GPU를 이용한 Polymorphic worm 탐지 기법 구현 및 GPU 플랫폼에 따른 성능비교

이선원*, 송치환*, 이인준*, 조태원*, 강재우* **
*고려대학교 정보통신대학, **교신저자
e-mail:{sunwonl, chihwan, seminoles, twjoh, kangj}@korea.ac.kr

Implementation of GPU Based Polymorphic Worm Detection Method and Its Performance Analysis on Different GPU Platforms

Sunwon Lee*, Chihwan Song*, Injoon Lee*, Taewon Joh*, Jaewoo Kang* **
*College of Information and Communications, Korea University
**Corresponding author

요 약

작년 7월 7일에 있었던 DDoS 공격과 같이 악성 코드로 인한 피해의 규모가 해마다 증가하고 있다. 특히 변형 웜(Polymorphic Worm)은 기존의 방법으로 1차 공격에서의 탐지가 어렵기 때문에 그 위험성이 더 크다. 이에 본 연구에서는 바이오 인포매틱스(Bioinformatics) 분야에서 유전자들의 유사성과 특징을 찾기 위한 방법 중 하나인 Local Alignment를 소개하고 이를 변형 웜 탐지에 적용한다. 또한 수행의 병렬화 및 알고리즘 변형을 통하여 기존 알고리즘의 $O(n^4)$수행시간이라는 단점을 극복한다. 병렬화는 NVIDIA사의 GPU를 이용한 CUDA 프로그래밍과 AMD사의 GPU를 사용한 OpenCL 프로그래밍을 통하여 수행되었다. 이로써 각 GPGPU 플랫폼에서의 Local Alignment를 이용한 변형 웜 탐지 알고리즘의 성능을 비교하였다.

1. 서론

OECD의 2008년 보고서에 의하면, 악성 코드에 의해 영국의 한 은행의 직접적 피해 규모는 2004년 12.2백만 파운드에서 2006년 33.5백만 파운드로 90% 증가했다고 한다. 하지만 이것은 온라인 거래의 고객 신뢰도 감소, 이미지 실추 등으로 인한 피해는 반영되지 않은 것으로 실제 피해 규모는 더 클 것이다. 52명의 기술 전문가의 조사에 의하면 악성 코드로 인한 피해는 2004년 122억 유로에서 2006년 93억 유로로 감소하였는데 이런 감소는 간접 혹은 2차 피해가 실제적으로 증가하였기 때문이라고 분석하고 있다. 또 다른 조사는 미국의 매년 손실이 672억 달러에 달한다고 평가했다.

반면, 악성코드로 인한 개인의 피해도 상당할 것이라 예상되지만 정확한 피해 규모의 측정이 어렵다. 미국의 예로, 개인 사용자들은 지난 2년간 바이러스나 스파이웨어에 감염된 시스템을 수리하거나 교체하기 위해 78억 달러를 지불한 것으로 나타난다[2].

국내에서도 최근 악성코드가 개인 PC를 좀비화하여 특정 사이트의 DDoS공격에 이용하는 등 악성코드의 이용 범위와 그 위협의 정도가 심해지고 있다.

그 중에서도 변형 웜(polymorphic worm)은 코드 자기 변형 기술(polymorphic techniques)을 사용하여, 희생이 되는 호스트를 옮겨갈 때 마다 수행 동작은 바뀌지 않으면서 자신의 내용을 바꾸는 악성코드(malware)이다. 일반적으로 악성코드를 탐지하기 위해 두 가지의 접근 방법이 사용 된다. 하나는 행동 기반 접근 방식(behavior-based approach)이고 다른 하나는 내용 기반 접근 방식(content-based approach)이다[1]. 행동 기반 접근 방식에서는 악성코드로 인한 네트워크 트래픽의 이상 현상이 연구의 대상이 된다. 반대로, 내용 기반 접근 방식은 악성코드 저장소에서 악성 코드의 시그니처를 만들어서 새롭게 들어오는 악성코드를 탐지한다. 이 논문은 내용 기반 접근 방식에 근거해서 변형 웜을 잡는 것에 기반을 둔다.

기존의 침입 탐지 시스템(Intrusion Detection Systems :IDSs)은 유입된 바이너리 데이터와 보유하고 있는 악성코드 시그니처를 비교하여 바이너리 데이터가 악성코드인지 식별한다. 보안 전문가들은 네트워크 트래픽의 이상이나 호스트 PC의 실질적 피해에 의해 파악된 악성코드를 통하여 시그니처를 생성하고 이를 IDS에서 사용하는 시그니처 데이터베이스에 추가하게 된다. 이러한 이유로, 변형 웜의 초반 공격은 발견되지 못한다.

이러한 변형 웜에 대한 피해를 사전에 막기 위해 많은 연구가 있어왔다. 변형 웜이 시그니처 기반의 IDS들을 피하기 위해 자신의 패턴을 변형시킬 지라도, 페이로드 부분에 변형되지 않는 내용이 존재한다[1]. 변형 웜에 대한 연구자들은 이 사실을 이용하여 시그니처들을 만든다.

페이로드 부분의 변하지 않는 부분이 있다는 사실에 착안하여 이 연구는 바이오 인포매틱스(Bioinformatics) 분야에서 유전자의 유사성을 비교하기 위한 방법 중 하나인 Local Alignment 알고리즘을 적용하여 변형 웜 탐지의 근간 기술을 제안하려한다.

또한, 대표적인 GPU를 이용한 병렬 프로그램인 NVIDIA 사의 CUDA와 AMD GPU를 사용하는 OpenCL을 이용하여 변형 웜 탐지 기술을 구현한다. 병렬 프로그램을 통한 변형 웜 탐지 기술의 구현은 웜 탐지에 대한 시간적 효율성을 높일 뿐 아니라, CUDA와 OpenCL의 가격대비 성능을 직접적으로 비교할 수 있는 좋은 기회가 될 것이다.

이 논문은 다음의 형식으로 구성되어있다. 2장에서는 변형 웜을 탐지하기위해 사용된 알고리즘과 데이터 셋을 소개한다. 3장에서는 CUDA와 OpenCL을 통한 병렬화의 특징을 소개한다. 4장에서는 2장에서 소개한 알고리즘의 병렬화 방법에 대하여 소개한다. 5장에서는 CUDA와 OpenCL의 속도 성능을 구현된 알고리즘을 통하여 비교한다. 6장에서는 이 논문의 맺음을 한다.

2. 관련 연구

이 논문에서 다루는 주제는 크게 두 가지이다. 하나는 변형 웜을 탐지하기 위함이고, 다른 하나는 이를 GPU를 사용하여 구현하고, 각 플랫폼에서의 성능의 차이를 보기 위한 것이다. 먼저, 변형 웜을 찾기 위해, 많은 연구들이 수행되었다.

LCS 알고리즘을 사용하여 변형웜의 탐지에 사용할 시그니처를 생성하는 시스템은 다음과 같다.

Polygraph[4]는 변형 웜에도 protocol framing이나 return address처럼 변하지 않는 부분 문자열들이 존재한다는 사실에 착안한다.

해당 변형 웜의 특징을 나타내는 이들 부분 문자열들을 찾고, 이를 조합하여 새로운 변형 시그니처를 생성하는 몇 가지 방법을 보여준다.

Honeycomb[5]은 네트워크 침입 탐지 시스템으로 네트워크 패킷의 흐름상에서 공통의 시퀀스를 찾는데 이를 사용한다.

이 시스템은 시그니처로 가장 긴 하나의 공통 문자열만을 사용하기 때문에 공통의 시퀀스의 길이가 짧은 변형웜의 경우에는 탐지가 용이하지 않다는 단점이 있다.

GPU 프로그래밍이 이전에는 그래픽스 분야에서만 사용되는 특수한 영역이었지만, 근래 들어 CUDA와 같은 범용 아키텍처가 발표 되면서 과학, 인공지능 등의 분야에서도 사용하게 되었다. Michael C. Schatz와 그의 연구팀은 기존에 사용하고 있던 Gene Secquence Alignment 도구인 MUMmer를 CUDA를 이용한 병렬화를 수행하여 성능을 개선하였다.[6] 이 연구에서 그들은 약 3.5배 이상의 성능 향상을 기록하였다. 이 외에도, Joao Filipe Ferreira의 연구팀이 수행한 연구에서는 CUDA를 이용하여 실시간 Bayesian 인식 알고리즘을 구현하였다.[7] 그들은 인공시각 시스템에 GPU를 사용하는 추론시스템을 추가하여 실시간으로 주변상황을 인식하도록 하였다. 이와 비슷하게, 인공신경망을 CUDA로 구현한 사례도 있었다.[8] GPU를 Co-processor로서 사용한 아키텍처는 계산 집약적인 시뮬레이션에도 많이 이용된다. 예를 들어, Dipankar Das 연구팀은 SoC에서의 작업 스케줄에 따른 발열 시뮬레이션을 CUDA를 통해 구현하였고,[9] Xiaohui Ji 및 그의 동료는 그들의 연구에서 대량의 지하수 흐름에 대한 시뮬레이션을 GPU를 사용하여 수행하였다.[10]

3. 배경

3.1. Local alignment

```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                      ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

그림 1 Local Alignment 예제

Local alignment[11]는 바이오인포매틱스 분야에서 유전자의 유사성을 측정하기위해 사용되는 알고리즘 중 하나이다. 예를 들어 homeobox 유전자들의 종들 사이에서 보존되는 homeodomain이라 불리는 작은 부분을 찾기 위해 local alignment가 이용된다. Local alignment는 **그림 1**과 같이 길이 n, m의 두 유전자 배열 A, B가 있을 때 배열이 일치하는 A[i]~A[i']와 B[j]~B[j']를 찾는 문제이다.

문제의 유용성에도 불구하고 실제 생활에 적용하기 어려운 부분이 있는데 바로 수행 속도이다. n 길이의 두 시퀀스를 비교하기 위해 n^2 크기의 그리드가 존재하고 그리드의 각 vertex에서 시작하는 alignment를 찾기 위한 n^2의 수행속도가 필요하다. 결국 $O(n^4)$의 수행시간을 가지게 된다.

3.2. CUDA와 OpenCL

NVIDIA의 CUDA[12]는 패러렐한 컴퓨팅 엔진이 많은 CPU에서 많은 시간을 필요로 하는 복잡한 문제들을 해결 가능하도록 하기 위해 도입한 일반적 목적의 패러렐 컴퓨팅 아키텍처이다. 그것은 CUDA 명령어 셋 아키텍처와 GPU에서의 패러렐 엔진 계산을 포함한다. CUDA는 두 개의 주된 기능을 수행하는데, GPU의 스트림 프로세서를 사용함으로써 CPU의 사용을 감소하도록 도움을 주는 것과, CUDA를 사용한 계산 프로세스를 가속화 하는 것이다.

NVIDIA는 100만개 이상이나 되는 CUDA 사용가능한 GPU를 판매해 왔으며, 그것들은 수천명의 소프트웨어 개발자들에 의해 지지 받으며 NVIDIA 얘기하는 것처럼 무료 CUDA 소프트웨어 개발 툴을 사용하여 다양한 전문적인 혹은 가정용 응용 프로그램으로 다양한 문제를 해결하는 데 사용된다.

또 다른 그래픽 하드웨어의 메이저 회사인 ATI는 진보된 하드웨어와 소프트웨어 기술에 기초하여 ATI 스트림

이라는 본격적인 GPGPU의 본격적인 병렬 프로세싱 기술을 선보였다. 이 기술을 통하여 OpenCL[13]의 실행에 놀라운 진전을 가져왔으며, 올 하반기에 OpenCL의 퍼블릭 베타와 프로덕션 버전을 ATI 스트림 SDK 2.0의 일부로 출시 될 예정이다.

원래 OpenCL은 애플사가 최초로 개발한 병렬 프로그래밍 언어로써 크로노스 컴퓨팅 워킹 그룹에 의해 OpenCL 1.0이 완성되었는데, 2008년 11월 18일 기술적인 상세를 담은 OpenCL 1.0 명세서 역시 완성되어진바 있다.
OpenCL을 사용하는 ATI 스트림과 CUDA는 지금도 계속적으로 그래픽 프로세서가 그래픽을 넘어서서 많은 응용을 가속화 하도록 소프트웨어와 하드웨어의 기술적 진보를 이루고 있다.

4. Local Alignment의 알고리즘과 병렬화

일반적인 local alignment 알고리즘에서는 길이 n, m의 두 시퀀스에 대해 n * m 크기의 행렬을 만들고, 각 셀이 시작점으로 되는 모든 경우의 가능한 서브시퀀스를 탐색하게 된다. 본 연구에서는 이의 병렬화를 위해서 알고리즘을 4단계로 세분화 시켰다.

4.1 Labeling (병렬처리)

Labeling 단계에서는 n * m 크기의 행렬의 각 셀에 바이너리 데이터의 일치 여부를 1과 0으로 기록하게 된다. 프로세스는 다음과 같다. 각 thread는 자신이 비교해야하는 두 시퀀스의 서브시퀀스인 n′, m′을 전달 받아 바이너리의 비교를 하게 된다. 두 서브 시퀀스의 바이너리 n′[i]와 m′[j]가 일치한다면 n′ * m′크기의 행렬 mat′[i][j]에 1을, 일치하지 않다면 0을 기록한다. 메인 프로그램은 각 thread의 mat′을 취합하여 n * m 크기의 행렬을 갖게 된다. 이 단계의 수행시간은 $O(n^2)$이다. 병렬화를 통해 하나의 thread가 k 개의데이터를 동시에 처리할 때 thread의 개수는 n * (n/k)이기 때문에 최종 수행시간은 다음과 같이 줄일 수 있다.

$$O\left(\frac{n^2}{n\times\frac{n}{k}}\right)=O(k)$$

수식 1 Labeling의 수행속도

4.2 Rotation

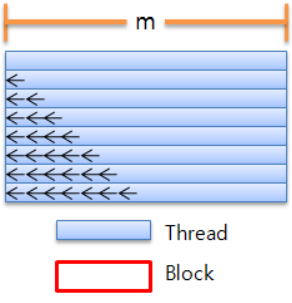

그림 2 Rotation

Rotation은 메인 프로그램이 처리하는 부분으로 이후 프로세스의 병렬화를 위해 필요한 과정이다. **그림 2**와 같이 각 행은 자신의 index 번호만큼 왼쪽으로 shift를 하게 된다. 결과적으로 연속의 match된 시퀀스는 세로로 정렬하게 된다. 이 단계의 수행시간은 O(n)이다.

4.3 Transpose (병렬처리)

Transpose 는 4.3을 병렬처리하기 위한 행렬의 행렬 변환이다. 각 thread는 부분행렬 mat'를 받아서 $mat'[i][j]$의 값을 $mat''[i][j]$에 저장하게 된다. 메인 프로그램은 mat''을 취합하여 행렬 변환된 행렬을 얻게 된다. 이 단계의 수행시간은 $O(n^2)$이다. 병렬화를 통해 하나의 thread가 k 개의데이터를 동시에 처리할 때 thread의 개수는 n * (n/k)이기 때문에 최종 수행시간은 다음과 같다.

$$O\left(\frac{n^2}{n\times\frac{n}{k}}\right)=O(k)$$

수식 2 Transpose의 수행속도

4.4 Finding the longest alignment (병렬처리)

4.2절을 통해 시퀀스의 매칭여부는 행방향의 0과 1로 표현이 되었다. 각 행은 각 thread로 보내어져 각 행의 max score와 max score를 기록한 index를 찾게 된다. 분기문을 사용하지 않고 max score와 index를 얻는 방법은 **그림 3**의 의사코드를 통하여 구현하였다. 이 단계의 수행시간은 O(n)이다.

```
1  for(int j=0;j<m;j++)
2  {
3    score = __mul24((score+ 1),mat[i+j]);
4    max_pos[max_score[1] < score] =j;
5    max_score[max_score[1] < score] = score;
6  }
```

그림 3 Scoring의 Pseudo Code

4.5 수행속도

전체 수행시간은 다음과 같다.

$$O(k+n+k+n)=O(n)$$

수식 3 전체 process의 수행속도

5. 테스트 및 결과

5.1 테스트 환경

테스트는 두 대의 PC로 수행되었고 두 PC의 CPU 및 메모리는 동일하다. GPU는 NVIDA GTS250과 ATI Radeon HD 4870을 이용하였고 각각의 사양은 표 1에 자세히 나와있다.

	NVIDIA GTS250	ATI Radeon HD 4870
Clock rate	1.85 GHz	775 MHz
Computing Unit	128 Cores	800 stream units
Memory	1 Gigabytes	1 Gigabytes
Price	237,000원	320,000원

표 1 테스트 GPU 비교

5.2 성능 비교

실험은 변형 웜 샘플 2개와 정상 바이너리의 샘플 2개, 총 4개의 샘플 중, 2개의 조합으로 비교하여 6번의 scoring을 하였다. 변형 웜의 샘플로는 국내 안티 바이러스 프로그램인 '알약'이 'ply 5133'으로 판명한 크기 5090, 5060 바이트의 샘플 2개를 이용하였다. 'ply'는 [3]에서 변형 웜의 한 종류라고 소개하고 있는데, 실제로 바이너리 파일을 비교한 결과 dummy와 같은 부분을 코드 앞부분에 추가하는 방법의 polymorphic technique를 사용하여 변형되었음을 확인 했다. 정상 바이너리 샘플은 윈도우 시스템 라이브러리 파일에서 가져온 크기 5980 바이트의 dll 파일과 크기 6000 바이트의 비트맵을 이용하였다.

	GTS 250		HD 4870	
	Time	Score	Time	Score
n1-n2	1722	10	1795	10
n1-m1	1438	10	1571	10
n1-m2	1443	10	1522	10
n2-m1	1423	23	1578	23
n2-m2	1434	23	1575	23
m1-m2	1338	5185	1411	5185

표 2 성능 비교

실험 결과는 표 2로 기술하였다. 표에서 n1과 n2는 정상 바이너리이고 m1과 m2는 변형 웜이다. Time은 ms 단위이고 score는 가장 긴 서브시퀀스의 길이이다. 속도에서 GTS 250이 70ms 정도의 성능 우위를 보이고 있다. 보통의 경우 GPU의 성능을 비교하기위해 가격대비 flops 수치 비교를 하지만 실수 연산이 아니므로 위의 결과로 가격대비 flops 수치를 구하는 것은 큰 의미가 있다고 하기는 어렵다.

6. 결론

Local Alignment를 이용한 변형 웜의 탐지가 가능함은 스코어링의 결과로 알 수 있었다. 또한 알고리즘의 병렬화를 통해서 실시간 탐지가 가능한 속도를 이끌어 낼 수 있었다. 작년 7.7 인터넷 대란과 같은 악성 코드에 의한 위험성이 대두되고 있는 시점에 변형 웜의 실시간 탐지 기술은 경제적으로도 학문적으로도 큰 의미를 갖는다.

실험 결과 CUDA 와 OpenCL 에서의 큰 성능 차이가 있다고는 결론지을 수 없었으나 GPU와 같은 별도의 계산 노드를 이용한 병렬화를 통해 변형 웜 탐지 기법의 가속화를 이룰 수 있을 것으로 기대한다.

참고문헌

[1] T. F. Smith, M. S. Waterman, "Identification of Common Molecular Subsequences", *J.Mol.Biol.*,1981.

[2] "Malicious Software (Malware): A Security Threat to the Internet Economy" OECD Ministerial Meeting on the Future of the Internet Economy, Seoul, Korea, 2008.

[3] Kaspersky Lab viruslist,'http://www.viruslist.com'.

[4] J. Newsome, B. Karp and D. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms," IEEE Symposium on Security and Privacy, 2005.

[5] C. Kreibich and J. Crowcroft, "Honeycomb: Creating Intrusion Detection Signatures Using Honeypots", ACM SIGCOMM Computer Communications Review, 2004.

[6] M. Schatz, et al., "High-throughput sequence alignment using Graphics Processing Units," BMC bioinformatics, vol. 8, p. 474, 2007.

[7] J. Ferreira, et al., "Bayesian real-time perception algorithms on GPU," Journal of Real-Time Image Processing, pp. 1-16, 2010.

[8] H. Jang, et al., "Neural Network Implementation Using CUDA and OpenMP," Digital Image Computing: Techniques and Applications, pp. 155-161, 2008 Digital Image Computing: Techniques and Applications, 2008

[9] D. Das, et al., "Thermal analysis of multiprocessor SoC applications by simulation and verification," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 15, pp. 1-52, 2010.

[10] X. Ji, et al., "A simulation of large-scale groundwater flow on CUDA-enabled GPUs," presented at the Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 2010.

[11] Neil C. J., Pavel A. P., "An Introduction to Bioinformatics Algorithms", The MIT Press.,2004.

[12] 'http://www.NVIDIA.com/object/CUDA_home.html'

[13] 'http://www.KHRONOS.org/OpenCL'