OpenMP를 활용한 LSH DRBG 병렬 최적 구현

권혁동* 안규황* 권용빈* 서화정*†

*한성대학교 IT응용시스템공학과

korlethean@gmail.com tigerk9212@gmail.com vexyoung@gmail.com hwajeong84@gmail.com

LSH DRBG parallel optimization using OpenMP

Hyeok-Dong Kwon* Kyu-Hwang An* Youg-Been Kwon*

Hwa-Jeong Seo*†

*Department of Applied IT Engineering, Hansung University.

요 약

결정론적 난수 발생기(Deterministic Random Bit Generator, DRBG)는 미국 표준 기술 연구소(National Institute of Standards and Technology, NIST)에서 권고하는 난수 발생 알고리즘으로 입력에 따라 의사 난수를 생성한다. DRBG의 내부에는 의사 난수 함수를 사용하는데 아직까지 구현된 적 없는 LSH DRBG를 구현하며, 나아가 DRBG의 규격 별로 또는 DRBG의 함수 중 유도함수와 내부 출력 생성 함수에서 반복적으로 호출되는 의사 난수 함수를 OpenMP를 통해 병렬화 처리를 적용하여 최적화를 하고자 한다.

I. 서론

일반적으로 난수 발생기는 무작위 값을 발생시켜야 하나 컴퓨터상에서는 완전 난수 생성이 어려우므로 입력 값에 따라 출력 값이 고정된 결정론적 난수 발생기를 사용한다.

결정론적 난수 발생기의 내부에는 의사 난수 함수를 사용하는데 해당 부분이 반복적으로 호출되며 결과 값은 독립적이므로 병렬화 적용이 가능하다. 이에 따라 국산 해시 함수인 LSH를 사용한 결정론적 난수 발생기를 구현하며 이에 병렬화를 적용하여 효율적인 최적화를 본다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구 동향을 확인하며 3장에서 제안 기법에 대해 살펴보며 4장에서 성능평가를 내리고 5장에서 결론을 맺는다.

II. 관련 연구 동향

2.1 결정론적 난수 발생기 (Deterministic Random Bit Generator)[1]

입력 값에 따라 출력이 고정된 난수 발생기를 의미한다. 알고리즘 내부에 적용하는 의사 난수 함수의 종류에 따라 해시 기반 DRBG와 HMAC 기반 DRBG로 나뉘지만 둘의 내부 구조는 다르다. 본 논문에서 언급하는 DRBG는 모두 해시 기반 DRBG 임을 명시하는 바이다.

2.2 LSH (Lightweight Secure Hash)[2]

LSH는 2014년 국가 보안 기술연구소에서 개발한 암호 학적 해시 함수로 Wide-pipe merkle Damård 구조가 적용되었으며 224, 256, 384, 512 비트의 출력 규격을 제공한다. 현재 국내 TTA 표준으로 제정되어 있다.

2.3 SSE (Streaming SIMD Extensions)[3]

SSE는 SIMD(Single Instruction Multiple Data)에 속하는 명령어 셋으로 SIMD의 특성을 활용하여 하나의 명령어로 절차 지향적 데이터

를 한 번에 처리할 수 있기에 전체적인 동작 속도를 향상시킨다.

2.4 AVX (Advanced Vector Extensions)[4]

AVX는 2008년 발표된 SIMD 명령어 셋으로 기존의 16바이트에서 증가한 32바이트를 동시에 계산 가능하다. 특히 부동소수점 연산이 뛰어나기 때문에 모델링, 실험, 그래픽 분야에서 유용하게 활용할 수 있다.

III. 제안 기법

DRBG에 병렬화를 적용하는 방법에는 데이터 병렬화와 태스크 병렬화 두 가지 방법이 있으며 이에 대해 서술한다.

3.1 데이터 병렬화 (Data Parallelism)

LSH는 총 6개 규격을 지원하며 후행 규격은 선행 규격의 출력을 대기하므로 병렬화를 거친다면 성능 향상을 기대할 수 있다. 단, 모든 규격별로 작업 시간이 동일하지는 않기 때문에 프로그램이 종료되기 위해서는 모든 규격이 완료되기를 대기해야 하며 이러한 요구 사항을 적용한 모델이 [그림. 1]로 각 규격마다 병렬적으로 동작함을 확인할 수 있다.

Algorithm 1

```
1. read test vectors
2. loop count = number of hash output types
3. operate OpenMP to for loop
4. For i = 1 to loop count
   4.1 DRBG using hash[i] type
   4.2 write DRBG result
   4.3 waiting until other processing finished
5. DRBG finished
```

[그림. 1] 데이터 병렬화가 적용된 LSH DRBG

3.2 태스크 병렬화 (Task Parallelism)

DRBG에서 유도함수와 내부 출력 생성 함수의 반복적인 의사 난수 함수 호출 부분을 병렬화한다. 반복 횟수 산정은 유도함수에서는 규격에 따라 2회, 3회로 고정되며 내부 출력 생성 함수는 출력 길이에 따라 비례한다. 각 함수마다 병렬화가 적용된 모델은 [그림. 2], [그림. 3]과 동일하다.

Algorithm 2

```
1.1 If hash bits is 224 or 256
      seed bits = 440
1.2 If hash bits is 384 or 512
      seed bits = 888
2. len = ceil (seed bits / hash bits)
3. counter = 0x01
4. input = counter || seed bits || msg from parameter
5. operate OpenMP to for loop
6. For i = 1 to len
   6.1 output[i] = Hash(input)
   6.2 counter + = 1
   6.3 refresh input value
   6.4 waiting until other processing finished
7. final output = (output[1] || … || output[n]) mod 2^seed bits
```

[그림. 2] 태스크 병렬화가 적용된 유도함수

Algorithm 3

```
1. len = ceil (output bits from parameter / hash bits)
2.1 If hash bits is 224 or 256
      seed bits = 440
2.2 If hash bits is 384 or 512
      seed bits = 888
3. data = state V from parameter
4. operate OpenMP to for loop
5. For i = 1 to len
   5.1 output[i] = Hash(data)
   5.2 data = (data + 1) mod 2^seed bits
   5.3 waiting until other processing finished
6. final output = (output[1] || … || output[n]) mod 2^output bits
```

[그림. 3] 태스크 병렬화가 적용된 내부 출력 생성 함수

IV. 성능평가

성능 평가에 앞서 DRBG 구현 및 작업 환경은 [표. 1]과 동일함을 명시하며 테스트에 사용한 구현물은 깃허브[1]에서 열람이 가능하다.

[표. 1] 작업 환경

운영체제	Windows 10 Pro
프로세서	Intel Core i7-8550U CPU 1.8GHz
컴파일러	MinGW
IDE	Eclipse Photon (4.8.0)
언어	C

4.1 데이터 병렬화 성능 평가

LSH 규격별로 병렬화를 적용하여 6개 규격이 동시에 출력을 진행하도록 한다. 모든 규격 출력이 완료된 시점을 1회로 설정하여 테스트 벡터 60종을 1,000회 반복하며 예측 내성 지원

1) https://github.com/korlethean/drbg_with_mp

설정을 한다. 대조군은 동일 환경에서 병렬화를 적용하지 않으며 실험 결과는 [표. 2]와 같다.

[표. 2] 데이터 병렬화 비교

	평균 수행 시간(ms)	cpb
대조군	73	491
병렬화	27	181

객관적인 평가를 위해 수행 시간을 CPB(Clock cycle Per Bytes)로 환산한다. 비교 결과 병렬화 적용 모델이 약 2.71배의 효율을 보인다.

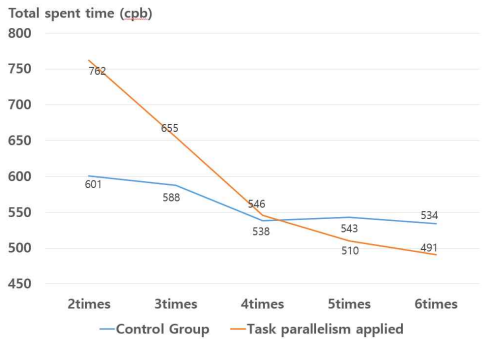
4.2 태스크 병렬화 성능 평가

DRBG의 내부 구조에서 의사 난수 함수를 반복적으로 호출하는 부분에 병렬화를 적용한다. 4.1절과 동일한 환경에서 진행하며 비교 결과는 [표. 3]과 같다.

[표. 3] 태스크 병렬화 비교

	평균 수행 시간(ms)	cpb
대조군	73	491
병렬화	149	1001

태스크 병렬화를 적용할 경우 오히려 대조군에 비해 약 2.04배 저하된 성능을 보여준다. 이는 병렬화 과정에서 발생하는 오버헤드로 인한 것으로 판단되었다. 오버헤드는 필연적으로 발생하기 때문에 반복 횟수를 증가시키면 효율이 증가한다. 이에 따른 결과는 [그림. 4]와 같다.



[그림. 4] 출력 길이별 비교 그래프

출력 길이가 기존 대비 5배 이후일 때 병렬화 모델의 동작 효율이 좋으며 출력 길이가 증가하더라도 동작 효율의 변화가 크기 않음을 확인 가능하다.

V. 결론

본 논문에서는 DRBG 구조에 병렬화를 적용하여 최적화 구현을 시도하였고 그 성능이 통상적인 DRBG 동작보다 뛰어남을 확인하였다.

데이터 병렬화는 다수 규격의 DRBG 출력에 유리하며 태스크 병렬화는 표준 규격에는 부적합하지만 이론상으로 병렬화 적용을 통해 성능 향상이 가능함을 보였다.

병렬화 기법은 성능 개선을 위한 방법 중 하나로 다양한 분야에 활용된다. 이때 병렬화를 적용하면 필연적으로 오버헤드가 발생한다는 점을 항상 염두에 두며 맹목적인 적용이 아닌 적절한 병렬화 적용을 시도해야만 의도한 성능 향상을 이끌어 낼 수 있다.

[참고문헌]

[1] National Institute of Standards and Technology, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators”, Available: https://www.nist.gov/publications/recommendation-random-number-generation-using-deterministic-random-bit-generators-2

[2] Telecommunications Technology Association, “Hash function Full structure and compression function of LSH”, Available: http://commitee.tta.or.kr/data/standard_view.jsp?nowPage=2&pk_num=TTAK.KO-12.0276&commit_code=TC5.

[3] Srinivas K. Raman, Vladimir Pentkovski, Jagannath Keshava, “Implementing Streaming SIMD Extensions on the Pentium III processor”, Available: https://www.computer.org/csdl/mags/mi/2000/04/m4047.pdf

[4] Chris Lomont, “Introduction to Intel® Advanced Vector Extensions”, Available: https://software.intel.com/sites/default/files/m/d/4/1/d/8/Intro_to_Intel_AVX.pdf