유니커널을 이용한 Haskell 병렬 프로그래밍 기법의 성능 분석

김연어[O1], 류샤오[1], 변석우[2], 우균[1,3]
[1]부산대학교 전기전자컴퓨터공학과, [2]경성대학교 컴퓨터공학과, [3]LG전자 스마트 제어 센터
yeoneo@pusan.ac.kr, liuxiao@pusan.ac.kr, swbyun@ks.ac.kr, woogyun@pusan.ac.kr

A Performance Analysis of Haskell Parallel Programming Methods Using Unikernel

Yeoneo Kim[O1], Xiao Liu[1], Sugwoo Byun[2], Gyun Woo[1,3]
[1,3]Dep. of Electrical and Computer Engineering, Pusan National University,
[2]School of Computer Science & Engineering, Kyungsung University,
[3]Smart Control Center of LG Electronics

요 약

클라우드 컴퓨터는 계산 환경의 발달과 네트워크 고속화로 현재 널리 사용되고 있는 서비스이다. 특히 클라우드 컴퓨터를 효과적으로 제공하기 위한 운영체제 중 하나인 유니커널은 매니코어 환경에서도 주목을 받고 있다. 이 논문에서는 매니코어 환경에서 병렬 프로그래밍 성능이 우수한 Haskell이 유니커널에서 동작하는 경우의 성능을 분석하고자 한다. 이를 위해 이 논문에서는 HaLVM을 대상 운영체제로 선택하였으며, 다양한 HaseklI 병렬 프로그래밍 모델을 분석 및 실행하였다. 그 결과 HaLVM에서는 IVC만 확장성이 확보되는 것을 확인하였다. 그리고 실험을 통해 리눅스 환경과 확장성을 비교한 결과 코어 수가 늘어날수록 HaLVM의 확장성이 더 확보되는 것을 확인하였다.

1. 서 론

최근 계산 환경의 발달과 네트워크의 고속화로 인해 많은 IT 서비스가 네트워크를 통해 이루어지고 있다. 많은 IT 서비스 중 클라우드 컴퓨터는 네트워크를 통해 컴퓨팅 환경을 제공하는 것이다. 이러한 클라우드 컴퓨터는 범용 목적으로 사용될 수 있는 가상 서버 호스팅에서부터 최근 구글에서 발표한 게임 전용 클라우드 컴퓨터인 스태디아(Stadia)까지 다양한 목적으로 사용되고 있다.

특히 최근에는 전용 목적의 클라우드 서비스를 효과적으로 제공하기 위해 유니커널(unikernel)이라 불리는 운영체제가 관심을 받고 있다. 유니커널은 응용 프로그램에 특화된 운영체제로 라이브러리 운영체제(library operation system)라고도 불리고 있다[1]. 이 운영체제는 기존 운영체제와 달리 범용 목적이 아닌 하나의 서비스를 제공하는데 특화된 운영체제로 클라우드 환경에 적합하다.

이러한 유니커널은 클라우드 서비스뿐만이 아니라 매니코어 환경에서도 관심을 받고 있다. 매니코어 환경에서 기존 리눅스를 그대로 사용하면 복잡한 구조로 인해 확장성을 확보하기 어렵다. 그렇기 때문에 매니코어 환경에서는 기존 리눅스보다 단순하면서 오버헤드가 작은 유니커널이 관심을 받고 있다[2].

또한, 매니코어 환경을 고려한다면 Haskell을 빼놓을 순 없다. Haskell은 순수 함수형 언어로 병렬 프로그래밍에 적합하다고 알려져 있으며, 매니코어 환경에서도 높은 확장성을 보인다. 또한, 다양한 병렬 및 동시성 프로그래밍 모델을 제공하기 때문에 상황이나 환경에 따라 병렬 프로그래밍 모델을 선택할 수 있다.

그렇기 때문에 이 논문에서는 매니코어 환경에서 유니커널과 Haskell 병렬 프로그래밍이 만났을 때 어떤 효과를 보이는지를 살펴보고자 한다. 이를 위해 이 논문에서는 기존의 다양한 유니커널 중 Haskell로 작성된 HaLVM(Haskell lightweight virtual machine)[3, 4]을 대상으로 한다. 그리고 HaLVM에서 다양한 병렬 프로그래밍 모델을 작성하여 프로그램 동작 여부와 성능을 분석하고자 한다.

이 논문의 구성은 다음과 같다. 우선 2장에서 관련 연구로 유니커널에 대해 소개하고자 한다. 그리고 3장에서는 HaLVM에서 검증할 Haskell 병렬 프로그래밍 모델을 소개하고자 하고 HaLVM에서 동작 여부를 확인하고자 한다. 이후 4장에서는 실험을 통해 HaLVM 기반 Haskell 병렬 프로그램과 기존 리눅스 환경과 확장성을 비교하고자 한다. 그리고 5장에서는 실험 내용에 대해 논의하고 6장에서 결론을 맺고자 한다.

2. 관련 연구

이 장에서는 HaLVM 기반의 Haskell 병렬 프로그래밍의 성능 분석에 앞서 관련 연구로 유니커널과 HaLVM에 대해 설명하고자 한다. 유니커널은 기존 범용 운영체제인 리눅스와 달리 응용 프로그램에 특화된 운영체제로 라이브러리 운영체제라 불리고 있다. 유니커널의 기본 개념은 단일 유저, 단일 주소 공간, 단일 프로세스로 구현되었으며, 이 때문에 커널/유저 모드 전환이 없어 오버헤드가 없는 것이 특징이다. 또한, 기존 범용 운영체제와 달리 불필요한 기능을 제거하였기 때문에 구동 속도가 빠르며, 공격에 노출되는 영역이 작아 보안이 우수하다.

이러한 유니커널은 다양한 연구 및 구현이 진행되고 있으며, 연구 방향은 크게 두 가지로 나누어져 있다. 첫 번째 연구 방향은 기존(legacy) 코드를 지원하는 방향이다. 이 방향은 기존 리눅스나 유닉스에서 사용되던 POSIX를 유니커널에서도 지원하여 기존 프로그램을 그대로 사용할 수 있게 하는 것이다. 이 분야의 구현으로는 IncludeOS(C/C++) Osv(Java, Lia, Go), Rumprun(Ruby, Go, Python), Graphhene(C), HermitCore(GCC)가 있다.

두 번째 방향은 운영체제를 처음부터 새로(clean-slate) 구현하는 것이다. 이 방향에서는 기존 운영체제가 C 언어로 작성되어 안전성이 떨어지며 검증이 어렵다는 판단하에, 명령형 언어보다 안전한 함수형 언어를 이용하여 커널을 처음부터 구현한 것이다. 이 때문에 기존 프로그램과 호환성이 없지만, 안전성이 높은 것이 특징이다. 이 분야의 구현으로는 MirageOS(Ocaml), HaLVM(Haskell), LING(Erlang)이 있다.

3. HaLVM 기반 Haskell 병렬 프로그래밍 분석

3.1 HaLVM

이 절에서는 HaLVM에서 Haskell 병렬 프로그래밍이 잘 동작하는지 살펴보기에 앞서 HaLVM의 구조 및 특징에 대해 살펴보고자 한다. HaLVM은 Galois에서 Haskell로 오픈 소스로 구현한 유니커널로, 기존 마이크로커널 기반의 운영체제의 문제점을 해결하기 위해 개발되었다. HaLVM은 Xen을 기반으로 작성된 유니커널로 소규모 단일 용도 프로그램이면서 작은 의존성을 가진 프로그램에 적합하다고 알려져 있다. 이러한 HaLVM은 많은 기능을 Haskell로 구현하였으며, 구조는 그림 1과 같다.

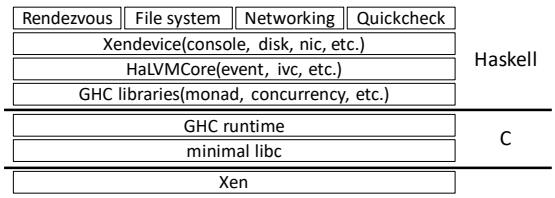

그림 1 HaLVM의 구조[3]

그림 1의 HaLVM 구조를 살펴보면 Haskell로 구현된 부분과 C로 구현된 부분으로 나누어져 있는 것을 알 수 있다. C로 구현된 부분은 기존 Haskell 컴파일러에서 사용되던 부분으로 Haskell 코드를 기계어 코드로 변경하는 것과 Haskell 가상 기계(virtual machine)와 관련된 코드가 포함된 부분이기에 필요한 기능이다.

Haskell로 구현된 부분은 가상 기계 관련이나 기계어 변환 코드 이외의 모든 부분이다. 대표적으로 기존에는 파일시스템이나 네트워크, Xen 관련 코드는 모두 C나 C++로 작성되었다. 하지만 HaLVM에서는 이러한 기능을 비롯하여 다양한 라이브러리가 모두 Haskell로 구현된 것이 특징이다.

HaLVM의 다양한 라이브러리 중 병렬 프로그래밍에서 중요한 모듈로는 Rendezvous가 있다. 이 모듈은 HaLVM에서 유니커널 간의 통신을 위해 제공하는 것으로 HaLVM에서는 이를 이용한 통신은 IVC(inter virtual machine communication)라 표현하고 있다. IVC는 프로그램 실행 전 하이퍼바이저(hypervisor)가 동작하는 dom0에서 Rendezvous 모듈을 실행되며 IVC를 이용한 데이터는 Xen을 통해 연결된 다른 가상 기계에 전달되는 구조이다.

3.2 HaLVM 기반 Haskell 병렬 프로그래밍 분석

이 절에서는 다양한 Haskell 병렬 프로그래밍 모델을 HaLVM에서 동작할 수 있는지에 대해 살펴보고자 한다. Haskell은 기본적으로 다양한 병렬 프로그래밍 모델을 제공하여 상황에 따라 적합한 병렬 프로그래밍 모델을 선택할 수 있다. 하지만 이러한 병렬 프로그래밍 모델은 리눅스를 기반으로 제작되었기 때문에 HaLVM에서 잘 동작할 수 있는지에 대한 확인이 필요하다. 이 논문에서 검증을 위해 5종의 병렬 프로그래밍 모델을 사용하였다.

첫 번째 병렬 프로그래밍 모델은 Eval 모나드이다. Eval 모나드는 병렬 전략(strategy)을 이용하여 병렬 프로그램을 수행하는 모델이다. Eval 모나드는 GHC에서 제공하는 병렬 단위 중 가장 작은 스파크(spark)를 이용하여 병렬 작업이 수행되며 항상 병렬 실행이 보장되지는 않는 모델이다.

두 번째 병렬 프로그래밍 모델은 forkIO이다. 이 모델은 Haskell 언어에서 제공하는 스레드를 사용하여 병렬 프로그램을 실행하는 것이다. forkIO는 언어에서 제공하는 스레드를 사용하기 때문에 병렬 단위는 그린 스레드(green thread)로 분류할 수 있으며, 해당 스레드는 운영체제에서 제공하는 스레드보다 작기 때문에 오버헤드가 작은 것이 특징이다.

세 번째 병렬 프로그래밍 모델은 forkOS이다. 이 모델은 OS에서 제공하는 스레드를 사용하여 병렬 프로그램을 실행하는 것이다. 즉, 해당 모델의 병렬 단위는 스레드라 볼 수 있다. 그리고 forkOS는 forkIO와 타입이 같기 때문에 사용 방법은 같지만 forkIO보다 큰 메모리를 사용하는 스레드가 실행되는 것이 특징이다.

네 번째 병렬 프로그래밍 모델은 Cloud Haskell이다. 이 모델은 액터(actor) 모델 기반의 병렬 프로그래밍 모델로 노드 간 메시지 전달은 소켓 통신을 통해 이루어진다[5]. 이 모델의 병렬 단위는 프로세스이며, 이 때문에 지정한 수 만큼 프로세스가 실행되는 것이 특징이다.

마지막 병렬 프로그래밍 모델은 IVC이다. IVC는 HaLVM에서 제공하는 병렬 모델로 3.1절에서 언급한 Rendezvous를 사용하는 모델이다. 이 모델의 병렬 단위는 가상 기계로 볼 수 있다. IVC는 Xen의 dom0에 설치된 프로그램을 통해 데이터를 주고받는 것이 다른 모델과 차이점으로 볼 수 있다.

위와 같이 살펴본 5종의 병렬 프로그래밍 모델을 요약하고 HaLVM에서 실행 검증을 한 결과는 표 1과 같다.

표 1 HaLVM 실행 분석 결과

병렬 모델	병렬 단위	실행 여부	확장성
Eval 모나드	스파크	O	없음
forkIO	그린 스레드	O	없음
forkOS	OS 스레드	X	-
Cloud Haskell	프로세스	X	-
IVC	VM	O	있음

표 1의 결과는 HaLVM에서 Haskell 병렬 프로그래밍 모델을 분석한 결과이다. 이 논문에서는 분석을 위해 각 병렬 프로그래밍 모델에 맞게 피보나치 수를 구하는 프로그램을 작성한 뒤 HaLVM에서 이를 실행하였다. 그리고 프로그램 실행 시 사용되는 코어를 늘려갈 때 실행 속도가 빨라지는 경우 확장성이 있는 것으로 판단하였다. 우선 결과를 살펴보면 IVC를 제외한 모든 병렬 프로그래밍 모델이 확장성이 없는 것으로 나타났다. IVC 프로그램은 Cloud Haskell과 유사한 메시지 전달 스타일로 작성하였으며, 실행 결과 확장성이 확보되는 것으로 나타났다.

확장성이 없는 병렬 프로그래밍 모델의 원인을 분석해보면 스레드나 프로세스 간 통신 문제로 파악된다. Eval 모나드 프로그램의 경우 병렬 단위인 spark는 정상적으로 생성되는 것을 확인할 수 있다. 하지만 생성 이후 실행 시간의 변화가 없는 것으로 나타났다. forkIO 역시 비슷하게 Task는 생성되지만, 실행 시간의 변화가 없는 것으로 나타났다. 반면에 실행이 되지

않는 forkOS는 실행 후 프로그램이 정지되는 문제가 발생하였다. 마지막으로 Cloud Haskell의 경우 소켓 통신의 문제로 프로그램 실행 단계에서 오류가 발생하는 것으로 나타났다. 이 문제는 Cloud Haskell에서 사용하는 유니캐스트(unicast) 통신 문제로 HaLVM에서 제공하는 네트워크 스택에서 유니캐스트를 지원하지 않아 발생한 것이다.

4. 실험

이 장에서는 앞선 분석 결과에서 확장성이 있다고 나타난 병렬 프로그래밍 모델인 IVC를 통해 프로그램을 실행한 결과와 기존 리눅스 환경에서 병렬 프로그램을 실행한 결과를 비교하고자 한다. 이 논문에서는 실험을 위해 39의 피보나치 수를 10번 구하는 프로그램을 기준으로 실험을 진행하였다. 그리고 실험 환경은 인텔 KNL 64코어 단일 노드 환경에서 진행하였으며, 실험에 사용된 OS는 Fedora 22와 HaLVM 2.4.0이다. 컴파일러는 GHC 8.0.2 버전을 사용하였다. 두 환경에서 프로그램을 실행한 뒤 확장성을 측정한 결과는 그림 2와 같다.

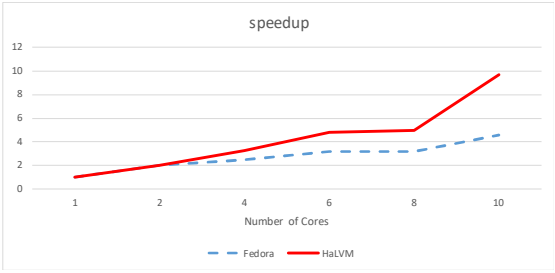

그림 2 리눅스와 HaLVM 환경의 확장성 비교

그림 2는 리눅스 환경에서 실행한 병렬 프로그램의 확장성과 HaLVM에서 실행한 병렬 프로그램의 확장성을 비교한 결과이다. 리눅스 환경에서는 HaLVM과 최대한 유사한 병렬 성능을 테스트하기 위해 forkOS를 병렬 모델로 사용하였으며, HaLVM은 IVC를 병렬 모델로 사용하였다.

실험 결과를 살펴보면 확장성은 HaLVM이 코어가 늘어남에 따라 계속 증가하는 것을 알 수 있다. 코어가 2개인 경우 두 환경의 확장성 차이는 0.69%로 HaLVM이 더 높은 것을 알 수 있다. 하지만 코어가 10개까지 증가한 경우 확장성은 112.42% 차이가 나는 것을 확인할 수 있다. 이는 코어가 늘어남에 따라 발생하는 문맥 전환(context switch)이나 커널/유저 모드 전환이 일어나지 않기 때문으로 판단된다. 즉 코어가 많은 매니코어 환경에서는 HaLVM이 유리한 것을 알 수 있다.

5. 고찰

이 장에서는 본 논문에서 병렬 프로그래밍 간 성능을 비교하기 위한 지표로 확장성을 선택한 이유와 3장에서 언급한 HaLVM에서 기존 병렬 프로그래밍 모델의 실행 문제에 대해 논의하고자 한다. 우선 비교 지표로 확장성을 선택한 이유는 리눅스 환경과 HaLVM 환경의 초기 실행 시간이 크게 차이가 나기 때문이다. 단일 코어 환경에서 리눅스의 실행 시간은 28.8초로 측정되었지만 HaLVM의 실행 시간은 877.6초로 나타났다.

이는 가상 환경과 기계 상에서 직접 돌아가는 환경을 감안하여도 큰 차이라고 볼 수 있다. 그렇기 때문에 실행 시간을 기준으로 두 운영체제를 비교하는 것은 무리가 있다. 이 문제는 Xen의 문제일 수도 있고 HaLVM의 문제일 수도 있다. 하지만 이 논문에서는 유니커널이 Haskell과 만났을 때 매니코어 환경에서 확장성을 보일 수 있는지 확인하는 것이 목적이기 때문에 확장성을 비교 지표로 선택한 것이다.

또한 HaLVM에서 대부분의 Haskell 병렬 프로그래밍 모델이 잘 동작하지 않는 문제가 있다. 문제가 있는 4종의 모델 중 스레드 단위의 병렬 프로그래밍 모델은 우선 설계상 의도된 것인지 아닌지 확인이 필요하다. 유니커널은 기본적으로 단일 프로세스가 동작하는 환경이지만 스레드를 제공하지 않는 것은 아니다. 하지만 HaLVM은 밑바닥부터 설계된 운영체제이기 때문에 스레드를 제공하지 않거나 막아놓았을 가능성도 있다. 그렇기 때문에 소스 코드를 확인할 필요가 있다. Cloud Haskell의 경우 유니커널 통신의 문제이기 때문에 후단부 통신 채널을 확장성이 확보되는 IVC로 변경하면 쉽게 해결할 수 있다.

6. 결론

이 논문에서는 매니코어 환경에서 유니커널을 사용한 경우 Haskell 병렬 프로그래밍의 성능이 확보될 수 있는지에 대한 분석을 진행하였다. 이를 위해 이 논문에서는 HaLVM을 대상 유니커널로 선택하였으며, 총 5종의 병렬 프로그래밍 모델을 대상으로 실행 가능성 및 확장성 확보 가능성을 분석하였다.

분석 결과 HaLVM에서는 IVC 이외에는 확장성이 확보되지 않는 것으로 나타났다. 그리고 확장성이 확보된 HaLVM의 IVC와 기존 리눅스 커널을 사용한 Haskell 병렬 프로그램의 확장성을 비교하는 실험을 진행하였다. 실험 결과 코어가 늘어남에 따라 HaLVM의 확장성이 리눅스보다 우수한 것으로 나타났다.

향후 연구로는 고찰에서 언급한 HaLVM에서 문제가 있는 병렬 프로그래밍 모델을 개선할 수 있는 방법을 연구하고자 한다. 이를 위해 기존 GHC와 HaLVM의 RTS(runtime system) 코드를 비교하는 연구를 진행하고자 한다. 그리고 현재 단순한 응용을 벗어나 시스템 호출이 많은 예제를 구현하여 유니커널과 리눅스의 성능 차이를 좀 더 자세히 분석하고자 한다.

ACKNOWLEDGMENT

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (2014-3-00035,매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터)).
*교신 저자 : 우균(부산대학교, woogyun@pusan.ac.kr).

참 고 문 헌

[1] A. Madhavapeddy and D. J. Scott, “Unikernels: Rise of the virtual library operating system,” *Queue - Distributed Computing*, Vol. 11, Issue 11, pp. 1-30, 2013.

[2] 차승준, 전승협, 람닉, 김진미, 정연정, 정성인, “유니커널의 동향과 매니코어 시스템에 적용,” *전자통신동향분석*, 제33권, 제6호, pp. 129-138, 2018.

[3] Adam Wick, “The HaLVM: A Simple Platform for Simple Platforms,” XenSummit, 2012.

[4] acw, “The Haskell Lightweight Virtual Machine (HaLVM): GHC running on Xen,” https://github.com/GaloisInc/HaLVM, (last checked 2018.12.03.).

[5] J. Epstein, A. P. Black and S. Peyton-Jones, “Towards Haskell in the cloud,” *ACM SIGPLAN Notices*, Vol. 46. No. 12. pp. 118-129, 2011.