그래프 중간표현 형태를 기반으로 한 병렬 프로그래밍 환경

A Parallel Programming Environment using Graph Type Intermediate Representation Form

이 원 용* 박 두 순**
Won-Yong Lee Doo-Soon Park

요 약

본 논문에서는 사용자의 병렬 프로그램 작성을 도와주는 병렬 프로그래밍 환경을 제공한다. 병렬 프로그램은 다양한 하드웨어의 특성에 따라 또는 프로그램의 특성에 따라 사용자가 병렬 프로그램을 작성하여야 하기 때문에 프로그래머는 병렬 프로그램을 작성하는 것이 매우 어렵다. 본 논문에서는 사용자가 쉽게 프로그램을 작성할 수 있도록 하기 위하여 많은 병렬화 연구에서 제시되고 있는 그래프 중간 표현 형태를 그래프 사용자 인터페이스로 구현하였다. 이 병렬 환경에서는 프로그램 편집기능, 종속성 분석기능, 루프 변환기능, CFG, PDG, HTG 등 중간 코드를 그래프 중간 표현 형태를 통해 보여 줌으로 사용자에게 병렬화, 최적화 작업에 용이하도록 한다.

Abstract

This paper describes a parallel programming environment to help programmer to write parallel programs. Parallel program must be write according to the character of the various hardware or program. So it is difficult for the programs to write the parallel programmer. In this paper, we propose and implement a parallel programming environment using graph type intermediate representation form, and graph user interface is provided for programmer to get parallel programs easily. This parallel environment supports special functions using graph type intermediate representation form. The special functions involve program editing, data dependence analysis, loop transformation, CFG, PDG, HTG. This parallel environment helps users make parallelism and optimization easy through showing the intermediate code with graph.

1. 서 론

정보산업의 발전에 따라 컴퓨터의 응용분야는 넓어지고 컴퓨터가 처리해야 할 정보의 양은 점점 방대하여 지고 있다. 이에 따라 좀더 빠른 시간에 많은 양의 정보를 처리 할 수 있도록 연구가 진행되고 있다. 이 방대한 양의 정보를 처리하기 위하여 병렬 컴퓨터가 등장하게 되었다. 그러나 병렬 컴퓨터를 사용하는 과정에서 많은 어려움이 발생되고 있다. 이런 문제를 해결하기 위해서 병렬 프로그램 작성을 쉽게 도와 줄 수 있는 여러 방법이 연구되고 있다. 이 방법은 병렬 프로그램을 작성하는 과정에서 미리 사용할 컴퓨터의 특성을 파악하여 병렬 지시어나 병렬 구조를 이용하는 방법과, 기존의 순차 프로그램을 입력하여 자동적으로 병렬 프로그램으로 바꾸어 주는 방법들이 제시되고 있다[1]. 대부분의 사용자들은 전자의 경우는 컴퓨터의 특성을 이해하고, 그 컴퓨터에 맞는 병렬 프로그램을 작성해야 되기 때문에 후자의 방법을 이용하여 병렬 프로그램을 작성하는 것이 효과적이다[2].

본 논문에서는 그래프 중간 표현 형태를 기반

* 종신회원 : 혜전대학 컴퓨터 계열 부교수
wylee@hyejeon.ac.kr
** 정 회 원 : 순천향대학교 정보기술공학부 교수
parkds@sch.ac.kr

으로 한 병렬 프로그래밍 환경을 제공한다. 기존 순차 프로그램을 입력받아 중간코드를 그래픽 인터페이스를 이용하여 그래프 형태로 사용자 정보를 제공하여 좀더 효과적이고, 효율적인 프로그램 작성을 도와주는 것을 목적으로 한다. 이를 위하여 그래픽 사용자 인터페이스를 설계, 제안하고 이에 따라 일련의 작업을 수행하면서 중간코드에서 발생되는 여러 정보를 사용자에게 그래프를 통하여 변환과정을 쉽게 분석, 이용할 수 있도록 환경을 제공한다.

이 병렬 프로그래밍 환경은 메뉴선택 방식을 이용하여 최초 사용자도 변환기 내에서 프로그램의 입력, 수정, 편집이 가능하며 기존 프로그램도 수정 작업이 가능하도록 설계하였다. 또한 프로그램을 입력받아 작업명령에 따라 문장간의 종속성 정보를 제공하며, 분석정보를 그래픽을 통하여 종속성 정보 표현한다. 그리고 이 정보를 이용하여 루프문장의 교환(Interchange), 분산(Distribution), 융합(Fusion)의 가능성 판별, 가능하면 변환 후의 프로그램을 보여준다.

따라서 사용자는 이 병렬 환경에서 제공된 정보를 이용하여 프로그램의 병렬화, 최적화에 필요한 정보를 이용할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 병렬 프로그램 및 그래프 중간 표현 형태의 연구동향을 살펴보고 종속성을 알아본다. 그리고 3장에서는 본 논문에서 제시한 그래프 중간 표현 형태로 구현한 중간 그래프의 구현방법을 기술하고 4장에서는 이를 구현하기 위한 사용자의 환경 및 구현된 정보를 보여준다. 그리고 결론에서는 본 환경에서 제시한 병렬 환경의 특징 및 우수성을 기술하였다.

2. 관련 연구

2.1 병렬 프로그래밍 환경

하드웨어의 발달에 따라 여러 프로세서가 내장된 다중 프로세서가 개발되고 이 다중 프로세서를 효과적으로 사용하기 위한 여러 방법이 연구되고 있다[2,3]. 일반적으로 병렬 프로그래밍 환경은 병렬 컴파일러를 배경으로 수행되고 있으며, 그래프 중간 표현 형태들은 병렬화나 최적화 과정에서 최적의 성능을 얻기 위한 기반으로 연구되고 있다. 병렬 프로그래밍 환경 연구동향을 살펴보면 다음과 같다.

ParafraseII는 일리노이대학에서 개발한 컴파일러로 중간코드 형태는 본 논문에서 구현한 HTG를 적용하여 구현하였다. 이 컴파일러는 100여개 이상의 코드변환 및 알고리즘을 이용하였으며 루프레벨의 혼합노드를 이용하여 함수형 병렬성을 추출하였다. 이 병렬 프로그래밍 환경에서의 사용자는 중간코드의 변환과정들의 순서와 종류를 선택하여 수행하므로 다양한 변화에 이용할 수 있도록 설계되어 있다[4].

SUIF는 스탠포드 대학에서 개발한 컴파일러 시스템의 중간언어이다. 이 시스템은 컴파일러 기능을 담당하는 모듈과 이 모듈들이 공통으로 사용하는 라이브러리 형태로 구성되어 있다[5].

PFC는 Rice 대학에서 개발한 벡터화 컴파일러로 중간코드는 추상구문 트리를 이용하여 Fortran 66 또는 Fortran 77을 이용하여 작성된 프로그램을 입력받아 병렬처리를 수행한다. 이 PFC는 루프의 향상에 중점을 두고 개발하였으며 주로 루프의 반복문 개선에 중점을 두었다[2].

Start/Pat는 조지아 공대와 캘리포니아 대학에서 개발한 병렬 컴파일로 병렬화 변환기, 동적 분석기, 성능 분석기 등의 여러 중간 언어의 기능을 제공하는 컴파일러이다[6].

2.2 그래프 중간 표현 형태를 기반으로 한 연구

병렬화와 벡터화 관련 연구들은 기존의 프로그램 텍스트에 탐지된 정보를 유지하면서 변환작업을 수행하는 방식과 그래프 중간표현 형태를 이용하여 여기에 탐지된 정보를 유지하면서 수행하는

방식이 있다. 먼저 프로그램 텍스트를 이용하는 방식은 변환과정에서 현재의 상태를 파악하기 쉽다는 장점을 가지고 있고, 그래프 형태를 유지하면서 변환하는 방식은 내부적으로 변환과정의 상태 유지를 효과적으로 할 수 있다는 장점이 있다.

이렇게 그래프 형태로 프로그램을 유지하면서 프로그램의 벡터화나 병렬화와 더불어 최적화에 이용되는 그래프 중간표현 형태에 대한 다양한 방식이 제시되고 있는데, DAG(Direct Acyclic Graph)과 HTG(Hierarchical Task Graph : 계층적 태스크 그래프), DFG(Dependence Flow Graph : 종속성 흐름 그래프), 그리고 PDG(Program Dependence Graph : 프로그램 종속성 그래프)등이 있다[7].

DAG은 프로그램 문장상의 수식에서 연산자와 피 연산자의 연결을 그래프로 표현하였다. HTG는 병렬 프로그램의 중간 형태로 자료종속성과 제어종속성을 그래프 상에서 최소화하여 표현하였다. 또 기본 블럭(basic block)의 변형인 태스크(task) 단위 또는 함수단위의 병렬성을 추출하고 표현하였다[4].

DFG는 Pingali에 의해 제시되었는데 자료 종속성 그래프와 데이터 플로우 계산 모델을 종합하여 제시한 개념이다. 즉 자료종속성 그래프 측면에서 DFG는 에지가 연산간의 종속성을 표시한다[8].

PDG는 Ferrante등에 의해 제시된 중간 그래프로 벡타화 또는 병렬화 하기에 용이한 프로그램 중간표현 형태로 병렬성의 발견이나 최적화가 용이하다는 장점 때문에 많은 연구가 진행되고 있다[9].

2.3 종속성

하나의 프로그램이 수행되는 과정은 일련의 문장들이 정해진 순서에 따라 작업을 수행하는 것으로 각 문장에서는 상수 또는 이미 정해진 변수들을 사용하여 새로운 값을 계산하고 그 결과를 변수에 저장한다. 또 그 값은 다음에 수행되는 문장에 사용되기도 한다.

변수의 수행과정에서 몇 개의 문장들은 병렬수행이 가능하나 몇 개의 문장은 반드시 임의의 순서에 따라 수행되어야 하는 경우도 있다. 후자와 같이 반드시 순서대로 수행되어야 하도록 만드는 성질을 종속성이라 한다. 따라서 두 개의 서로 다른 자료 또는 문장들이 병렬로 수행될 수 있는가는 종속성 유무에 따라 결정된다. 이 종속성은 프로그램에서 참조하는 변수 값의 사용 또는 정의하는 위치에 따라 여러 가지 형태의 종속성이 나타날 수 있다.

종속관계는 크게 두 가지로 분류하는데 하나는 자료종속성이고 다른 하나는 제어 종속성이다.

자료종속성은 두 문장 사이에서 동일한 변수에 대해 값을 정의하거나 사용하는 순서에 따라 종속성이 표현되는데, 일반적으로 두 문장 A와 B사이의 자료 종속성은 'A δ B' 형태로 표현하여 문장 B가 A에 종속됨을 나타내는데, 흐름 종속, 출력 종속, 반 종속 세 가지 유형으로 나눌 수 있다[10,11].

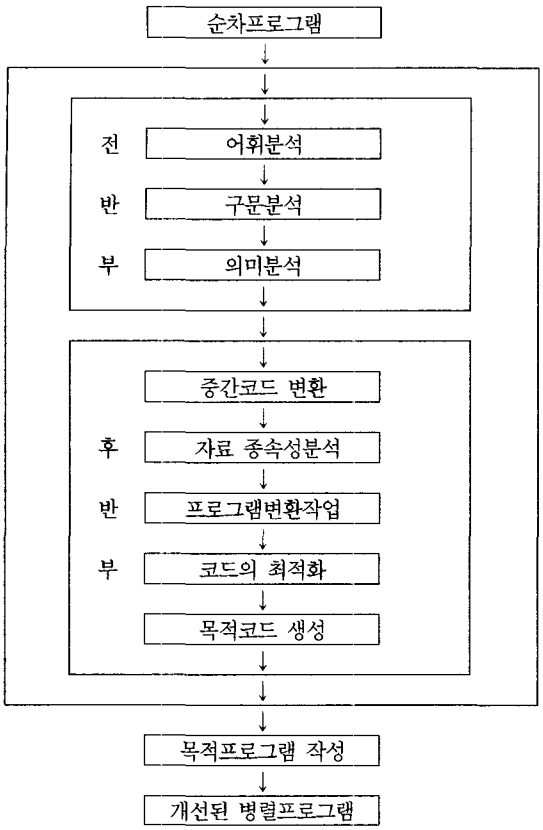
제어종속성은 프로그램 실행 중에 프로그램 명령에 의해 결정되는 흐름을 의미하는데 이 제어 종속성을 나타내기 위해서는 제어흐름그래프(control flow graph: CFG)를 이용한다. 이 제어 종속성은 포스트 도미네이터(post-dominator)개념을 이용하여 종속성을 계산한다[7].

3. 병렬 프로그래밍의 환경 구현 방법

일반적으로 병렬 프로그래밍의 환경은 그림 1에서 보는 바와 같이 어휘분석, 구문분석, 자료흐름분석, 자료 종속성 분석, 병렬화 프로그램 변환 과정을 거쳐 병렬 프로그램을 얻을 수 있다. 본 장에서는 그래프 중간 표현 형태들과 병렬화에 대해 기술한다.

3.1 기본 블록

기본 블록은 작업을 할 수 있는 문장 단위로 1개



(그림 1) 병렬프로그램 환경

의 입구점(entry point)과 1개의 출구점(exit point)을 갖는 최대 길이의 순차 문장으로 구성되며 이는 프로그램에서 노드의 역할을 한다. 기본 블록을 구현하기 위해서는 프로그램을 3-주소로 변환 후 이 코드를 기본 블록으로 변환하는 과정을 거친다[12].

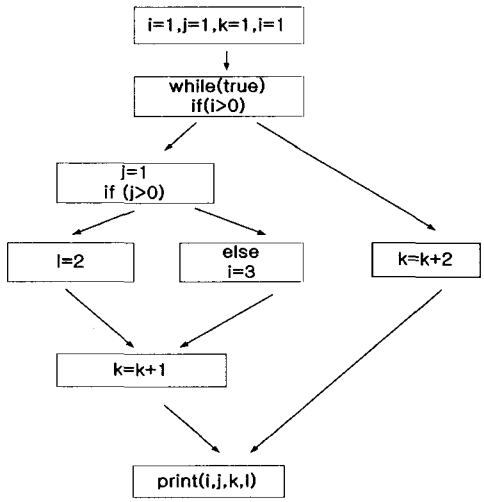
첫째 3-주소코드로 변환하는 단계로 3-주소코드는 복잡한 원시코드의 문장들을 3-주소코드 형태로 분할하거나 변화시킨 코드형태를 말한다. 3-주소코드에서 기본 블록으로 구분하여 재구성하는 과정에서 각 기본 블록의 구분 단위가 되는 첫 문장을 선도문장 이라고 한다. 이 선도문장을 찾아 그 앞의 기본의 헤더 부분을 삽입한다. 둘째 기본 블록으로 변환하는 단계인데 기본 블록으로 변화를 위해 3-주소 코드를 탐색하여 기본 블록을 찾는다. 그림 2는 기본 블록을 나타낸 것이다.

```
S0: Progrma
S1: READ(L)
S2: N = 0
S3: K = 0                          기본블록1
-------------------------------------------
S4: M=1
S5: K = K + M
S6: t1 = K > L
S7: IF t1 THEN goto S11            기본블록2
-------------------------------------------
S8: N = N + 1
S9: M = M + 2
S10: GO TO S5                      기본블록3
-------------------------------------------
S11: WRITE(N)
S12: END                           기본블록4
-------------------------------------------
```

(그림 2) 기본 블록의 표현

3.2 흐름 그래프(flow graph)

흐름 그래프는 중간코드로 표현된 프로그램의 정보를 나타내는 방향 그래프로 한 노드는 기본 블록을 의미한다[12]. 흐름그래프 G=(N, E, START, END)로 나타내는데 N은 노드의 집합이고, E는 흐름 그래프의 연결선(edge)을 나타내며, START는 노드 집합의 시작노드, END는 노드의 끝 노드를 의미한다. 일반적으로 특별한 표현이 없으면 단일 프로시져를 나타낸다. 그림 3은 흐름그래프로 나타낸 것이다.



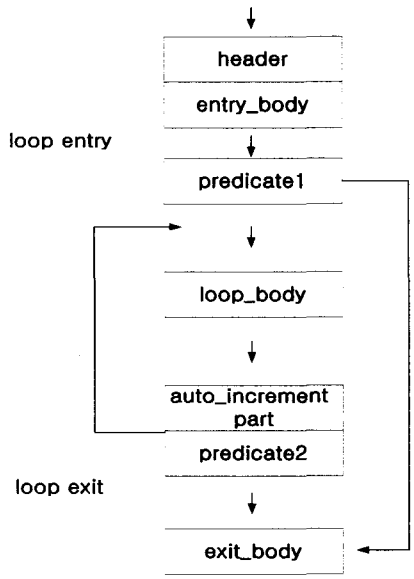
(그림 3) 흐름그래프

3.3 CFG

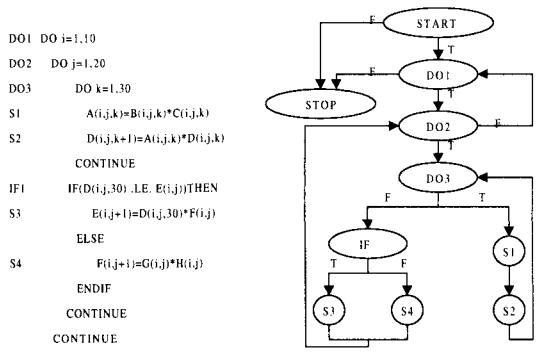
CFG는 제어흐름에 의한 프로그램의 방향성 그래프(directed graph)로서 G=(V, E) 형태로 나타내는데, 여기서 V는 노드(node)를 나타내고 노드는 프로그램의 기본 블록이나 한 문장단위를 나타낸다. E는 연결선을 나타내는데 두 노드 사이의 제어흐름을 나타낸다. 제어흐름 그래프는 한 개의 START 노드와 STOP 노드를 갖는다.

CFG는 제어 종속관계를 나타내기 위해 이용되는데 제어 종속성을 나타내기 위해서는 포스트 도미네이터(post-dominator) 개념을 이용하여 제어 종속성을 계산한다[13].

CFG는 기본 블록에서 CFG를 유도한다. 이 CFG는 루프의 존재여부에 따라 달라진다. 루프가 포함되지 않은 기본 블록들은 서로 제어 흐름에 따라 연결하면 CFG 구성할 수 있다. 그러나 루프가 포함된 구조는 반복작업을 하면서 루프 노드를 구성한다. 그림 4는 루프 노드의 처리방법을 나타낸 것이고, 그림 5는 예제 프로그램과 CFG를 나타낸 것이다.



(그림 4) 루프 노드의 처리 방법

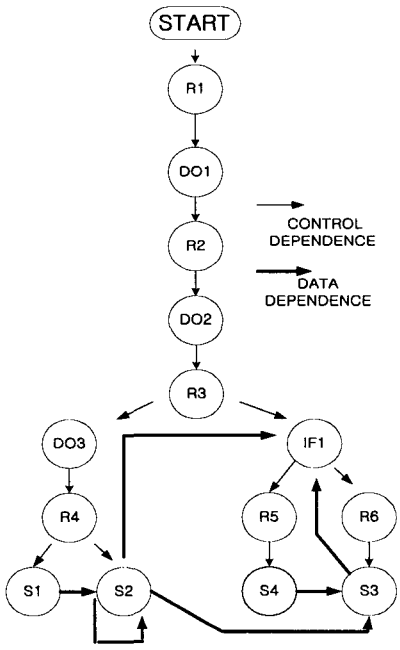


(a) 예제 프로그램 (b) 제어흐름 그래프(CFG)

(그림 5) 예제 프로그램과 CFG

3.4 PDG

PDG는 CFG로부터 유도할 수 있는데 PDG를 유도하기 위해서는 먼저 제어 종속 부 그래프와 자료 종속 부그래프를 유도하여야 한다[13,9]. 이 두 그래프를 합하여 제어종속과 자료종속을 동시에 표현한다. 그림 5의 예제 프로그램에서 PDG를 유도하기 위해서는 먼저 CFG를 유도하고 이 유



(그림 6) 그림 5(a)의 PDG 표현

도된 CFG에서 노드들간의 자료종속성 관계를 추출한 다음 기본 블록에 제어종속성을 나타내는 연결선으로 연결하여 제어종속 부 그래프를 구성한다. 이 과정에서 분기문이 포함된 노드는 후속자의 영역에 추가되어 표현한다. 이 표현은 제어종속성과 자료 종속성이 구분되는 연결선으로 표현하면 PDG가 완성된다. 그림 6은 그림 5(a)의 원시프로그램에서 유도된 PDG를 보여준다.

3.5 HTG

Girkir에 의해 제안된 HTG는 순차 프로그램으로부터 함수형 병렬성을 유도하는 중간 그래프이다. 이 그래프는 계층적 태스크 루프의 개념을 도입하고 자료 제어간의 종속성을 제거하여 중간 그래프를 완성하였다[14].

그림 7은 예제 프로그램을 나타내며 그림 8은 HTG를 나타낸 것이다. 이 HTG는 3 단계의 수직 계층구조 설명할 수 있다.

즉 최상위 단계는 4개의 노드로 구성되어 있는데 즉 루프로 구성된 노드 A, B와 단순노드인 (2), 혼합노드인 D로 구성되어 있다. 그 다음 단계에서는 B는 3개의 노드로 구성되어 있으며 기본노드인 (5), (6), 혼합노드 C로 구성되었는데 그 중 노드 C는 다시 루프 구조를 갖고 있다. 그리고 마지막 단계는 16개의 기본 블록으로 구성되어 있다. 위 그래프에서 각 계층별 노드는 그 노드 자신의 계층구조에 준하는 하나의 입구와 하나의 출구를 갖는 구조적 프로그램으로 구성되어 있다.

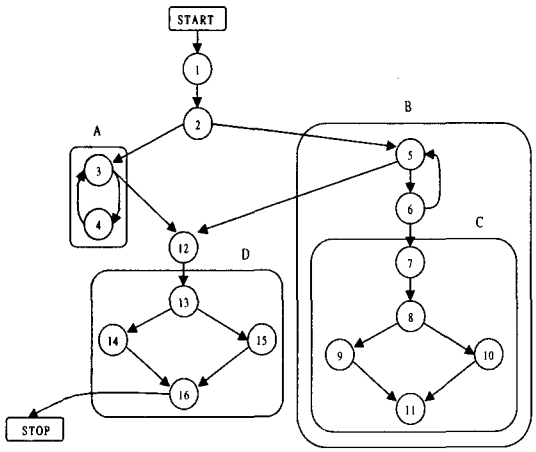
HTG는 CFG로부터 유도되는데 첫 번째 단계는 CFG(그림 9)를 DFS를 수행하여 후위경로를 추출하여 그와 관계된 경로를 계층화하여 HTG를 구축한다. 그림 10은 CFG를 DFS를 수행한 경과를 나타낸 것이다[5].

DFS를 수행 결과 4가지 경로를 발견할 수 있다.

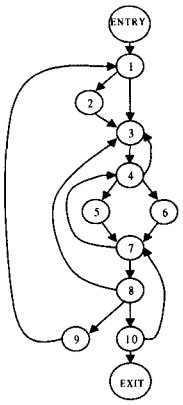
```
Begin
BB1 ----------------------------- (1)
IF C1 then ---------------------- (2)
  DO I=1, n ------------------- (3)
    BB_2 ------------------- (4)
  enddo
else
  DO i=1,m ------------------- (5)
    BB_3 ------------------- (6)
     DO j=1,k ------------------ (7)
      IF C2 then ------------- (8)
         BB_4 ------------- (9)
      else
         BB_5 ------------- (10)
      end if
      BB_6 ----------------- (11)
     end do
   end do
end if
call subroutine x ---------------- (12)
end

subroutine x
if C3 then ---------------------- (13)
   BB_7 ---------------------- (14)
else
   BB_9 ---------------------- (15)
end if
BB_9 ------------------------------ (16)
```

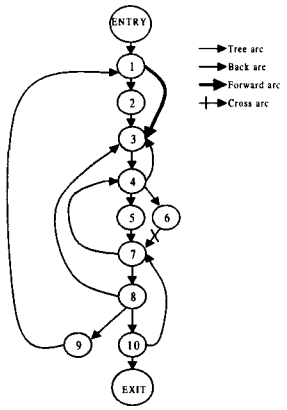
(그림 7) HTG의 예제 프로그램



(그림 8) HTG의 표현



(그림 9) CFG



(그림 10) CFG의 DFS 경로

1) 트리경로(tree arc)는 노드 y가 방문치 않는 경로 (x, y) 가리키며 {(1, 2), (3, 4), (4, 5), (4, 6), (5, 7), (7, 8), (8, 9), (8, 10)}로 표시된다.
2) 후위경로(back arc)는 트리로 구성된 y로부터 x로의 경로는 하나 존재하는 경우를 말하며 이 경로를 트리 경로라 하며, y는 DFS 트리 경로에서 x의 선행자라고 하며 {(9, 1), (8, 3), (7, 4), (4, 3), (10, 7)}로 표시 할 수 있다.
3) 전위경로(forward arc)는 노드 y는 DFS 트리에서 x의 후행자인 경우로 (x, y)는 전위경로라 하며 {(1, 3)}으로 표시된다.
4) 교차경로(cross arc)는 y는 방문했지만 x의 선행자도 후행자도 아닌 경우로 이 경우 (x, y)는 교차경로라 하며 이는 {(6, 7)}로 표시할 수 있다.

두 번째 단계는 후위경로 갖는 노드의 집합 H(G)를 구하는 과정이다. 노드의 집합 H(G)를 후위경로에 대해 입구인 모든 노드의 집합이라고 하면, 여기서 (y, x)는 후위경로로 표시할 수 있고 이 H(G)는 다음과 같이 나타낼 수 있다. 그림 10의 예제 그래프에 대해서 H={1, 3, 4, 7} 표시할 수 있다.

세 번째 단계는 H(G)에 대하여 루프를 표시하고 정의하는 과정이다. T(x)는 DFS 경로에서 x의 후행자를 나타내고 있다고 가정하면 이 경우 후위 경로에 대한 루프는 표 1로 표시 할 수 있다.

(표 1) CFG의 루프 표현

back arc	loop
L(9,1)	{1,2,3,4,5,6,7,8,9,10}
L(8,3)	{3,4,5,6,7,8,10}
L(7,4)	{4,5,6,7,8,10}
L(10,7)	{7,8,10}
L(4,3)	{3,4,5,6,7,8,10}

다음으로 H(G)내에서 모든 노드 x에 대해서 강하게 연결 돼 있는 일련의 영역 I(x)을 정의하고 다음과 같이 나타낼 수 있다.

(표 2) CFG 연결 관계 표현

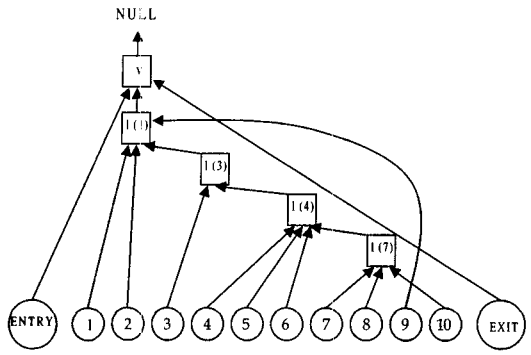
Node x	Region I(x)
I(1)	{1,2,3,4,5,6,7,8,9,10}
I(3)	{3,4,5,6,7,8,10}
I(4)	{4,5,6,7,8,10}
I(7)	{7,8,10}

네 번째 단계는 루프 멤버들의 선행관계 정의하여 계층화하는 단계이다. 표 3은 멤버들의 선행관계를 나타낸 것이다.

(표 3) (그림 4)의 멤버들의 선행관계

A	선행자 집합 P(A)
I(1)	{V}
I(3)	{I(1),V}
I(4)	{I(3),I(1),V}
I(7)	{I(1),I(3),I(4),V}
V	{ }

이 HTG는 수직 계층성을 이용하여 각 노드를 계층화 한 것을 그림 11에 보여 주고 있다.



(그림 11) 계층적 태스크 그래프의 구성

3.6 루프 변환

원시 입력 프로그램 내에서 다중 루프가 존재할 경우 루프 안쪽에 위치하고 있는 루프가 벡터화가 될 수 없고, 비효율적인 벡터코드가 생성될 경우가 발생된다. 이런 경우 원시 프로그램의 의미가 손상되지 않는 범위 안에서 루프의 수행 순서를 변경할 수 있다[8,15].

루프 교환을 구현하기 위해서는 각각의 루프 인덱스 역할을 하는 숫자를 부여하고 그리고 종속성 정보를 나타내는 방향벡터 중에서 각각의 루프의 인덱스에 해당되는 부분을 찾아내 루프교환여부가 가능한지를 판단한다. 그리고 전체의 루프에서 교환할 수 있는 루프의 인덱스 쌍을 조합하여 보관하고, 전체 집합에서 루프교환 가능조건에 만족하는 경우만 인덱스 쌍으로 지정하여 교환한다.

루프 융합은 이웃하고 있는 두 루프를 하나의 루프로 변환하는 것으로 루프 변수의 영역이 동일 해야하며 융합했을 때 자료 종속성 정보가 변하지 말아야 한다. 그러나 융합을 해서 자료 종속 정보가 유지되지 못할 경우는 루프 융합을 할 수 없다. 이런 루프 융합의 타당성 검증은 먼저 이웃하는 두 개의 루프를 하나의 루프로 변환한 뒤 문장간의 종속성 여부를 검증하여 역 종속이 존재하면 루프융합이 불가능하다. 이 변환은 루프 반복의 낭비를 줄이고 프로그램의 효율을 높이기 위해 사용한다.

루프 분산은 하나의 루프 안에 있는 여러 개의 문장들을 자료 종속관계에 따라 좀더 작은 단위로 분할하는 방법이다. 이 나뉘어진 새로운 루프들은 원래의 프로그램의 문장들을 나누어 갖게되고 원래의 루프와 같은 회전 범위를 가지고 수행한다. 이 루프 분산의 목적은 하나의 루프 안에서 반복적으로 접근되는 자료의 양과 수행되는 명령들의 수를 줄여 자료 지역성을 증가시켜 캐시의 효율화를 위해 이용한다[15].

4. 병렬 프로그래밍 환경

본 장에서는 그래프 중간 표현 형태를 기반으로 한 병렬 프로그래밍 환경에 대하여 소개하고 그 기능 및 작동방법에 대하여 기술한다. 이 논문에서 제안한 그래픽 중간 표현 형태를 기반으로 한 병렬 프로그래밍 환경은 다음과 같은 특징을 가지고 있다.

첫째 사용자가 프로그램 작성이 용이하도록 그래픽 인터페이스 방식과 메뉴 선택 방식 을 이용하여 순차 프로그램을 화면에서 직접 입력, 수정, 편집이 가능하도록 설계하였다.

두 번째 이 입력된 프로그램을 수행하여 문장간의 자료 종속성을 분석하여 종속성 정보를 제공하며 종속성 정보를 여러 가지 그래프 정보로 보여준다.

세 번째는 병렬화 단계로 종속성 분석 후에 이를 기초로 루프의 변환 가능성 여부를 사용자에게 제공하고, 루프를 변환 관련 정보를 제공한다. 그림 12는 그래프 중간 표현 형태를 기반으로 한 병렬 프로그래밍 환경을 보여주고 있다.

4.1 사용자 환경

병렬 프로그래밍 환경은 사용자와 병렬 프로그램 환경사이에 상호 작용이 용이하기 위해 그래픽 인터페이스를 사용하며, 메뉴 선택방식을 이용

(그림 12) 그래프 중간 표현 형태를 기반으로 한 병렬 프로그래밍 환경

하여 병렬 프로그래밍 환경을 제공하였다. 본 논문의 병렬 프로그래밍 환경은 네 가지 기능적인 환경을 제공한다.

첫째는 사용자에게 프로그램의 작성을 도와주는 파일 편집 기능이며, 둘째는 종속성 정보를 분석하여 사용자에게 종속성 정보를 제공하고 종속관계를 그래픽으로 제공하는 기능이며, 셋째 중간코드의 병렬환경을 제공하는 기능으로 루프의 변환 정보와 순차 프로그램을 입력하여 중간 그래프를 사용자에게 제공한다. 마지막으로 출력 및 도움말 기능을 제공한다. 표 4는 사용자 메뉴와 기능을 나타낸다.

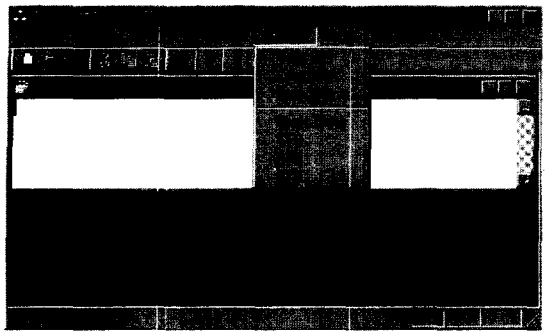
(표 4) 사용자 메뉴

작업구분	메뉴	작업 기능
파일	newfile	새프로그램 입력
	open	기존 프로그램 열기
	load	기존 프로그램 수정
	save	프로그램 저장
	save as	다른이름으로 저장
	quit	프로그램 종료
편집	cut	편집내용 일부삭제
	add	편집내용 덧붙임
	delete	프로그램 삭제
기본블록	기본블록	기본 블록 나누기
종속성	종속성 분석	종속성 정보 출력
		종속성 그래프 표현
중간 그래프	CFG	제어 흐름 그래프
	PDG	프로그램 종속성그래프
	HTG	계층적 태스크 그래프
루프 변환	loop interchange	루프 교환
	loop distribution	루프 분산
	loop fusion	루프 융합
출력	화면내용 출력	
도움말	작업내용 도움말	

4.2 병렬 프로그래밍 환경 구현

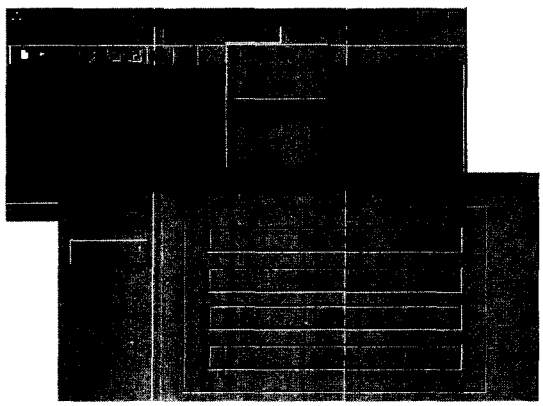
병렬 프로그래밍 환경에서 사용언어는 비주얼 C++ 언어로 구현하였으며 win 98 운영체제를 사용하였다. 단말에서 "icgen"을 수행하면 병렬 프로그래밍 메뉴가 떠오른다. 사용자는 병렬 프로그램을 작성하기 위하여 "newfile"을 선택하면 파일을 편집할 수 있는 창이 나타난다. 그림 13은 병렬환경에서의 초기 메뉴 창을 보여준다.

그림 14와 그림 15는 순차 프로그램을 기본 블록으로 표현한 것을 보여준다. 일반적으로 기본 블록으로 표현하기 위해서는 3-주소형태로 변화하고 루프의 포함여부를 분석하여 리더를 첨가하여 구성한다.

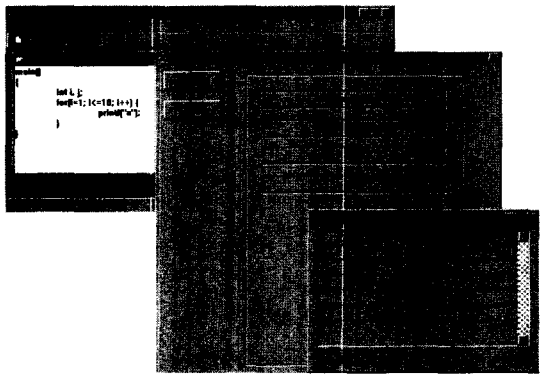


(그림 13) 초기 메뉴창의 모습

이 그래프는 제어 흐름 정보를 분석하는데 사용되며 모든 중간코드의 그래프 구성에 기본적으로 사용된다. 그림 14는 기본 블록을 수행했을 때의 모습을 보여준다.

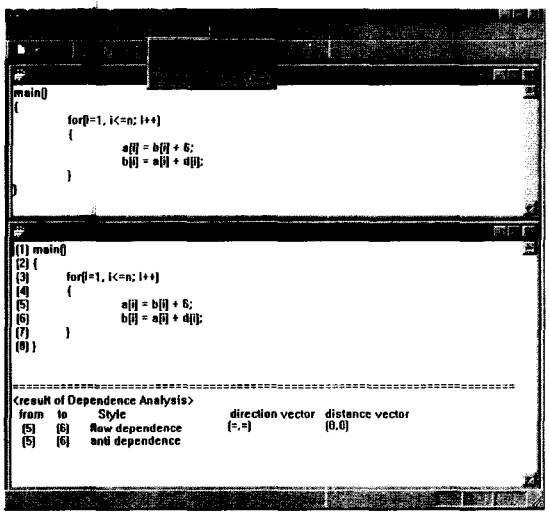


(그림 14) 기본 블록 메뉴 창

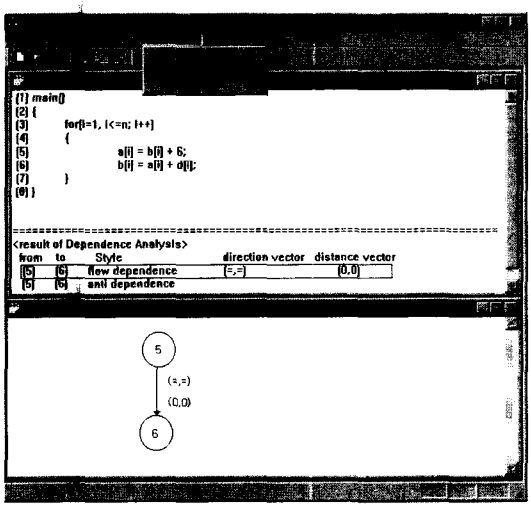


(그림 15) 기본 블록 표현

종속성분석은 특정 프로그램을 선택하여 종속성 관계를 분석하고자 할 때 사용한다. 사용자는 종속성 분석을 하기 위해서는 먼저 분석 하고자 하는 프로그램을 오픈하고 해당 프로그램을 로드 시켜야 한다. 그림 16는 특정 프로그램을 종속성분석을 하였을 경우의 창의 모습을 보여주고 있으며 그림 17은 종속성 그래프를 보여주고 있다.

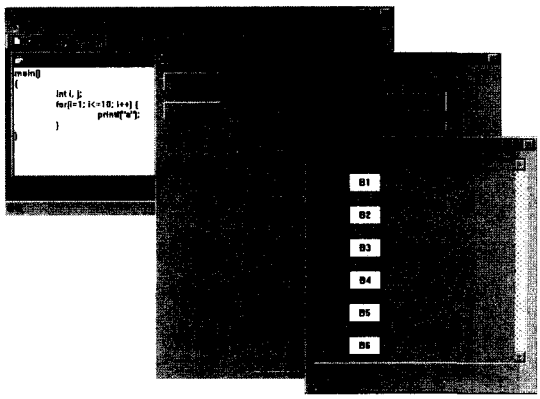


(그림 16) 종속성 분석 결과



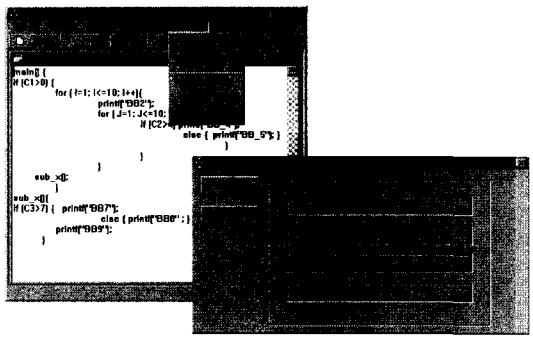
(그림 17) 종속성 표현

CFG는 기본블록에서 제어 정보를 추가하여 그래프로 표현한 것이다. 그림 18은 순차 프로그램에서 CFG로 작성된 것을 보여주고 있다. 이 정보는 프로그램의 제어 흐름정보를 알 수 있고 이 CFG는 다른 중간 그래프를 작성하는데 이용된다.

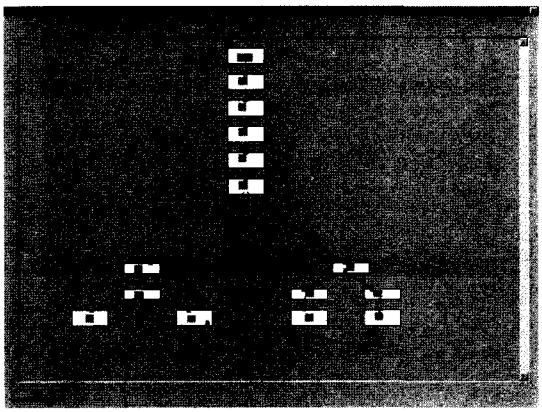


(그림 18) CFG 의 표현

PDG는 제어 종속성 정보와 자료 종속성 정보를 한 그래프에 표시한 중간 그래프이다. 이 그래프를 표현하기 위해서는 문장을 기본 블록으로 나누어 CFG를 작성한 후 노드간의 자료종속성, 자료 종속성을 분석하여 그 정보를 각각 한 그래프를 표현한다. 그림 19는 PDG의 프로그램과 메뉴를 나타낸 것이고 그림 20은 PDG를 표현한 것이다.

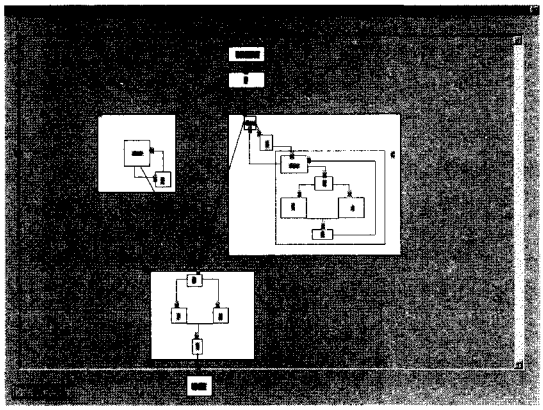


(그림 19) PDG 메뉴



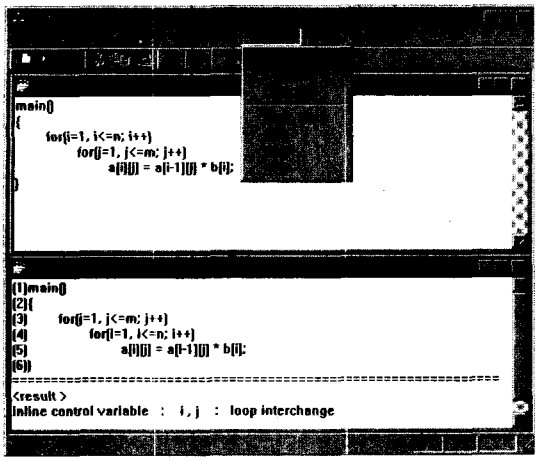
(그림 20) PDG 표현

그림 21은 HTG를 표현한 중간 그래프이다. 이 HTG는 구조적 프로그램 형태로 구성되어야 하는데 이 HTG를 표현하기 위해서는 먼저 CFG를 작성한 후 이를 DFS를 수행하여 후위 경로 추출하여 HTG를 그린다. 그림 21은 HTG를 나타낸 것이다.



(그림 21) HTG의 표현

루프교환은 사용자가 먼저 해당 프로그램을 load하여 종속성 분석 메뉴를 이용하여 종속성 여부를 파악한 후에 루프 교환 가능성을 판단한다. 루프 교환은 원시 프로그램의 의미가 손상되지 않도록 루프의 수행 순서를 바꾸는 것이다. 그림 22는 변경 전, 후의 창의 모습을 보여주고 있다.



(그림 22) 루프 교환 전, 후의 창의 모습

5. 결 론

본 논문에서는 일반적인 병렬 프로그래밍 환경에 대해 소개하고, 병렬 프로그래밍 환경을 제공하기 위한 그래프 중간 표현 형태를 기반으로 한 병렬 프로그래밍 환경을 설계하고 구현하였다.

논문에서 제공하는 병렬 프로그래밍 환경은 메뉴 선택방식을 적용하여 사용자가 쉽게 사용할 수 있도록 설계 및 구현하였으며, 그래프 중간 표현 형태를 자동 병렬 변환 환경을 제공하여 원시 코드를 입력하면 즉시 중간 그래프를 제공한다. 또 특정 하드웨어와는 상관없이 작업할 수 있는 개방 사용자 환경, 종속성 분석 기능, 루프 병렬화 기능 등을 제공한다. 사용자는 병렬 프로그래밍 환경에서 작업하므로 작업시간의 단축은 물론 메모리 및 캐시 메모리를 효율적으로 이용할 수 있으며 병렬화, 최적화를 위한 각종 정보를 제공받을 수 있다

향후 연구 방향으로는 좀더 확장된 병렬환경을 제공하기 위해 본 연구에서 제공되지 않은 중간 그래프를 더 추가하여 개발하고, 하드웨어적인 특성을 고려하여 이용할 수 있도록 병렬 디버거, 성능 분석기 등의 개발하면 최적의 병렬 프로그래밍 환경이 될 것이다.

참 고 문 헌

[1] Allen J. R. and K. Kennedy, "A Parallel Programming Environment," IEEE Software, Vol. 2, No 4, pp. 21-29, July, 1985.

[2] Allen R. and K. Kennedy, "Automatic Translation of FORTRAN to Vector Form," ACM Transactions on Programming Languages and Systems, Vol. 9, No. 4, pp. 491-542, October 1987.

[3] Allen J. and K. kennedy, "PFC : A Program to convert Fortran to Parallel Form," Technical Report MASC-TR82-6, Rice University, March 1982.

[4] Girkar M. and D. Polychronopoulos "Automatic Extraction of Functional Parallelism from Ordinary Programs," IEEE Transactions on Parallel and Distributed Systems Vol. No. 2, pp. 166-178, March 1992.

[5] The Stanford SUIF Compiler Group, SUIF Compiler System, Available via URL, http://suif.stanford.edu/suif.

[6] Smith K. and W. F. Appelbe, "PAT: Interactive Fortran Parallelizing Assistant Tool," Processing of 1988 International Conference on Parallel Processing, pp. 58-62, August 1988.

[7] 박성순, 그래프 중간 표현 형태를 이용한 프로그램 벡타화, 고려대학교, 박사학위논문 논문, 1993.

[8] Beck M. and K. Pingali, "From Control Flow to Dataflow," Proceeding of the '90 International Conference on Parallel Processing, Vol. 2, pp. 43-52, 1990.

[9] Jeanne Ferrante, Karl J. Ottenstein, Joe D. Warren, "The Program Dependence Graph and It's Use in Optimization," Lecture Notes in computer Science Vol. 167, Springer-Verlag, pp. 125-132, 1984.

[10] Kuck, D.J. The Structure of Computers and Computations, Vol. 1. John Wiley and Son, New

York, 1978.

[11] 송월봉, 박두순, "최대 병렬성 추출을 위한 자료 종속성 제거 알고리즘", 정보과학회 논문지 제26권 제1호, pp. 139-149, 1999.

[12] Alfred V. Aho, Ravi sethi, Jeffrey D. Ullman, Compilers : principles, Technique, and Tools Assision-Wesley, Reading, Massachusets, 1986.

[13] Ferrante J., K. J. Ottenstein and J. Warren, "The Program Dependence Graph and Its Use in Optimization," ACM Transaction on Programming Language and System, Vol. 9, No. 3, pp. 319-349, July 1987.

[14] Girkar, Milind Baburao, "Functional Parallelism : Theoretical Foundations and Implemention," Ph. D. These, University of Illions at Urbana-Champaign, 1992.

[15] 이만호, "병렬화를 위한 루프의 구조의 변환", 정보과학회지 제12권 제5호, pp. 54-56, June, 1994.

◐ 저 자 소 개 ◑

이 원 용

1980년 중앙대학교 전자계산학과 졸업(공학사)
1985년 중앙대학교 국제경영대학원 경영정보학과 졸업(경영학석사)
1998년 순천향대학교 대학원 전산학과 박사과정 수료
1993～현재 : 혜전대학 컴퓨터계열 조교수
관심분야 : 병렬처리, 컴파일러, 프로그램언어
E-mail : wylee@hyejeon.ac.kr

박 두 순

1981년 고려대학교 수학과 졸업(이학사)
1983년 충남대학교 대학원 전산학과 졸업(이학석사)
1888년 고려대학교 대학원 전산학 전공(이학박사)
1992년～1993년 미국 U. of Illinois at Urbana-Champaign CSRD 객원교수
2000년～현재 : 순천향대학교 컴퓨터교육원 원장
1985년～현재 : 순천향대학교 정보기술공학부 교수
관심분야 : 병렬처리, 컴파일러, 멀티미디어 정보검색, 가용성, 컴퓨터교육
E-mail : parkds@sch.ac.kr