The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 21, No. 6, pp.141-145, Dec. 31, 2021. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2021.21.6.141
JIIBC 2021-6-21

차세대 저전력 멀티뱅크 메모리를 위한 컴파일러 최적화 기법

Compiler Optimization Techniques for The Next Generation Low Power Multibank Memory

조두산*

Doosan Cho*

요 약 다양한 형태의 메모리 아키텍처가 개발되었고, 이를 효과적으로 사용하기 위한 여러 컴파일러 최적화 기법이 연구되었다. 특히, 모바일 컴퓨팅 디바이스에서 메모리는 성능을 결정하는 주요 컴포넌트이기 때문에 이를 지원하기 위한 다양한 최적화 기법들이 개발되었다. 최근에는 하이브리드 형태의 메모리 아키텍처에 대한 연구가 많이 진행되고 있기 때문에 이를 지원하기 위한 다양한 컴파일러 기법이 연구되고 있다. 시장의 요구조건에 맞추어 저전력에 대한 제약조건과 필요한 최소한의 성능을 달성하기 위하여 기존의 컴파일러 최적화 기법들이 사용될 수 있다. 이러한 최적화 기법들을 활용한 저전력 효과 및 성능 개선 정도를 파악하기 위한 레퍼런스가 제대로 제공되지 못하고 있는 실정이다. 본 연구는 기존의 컴파일러 기법에 대한 실험 결과를 멀티뱅크 메모리 아키텍처 개발의 레퍼런스로 제공하기 위하여 진행되었다.

Abstract Various types of memory architectures have been developed, and various compiler optimization techniques have been studied to efficiently use them. In particular, since a memory is a major component that determines performance in mobile computing devices, various optimization techniques have been developed to support them. Recently, a lot of research on hybrid type memory architecture is being conducted, so various compiler techniques are being studied to support it. Existing compiler optimization techniques can be used to achieve the required minimum performance and constraint on low power according to market requirements. References for determining the low-power effect and the degree of performance improvement using these optimization techniques are not properly provided yet. This study was conducted to provide the experimental results of the existing compiler technique as a reference for the development of multibank memory architecture.

Key Words : Compiler, Instruction code, Low power, Memory, Optimization, Program

*정회원, 순천대학교 전자공학과
접수일자 2021년 7월 12일, 수정완료 2021년 11월 10일
게재확정일자 2021년 12월 10일

Received: 12 July, 2021 / Revised: 10 November, 2021 / Accepted: 10 December, 2021
*Corresponding Author: dscho@scnu.ac.kr
Dept. of Electrical & Electronic Engineering, Sunchon National University, Korea

Ⅰ. 서 론

클라우드 컴퓨팅, 에지 컴퓨팅, 웨어러블 컴퓨팅 등 모바일 디바이스 시장이 성장하면서 반도체 시장 또한 급속히 성장하고 있다. 이렇게 임베디드 시스템으로 구성되는 모바일 디바이스는 성능과 가격의 방정식으로 단가가 결정된다. 성능은 비메모리 반도체와 메모리 반도체의 선택에 따라 상당 부분 결정된다. 일반적으로 고성능 설계의 높은 요구사항에 따라 가격이 상승한다. 제품 개발은 디바이스가 사용되는 응용 분야에서 요구되는 성능에 맞추어 설계를 한다. 일반적으로 개발 단가가 어느 정도 결정되면 하드웨어만으로 요구되는 스펙의 성능에 도달하기 어렵기 때문에 소프트웨어 최적화가 갈수록 중요해지고 있다.

소프트웨어 최적화는 일반적으로 컴파일러의 최적화 모듈을 이용하여 적용된다. 하드웨어의 성능을 최적으로 사용하기 위한 프로그램 코드를 제공하는 것이 바로 컴파일러의 목적이기 때문이다. 이러한 컴파일러에는 다양한 최적화 기법이 적용되어 있다. 최근 많이 사용되는 마이크로소프트 비쥬얼 스튜디오 컴파일러, GCC 혹은 LLVM과 같은 컴파일러 툴은 다양한 하드웨어 아키텍처를 위한 여러 방면의 최적화 기법을 내재하고 있기 때문에 이를 적극적으로 사용한다면 최적의 성능을 이끌어낼 수 있게 된다.

본 연구에서는 다양한 형태의 임베디드 아키텍처를 위하여 사용 가능한 컴파일러 기법에 대하여 정리하고, 그 성능을 GCC 컴파일러 프레임워크를 이용하여 평가해보도록 한다. 다음장에서 사용 가능한 최적화 기법에 대하여 정리하고, 3장에서는 메모리 성능개선을 위하여 사용 가능한 최적화 기법에 대하여 정리하겠다. 마지막장에서는 이러한 최적화 기법을 적용하였을 때 얻을 수 있는 결과에 대하여 기술하겠다.

Ⅱ. GCC 컴파일러 기법

1. 최적화 기법의 분류

GCC 컴파일러는 -O0, -O1, -O2, -O3과 같이 레벨에 따라 최적화 기법을 분류하여 적용하고 있다. 최적화 레벨 O0은 최적화 기법을 적용하지 않기 때문에 프로그램 코드 그대로의 성능을 확인할 수 있다. O1은 최소한의 최적화 기법을 사용한다. 따라서 최소한의 성능 개선을 기대할 수 있다. 이때 적용되는 대표적인 기법은 표 1과 같다.

Delayed-branch는 분기 전 지연시간에 분기에 독립적인 명령을 수행할 수 있도록 delay slot에 독립적인 명령어를 배치하는 최적화 기법이다. forward-propagate은 불필요한 복사 명령어를 제거하고, 복사 대상이 사용되는 지점에 해당 값을 대입함으로서, 성능향상을 얻는 최적화 기법이다. if-conversion은 하드웨어에서 predication을 지원하면 분기문을 없애고 모든 명령어를 predication으로 변환하는 것으로 성능을 개선하는 기법이다.

표 1. O1 레벨 최적화 기법
Table 1. O1 level optimization technique

delayed-branch
forward-propagate
if-conversion
merge-constants
move-loop-invariants

merge-constant는 동일하게 사용되는 상수를 병합한다. move-loop-invariant는 루프 코드에 영향을 주지않아서 루프 코드 외부로 이동할 수 있는 코드를 루프 밖으로 이동하는 코드를 의미한다. 최적화 레벨 O2는 loop 최적화 기법을 제외한 모든 최적화 기법이 적용되는 것을 의미한다. 대표적인 최적화 기법은 표 2와 같다. code-hoisting은 loop invariant code motion의 다른 호칭이다. 루프 명령어에서 독립적으로 실행될 수 있기 때문에 루프 위로 코드를 끌어올려 루프의 실행시간을 단축하는 최적화 기법이다. inline-function은 함수의 호출이 있는 위치에 호출 명령을 제거하고, 함수를 복사하여 붙이는 방식이다. 이렇게 함수의 호출을 제거하면 함수 호출에 의하여 발생하는 오버헤드를 제거할 수 있어 성능향상에 기여할 수 있게 된다.

표 2. O2 레벨 최적화 기법
Table 2. O2 level optimization technique

code-hoisting
inline-functions
peephole
schedule-insns
store-merging
tree-switch-conversion

peephole 최적화는 다양한 기본적인 최적화 기법을 포괄하여 지칭하는 기법이다. 예를 들면, redundant

load, store elimination 혹은 constant folding과 같은 기법들이 peephole 최적화에 포함된다. 그 외에도 strength reduction, null sequences, combine operation 등도 peephole에 포함되는 최적화 기법이다. 포함되는 기법들은 특정한 조건에 부합하는 특정 명령어들을 대상으로 하기 때문에 단일 기법으로는 큰 성능 개선을 얻기 어려우나 모든 기법들이 적용된 후에는 의미있는 개선을 얻을 수 있다. schedule-insns는 컴파일러에 있는 명령어 스케쥴러를 활성화하는 기법이다. 알고리즘에 따라 성능 개선 기준을 달성하도록 명령어 스케쥴링이 적용된다. store-merging은 중복된 store를 하나로 통합하는 기법이다. tree-switch-conversion는 switch문 안에서 변수들의 초기화를 정리해주는 최적화 기법이다. 최적화 레벨 O3는 다양한 루프 코드 최적화 기법을 포함하고 있는데 이러한 최적화 기법은 메모리 아키텍처를 위한 최적화로 다양하게 사용되기 때문에 다음장에서 자세히 기술하도록 하겠다.

Ⅲ. 메모리 아키텍처를 위한 컴파일러 최적화 기법

아래 표 3에 기술된 최적화 기법이 GCC 컴파일러의 최적화 옵션 O3에 해당하는 대표적인 기술들이다. loop interchange는 루프의 실행 순서를 변경하여 루프 안에서 사용되는 데이터의 사용 순서를 변경할 수 있다. 결과적으로 캐시 메모리와 같은 온칩 메모리의 성능을 개선시킬 수 있다.

표 3. O3 레벨 최적화 기법
Table 3. O3 level optimization technique

loop-interchange
loop-unroll-and-jam
peel-loops
split-loops
tree-loop-distribution
tree-loop-vectorize

unroll and jam은 루프 코드를 펼쳐서 루프의 반복 횟수를 줄이는 기법이다. 이 최적화 기법이 적용되면 명령어 단계 병렬성이 명확해지고 따라서 명령어 스케쥴러의 성능을 개선할 수 있다. peel-loop은 루프 코드안에서 unwinding과 같이 연산량을 줄일 수 있는 경우 이를 줄임으로서 성능 개선을 얻는 기법이다. split-loop과 loop distribution은 루프를 분리하여 데이터 지역성과 명령어 지역성을 개선시키는 최적화 기법이다. loop-vectorize는 컴파일러가 하드웨어에 있는 벡터 관련 컴포넌트를 파악하고 이를 이용하기 위한 벡터 명령어를 자동으로 생성하는 기능을 의미한다.

IoT 혹은 에지 컴퓨팅[1], 메디컬 메모리[2] 등을 위한 차세대 메모리는 멀티뱅크 메모리[3] 형태 혹은 지능형 캐시[4]와 같은 하이브리드 메모리 형태로 설계되고 있다. 이러한 메모리를 위한 최적의 코드와 데이터 생성을 위하여 머신러닝[1]과 같은 인공지능[6] 기법이 사용될 수 있다. 또한 스필데이터 절감 기법[7], 저가형 메모리를 위한 최적화 기법[8] 등을 LLVM[9]과 같은 컴파일러 프레임워크에 구현하거나 새로운 메모리를 위한 프로토타이핑 도구[10]을 개발할 수 있다. 데이터 대역폭이 특히 넓게 요구되는 멀티미디어 스트리밍 애플리케이션[11]과 이러한 데이터 스트리밍 애플리케이션에 특히 효과적으로 사용가능한 멀티뱅크 메모리[12]를 위한 컴파일러 기법이 사용되고 있다. 또한 소프트웨어 취약점 분석[13], 헬스케어 디바이스 취약점 분석[14], AI기반 IoT개발 환경 자동화 구현[15]에 컴파일러 기법이 필수적으로 요구된다.

```
/* Candidate for Loop Distribution */
for( i=0; i<n; i++ ){
     h[i] = j[i] / k[i] - u[i];          /*S1*/
     q[i+1] = (q[i-1] * q[i] - q[i+1])*10;   /*S2*/
     j[i] = k[i] * h[i];                      /*S3*/
}
```

그림 1. 루프 분할 예제 코드
Fig. 1. An example code of lopp distribution

멀티 뱅크 메모리에서 효과적으로 사용될 수 있는 기법들중에서 루프 분할을 예를 들어 살펴보겠다. 그림 1에 루프 분할을 적용할 예제 코드를 나타내었다. 루프 코드 S1에서 변수 h, j, k, u가 이터레이션 i에서 사용되었다. 루프 코드 S2는 변수 q가 이터레이션 i-1, i, i+1에서 사용되었다. 루프코드 S3에서 S1에서 사용된 변수 j, k, h가 이터레이션 i에서 다시 사용되었다.

루프 코드 S1과 S3는 사용된 변수 사이에 의존성이 있기 때문에 루프 분할 기법의 대상이 아니다. 루프 코드 S2는 S1/S3에서 사용된 변수 h, j, k, u가 사용되지 않았기 때문에 의존성이 없다. 즉, S2는 독립된 루프를 새롭게 구성할 수 있다. 이렇게 그림 1의 루프 코드에 루프 분할 기법이 적용된 결과가 그림 2에 나타나 있다. 루프

코드 L1에서 문장 S1과 S2가 계산되며, 루프 L2에서 문장 S2가 계산된다. 루프 L1과 L2는 수행하는 연산에서 상호 의존성이 없기 때문에 병렬 실행이 가능하다. 즉, 그림 1의 코드는 루프 한 개에 묶여 있기 때문에 프로세싱 유닛 한 개와 연결된 온칩 메모리에서 모든 연산이 처리된다. 그림 2의 코드는 L1과 L2가 각각 서로 다른 프로세싱 코어와 온칩 메모리 뱅크에서 병렬적으로 동시 계산이 가능하다. 그림 1 코드는 일반적인 디지털 프로세싱 칩에서 실행시간이 일반 연산기 1cycle, 곱셈 연산기 3cycle 소요된다고 가정하면, n이 100일 때 1900cycle이 소요되지만, 그림 2의 코드는 1000cycle에 완료할 수 있다. 멀티 코어 멀티 뱅크 환경에서 메모리 대역폭이 충분한 경우 47.3%의 성능개선이 가능하게 되는 것이다.

```
/* L1 */
for( i=0; i<n; i++ ){
        h[i] = j[i] / k[i] - u[i];                  /*S1*/
        j[i] = k[i] * h[i];              /*S3*/
}

/* L2 */
for( i=0; i<n; i++ ){
        q[i+1] = (q[i-1] * q[i] - q[i+1])*10;       /*S2*/
}
```

그림 2. 루프 분할 결과
Fig. 2. Result of loop distribution

Ⅳ. 실험 및 결론

1만개의 데이터를 정렬하는 퀵정렬/버블정렬 알고리즘에 G++ 컴파일러 버전 9.3.0을 사용하여 성능을 측정하였다. 최적화가 없는 O0을 기준으로 선택하여 분석을 하였다. 레벨 O2에서 성능이 29% 개선됨을 확인하였다. O3는 O2에 비하여 17% 성능이 저하됨을 확인하였다. 이러한 결과는 O2에서 기본적인 모든 최적화가 적용되기 때문에 발생하는 것으로 확인되었다.

표 4. 최적화 적용 실험 결과
Table 4. experimental results of applied optimizations

name/optimization level	O0 (sec)	O1 (sec)	O2 (sec)	O3 (sec)
Quick sort	0.001912	0.001300	0.001362	0.001595
Buble sort	0.065015	0.032387	0.000056	0.000057

O3의 경우 루프 코드를 특정 메모리에 적합하게 최적화하는 내용이 적용되기 때문에 메모리 아키텍처 정보가 컴파일러에 명확하게 기술되지 않은 경우 성능에 부정적인 결과가 발생하는 것을 알 수 있었다. 따라서 컴파일러가 메모리 구조에 최적의 코드를 생성하기 위해서는 메모리 구조에 대한 명확한 기술 정보가 컴파일러에 제공되어야 한다.

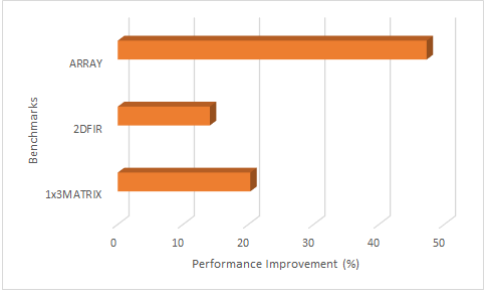

그림 3. 성능 개선 결과
Fig. 3. Performance Improvement Result

명확한 메모리 구조가 컴파일러에 전달된 경우, 프로세싱 코어 4개와 연결된 온칩 메모리 뱅크로 구성된 환경에서 loop-unroll-and-jam, loop pipelining 기법들을 1x3 크기 행렬 곱 연산 코드에 적용하였을 때 20.3%의 성능 개선을 얻을 수 있었고, 동일한 기법들을 2차원 FIR 필터 코드에 적용하였을 때 14.1% 개선됨을 확인할 수 있었다. 프로세싱 코어 2개와 직접 연결된 온칩 메모리 뱅크로 구성된 환경에서 loop distribution 기법을 배열 연산 코드에 적용하였을 때 47.3%의 성능 개선이 가능하였다. 실험 결과를 그림 3에 정리하였다.

본 연구를 통하여 컴파일러에 포함된 최적화 기법들에 대하여 살펴보았으며, 이를 활용하여 얻을 수 있는 성능 개선 효과들을 파악할 수 있었다. 효과적인 최적화 기법들이 기존 컴파일러에 있었으며, 이러한 기법들을 보다 효과적으로 활용하기 위해서는, 타겟 하드웨어 구조에 대한 정보를 컴파일러에 명확히 기술해야 함을 확인할 수 있었다.

References

[1] D. Cho, "A Study on Improvement of Low-power Memory Architecture in IoT/edge Computing", Journal of the Korean Society of Industry Convergence, Vol. 24, No. 1, pp. 69-77, Feb. 2021.

DOI: https://doi.org/10.21289/KSIC.2021.24.1.69

[2] J. Cho, J. Lee, D. Cho, "Efficient memory design for medical database", Basic & Clinical Pharmacology & Toxicology, Vol. 125, pp. 198, July. 2019.

[3] J. Cho, JM Youn, D. Cho, "An Automatic Array Distribution Technique for Multi-Bank Memory of High Performance IoT Systems", World, Vol. 3, No. 1, pp. 15-20, Nov. 2019. DOI: https://gvpress.com/journals/WJWDE/vol3_no1/vol3_no1_2019_03.html

[4] D. Cho, "A Smart Cache Lockdown Technique for IoT System", Journal of Physics: Conference Series, Vol. 1927, No. 1, pp. 012003, May. 2021. DOI:https://doi.org/10.1088/1742-6596/1927/1/012003

[5] D. Cho, "Study on Memory Performance Improvement based on Machine Learning", The Journal of the Convergence on Culture Technology, Vol. 7, No. 1, pp. 615-619, Feb. 2021. DOI: https://doi.org/10.17703/JCCT.2021.7.1.615

[6] D. Cho, "Memory Design for Artificial Intelligence", International Journal of Internet, Broadcasting and Communication, Vol. 12, No. 1, pp. 90-94, Dec. 2020. DOI: https://doi.org/10.7236/IJIBC.2020.12.1.90

[7] J. Yoon, D. Cho, "A spill data aware memory assignment technique for improving power consumption of multimedia memory systems", Multimedia Tools and Applications, Vol. 78, No. 5, pp. 5463-5478, Mar. 2019. DOI: https://doi.org/10.1007/s11042-018-6783-x

[8] J. Cho,, J. Lee, D. Cho, "Low-End Memory Subsystem Optimization Process", International Journal of Smart Home, Vol. 13, No. 2, pp. 11-16, Oct. 2019. DOI: http://dx.doi.org/10.21742/ijsh.2019.13.2.02

[9] J. Cho, D. Cho,, Y. Kim "Study on LLVM application in Parallel Computing System", The Journal of the Convergence on Culture Technology, Vol. 5, No. 1, pp. 395-399, Feb. 2019. DOI: http://dx.doi.org/10.17703/JCCT.2019.5.1.395

[10] J. Cho, D. Cho, "Development of a Prototyping Tool for New Memory Subsystem", International Journal of Internet, Broadcasting and Communication, Vol. 11, No. 1, pp. 69-74, Jan. 2019. DOI: http://dx.doi.org/10.7236/IJIBC.2019.11.1.69

[11] J. Yoon, D. Cho, "Improving memory system performance for multimedia applications", Multimedia Tools and Applications, Vol. 76, No. 4, pp. 5951-5963, 2017. DOI: https://doi.org/10.1007/s11042-015-2807-y

[12] J. Cho, J. Yoon and D. Cho, "Improvement Energy Efficiency for a Hybrid Multibank Memory in Energy Critical Applications", Technical gazette, vol.27, no. 6, pp. 1946-1955, 2020. DOI: https://doi.org/10.17559/TV-20200826040830

[13] K. Lee, J. Park, "A Software Vulnerability Analysis System using Learning for Source Code Weakness History", Journal of the Korea Academia-Industrial, Vol. 18, No. 11 pp. 46-52, 2017 DOI: http://dx.doi.org/10.5762/KAIS.2017.18.11.46

[14] H. Jeon, S. Lee, "Analysis of Remote Update Vulnerabilities of IoT Healthcare Devices", The Journal of Korean Institute of Information Technology, Vol. 19, No. 1, pp. 87-97, Jan. 31, 2021 DOI: https://doi.org/10.14801/jkiit.2021.19.1.87

[15] S. Kim, Y. Yun, S. Eun, S. Cha, J. Jung, "Implementation of Autonomous IoT Integrated Development Environment based on AI Component Abstract Model", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC) Vol. 21, No. 5, pp.71-77, Oct. 31, 2021. DOI: https://doi.org/10.7236/JIIBC.2021.21.5.71

저 자 소 개

Doosan Cho(정회원)

- 2010~current : Sunchon National Univ. EE. Professor
- 2019~2021 : Georgia Tech. Visiting Scholar
- Research Area : Low Power Memory, Optimizations, AI, IoT

※ This paper was supported by Sunchon National University Research Fund in 2019.(Grant number: 2019-0005)