The Journal of the Convergence on Culture Technology (JCCT)
Vol. 5, No. 1, pp.395-399, February 28, 2019. pISSN 2384-0358, eISSN 2384-0366

http://dx.doi.org/10.17703/JCCT.2019.5.1.395
JCCT 2019-2-50

병렬 컴퓨팅 시스템에서 LLVM 응용 연구

Study on LLVM application in Parallel Computing System

조중석*, 조두산**, 김용연***

Jungseok Cho*, Doosan Cho, Yongyeon Kim*****

요 약 다양한 병렬 컴퓨팅 시스템을 지원하기 위해서는 LLVM IR을 벡터/행렬을 보다 효과적으로 지원할 수 있도록 확장하는 것과 LLVM IR을 machine code로 바꾸어 주는 부분을 새로운 알고리즘으로 설계하여 구현하면 된다. IR 예제에서 보았듯이 기본적으로 RISC 명령어로 구성되어 있기 때문에 RISC 명령어 생성은 자연스럽게 생성되며, 벡터 또한 현재 지원가능한데 행렬 명령어는 지원되지 못하고 있다. 벡터/행렬을 보다 강력하게 지원하기 위한 새로운 IR 구조, 명령어 생성 알고리즘 및 관련 부분의 확장이 필요하다. 이를 위해 LLVM IR의 각 명령어를 (벡터/행렬을 위한) target architecture의 적당한 명령어로 mapping을 해주는 부분 (instruction selection 알고리즘)이 중요하다. LLVM IR 명령어의 의미를 파악하고, target architecture의 각 명령어 의미와 syntax를 비교하여, 패턴이 일치하는 명령어를 선택하여 mapping을 효율적으로 해줘야 한다.

주요어 : 병렬 컴퓨팅, 범용그래픽프로세서, 멀티코어, 매니코어 시스템, 컴파일러, LLVM, 코드 생성

Abstract In order to support various parallel computing systems, it is necessary to extend LLVM IR to more efficiently support vector / matrix and to design LLVM IR to machine code as a new algorithm. As shown in the IR example, RISC instruction generation is naturally generated because the RISC instruction is basically composed of the RISC instruction, and the vector instruction is also not supported. There is a need for new IR structures, command generation algorithms and related extensions to support vector / matrix more robustly. To do this, it is important to map each instruction in the LLVM IR to the appropriate instruction in the target architecture (vector / matrix) (instruction selection algorithm). It is necessary to understand the meaning of LLVM IR command, to compare the meaning of each instruction of the target architecture with syntax, and to select the instruction that matches the pattern to make mapping efficient.

Key words : Parallel computing, GPGPU, multicore, manycore system, compiler, LLVM, code generation

Ⅰ. 서 론

.LLVM [1]이라는 이름은 약자가 아닌 컴파일러 플랫폼 브랜드로 “엘엘브이엠”으로 불린다. LLVM은 LLVM 관련 프로젝트, LLVM 중간 표현 (LL), LLVM 디버거, LLVM의 C ++ 표준 라이브러리 구현에 적용되는 브랜드이다. LLVM은 프로그래밍 언어로 작성된 프로그램의 컴파일 타임, 링크 타임 및 런타임 최적화를

*정회원, 국립순천대학교 전기전자공학부 (제1저자)
**정회원, 국립순천대학교 전기전자공학부 (교신저자)
***정회원, ETRI 연구소 (참여저자)
접수일: 2018년 10월 7일, 수정완료일: 2018년 11월 13일
게재확정일: 2018년 12월 일

Received: October 07, 2018 / Revised: November 13, 2018
Accepted: December 26, 2018
*Corresponding Author: dscho@scnu.ac.kr
Dept. of EE, Sunchon National Univ, Korea

위해 설계된 매우 강력한 컴파일러 인프라 프레임워크(framework)이다. LLVM은 여러 플랫폼에서 작동하며, 주된 목적은 빠르게 실행되는 코드를 생성하는 것이다. LLVM은 기본적으로 중간언어표현 (IR) 생성기가 핵심인데, 이것을 사용하면 원하는 언어를 연결할 수 있는 프론트엔드와 함께 컴파일러 개발 전체 플로우 [2] (프론트 엔드 분석기 + IR 생성기 + LLVM 백엔드)로 커스텀 컴파일러 생성이 단순화된다. LLVM은 프론트엔드(다른 명칭은 프로그램 언어 파서)에서 중간표현 (IR) 코드를 가져와서 최적화된 IR을 생성하면서 완전한 컴파일러 시스템의 중간코드를 제공할 수 있다. 그런 다음 이 새로운 IR을 변환하여 대상 플랫폼(intel, AMD, Mips 등)에 대한 기계 종속 어셈블리 언어 코드를 생성할 수 있다. LLVM은 GNU 컴파일러 컬렉션 (GCC 프론트엔드) 툴체인으로부터 IR을 받아들이므로(혹은 CLANG [3]) 이 프로젝트를 위해 작성된 다양한 현존 컴파일러와 함께 사용할 수 있다. LLVM은 컴파일 타임이나 링크 타임 또는 실행 시 이진 머신 코드에서도 재배치 가능한 머신 코드를 생성할 수 있다. 아래 코드는 LLVM IR 예제를 나타낸다. 아래 IR 예제코드에서 볼 수 있듯 LLVM IR은 저수준의 RISC와 같은 가상 명령어 집합이다. 실제 RISC와 같이 더하기, 빼기, 비교, 분기 등의 간단한 명령어들을 지원한다. 이 명령들은 3 어드레스 형식인데, 이는 명령어가 몇 개의 입력을 받고, 입력이 아닌 다른 레지스터에 결과를 저장한다는 것을 의미한다. LLVM IR은 레이블도 지원하고, 일반적으로는 좀 이상한 형태의 어셈블리 언어처럼 보인다. 대부분의 RISC 명령어 집합과 다른 점으로는, LLVM IR은 간단한 타입 시스템을 가진 강 타입 언어이고, 실제 기계의 상세한 부분을 추상화시켜 버렸다는 부분을 들 수 있다. 예를 들어 call, ret와 명시적인 인자 전달을 통하여 호출 규약(calling convention)을 추상화시켜 버렸다. 기계어와 또 다른 중요한 차이점은 레지스터의 수이다. LLVM IR 은 임시로 이름을 붙일 수 있는 무한한 수의 레지스터를 가질 수 있다. 임시 이름은 %를 앞에 붙임으로써 생성할 수 있다.

```
define i32 @add1(i32 %a, i32 %b) {
entry:
  %tmp1 = add i32 %a, %b
  ret i32 %tmp1
}

define i32 @add2(i32 %a, i32 %b) {
entry:
  %tmp1 = icmp eq i32 %a, 0
  br i1 %mp1, label %done, label %recurse
recurse:
  %tmp2 = sub i32 %a, 1
  %tmp3 = add i32 %b, 1
  %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
  ret i32 %tmp4
done:
  ret i32 %b
}
```

위의 LLVM IR로 작성된 두 함수는 각각 아래의 C 코드를 표현하는 것으로, 두 정수를 더하는 서로 다른 방법이다.

```
unsigned add1(unsigned a, unsigned b) {
  return a+b;
}
unsigned add2(unsigned a, unsigned b) {
  if (a==0) return b;
  return add2(a-1, b+1);
}
```

II. Intermediate Representation (IR)

LLVM IR은 언어 독립적인 명령어 세트 및 타입 시스템을 지원한다. 위 예제 코드에서 보았듯이 각 명령어는 정적 단일 지정 양식 (Static Single Assignment [4])으로 생성된다. 즉, 데이터 유형이 지정된 레지스터라고 하는 각 변수가 한 번 할당된 다음 고정된다. 이는 변수 간의 종속성 분석을 단순화하는데 도움이 된다. LLVM을 사용하면 기존 GCC 시스템에서처럼 정적으로 코드를 컴파일하거나 JIT (Just-In-Time Compilation)를 통해 IR에서 머신 코드로 늦게 컴파일 할 수 있다. 데이터 타입 시스템은 정수 또는 부동 소수점 수와 같은 기본 유형과 포인터, 배열, 벡터, 구조 및 함수와 같은 5개의 파생 유형

으로 구성된다. C ++의 클래스는 구조체, 함수 및 함수 포인터의 배열을 혼합하여 나타낼 수 있다. LLVM의 핵심은 어셈블리와 유사한 저수준 프로그래밍 언어인 중간 표현 (IR)이다. IR은 대상의 세부 정보를 추상화하는 강력한 형식의 RISC (reduced instruction set computing) 명령 집합이다. 예를 들어, 호출 규칙은 명시적 인수를 사용하여 call 및 ret 명령을 통해 추상화 된다. 또한 고정된 레지스터 집합 대신에 IR은 %0, %1 등의 형식의 임시 집합을 사용한다. LLVM은 인간이 읽을 수 있는 어셈블리 형식인 3주소형식 IR을 지원한다. 컴파일러는 보통 C/C++/Java 등 프로그래밍 언어로 짠 코드를 target architecture에 독립적인 중간 표현(IR: Intermediate Representation)으로 바꾸는 부분과 IR을 target architecture의 machine code로 바꾸는 부분으로 나뉜다. 다양한 병렬 컴퓨팅 시스템을 지원하기 위해서는 LLVM IR을 벡터/행렬을 보다 효과적으로 지원할 수 있도록 확장하는 것과 LLVM IR을 machine code로 바꾸어 주는 부분을 새로운 알고리즘으로 설계하여 구현하면 된다. IR 예제에서 보았듯이 기본적으로 RISC 명령어로 구성되어 있기 때문에 RISC 명령어 생성은 자연스럽게 생성되며, 벡터 또한 현재 지원가능한데 행렬 명령어는 지원되지 못하고 있다. 벡터/행렬을 보다 강력하게 지원하기 위한 새로운 IR 구조, 명령어 생성 알고리즘 및 관련 부분의 확장이 필요하다.

이를 위해 LLVM IR의 각 명령어를 (벡터/행렬을 위한) target architecture의 적당한 명령어로 mapping을 해주는 부분 (instruction selection 알고리즘)이 중요하다. LLVM IR 명령어의 의미를 파악하고, target architecture의 각 명령어 의미와 syntax를 비교하여, 패턴이 일치하는 명령어를 선택하여 mapping을 효율적으로 해줘야 한다.

컴파일러의 관점에서 어려운 문제는 데이터의 배치와 명령어 생성을 모두 컴파일러가 하게 되는데, 해당 두 문제가 모두 NP-complete [5]이면서, 두 문제가 독립적인 문제가 아니라 복합적인 문제이기 때문에 동시에 해결해야 최적 솔루션을 찾을 수 있다는 점이다. 따라서 병렬 컴퓨팅 시스템 컴파일러 플랫폼을 구성하기 위해서는 각각의 아키텍처에서 명령어 집합의 공통분모와 차이점을 분석하여 컴파일러 중간 표현 (IR)을 설계해야 하고, 효율적인 구조의 IR을 바탕으로 데이터 배치와 명령어 생성 알고리즘을 구성해야 한다. 대부분의 병렬 컴퓨터는 기본적으로 벡터/행렬 명령어와 유사한 형태의 명령어를 갖고 있다. 응용 프로그램의 데이터 병렬성을 처리하기 위하여 벡터 명령어 혹은 행렬 명령어 처리기를 보유하고 있고, 이를 고속으로 처리하기 위하여 멀티 UNIT으로 구현된다는 점이 병렬 컴퓨팅 시스템의 철학이다. LLVM IR은 다행히도 이 모든 정보를 담을 수 있도록 구성되어 있다. 심지어 벡터 명령어까지 지원가능한데 다만 행렬은 지원이 안된다.

LLVM IR 구조와 명령어 생성과정을 살펴보가 위하여 아래 예제를 살펴보자. LLVM에서 Clang -emit-llvm option을 이용하면 아래 소스코드가 다음 IR로 변환된다.

소스코드:

```
int foo(int aa, int bb, int cc) {
  int sum = aa + bb;
  return sum / cc;
}
```

LLVM의 중간표현 IR:

```
ls transforms into the LLVM IR: clang -cc1 foo.c -emit-llvm
define i32 @foo(i32 %aa, i32 %bb, i32 %cc) {
  entry:
    %add = add nsw i32 %aa, %bb
    %div = sdiv i32 %add, %cc
    ret i32 %div
  }
```

생성된 IR은 컴파일러의 명령어 생성기에 의하여 DAG (directed acyclic graph) 형태로 변환되어, 일종의 패턴 매칭 기법에 의하여 적합한 타겟 명령어로 최종 결정된다. 명령어 생성기의 패턴 매칭 알고리즘에 의하여 차후 명령어에 맞게 변환되는 과정을 거치게 된다. 그림에는 구성된 DAG을 이용하여 명령어가 선택되는 과정을 나타내고 있다. IR이 (fadd:f32 (fmul:f32 (fadd:f32 W, X), Y), Z)와 같이 구성되어 있다. 타겟 명령어 중에서 vloating point multiply-and-add (FMA)명령어가 있다면, DAG은 우측 DAG처럼 변환될 수 있다. 즉, IR (FMADDS (FADDS W, X), Y, Z)와 같이 되는 것이다.

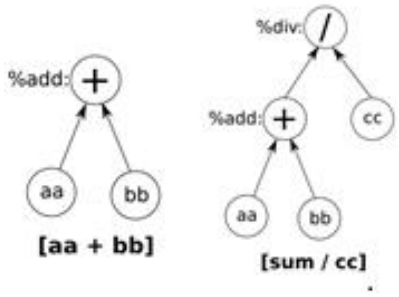
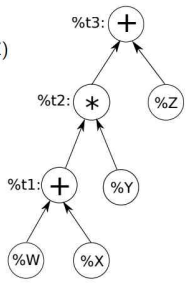


그림 1. DAG 예제
Figure 1. DAG example

또한 LLVM은 벡터 명령어 생성 또한 지원하고 있다. 아래 C코드가 IR로 다음과 같이 변환될 수 있다.

The previous IR code would be equivalent to the following DAG:
(fadd:f32 (fmul:f32 (fadd:f32 W, X), Y), Z)



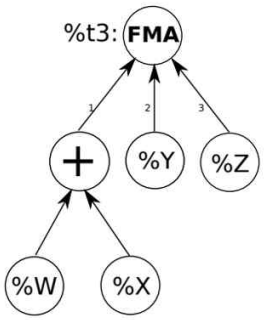


그림 2. 명령어 생성 예제
Figure 2. An example of instruction generation

C 코드 예제:

```
int a[8]; int b[8];
for (i=0; i < 8; i++)
    a[i] = a[i] + b[i];
```

LLVM IR 코드:

The LLVM IR supports vector data types natively:

```
%vec1 = load <8 x i32>* %addr1
%vec2 = load <8 x i32>* %addr2
%vec3 = fmul <8 x i32> %vec1 , %vec2
```

보시다시피 %vec1이 8개의 i32(정수 32비트) 데이터를 동시에 load하는 것을 나타내고 있다. 따라서 LLVM IR은 벡터 단위까지는 지원을 하고 있으며, 행렬은 IR에서 표현되지 못하는 상태이다. IR이 행렬 부분까지 확장될 수 있다면 행렬 명령어 생성지원이 가능하게 될 것이다.

기존의 RISC를 위한 컴파일러의 코드생성기, 데이터 최적화기, 스케쥴러 등은 벡터-행렬 곱과 같은 복합 연산 명령어 생성을 고려하여 설계되지 않았기 때문에 그것을 그대로 사용하는 것은 행렬 하드웨어 UNIT을 사용하지 않겠다는 것과 같은 말이다 (벡터 행렬 명령어 생성안됨). 게다가 VLIW 형식 (Myriad의 SHAVE) 프로세서는 스케쥴러가 성능의 핵심인데 데이터 배치 최적화, 명령어 생성, 스케쥴링까지 동시에 해결해야 하는 NP-complete X 3 문제를 해결해야 병렬 컴퓨팅 시스템을 위한 최적 코드를 생성할 수 있다. 따라서 LLVM IR이 행렬 부분까지 확장되는 것과 별도로 데이터 배치 최적화기, 명령어 생성기, 명령어 스케쥴링 알고리즘들이 다양한 병렬 컴퓨팅 시스템을 지원하는 벡터/행렬 명령어를 최적으로 생성할 수 있도록 새롭게 개발되어야 한다.

III. 결 론

병렬 컴퓨팅 시스템에서 소프트웨어를 효과적으로 실행시키려면 병렬 컴파일러 환경이 필요하다. 이것을 구현하려면 보통 LLVM과 같은 프레임워크를 이용하여 개발한다. 하지만 LLVM은 병렬 컴퓨터를 지원하기 위한 다양한 요소를 포함하지 못하고 있기 때문에 이때 필요한 요소들에 대한 고려와 함께 개발되어야 한다. 본 연구에서는 이에 대한 개발 포인트와 함께 기술개발 제안을 정리하였다. 이러한 연구 결과는 다양한 가상화 시스템 [6] 및 3D 영상 등의 병렬 컴퓨팅 환경에서 효과적으로 활용될 수 있을 것이다.

References

[1] The LLVM Compiler Infrastructure, online : https://llvm.org/
[2] Compiler Construction/Introduction, online : https://en.wikibooks.org/wiki/Compiler_Construction/Introduction
[3] Clang: a C language family frontend for LLVM,

online : https://clang.llvm.org/
[4] Static single assignment form, online : https://en.wikipedia.org/wiki/Static_single_assignment_form
[5] NP-completeness, online : https://en.wikipedia.org/wiki/NP-completeness
[6] Jong-Youel Park, Young-Hyun Chang, "Study on Arduino Kit VR contents modularization based on virtualization technology in software education field," The Journal of the Convergence on Culture Technology (JCCT), Vol. 4, No. 3, pp.293-298, August 31, 2018.
http://dx.doi.org/10.17703/JCCT.2018.4.3.293

※ This work was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF - 2018R1D1A1 B07050054).