# 데이터 이동 명령어 최소화를 위한 레지스터 할당 기법

홍성현°, 김진표, 문수묵
서울대학교 전기공학부 MASS 연구실, 삼성전자, 서울대학교 전기공학부 MASS 연구실
hongshow@altair.snu.ac.kr°, jinpyo11@hotmail.com, smoon@altair.snu.ac.kr

# Early-Split register coalescing for reducing data move instruction

SungHyun Hong°, Jinpyo Kim, Soo-mook Moon
MASS lab school of EE SNU, Samsung Electronics, MASS lab school of EE SNU

요 약

copy, load, store와 같은 데이터 이동 명령은 프로그램의 수행시간을 늘리며 코드의 크기도 증가시킨다. 따라서, 최적화 컴파일러의 레지스터 할당 단계에서 이런 데이터 이동 명령들을 줄이는 것이 중요하다. 데이터 이동 명령을 줄이기 위해서 그래프 컬러링 기반의 레지스터 할당 기법의 다양한 개선안이 나와있다.

여기서는 이 중에서 조기 분할 레지스터 융합 기법을 VLIW 시뮬레이터에서 구현하여 그 성능을 확인해본다. 조기 분할 레지스터 융합 기법은 융합된 가상 레지스터가 가장 적은 비용을 가지는 레지스터를 포함하고 있는 경우, 융합된 레지스터 자체를 스필하지 않고, 잠재적 스필 단계에서 분리하여 적은 비용의 레지스터만을 스필하도록 하는 것이다.

## 1. 서론

컴퓨터 메모리 구조 중에서 레지스터는 접근에 필요한 시간이 따로 없이 접근할 수 있는 가장 빠른 메모리이다. 또한, 컴퓨터의 연산을 위해서는 사용될 값이 레지스터에 담길 필요가 있다. 따라서, 우리는 연산하고자 하는 값들이 모두 레지스터에 담겨있기를 원한다. 하지만 실제 사용되는 머신들에서 레지스터의 개수는 한정되어 있어서 레지스터 간에 혹은 레지스터 외의 메모리 구조들에서 레지스터로 데이터를 이동시키는 명령어가 필요하다. 일반적으로는 copy, load, store의 세 가지 정도의 데이터 이동 명령어가 주어진다. copy는 레지스터에서 레지스터로 데이터를 이동시키고, load는 캐쉬나 메인 메모리에서 레지스터로 데이터를 이동시키고, store는 레지스터에서 캐쉬나 메인 메모리로 데이터를 이동시킨다. 이런 데이터를 이동시키는 명령어는 실제적인 연산작용을 하지는 않지만, 캐쉬나 메인 메모리에 접근해야 하므로 CPU의 cycle을 많이 사용하게 된다. copy 명령어의 경우는 한 cycle만을 사용하지만, 코드의 크기를 늘리게 된다. 코드 크기의 증가는 명령어 캐쉬의 불일치를 일으키게 됨으로 피해야만 한다.

따라서, 데이터 이동 명령어들 최소화 시키는 것은 중요하다. 이런 최적화는 레지스터 할당 단계에서 적용할 수 있다. 데이터 이동 명령어를 최소화 시킬 수 있도록 개선한 레지스터 할당 기법은 다양하게 제안되었다. 이미 제안된 할당 기법으로 적극적 기법, 신중한 기법, 반복적 기법, 낙천적 기법, 조기 분할 기법 등이 있다. 이중 조기 분할 기법은 많은 환경에서 적용되어 실험되지 않았다. 여기서는 조기 분할 기법을 VLIW용 시뮬레이터상에서 구현하여 그 성능을 살펴보도록 하겠다.

## 2. 그래프 컬러링 기반의 레지스터 할당 기법

그래프 컬러링이란 선에 의해 연결된 여러 개의 점들에 색을 칠할 때 하나의 선에 의해 연결된 두 개의 점이 같은 색을 가지지 않도록 하여 모든 점에 색을 매기는 문제를 이야기 한다. 그래프 컬러링 기반의 레지스터 할당 기법에서 하나의 점은 프로그램에서 하나의 가상 레지스터의 live range를 뜻하고, 칠할 수 있는 색의 수는 머신의 실제 레지스터의 개수를 뜻한다. Live range가 포개진 두 개의 가상 레지스터는 간섭한다고 말하고 이 간섭하는 두 개의 가상의 레지스터는 그래프상에서 선으로 연결된 것으로 표현된다.

k개의 색으로 그래프 G를 색칠하는 것은 NP-complete어서 휴리스틱 알고리즘이 필요하게 된다. Chaitin et al.은 다음의 정리를 가지고 그래프 컬러링 기반의 레지스터 할당 기법을 구현했다.[Cha82]

**K개보다 적은 연결된 점을 가지는 점 x는 G-{x}의 컬러링 가능 여부에 상관없이 항상 컬러링이 가능하다.**

X와 같은 점을 낮은 정도의 점이라고 하고, K개 혹은 그 이상의 연결된 점을 가지는 점은 상당한 정도의 점이라고 한다. Chaitin의 할당 기법은 그래프에서 반복적으로 낮은 정도의 점을 제거해서 그 점을 스택에 넣는 것을 반복한다. 그래프가 비게 되거나 남은 점들은 모두 상당한 정도의 점으로 구성되어 더 이상 제거할 수 있는 점이 없다면 멈추게 된다. 이것을 ' 단순화' 과정이라고 한다. 그래프가 비었다면 스택에서 쌓여있는 점들을 꺼내서 색을 칠하게 된다. 이 때, 이미 칠이 된 연결된 점들의 색과는 다른 색을 칠한다. 이 과정을 ' 선택' 과정이라고 부른다. 반면, 그래프가 비지 않았지만, 모두 상당한 정도의 점들이어서 더 이상 단순화할 수 없다면, 하나의 점을 골라서 ' 스필' 을 한다. 이 ' 스필' 단계에서는 골라진 점의 live range에서 사용과 정의에 load과 store 명령어를 넣는다. 그리고, 다시 레지스터 할당을 처음부터 다시 실행하게 된다. 그림 1-1은 chaitin의 레지스터 할당 기법의 흐름도이다. ' live range 구하기' 단계는 가상 레지스터의 live range를 계산하는 단계이고, ' 구축' 단계는 간섭 그래프를 그리는 단계이다. (' 적극적 융합' 단계는 다음 장에서 설명한다.)

스필이 되는 점의 수를 줄이기 위해 Briggs는 ' 낙천적 컬러링' 을 제안했다.[BCT94] 모든 점들이 상당한 정도를 가지게 되면 골라진 점을 스필을 시키지 않고 대신 스택에 쌓는다. 이것을

잠재적 스필이라고 부른다. 이후의 선택 단계에서 이 점에 색을 칠할 수 없다면 실제적 스필을 실행하게 된다. 이것은 상당한 정도를 가지는 점이라도 주변의 점들의 선택된 색에 따라서 때때로 겹치지 않는 색을 칠 할 수 있기 때문이다. 그림 1-2는 briggs의 낙천적 컬러링 기법의 흐름도이다.

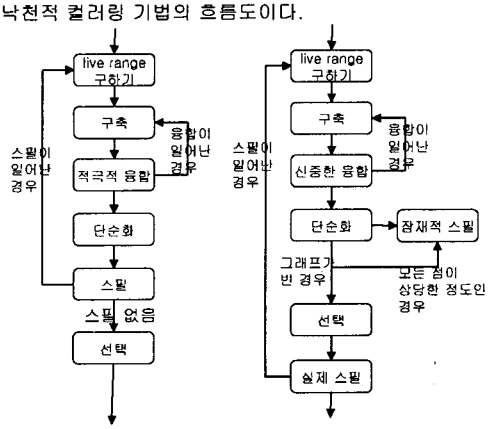


그림 1-1 chitin의 할당 기법 그림 1-2 Briggs의 할당 기법

## 3. 레지스터 융합 기법

레지스터 할당 기법에서 copy명령어의 소스 가상 레지스터와 목적 가상 레지스터를 같은 실제 레지스터에 할당하여 copy를 제거할 수 있다. 그래프 컬러링에서는 두 개의 점을 하나로 통합시킴으로써 하나의 색으로 컬러링할 수 있다. 이런 과정을 융합이라고 부른다.

융합은 긍정적인 면과 부정적인 면 모두를 가진다. 융합된 점은 원래의 두 점이 간섭하던 모든 점과 간섭하게 된다. 따라서, K개의 색으로 컬러링이 가능하던 그래프를 컬러링이 불가능하도록 바꿀 수도 있다. 반면에, 융합된 두 점과 모두 간섭하던 점이 있다면 이 점은 연결된 점이 하나 줄게 된다. 따라서, 이 점은 단순화될 가능성이 더 높아질 수 있다.

### 3.1 적극적 융합 방식

chaitin의 할당 방식에서 사용된 적극적 융합 방식은 copy만으로 연결된 모든 점들을 융합시키게 된다. 따라서, copy를 최대한 제거할 수 있다. 하지만, 많은 점들이 원래보다 많은 이웃한 점을 가지게 되어 스필이 되는 점들이 많아져서 load와 store의 추가가 많아지게 된다.

### 3.2 신중한 융합 방식

적극적 융합 방식의 문제점 때문에 Briggs는 신중한 융합 방식을 택하게 된다. 융합을 하더라도, 그래프의 컬러링에 문제가 생기지 않도록 한다는 것이다. 어떤 그래프 G에서 융합된 점과 연결된 점 중에서 상당한 정도를 가지는 점의 수가 k개 미만인 경우에만 융합하도록 하면, k개의 색으로 컬러링되는 그래프는 융합된 뒤에서 k개의 색으로 컬러링 할 수 있다.

### 3.3 반복적 융합 방식

반복적 융합 방식은 신중한 융합 방식을 기본으로 하되 단순화 단계와 융합 단계를 결합하여 단순화 단계에서 낮은 정도의 점들을 미리 제거하여 copy에서 사용되는 가상 레지스터와 간섭하는 상당한 정도를 가지는 점의 수를 줄여서 더 많은 copy를 제거하는 방식이다.[GA96]

### 3.4 낙천적 융합 방식

적극적 융합 방식은 copy를 최대한 제거하였지만 융합된 스필에 대해서는 신경을 쓰지 않았다. 신중한 융합 방식과 반복적 융합 방식은 융합된 점의 스필은 잘 막았지만, 안전한 융합을 위해서 많은 기회를 놓치고 있다. 가능한 많은 copy를 제거하면서도 융합된 점의 스필은 가능한 줄이는 낙천적 융합방식을 박진표는 제안했다.[Park04] 적극적 융합 기법을 사용하여 가능한 많은 copy를 제거하도록 노력한 뒤에, 실제 선택 단계에서 융합된 점이 스필이 될 것으로 결정된다면, 융합을 취소하여 각각의 점들에 색을 칠하거나 융합이 취소되어도 색을 칠할 수 없다는 스필을 하게 된다. 그림 2는 낙천적 융합 기법을 사용한 낙천적 할당 기법의 흐름도이다.

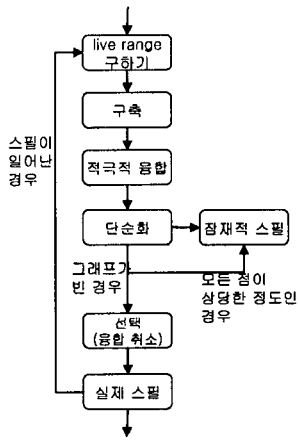


그림 2 낙천적 융합을 사용한 낙천적 할당 기법

## 4. 잘못된 스필 비용 문제

낙천적 융합 방식은 효율적인 융합 방식이지만 여전히 문제점은 있다. 그래프가 상당한 정도의 점들만이 남았을 때 잠재적 스필로 선택되는 점은 스필 비용이 가장 싼 점이다.

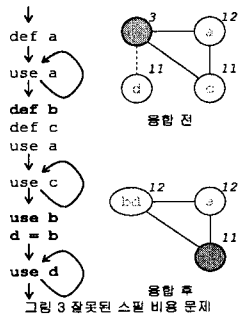


그림 3 잘못된 스필 비용 문제

하지만, 그림 3과 같은 경우에는 실제 스필 비용이 가장 싼 점 b가 선택되지 않고 c가 선택된다. 이것을 잘못된 스필 비용 문제라고 하자.

### 5. 조기 분할 레지스터 융합 기법

잘못된 스필 비용 문제는 두 개의 점이 융합되었을 때, 융합된 점도 하나의 일반적인 점으로 취급되어 잠재적 스필 선택의 대상이 된다는 점이다. 이런 문제는 적극적 융합 기법과 낙천적 융합 기법에서만 발생된다. 조기 분할 융합 기법은 낙천적 융합 기법의 잘못된 스필 비용 문제를 해결하려고 제안되었다.[Park03]

낙천적 융합 기법과 조기 분할 융합 기법의 가장 큰 차이는 융합 취소가 일어나는 순서상의 위치이다. 낙천적 융합 기법에서는 실제 스필이 일어날 때 수행하지만, 조기 분할 융합 기법에서는 잠재적 스필에서 수행하게 된다.

잠재적 스필 단계에서 상당한 정도의 점들 중에 융합된 점이 있다면, 융합된 점의 구성 점들을 나눠서 스필 비용을 산출한다. 일반적인 점이 선택된다면 상관없지만 융합된 점이 선택된다면, 융합을 취소하여 선택된 점을 스필하고 분리된 나머지 점은 다시 그래프에 넣어서 단순화를 진행시키게 된다. 그림 3의 경우를 보면 잠재적 스필 단계에서 융합 전의 그래프를 상정하여 b가 선택되면 융합된 점 bd를 분리해서 b는 잠재적 스필로 스택에 쌓고, d는 그래프에 다시 넣게 된다.

그림 4는 조기 분할 레지스터 융합 기법의 흐름도이다. 흐름도에서 부분 단순화는 그래프가 같은 스필 비용을 가지는 융합된 점들로 이루어질 경우에 발생하게 된다.

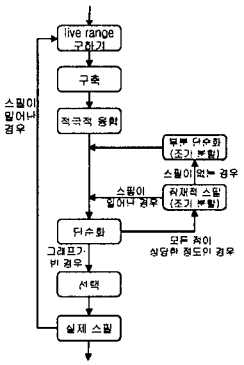

그림 4 조기 분할 레지스터 할당 방식

### 6. 실험 환경

실험 환경은 SPARC 명령어 모음을 사용한 VLIW 시뮬레이터에서 실행했으며, 적극적인 명령어 스케쥴링을 통해 많은 copy를 생성한 코드를 실험 대상으로 한다. 시뮬레이터는 16-ALU와 8-way branching을 가지며, 32개의 일반 레지스터와 16개의 condition 레지스터를 가진다. 캐쉬 미스는 발생하지 않는다고 가정하며, 명령어는 모두 1cycle에 수행된다.

표 1 실험의 비교 대상

| 휴리스틱 | 단순화 방식 | 융합 방식 | 융합 취소 여부 |
|---|---|---|---|
| 기본 | chaitin | 적극적 | 없음 |
| 신중한 | 낙천적 | 신중한 | 없음 |
| 반복적 | 낙천적 | 반복적 | 없음 |
| 적극적 | 낙천적 | 적극적 | 없음 |
| 낙천적 | 낙천적 | 적극적 | 실제 스필단계 |
| 조기 분할 | 낙천적 | 적극적 | 잠재적 스필단계 |

벤치마크는 모두 7개로 eqntott, espresso, sed, compress, yacc, sed, gzip, li을 사용한다. 실험을 통해 비교되는 대상은 표 1과 같다.

### 7. 실험 결과

표 2 기본 휴리스틱과 비교한 제거한 카피의 비율(%)

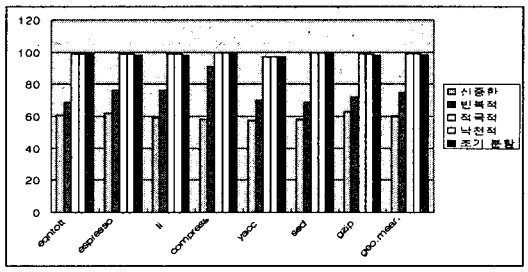


표2에서 낙관적 융합 기법에 비해서 조기 분할 기법은 좀 더 많은(약 0.5%) copy 명령어를 남김을 볼 수 있다. 반면에 스필 명령어의 개수는 약 0.1%정도 적게 나왔다. 이것은 예측과는 전혀 다른 결과로, 이전의 연구[Park03]에서는 조기 분할 기법은 낙관적 기법에 비해 0.5%정도 많은 copy를 남기긴 했지만, 스필 명령어의 개수는 6%정도 적게 생성했었다. 좀 더 많이 남은 copy 명령과 큰 차이 없이 생성된 스필 명령어의 영향으로 수행 cycle수 또한 비슷하게 나타나고 있다.

### 8. 결론

조기 분할 융합 기법은 낙관적 융합 기법에 비해서 좀 더 많은 copy명령어를 남기는 대신 비싼 비용의 스필이 일어나는 것을 가능한 줄이고자 노력했다. 실제로 copy는 조금 더 남기지만, 스필 명령어의 개수는 별 차이를 보이지 못하고 있다. 이것은 기법이 적용된 실험 환경의 변화에 의한 것으로 보인다. 이전의 연구[Park03]에서도 낙관적 기법은 시뮬레이터 환경에서는 모든 기법들 보다 좋은 결과를 보였지만, gcc환경에 적용했을 때는 다른 기법들에 비해 나쁜 결과를 보인 적이 있다.

### 참고문헌

[Cha82] G. Chaitin. Register Allocation and Spilling via Graph Coloring. In Proceedings of the ACM SIGPLAN 1982 Symposium on Compiler Construction, pages 201-207, 1982

[BCT94] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to Graph Coloring Register Allocation. ACM Transactions on Programming and Systems, Vol 16, No.3, pages 428-455, May 1994

[Park04] Jinpyo Park and Soo-mook Moon. Optimistic Register Coalescing. ACM Transactions on Programming Languages and Systems, Vol. 26, No. 4, Pages 735-765, July 2004

[GA96] L. George and A. W. Appel. Iterated register coalescing. ACM Transactions on Programming Languages and Systems, Vol 18, No. 3, pages 300-324, May 1996

[Park03] Jinpyo Park. A Study on Data Movement Elimination. 서울대학교. Feb 20003.