Haskell에서 배열과 리스트의 병렬화 성능 비교

김연어[O1], 변석우[2], 우균[3*]
[1]부산대학교 전기전자컴퓨터공학과, [2]경성대학교 컴퓨터공학과, [3]부산대학교 전기컴퓨터공학부
yeoneo@pusan.ac.kr, swbyun@ks.ac.kr, woogyun@pusan.ac.kr

A Comparison of the Performance of the Parallelizations of Array and List in Haskell

Yeoneo Kim[O1], Sugwoo Byun[2], Gyun Woo[3*]
[1]Department of Electrical and Computer Engineering, Pusan National University,
[2]School of Computer Science & Engineering, Kyungsung University,
[3]School of Electrical and Computer Engineering, Pusan National University

요 약

현재 IT 환경은 업무용 PC에서 스마트폰까지 멀티코어 환경이 기본으로 탑재되고 있지만, 멀티코어 환경에 적합한 프로그램을 기존 명령형 언어로 개발하는 것은 어려운 일이다. Haskell은 순수 함수형 언어이기 때문에 명령형 언어보다 병렬 처리에 효과적인 특징이 있다. 이 논문에서는 Haskell을 자료 구조 중 리스트와 배열의 병렬 처리 방법에 대해 알아본다. 그리고 간단한 리스트와 배열 프로그램을 병렬화하여 성능을 비교한다. 두 자료 구조를 비교해본 결과 배열이 리스트보다 병렬화하기에 더 간단하며 속도 상승도 약 59% 높은 것으로 나타났다.

1. 서 론

최근 컴퓨터 환경은 단일 코어의 성능을 올리기보다 멀티코어를 이용하여 성능을 올리는 방향으로 발전하고 있다. 특히 이러한 추세는 서버나 가정용 PC에 그치지 않고 스마트폰 환경에서도 멀티코어를 지원하고 있다. 이러한 시장 변화 덕분에 멀티코어를 효과적으로 사용하는 병렬화 기술이 IT 산업에서 중요한 기술로 떠오르고 있다. 이에 발맞춰 프로그램 개발 시 자동으로 병렬화를 적용하는 방법에 관한 연구가 활발히 이루어지고 있다.

하지만 코어 수에 대응하여 프로그램을 병렬화하는 것은 어려운 문제이며, 기존의 개발된 프로그램을 병렬화가 가능하도록 구조를 변경하는 작업은 더 어려운 문제이다. 이는 현재 많은 프로그램이 명령형 언어를 이용하여 작성되었기 때문에 발생하는 문제로 볼 수 있다. 명령형 언어로 작성된 프로그램을 병렬로 처리하기 위해서는 작게는 문장에서부터 크게는 모듈 단위의 의존성을 분석해야 프로그램 본래의 의미를 바꾸지 않고 병렬화를 할 수 있다. 명령형 언어에서 병렬화 작업은 컴파일 시 분석 기법을 이용하여 일부분을 자동으로 병렬화하거나 프로그램 설계단계에서 병렬화 가능한 모듈로 작성하여야 병렬화가 가능하므로 어려움이 따른다. 이러한 명령형 언어의 문제점 때문에 함수형 언어인 Haskell이 멀티코어 환경에서 프로그래밍 대안으로 제시되고 있다.

Haskell은 순수한 함수형 언어를 위해 제안된 언어이다[1]. Haskell에서 함수는 부수효과(side-effect)가 없어서 언제 계산을 하여도 같은 값을 보장한다. 이러한 특징 때문에 Haskell은 병렬처리에서 명령형 언어보다 효과적이라 볼 수 있다.

이 논문에서는 Haskell의 리스트(list)와 배열(array)의 병렬처리 방법에 대해 알아보고 두 자료 구조의 성능을 비교하고자 한다. Haskell은 기본적으로 강력한 리스트 기능을 제공하여 프로그램 작성의 편의성을 높이고 있다. 또한, 리스트 이외에도 인덱스로 접근할 수 있는 배열을 라이브러리를 통해 제공하고 있다. Haskell에서 연속된 자료를 표현하기 위해서 리스트 혹은 배열을 사용하는데 병렬 처리 환경에서 이 두 자료 구조가 어떻게 사용되는지를 알아보고 성능을 비교함으로써 병렬 처리에 적합한 자료 구조를 판단하고자 한다.

이 논문의 다음과 같이 구성되어 있다. 2장에서는 관련 연구로 명령형 언어의 병렬 처리 방법과 Haskell의 병렬 처리 방법에 대해 살펴본다. 3장에서는 Haskell에서 리스트와 배열의 병렬화 방법에 대해 살펴본다. 그리고 4장에서는 실험을 통해 두 자료구조의 병렬화 성능을 비교한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 명령형 언어 병렬화 처리 기법

기존 명령형 언어에서는 병렬화 처리 기법으로 널리 사용되는 방법으로 자동 병렬화 기법이 있다. 자동 병렬화 기법은 컴파일 시 정적 분석(static analysis)을 통해 문장이나 함수간의 의존성 정보나 반복문 내부의 의존성 정보를 분석하여 분리 가능한 내용을 병렬화하는 방법이다[2]. 이 방법은 많은 반복이 일어나는 반복문 병렬화 기법으로 널리 이용되고 있다. 이외에도 정적 분석과 더불어 프로파일링이나 런타임 정보를 분석하여 자주 사용되는 파트의 병렬화를 높이는 예측적 자동 병렬화 기법(speculative automatic parallelism)도 제안된 바 있다[2].

2.2 Haskell 병렬화 처리 기법

Haskell의 병렬화 방법은 다른 언어에 비해 간단한 편이다. Haskell은 순수한 함수형 언어의 특성 때문에 계산 순서와 무관하게 연산을 할 수 있다. 이러한 특징은 각 함수를 병렬로 처리할 수 있어 병렬화 처리에서 다른 언

어보다 유리한 장점이 된다. 이 때문에 Haskell에서는 다양한 병렬화 처리 기법이 연구되고 있다[3, 4].

Haskell에서 대표적인 병렬화 처리 기법으로 희소 행렬(sparse matrix)과 같은 반복된 데이터가 나타나는 구조에서 병렬 처리를 위한 라이브러리인 Data Parallel Haskell이 있다. 이 라이브러리는 Haskell 컴파일러인 GHC(Glasow Haskell Compiler)를 통해 제공되고 있다. 이외에도 리스트나 배열을 병렬화하는 방법에서부터 GPU를 병렬화하는 라이브러리까지 다양한 병렬화 라이브러리를 지원하고 있다[3, 4]. 이러한 라이브러리는 기존 프로그램을 크게 변경하지 않고 map과 같은 고차원 함수(higher-order function)의 변경이나 병렬화 자료 구조로 교체하는 것만으로 손쉽게 병렬화를 수행할 수 있다.

3. 리스트와 배열의 병렬 처리

Haskell은 같은 타입의 데이터로 이루어진 자료 구조를 표현하기 위해서 기본적으로 리스트와 라이브러리를 이용한 배열을 사용할 수 있다. 또한, Haskell에서 리스트나 배열은 연속된 데이터를 한 번에 처리하기 위해 map과 같은 고차원 함수를 이용하여 계산하는 경우가 많다. 이러한 특징 때문에 Haskell에서는 고차원 함수를 변경하여 병렬화를 수행할 수 있다. 이 장에서는 Haskell에서 간단한 리스트와 배열 프로그램을 어떻게 병렬 처리가 가능한 프로그램으로 변경하는지를 보이고자 한다.

3.1 리스트

리스트는 Haskell에서 가장 널리 사용되는 기본적인 자료 구조이다. 여러 데이터 타입의 집합을 표현하거나 연속된 데이터를 사용할 때 주로 이용된다. 또한, 수학에서 사용되는 조건제시형 리스트(list comprehension) 기능을 이용하여 리스트 내용 선언도 지원하고 있다. 리스트를 사용하는 프로그램의 예는 그림 1의 코드와 같다.

```
fibo :: Int -> Int
fibo 0 = 0
fibo 1 = 1
fibo n = fibo (n-1) + fibo (n-2)
main = do let list = [1..40]
          print (map fibo list)
```

(그림 1) 리스트를 사용하는 Haskell 코드.

그림 1은 1에서 40 사이인 값을 가지는 리스트를 생성하고 리스트의 각 값의 피보나치 수를 구하는 프로그램이다. 이 프로그램은 `map` 함수를 이용하여 `list`에 `fibo` 함수를 적용하고 있다. 즉 `map`이 병렬화가 된다면 간단하게 병렬화를 할 수 있는 프로그램이다. 그림 2는 그림 1의 피보나치 프로그램을 병렬화한 프로그램이다.

```
import Control.Parallel.Strategies
main = do let list = [1..40]
          let result = parMap rseq fibo list
          print result
```

(그림 2) 그림 1의 병렬화 프로그램.

그림 2는 그림 1에서 `map` 함수를 병렬 map 함수인 `parMap` 함수로 변경한 프로그램이다. `parMap` 함수는 기존 `map` 함수와 달리 첫 번째 인자로 병렬 처리를 어떻게 수행할 것인지에 대한 전략(strategy)을 추가로 받게 된다. `parMap`은 넘겨받은 전략에 따라 병렬 처리 방법을 결정하게 된다.

3.2 배열

배열은 인덱스로 접근 가능한 자료 구조를 표현하는 타입이다. 배열은 리스트와 달리 자료의 집합 중 특정 원소 접근을 상수 시간에 할 수 있기 때문에 자료 접근이 빈번하거나 크기가 큰 자료를 표현하기에 적합하다. 또한, Haskell에서 배열은 값의 변경 여부와 원소의 표현 방법등 다양한 조건별로 적합한 배열을 제공하고 있다. 특히 Repa 패키지에서 제공하는 배열이 병렬화에 적합한 것으로 알려졌다[4, 5]. 그림 3은 Repa 배열을 사용하는 프로그램의 예이다.

```
import Data.Array.Repa as Repa
import Control.Monad
main = do
          let array1 = fromFunction
          (Z:.1000)(\(Z:.i) -> i :: Int)
          let result = computeS(Repa.map (+1)
          array1) :: Array U DIM1 Int
          forM_ [1..1000] (\i -> print
          (result ! (Z:.i)))
```

(그림 3) 배열을 사용하는 Haskell 코드.

그림 3은 1에서 1000 사이의 값을 가지는 배열을 생성하고 해당 배열에 모두 1씩 가산해준다. 그리고 가산된 결과를 인덱스 접근을 이용해서 출력하는 프로그램이다. Repa에서 사용되는 `map`은 결과를 반환한 시점에서 평가가 완료되지 않았기 때문에 `computeS` 함수를 통해서 결과를 평가해야 한다. 그리고 그림 4는 그림3의 Haskell 코드를 병렬화한 프로그램이다.

```
import Data.Array.Repa as Repa
import Control.Monad
main = do
          let array1 = fromFunction
          (Z:.1000)(\(Z:.i) -> i :: Int)
          result <- computeP(Repa.map (+1)
          array1) :: IO(Array U DIM1 Int)
          forM_ [1..1000] (\i -> print
          (result ! (Z:.i)))
```

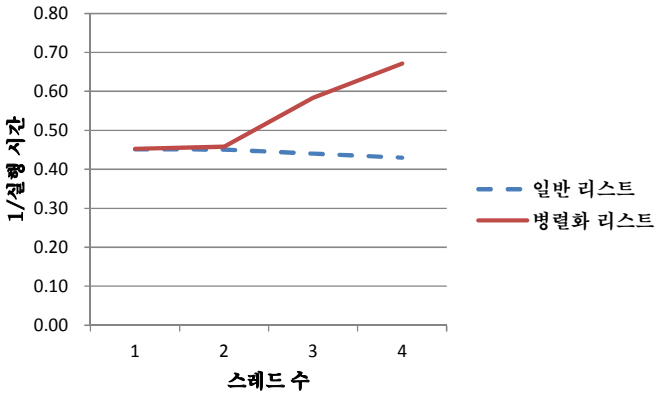
(그림 4) 그림 2의 병렬화 프로그램.

그림 4는 그림 3의 프로그램을 병렬화한 프로그램이다. Repa의 배열은 `map` 함수 수행 이후 실제 값을 평가 함수인 `computeS`를 사용한다. Repa의 배열은 `computeS` 함수를 병렬로 평가하는 함수인 `computeP` 함수로 변경하는 것만으로 병렬화를 할 수 있다. 그리고

computeP는 computeS와 달리 IO 모나드를 반환하는 함수이기 때문에 이에 대한 처리를 해주어야 한다.

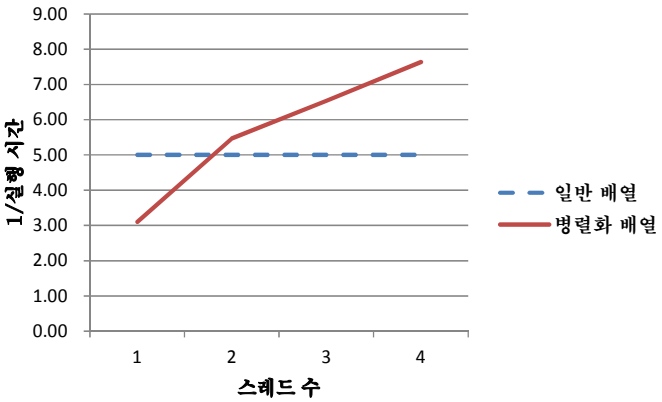
4. 실험 및 평가

이번 장에서는 3장에서 보인 병렬화 프로그램을 대상으로 실행 시간을 측정하여 병렬화 성능을 알아보고자 한다. 실험은 CPU Core i5 2.4GHz, Ram 8GB, SSD 256GB, OS는 OS X Yosemite 10.1.0.2 환경에서 실험을 진행하였다. 그리고 각 프로그램은 스레드 수에 따라 10번을 실행하여 측정된 결과의 평균값을 이용하였다. 그림 5는 리스트 프로그램을 실행한 결과이다.



(그림 5) 리스트 병렬화 성능 비교.

그림 5의 실험 결과 병렬화 이전 리스트는 스레드 수가 늘어남에 따라 미세하게 성능이 떨어지는 것을 확인할 수 있다. 그리고 병렬화 시 스레드가 2개 이상이 되면 성능 올라가는 것을 확인할 수 있다. 또한, 리스트를 병렬화한 결과 속도 상승은 약 1.48이 올라간 것을 확인할 수 있다. 그리고 그림 6은 행렬 프로그램을 실행한 결과이다.



(그림 6) 배열 병렬화 성능 비교

그림 6의 실험 결과 병렬화 이전 배열에서는 스레드의 수를 늘려도 실행 시간에 변화가 없는 것으로 나타났다. 그리고 computeP를 이용하여 병렬화를 한 배열은 성능이 스레드 수와 비례하여 올라가는 것을 확인할 수 있다. 그 결과 배열을 병렬화한 결과 속도 상승은 약 2.47이 올라간 것을 확인할 수 있다.

실험 결과 배열이 리스트보다 병렬화하기에 더 적합한 것으로 나타났다. 병렬화 과정만을 고려하여도 배열이 리스트보다 더 쉽게 변환할 수 있다. 또한, 병렬화 성능 역시 배열의 속도 상승이 2.47로 나타났기 때문에 리스트의 1.48보다 약 59% 더 효과적이다.

5. 결 론

이 논문에서는 Haskell에서 리스트와 배열이 어떻게 병렬화를 하는지 알아보았다. Haskell에서 리스트와 배열의 병렬화는 고차원 함수나 평가 함수를 병렬화 함수로 변경하는 것으로 지원할 수 있으며, 이를 간단한 Haskell 프로그램을 통하여 보였다. 그리고 병렬화된 리스트와 배열 프로그램의 스레드 수에 따른 실행 시간을 측정한 결과 배열이 리스트보다 속도 면에서 병렬화에 적합한 것으로 나타났다.

향후 연구에서는 리스트 병렬화 시 스레드 수가 2개인 경우 속도 상승이 거의 없는 것으로 나타나는 현상에 관해 연구를 진행할 예정이다. 이외에도 Repa 배열 이외의 다른 배열에 대해서도 성능을 평가해서 각 배열 별 병렬화 성능 차이를 분석해볼 예정이다. 그리고 실제 사용되는 응용 프로그램에 병렬화 기능을 추가하여 성능 평가를 진행하여 Haskell이 병렬화에 유리한 것을 보일 예정이다.

ACKNOWLEDGMENT

본 연구는 미래창조과학부의 SW컴퓨팅산업원천기술개발사업의 일환으로 수행하였음(B0101-15-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구).

*교신 저자 : 우균(부산대학교, woogyun@pusan.ac.kr).

참 고 문 헌

[1] M. Lipovaca, *Learn you a haskell for great good!: a beginner's guide*, No Starch Press, 2011.

[2] 심경주, 김한준, "자동 병렬화 기술 동향," *정보과학회지*, 제32권, 제5호, pp. 77 - 81, 2014.

[3] M. M. T. Chakravarty, R. Leshchinskiy, S. P. Jones, G. Keller and S. Marlow, "Data Parallel Haskell: a status report," *Proc. of the 2007 workshop on Declarative aspects of multicore programming*, pp. 10 - 18, 2007.

[4] S. Marlow, *Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming*, O'Reilly Media, 2013.

[5] G. Keller, M. M. T. Chakravarty, R. Leshchinskiy, S. P. Jones and B. Lippmeier, "Regular, shape-polymorphic, parallel arrays in haskell," *ACM Sigplan Notices*, Vol. 45. No. 9. pp. 261-272, 2010.