논문 23-1-R02

Dynamic-Partitioned SIMD 기법을 사용한 Reconfigurable Processor와 이를 위한 컴파일러 설계

Design of Reconfigurable Processor using Dynamic-Partitioned SIMD and Compiler

권용인 · 윤종희 · 이종원 · 김용주 · 백윤흥

서울대학교 전기 · 컴퓨터공학부 전기공학전공

yikwon@optimizer.snu.ac.kr, jhyoun@optimizer.snu.ac.kr, jwlee@optimizer.snu.ac.kr
yjkim@optimizer.snu.ac.kr, ypaek@snu.ac.kr

요 약

최근 다양한 응용프로그램을 위한 임베디드 프로세서가 많이 개발 되고 있다. 또한 반도체 기술의 발달과 새로운 응용프로그램들에서의 요구로 인해서 하나의 칩에서 수행되어야 하는 작업들이 더욱 많아지고 있다. 전통적으로 DSP와 ASIP은 코드의 성능을 높이기 위하여 손으로 assembly 작성을 많이 해왔다. 그러나 응용프로그램들의 복잡성과 time-to-market 제약조건 때문에 임베디드 시스템 개발자들은 High Level Language와 컴파일러를 이용하여 코드를 생성함으로써 직접 손으로 assembly 코드를 만드는 과정에서 오는 부담을 줄이기를 원하고 있다.

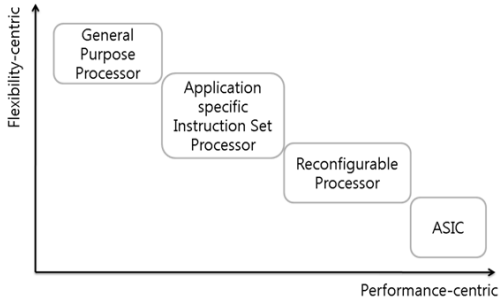
한 편 Multimedia Processing을 위한 Reconfigurable Processor ERP는 DP-SIMD(Dynamic-Partitioned Single Insturction Multiple Data)를 사용한다. DP-SIMD기법은 Reconfigurable Processor의 Processing Unit들이 동시에 네 가지의 Instruction 중 하나를 선택하여 수행할 수 있게 함으로써 SIMD(Single Instruction Multiple Data)에 비해 Operation Efficiency를 높임으로서 H.264의 성능을 향상시켰다.

하지만 DP-SIMD를 활용하기 위한 assembly 코딩 작업은 많은 노력과 시간을 필요로 하기 때문에 컴파일러의 필요성이 증대되었다. 이에 대해 본 논문은 ERP를 위한 컴파일러를 개발 설계하고, H.264의 de-blocking filter(DF), intra prediction(IP)를 실험하였다. 실험결과 기존 SIMD Compiler에 비해 DF의 경우 1.48배, IP의 경우 1.83배 성능이 향상되었음을 알 수 있었다.

1. 서 론

오늘날, H.264는 비디오 압축 기술로 널리 사용되고 있다. 점점 더 높은 성능의 비디오 압축 기술이 필요해지고 있고, 따라서 H.264를 위한 다양한 하드웨어가 개발되고 있다[1-8]. 그 중 하나로 Application-specific integrated circuit(ASIC)을 사용하여 H.264의 일부분이나 전체가 개발되고 있다. ASIC을 사용하여 개발된 H.264 encoder와 decoder는 매우 높은 성능을 보여주고 있으나 수정이 필요할 때 flexibility에 제한이 있다는 단점이 있다. 반면, Application specific in-

struction set processor(ASIP)과 Configurable processor는 변화에 대해 높은 flexibility를 가지고 있으나 성능이 ASIC에 비해 떨어진다. General Purpose Processor 역시 다양한 시스템에 적용될 수 있도록 flexibility가 높지만 ASIC과 같은 성능을 내기 위해서는 하드웨어 비용이 높아지고, 발열량과 전력소모량이 커지며 그에 따라 크기가 커지기 때문에 휴대성 또한 떨어지게 된다. 반면 Reconfigurable Processor는 General Purpose Processor와 ASIC이 가진 장점인 높은 성능과 높은 flexibility 모두를 지니고 있다. [그림 1]은 Performance와 Flexibility 사이의 Tradeoff를 나타내고 있다.

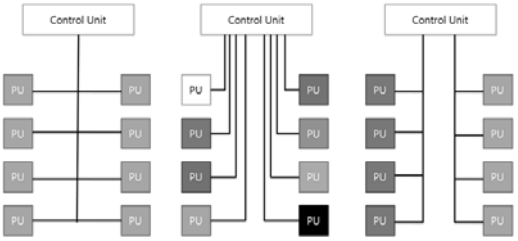


[그림 1] tradeoff between flexibility and performance

또한 이렇게 다양한 하드웨어가 개발되는 만큼 어플리케이션 개발속도도 중요하다. 특히 컴파일러를 이용하여 어플리케이션을 개발할 경우 개발기간을 크게 단축시킬 수 있으리라 본다. 이 논문은 DP-SIMD 프로세서의 특징과 Soargen을 사용하여 컴파일러를 생성하는 과정을 보여주고, 실험을 통해 그 성능측정을 하였다.

2. Dynamic-Partitioned SIMD 아키텍쳐

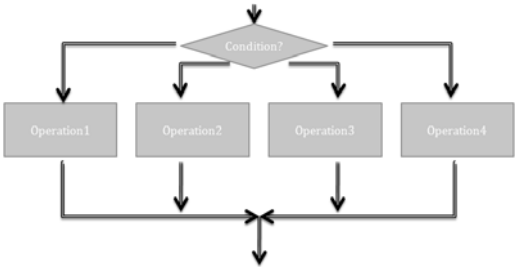
Reconfigurable Processor는 보통 성능향상을 위해 Array Architecture를 가지고 있고, 효율을 높이기 위해서 Array를 현명하게 컨트롤 해야할 필요성이 있다. Reconfigurable Array를 컨트롤하는 방법을 크게 두가지로 나눌 수 있다. 첫 번째는 Single instruction multiple data(SIMD)이고 두 번째는 Multiple instruction multiple data(MIMD) 이다. MIMD 컨트롤 방식은 각 Processing Unit에 다른 Processing을 수행하게 하여 Processing 성능을 향상시킬 수 있다. 하지만 이 방식은 Command가 길어진다는 단점을 갖고 있으며, 이로 인해 프로그램 메모리에 부담이 커지게 된다. 반면에 SIMD 컨트롤 방식은 각 Processing Unit이 동일한 Processing을 수행하게 하는 방식이다. 따라서 MIMD 컨트롤 방식이 가진 단점을 해소할 수는 있지만 성능은 MIMD에 비해 낮아지게 된다. SIMD의 단점을 개선하기 위해 Partitioned SIMD(P-SIMD) 컨트롤 방식이 제안되었고, P-SIMD는 Processing Unit들을 여러개의 SIMD 그룹으로 나누어 그룹마다 서로다른 Instruction을 수행하도록 한다[9]. 하지만 FFT와 Integer Transform 등과 같은 여러 그래픽 알고리즘에서 뛰어난 성능을 보이지만 H.264의 De-blocking Filter와 같이 각각의 Processing Unit들이 Dynamic하게 그룹이 형성된다면 P-SIMD의 기능만으로 성능 향상을 기대하기 힘들다. 그래서 새롭게 제안된 Dynamic-Partitioned SIMD (DP-SIMD) 아키텍쳐는 기존 SIMD가 가진 문제점들을 해결하고 시스템 성능을 높일 수 있다[10].



[그림 2] SIMD vs MIMD vs P-SIMD

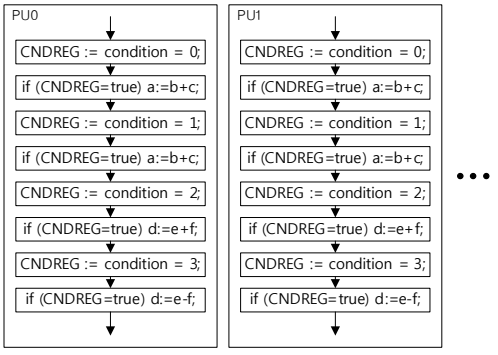
2.1. Dynamic-Partitioned SIMD의 Control Method

H.264는 Variable Length Decoding(VLD), Inverse Transform and Quantization(ITQ), Motion Compensation (MC, Intra Prediction(IP), De-blocking filter(DF)로 이루어져있다. 이 중 VLD는 병렬수행이 불가능하고 ITQ는 SIMD Control Method으로 충분히 성능을 낼 수 있다. 하지만 IP, MC, DF는 SIMD Control Method보다 MIMD Control Method를 사용할 경우 성능이 훨씬 더 좋아진다. 왜냐하면 SIMD Control Method를 사용하면 Conditional Operation이 많아져 Efficiency를 떨어뜨리기 때문이다.



[그림 3] Conditional Operation

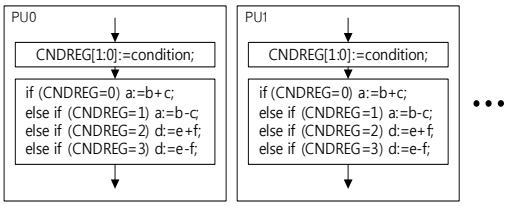
[그림 3]은 Conditional Operation의 예이다. Condition에 따라 4가지 다른 Operation을 수행해야 하는 경우를 나타내고 있다.



[그림 4] Conditional Operation for SIMD

모든 Processing Unit들은 같은 Operation을 수행해야하기 때문에 SIMD Control Method으로 Conditional Operation을 수행하기 위해서는 위 [그림 4]와 같은 코드가 수행되어야한다. 우선 flag register인 CNDREG가 설정되고 그 다음에 오는 Operation은 CNDREG에 따라 수행될지 말지를 결정한다. 그리고 다음 Conditional Operation들도 이와 같은 작업이 수행된다. 따라서 Conditional Operation이 존재 할때 SIMD Control Method은 시스템의 성능을 떨어뜨리게 된다. 반면에 MIMD Control Method은 Conditional Opeartion이 있을 때 병렬로 Operation을 수행함으로 성능을 향상시킬 수 있다.

한편, DP-SIMD에서는 Processing Unit들이 자신이 가지고 있는 데이터에 따라 Instruction을 선택할 수 있다. 따라서 병렬 프로세서들이 매번 다이나믹하게 Instruction을 선택 수행하게 되는 것이다.

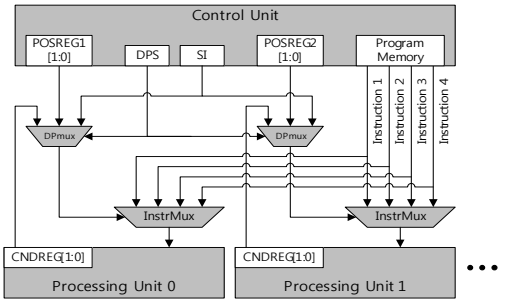


[그림 5] Conditional Operation for DP-SIMD

[그림 5]는 DP-SIMD Control Method에 의해 각 Processing Unit들이 수행하는 Operation을 보여주고 있다. 2bit의 CNDREG를 Condition에 따라 설정한 후, CNDREG에 따라 Processing Unit이 Instruction을 선택하여 수행하게 되어, [그림 3]의 SIMD Control Method에 비해 성능이 4배 빨라졌음을 알 수 있다. 이와 같이 한 번에 네 개의 인스트럭션 중 하나를 선택하여 수행해야할 경우, DP-SIMD Control Method를 사용하면 SIMD Control Method에 비해 최대 4배의 성능이 향상된다.

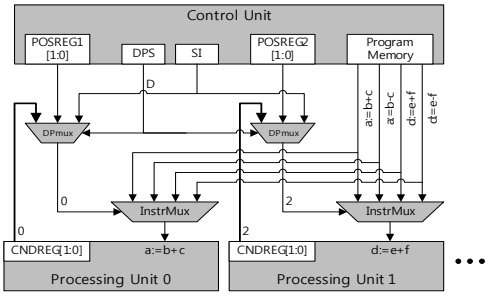
2.2. Dynamic-Partitioned SIMD 아키텍쳐

Processing Unit들에 DP-SIMD Control Method을 적용함으로써 낮은 Operation Efficiency 문제는 해결되었다. DP-SIMD Instruction은 여러개의 Instruction을 하나로 묶는다. Very Long Instruction word(VLIW)와 비슷하지만 VLIW와는 다르게 Function Block에 따라 순서가 정해져있지 않고 Processing Unit이 Condition에 따라 Instruction을 선택하여 수행하게 된다.



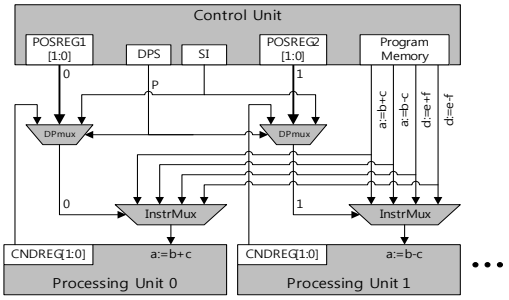
[그림 6] 프로세서의 구조

[그림 6]은 DP-SIMD Control Method을 사용하는 프로세서의 구조를 나타내고있다. 각 Processing Unit들은 병렬로 연결되어있고 Control Unit들은 Processing Unit들을 컨트롤한다. Control Unit은 Program Memory로부터 Instruction들을 읽어들여 Processing Unit으로 전달하고 Processing Unit은 Instruction들 중 하나를 선택하여 수행하게 된다.



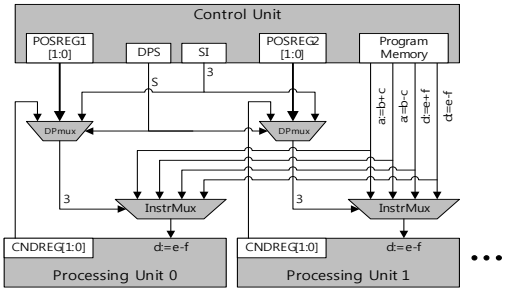
[그림 7] Data based partitioned SIMD control

DP-SIMD Control Method으로 Processing Unit을 컨트롤 하는 방법은 세가지가 있다. 첫번째로 Processing Unit에 의해 계산된 데이터를 바탕으로 Processing Unit을 컨트롤 하는 방법이다. 이를 'data based partitioned SIMD control'이라고 부른다. [그림 7]은 data based partitioned SIMD control를 나타내고 있다. 이와 같은 경우 conditional operation을 수행하기 위해 PU는 CNDREG Register에 값을 넣고 Control Unit은 DPS 시그널에 D를 인가하여 CNDREG의 값에 따라 Instruction 넷 중 하나를 선택하여 수행하게 한다.



[그림 8] Position based partitioned SIMD control

두 번째는 Control Unit에 의해 어느 Processing Unit이 어느 Instruction을 수행할 지가 미리 정해져있는 경우이다. 이를 'position based partitioned SIMD control'이라 부른다. [그림 8]은 예를 보여주고 있다. Control Unit에 의해 POSREG Register들이 설정되고 DPS에 P Singal을 인가시키면 Processing Unit들은 Control Unit에 의해 선택된 Instruction을 수행하게 된다.



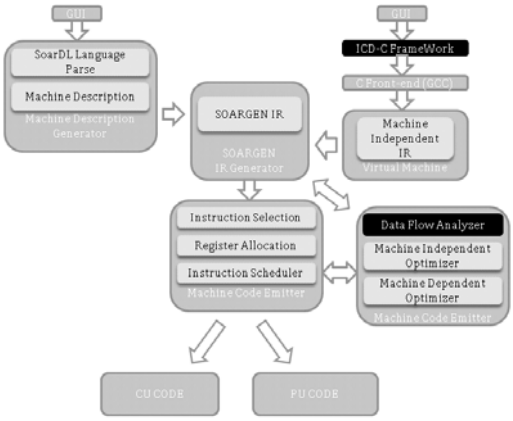
[그림 9] SIMD control

마지막으로 세 번째 경우는 모든 Processing Unit들이 같은 Instruction을 수행하여 마치 SIMD Control Method과 같다. 이를 'SIMD contrl'이라 부르고 [그림 9]는 그 예이다. Control Unit은 DPS Signal에 S를 인가하고 SI signal로 넷 중 어느 Instruction을 수행할 지 결정한다. PU는 Control Unit이 정해주는 Instruction을 수행하게 된다.

3. Soargen Retargetable Compiler

SoarGen은 소프트웨어 최적화 연구실에서 개발한 Retargetable compiler platform이다. Code generation에 필요한 각 알고리즘이 component로 구성되어 이 component를 서로 연결하여 code generation 작업이 수행되도록 고안되었다. 각 Component중에서 target machine의 정보가 필요한 부분은 SoarDL parser로부터 자동으로 생성된다.

[그림 10]은 ERP를 위한 SoarGen compiler의 전체 구조를 나타낸 그림이다.



[그림 10] Soargen Compiler for Dynamic-partitioned SIMD

SoarGen compiler는 크게 다음의 component들로 구성되어 있다.

- front end
- virtual machine & virtual assembly
- soargen intermediate representation(IR)
- analyzer
- optimizer
- local code generator
- global register allocator
- machine description information

여기에 DP-SIMD 기능을 위해 추가한 component는 다음과 같다.

- ICD-C FrameWork
- analyzer
  - Function Call to Instruction
  - CU / PU Seperator
- local code generator
  - DP-SIMD Mode
  - IF Mode

3.1. ICD-C FrameWork

C Language로는 변수 선언 시 Control Unit (CU)의 변수인지 Processing Unit(PU)의 변수인지를 표현하지 못한다. 하지만 C Language를 확장한다면[11] 이 문제를 해결 할 수 있다. 본 컴파일러에서는 #pragma를 이용하여 변수에 CU, PU구분을 해 주었다. ICD-C FramWork 에서는 pre-processing을 하여 #pragma를 C Language가 readable한 Code로 만들어준다.

3.2. front end

front end는 일반적으로 compiler에서 source code를 분석하고 문법적인 오류를 checking한 뒤에 이를 compiler내부에서 사용되는 intermediate representation을 변환해주는 역할을 하는 module이다. 우리는 독자적인 front end를 제작하지

않고 gcc compiler를 우리의 front end로 사용하였다.

3.3. Soargen intermediate representation(IR)

Soargen IR은 SoarGen compiler 내부에서 code를 생성하기 위해 필요한 intermediate re-epresentation이다. 특히 tree code generation을 하기 위해서 virtual assembly를 graph의 형태로 표현한다.

3.4. analyzer

analyzer는 virtual assembly code에 대해서 data flow analysis를 한다. 여기에서는 reaching definition analysis, DU/UD chain, web, live range analysis 등의 data flow analysis를 하고 ERP 를 위해 Function Call To Instruction과 CU/PU Seperator를 수행한다.

Instruction	Description
MAX a b > c	c := (a > b) ? a : b
MIN a b > c	c := (a < b) ? a : b
CLIP a b > c	c := max(0, min(a, b))
SUBABS a b > c	c := abs(a - b)
SUBABS4 a b c d > e	e := abs(a - b) + abs(c - d)
SRAC a b > c	c := a >> b + carry
CMP a cnd b > c	c := (a cnd b) ? 1 : 0

[그림 11] special instruction

[그림 11]의 special instruction들은 Vedio Processing을 위해 필수적인 instruction이지만 C language로 표현이 불가능하다. 따라서 C Language로는 Function Call 형태로 기술하고 analyzer단계에서 이를 Instruction으로 바꿔준다. 알고리즘은 다음과 같다.

1. *Function Call의 Out Parameter를 Control Flow Graph에서 모두 찾아낸다.*
2. *Function Call의 Return Register가 사용되는 instruction들을 모두 찾아낸다.*
3. *Function Call의 Memory Argument를 GPR로 바꾼다.*
4. *Function Call을 Instruction 으로 바꾼 후 Destination Register를 할당한다.*
5. *Return Register를 4에서 할당한 Register로 바꾼다.*

Function Call To Instruction을 수행하고 난 후 analyzer에서는 web을 만든다. web을 만들기 위해서 우선 reaching definition analysis를 하여 definition과 use 사이의 관계를 파악한 뒤에 UD chain을 만들고 미리 생성된 정보를 이용하여 DU/UD chain 을 만든다. DU/UD chain 의 maximal이 web이 된다. 이렇게 해서 생성된 web으로 묶인 symbolic variable은 renaming을 거쳐서 unique한 이름을 갖도록 바꾸어 준다. unique한 이름을 가지는 이유는 나중에 global register allocation을 쉽게 하도록 하기 위함이다.

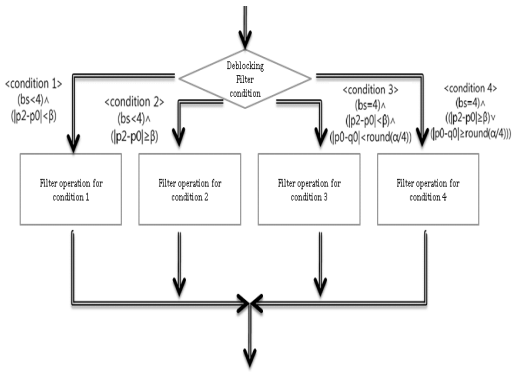
그런 후 pre-processing때 전달되었던 CU/PU 정보를 통해 모든 Register에 대해 PU class와 CU class로 나눠준다. Data flow graph을 이용하여 모든 instruction을 탐색하면서 만약 PU Register일 경우 Operator를 PU용 Operator로 바꿔준다. 예를 들어 ss_plus의 source, destination register가 PU register였을 경우 ss_plus를 pu_ss_plus로 바꿔 저장하게 된다.

4. 실험

De-blocking Filter(DF)와 Intra Prediction(IP)은 H.264의 성능에 큰 부분을 차지하고있다. DP-SIMD Control Method을 사용하여 DF와 IP를 구현하였을 때 기존 SIMD Control Method에 비해 얼마나 빨라졌는지 실험 결과를 통해 알 수 있다.

4.1. De-blocking Filter

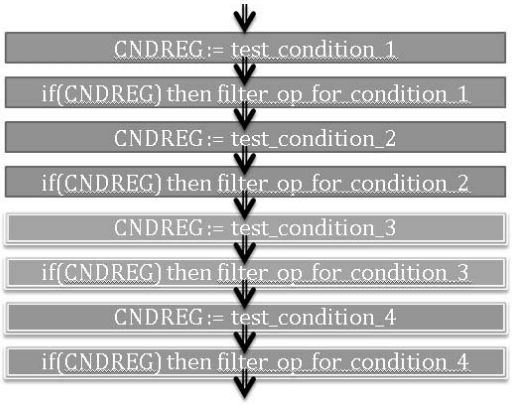
De-blocking Filter란 영상을 여러개의 블록으로 나눠서 인코딩 하는 경우 부작용으로 블록화 현상이 생기게 되는데 이런 블록화 현상이 있는 영상을 보정해주는 것을 말한다. Deblocking Filter의 알고리즘은 다음 [그림 12]와 같다.



[그림 12] De-blocking Filter Algorithm

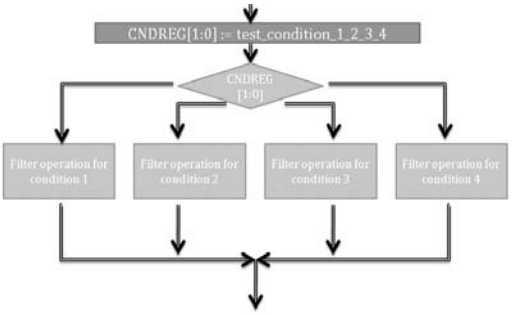
Boundary Strength(bs) 값과 p2, p0값 등에 따라서 네가지 경우로 나뉘어 져 각기 다른 Operation을 수행한다. 먼저 bs값이 4보다 작을 경우 condition 1과 condition 2로 분류된다. 그리고 p2와 p0의 차 값에 따라 다시 분류된다. 두번째로 bs값에 4보다 클 경우 condition 3과 condition 4로 분류된다. 그리고 p2와 p0의 차 값에 따라 다시 condition 3과 condition 4가 분류 된다.

[그림 13]은 DF를 SIMD Control Method으로 작동시켰을 때의 Operation method를 보여준다. 먼저 CNDREG을 condition 1에 의해 셋팅한다. 그리고 conditional operation으로 CNDREG가 1일 경우 Filter Operation 1을 수행한다. 그런 다음 CNDREG를 condition 2에 의해 셋팅하고 Filter Operation 2을 수행한다. 이런 방법으로 Filter Operation 4까지 모두 수행을 하고나면 DF는 종료된다.



[그림 13] De-blocking Filter for SIMD

하지만 DP-SIMD Control Method을 사용하면 [그림 14]과 같이 Operation을 수행할 수 있다. 먼저 모든 condition에 대해 2bit의 CNDREG에 셋팅하고 병렬로 네가지 operation 중 하나를 수행하게 한다.



[그림 14] De-blocking Filter for DP-SIMD

De-blocking filter를 SIMD를 지원 하는 컴파일러와 DP-SIMD를 지원 하는 컴파일러를 각각 사용하여 구현하였다. 결과는 아래 표와 같다.

	Old Compiler	DP-SIMD Compiler	Efficiency(Old compiler / DP-SIMD Compiler)
cycle	222	150	1.48

DP-SIMD Control Method으로 코드를 생성하는 Compiler의 결과가 SIMD Control Method

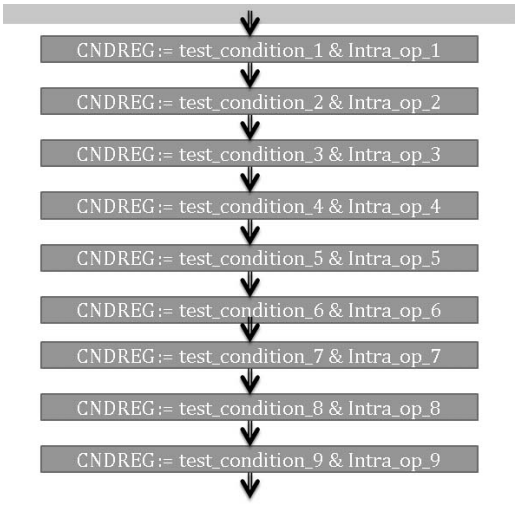
으로 코드를 생성하는 Compiler에 비해 성능이 1.48배 빨라졌음을 볼 수 있다. DF를 Hand-optimize한 코드의 결과는 아래 표와 같다.

	SIMD	DP-SIMD	Efficiency (SIMD / DP-SIMD)
cycle	78	50	1.56

Hand-Optimized 코드 역시 DP-SIMD 코드가 SIMD 코드에 비해 성능이 뛰어났으며 Compiler에 비해 hand-optimized Code가 성능이 3배 더 좋음을 알 수 있다.

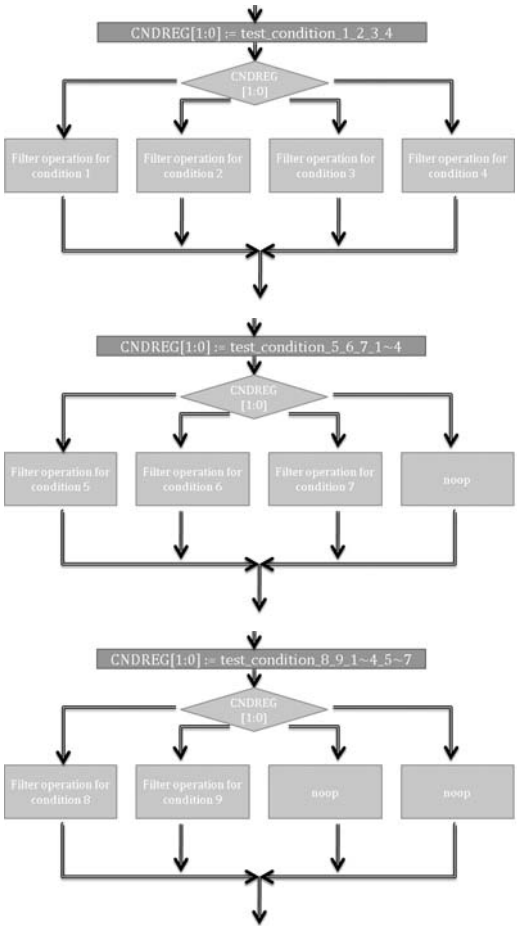
4.2. Intra Prediction

영상은 자기 주변의 화소값들이 대체로 비슷한 값을 가진다. H.264의 최소 block인 4x4나 16x16 block들도 이웃한 block 들과 값이 비슷하다. 이러한 block 간의 값의 유사성을 이용해서 block 간의 값의 차를 인코딩한다. Intra prediction은 9가지 모드가 있다. 9가지 모드에 따라 9가지 다른 Operation을 수행해야한다. SIMD 컨트롤 방법으로 구현하면 [그림 15]와 같다.



[그림 15] Intra Prediction for SIMD

9개의 Operation을 위해 9번 CNDREG를 셋팅하고 각각의 Operation을 수행하게 된다. 반면에 DP-SIMD 컨트롤 방법으로 작동을 시키려면 먼저 9가지 모드 중 네가지 모드에 대해 DP-SIMD로 Operation을 수행한 후, 추가로 세가지 모드에 대한 Operation과 이미 Operation이 수행된 모드들에 대해서는 nop으로 채운 Operation을 수행시킨다. 그다음 단계에서 남은 두가지 모드에 대한 Operation과 이미 수행된 모드들에 대한 nop Operation을 수행하면 9가지 모드에 대한 DP-SIMD Operation을 모두 수행할 수 있게 된다. [그림16]은 DP-SIMD로 IP를 구현한 모습이다.



[그림 16] Intra Prediction for DP-SIMD

intra prediction을 SIMD를 지원 하는 컴파일러와 DP-SIMD를 지원하는 컴파일러를 각각 상용하여 구현해 보았다. 결과는 아래 표와 같다.

	Old Compiler	DP-SIMD Compiler	Efficiency(Old compiler / DP-SIMD Compiler)
cycle	384	210	1.83

DP-SIMD 컨트롤 방법으로 코드를 생성하는 Compiler의 결과가 SIMD 컨트롤 방법으로 코드를 생성하는 Compiler에 비해 성능이 1.83배 좋아졌음을 볼 수 있다. IP를 Hand-optimize한 코드의 결과는 아래 표와 같다.

	SIMD	DP-SIMD	Efficiency (SIMD / DP-SIMD)
cycle	199	105	1.9

5. 결론

본 논문에선 Retargetable Compiler인 SoarGen을 사용하여 DP-SIMD 컨트롤 방법을 지원하는 ERP 프로세서용 컴파일러를 제작하였고, 이 컴파일러 사용해서 생성된 H.264의 De-blocking Filter와 Intra Prediction을 코드를 사용해 성능을 검증하였다. 실험 결과는 사람이 직접 작성한 코드보다 다소 성능이 떨어지는 것으로 나왔다. 물론 Hand-optimized code보다 성능이 뛰어난 코드를 컴파일러가 생성하기는 거의 불가능하지만 Soargen Compiler의 성능이 떨어지는 이유는 다음과 같다. 우선 Array의 load / store 시 hand-optimized Code의 경우에는 주소를 바로 입력해서 사용하지만 컴파일러의 경우 주소를 4씩 계속 더해서 사용하기 때문에 Code가 길어진다. 두 번째는 ERP의 경우 R0는 항상 0이 assign되어있기 때문에 immediate value로 0을 사용할 시 R0를 사용하면 되나 Soargen Compiler는 이를 위한 최적화 기능이 없어 항상 MOVI를 이용하여 0을 assign해야한다. 특히 Processing unit은 MOVI가 없기 때문에 Control Unit에서 MOVI를 한 다음 Processing Unit으로 전달을 해 주어야 하기 때문에 더욱 코드가 길어지게 된다. 세 번째로 Soargen Compiler에 현재 구현되어있는 최적화 기법은 DAG을 Tree로 바꿔주는 Dismantle 밖에 없기 때문에 여러 redundant 한 코드들이 남아있다. 이는 추후에 최적화 기법을 적용함으로써 개선될 수 있으리라 본다. 이런 단점에도 불구하고 컴파일러를 사용하면 어플리케이션 개발 시간이 크게 단축될 수 있다. DF의 경우 최적화된 Hand-optimized code를 만들기 위해 compiler로 code를 생성할 때 보다 10배 정도의 시간이 소모된다. 컴파일러를 이용하여 어플리케이션을 작성한 뒤 이를 이용하여 테스트를 하거나 hand-optimizing을 하는 것 또한 좋은 방법이다. 따라서 Soargen compiler를 사용함으로써 생기는 performance loss를 감안하더라도 이의 사용은 충분히 가치가 있는 일이다.

참 고 문 헌

[1] M. Horowitz, A. Joch, F. Kossentini, etal., “H.264/AVC Baseline Profile Decoder Complexity Analysis,” IEEE Trans. Circuits Syst. for Video Technol., vol.13, July 2003, pp.704-716

[2] S. López, F. Tobajas, G. M. Callicó, etal., “A Novel High Performance Architecture for H.264/AVC Deblocking Filtering,” ETRI Journal, vol.29, no.3, Jun. 2007, pp.396-398

[3] M. Oh, W. Lee, Y. Jung, et al., “Design of High-Speed CAVLD Decoder Architecture for H.264/AVC,” *ETRI Journal,* vol. 30, no.1, Feb. 2008, pp.167-169

[4] S. M. Park, M. Lee, S. Kim, et al., “VLSI Implementation of H.264 Video Decoder for Mobile Multimedia Application,” ETRI

Journal, vol.28, no.4, Aug. 2006, pp.525-528
[5] D. Yeo and H. Shin, "High Throughput Parallel Decoding Method for H.264/AVC CAVLC," *ETRI Journal,* vol. 31, no. 5, Oct. 2009, pp.510-517
[6] K.-S. Choi and S.-J. Ko, "Adaptive Scanning Based on a Morphological Representation of Coefficients for H.264/ AVC," *ETRI Journal,* vol.31, no.5, Oct. 2009, pp.607-609
[7] Y. Kun, Z. Chun, and W. Zhihua, "Application Specific Processor Design For H.264 Baseline Profile Bit-Stream Decoding," *Proceedings of The 8th International Conferenceon Signal Processing,* 2006, pp.16-20
[8] J. H. Han, M. Y. Lee, Y. Bae, et al., "Application Specific Processor Designfor H.264 Decoder with a Configurable Embedded Processor," ETRI Journal, vol. 27, no.5, Oct.2005, pp.491-496
[9] H. Singh, M.-H. Lee, G. Lu, etal., "Morpho Sys: An Integrated Reconfigurable System for Data-Paralleland Computation-Intensive Application," *IEEE Transactionson Computers,* vol.49, no.5, May 2000, pp.465-481.
[10] Chun-Gi Lyuh, Jung-Hee Suk, Ik-Jae Chun, and Tae Moon Roh, "A Novel Reconfigurable Processor Using Dynamically Partitioned SIMD for Multimedia Applications" ETRI Journal, vol.31, no.6, Dec. 2009, pp.709-716.
[11] Shorin Kyo, Shin'ichiro Okazaki, Tamio Arai, "An Integrated Memory Array Processor Architecture for Embedded Image Recognition Systems", ACM SIGARCH Computer Architecture News, vol 33, issue 2, 2005, pp.134-145
[12] JVT H.264/AVC Joint Model Reference S oftware version 11.0, http://iphome.hhi.de/suehring/tml/download/old_jm/jm11.0.zip, Aug. 2007

권 용 인
2002~2007년 한국과학기술원 전기 및 전자공학과(학사)원 전산학과(학사)
2008년~2009년 서울대학교 전기컴퓨터공학부(석사)
관심분야: 프로그래밍 언어, 컴파일러

윤 종 희
1972년~1976년 서울대학교 전
2003년 경북대학교 전기공학부(석사)
2003년~현재 서울대학교 전기컴퓨터공학부 석박사통합과정
관심분야 : 임베디드소프트웨어, 컴파일러

이 종 원
2007년 서울대학교 전기공학부(학사)
2007년~현재 서울대학교 전기컴퓨터공학부 석박통합과정
관심분야 : 임베디드소프트웨어, 컴파일러

김 용 주
2006년 서울대학교 전기공학부(학사)
2006년~현재 서울대학교 전기컴퓨터공학부 석박통합과정
관심분야 : 임베디드소프트웨어, 저전력 설계

백 윤 흥
1988년 서울대학교 컴퓨터공학과 (학사)
1990년 서울대학교 컴퓨터공학과 (석사)
1997년 UIUC 전산과학(박사)
1997년~1999년 NJIT 조교수
1999년~2003년 KAIST 전자전산학과 부교수
2003년~2007년 서울대학교 전기컴퓨터공학부 부교수
2007년~현재 서울대학교 전기컴퓨터공학부 교수
관심분야 : 임베디드 소프트웨어, 임베디드 시스템 개발 도구, 컴파일러, MPSoCs