4-코어에서 병렬처리하기 위하여 동기화가 삽입되는 부하 분산 방법(방법1-1: CTR을 먼저 처리하고 CBC-MAC을 처리하며, CBC-MAC 처리에 주어진 코어를 모두 이용)을 생각할 수 있다. (설명을 위하여 본 논문에서 "4-코어"는 4개의 코어를, "코어#4"는 4번째 코어를 구분하여 의미한다. 또한, [그림 1]과 [그림 2]에서는 12개의 블록으로 구성된 데이터를 AES-CCM으로 변환하는 예를 나타낸다.)

그러나, 실제 멀티코어 프로세서에서 코어간 동기화 오버헤드가 상당하고 코어의 개수에 따라 증가하기 때문에, 동기화 오버헤드를 고려한 메시지 인증 부분(CBC-MAC) 전체에 할당될 최적 코어 수를 도출할 필요가 있다. 즉, 블록 데이터 사이의 종속성이 없어 주어진 4-코어를 모두 이용하는 CTR 모드와 달리, 블록 데이터 사이 및 블록내 계산 단계의 종속성이 있는 CBC-MAC 전체를 동일한 수의 코어로 수행하는 경우의 최적 코어 수를 계산할 필요가 있다. 예를 들어 4개 코어를 이용한 경우, $\min_{1 \le i \le 4} \{ \frac{T_{SubBytes} + T_{ShiftRows} + T_{MixColumns} + T_{AddRoundKey}}{i} + 4T_{syn\_i} \}$(여기서 Tsyn_i는 i개 코어의 동기화 시간이며, Tsyn_1=0으로 한다. 또한, 4개 코어를 이용하는 경우 i의 최대값은 4이며, Tsyn_i가 4번 곱해지는 이유는 블록내 계산 단계가 4개 단계로 구성되기 때문임.)에 의하여 최적 코어 수를 결정할 수 있다. [그림 1](b)에 동기화 오버헤드와 CBC-MAC 계산의 상관관계에 의하여 CBC-MAC 부분 전체를 2개의 코어에서 수행하는 부하 분산 방법(방법1-2: CTR을 먼저 처리하고

CBC-MAC을 처리하며, CBC-MAC 전체에 최적의 코어 수 도출)의 예를 나타내었다.(4개 코어를 이용한 경우, 최적 코어 수가 1/2/3/4의 네가지 경우가 있을 수 있다.)

마지막으로, CBC-MAC의 블록내 계산 단계인 SubBytes, ShiftRows, MixColumns, AddRoundKey 각각은 서로 다른 처리량을 요구하므로, 동기화 오버헤드를 고려하여 각각의 계산 단계에 최적인 코어를 다르게 설정하는 부하 분산 방법(방법1-3: CTR을 먼저 처리하고 CBC-MAC을 처리하며, CBC-MAC 단계별 최적 코어 수 도출)도 생각할 수 있다.

본 논문에서는 1) "단계별 최적 코어 수" 결정, 2) "동기화 수행 코어 수" 결정, 3) "동기화 제거 여부" 결정으로 구성된 알고리즘을 제안한다. 예를 들어 4개 코어를 이용한 경우, $\underset{1 \le i \le 4}{Min}\left\{\frac{T_{SubBytes}}{i}+T_{syn\_i}\right\}$, $\underset{1 \le i \le 4}{Min}\left\{\frac{T_{ShiftRows}}{i}+T_{syn\_i}\right\}$, $\underset{1 \le i \le 4}{Min}\left\{\frac{T_{MixColumns}}{i}+T_{syn\_i}\right\}$, $\underset{1 \le i \le 4}{Min}\left\{\frac{T_{AddRoundKey}}{i}+T_{syn_i}\right\}$로 계산된 단계 j별 최적 코어수 nj가 각각 2(코어#1/코어#2 참여), 1(코어#1 참여), 2(코어#1/코어#2 참여), 3(코어#1/코어#2/코어#3 참여)이라 가정하자([그림 1](c) 참조). 이때 데이터 종속성이 있는 CBC-MAC의 정확한 계산을 위해서는 첫번째 단계인 SubBytes에 참여한 코어#2가 SubBytes-ShiftRows 사이의 동기화에 참여해야 하고, 두번째 단계인 ShiftRows에 참여하지 않은 코어#2가 ShiftRows-MixColumns 사이의 동기화에 참여해야 한다. 유사한 이유로 SubBytes/ShiftRows/MixColumns 계산에 참여하지 않지만 네번째 단계인 AddRoundKey에 참여하는 코어#3는 MixColumns-AddRoundKey 사이의 동기화 뿐만 아니라 SubBytes-ShiftRows, ShiftRows-MixColumns 사이의 동기화에 참여해야 한다. 유사한 이유로 SubBytes/ShiftRo-ws/MixColumns 계산에 참여하지 않지만 네 번째 단계인 AddRoundKey에 참여하는 코어#3는 MixColumns-AddRoundKey 사이의 동기화 뿐만 아



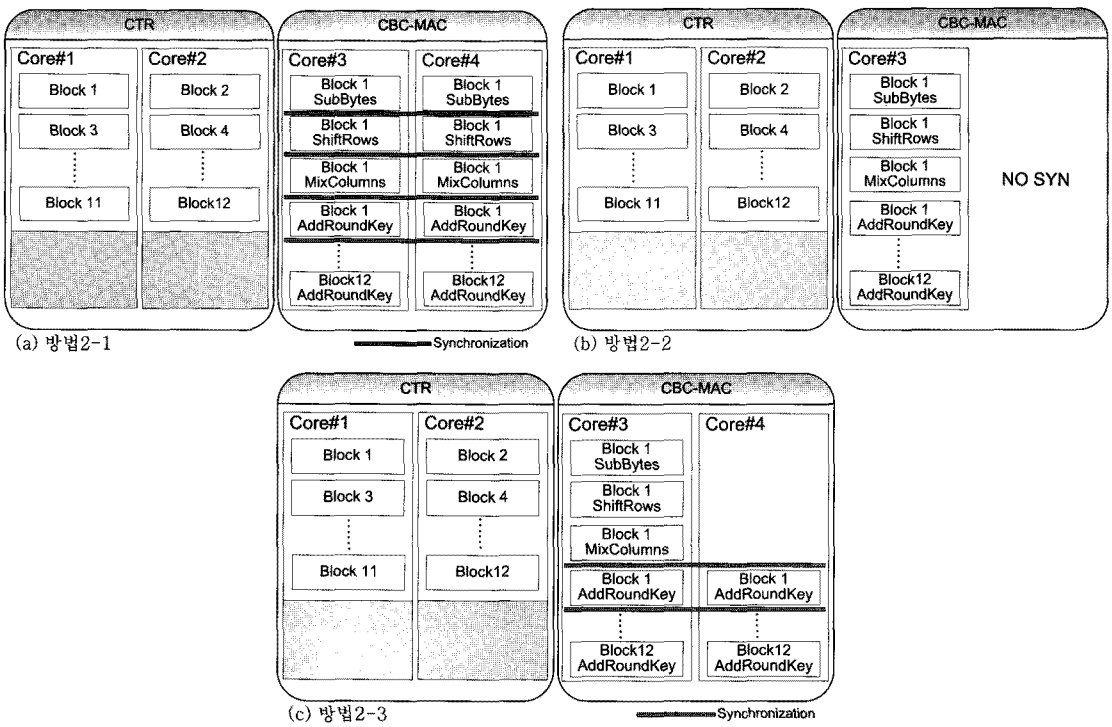
〔그림 1〕 4-코어에서 "데이터 수준 병렬성"만을 이용한 AES-CCM 병렬처리 방법

니라 SubBytes-ShiftRows, ShiftRows-MixColumns 사이의 동기화에 참여해야 한다.(병렬 프로그래밍 특성상 동기화 시점을 명시할 수 없는 코어#3는 MixColumns-AddRoundKey 사이의 동기화에만 참여하면 되지만, 동기화 함수를 한번만 호출하고 AddRoundKey 계산을 시작하면 코어#1이 첫 번째 동기화 함수 호출 후 ShiftRows를 계산하는 시점에 코어#3가 AddRoundKey 계산을 수행하여 오류가 발생한다는 문제가 있다.)

즉, 최적 코어수 nj를 필요로 하는 단계 j와 최적 코어수 nj+1을 필요로 하는 단계 j + 1로 구성된 프로그램의 올바른 동작을 보장하기 위해서는 Max(nj, nj+1)개의 코어간 동기화가 필요하며, 최종적으로 4개 단계로 구성된 CBC-MAC 전체적으로는 max = Max(nj)개의 동기화 수행 코어 수가 적용된다. 예를 들어, 〔그림 1〕(d)에 계산 단계별 최적 코어 수가 각각 n1 = 2, n2 = 1, n3 = 2, n4 = 3으로 결정된 경우 max = 3이 되며, 매 단계별 계산에 참여하는 코어#1, 코어#2, 코어#3간의 동기화가 필요하다.(각 단계는 병렬로 처리 가능하므로, 과정 1)에서 계산된 SubBytes/ShiftRows/MixColumns 단계의 최적 코어수가 각각 2/1/2이지만 과정 2)에서 오류 방지를 위하여 3개 코어의 동기화가 필요한 만큼, 각 단계에 3개 코어가 다 참여하는 것이 수행시간을 단축할 수 있음)

마지막으로 과정 2)에서 결정된 동기화들 각각에 대하여, 제거하는 것(즉, k번째 동기화에 대하여 인접 계산 단계인 k 및 k+1 계산을 순차 처리)과 제거하지 않는 것(즉, k 및 k+1 계산을 병렬 처리)과의 비교가 필요하다. 예를 들어, SubBytes-ShiftRows 사이의 동기화 #1에 대하여 $\min(\frac{T_{SubBytes}}{3}+\frac{T_{ShiftRows}}{3}+T_{syn_3}, T_{SubByte}+T_{ShiftRows})$을 구한다. 이때, 오류 방지를 위하여 nk = max가 되는 계산 단계 k의 인접 동기화 #k-1, #k는 제거할 수 없다(즉, AddRoundKey의 인접 동기화 #3, #4는 제거할 수 없다). 예를 들어, 동기화 #1, #2가 제거되어 SubBytes/ShiftRows/Mix- Columns은 순차 처리하고 AddRoundKey는 3개 코어로 병렬 처리하는 과정 3)의 결과를 나타내면 〔그림 1〕(e)와 같다.

이상 언급한 부하 분산 방법들은 "데이터 수준 병렬성"만을 이용하였으나, "데이터 수준 병렬성" 외에 "태



〔그림 2〕 4-코어에서 "데이터 수준 병렬성"과 "태스크 수준 병렬성"을 동시에 이용한 AES-CCM 병렬처리 방법

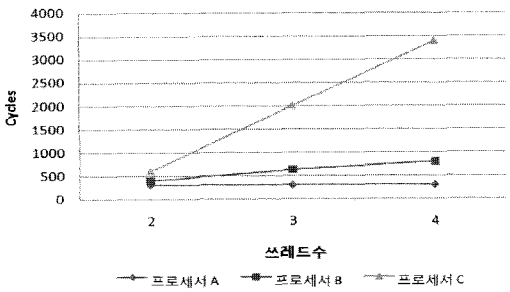
스크 수준 병렬성"까지 동시에 이용한다면 더 좋은 성능을 기대할 수도 있다. 즉 암호화 부분과 메시지 인증 부분의 계산 시간이 유사하기 때문에, [그림 2]와 같이 CTR과 CBC-MAC에 각각 2-코어를 할당하여 두 부분을 함께 병렬 처리하는 방법(방법 2-1: CTR과 CBC-MAC을 동시에 처리하며, CBC-MAC 처리에 남은 코어를 모두 이용)을 생각할 수 있다.

또한, "데이터 수준 병렬성"만을 이용한 솔루션의 경우처럼, 동기화 오버헤드와 CBC-MAC 계산의 상관관계에 의하여 CBC-MAC 부분에 할당될 최적의 코어 수를 도출할 수 있다(방법2-2: CTR과 CBC-MAC을 동시에 처리하며, CBC-MAC 전체에 최적의 코어 수 도출). 예를 들어, 동기화 오버헤드를 고려할 때 CBC-MAC 부분을 1개의 코어에서 수행하는 것이 2개의 코어를 사용하는 것보다 더 좋은 성능을 제공하는 경우를 [그림 2](b)에 나타내었다. 주의할 점은, 이 경우 블록 데이터간 종속성이 없는 암호화 부분에 3개의 코어를 할당할 수 있으나, 실질적인 성능 개선으로 연결되지는 않는다는 점이다. 즉, 암호화 부분에 3개의 코어를 할당하여 암호화 부분의 수행시간을 1/3로 줄일 수는 있으나, 전체 수행 시간은 암호화 부분과 유사한 계산 처리량을 요하는 메시지 인증 부분에 의하여 결정되기 때문이다. 마찬가지 이유로 암호화 부분에 1개의 코어를 할당하고 메시지 인증 부분에 3개의 코어를 할당하는 방법도 암호화와 인증 부분 각각에 2개의 코어를 할당하는 방법보다 좋은 성능을 제공할 수 없다.

마지막으로, CBC-MAC의 블록내 계산인 SubBytes, ShiftRows, MixColumns, AddRoundKey 각각의 서로 다른 처리량과 동기화 오버헤드를 고려하여 각각의 계산에 최적인 코어를 다르게 설정하는 부하 분산 방법(방법2-3: CTR과 CBC-MAC을 동시에 처리하며, 단계별 최적 코어 수 도출)도 생각할 수 있다. 예를 들어, [그림 2](c)에 SubBytes는 1-코어, ShiftRows는 1-코어, MixColumns는 1-코어, AddRoundKey는 2-코어로 할당된 예를 나타내었다.

III. 구현 및 실험 결과

2장에서 설명한 다양한 부하 분산 방법의 성능을 분석하기 위하여, 먼저 AES 블록내 계산인 SubBytes, ShiftRows, MixColumns, AddRoundKey 각각의 수행 시간을 멀티코어 시뮬레이터인



[그림 3] 프로세서별 동기화 오버헤드의 예

Rapsim[5]으로 측정하였다. 그리고, 코어의 개수를 P라 할때 멀티코어간 동기화를 구현하는 다양한 기법들을 직접 하드웨어로 구현한 방법(cost = O(1)), 효과적인 소프트웨어로 구현한 방법(cost = O(logP)), 효과적이지 못한 소프트웨어로 구현한 방법(cost = O(P))으로 구분할 수 있다.

예를 들어, SubBytes = 14,707 싸이클, ShiftRows = 26 싸이클, MixColumns = 212 싸이클, AddRoundKey = 140 싸이클이 소요되고, [그림 3]과 같은 동기화 오버헤드를 갖는 가상의 4-코어 프로세서 A, B, C를 가정하자. 그러면, 각각의 프로세서를 이용하여 AES 블록내 계산인 SubBytes, ShiftRows, MixColumns, AddRoundKey를 1-코어, 2-코어, 3-코어, 4-코어로 수행한 결과(사용 코어 개수별 한번의 동기화 시간 포함)를 요약하면 [표 1]과 같다.

또한, [표 1]을 이용하여 100 블록 크기의 데이터를 블록당 10회 라운드로 AES-CCM 처리하는 경우, 2장에서 언급한 부하 분산 방법들의 수행 시간을 계산하면 [표 2]와 같다.(계산의 편의상 Key-Expansion 시간은 포함하지 않았고, 방법1-2에서 "1/2/3/4-코어"란 CBC-MAC 전체를 코어 1/2/3/4개에서 수행시킨 경우를 의미한다.) 즉, 프로세서 및 동기화의 성능에 따라 부하 분산 방법이 달라지는 방법1-2에서는 프로세서 A와 B에서 전체 CBC-MAC을 4-코어로, 프로세서 C에서는 2-코어로 할당하는 것이 최적의 수행시간을 보장한다. 반면, 방법1-3에서는 프로세서 A와 B의 경우 SubBytes만 4-코어로 수행하고 나머지 ShiftRows Mixcolumns, AddRoundKey는 1-코어에서 수행하는 것이 최적의 성능을 제공한다. 그러나, 프로세서 C에서는 SubBytes를 3-코어로 수행하고 나머지는 1-코어로 수행하는 것이 최선이다. 유사하게 방법2-2에서는 모든 프로세서가 전체 CBC-MAC을 2-코어로, 방법2-3

〔표 1〕 프로세서별 한개 블록의 AES 수행 시간
(단위 : 싸이클)

	계산	1-core	2-core	3-core	4-core
A	SubBytes	14,707	7,653	5,202	**3,976**
	ShiftRows	**26**	313	308	306
	MixColumns	**212**	406	370	353
	AddRoundKey	**140**	370	346	335
B	SubBytes	14,707	7,753	5,522	**4,476**
	ShiftRows	**26**	413	628	806
	MixColumns	**212**	506	690	853
	AddRoundKey	**140**	470	666	835
C	SubBytes	14,707	7,953	**6,902**	7,076
	ShiftRows	**26**	613	2,008	3,406
	MixColumns	**212**	706	2,070	3,453
	AddRoundKey	**140**	670	2,046	3,435

에서는 SubBytes만 2-코어로 수행하고 나머지를 1-코어에서 수행하는 것이 최적의 성능을 제공한다. 종합적으로, 프로세서 A, B, C 모두 방법2-3이 가장 좋은 성능을 제공할 수 있다.

이상은 2장에서 언급한 다양한 부하 분산 방법의 성능을 비교하기 위하여 가상의 4-코어 프로세서를 가정하였으나, 실제 4-코어 프로세서를 이용하여 AES-CCM으로 처리한 결과는 다음과 같다. 먼저 다양한 프로세서 환경을 가정하여 Pthread〔6〕로 부하 분산 방법을 구현하였다. 그리고, Intel Core2 Quad CPU(2.33 GHz)와 AMD Phenom II CPU(3.2 GHz)로 측정한 2/3/4-코어에서의 동기화 오버헤드는 각각 8,732/13,1230/16,854 싸이클과 73,219/116,328/206,025 싸이클이었다. 예상과 달리 두 개의 프로세서 모두에서 상당한 크기의 동기화 오버헤드가 있음을 확인하였는데, 이는 Pthread 프로그램을 수행하기 위해 포팅한 리눅스가 실제 프로세서들의 특성을 최대한 이용하지 못한 것으로 판단된다. 그리고 이렇게 높은 동기화 오버헤드를 고려하면, 방법2-2에서 1-코어로 CBC-MAC 전체를 수행시킨 경우가 Intel(759,500 싸이클)과 AMD(2,420,000 싸이클) CPU 모두에서 가장 좋은 성능을 제공함을 확인하였다. 참고로 Intel과 AMD CPU의 1-코어를 이용한 순차 처리는 각각 1,519,000 싸이클과 4,840,000 싸이클이 소요되었다.

IV. 결 론

본 논문에서는 현재 암호화/메시지 인증 표준으로 제안되고 있는 AES-CCM을 병렬처리하여 대용량 데이터의 기밀성과 무결성을 효과적으로 보장하는 부하 분산 방법들을 제안하였다. 즉, AES-CCM의 계산 특성과 멀티코어 프로세서의 동기화 오버헤드를 고려하고 "데이터 수준 병렬성"과 "태스크 수준 병렬성"을 동시에 활용하는 다양한 부하 분산 방법을 고려하였다. 또한, 주어진 프로세서에서 AES 각 계산별 수행시간과 코어 개수별 동기화 시간이 주어지면, 전체 수행시간을 최소화할 수 있는 부하 분산 방법을 효과적으로 도출하는 방법론을 제안하였다. 시뮬레이션과 실측치를 이용한 4-코어 성능을 비교한 결과, 동일한 부하 분산 방법에 대하여 동기화 성능에 따라 최대 2.4배까지 차이가 나며, 동일한 프로세서에서도 부하 분산 방법에 의하여 최대 2.5배까지 성능 차이가 날 수 있음을 확인하였다.

〔표 2〕 부하 분산 방법별 수행 시간(단위 : 싸이클)

		A	B	C
방법1-1		8,741,250	10,741,250	21,141,250
방법1-2	1-코어	18,856,250	18,856,250	18,856,250
	2-코어	12,513,250	12,913,250	13,713,250
	3-코어	9,997,250	11,277,250	16,797,250
	4-코어	8,741,250	10,741,250	21,141,250
방법1-3		8,125,250	8,625,250	11,051,250
방법2-1		8,742,000	9,142,000	9,942,000
방법2-2	1-코어	15,085,000	15,085,000	15,085,000
	2-코어	8,742,000	9,142,000	9,942,000
방법2-3		**8,031,000**	**8,131,000**	**8,331,000**

참고문헌

[1] N. Dworkin, "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality," NIST Special Publication 800-38C, 2002.

[2] S. Akhter and J. Roberts, Multi-Core Programming - Increasing Performance through Software Multi-Threading, Intel Press, 2006.

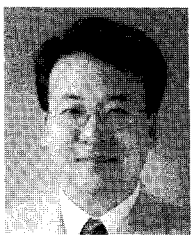
[3] J. Sampson, R. Gonzalez, J. Collard, N. Jouppi, M. Schlansker, and B. Calder, "Exploiting Fine-Grained Data Parallelism with Chip Multiprocessors and Fast Barriers," Proc. of MICRO, 2006.

[4] J. Sartori and R. Kumar, "Low-Overhead, High-Speed Multi-core Barrier Synchronization," LNCS, Vol. 5952, pp. 18-34, 2010.

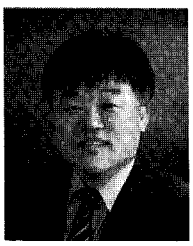
[5] K. Park, S. Choi, Y. Chung, W. Hahn, and S. Yoon, "On-Chip Multiprocessor with Simultaneous Multithreading," ETRI Journal, Vol. 22, No. 4, pp. 13-24, 2000.

[6] B. Barney, POSIX Threads Programming, http://www.llnl.gov/computing/tutorials/pthreads, 2006.

〈著者紹介〉



정 용 화 (Yongwha Chung) 종신회원
1984년: 한양대학교 전자통신공학과 학사
1986년: 한양대학교 전자통신공학과 석사
1997년: 미국 Univ. of Southern California 전기공학과(컴퓨터공학 전공) 박사
1986년~2003년: 한국전자통신연구원 생체인식기술연구팀 팀장
2003년~현재: 고려대학교 컴퓨터정보학과 교수
〈관심분야〉 성능 평가, 정보 보호, 멀티미디어데이터 보호



김 상 춘 (Sangchoon Kim) 종신회원
1986년: 한밭대학교 전자계산학과 학사
1989년: 청주대학교 전자계산학과 석사
1999년: 충북대학교 전자계산학과 박사
1983년~2001년: 한국전자통신연구원 정보보호연구단
2001년~현재: 강원대학교 정보통신공학과 부교수
〈관심분야〉 정보보호, 보안 시스템 설계 및 구현, 네트워크 및 RFID 보안, 개인정보보호, 융합보안