제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

지정 레지스터 수 증가 최소화와 노드 병합을 이용한 레지스터 할당

박승진 0* 한경숙** 표창우*
홍익대학교 컴퓨터공학과*, 한국산업기술대학교 컴퓨터공학과**
sjpark@cs.hongik.ac.kr0*, khan@kpu.ac.kr**, pyo@cs.hongik.ac.kr*

Register Allocation Minimally Incrementing the Number of Assigned Registers and Using Node Merging

Seung-jin Park0* Kyung-sook Han** Changwoo Pyo*
Dept. of Computer Engineering, Hongik University* & Korea Polytechinc University**

요 약

노드 병합을 이용한 레지스터 할당 방법은 그래프 감축 단계에서 블록 되었을 경우 효율적인 비용 계산을 이용하여 그래프 감축이 지속될 가능성을 발생시키는 방법이다. 이와 함께 지정 레지스터 수의 증가를 최소화하는 레지스터 할당 방법은 컬러링 과정에서 좀 더 적은 수의 레지스터를 사용하도록 하기 위하여 제안된 방법이다. 이 두 가지 알고리즘을 함께 적용한 경우 기존의 레지스터 할당 알고리즘 보다 우수한 결과를 보였다. Appel 의 간섭 그래프들을 사용하여 제시된 레지스터 할당 방법과 Briggs 의 알고리즘을 비교할 때 500 개 이상의 에지를 포함하는 그래프중에 5.81%의 그래프에서 레지스터 요구 수가 감소되었다. 제시된 알고리즘은 코드 길이가 길거나 사용가능한 레지스터 수가 적은 경우에 좋은 성능을 가져올 것으로 예측한다.

1. 서 론

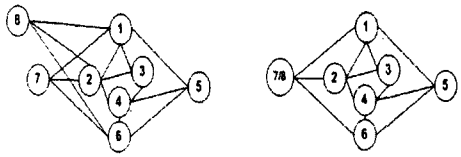
그래프 컬러링 문제는 NP-complete문제로[1] Chaitin의 레지스터 할당 방법[2, 3]을 바탕으로 그래프 감축 단계와 컬러링 단계에 관한 연구가 현재까지 진행되고 개발되어 왔다. 본 논문에서는 Chaitin 이후로 개선된 Briggs의 레지스터 할당 방법[4]과 노드 병합을 이용한 레지스터 할당 방법을 기반으로 하여 지정 레지스터 수의 증가를 최소화하는 레지스터 할당 방법을 제시하였으며 이에 그래프 감축이 지속될 가능성을 발생시키는 노드 병합 방법을 함께 적용한 알고리즘을 제시하였다.

지정 레지스터 수의 증가를 최소화하는 레지스터 할당 방법은 컬러링 과정에서 좀 더 적은 수의 레지스터를 사용하도록 하기 위하여 제안된 방법이다.

본 논문의 구성은 다음과 같다. 2절에서는 노드 병합에 대해 알아보고, 3절에서는 노드 병합 이후 지정 레지스터 수 최소화를 이용한 레지스터 할당 알고리즘에 대해 설명한다. 4절에서는 실험을 통하여 본 논문에서 제시한 레지스터 할당 방법의 우수성을 증명하였으며, 5절에서는 결론 및 향후 연구 방향을 기술하였다.

2. 관련 연구

노드 병합을 이용한 레지스터 할당 방법은 그래프 감축 단계에서 더 이상 감축 가능한 노드가 존재하지 않는 경우 비용 에 의해 두 개의 노드를 병합시킨다[5]. 이에 의해 그래프 감축 단계 수행 중에 병합하는 두 노드와 동시에 간섭하는 노드의 간섭 수를 줄여 그래프 감축이 지속될 가능성을 발생시키는 방법이다. 노드 병합에 의해 병합된 두 노드는 컬러링 단계에서 같은 컬러를 할당 받으며, 사용 가능한 실제 레지스터 수보다 간섭 수가 작은 노드가 더 이상 존재하지 않는 경우 병합에 의해 간섭 수를 줄이고자 하는 것이다. 노드 병합을 이용한 간섭 수 감소의 예를 <그림 1>과 <그림 2>를 통하여 살펴본다. <그림 1>의 간섭 그래프에서 간섭 수를 가장 최소화할 수 있는 두 노드는 노드 7과 노드 8이다. 노드 7은 노드 (1, 2, 6) 과 간섭하며 노드 8도 노드 (1, 2, 6) 과 간섭한다. 이 두 노드를 병합 시킬 경우 <그림 2>의 간섭 그래프와 같이 총 간섭 수는 15개 에서 12개로 줄여 그래프 감축이 지속될 가능성을 발생시킨다.



<그림 1> 노드 7과 노드 8의 병합 전 그래프 <그림 2> 노드 7과 노드 8의 병합 후 그래프

3. 지정 레지스터 수의 증가를 최소화하는 레지스터 할당

간섭 그래프에서 간섭 관계에 있는 노드들은 서로 같은 컬러를 할당 받지 못한다. 즉, 같은 시점에 생존하는 변수들은 간섭 관계에 있으며 생존 범위가 큰 변수는 생존 범위가 작은 변수보다 더 많은 변수들과 동시에 생존할 확률이 높아지므로 다른 노드들과 같은 컬러를 할당받을 가능성이 적어지게 된다. 따라서 하나의 컬러를 할당 받은 노드들의 간섭 관계를 병합한 간섭 관계를 가지고 컬러를 할당 받아야 할 노드에 컬러의 할당 여부를 결정하게 된다.

기존의 레지스터 할당 방법은 그래프 감축 단계 진행 후 노드의 컬러링 순서가 정해지면, 할당 대상이 되는 레지스터 개수 내에서 할당 가능한 컬러를 결정한다. 컬러를 할당 받는 시점에서의 노드는 할당 받을 수 있는 컬러의 수가 다수일 경우도 있고, 한 개일 경우, 또한 없을 경우도 존재한다. 이때 할당 받을 수 있는 컬러가 여러 개일 경우 어떤 컬러를 할당 받느냐에 따라 다음에 컬러를 할당받을 노드들의 컬러링이 가능할 수도 있고 불가능할 수도 있다. 이 논문에서 제시된 레지스터 할당 방법은 기존의 레지스터 할당 방법보다 같은 레지스터 개수에 좀 더 많은 노드를 할당함으로써 대피 코드의 수를 감소시키기 위한 방법으로 이미 컬러를 할당 받은 노드들의 간섭 관계를 병합한 간섭 관계에서 간섭 수의 증가를 고려한 지정 레지스터 수의 증가를 최소화할 수 있는 방법이다.

예를 들어 <그림 2>의 노드 병합 이후 간섭 그래프를 컬러링 한다고 가정한다. 이 간섭 그래프를 3-컬러링 하는 과정을 통해 지정 레지스터 수 증가 최소화를 통한 레지스터 할당 방법을 볼 수 있다. <그림 3>에서 사용하는 용어와 그 의미는 다음과 같다.

C : 컬러, n : 노드

adjacent(n) : 노드 n과 간섭 관계에 있는 노드들의 집합

$degree(n) = |adjacent(n)|$

color(n) : 노드 n에 할당된 색

$$interf(C) = \bigcup_{color(n) = C} adjacent(n)$$

$$uinterf(C, n) = interf(C) \cup adjacent(n)$$

$$interfNo(C) = |interf(C)|$$

$$incInterNo(C, n) = |Uinterf(C, n)| - |interf(C)|$$

adjacent(n)은 노드 n과 간섭 관계에 있는 노드들의 집합으로 노드 n 과 같은 컬러를 할당받을 수 없는 노드의 집합을 말한다. degree(n)은 노드 n의 간섭 수로 노드 n과 간섭하는 노드들의 수를 나타내며 color(n)은 노드 n에 할당된 색이다. interf(C)는 컬러 C와 간섭 관계에 있는 노드들의 집합으로, 컬러 C를 할당받은 모든 노드들의 간섭 관계를 통합한 것이다.

interf(C)에 포함되어 있는 노드들은 컬러 C를 할당받은 노드 중에 간섭 관계에 있는 노드가 존재하는 것이므로 컬러 C를 할당받을 수 없다. 컬러가 정해지지 않은 상태에서 계산하는 Uinterf(C, n)는 노드 n에 컬러 C를 할당했을 경우 컬러 C와 간섭 관계가 되는 노드들의 집합이다. 즉 컬러 C에 포함되어 있는 노드들과 간섭 관계에 있거나 노드 n과 간섭 관계에 있는 노드들의 합집합으로 볼 수 있다. interfNo(C)는 interf(C)에 포함되어 있는 노드의 수로 컬러 C의 간섭 수를 나타내며 incInterfNo(C, n)는 노드 n에 컬러 C를 할당했을 경우 컬러 C의 간섭 수 증가분을 나타낸다. 본 논문에서는 노드 병합 이후 지정 레지스터 수 증가를 최소화하기 위해 간섭 수 증가분을 최소화하는 컬러를 선택하므로 노드를 컬러링 하는 단계에서 incInterfNo(C, n) 값을 최소로 하는 컬러 C를 선택하게 된다.

<그림 3>은 노드 병합 후 지정 레지스터 수 증가 최소화를 이용한 레지스터 할당의 예를 보여주고 있다. (a)에서는 <그림 2>의 간섭 그래프에 대한 각 노드의 간섭 관계를 보여주고 있다. 그래프 감축 단계에서 감축 스택에 삽입되는 순서는 1, 3, 2, 4, 5, 6, 7/8 이다. 노드들의 삽입된 순서에 따라 (b) ~ (e)는 그래프 컬러링하는 단계를 보여주고 있다. 감축 스택에 삽입된 역순서로 컬러를 할당하는 과정을 살펴보자. 노드 7/8은 어떤 컬러를 할당받아도 각 컬러의 간섭 수 증가분을 계산하면 각각 컬러에 대해 간섭 수가 3이 된다. 즉, 어떤 컬러를 할당 받아도 상관없다. 노드 7/8에 컬러 R을 할당하면

interf(R) = adjacent(7/8) = {1, 2, 6}, interfNo(R) = 3

이 된다. 노드 7/8의 컬러가 정해졌으면, 다음으로 노드 6의 컬러를 결정한다. 노드 6은 노드 7/8과 간섭하므로 노드 7/8이 할당받은 컬러 R을 제외한 컬러 G와 컬러 B중에서 할당 받을 수 있다. 노드 6의 컬러 G와 컬러 B에 대한 간섭 수는 같으므로 컬러 G를 할당했다고 가정하면 노드 5의 컬러링 시점에서 노드 6과 간섭하므로 노드 7이 할당받은 컬러인 R과 할당되지 않은 B중 어느 컬러를 할당할지 선택해야 한다. 노드 5에 컬러를 할당하는 시점에서 컬러 R을 할당하는 경우와 컬러 B를 할당하는 경우에 따라 전체 그래프가 요구하는 컬러 수 즉, 레지스터 수에 영향을 미친다. 레지스터 요구 수가 많아지면 대피 코드를 삽입하는 경우도 많이 발생하여 메모리 접근도 많아지므로 프로그램의 실행 성능을 저하시킨다.

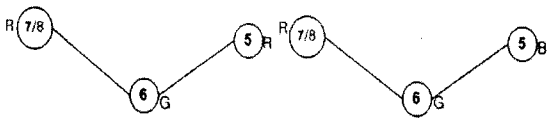
본 논문에서 제안하는 알고리즘은 각 노드에 컬러를 할당하는 시점에서 이미 컬러에 할당된 노드들의 간섭수와 할당하고자하는 노드의 간섭수 interfNo(C) 를 계산한다. 그 후 간섭수에 대한 간섭수 증가분을 최소화하는 컬러 incInterfNo(C, n)를 찾아내어 컬러 C를 선택하게 된다.

<그림 3>의 (b)와 (b)'에서 노드 5에 컬러 R과 컬러 B를 할당할 경우 간섭 수와 간섭 수 증가분을 계산하면 다음과 같다.

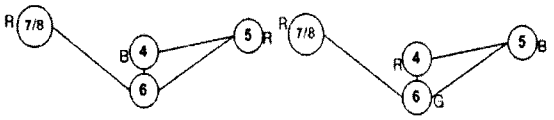
interf(R) = adjacent(7/8) = {1, 2, 6},
interfNo(R) = | interf(R) | = 3
interf(B) = { },
interfNo(B) = | interf(B) | = 0
< 노드 5에 컬러 R을 할당하는 경우 >
Uinterf(R, 5) = interf(R) ∪ adjacent(5) = {1, 2, 4, 6}
incInterfNo(R, 5) = | Uinterf(R) | - | interf(R) | = 1
< 노드 5에 컬러 B를 할당하는 경우 >
Uinterf(B, 5) = interf(B) ∪ adjacent(5) = {1, 4, 6}
incInterfNo(B, 5) = | Uinterf(B) | - | interf(B) | = 3

adjacent(1) = {2, 3, 5, 7/8} degree(1) = 4
adjacent(2) = {1, 3, 4, 7/8} degree(2) = 4
adjacent(3) = {1, 2, 4} degree(3) = 3
adjacent(4) = {2, 3, 5, 6} degree(4) = 4
adjacent(5) = {1, 4, 6} degree(5) = 3
adjacent(6) = {4, 5, 7/8} degree(6) = 3
adjacent(7/8) = {1, 2, 6} degree(7/8) = 3

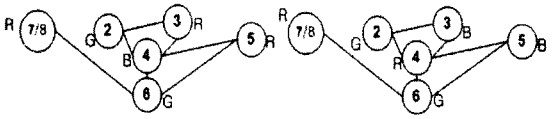
(a) 간섭 그래프



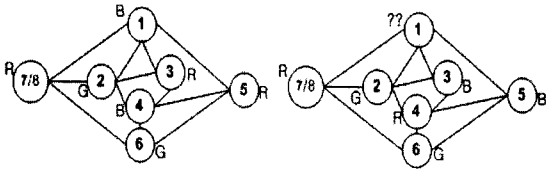
(b) 노드 5 컬러링 (b)' 노드 5 컬러링



(c) 노드 4 컬러링 (c)' 노드 4 컬러링



(d) 노드 2-> 노드 3 컬러링 (d)' 노드 2->노드 3 컬러링



(e) 노드 1 컬러링 (e)' 노드 1 컬러링

<그림 3> 노드 병합 이후 간섭 수 증가 최소화를 이용한 그래프 컬러링의 예

위의 계산결과를 보면 컬러 R을 할당할 경우 incInterfNo(R, 5) 는 1만큼 증가하고, incInterfNo(B, 5) 는 컬러 B를 할당할 경우 처음 할당되므로 3만큼 증가하게 된다. 간섭 수가 증가할 경우 앞으로 컬러링될 노드들이 해당 컬러를 받을 가능성이 줄어들게 된다.

<그림 3>의 (c) ~ (e)는 노드 5에 컬러 R을 할당한 경우 이후 과정을 나타냈고, (c)' ~ (e)'는 노드 5에 컬러 B를 할당한 경우 이후 과정을 도시화하였다. 노드 5에 컬러 B를 할당 한 경우에 (g)'에서는 할당할 색이 존재하지 않음을 알 수 있다.

4. 성능평가

지정 레지스터 수 증가 최소화와 노드 병합을 이용한 레지스터 할당 방법을 구현하여 최소 레지스터 수를 산출하였다. 위의 산출 방법은 Appel[6]의 그래프 데이터베이스를 이용하여 실험한 결과이다. Appel의 간섭 그래프 데이터 베이스는 53개의 빈 그래프와 더불어 10,000개 이상의 간섭 수를 가진 그래프는 67개로 그래프 크기의 변화가 매우 다양하며, 100개 미만의 간섭 수를 포함한 그래프가 22,825개로 많은 간섭그래프들이 소량의 간섭 수를 포함하고 있다.

[표 1]에서 지정 레지스터 수 증가 최소화와 노드 병합을 이용한 알고리즘은 DegInc/Merge로 표시하였으며 지정 레지스터 수 증가를 고려한 알고리즘은 DegInc로 표시했다. 실험 결과 Briggs의 레지스터 할당 방법과 DegInc/Merge를 비교할 때 1.08%의 그래프에서 레지스터 요구 수가 감소하는 것을 볼 수 있었다. Briggs의 레지스터 할당 방법과 DegInc와 비교한 결과를 보면 1.02%로 지정 레지스터 수 증가만을 고려한 알고리즘보다 노드 병합 방법까지 적용 시킨 알고리즘이 레지스터 요구 수를 더 감소시킴을 알 수 있었다.

[표 1]의 큰 그래프는 500개 이상의 간섭 수를 포함하고 있고, Appel의 간섭 그래프 데이터 베이스에서 큰 그래프는 1170개로 구성되어 있다. Briggs의 레지스터 할당 방법과 DegInc/Merge를 비교한 결과 5.81%의 큰 그래프에서 레지스터 요구 수가 감소했고, Briggs의 레지스터 할당 방법과 DegInc를 비교한 결과 6.0%의 큰 그래프에서 레지스터 요구 수를 감소시켰다.

[표 1] 알고리즘에 따른 레지스터 요구수

레지스터 요구수	그래프 수 및 비율	큰 그래프 수 및 비율
Briggs > DegInc/Merge	301 (1.08%)	68 (5.81%)
Briggs = DegInc/Merge	27,608 (98.88%)	1,095 (93.59%)
Briggs < DegInc/Merge	12 (0.04%)	7 (0.6%)
Briggs > DegInc	284 (1.02%)	70 (6.0%)
Briggs = DegInc	27,588 (98.8%)	1,078 (92.1%)
Briggs < DegInc	49 (0.18%)	22 (1.9%)

Briggs의 알고리즘보다 레지스터를 적게 요구하는

이유는 병합에 의해 간섭 수를 줄이고, 할당 가능한 컬러 중 간섭 수를 적게 증가시키는 컬러를 할당하기 때문이다. 노드 병합을 이용하여 간섭 수와 노드 수를 줄인 다음 간섭 수가 적게 증가되는 컬러를 할당하게 되면 현재 사용되고 있는 컬러에 좀 더 많은 노드를 포함할 가능성이 증대되어 적은 수의 컬러로 컬러링이 가능해질 수 있게 된다.
[표 1]의 큰 그래프에서 레지스터 요구 수가 증가한 결과를 보면 DegInc/Merge는 DegInc보다 1.3%의 그래프에서 레지스터 요구 수가 적게 증가함을 알 수 있었다. 이는 노드 병합을 이용한 결과 간섭 수와 노드 수에 변화를 가져왔기 때문이라고 추측할 수 있다.

5. 결론 및 향후과제

본 논문에서는 실제 레지스터를 할당하는 단계에서 레지스터 선택 방법을 제안하였으며 이에 노드 병합을 함께 적용하여 그래프 감축 단계에서 감축할 가능성을 높임으로써 보다 효과적인 레지스터 할당방법이 적용되었다.

지정 레지스터 수 증가 최소화와 노드 병합을 이용한 레지스터 할당방법은 Appel의 간섭그래프 데이터베이스[6]를 이용하여 실험하였으며, 그 결과 기존의 알고리즘보다 레지스터 요구 수를 적게 요구함으로써 성능 향상을 가져왔다.
앞으로 그래프 감축 단계에서 보다 효과적인 감축 순서에 대한 연구가 이루어져야 하며 실제 컴파일러에 적용하여 성능 실험도 이루어져야 할 것이다.

6. 참고문헌

[1] NARSINGH DEP, "Graph Theory with Applications to Engineering and Computer Science", Prentice-Hall, 1974
[2] Gregory J. Chaitin, Mark A. Auslander, Ashok K. Chandra, John Coke, Martin E. Hopkins, Peter W. Markstein, "Register Allocation via Coloring", Computer Languages, 1981
[3] Gregory J. Chaitin, "Register Allocation & Spilling via Graph Coloring", Proceedings of the ACM SIGPLAN '82 Symposium on Compiler Construction, 1982
[4] Preston Briggs, Keith D. Cooper, Ken Kennedy, Linda Torczon, "Coloring Heuristics for Register Allocation", Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implementation, 1989
[5] Steven R. Vegdahl, "Using Node Merging to Enhance Graph Coloring", PLDI, 1999
[6] Andrew W. Appel, A Sample Graph Coloring Problems,
URL:http://www.cs.princeton.edu/fac/appel /graphdata, 1996