재구성형 프로세서를 위한 아키텍처 명세 언어: SoarDL Extension for CGRA

양승준, 윤종희, 김용주, 백윤흥
서울대학교 공과대학 전기정보공학부
e-mail : sjyang, jhyoon, yjkim@sor.snu.ac.kr, ypaek@snu.ac.kr

Architecture Description Language for Reconfigurable Processors: SoarDL Extension for CGRA

Seungjun Yang, Jonghee Yoon, Yongjoo Kim, Yunheung Paek
Dept. of Electrical and Computer Engineering, Seoul National University

요 약

재구성형 프로세서는 높은 성능과 낮은 전력 소모, 재구성이 가능하다는 점에서 갈수록 높아지는 모바일 및 소형 전자기기 시장의 요구 조건을 충족시키기에 적합한 특성을 가지고 있다. 이 논문에서는 아키텍처 명세 언어인 SoarDL 언어를 확장하여 재구성형 프로세서를 효과적으로 기술할 수 있는 방법과 함께, 이를 바탕으로 재구성형 프로세서를 위한 컴파일러를 생성할 수 있는 방안을 제시한다.

1. 서론

지난 수 년간 스마트폰과 같은 모바일 기기 및 소형 전자기기 시장이 크게 성장하면서, 이들 기기에서 동작하는 응용(Application) 또한 매우 다양하고 복잡해졌다. 이에 따라 이전까지의 단일 프로세서 혹은 단일 ASIC 방식으로는 이와 같은 다양한 응용들의 요구 조건을 동시에 만족시키기가 매우 어려워졌다.

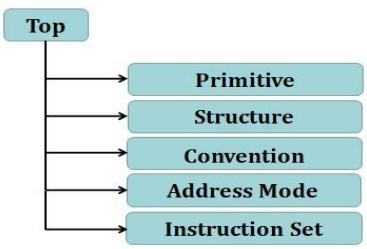
재구성형 프로세서(Coarse Grained Reconfigurable Architecture)는 높은 성능과 낮은 전력 소모, 그리고 재구성이 가능하다는 점에서 이러한 문제점을 해결하기에 매우 적합한 구조를 가지고 있다. 재구성형 프로세서는 일반 프로세서에 비해 적게는 열 배에서 많게는 수십 배에 달하는 연산 장치(Processing Element)를 가지고 있어 동시에 여러 개의 연산을 수행할 수 있으며, 이러한 연산 장치들을 configuration 메모리를 통해 재구성 할 수 있어 재구성이 불가능한 ASIC 에 비해 유연성이 높아 다양한 응용의 높은 요구 조건을 충족시키는 데 매우 유리하다.

이러한 재구성형 프로세서의 장점을 극대화하기 위해서는 이에 최적화된 컴파일러의 존재가 필수적이다. 예를 들어, 동작시키고자 하는 코드 내에 분기문이 존재할 경우 재구성형 프로세서 내부의 연산 장치에 할당하는 것이 매우 어려워질 뿐만 아니라 효율 또한 떨어지게 된다. 따라서 이러한 분기문에 대한 효율적인 처리를 해 줄 수 있는 컴파일러가 필수적이다. 하지만 재구성형 프로세서에 대한 지금까지의 연구는 컴파일러 보다는 프로세서 구조 자체에 대한 연구가 주를 이루었던 것이 사실이다.

본 논문에서는 재구성형 프로세서를 위한 컴파일러를 쉽고 빠르게 생성하기 위해, 아키텍처 명세 언어(Architecture Description Language)로 재구성형 프로세서의 특징 및 구조를 기술하고 이로부터 컴파일러를 생성할 수 있는 방안을 제시한다. 먼저 2 장에서는 아키텍처 명세 언어인 SoarDL 언어를 소개한다. 3 장에서는 재구성형 프로세서를 기술하기 위해 SoarDL 언어를 어떻게 확장하였는지 설명한다. 4 장에서는 실제 컴파일러 생성을 위해 필요한 향후 연구 방향을 설명하고, 마지막 5 장에서는 전체 내용을 요약하고 결론을 제시한다.

2. SoarDL 아키텍처 명세 언어

SoarDL 언어는 아키텍처 명세 언어로써 SoarGen 재겨냥성 컴파일러 플랫폼의 기반 언어이며, 프로세서의 구조 및 특성을 기술하도록 고안되었다. 개발자가 SoarDL 언어를 이용하여 개발하고자 하는 프로세서의 구조와 특성을 기술하면 SoarGen 은 이를 바탕으로 해당 프로세서를 위한 컴파일러를 자동으로 생성해준다.

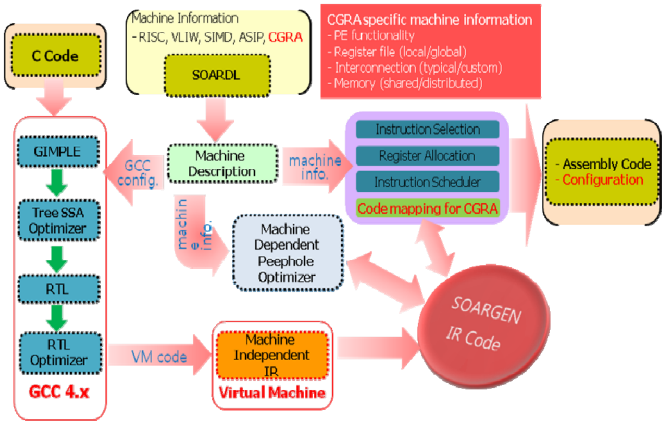


(그림 1) SoarDL 계층 구조

그림 1 에서 보는 것과 같이 SoarDL 언어는 크게 다섯 가지의 주 항목으로 나뉠 수 있으며, 각각의 항목은 프로세서의 특정한 구조 및 특징을 기술하도록 되어 있다. 예를 들어 Structure 항목에서는 프로세서의 메모리 구조나 파이프라인 구조 등을 기술할 수 있으며, Convention 항목에서는 프로세서가 함수 호출을 처리하는 방식을 기술한다. Address Mode 나 Instruction Set 항목의 경우 여러 개의 세부 항목을 가질 수 있으며, 각각의 세부 항목과 이를 포함하는 상위 항목은 계층 구조를 이루도록 고안되었다. 이는 다수의 명령어와 여러 개의 주소 모드를 가질 수 있는 프로세서의 특성을 간편하고 효과적으로 기술할 수 있도록 하기 위함이다. 각 항목 내에서 기술되는 프로세서의 정보는 형식론에 입각하여 기술되도록 되어 있으며, 이는 프로세서 개발자로 하여금 원하는 컴파일러를 쉽고 간편하게 얻을 수 있도록 한다.

3. 재구성형 프로세서를 위한 SoarDL 언어 확장

이전 장에서 설명한 SoarDL 언어는 일반적인 프로세서의 구조 및 특성을 쉽고 간편하게 기술하도록 고안되었으나, 재구성형 프로세서의 경우 일반적인 프로세서와 상이한 구조를 가지고 있기 때문에 기존의 SoarDL 언어로 이를 기술하는 데에는 무리가 있었다. 이러한 문제점을 해결하기 위해 SoarDL 언어에 재구성형 프로세서를 기술할 수 있는 항목을 추가하고, 이 항목에 재구성형 프로세서의 구조 및 특성을 기술하여 이를 바탕으로 SoarGen 컴파일러가 재구성형 프로세서를 위한 컴파일러를 생성할 수 있도록 하였다. 다음 그림은 재구성형 프로세서를 기술하기 위해 SoarDL 언어에 추가된 정보와 이를 기반으로 하는 SoarGen 컴파일러의 전체적인 구조를 나타낸 것이다.



(그림 2) CGRA 를 위한 SoarGen 컴파일러의 구조

재구성형 프로세서를 위한 컴파일러를 생성하기 위해서는 개별 연산 장치들의 특성 및 연결, 메모리 및 레지스터 파일의 구조 등에 대한 정보가 필요하다. 이러한 정보를 기술하기 위해 SoarDL 언어에 추가된 항목은 다음과 같다.

- 연산 장치 (Processing Element, PE)

재구성형 프로세서는 응용을 효과적으로 수행하기 위해 적게는 십 수개에서 많게는 수십 개의 연산 장치를 가지고 있다. 이러한 연산 장치들은 산술 연산 장치(Arithmetic Logic Unit)와 같이 다양한 연산들을 수행할 수 있으며, 레지스터 파일과 같은 저장 공간을 내부적으로 포함할 수 있다. 이전까지는 개별 연산 장치의 구조가 동일한 동종 재구성형 프로세서에 대한 연구가 주로 이루어졌으나, 최근에는 연산 장치의 구조 및 특성이 서로 상이한 이종 재구성형 프로세서에 대한 연구가 주를 이루고 있다. 동일한 구조의 연산 장치를 가지는 경우보다 서로 다른 구조의 연산 장치를 가지는 경우에 재구성형 프로세서의 성능이 보다 높기 때문이다.

이러한 이유로 아키텍처 명세 언어를 이용하여 고성능의 재구성형 프로세서를 위한 컴파일러를 생성하기 위해서는 개별 연산 장치의 서로 다른 특성을 각각 기술할 수 있어야 한다. 이를 위해 확장된 SoarDL 언어에서는 'processingElement' 라는 항목을 추가하여 개별 연산 장치의 특성을 기술할 수 있도록 하였다. 다음은 확장된 SoarDL 언어를 이용하여 연산 장치를 기술한 예이다.

```
processingElement PE0 : SoarRAConfig {
    register        r[4];
    register        output;
    functional_unit  {
        multiplier  { latency 4; };
        ld_st       { latency 1; };       };
    pipeline        { IF; ID; EX; WB;     };
    operation       { mul | ld | st };
}

processingElement PE1 : SoarRAConfig {
    register        r[4];
    register        output;
    functional_unit {
        alu         { latency 1; };       };
    pipeline        { IF; ID; EX; WB;     };
    operation       { alu };
}
```

(그림 3) 확장된 SoarDL 언어로 기술된 개별 PE

위의 예에서 두 연산 장치는 네 개의 레지스터를 가지는 레지스터 파일과 계산된 결과를 연산 장치 외부로 전달하기 위해 쓰이는 출력 레지스터 하나를 포함하고 있다. 파이프라인의 구조 또한 서로 동일하다. 반면, 실제적인 연산을 위해 포함하고 있는 장치의 종류는 서로 다르다. 첫 번째 연산 장치 PE0 의 경우 곱셈기 하나와 메모리 접근 유닛 하나를 가지고 있는데 반해 두 번째 연산 장치 PE1 의 경우 일반적인 산술 연산 장치 하나를 포함하고 있다. 수행할 수 있는 명령어의 종류 또한 PE0 가 곱셈 및 메모리 접근 명령어를 수행할 수 있는 반해 PE1 은 일반적인 산술

연산 명령어만을 수행할 수 있다. 이처럼 확장된 SoarDL 언어에서는 개별 연산 장치의 특성을 따로 기술할 수 있도록 하여 컴파일러 생성 시에 이러한 정보를 활용할 수 있도록 하였다.

- 연결 구조 및 전역 저장 공간 (Connectivity and Global Storage)

재구성형 프로세서를 위한 컴파일러를 생성하는 데 있어 개별 연산 장치에 대한 정보 못지 않게 중요한 것이 바로 연산 장치들의 연결 구조(Connectivity)이다. 연산 장치들이 어떤 방식으로 연결되어 있는지에 따라 주어진 코드를 각각의 연산 장치에 할당하는 방식이 달라질 수 있기 때문이다. 따라서 재구성형 프로세서를 위한 아키텍처 명세 언어는 일반적인 메시(mesh) 연결 구조에서부터 임의의 복잡한 연결 구조까지 효과적으로 기술할 수 있어야 한다.

재구성형 프로세서는 다수의 연산 장치가 공동으로 접근할 수 있는 전역 저장 공간을 가질 수 있다. 모든 연산 장치가 자유롭게 값을 쓰거나 읽을 수 있는 전역 레지스터 파일이나 전역 메모리가 이에 해당한다. 이와 같은 전역 저장 공간을 기술하기 위해서는 전역 저장 공간 자체에 대한 정보 이외에도 어떤 연산 장치들이 어떤 전역 저장 공간에 접근할 수 있는지 명시할 수 있어야 한다. 특히 최근의 재구성형 프로세서의 경우 전역 메모리가 뱅크(Bank)화 되어 메모리의 일정 주소 영역마다 접근할 수 있는 연산 장치가 다른 경우가 많기 때문에, 이러한 복잡한 구조를 기술할 수 있는 효과적인 방법이 반드시 필요하다.

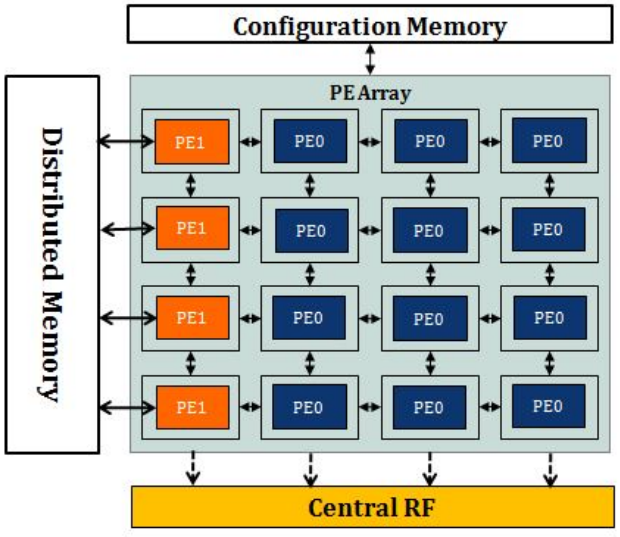
이러한 연결 구조 및 전역 저장 공간을 효과적으로 기술하기 위해, 연결 구조를 위한 새로운 항목을 SoarDL 언어에 추가하는 동시에 저장 공간을 기술하던 기존의 방식을 확장하여 저장 공간에 접근할 수 있는 연산 장치를 명시할 수 있도록 하였다. 다음은 이렇게 확장된 SoarDL 언어를 이용하여 연산 장치의 연결 구조 및 전역 저장 공간을 기술한 예이다.

```
CGRA SoarRAConfig : SoarRA {
    memory    byte     ra_mem {
        latency 1 @ [0x000000..0x7fffff];
    } read_port(8) write_port(4);

    register    word    central_RF[32]<0..15>;
    register    bit     pred_rf[32]<0..15>;

    PEarray             PEs[4 * 4] = {
        PE1, PE0[3],
        PE1, PE0[3],
        PE1, PE0[3],
        PE1, PE0[3]  };

    connectivity        mesh ;
}
```

(그림 4) 연결 구조 및 전역 저장 공간 기술의 예

위의 예에서 기술된 재구성형 프로세서는 한 개의 전역 메모리와 두 개의 전역 레지스터 파일을 가지고 있다. 먼저 메모리를 위한 기술에서는 해당 기술이 메모리를 위한 기술임을 표시하는 지시어 'memory'와 메모리에 저장된 데이터의 기본 단위 'byte', 그리고 메모리의 이름 'ra_mem' 이 기술되어 있다. 또한 메모리에 접근할 때의 지연 시간과 주소 공간의 시작 및 끝 주소, 그리고 메모리에 접근하기 위한 포트의 개수를 명시하고 있다. 레지스터 파일 기술도 메모리 기술과 비슷한 방식을 가지고 있는데, 레지스터 파일의 이름 뒤에 대괄호 및 숫자를 이용하여 해당 레지스터 파일이 가지고 있는 레지스터의 개수를 표시할 수 있도록 하였다. 또한 해당 레지스터에 접근할 수 있는 연산 장치의 종류를 '<0..15>' 와 같이 명시하였는데 이는 0 번부터 15 번 연산 장치까지 해당 레지스터 파일에 접근할 수 있다는 것을 나타낸다.

확장된 SoarDL 언어에서는 재구성형 프로세서의 연산 장치의 개수 및 종류, 그리고 연결 구조를 기술할 수 있다. SoarDL 언어에서 연산 장치의 개수 및 종류는 C 언어의 배열 정의와 비슷한 방식으로 기술된다. 위의 예에서 지시어 'PEarray'를 이용하여 기술된 부분이 바로 그것인데, 재구성형 프로세서가 총 16 개의 연산 장치를 가지며 4 행 4 열의 구조를 이룬다는 것을 나타내고 있다. 연산 장치의 종류 또한 기술하고 있는데, 2 장에서 소개한 PE0 연산 장치와 PE1 연산 장치가 각각 4 개와 12 개라는 것과 함께 이들의 구성을 행렬 구조로 보여주고 있다. 마지막 지시어인 'connectivity'는 연산 장치의 연결 구조를 기술하는 데 사용된다. 위의 예에서는 연산 장치들의 연결 구조가 메시 구조라는 것을 지시어 'mesh'를 이용하여 기술하고 있다.



(그림 5) 기술된 재구성형 프로세서의 구조

이처럼 확장된 SoarDL 언어에서는 전역 저장 공간 및 연산 장치의 연결 구조를 효과적으로 기술할 수

있다. 위의 그림 5 는 그림 3 과 그림 4 에서 기술한 재구성형 프로세서의 전체 구조를 그림으로 나타낸 것이다.

4. 향후 연구 방향

앞선 2 장과 3 장에서 재구성형 프로세서를 기술하기 위해 SoarDL 언어가 어떻게 확장되었는지 설명하였다. 이렇게 확장된 SoarDL 언어로 재구성형 프로세서를 기술하고 나면 SoarGen 컴파일러가 이를 입력으로 받아들여 재구성형 프로세서를 위한 동작 코드를 생성하게 된다. 이를 가능케 하기 위해서는 SoarDL 언어로 기술된 재구성형 프로세서의 정보를 적절한 인터페이스 자료 구조를 이용해 저장하는 것이 필요하다. 이렇게 저장된 정보는 SoarGen 컴파일러 내에서 적절히 처리되어 실제 동작 코드를 생성하는 데 사용된다. 현재 주어진 응용을 개별 연산 장치에 할당하여 실제로 동작할 수 있는 코드를 생성할 수 있는 내부 모듈이 구현된 상태이며, 향후 인터페이스 자료 구조를 정의하여 SoarDL 언어로 기술된 정보를 구현된 내부 모듈로 전달할 수 있도록 할 예정이다.

재구성형 프로세서를 위한 보다 효과적인 코드를 생성하기 위해서는 적절한 최적화 과정이 필수적이다. 예를 들어 코드 내에 분기문이 존재할 경우 연산 장치로 할당하는 것이 매우 어려워지는 동시에 전체적인 코드의 수행 성능이 떨어지게 되는 문제가 있다. 이를 해결하기 위해 예측 수행(Predicate Execution)과 같은 최적화 기법이 필요한데, 이러한 최적화 기법이 반영된 코드를 생성하기 위해서는 실제 코드를 생성하는 모듈이 이러한 최적화를 반영할 수 있도록 변경되어야 한다. 앞으로의 연구에서는 SoarDL 언어로 기술된 정보를 바탕으로 실제 코드를 생성하는 모듈을 변경하여 재구성형 프로세서를 위한 보다 효율적인 코드를 생성하도록 할 예정이다.

5. 결론

본 논문에서는 아키텍처 명세 언어인 SoarDL 언어를 확장하여 재구성형 프로세서의 구조 및 특성을 기술하는 방안을 제시하였다. SoarDL 언어로 기술된 재구성형 프로세서의 정보는 SoarGen 컴파일러로 하여금 재구성형 프로세서에서 동작하는 코드를 생성하고 보다 효율적인 코드를 생성하기 위한 최적화 기법을 적용하는 데에 이용된다. 앞으로는 SoarDL 언어로 기술된 정보를 효과적으로 저장하고 전달할 수 있는 인터페이스 자료 구조의 개발과 함께 이러한 정보를 바탕으로 보다 효율적인 코드를 생성하기 위한 최적화 기법을 연구하고자 한다.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2011-0000975), 2011 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.2011-0018609) 및 IDEC 의 지원을 받아 수행되었습니다.

참고문헌

[1] K..Karuri, A.Chattopadhyay, X.Chen, D.Kammler, L.Hao, R.Leupers, H.Meyr, G.Ascheid, “A Design Flow for Architecture Exploration and Implementation of Partially Reconfigurable Processors”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008

[2] F.Bouwens, “Power and Performance Optimization for ADRES”, M.S Thesis, Delft University of Technology, 2006

[3] J.Yoon, J.Lee, S.Park, Y.Kim, Y.Paek, D.Cho, “I2CRF: Incremental Interconnect Customization for Embedded Reconfigurable Fabrics”, Design, Automation and Test in Europe, 2011

[4] 김용주, 허인구, 양승준, 이종원, 최영규, 백윤흥, “재구성형 프로세서 성능과 레지스터와의 상관 관계 탐구”, 제 33 회 한국정보처리학회 춘계학술발표대회, 2010

[5] 윤종희, 김용주, 이진용, 백윤흥, “랜덤 연결 구조를 갖는 재구성형 프로세서의 성능 분석”, 대한전자공학회 정기총회 및 추계학술대회, 2010

[6] 윤종희, 김용주, 박상현, 조두산, 이종원, 김경원, 백윤흥, “CGRA 를 위한 전력이 고려된 어플리케이션 매핑에 관한 연구”, 제 31 회 한국정보처리학회 춘계학술발표대회, 2009