韓國컴퓨터情報學會 論文誌
第17卷 第1號, 2012. 1.

2012-17-1-3-5

멀티프로세서 기반의 병렬 AES 암호 알고리즘에 관한 연구

박 중 오*, 오 기 욱**

A Study on Parallel AES Cipher Algorithm based on Multi Processor

Jung-Oh Park*, Gi-oug, Oh**

요 약

본 논문은 대칭키 기반의 암호 알고리즘으로 사용하는 AES 암호 알고리즘을 정의하고, 멀티코어 프로세서의 자원을 최대 활용하기 위한 병렬 암호 알고리즘 설계를 제안한다. 제안한 병렬 암호 알고리즘은 코어의 개수에 따라 암호 알고리즘을 쓰레드별로 할당하여 암호 연산의 병렬 수행을 확인 하였고, AES 암호 알고리즘에 비해 약 30% 성능향상을 확인 하였다. 암호 알고리즘의 암·복호화 성능은 바이너리 비교 분석 툴을 통해 확인하여, AES 암호 알고리즘과 제안한 병렬 암호 알고리즘의 바이너리는 동일 결과를 확인 하였으며, 복호화한 바이너리 또한 동일하였다. 본 논문에서 제안한 멀티코어 프로세서 환경의 병렬 암호 알고리즘은 개인 PC, 노트북, 서버, 모바일 환경에서 금융 서비스의 인증 및 결제에 적용 가능하고, 대형 데이터의 고속 암호화 연산이 필요한 분야에서 활용 가능하다.

▸ Keyword : 멀티코어프로세서, AES, 병렬화, non-Feistel 구조

Abstract

This paper defines the AES password algorithm used as a symmetric-key-based password algorithm, and proposes the design of parallel password algorithm to utilize the resources of multi-core processor as much as possible. The proposed parallel password algorithm was confirmed for parallel execution of password computation by allocating the password algorithm according to the number of cores, and about 30% of performance increase compared to AES password algorithm. The encryption/decryption performance of the password algorithm was confirmed through binary

• 제1저자 : 박중오 • 교신저자 : 오기욱
• 투고일 : 2011. 10. 05, 심사일 : 2011. 11. 07, 게재확정일 : 2012. 01. 14
* 성결대학교 정보산업기술연구소(Sungkyul University)
* 안양대학교 교양학부(Division of Liberal Arts, Anyang University)

comparative analysis tool, which confirmed that the binary results were the same for AES password algorithm and proposed parallel password algorithm, and the decrypted binary were also the same. The parallel password algorithm for multi-core environment proposed in this paper can be applied to authentication/payment of financial service in PC, laptop, server, and mobile environment, and can be utilized in the area that required high-speed encryption operation of large-sized data.

▸ Keyword : Multi-core Processors, AES, Parallel, non-Feistel

I. 서 론

오늘날 싱글코어 프로세서는 처리 속도 측면이나 복잡도에 있어서 물리적 한계에 도달하고 있고, 이에 따라 멀티코어 프로세서와 관련된 산업이 점차 성장하고 있다. 개인, 기업 등에서 사용하는 PC, 서버 및 모바일 기기는 듀얼코어나 쿼드코어 기반의 제품이 주목 받고 있다.

멀티코어 프로세서는 기존의 싱글코어 프로세서 보다 성능 측면에 있어 효율성이 높은 다중 처리와 부하 균등, 소비 전력 절감의 이점이 있으며, 멀티코어 프로세서를 활용하여 집약형인 작업 처리의 대부분을 병렬 방식으로 처리가 가능하여 시스템의 효율성과 응용 프로그램의 성능향상이 가능하다.

하지만 싱글코어 프로세서에서 대용량의 데이터 암호 연산에서 1개의 프로세서에만 집중하기 때문에 프로세서의 효율성이 낮다.

또한 데이터 전송이 가능한 매체의 종류는 증가하고 있고, 전송하는 데이터의 크기와 종류도 다양하게 변화 하고 있으며, 이를 안전하게 보호하는 암·복호화 알고리즘 연산 또한 멀티코어 프로세서에 적합한 병렬 처리가 필요하다.

따라서 본 논문에서는 멀티코어 프로세서의 자원을 활용하기 위한 AES 암호 알고리즘을 각 멀티쓰레드에 할당하여 연산하는 병렬화를 제안하고, 모바일 기기와 같은 저성능 멀티코어 프로세서에서 암호 알고리즘의 효율성을 증대하며, 암호 연산 수행 시간을 최소화한다.

II. 관련 연구

2.1 AES(Advanced Encryption Standard) 암호 알고리즘

1997년 NIST(National Institute of Standard and Technology)는 Advanced Encryption Standard 암호 알고리즘을 공모하여, 벨기에 학자 Joan Daemen과 Rijnmen이 설계한 블록 암호 Rijndael을 AES 암호 알고리즘으로 선정하게 되며, AES 암호 알고리즘은 대칭키 기반의 128, 192, 256 비트 블록 암호 알고리즘으로 non-Feistel 구조에 속한다[1][3].

키 확장 알고리즘으로부터 생성되는 라운드 키 크기는 평문(Plaintext)과 암호문(Ciphertext)의 크기와 동일한 128비트로 출력하고, SubBytes, ShiftRows, MixColumns, AddRoundKey 단계의 함수 연산을 수행하여 암호화 한다.

2.1.1 SubBytes

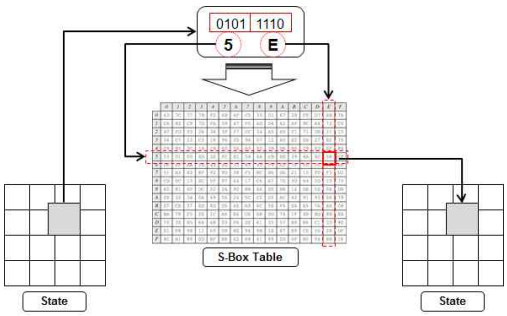

그림 1. SubBytes 동작 과정
Figure 1. SubBytes Operation Process

SubBytes 함수는 AES 암호 알고리즘의 암호화 과정에서 사용하는 대치함수를 말한다[3]. 하나의 바이트를 대체하기 위해 각 바이트를 4비트씩 2개의 16진수로 계산하여 왼쪽 4비트를 S-Box의 행으로 오른쪽 4비트 열로 S-Box 테이블에서 읽고, 두 16진수의 행과 열이 교차하는 부분의 바이트 값을 출력한다[2].

AES는 128비트의 스테이트를 한 바이트의 성분을 4×4 행렬로 나타내고, 바이트에 대한 독립 SubBytes 함수를 수행하며, 행렬에서의 성분들의 위치는 변하지 않는다.

2.1.2 ShiftRows

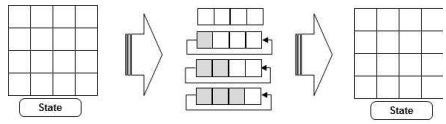

그림 2. ShiftRows의 변환
Figure 2. ShiftRows Conversion

ShiftRows는 행렬의 행을 왼쪽으로 이동하는 수행을 한다. 이동하는 수는 스테이트 행렬의 행 번호 0에서 3에 의존하며, 행 번호 0은 이동을 하지 않는다. 1번째 행으로 시작하여 왼쪽으로 1바이트만큼 순환을 하고, 2번째 행에서는 2바이트만큼 순환하며, 마지막 3번째 행에서는 3바이트만큼 이동 순환을 한다.

2.1.3 MixColumns

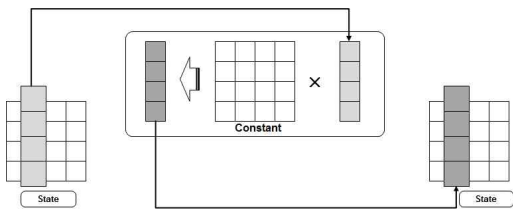

그림 3. MixColumns의 변환
Figure 3. MixColumns Conversion

MixColumns는 인접한 바이트들 안에 있는 비트를 기반으로 비트 값을 변환하는 함수이다. 즉, 바이트 내부 변환을 수행하는 바이트 섞음을 수행한다. 행렬에서 열 단위 연산을 순차적으로 수행하고, 각 스테이트의 열을 새로운 열로 변환을 한다.

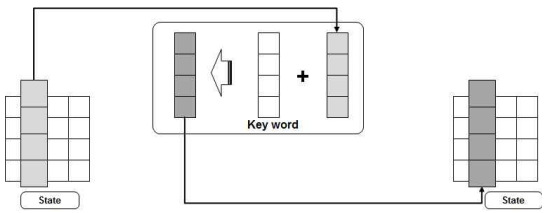

그림 4. AddRoundKey의 변환
Figure 4. AddRoundKey Conversion

AddRoundKey는 한 번에 행렬의 한 열씩 XOR 연산 수행을 하는데, MixColumns와 유사한 연산을 수행하지만, MixColumns는 각 스테이트에 열행렬에 상수 정방행렬의 곱셈을 연산하고, AddRoundKey는 각 스테이트 열행렬에 라운드 Key Word를 XOR 연산을 한다. XOR의 연산 은 덧셈과 뺄셈 연산이 동일하므로 AddRoundKey 변환은 자신의 역변환을 나타낸다.

AES 암호 알고리즘의 4단계의 함수 연산과정을 포함한 암호화와 복호화 과정을 수행한다. AES 암호 알고리즘은 대칭키 기반으로 각 함수마다 출력되는 값이 서로 교환되어 암호문을 생성하지 않는 non-Feistel 암호 구조이고, AES의 암·복호화는 각 라운드의 각 함수 연산에서 자신의 역변환 성질을 이용하여 수행한다. 또한 각 함수 연산은 서로 상쇄되는 방법으로 수행하고, 라운드 키는 역순으로 사용한다[1][2].

2.2 멀티코어 프로세서 활용 기술

멀티코어 프로세서 기술이 등장하게 된 배경에는 프로세서 속도를 기하급수적으로 증가시켜 성능을 높이는 방식이 한계에 도달하였다. Intel 사의 무어가 발표한 법칙에 따라 매년 프로세서의 성능이 기하급수적으로 성능이 빨라짐으로써 소프트웨어 성능 또한 향상되고, 이에 따라 시장규모도 기하급수적으로 성장하고 있다.

하지만 CPU의 트랜지스터 직접도를 더 이상 증가시킬 수 없는 물리적인 한계와 클록 스피드 증가에 따라 발생하는 문제가 있다. 이러한 문제를 해결하기 위한 멀티코어 프로세서 기술은 클록 스피드는 낮지만, 코어의 수를 늘려 병렬로 처리하는 방식으로 전력대비 효율성을 증가시키는 방향으로 발전하고 있다.

싱글코어 프로세서의 경우 멀티태스킹을 수행하면 CPU의 모든 용량을 활용하고, 태스크 처리 가능할 때 까지 대기하는 비효율적인 처리를 한다. 그러나 멀티코어 프로세서는 각 코어에 고유한 캐시가 장착되어 충분한 리소스의 확보가 가능하고, 연산 집약형인 작업의 대부분을 병렬 방식으로 처리가 가능하기 때문에 효율성과 응용프로그램의 성능향상이 가능하다[4].

멀티코어 프로세서의 클록과 코어 개수를 증가하여 프로세서의 성능은 발전하고 있지만, '암달의 법칙(Amdahl's Law)'에 의해 병렬화율을 높이지 않을 경우, 성능향상이 높지 않음을 나타낸다[5][6]. 이는 4개의 코어를 사용하는 쿼드코어 프로세서에서 처리하는 응용 프로그램의 병렬처리가 가능한 부분과 병렬처리가 불가능한 순차적인 부분에 의해 병렬화 처리에 대한 성능이 향상 되지 않는 한계가 존재하는 법칙이다. 처리해야 할 태스크의 20%를 병렬 처리로 구현할 때, 성능향상은 대략 1.176배의 성능향상을 나타낸다. 또한

같은 20% 병렬화 조건에서 코어의 수를 무한대(∞)로 증가하였을 경우, 최대 1.25배의 성능향상을 나타낸다[7].

2.3 컴파일러 기술과 멀티코어 기술

1997년에 산업표준으로 채택된 공유메모리 병렬프로그래밍 모델인 OpenMP는 표준 C/C++와 Fortran 77/90을 확장하는 병렬화 디렉티브(Directives)와 라이브러리들의 집합, SPMD(Single Program Multiple Data)의 수행 환경에 대해 명시를 하고 있다. OpenMP 병렬성의 특징은 크게 내포 병렬성과 록킹(Locking)을 통한 임계구역(Critical Section)의 동기화 구조를 가지고 있으며, 'Coarse-Grained' 병렬성의 구현을 위한 'Orphan' 디렉티브 개념을 지원하고, 병렬 프로그래밍의 확장성을 높였다[8][9][10].

OpenMP에서 제공하는 디렉티브는 새로운 쓰레드 그룹을 생성하는 병렬화 디렉티브가 있고, 병행하는 쓰레드 간에 작업을 나누어 수행하는 작업공유 디렉티브가 있으며, 병렬 영역 외부에 정의되어 변수 공유를 제어하는 데이터 환경 디렉티브가 있으며, 병렬처리를 수행하는 쓰레드간의 수행 순서를 제어하는 동기화 디렉티브 등이 있다[13].

OpenMP는 Master 쓰레드로 수행을 시작하고, 생성/합류 병렬화 모델을 사용하는데, Multi 쓰레드에 의해 병행 수행되는 병렬 영역은 OpenMP의 기본적인 병렬화 디렉티브인 'Parallel'에 정의 하고 있으며, Master 쓰레드가 병렬 영역과 경합되면, Multi 쓰레드를 갖는 하나의 그룹을 생성하여 병렬화를 수행한다[11].

III. AES 암호 알고리즘의 병렬화

전자산업과 정보통신 기술의 발전에 대응하여 효율적으로 작업 처리가 가능한 멀티코어 프로세서 환경의 응용 프로그램이 필요하다. 데이터를 안전하게 전송하기 위한 암호 알고리즘 또한 싱글코어 프로세서 환경에서 멀티코어 프로세서 환경에 적합한 암호 알고리즘이 필요하다. 대칭키 기반의 AES 암호 알고리즘은 멀티코어 프로세서 기술이 발전하기 이전에 공모, 채택 및 제정되었으므로 멀티코어 프로세서 환경에 적합한 암호 알고리즘이 필요하다.

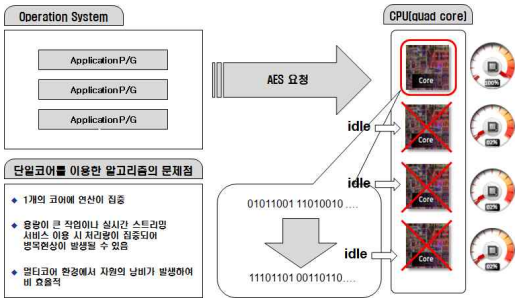

그림 5. 멀티코어 환경에서 단일연산 AES 암호 알고리즘의 비효율성
Figure 5.. Inefficiency of single-operation AES password algorithm in multi-core environment

그림 5는 싱글코어 프로세서에서 AES 암호 알고리즘을 멀티코어 프로세서 환경에서 수행하였을 때 나타나는 비효율적 문제점을 나타낸다. 멀티코어 프로세서 기반에서 AES 암호 알고리즘을 수행하였을 때 남은 코어들은 운영체제에 의해 다른 작업을 수행하거나 그렇지 않을 경우에는 Idle 상태로 자원을 활용하지 않는 상태로 대기한다. 이러한 멀티코어 환경에서 낭비되는 프로세서의 코어를 최대한 활용하기 위한 멀티코어 기반의 암호 알고리즘이나 응용 프로그램이 필요하다.

따라서 본 논문은 기존 싱글코어 기반의 AES 암호 알고리즘을 멀티코어 프로세서 환경에 적합한 알고리즘으로 재해석하고, 암호화 및 복호화 수행에 멀티코어를 활용하여 효율성을 높이는 암호 알고리즘을 제안한다.

3.1 SubBytes의 병렬 알고리즘

그림 1과 같은 기존 AES 암호 알고리즘의 'SubBytes' 함수 부분을 병렬연산을 하기위한 병렬 SubBytes 함수를 설계한다.

4byte 씩 4개의 행으로 이루어진 1개의 스테이트를 전체적으로 진행하는 과정에서 행별로 각각의 쓰레드가 작업을 병렬적으로 동시 수행하여 4×4의 스테이트를 연산한다.

즉, 각각 쓰레드는 4개의 행 블록(32bit)을 한번에 각각의 쓰레드로 할당되어 연산을 수행하고, 각 행들은 1byte의 블록으로 나눈 다음 각 블록에 해당하는 연산을 S-Box를 참고하여 연산한다.

표 1은 OpenMP의 문법으로 병렬 SubBytes 함수를 나타낸다.

표 1. 제안하는 병렬 SubBytes 알고리즘
Table 1. Proposed Parallel SubBytes Algorithm

```
#pragma omp parallel
{
    #pragma omp for private(i, j)
    {
        for(i=0; i<4; i++)
        {
                for(j=0; j<4; j++)
                {
                state[i][j] = getSBoxValue(state[i][j]);
                }
        }
    }
}
```

3.2 ShiftRows의 병렬 알고리즘

병렬 SubBytes 함수 연산을 통해 나온 행렬 스테이트는 다시 병렬 ShiftRows 함수로 연산을 수행하는데, 그림 2와 같이 기존 AES 암호 알고리즘은 각 행을 순차적으로 연산하는 과정이었다. 하지만, 기존 AES의 ShiftRows 함수 또한 병렬 연산 이 가능한 함수이므로, 본 절에서는 병렬 ShiftRows 함수를 설계한다.

ShiftRows의 함수를 병렬 연산을 수행하기 위한 알고리즘은 표 2와 같으며, 제안한 병렬 SubBytes 함수 연산을 통해 나온 스테이트는 각 행 번호 0에서 3까지의 행들은 각각의 Thread에 할당하여 이동 순환을 한다. 각 코어에 할당된 Thread에서, 0행은 바이트 이동 순환을 행하지 않으므로 제안하는 병렬 ShiftRows 함수 또한 바이트 이동 연산을 하지 않는다. 즉, 나머지 각 코어에 할당한 1행과 2행 행 번호 3은 각각 왼쪽으로 1바이트, 2바이트, 3바이트 씩 이동 연산을 수행한다.

표 2. 제안하는 병렬 ShiftRows 알고리즘
Table 2. Proposed Parallel ShiftRows Algorithm

```
#pragma omp parallel shared(temp)
{
    #pragma omp sections
    {
                #pragma omp section
                {
                                temp=state[1][0];
                                state[1][0]=state[1][1];
                                state[1][1]=state[1][2];
                                state[1][2]=state[1][3];
                                state[1][3]=temp;
                                temp=state[2][0];
                                state[2][0]=state[2][2];
                                state[2][2]=temp;
                }
                #pragma omp section
                {
                                temp=state[2][1];
                                state[2][1]=state[2][3];
                                state[2][3]=temp;
                                temp=state[3][0];
                                state[3][0]=state[3][3];
                                state[3][3]=state[3][2];
                                state[3][2]=state[3][1];
                                state[3][1]=temp;
                }
    }
}
```

3.3 MixColumns의 병렬 알고리즘

표 3과 같이 제안하는 병렬 MixColumns 함수는 멀티쓰레드 생성을 요청하며 시스템의 자원을 Fork로 생성된 쓰레드들에게 할당한다.

평문이 기존 AES 암호 알고리즘과 동일한 AddRoundKey 함수와 병렬 SubBytes 함수를 거쳐, 병렬 MixColumns 함수의 연산은 병렬로 수행한다. 4×4 행렬 스테이트에서 각 열들은 멀티코어 프로세서의 개수에 맞게 치환 연산을 수행하고, 멀티코어 프로세서의 개수에 맞는 Thread에 할당하여 정해진 상수행렬(Constant Matrix)과 행렬 곱셈 연산을 수행하게 된다. 각 코어에서 함수 연산의 수행을 통해 나온 각 행들은 4×4 행렬 스테이트로 다시 조합한다.

표 3. 제안하는 병렬 MixColumns 알고리즘
Table 2. Proposed Parallel MixColumns Algorithm

```
#pragma omp parallel
{
unsigned char Tmp,Tm,t;
#pragma omp for private(i) shared(Tmp,Tm,t) reduction()
    {
    for(i=0;i<4;i++)
                {
                t=state[0][i];
                Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^
                        state[3][i];
                Tm = state[0][i] ^ state[1][i];
            Tm = xtime(Tm);
            state[0][i] ^= Tm ^ Tmp;
                Tm = state[1][i] ^ state[2][i];
            Tm = xtime(Tm);
            state[1][i] ^= Tm ^ Tmp;
                Tm = state[2][i] ^ state[3][i];
            Tm = xtime(Tm);
            state[2][i] ^= Tm ^ Tmp;
                Tm = state[3][i] ^ t;
            Tm = xtime(Tm);
            state[3][i] ^= Tm ^ Tmp;
                }
    }
}
```

본 논문에서는 기존의 AES 암호 알고리즘과 동일한 4×4 행렬 스테이트를 각 함수마다 행 단위로 분할하여 각각의 코어에서 연산을 수행하는 SubBytes, ShiftRows, MixColumns 함수에 대한 병렬화를 제안하였다.

Ⅳ. 성능 분석 및 평가

본 논문에서 제안한 멀티코어 프로세서 기반의 병렬 AES 암호 알고리즘에 대한 성능을 평가한다. 성능 평가 및 분석을 위한 시스템 환경은 Intel Core2 CPU T5500 1.66GHz, 1GB RAM PC에서 Windows XP SP3 환경으로 구성하여 평가하였다.

평가 항목은 파일종류에 따른 기본적인 연산의 정상 수행을 확인하기 위한 암·복호화 결과 평가를 정성적 나타냈으며, 기존의 AES 암호 알고리즘과 본 논문에서 제안한 병렬 AES 암호 알고리즘을 시간 복잡도를 이용하여 효율성을 분석 하였으며, 프로파일링 툴을 통해 암호 알고리즘에 대한 성능을 분석하였다.

4.1 병렬 AES 암호 알고리즘 수행 평가

기존의 AES 암호 알고리즘과 본 논문에서 제안한 병렬 암호 알고리즘을 Text, MP3, MP4, Docx, Hwp의 파일 포맷 종류로 암호화하여, 이를 바이너리 분석 툴로 비교하여 암호화한 결과를 아래 표 4와 같이 정성적으로 비교 하였다.

표 4. 암·복호화 연산 수행에 대한 바이너리 비교 분석
Table 4. Binary Comparative Analysis for Encryption·Decryption Operation Execution

포맷	기존 AES 암호 알고리즘		제안 암호 알고리즘	
	암호화	복호화	암호화	복호화
Text	◎	○	◎	○
MP3	◎	○	◎	○
MP4	◎	○	◎	○
Docx	◎	○	◎	○
Hwp	◎	○	◎	○

◎ : 암호화에 대한 바이너리
○: 복호화에 대한 바이너리

기존 AES 암호 알고리즘과 본 논문에서 제안한 알고리즘으로 암호화 및 복호화를 수행하였을 때 바이너리 값은 동일하였으며, 파일 포맷별로 암호화를 수행한 파일들은 해당 응용프로그램에서 확인이 불가능 하였다. 이는 암호화로 인한 결과로 나타낼수 있다. 또한 복호화를 수행한 파일들의 바이너리 값도 동일하였고, 암호화한 파일과는 달리 해당 응용프로그램에서 정상적인 확인이 가능하였다. 제안한 알고리즘을 통해 생성된 암호문과 복호화한 평문은 기존의 AES 암호 알고리즘과 동일한 바이너리를 갖는 정상 수행을 확인하였다.

4.2 시간 복잡도를 이용한 효율성 분석

제안한 암호 알고리즘과 기존의 암호 알고리즘에서 4가지 함수에 대한 복잡도를 비교하여 평가하였다. AES 암호 알고리즘의 SubBytes, ShiftRows, MixColumns에 대한 명령어 실행 횟수를 분석하여 복잡도를 산출 하였으며, 복잡도는 시간복잡도를 이용하여 비교 및 평가하였다.

4.2.1 SubBytes 함수의 시간복잡도를 이용한 효율성 분석

기존 암호 알고리즘의 SubBytes 함수를 C 코드로 구현하였고, C코드에서 각각 구문별 소요되는 명령어를 분석하여 카운트하였다. 함수 내부에서 두 개의 루프가 사용되었으며, 최소 n×n 번 수행한다. for 구문 시작부에서는 n+1개의 명령어가 사용되며 중첩된 for 구문에서도 n+1 개의 명령어가

수행된다. 따라서 이 함수가 순차적으로 실행된다면 총 실행 횟수는 4n2+2n+1 번이 수행된다.

T(n)=4n2+2n+1 -> O(n2) ····· (식4 - 1)

제안한 알고리즘의 PA_SubBytes() 함수를 C 코드로 구현하였고, C코드에서 각각 구문별 소요되는 명령어를 분석하여 카운트하였다. 함수 내부에서 기존 암호 알고리즘과 마찬가지로 두 개의 루프를 사용한다. 단, 멀티코어 프로세서에게 병렬 연산을 명령하기 위하여 “#pragma omp parallel” 와 “#pragma omp for private(i, j)”를 사용하였다. 이 명령어는 각 구문 당 1개의 명령어를 수행함으로 명령어 실행 횟수는 각 한 번씩이다. 마찬가지로 for 구문 시작부에서는 n+1 개의 명령어가 사용되며 중첩된 for 구문에서도 n+1 개의 명령어가 수행된다. 따라서 이 함수가 병렬명령을 포함하기 때문에 병렬 계산된다.

T(n)=2n2+n+3 -> O(n2) ····· (식4 - 2)

Core 1과 Core 2는 동시에 실행되기 때문에 가장 늦은 시간인 Core 1이 실제적인 응답시간이 된다. 식 3과 같이 시간 복잡도 함수는 T(n)=2n2+n+3이 된다.

표 5. SubBytes의 시간복잡도 비교
Table 5. Comparison of Time Complexity SubBytes

SubBytes()(AES)	시간복잡도 (시간소비)	PA_SubBytes() (제안 알고리즘)
T(n)=4n2+2n+1	>	T(n)=2n2+n+3

따라서 기존 암호 알고리즘은 T(n)=4n2+2n+1, 제안 알고리즘은 T(n)=2n2+n+3 의 시간복잡도를 갖는다. 세부적으로 4n2와 2n2는 2n+1 과 n+3 에 비하여 전체 해에 큰 영향을 미치게 된다. 따라서 n의 값이 커질수록 수행시간의 격차가 커짐을 확인하였다.

표 6. SubBytes에 대한 코어 개수 변화에 따른 시간복잡도
Table 6. Time Complexity according to Change in Number of Cores for SubBytes

프로세서 코어 수	시간복잡도
Single Core	4n2+2n+1
Dual Core	2n2+n+3
Quad Core	n2+n/2+3

또한 표 6과 같이 프로세서의 코어 개수를 변화하였을 때 예상되는 시간복잡도를 나타냈다. Quad Core에서 제안 알고리즘이 수행되었을 때의 시간복잡도는 T(n)=n2+n/2+3 이 된다.

4.2.2 ShiftRows 함수의 시간복잡도를 이용한 효율성 분석

C코드에서 각각 구문별 소요되는 명령어를 분석하여 카운트하였고, 함수 내부에서는 특정 루프가 사용되지 않았고, 단순한 배정연산이 여러번 사용되었다. 따라서 기존 암호 알고리즘에서 각 배정연산이 사용된 횟수는 총 16번이다.

T(n)=16 ····· (식4 - 3)

위와 같이 시간복잡도 함수 T(n)=16으로 항상 상수형태로 n이 증가하여도 고정 길이의 시간을 소비하며, 함수 내부에서 기존 암호 알고리즘과 마찬가지로 특정 루프가 사용되지 않았고, 단순한 배정연산이 여러 번 사용되었지만, 멀티코어 프로세스로 병렬 연산을 명령하기 위하여 “#pragma omp parallel shared(temp)”, “#pragma omp sections”, “#pragma omp section”를 사용하였다. “#pragma omp parallel shared(temp)” 구문에서는 각 section 별로 공통으로 사용되는 temp를 2번 생성하므로 3개의 명령어를 사용한다. 따라서 병렬 명령을 포함하기 때문에 병렬 계산을 하고, 코어별로 총 실행 횟수는 Core 1은 13회, Core 2는 8회를 수행한다.

T(n)=13 ····· (식4 - 4)

표 7. ShiftRows의 실행명령어 개수와 코어 개수 변화에 따른 시간복잡도
Table 7. Time Complexity according to Change in Number of Execution Commands and Number of Cores for ShiftRows

ShiftRows()(AES)	시간복잡도 (시간소비)	PA_ShiftRows()(제안 알고리즘)
T(n) = 16	>	T(n) = 13

상수 값의 시간복잡도로, 세부적으로 16과 13의 차이약 16:13의 시간비를 갖는다. 이함수에서는 고정길이의 시간을 소비하므로 n값의 변화에 따라 영향을 받지 않는다.

4.2.3 MixColumns 함수의 시간복잡도를 이용한 효율성 분석

기존 암호 알고리즘의 함수 내부에서는 한 개의 루프가 사용되었으며, 최소 n번 수행함을 확인하였고, for구문 시작부에서는 n+1개의 명령어를 사용한다. 기존 암호 알고리즘의 함수가 순차적으로 실행하는 총 실행 횟수는 23n+1 번이다.

T(n)=23n+1->O(n) ····· (식4- 5)

제안하는 암호 알고리즘의 함수 내부에서 동일한 한 개의 루프를 사용하였고, 최소 n번 수행한다. for구문 시작부에서는 n+1개의 명령어가 사용된다. 단, 멀티코어에게 병렬 연산을 명령하기 위하여 “#pragma omp parallel” 와 “pragma omp for private(i) shared(Tmp, Tm, t)”를 사용하였다. “#pragma omp parallel” 구문은 1개의 명령어를 수행하고, “pragma omp for private(i) shared(Tmp, Tm, t)” 구문은 각 인덱스로 사용하는 i와 Tmp, Tm, t를 각 1, 2, 2, 2번씩 생성한다. PA_MixColumns함수는 병렬명령을 포함하기 때문에 병렬 계산을 수행하며, 코어별로 총 실행 횟수는 Core 1는 10n+n/2+10회, Core 2는 10n+n/2+1회 실행한다.

T(n) = 10n + n/2 + 10->O(n) ····· (식4- 6)

Core 1과 Core 2는 동시에 실행되기 때문에 가장 늦은 시간인 Core 1이 실제적인 응답시간이 된다.

표 8. MixColumns의 실행명령어 개수와 코어 개수 변화에 따른 시간복잡도
Table 8. Time Complexity according to Change in Number of Execution Commands and Number of Cores for MixColumns

MixColumns ()(AES)	시간복잡도 (시간소비)	PA_MixColumns() (제안 알고리즘)
T(n) = 23n + 1	>	T(n) = 10n + n/2 + 10

기존 암호 알고리즘은 T(n)=23n+1, 제안 알고리즘은 T(n)=10n+n/2+10 의 시간복잡도를 갖는다. 이 시간복잡도 함수에서 해의 큰 영향을 미치는 항은 23n과 10n으로 n 값의 변화에 따라 두 함수 결과에 대한 격차가 커짐을 확인하였다.

표 9. MixCoulmns에 대한 코어 개수 변화에 따른 시간복잡도
Table 9. Time Complexity according to Change in Number of Cores for MixCoulmns

프로세서 코어 수	시간 복잡도
Single Core	23n+1
Dual Core	10n+n/2+10
Quad Core	5n+n/4+10

표 9는 프로세서의 코어 개수를 변화하였을 때 예상되는 시간복잡도를 나타낸다. Quad Core에서 제안하는 암호 알고리즘이 수행되었을 때의 시간 복잡도는 T(n)=5n+n/4+10이 된다.

이는 소비되는 시간이 현저히 감소하고 있음을 나타내고, 계산된 결과 값들은 시간복잡도에 의해 계산된 형태로, 실제 플랫폼에서 제안한 병렬화된 알고리즘이 수행될 경우 잦은 Context Switching으로 오버헤드가 발생하기 때문에 그래프와 다를 수 있다. 이는 암달의 법칙으로 설명할 수 있는데, 이는 성능이 비례적 선형으로 향상되지 않음을 나타낸다.

4.3 프로파일링 툴을 통한 성능 분석

알고리즘 성능 분석을 위하여 IPS(Intel Parallel Studio) 프로파일링 도구를 사용하였다. 프로파일링 도구를 통하여 각 함수 별 수행시간 측정, 쓰레드의 정상적 생성 검사, 전체 수행시간 측정, 효율성 측정을 하였다.

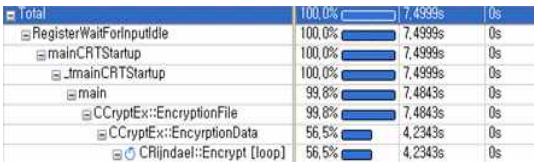

그림 6. 기존 AES 암호 알고리즘의 각 Function 별 CPU Time
Figure 6. CPU Time per Each Function for Existing AES Algorithm

기존 암호 알고리즘의 성능을 IPS도구를 통하여 분석하였다. 그림 6과 같이 전체 수행시간을 7.4999초로 이는 100MB의 텍스트 파일을 암호화 시간이다. 하위 Function들의 시간들 중 ‘Encrypt’ Function은 4.2343초를 차지하고 나머지 시간들은 파일 오픈 등과 같은 입출력 동작과 메모리 할당 및 회수 등의 연산에 의해 소비되는 시간을 나타낸다.

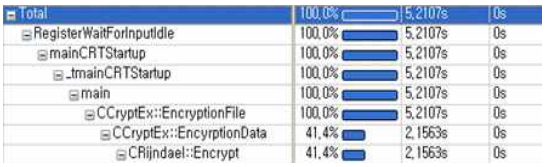

그림 7. 제안한 병렬 암호 알고리즘의 함수별 CPU Time
Figure 7. CPU Time per Each Function for Proposed Parallel Password Algorithm

그림 7은 제안한 병렬 암호 알고리즘을 수행하면서 IPS 도구로 분석하였다. 분석결과 100MB 텍스트 파일을 암호화하기 위해 전체시간이 5.0127초로 기존 암호 알고리즘보다 2.4872초(7.4999초-5.0127초) 빨랐다. 약 33%의 속도 성능향상을 나타냈으며, 듀얼코어 프로세서 기반의 하드웨어에서 병렬처리는 50%의 성능향상을 기대 할 수 없다. 이는 병렬화를 통한 문맥교환에 대한 오버헤드가 증가하고 빈번한 병렬처리 fork과 join으로 인한 오버헤드도 크기 때문이다. 약

33%의 단축된 속도는 상당히 향상된 수치이고, 복호화 연산도 동일한 수치를 나타냈다.

4.4 암호 알고리즘에 대한 CPU의 효율성 측정

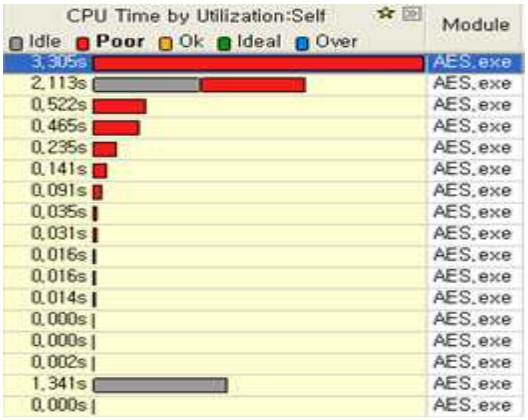

그림 8. 기존 AES 암호 알고리즘의 CPU 효율성
Figure 8. Existing AES algorithm CPU efficiency

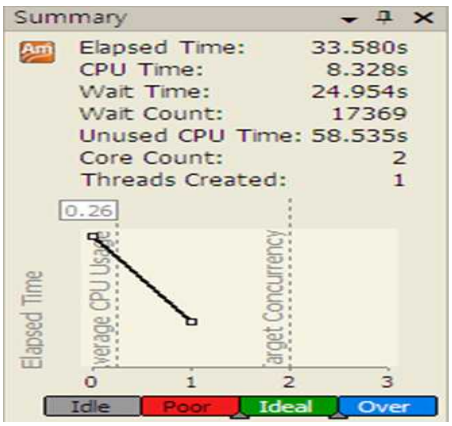

그림 9. 기존 AES 암호 알고리즘의 CPU 효율성 요약
Figure 9. summary of Existing AES algorithm CPU efficiency

그림 8과 그림 9는 기존 AES 암호 알고리즘의 멀티코어 활용 효율성과 생성된 멀티쓰레드의 개수를 측정하고 해당 알고리즘의 펑션별 수행시간과 전체적 효율성을 점검하였다. 그림 8에서 중앙에 표시된 막대그래프는 해당 알고리즘의 효율성을 나타낸 것으로 수행단계가 순차적으로 실행되며 단일 프로세서(단일 쓰레드)에서 수행되고 있음을 나타낸다.

그림 9의 그래프는 현재 멀티 코어의 동시작업 여부를 나타내고 있다. 'Thread Created'가 1인 의미는 쓰레드의 개수가 단일 쓰레드가 생성되었으며 비효율적인 단일 쓰레드로, 순차적으로 연산을 진행하였다는 결과를 나타낸다.

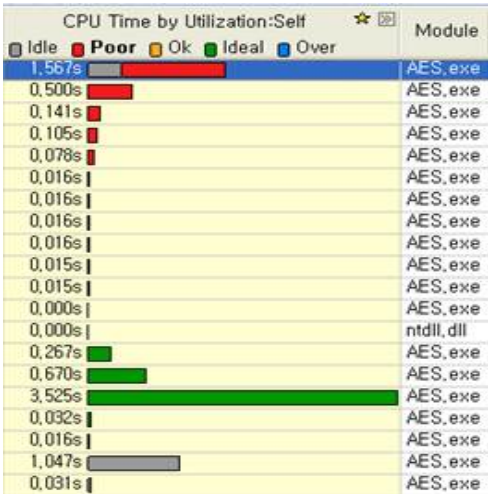

그림 10. 제안한 암호 알고리즘의 CPU 효율성
Figure 10. proposed AES algorithm CPU efficiency

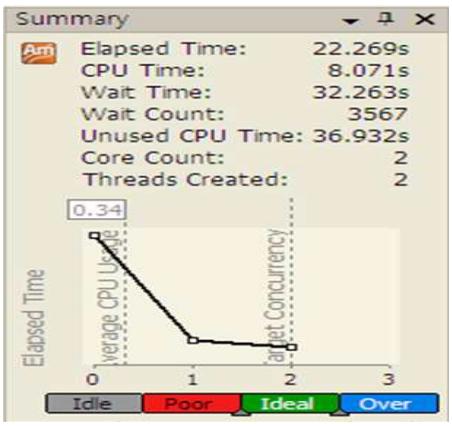

그림 11. 제안한 암호 알고리즘의 CPU 효율성 요약
Figure 11. Summary of proposed AES algorithm CPU efficiency

그림 10과 그림 11는 제안한 병렬 암호화 연산에 대한 효율성을 측정하였다. 그림 11의 막대그래프 구간이 그림 8보다 줄어들었으며, 'EncryptBlock' 부분은 이상적인 부분으로써 효율성이 높고, 멀티 쓰레드가 생성되어 병렬 연산 수행을 나타낸다. 그림 14와 같이 생성된 쓰레드의 개수는 2개로 듀얼코어의 코어를 최대한 활용하여 효율적인 연산을 수행함으로써 시간이 단축됨을 검증하였다.

V. 결론

본 논문에서는 효율성이 향상된 병렬 AES 암호 알고리즘을 제안하였다. 제안한 알고리즘은 최근 빠르게 발전하는 멀티코어 프로세서 환경에서 알고리즘을 효율적으로 최적화 하여 사용가능한 수행 성능을 평가하였다. 기존 AES 암호 알고리즘을 분석하고 재해석하여 논리적인 기능을 멀티코어 프로세서 환경에 적용하였다. 각 코어에 명령을 전달하여 128비트의 스테이트에서 각 행을 기준으로 일괄적인 연산을 제안하였다. 병렬 연산을 위해 OpenMP API를 활용하였고, 암·복호화 연산 수행시 자원 사용을 효율적으로 높였으며, 수행 속도 향상을 확인하였다.

제안한 병렬 연산 기법에 대한 성능과 시간복잡도를 통해 효율성을 분석하여 기존 알고리즘 신뢰적 수준 만족을 평가하였고, 프로파일링 툴을 통해 기존 암호화 알고리즘 보다 약 21%의 성능 향상이 있었으며, 암호화에 대한 보안 강도 또한 같음을 확인하였다.

전자·정보 통신 기술의 발전으로 급격히 발전하고 있는 멀티코어 프로세서 환경에 필요한 암호화 알고리즘을 병렬연산이 가능함을 확인 하였으며, 나아가 서버나 개인 PC 환경에만 국한 된 것이 아니라, 모바일 기기에서 결제나, 인증 또는 금융 서비스에 적용하여 암호 알고리즘의 경량화로 인한 취약점 해소가 가능하며, 향후 연구과제로, 소형 멀티코어 프로세서 환경에 맞는 최적화된 병렬 연산 암호 알고리즘에 대한 연구가 필요한 것으로 판단한다.

참 고 문 헌

[1] L. Di e, W., and Hellman, M., “New Direction in Cryptography,” IEEE Trans. Inform. Theory IT-22, pp.644-654, 1976

[2] Behrouz A. Forouzan, “Cryptography and Network Security”, McGraw Hill. 2007.

[3] William Stallings, “Cryptography and Network Security, Principles and Practices”, 4th ed. Pearson Education, 2006.

[4 L. T. Smit. G. K. Rauwerda, A. Molderink, P. T. Wolkotte, G. J. M. Smit, “Implementation of A 2-D 8x8 IDCT on The Reconfigurerable Montium Core,” International Conference on Field Programmable Logic and Applications, 2007, pp.562-566, Aug. 2007.

[5] Feng Liu and Vipin Chudhary, “Extending OpenMP for Heterogeneous Chip Multiprocessors,” Proceedings of the 2003 International Conference on Parallel Processing, Kaohsiung, Taiwan, pp.161-165, Oct. 2003.

[6] Amdahl, G.M. “Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities,” In AFIPS Conference Proceedings, AFIPS Press, pp. 483-485, Apr. 1967.

[7] J.M. Paul and B.H. Meyer, “Amdahl’ Law Revisited for Single Chip Systems,” International Journal of Parallel Programming, Vol. 35, No.2, pp.101-123, Apr. 2007.

[8] Audenaert. K., “Maintaining Concurrency Information for On-the-fly Data Race Detection,” Parallel Computing 97, North-Holland, pp.1-8, Sept. 1997.

[9] OpenMP Architecture Review Board, “OpenMP Fort ran Application Program In-terface Version 2.0,” Nov. 2000.

[10] Fei Cai, Shaogang Wu, Longbing Zhang and Zhiming Tang, “Parallel Program Performance Evaluation and Their Behavior Analysis on An OpenMP Cluster,” IEEE International Symposium on Cluster Computing and the Grid(CCGrid 2004), pp.508-514, Apr. 2004.

[11] Dagum, L, and R. Menon, “OpenMP: An Industry-Standard API for Shared-Memory Programming,” IEEE Computational Science and Engineering, Vol. 5, No. 1, pp.46-55, Mar. 1998.

[12] Vassilios V. Dimakopoulos and Elias Leontiadis, “A portable C Compiler for OpenMP V.2.0” In the proceeding of 5th European Workshop on OpenMP(EWOMP 2003), Aachen, Germany, pp.5-11, Sept. 2003.

[13] M. D. Jones, R. Yao, and C. P. Bhole, “Hybrid MPI-OpenMP Programming for Parallel OSEM PET Reconstruction,” IEEE Transactions on Nuclear Science, Vol. 53, No. 5, pp.2752 - 2758, Oct. 2006.

저 자 소 개

박 중 오

2000년 : 성결대학교 컴퓨터공학과
2003년 : 명지대학교 전자계산교육 석사
2011년 : 숭실대학교 컴퓨터학과 공학박사
2004년 : 성결대학교 객원교수
2006년~현재 : 성결대학교 정보산업기술연구소 전임연구원
관심분야 : 인터넷보안, RFID, 네트워크 보안, PKI,, 암호알고리즘,
Email : jopark02@sungkyul.edu

오 기 욱

1993년 : 숭실대학교 컴퓨터학과 공학석사
2007년 : 숭실대학교 컴퓨터학과 공학박사
2008년 : 강원관광대학 컴퓨터정보과 조교수
2010년~현재 : 안양대학교 교양학부 조교수
관심분야 : 컴퓨터공학
Email : ohgiug@paran.com