재구성형 어레이 아키텍처에서 데이터 복사 흐름을 고려한 코드 매핑 기법

조두산
순천대학교 전자공학과
e-mail : dscho@sunchon.ac.kr

A Code Mapping Technique Considering With Data Copying Flow On Coarse-Grained Reconfigurable Array Architectures

Doosan Cho
*Dept. of Electrical Engineering, Sunchon National University

요 약

고성능 재구성형 어레이 아키텍처는 애플리케이션에 내재된 병렬성을 충분히 활용하도록 풍부한 하드웨어 리소스로 구성되어 있다. 이러한 하드웨어 리소스는 소프트웨어 파이프라이닝 기반 코드할당 기법을 통하여 사용된다. 이러한 코드할당 기법은 기존의 소프트웨어 파이프라이닝 기법에 FPGA 에서의 라우팅 & 위치선정기법 [1]이 연결된 형식으로 구성된다. 이러한 기존의 연구들은 데이터 흐름 (data flow)을 단순한 형태로 가정하여 개발되었다. 따라서 루프 코드 펼침 (loop unrolling)에 따라서 발생되는 데이터 복사에 의한 흐름 (copy flow)은 코드 매핑할 때 고려하지 않기 때문에 소프트웨어 파이프라이닝 적용시 네트웍 리소스의 중복사용으로 인한 데이터 충돌문제(data congestion)로 Minimum Initiation Interval (MII)증가에 따르는 성능 저하가 발생할 수 있다. 본 연구에서는 다양한 데이터 복사 흐름까지 고려하도록 데이터 의존도 그래프 (Data Dependence Graph, DDG)를 확장하여 스케쥴링 단계에서 데이터 충돌 지연에 의한 MII 증가를 방지하여 최적의 시스템 성능을 얻도록 코드 할당 기법을 개발하였다.

1. 서론

재구성형 시스템은 프로세서 설계에 있어서의 고려대상인 유연성과 성능사이의 트레이드오프(tradeoff)에 있어서 새로운 패러다임으로 등장하였다. 재구성형 어레이 프로세서의 특징은 재프로그래밍 가능한 로직블럭과 유연성을 제공하는 블록로직 사이의 상호연결구조 이다. 이러한 아키텍처 유형은 동작에 있어서 완전한 병렬형이라는 점에서 기존의 마이크로 프로세서와는 확연히 구분된다.

재구성형 아키텍처는 현재 기존의 프로세서와의 확연한 구조적인 차이로 인하여 애플리케이션 코드를 프로세서 위에서 올바르게 동작시키기 위하여 필수적인 툴이 아직 개발되어 있지 못한 실정이다. 특히 다양한 애플리케이션을 커버 할 수 있는 일반화된 컴파일러 툴은 없는 상태이며, 다만 멀티미디어 영역에서 간단한 코드에 대하여 코드 매핑이 가능하다는 결과가 지속적으로 보고되고 있다.

올바른 코드 매핑을 위한 컴파일러 개발이 어려운 이유는 코드매핑관련 리소스 할당 문제가 NP 문제인 만큼 얼마나 효율적인 휴리스틱을 적용하는지에 따라 결과가 크게 달라지기 때문이다. 따라서 현재는 소프트웨어 파이프라이닝 기반의 코드매핑 기법을 사용하고 있는 수준에 있다.

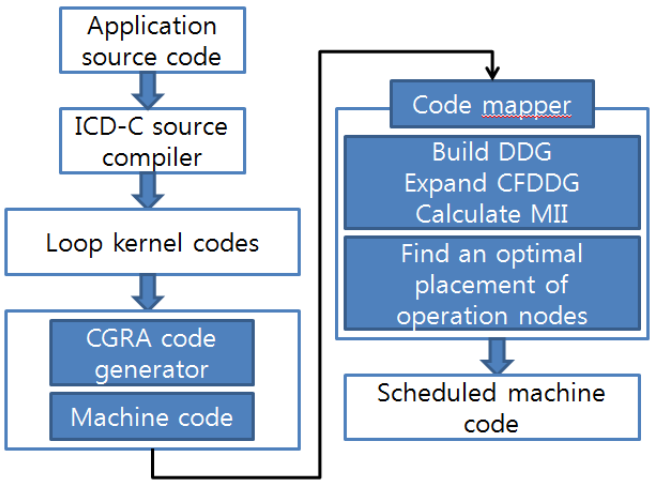
2. 목적 프로세서

본 연구에서는 다양한 CoarseGrained Reconfigurable Array Processors (CGRA)를 위하여 코드매핑 알고리즘을 사용할 수 있도록 하기 위하여 가상의 일반화된 CGRA [2]를 대상으로 코드매핑 기법을 설계하였다. 이것은 대부분의 주요 CGRA 아키텍처를 기반으로 하고 있다. 일반적인 CGRA 아키텍처는 크게 4 개의 요소로 구성된다. 첫째, 연산을 수행하는 프로세싱 유닛 (Processing Unit, PU)들은 2 차원 연결 네트웍 (2D inter-connection network)으로 구성된다. 둘째, 데이터를 저장하는 온칩 메모리인 스크래치패드 메모리 (scratchpad memory)와 액세스 버스를 포함하는 메모리 인터페이스. 셋째, PU 들의 동작을 지정해주는 구성 메모리 (configuration memory). 넷째, 메인 데이터를 저장하는 메인 데이터 메모리. 이상과 같은 구성요소로 이루어진 [2]에 기술된 일반화된 CGRA 의 아키텍처 템플릿을 대상으로 2 차원 공간애서 효율적인 코드 매핑기법을 다음 장에서 기술하도록 하겠다.

3. 제안하는 기법의 개요

아래 그림 1 에 제안하는 코드 매핑 기법의 개요를 나타내었다. 알고리즘의 입력은 애플리케이션 루프 코드이다. 우리는 소스레벨에서 우선 코드를 매핑 가능하도록 변형한다. 이를 위하여 소스레벨 컴파일러인 ICD-C 컴파일러 프레임워크 [3]를 사용하여 애플리케이션 소스 코드에서 CGRA 로 매핑할 부분을 추출하여 사용하였다.

추출된 루프 소스코드는 기본적인 연산의 명령어 집합으로 생성되어 각 명령어의 데이터의 흐름에 따라 DDG 를 구성한다. 이때 PU 의 개수가 DDG 를 구성하는 노드의 개수보다 2 배수 이상 많다면 성능 개선을 위하여 루프 펼침 최적화를 적용할 수 있다. 이때 매트릭스 곱셈과 같은 응용의 특성상 하나의 데이터는 여러 PU 사이에 공유가 발생한다. 이때 데이터 메모리에서 읽어 들인 최초의 PU 는 데이터를 사용할 다른 PU 들에게 해당 값을 복사 전달하게 된다. 이러한 복사 전달 흐름을 DDG 상에 표기할 수 있도록 복사 흐름 에지를 추가하여 DDG 를 확장하였다. 다시 말하면 응용 알고리즘의 특성과 루프 펼침 기법에 의하여 같은 값이 여러 PU 에서 사용되는 경우, 소프트웨어 파이프라이닝된 루프의 반복 순서에 따라 적절한 시점에서 복사 전달 이루어지게 됨으로 DDG 에 이를 표기하는 에지를 추가하였다. 이렇게 완성된 그래프를 우리는 데이터 복사흐름 의존도 그래프(Data Copy Flow & Dependence Graph, CFDDG)라고 부르겠다.



(그림 1) 알고리즘 개요

제안하는 매핑 기법은 스윙 모듈로 스케쥴링 (swing modulo scheduling, SMS) 기법[5]을 기반으로 구성되었다. SMS 기반 매핑 알고리즘으로 시작한다. CFDDG 에서 MII 까지 계산 완료하고 해당 루프코드 안에서 데이터 흐름의 사이클을 구성하는 역방향 에지 (backward edge)를 검색하여 II 를 결정하는 사이클 단위로 코드 할당기 (code mapper)에서 스케쥴링 순서를 결정한다. 여기서는 각 사이클을 구성하는 노드들을 각각의 집합(set)으로 구성한다. 이때 CFDDG 에서 데이터 흐름의 순서에 따라 위치선정 스케쥴링의 최종 순서를 결정하는데 이는 깊이 (depth) 를 기준으로 결정한다 [4]. 이렇게 스케쥴링에 앞서 순서 선정을 하면 사이클을 구성하는 노드들이 근접 영역에 할당될 수 있도록 하고 결과적으로 MII 가 증가되어 루프 코드의 실행시간이 늘어나는 것을 방지한다. 각 노드들의 위치선정 (placement)은 데이터 라우팅 지연 시간을 목적함수의 핵심 변수로 정하여 결정해야 한다. 제안하는 기법은 최적의 위치선정을 위하여 Exhaustive search 기법을 이용하여 결정하도록 하였다. 검색 공간이 지수증가 하기 때문에 컴파일 타임을 유효한 범위에서 제한하기 위하여 적절한 휴리스틱 (heuristic)을 사용하여 유효 (feasible) 솔루션을 찾도록 하였다.

```
Algorithm FindPlacement
 CFDDG //input of the algorithm
 While CFDDG is not empty do
  V = pop(CFDDG)
  While(!feasible)
   oldCost = computeCost()

   if (SearchPlacement(V)=true) then
    constructEdge(--feasible)
    newCost = computeCost()
    if(aceept(newCost -  oldCost) = false)) then
     restore relocated vertices
  end while
 end while
end algorithm
```

(그림 2) CFDDG 기반 위치선정 알고리즘

```
procedure SearchPlacement(v)
 found = false
 initialize n

 while (found = false || n > 0) do
  oldLoc  = save old location of v
  getNewPlacemnt(newLoc)
  if(| Dpeth(newLoc)-Depth(v)|>=|path(v,newLoc)|)
   v = newLoc
  newCost = computeCost (v, oldLoc)

  if (accept(newCost - oldCost)) then
   found ←true
   break
   n = n -  1
 end while
 return found
end procedure
```

(그림 3) 위치검색 프로시저

```
procedure computeCost (locV , v )
 sum ← 0

 for each e ∈ from locV to v
  c ← numberOfdelays(v, target(e))
  sum ← sum + c
 return sum
end procedure
```

(그림 4) 비용계산 프로시져

4. 제안된 알고리즘 상세

제안된 알고리즘은 우선 각 PU 들이 충분한 레지스터를 갖고 있는 상태를 가정한다. 즉, 스케쥴링 프로세스 이후 스필코드가 발생하는 것을 고려하고 있지 않다. 앞서 언급한대로 SMS 기법안에서 계산된 MII 를 되도록 벗어나지 않는 범위에서 CFDDG 노드들의 위치선정 솔루션을 찾을 때까지 알고리즘 FindPlacement 가 반복 수행된다. 우선 CFDDG 에서 노드들이 사이클을 구성하는 집합으로 분류되고 그 안에서 깊이에 따라 스케쥴링할 순서대로 정렬하여 그 순서에 따라 순차적으로 노드의 위치검색을 수행하게 된다. routeEdge()를 통하여 노드들 사이의 데이터 전달 비용을 계산하고 기존의 비용보다 개선 점이 있으면 이것을 위치선정 결과로 저장을 하게 된다. 이러한 작업을 반복하여 더 나은 결과가 없으면 다음 노드를 검색하게 된다. GetPlacementCost 는 얼마나 라우팅 패스가 짧은지를 지나가는 에지의 개수(지연 사이클)을 통하여 얻게 된다. 이때 검색공간이 너무 커지는 것을 방지하기 위하여 아래처럼 검색공간에 제약조건을 정하여 둔다.

제약조건 : | Dpeth(u) – Depth(v) | >= | path(v,u) |
이 제약조건은 노드 분류/정렬에 의하여 주어진 한 세트안의 노드쌍 (pair)의 라우팅 패스의 길이는 두 노드 사이를 연결하는 에지의 개수보다 작아야 한다는 것을 의미한다. 이 제약조건은 MII 가 증가하지 않는 범위에서 최적의 위치선정이 되도록 하는 안전장치이다. 이와 같은 방식으로 모든 노드들의 위치가 결정되면 그 결과를 유효 솔루션으로 보고한다. 만약 MII 범위에서 매핑이 실패할 경우 MII+1 과 | Dpeth(u) – Depth(v) |+1 의 값으로 위치검색을 반복한다.

5. 실험결과

모든 실험은 윈도우 7, 펜티엄 4 2.2GHz, 2GB 메모리에서 수행되었다. 벤치마크는 Matrix multiplication, DCT, FFT 에서 수행되었다. II 를 변경하면서 알고리즘의 성능 테스트를 수행하였고 CGRA 의 구조를 8x8, 16x16 등으로 변경하여 매뉴얼하게 테스트를 진행하였다. 실험은 우리가 제안하는 알고리즘과 기존의 일반적인 모듈로 스케쥴링 기반의 코드 매핑 알고리즘의 비교로 진행되었다. 각 코드의 실행시간을 측정하여 결과를 표 1 에 나타나있다.

<표 1> 실험결과

	8x8			16x16		
	Our	Comp	ratio	Our	Comp	ratio
Matrix	1.7	4.2	2.4	13	45	3.4
DCT	3.3	8.4	2.5	55	323	5.8
FFT	4.0	16.1	4.0	143	728	5.1

실행시간을 초단위로 기입하였으며 8x8 보다 16x16 에서 상대적으로 큰 입력 데이터를 사용하였기 때문에 실행시간이 더 길게 측정되었다. 전체적으로 2 배 이상 개선결과를 얻을 수 있음을 확인할 수 있었다.

Ack.nowledgment
이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (No. 2010-0024529). 본 지식재산권은 지식경제부 및 정보통신산업진흥원의 지원을 받아 수행된 연구결과임 (10-기반, 기술혁신사업).

참고문헌

[1] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns. “Placement and routing tools for the triptych fpga” IEEE Trans. VLSI Syst., 3:473-382, 1995.

[2] R. Hartenstein. “Coase Grain Reconfigurable Architectures” Proc. ASPDAC, 564-570, 2001.

[3] ICD-C compiler framework, www.icd.de/es/icd-c/, Informatik Centrum Dortmund

[4] J. Liosa, E. Ayguade, A. Gonzalez, M. Valero, and J. Eckhardt. “Lifetime-Sensitive Modulo Scheduling in a Production Environment” IEEE Trans. On Computers, 2001.

[5] Y. Tian, Edwin H., S. Chantrapornchai, and P. Kogge. “Efficient data placement for processor in memory array processors” ISATED Proc. Of Parallel and Distributed Computing and Systems, 1997.