멀티코어 환경에서 병렬 Haskell 프로그램의 GC 성능을 측정하는 방법

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

An Approach for Measuring GC Performance of Haskell Program in Multi-core Environment

Hwamok Kim[O1], Sugwoo Byun[2], Gyun Woo[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

매니 코어 기반의 병렬 프로그래밍에 관한 관심이 증가함에 따라 관련 연구가 활발히 이루어지고 있다. 순수 함수형 언어인 Haskell은 부수효과가 없어서 병렬화 측면에서 다른 언어들에 비해 유리하다. 병렬화의 목적은 사용하는 코어 개수에 맞게 프로그램 실행속도를 증가시키는 것이다. ghc-gc-tune은 최적의 메모리 환경을 출력해주는 튜닝 프로그램이다. 하지만 기존 튜닝 도구는 싱글코어 기반으로 작성된 프로그램이기 때문에 매니코어 환경에서는 한 번의 실행으로 원하는 결과를 출력해주지 않는다. 따라서 본 논문에서는 기존의 ghc-gc-tune을 개선하여 매니코어 환경에서 프로그래머가 구현한 병렬 프로그램의 최적화된 실행환경을 알려주는 도구를 제안한다. 그 결과 코어 개수에 맞는 최적의 결과를 출력해주는 것을 확인할 수 있었다. 그리고 실험결과 코어에 따른 최적의 메모리 환경은 다른 것을 확인할 수 있었다.

1. 서 론

하드웨어의 발전으로 매니코어 기반의 프로세스가 늘어나고 있다. 매니코어는 수십 개 이상의 코어를 하나의 CPU에 집적하여 성능을 구현한 것이다. 이러한 매니코어를 활용하기 위해 프로그램을 병렬화하는 연구가 활발히 진행되고 있다[1].

병렬화는 프로그램 연산 부분을 여러 개로 나누어 각각에 대한 연산을 여러 코어에 동시에 수행하게 하는 것이다. 병렬화의 장점으로는 프로그램의 실행시간을 단축시킬 수 있고 해결할 수 있는 문제의 규모를 키울 수 있다는 점을 들 수 있다. 하지만 동시에 이뤄지는 일을 처음부터 병렬 알고리즘으로 설계하는 것은 일반적으로 힘든 일이다. 이 때문에 대부분 순차 프로그램을 병렬화하는 방법을 채택하고 있다. 따라서 개발자는 프로그램을 병렬화하기 위해 시스템의 구조에 맞게 최적화된 병렬 프로그램을 작성하는 기술이 필요하다[2].

순수 함수형 언어(pure functional language)인 Haskell은 강력한 병렬화 도구를 지원한다. 그래서 기존의 작성된 일반 코드에 병렬화를 적용하는 일이 명령형 언어에 비해 유리하다[3]. 하지만 Haskell로 작성된 병렬 프로그램은 상태에 따라 부분적으로 병렬화 여부가 결정된다. 그러므로 항상 전체 프로그램이 병렬화가 이루어지지 않을 수도 있다. 이 때문에 코어 개수를 늘려가며 실행했을 때 실행속도가 항상 증가하는 것은 아니다.

ghc-gc-tune은 컴파일된 Haskell 프로그램에 대하여 여러 다양한 GC 플래그를 자동으로 조정하여 실행한 후 실행 시간을 보여줌으로써 최적의 메모리 환경을 확인할 수 있도록 하는 프로그램이다[4]. GC 플래그는 가비지컬렉션(garbage collection) 메모리와 힙(heap) 메모리 크기를 결정하는 옵션이다. 개발자는 출력된 결과를 확인해서 시스템 환경에 맞는 최적의 환경을 구성할 수 있다.

기존의 ghc-gc-tune은 싱글코어 기반으로 작성된 프로그램이다. 따라서 매니코어 기반의 병렬 프로그램은 한 번의 실행으로 원하는 결과를 얻기 어렵다. 따라서 이 논문에서는 매니코어 기반의 ghc-gc-tune를 제안하고 코어 개수에 맞게 최적의 환경을 출력하는지 알아보고자 한다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 순수 함수형 언어인 Haskell과 ghc-gc-tune에 대해서 간단히 설명한다. 3장은 본 논문에서 제시한 개선된 ghc-gc-tune을 소개하며 4장에서는 실험 및 성능 분석을 한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 순수 함수형 언어 Haskell

순수 함수형 언어인 Haskell은 프로그램 병렬화와 관련하여 다음과 같은 특징이 있다. 먼저 대입문(assignment)과 변수를 사용하지 않는다. 따라서 함수는 항상 같은 값을 가지게 되며 함수의 값이 변수에 의해 달라지는 명령형 프로그램과 달리 부수효과(side-effect)가 없다. 명령형 프로그램이 많은 코어를 사용하기 위해서는 프로그램을 복잡하게 만들어야 하는데 비해 순수 함수형 언어인 Haskell은 부수 효과가 없는 장점을 이용하여 간결하고 쉽게 많은 코어를 사용할 수 있다[5-7].

2.2 ghc-gc-tune

Haskell은 가비지컬렉션을 이용하는 가상 머신 위에서 동작한다[8]. 이러한 실행 환경은 대상 프로그램의 메모리 사용량에 따라 전체 실행 시간의 변동이 심한 특성이 있다. GHC에서는 이러한 문제를 해결하기 위해 메모리 튜너인 ghc-gc-tune을 제공하고 있다. ghc-gc-tune은 입력된 프로그램을 대상으로 두 메모리 옵션을 조정해가며 실행하여 최적의 실행 시간이 나오는 메모리 옵션을 찾아내며 그 결과를 그래프로도 표기하여 보여준다.

3. ghc-gc-tune을 이용한 최적의 코어환경 분석 구현

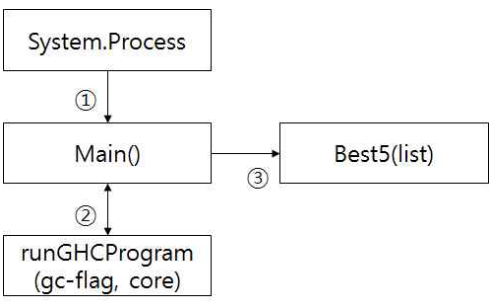
기존 ghc-gc-tune의 한 번의 동작으로 하나의 코어 환경의 결과만 얻을 수 있다. 하지만 그 결과가 모든 코어에서 최적의 옵션이라고 보장할 수 없다. 그러므로 이 논문에서는 한 번의 실행으로 모든 코어 환경에서의 최적의 메모리 옵션을 찾아내고자 한다. 그림 1은 기존 ghc-gc-tune을 실행한 결과이다.

```
Best settings for Running time:
26.89s:    +RTS -A67108864 -H1048576
27.10s:    +RTS -A134217728 -H2097152
27.17s:    +RTS -A16384 -H134217728
27.20s:    +RTS -A134217728 -H4194304
27.20s:    +RTS -A2097152 -H67108864
```

(그림 1) 기존의 ghc-gc-tune 실행 결과

그림 1은 Haskell로 만들어진 피보나치 수열의 40번째 항을 구하는 프로그램을 ghc-gc-tune으로 실행한 결과이다. 그림 1의 왼쪽 부분은 프로그램 실행시간을 출력한다. +RTS는 프로그램 실행을 위해 실행시간시스템(runtime system)의 명령어를 사용했다는 뜻이며, -A는 가비지컬렉션에 사용되는 메모리 범위를 알려준다. 마지막 -H는 가비지컬렉터에 사용된 힙 메모리의 크기를 알려준다. 기존 도구는 한 번의 실행으로 하나의 코어에 대한 출력결과를 보여주기 때문에 매니코어 환경에서 전체 결과를 실험하기에 적합하지 않다.

본 장에서는 매니코어 환경에서 병렬 Haskell 프로그램의 최적 실행환경을 출력하도록 개선하고자 한다. 코어별 환경에서의 실행시간을 오름차순으로 정렬한 뒤 가장 실행시간이 짧은 5개의 실행옵션 출력을 목표로 한다. 개선된 ghc-gc-tune 프로그램은 그림 2와 같이 크게 세 가지 동작으로 이루어져 있다.



(그림 2) 개선된 ghc-gc-tune 구조

매니코어 환경의 ghc-gc-tune을 구현하기 위해서는 사용될 코어의 정보가 필요하다. 따라서 그림 2-①에서는 Linux 시스템 명령어를 사용할 수 있는 라이브러리를 추가해서 시스템 코어 정보를 받아온다. 받아온 코어의 정보는 문자열 형태로 저장된다. 하지만 문자열 형태의 데이터는 정수형 데이터와 비교해 main() 함수에서 반복문 및 제어문에서 사용하기 적절하지 않으므로 정수형 데이터로 변환해야 한다. 그림 2-②에서는 2-①에서 정수형으로 변환된 코어의 값이 runGHCProgram() 함수를 통해 기존에 자동으로 정의되어 함께 넘겨받은 GC 플래그 옵션들과 함께 문자열 형태로 변환된다. 그리고 실제 Linux 터미널 창에서 RTS 옵션을 입력하여 실행하는 것과 같은 동작을 해서 코어별 실행시간을 계산한다. 계산된 값은 리스트로 저장되어 Main() 함수로 돌아간다. 마지막으로 그림 2-③에서는 best5() 함수를 통해 실행시간을 오름차순으로 정렬해 코어별로 가장 빠른 5개의 환경을 출력한다.

4. 실험 및 성능 분석

본 장에서는 매니코어 환경에 적합하게 개선한 ghc-gc-tune을 통해 두 가지 실험을 하고자 한다. 첫 번째 실험은 코어별로 계산된 결과를 출력하는 실험을 하고자 한다. 출력된 결과를 바탕으로 두 번째 실험은 가장 빠른 실행 시간의 코어 환경을 찾고 이를 검증하고자 한다.

실험은 Ubuntu 14.04.3, 개선된 ghc-gc-tune-0.3, CPU Opteron 6272 2개, 32GB RAM에서 이루어졌다. 그리고 실험에 사용할 프로그램은 피보나치 수열의 40번째 항을 구하는 프로그램이다. 그림 3은 첫 번째 실험을 실행한 결과이다.

```
Best settings for Running time:
26.89s:    +RTS -A67108864 -H1048576 -C1
27.10s:    +RTS -A134217728 -H2097152 -C1
27.17s:    +RTS -A16384 -H134217728 -C1
27.20s:    +RTS -A134217728 -H4194304 -C1
27.20s:    +RTS -A2097152 -H67108864 -C1
...
Best settings for Running time:
3.80s:    +RTS -A1048576 -H33554432 -C15
4.13s:    +RTS -A2097152 -H8388608 -C15
4.34s:    +RTS -A1048576 -H1048576 -C15
4.44s:    +RTS -A2097152 -H33554432 -C15
4.49s:    +RTS -A16384 -H134217728 -C15
...
Best settings for Running time:
4.39s:    +RTS -A2097152 -H33554432 -C32
4.62s:    +RTS -A1048576 -H4194304 -C32
4.95s:    +RTS -A1048576 -H8388608 -C32
4.99s:    +RTS -A1048576 -H33554432 -C32
5.02s:    +RTS -A1048576 -H16777216 -C32
```

(그림 3) 개선된 ghc-gc-tune 실행 결과.

그림 3과 같이 사용된 코어별 실행 시간을 오름차순으로 정렬해 가장 빠른 5개의 환경을 알려준다. -C는 새로

부여한 코어에 관한 정보이며 숫자는 사용한 코어의 개수이다. 실험결과 15개의 코어를 사용했을 때 가장 빠른 실행속도를 내는 것을 확인할 수 있었다. 표 1은 대표적으로 코어 1개, 32개 환경과 가장 빠른 실행속도를 보인 코어 15개의 환경을 표로 정리한 것이다.

(표 1) 코어별 실행 환경 비교표

코어	1개	15개	32개
실행시간(초)	26.89	3.80	4.39
-A(bit)	67108864	1048576	2097152
-H(bit)	1048576	33554432	33554432

표 1을 바탕으로 가장 빠른 코어 환경을 올바르게 출력했는지 검증하고자 한다. 1개와 32개 코어환경에서 최적의 성능을 보인 GC 플래그를 15개 코어에 적용했을 때 어떤 성능을 내는지 알아보고자 한다. 그림 4, 5는 두 번째 실험 결과이다.

```
./fibo +RTS -A67108864 -H1048576 -s -N15 -RTS

INIT time 0.03s (0.03s elapsed)
MUT time 168.57s (11.51s elapsed)
GC time 3.52s (0.24s elapsed)
EXIT time 0.07s (0.01s elapsed)
Total time 172.20s (11.80s elapsed)
```

(그림 4) 1개의 코어 환경에 15개 코어를 적용한 결과.

```
./fibo +RTS -A2097152 -H33554432 -s -N15 -RTS

INIT time 0.01s (0.01s elapsed)
MUT time 61.59s (4.15s elapsed)
GC time 8.97s (0.60s elapsed)
EXIT time 0.05s (0.01s elapsed)
Total time 70.63s (4.76s elapsed)
```

(그림 5) 32개의 코어 환경에 15개 코어를 적용한 결과.

그림 4, 5와 같이 표 1에서 확인한 3.80초보다 더 길어진 각각 11.80, 4.76초가 나온 것을 확인할 수 있다. 따라서 개선된 ghc-gc-tune에서는 코어별 최적의 환경을 출력해 주는 것으로 볼 수 있다. 출력된 결과에서 개발자의 판단 아래 적절한 환경을 선택하여 병렬처리 프로그램을 실행할 수 있다.

5. 결 론

본 논문에서는 기존의 싱글코어 기반의 GC-tuner를 매니코어 환경에 적합하게 개선했다. 개선된 GC-tuner는 병렬 Haskell 프로그램에서 최적 코어 환경을 찾아준다. 이를 간단한 실험으로 검증한 결과 개선된 GC-tuner는 최적 코어 환경을 정상적으로 출력하는 것을 알 수 있었다.

향후 연구에서는 병렬 Haskell 프로그램 실행 시 실행속도가 증가하다가 병목현상이 일어나는 부분을 찾아 출력하는 연구를 진행할 예정이다. 이외에도 GC 플래그 옵션 중 -A와 -H 옵션 사이의 상관관계를 분석할 예정이다. 그리고 실제로 사용되는 병렬 Haskell 응용 프로그램에 적용해 성능 평가를 진행할 예정이다.

ACKNOWLEDGMENT

본 연구는 미래창조과학부의 SW컴퓨팅산업원천기술개발사업의 일환으로 수행하였음(B0101-16-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구).
*교신 저자 : 우균(부산대학교, woogyun@pusan.ac.kr).

참고문헌

[1] 김진미, 변석우, 김강호, 정진환, 고광원, 차승준, 정성인, “매니코어 시대를 대비하는 Haskell 병렬 프로그래밍 동향,” 전자통신동향분석, 제29권 제5호, pp.167-175, 2014.

[2] A. Grama, *Introduction to parallel computing*, Pearson Education, 2003.

[3] M.KH. Aswad, P.W. Trinder and H.W. Loidl, “Architecture Aware Parallel Programming in Glasgow Parallel Haskell (GPH),” ICCS, Procedia Computer Science, vol. 9, pp.1807-1816, 2012.

[4] D. Stewart, ghc-gc-tunes, [Online].Available:http//hackage.haskell.org/package/ghc-gc-tune. (downloaded 2016. Apr. 26)

[5] J. Hughes, “Why functional programming matters,” The computer journal Vol. 32, No.2, pp.98-107, 1989.

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

[7] M. S. Aljabri, *GUMSMP: a scalable parallel Haskell implementation*, Diss. University of Glasgow, pp.56-61, 2015.

[8] P. M. Sansom and S. L. Peyton Jones, "Generational garbage collection for Haskell," Proceedings of the conference on Functional programming languages and computer architecture, pp. 106-116, 1993.