멀티코어 프로세서의 쓰레드-수준 병렬성을 활용한 AES-CCM 계산-통신 중첩화

(Computation-Communication Overlapping in AES-CCM Using Thread-Level Parallelism on a Multi-Core Processor)

이 은 지 † 이 성 주 †
(Eunji Lee) (Sungju Lee)

정 용 화 †† 이 명 호 †††
(Yongwha Chung) (Myungho Lee)

민 병 기 ††††
(Byoung-Ki Min)

요 약 최근 멀티코어 프로세서들이 범용 PC 뿐만 아니라 임베디드 시스템에서도 탑재될 만큼 그 사용이 보편화되고 있는 상황에서, 많은 멀티미디어 응용 프로그램이 이들을 활용하여 병렬화 되고 있다. 그러나 멀티미디어 데이터의 암호화와 같이 응용 프로그램에 데이터 종속성이 내재한 경우에는 멀티코어를 이용한 효과적인 병렬처리가 어렵다는 문제가 있다. 본 논문에서는 이러한 한계를 극복하기 위하여 유휴 코어를 이용하여 계산과 통신을 중첩시키는 병렬처리 기법을 제안한다. 특히, 주어진 멀티미디어 데이터를 처리하고 전송하는 문제를 응용 프로그램 수준의 파이프라인 설계 문제로 해석하여 최적의 파이프라인 단계 수를 도출하는 방법을 제안한다.

키워드 : 멀티코어, 쓰레드-수준 병렬화, 계산-통신 중첩화

Abstract Multi-core processors are becoming increasingly popular. As they are widely adopted in embedded systems as well as desktop PC's, many multimedia applications are being parallelized on multi-core platforms. However, it is difficult to parallelize applications with inherent data dependencies such as encryption algorithms for multimedia data. In order to overcome this limit, we propose a technique to overlap computation and communication using an otherwise idle core in this paper. In particular, we interpret the problem of multimedia computation and communication as a pipeline design problem at the application program level, and derive an optimal number of stages in the pipeline.

Key words : Multi-Core, Thread-Level Parallelism, Computation-Communication Overlapping

1. 서 론

최근 MPEG 비디오 스트림 등 대용량 데이터의 이용이 증가함에 따라, 이러한 대용량 데이터 전송시 정보 보호가 중요한 문제로 부각되고 있다. 전송되는 데이터의 정보 보호는 기밀성의 보장과 더불어 HMAC(Keyed-Hash Message Authentication Code) 등과 같은 메시지 인증 기법[1]을 추가적으로 적용하는 암호화 알고리즘[2]을 활용하여 무결성도 동시에 보장할 필요가 있다. 이러한 암호화 알고리즘은 높은 계산 요구량을 갖는다. 따라서 이들을 실시간으로 처리하기 위하여 전용 하드웨어 칩을 사용하는 것이 일반적인 추세이다.

본 논문에서는 전송되는 대용량 데이터의 실시간 정보 보호를 위한 전용 하드웨어를 사용하는 대신, 최근 범용 PC 뿐만 아니라 임베디드 시스템에서도 보편적으로 사용되고 있는 멀티-코어 프로세서[3]를 활용하여 암호화 알고리즘을 병렬화 함으로써 이러한 실시간 처리 요구를 만족시키는 방법을 제안한다. 현재 가장 널리 사용되는 블록 암/복호화 표준은 AES[4]인데, AES의 다양한 운용 모드 중에서도 CCM(Counter with CBC-MAC)[5]은 한 번의 수행만으로 기밀성과 무결성을 동시에 보장할 수 있는 장점 등으로 인하여 최근 많은 관심을 받는 방법이다[6]. 이러한 AES-CCM을 멀티코어 프로세서를 활용하여 병렬화한다면, 전용 하드웨어 칩을 사용하는 현재 솔루션과 비교하여 저렴하면서 실용적으

· 본 연구는 산업자원부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구결과임
· 이 논문은 제36회 추계학술발표회에서 '멀티코어 프로세서의 쓰레드-수준 병렬성을 활용한 계산-통신 중첩화'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 고려대학교 컴퓨터정보학과
achee@korea.ac.kr
peacfeel@korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터정보학과 교수
ychungy@korea.ac.kr
††† 정 회 원 : 명지대학교 컴퓨터소프트웨어학과 교수
myunghol@mju.ac.kr
†††† 정 회 원 : (주)아스텔 부설연구소 연구소장
min@astel.co.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 4월 16일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제8호(2010.8)

로 데이터의 기밀성과 무결성을 동시에 보장하는 솔루션을 확보할 수 있을 것으로 기대된다.

AES-CCM의 효과적인 병렬화를 위하여, 먼저 AES-CCM의 계산 특성을 분석한다. 특히, AES-CCM 처럼 데이터의 종속성이 알고리즘에 내재한 경우 일반적인 병렬처리 기법으로는 효과적인 성능 개선을 기대하기 어렵다[7,8]. 특히, CPU 칩 내의 코어 수가 증가하는 추세에서 응용프로그램 내에 존재하는 병렬성이 코어 수에 미치지 못하는 경우에는 CPU 내 귀중한 자원이 활용되지 못한다는 한계가 있다. 즉, AES-CCM 내의 순차성 때문에 일반적인 병렬처리 기법만을 적용[7,8]하면, 아무리 많은 코어를 사용하더라도 성능개선이 최대 2배(2.2절의 설명 참조)로 제약된다는 문제가 있다.

데이터 종속성으로 인한 성능의 한계를 극복하기 위하여 본 논문에서는 기본적인 병렬처리 기법 외에 추가로 멀티미디어 데이터의 계산 과정(Computation Process)과 통신 과정(Communication Process)을 중첩화(overlapping)한다. 일반적으로 계산-통신 중첩화는 네트워크 보조-프로세서(co-processor)가 장착된 분산메모리 플랫폼에서 non-blocking send/recv 명령어를 이용한 메시지 패싱 방식에서 많이 연구된 분야[9-12]이지만, 사용자 관점에서 통신 과정을 직접 제어할 수 있는 방법이 없다는 문제가 있다. 본 논문에서는 멀티-코어 중 유휴 코어를 이용하여 응용 프로그램 수준의 제어를 통한 중첩화를 제안하며, 이를 위하여 계산을 위한 코어와 통신을 위한 코어들이 파이프라인(pipeline) 방식으로 수행되도록 한다. 또한 그들이 Barrier Synchronization을 사용하여 동기화되는데 필요한 오버헤드를 최소화 할 수 있도록 최적의 파이프라인 단계 수를 도출하는 방법론을 함께 제안한다. 암호화 응용 프로그램의 병렬화 구현을 위하여 사용자 수준에서의 쓰레드-수준 병렬성(Thread-Level Parallelism)을 제공하는 Pthread[13]를 사용한다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장에서는 AES-CCM의 계산 특성과 병렬처리 시 발생되는 문제에 대해서 설명한다. 3장에서는 계산 과정과 통신 과정을 중첩시키는 병렬처리 방법 및 최적의 파이프라인 단계 수를 계산하는 방법론을 설명한다. 4장에서는 제안 방법의 실험 결과를 설명하고, 마지막 5장의 결론으로 논문을 마무리 한다.

2. 배 경

2.1 AES-CCM 계산 특성

현재 가장 널리 사용하는 블록암호화 알고리즘은 AES[4]이다. AES의 다양한 운용모드들 중에서 CCM[5]은 하나의 key로 기밀성과 무결성을 동시에 보장할 수 있다. 아래의 그림 1은 AES-CCM의 계산 특성을 보여주고 있다. CCM은 운용모드 중 하나인 CTR을 이용하여 데이터의 기밀성을 제공하면서, CBC-MAC을 이용하여 메시지 인증 및 무결성도 동시에 제공한다. 먼저, CBC를 이용하여 MAC으로 사용할 암호화 블록을 만들고, CTR을 이용해 메시지를 암호화한다.

AES-CCM은 알려진 두 가지 표준 기술을 사용하기 때문에 신뢰성이 높다. 또한, CBC-MAC에서 인증과 암호화에 다른 키를 사용하는 것과는 달리, 한 개의 키로 두 가지 모두 사용 가능한 장점이 있다. CCM을 이용할 경우, 암호화를 할 필요 없이 무결성 검증만 필요로 하는 데이터가 포함된 경우 추가적인 암호문 오버헤드 없이 암호화할 수 있어 데이터를 다루기가 쉽다. 뿐만 아니라 CCM은 암호화 하는 함수 하나만을 사용하여 암호화와 복호화를 모두 수행하기 때문에 비교적 작은 크기의 코드로 구현 할 수 있다. 이러한 특성의 CCM은 다른 운용모드들에 비해 융통성 있는 적용이 가능하다.

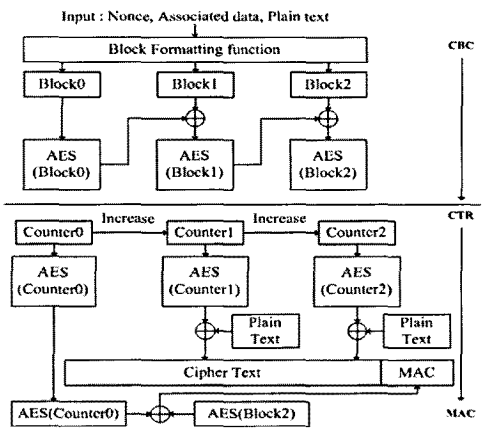


그림 1 AES-CCM 계산 특성

2.2 AES-CCM 병렬화

그림 2는 멀티코어 환경에서 효과적인 병렬화를 위한 AES-CCM의 구조를 보여준다. Key 초기화와 메시지의 헤더 검증을 위한 수행은 작은 데이터를 처리하기 때문에 전체 수행시간에 큰 영향을 미치지 않는다. MAC 계산도 암호화 처리가 된 블록에 대하여 XOR 연산만 실행하기 때문에 병렬처리가 필수적이지는 않다. 그러나 메시지 암호화/인증 부분은 전체 메시지를 암호화/인증 블록의 개수만큼 처리해야 하기 때문에 전체 수행시간의 대부분을 차지한다. 따라서 이 부분에 대한 병렬화가 필요하다.

메시지 암호화/인증은 CTR을 이용하여 메시지를 암호화하는 계산과 CBC-MAC을 이용하여 MAC 블록을 생성하는 계산을 수행한다. 이 두 과정은 모두 AES 암호화

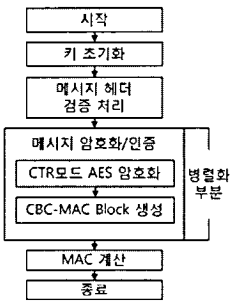


그림 2 AES-CCM 병렬화

를 이용하고 같은 크기의 메시지를 처리하기 때문에 거의 비슷한 수행 시간이 소요되는데, 전체 수행시간을 줄이기 위하여 이 두 부분을 효과적으로 병렬처리 해야 한다.

전체 수행시간의 대부분을 차지하는 메시지 암호화/인증에서 data-level parallelism을 활용[8]하기 위해서는 데이터 사이의 종속성이 없어야 한다. 그러나 CTR 모드와 달리, CBC-MAC의 경우 블록간의 체이닝으로 종속성이 존재하므로 데이터 병렬화가 가능한 CTR만을 병렬처리 하는 방법을 생각할 수 있다. 그러나 CTR의 메시지 암호화와 CBC-MAC 블록 생성의 수행시간이 거의 비슷하기 때문에 이러한 병렬화로는 수행시간을 크게 줄이지 못한다. 즉, 4-코어를 사용하는 경우 CTR은 1/4로 줄지만 CBC-MAC 블록 생성 시간은 감소하지 않기 때문에, 메시지 암호화/인증 부분의 speed-up은 대략 1.6(CTR: $\frac{T_{seq}}{2} \rightarrow \frac{T_{seq}}{2} \times \frac{1}{4}$, CBC-MAC: $\frac{T_{seq}}{2} \rightarrow \frac{T_{seq}}{2}$, 전체 실행시간: $T_{seq} \rightarrow \frac{T_{seq}}{8} + \frac{T_{seq}}{2} = \frac{5T_{seq}}{8}$)에 불과하다.

위의 문제를 해결하기 위해 CTR와 CBC-MAC을 함께 병렬처리, 즉 task-level parallelism을 활용하는 방법을 생각할 수 있다[7]. 즉, 병렬처리 구간에서 한 개의 코어는 CBC-MAC을 수행하고 나머지 코어들은 CTR를 수행하는 방법으로 병렬처리를 할 수 있다. 그러나 이러한 방법으로는 4-코어를 사용하더라도 speed-up이 최대 2 에 머무는 한계를 보인다.(전체 실행시간: $T_{seq} \rightarrow \max\left(\frac{T_{seq}}{8}, \frac{T_{seq}}{2}\right) = \frac{T_{seq}}{2}$) 더 좋은 성능을 얻기 위해서는 일반적인 병렬처리 기법을 적용[7,8]하는 것 외에 추가적인 기법의 개발이 필요하다.

3. 쓰레드-수준의 병렬성을 이용한 계산-통신 중첩

본 논문에서는 데이터 종속성이 있는 AES-CCM 계산 과정과 이를 전송하는 통신 과정으로 구성된 문제를 계산-통신 중첩(overlapping) 기법을 적용하여 병렬화 함으로써 성능을 향상시키는 병렬화 방법론에 대하여 설명한다. 간결한 설명을 위하여 2-코어 시스템을 이용하여 AES-CCM 계산 과정과 이를 전송하는 과정에서의 중첩과 관련된 이슈만을 기술한다.

3.1 응용 프로그램 수준의 계산-통신 중첩화

본 논문에서는 사용자 수준에서의 쓰레드-수준 병렬성(Thread-Level Parallelism)을 제공하는 Pthread[13]를 사용하여 AES-CCM을 병렬화한다. 병렬화를 위하여 계산과 통신을 중첩시키고, 이를 응용 프로그램 수준(즉, coarse-grain level)의 파이프라인(pipeline) 설계 문제로 해석한다. 특히, 파이프라인 단계(stage) 수를 증가시키면 비례적으로 성능이 개선되는 일반적인 파이프라인 문제와 달리, 계산 과정을 담당하는 코어와 통신 과정을 담당하는 코어를 매 단계마다 동기화(barrier synchronization) 시키는 오버헤드를 감안하여 파이프라인 단계 수를 결정해야 한다.

먼저, 그림 3에서는 2-코어 시스템에서 각각의 코어에 계산 과정과 통신 과정을 할당하여 중첩 수행시키는 모습을 보여준다. 전체 데이터의 크기가 M(단위: 바이트)이고 계산-통신 과정을 각각 m번의 파이프라인 단계에 걸쳐 수행한다(그림에서 $Comp_1$은 데이터1에 대한 계산 과정을, $Comm_1$은 데이터1에 대한 통신 과정을 의미한다). 이때 한번의 계산 단계가 수행되어야 해당되는 통신 단계가 수행될 수 있기 때문에, 그림 3과 같이 각 단계마다 동기화가 추가된 파이프라인(Pipeline) 형태로 계산-통신 과정을 중첩하여 병렬처리가 가능하다. 주의할 점은 특정 응용을 대상으로 한 임베디드 시스템을 가정하여 계산 시간은 일정하다고 볼 수 있으나, 여러 요소들에 의하여 영향을 받는 통신 시간 설정이 문제가 될 수 있다. 본 논문에서는 충분한 데이터 수집을 통하여 주어진 임베디드 시스템 환경의 통신 시간을 예측할 수 있다고 가정한다(또는 counter 등을 이용하여 barrier 보다 완화된 동기화 기법[14,15]을 이용할 수도 있다).

또한, 일반적인 병렬처리에서 공유 데이터를 동시에 접근하려고 할 때 발생하는 문제를 해결하기 위하여 동기화를 적용해야 한다. 통상 프로세서들이 하나의 칩 내에 집적된 멀티코어에서는 동기화 오버헤드가 적을 것으로 예상하나, 실제로 측정한 결과 상당한 오버헤드가 있다는 것을 확인하였다(4절 참조). 따라서 전체 수행시간을 줄이기 위해 파이프라인 단계 수 m을 늘려야 하

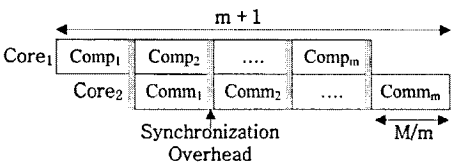


그림 3 2-코어 시스템의 계산-통신 중첩화 병렬처리

는 요구와 단계당 동기화 오버헤드를 줄이기 위해서는 단계 수를 줄여야 하는 상충 관계(trade-off)에서 최적의 단계 수를 구해야 하는 문제가 발생한다.

3.2 파이프라인 단계 수 도출

그림 3에서 제시된 예에서, 전체 데이터의 크기가 M (단위: 바이트)이고 계산-통신 과정 각각을 m 번의 단계로 구분하여 중첩하는 경우, 다음과 같은 수식으로 전체 수행 시간을 나타낼 수 있다.

$$(m+1)\frac{M}{m}R_{max} + mT_{syn} \quad (1)$$

여기서 R_{max}은 계산 또는 통신 과정 중 수행 시간이 더 오래 걸리는 과정의 처리 또는 전송 능력 지표(단위: 초/바이트)이다. 즉, AES-CCM으로 데이터를 암호화하는데 소요되는 처리 능력을 R_{comp}, 암호화된 데이터를 통신 선로로 전송하는데 소요되는 처리 능력을 R_{comm}으로 표시할 때, $R_{max} = \max(R_{comp}, R_{comm})$이 된다. 그러면, R_{max}에 전체 데이터 M을 m 등분한 만큼 곱하면 하나의 단계를 수행하는데 걸리는 시간을 얻을 수 있다. 따라서 전체 단계를 수행하는데 걸리는 시간은 총 단계 수인 m+1만큼을 곱해주어야 한다. 또한, 전체 수행시간에 동기화 오버헤드 T_{syn}를 m 번 더해주어야 한다. 최종적으로 식 (1)의 최소화 문제를 풀면 최적 m_{opt}는 다음과 같다.

$$m_{opt} = \sqrt{\frac{MR_{max}}{T_{syn}}} \quad (2)$$

4. 실험 결과

본 논문에서는 2.33GHz Intel Core2-Quad CPU의 실험환경에서 제안 방법의 효용성을 검증하였다. 먼저 코어간 동기화 오버헤드 T_{syn}를 확인하기 위해 Pthread barrier 명령어를 이용하여 측정한 결과, 2-코어 경우 약 10.1 μsec, 4-코어의 경우 약 49.2 μsec의 상당한 오버헤드가 있음을 확인하였다. 다음으로 다양한 데이터 크기 별로 AES-CCM 수행 시간을 측정한 결과, 그림 4와 같은 결과를 얻었다. 즉, R_{comp} = 0.037 μsec/Byte.

본 논문에서는 멀티코어 CPU에 다양한 유무선 네트워크 표준이 접속되는 것으로 가정하고, 각 유무선 표준의 최대 전송 능력의 일부가 실질적인 전송 능력이 됨을 감안하여, 전송 능력을 다음과 같은 범위로 설정하였다. 즉, 10Mbps, 100Mbps, 1000Mbps에 해당하도록 R_{comm}을 0.8 μsec/Byte, 0.08 μsec/Byte, 0.008 μsec/Byte로 설정하였다. 데이터 크기 M이 10MB와 10GB의 경우에 대하여, 식 (2)에 의한 최적 M_{opt}와 해당되는 speed-up(= 순차 수행 시간/계산-통신 중첩에 의한 수행 시간)을 정리하면 표 1과 같다. 여기서 순차 수행 시간은 MR_{comp} + MR_{comm}으로 계산하였다.

표 1의 결과를 다음과 같이 정리해 볼 수 있다 :

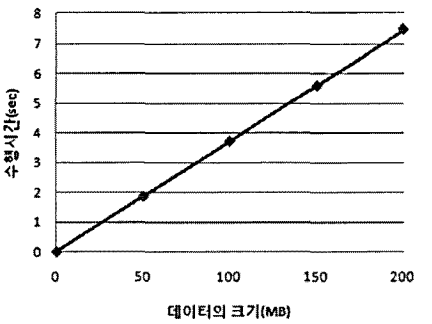


그림 4 싱글-코어를 이용한 AES-CCM 처리 시간

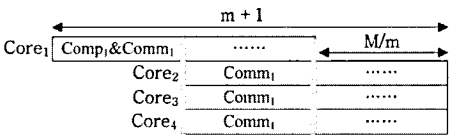
- 네트워크 성능이 증가할수록 최적 m_{opt}는 적어지고, 처리 또는 전송해야 할 데이터의 크기가 증가할수록 최적 m_{opt}는 증가한다.
- 특이한 점은 1000Mbps의 경우 이미 R_{comp} (0.037 μsec/B)가 R_{comm} (0.008μsec/B) 보다 커져서 R_{max}가 R_{comp}이 되며, 이러한 현상은 네트워크 성능이 더욱 증가하여도 적용된다.
- 100Mbps에서 가장 높은 speed-up을 달성할 수 있는데, 이는 10Mbps나 1000Mbps의 경우에 비하여 R_{comm}과 R_{comp}의 차이가 적기 때문이다.
- 만약 최적의 네트워크 환경(즉, R_{comm}이 R_{comp}와 같아져 각 단계 수행 시 계산 및 통신 과정의 대기 시간이 없고 완벽한 부하 분산을 얻는 환경)을 가정하면, 최대 speed-up = 1.98(즉, m = 10MB인 경우 m_{opt} = 192에서) 또는 2.00(즉, M = 10GB인 경우 m_{opt} = 6,082에서)를 기대할 수 있다.
- 주의할 점은 위에서 언급한 설명은 통신을 위한 버퍼 크기를 고려하지 않은 이상적인 경우이고, 실제 시스템에서는 통신용 버퍼 크기를 같이 고려해야 한다.

마지막으로, 본 논문에서는 설명의 편의상 2-코어 시스템을 가정하였으나, 더 많은 코어에서도 효과적으로 성능 개선을 기대할 수 있다. 즉, 계산 과정에서 더 많은 쓰레드-레벨 병렬성을 확보하기 위하여 2장에서 기술한 일반적인 병렬처리 기법을 추가로 적용하거나 복수의 네트워크 패스를 적용하면 응용 프로그램 수준에서 multiple-issue 파이프라인[16]과 같은 병렬처리가 가능하다. 또한, 이러한 추가 병렬성 확보는 앞서 언급한 R_{comm}과 R_{comp}의 균형을 맞추기 위한 용도로도 사용될 수 있다.

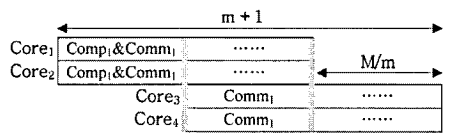
표 1 데이터 크기/네트워크 성능별 speed-up (2-코어)

	10Mbps (0.8 μs/B)	100Mbps (0.08 μs/B)	1000Mbps (0.008 μs/B)
M=10MB	1.04 (m_{opt} = 894)	1.45 (m_{opt} = 292)	1.20 (m_{opt} = 192)
M=10GB	1.04 (m_{opt} = 28,284)	1.46 (m_{opt} = 8,944)	1.21 (m_{opt} = 6,082)

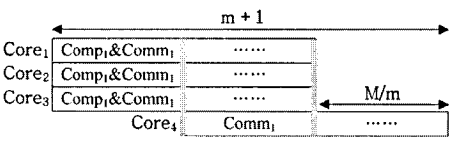
예를 들어 4-코어 시스템을 이용할 경우, $R_{comm} \gg R_{comp}$, $R_{comm} > R_{comp}$, $R_{comm} < R_{comp}$ 에 따라서 통신 과정 및 계산 과정에 할당되는 코어 수를 조정할 수 있다 (그림 5 참조). 즉, 계산 전용 프로세서와 통신 전용 보조 프로세서가 하나씩 장착된 분산메모리 플랫폼에서의 계산-통신 중첩화에 대한 기존 연구[9-12]와 달리, 주어진 임베디드 시스템 환경에서 계산 및 통신의 상대적인 부하에 따라 계산 및 통신 과정에 할당되는 코어 수를 적절히 조절하여 최적의 성능 개선을 기대할 수 있다.



(a) $R_{comm}(0.8\mu s/B) \gg R_{comp}(0.037\mu s/B)$



(b) $R_{comm}(0.08\mu s/B) > R_{comp}(0.037\mu s/B)$



(c) $R_{comm}(0.008\mu s/B) > R_{comp}(0.037\mu s/B)$

그림 5 4-코어 시스템의 계산-통신 중첩화 병렬처리

5. 결 론

본 논문에서는 멀티코어 임베디드 시스템 환경에서 현재 암호화/메시지 인증 표준으로 제안되고 있는 AES-CCM을 병렬화하여 대용량 데이터의 기밀성과 무결성을 효과적으로 보장하는 기법을 제안하였다. 제안한 기법은 AES-CCM의 데이터 종속성을 고려하였으며, 사용자 수준에서의 쓰레드-수준 병렬성을 이용하여 계산 과정과 통신 과정을 중첩함으로써 그들을 동시에 수행시켜 수행시간을 효과적으로 감소시킬 수 있었다. 특히, 파이프라인 단계 수를 증가시키면 비례적으로 성능이 개선되는 일반적인 문제와 달리, 동기화 오버헤드를 고려한 최적의 파이프라인 단계 수를 도출하는 방법론을 도출하였다.

2-코어 CPU를 이용한 실험을 통하여 제안 방법은 대용량 데이터의 기밀성 및 무결성을 동시에 보장하면서 수행시간이 기존의 순차적인 계산-통신 방법에 비하여 최대 50% 정도로 감소할 수 있음을 확인하였다. 특히, 제안 방법은 AES-CCM 병렬화를 위하여 잘 알려진 일반적인 병렬처리 기법(즉, Data-Level Parallelism, Task-Level Parallelism)을 적용[7,8]하는 것 외에 추가로 적용(1.4배까지 성능 개선이 가능) 될 수 있으며 CPU내 많은 코어 중 유휴 코어를 이용할 수 있어, 매우 효과적인 성능 개선 방법이라 할 수 있다.

참 고 문 헌

[1] D. Stinson, *Cryptography: Theory and Practice*, CRC Press, 2005.

[2] J. Black, "Authenticated Encryption," 2003.

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

[4] U. S. National Institute of Standards and Technology, "The Advanced Encryption Standard," *Federal Information Processing Standard(FIPS) 197*, 2002.

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

[6] A. Samiah, A. Aziz, and N. Ikram, "An Efficient Software Implementation of AES-CCM for IEEE 802.11i Wireless Standard," *Proc. of COMPSAC*, pp.689-694, 2007.

[7] E. Lee, S. Lee, S. Hong, H. Choi, W. Choi, Y. Chung, B. Min, "Parallel Processing of AES-CCM," *Proc. of the CISC 2009*, vol.19, no.1, pp.199-202, 2009. (in Korean)

[8] D. Bae, J. Kim, S. Park, and O. Song, "Design and Implementation of IEEE 802.11i Architecture for Next Generation WLAN," *Proc. of CISC 2005, LNCS 3822*, pp.346-357, 2005.

[9] A. Sohn, et al., "Identifying the Capability of Overlapping Computation with Communication," *Proc. of PACT*, p.133, 1996.

[10] K. Ishizaki, H. Komatsu, and T. Nakatani, "A Loop Transformation Algorithm for Communication Overlapping," *Intl. J. of Parallel Programming*, vol.28, no.2, pp.135-154, 2000.

[11] A. Danalis, et al., "Transformations to Parallel Codes for Communication-Computation Overlap," *Proc. of SC*, p.58, 2005.

[12] T. Hoefler and A. Lumsdaine, "Optimizing Non-Blocking Collective Operations for Infiniband," *Proc. of CAC*, 2008.

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

[14] C. Tseng, "Compiler Optimization for Eliminating Barrier Synchronization," *Proc. of PPoPP*, 1995.

[15] J. Lipman and Q. Stout, "A Performance Analysis of Local Synchronization," *Proc. of SPAA*, 2006.

[16] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach - Fourth Edition, Elsevier, 2007.