그래픽 사용자 인터페이스로 구현한 병렬 프로그래밍 환경

유 정 목† · 이 동 희†† · 이 만 호†††

요 약

본 논문에서는 사용자의 병렬 프로그램 작성을 도와주는 병렬 프로그래밍 환경에 대해 기술한다. 병렬 프로그래밍 환경은 일반적인 컴파일러의 기능의 전위 부분에 해당하는 어휘분석과 구문분석 기능 수행하고 있으며, 프로그램에서 이용하는 변수들의 데이터 흐름 분석과 데이터 종속성 분석, 그리고, 여러 가지 병렬 프로그램 변환 기법들을 수행한다. 특히 프로그래머가 병렬 프로그램을 용이하게 작성할 수 있도록 그래픽 사용자 인터페이스를 제공한다.

A Parallel Programming Environment Implemented with Graphic User Interface

Jeong-Mok Yoo† · Dong-Hee Lee†† · Mann-Ho Lee†††

ABSTRACT

This paper describes a parallel programming environment to help programmers to write parallel programs. The parallel programming environment does lexical analysis and syntax analysis like front-end part of common compilers, data flow analysis and data dependence analysis for variables used in programs, and various program transformation methods for parallel programming. Especially, graphic user interface is provided for programmer to get parallel programs easily.

1. 서 론

현대 사회의 정보화, 산업화 경향에 따라 컴퓨터가 처리해야 하는 정보의 양이 많아지게 되었다. 이 방대한 양의 정보 처리를 위해 병렬 컴퓨터가 등장하게 되었다. 병렬 컴퓨터는 방대한 양의 정보를 처리하는 과정에 많은 어려움을 느끼고 있던 과학자나 공학자들에게 새로운 대안이 되었다. 그러나, 기존에 순차 컴퓨터에 익숙해 있던 과학자나 공학자들은 병렬 컴퓨터들의 다양한 하드웨어적인 특성들로 인해 병렬 컴퓨터를 위한 효율적인 병렬 프로그램을 작성할 수 없었다. 이러한 병렬 프로그램 작성의 어려움을 해결할 수 있는 방법으로서, 병렬 프로그램을 작성하는 과정에서 미리 사용할 병렬 컴퓨터의 특성을 파악하여 병렬 지시어나 병렬 구조를 이용하는 방법과, 기존에 이미 사용되고 있는 순차 프로그램을 자동적으로 병렬 프로그램으로 변환해 주는 방법이 있다. 기존 순차 프로그램에 익숙해 있는 프로그래머들은 병렬 프로그램 작성의 어려움으로 인해 전자보다는 후자의 방법을 사용하여 병렬 프로그램을 작성하는 것이 효과적이다[5].

기존에 작성된 순차 프로그램으로부터 병렬 프로그램을 작성할 수 있도록 사용되는 시스템을 병렬 프로그래밍 환경이라고 한다[4]. 일반적으로 병렬 컴퓨터들은 다양한 하드웨어적인 특성을 가지고 있기 때문에, 병렬 프로그램 작성을 도와주는 병렬 프로그래밍 환경

※ 본 논문은 한국과학재단 특정연구 과제 연구비 지원에 의한 결과임(과제번호 : 94-0100-13-03-3).
† 정 회 원 : 충남대학교 대학원 컴퓨터과학과
†† 정 회 원 : 아이시스텍(주) 소프트웨어개발팀 팀장
††† 종신회원 : 충남대학교 정보통신공학부 교수
논문접수 : 1999년 6월 7일, 심사완료 : 2000년 7월 31일

을 이용하는 사용자들은 이러한 병렬 컴퓨터의 특성을 감안할 수 있어야 한다. 이를 위해서는 병렬 프로그래밍 환경은 사용자와의 상호작용을 통해 사용자가 여러 가지의 다양한 하드웨어적인 특성을 가지는 병렬 컴퓨터들을 위한 병렬 프로그램을 작성할 수 있도록 해 주어야 한다[8, 13, 14].

병렬 프로그래밍 환경은 새로운 병렬 프로그램의 작성뿐만 아니라, 기존에 작성된 병렬 프로그램의 최적화 작업에도 이용될 수 있다. 병렬 컴퓨터들은 다양한 하드웨어적인 특성을 가지기 때문에, 한 병렬 컴퓨터에서 이용되었던 응용 프로그램은 다른 하드웨어적인 특성을 가진 병렬 컴퓨터에서 직접 이용될 수 없다. 이 응용 프로그램을 이용하기 위해서는 응용 프로그램에 사용할 병렬 컴퓨터의 하드웨어적인 특성을 반영해야 한다. 이를 위해서 병렬 프로그래밍 환경은 이미 작성된 병렬 프로그램을 다른 형태의 병렬 프로그램으로 변환할 수 있어야 한다.

일반적으로 병렬 프로그래밍 환경은 병렬화 대상이 되는 프로그램을 입력받아 어휘 분석과 구문 분석을 수행하는 전위 부분과, 데이터 흐름 분석과 데이터 종속성 분석을 수행하고 이들로부터 얻는 분석 정보를 이용하여 병렬화를 수행하는 후위 부분으로 구성된다. 병렬화 변환은 종속성 분석 결과를 활용하여 병렬 컴퓨터의 하드웨어적인 특성을 효율적으로 이용할 수 있도록 해야 하므로, 성능을 예측할 수 있는 성능 분석기를 포함하는 것이 바람직하다. 또한 병렬화 변환된 프로그램은 원래의 프로그램의 의미가 다르게 변환되어서는 안 된다.

본 논문에서 고려한 병렬 프로그래밍 환경은 데이터 병렬 언어인 HPF의 부분집합을 병렬 프로그램 작성의 대상 언어로 하고 있다. HPF로 작성된 프로그램을 입력으로 받아 병렬화 변환을 한 후, 다시 HPF 언어로 된 병렬 프로그램을 생성한다. 프로그램의 병렬화 변환을 위해서, 환경은 프로그래머에게 데이터 종속성 분석 정보를 제공하고, 프로그래머는 병렬 컴퓨터의 하드웨어적인 특성을 고려하여 적절한 병렬화 변환 방법을 선택할 수 있도록 하고 있다. 이를 위해서 X-Window를 기반으로 하는 그래픽 사용자 인터페이스 환경을 제공하고 있다. 그리고, 본 논문은 특정한 병렬 컴퓨터를 대상으로 하지 않고 있으므로, 특정 하드웨어에 종속적인 병렬화 변환과는 무관하게 일반적으로 적용할 수 있는 방법만 기술하고 있으며, 병렬화에 관한 새로운 이론보다는 이미 잘 알려진 이론들을 이용해 병렬 프로그램을 작성할 수 있는 환경을 구축한 것에 대해 기술하고 있다.

2. 병렬 프로그래밍 환경의 사례

병렬 프로그래밍 환경에 관한 연구는 병렬 컴퓨터의 출현과 함께 여러 곳에서 수행되고 있다. 예를 들면, 라이스 대학의 Parascope[9, 10]와 PFC[5], 일리노이 대학의 Parafrase[12]와 Faust[7], 본 대학의 SUPERB [14], 조지아 공대의 Start/Pat[13], IBM의 Ptran[2] 등이 있다. 본 장에서는 이 중에서 몇 가지의 예제를 살펴본다.

2.1 Parafrase

Parafrase는 일리노이 대학에서 개발된 병렬화 컴파일러로서 많은 병렬화 컴파일러에 영향을 주었다. Parafrase의 입력은 Fortran 프로그램이며 출력은 확장된 병렬 Fortran 프로그램과 변환에 대한 오류, 성능 예상 등이다. Parafrase에서는 100여 개 이상의 코드 변환 및 병렬화 알고리즘을 제공하고 있는데, 각각의 알고리즘에 대해서는 선행되어야 할 분석이나 변환 알고리즘이 정의되어 있다. 사용자는 변환 과정들의 순서와 종류를 선택할 수 있으며, 이를 통해 다양한 응용에 적절한 병렬화 작업을 수행할 수 있도록 하고 있다.

2.2 PFC

PFC는 Rice 대학에서 개발한 벡터화 컴파일러이다. 중간 코드는 추상구문트리가 이용되며 Fortran 66 또는 Fortran 77로 작성된 프로그램을 입력으로 받아 프로시저간 분석, 전처리 코드 변환, 의존성 분석, 벡터 코드 생성의 네 가지 과정을 거쳐 Fortran 90 프로그램을 출력한다. PFC는 순차 프로그램 내에서 루프의 병렬성을 향상시키는 것에 중점을 두고 있다. 이는 병렬 컴퓨터를 위한 응용 프로그램들에서 발견되는 대부분의 계산 과정들이 루프 내에서 반복문의 형태로 이루어지기 때문이다. PFC는 다른 병렬 프로그래밍 환경에 비해 상대적으로 적은 수의 분석 및 변환 알고리즘을 적용하고 있으며, 각각의 패스가 내부 자료구조에 한꺼번에 적용되기 때문에 실행 속도가 빠르다고 한다.

2.3 Ptran

Ptran은 IBM에서 개발한 공유 메모리 컴퓨터를 위

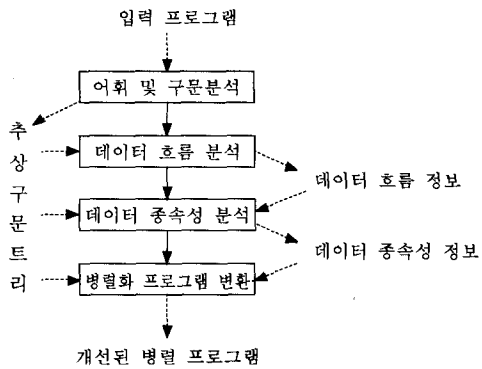
한 병렬 프로그래밍 환경이다. 입력 언어는 병렬 반복문을 첨가한 VS Fortran 77이며 출력은 VS Fortran 2.5나 Tobey와 같은 병렬 언어로 작성된 병렬 프로그램이다. Ptran 분석부에는 프로시저간의 분석과 프로시저내의 분석을 상호 보완적으로 이용한다. 분석부에서는 먼저 호출 그래프를 하향적으로 추적하면서 프로시저간 이명 분석(alias analysis)과 상수 전달을 수행한다. 그 후에 다시 호출 그래프를 상향으로 추적하면서 프로시저내의 상수 전달 및 의존성 분석, 프로시저 호출시의 부작용, 자료 흐름 분석 등을 수행하게 된다. 또한 병렬 프로그램 생성 시에는 수행될 병렬 컴퓨터에 따른 병렬 수행의 부담을 고려하여 수행 시간을 추정함으로써 속도의 향상을 최적화할 수 있도록 하였다.

2.4 Start/Pat

Start/Pat은 조지아 공대와 산타크루즈에 있는 캘리포니아 대학에서 개발한 병렬 프로그래밍 환경으로 병렬화 변환기, 정적 분석기, 동적 분석기, 동적 디버거, 성능 분석기 등으로 구성되어 있다. Start/Pat은 효율적이고 이식 가능한 멀티태스킹 프로그램의 작성, 디버깅 작업과 테스트가 쉬운 병렬 프로그램의 작성을 목표로 설계하고 구현되었다. Start/Pat은 라이스 대학에서 개발한 PTOOL과 유사하지만, 다음과 같은 차이점들이 있다. Start/Pat은 PTOOL과 달리 데이터 종속성 정보를 사용자에게 보여주는 것뿐만 아니라, 원시 입력 프로그램에 대한 변환 작업을 사용자에게 제안하고 수행한다. 그리고, 여러 가지 형태의 병렬 Fortran 언어를 인식할 수 있는 전위 부분을 가진 Fortran 77 컴파일러를 기반으로 구현되어 시스템 자체가 이식이 가능하다.

3. 병렬화 프로그램 변환 단계

일반적으로 병렬 프로그래밍 환경은 (그림 1)에서 보는 바와 같이 어휘 분석과 구문 분석, 데이터 흐름 분석, 데이터 종속성 분석, 병렬화 프로그램 변환 과정들과 같은 단계를 거쳐 병렬 프로그램을 얻을 수 있도록 해 준다. 입력 프로그램은 순차 프로그램일 수도 있으나 이미 병렬 수행이 가능한 병렬 프로그램일 수도 있다. 순차 프로그램이 입력되면 병렬화 변환이 주요 관심이겠으나, 병렬 프로그램이 입력되는 경우에는 보다 효율적인 병렬 프로그램을 얻는 것이 목적이 된다.



(그림 1) 병렬 프로그래밍 환경

어휘 분석과 구문 분석 단계에서는 원시 입력 프로그램인 사용자의 병렬 프로그램에 대한 구문적인 오류를 검사하고 프로그램에 대한 추상구문트리와 심볼 테이블을 작성한다. 추상구문트리는 데이터 흐름 분석 과정과 데이터 종속성 분석 과정에서 입력 프로그램에 대한 중간 코드로서 사용된다.

데이터 흐름 분석과 데이터 종속성 분석 과정에서 나오는 결과인 데이터 흐름 정보와 데이터 종속성 정보를 위한 자료구조를 별도로 두지 않고, 어휘분석과 구문분석 과정에서 구성된 추상구문트리의 문장 노드를 이용하여 데이터 흐름 정보와 데이터 종속성 정보를 저장하였다. 이것은 문장 단위로 이루어지는 프로그램의 병렬화 작업을 용이하게 할뿐만 아니라, 정보의 참조 및 갱신에 있어서 편리한 점이 많다.

병렬화 프로그램 변환 단계에서는 데이터 종속성 분석 단계에서 작성된 데이터 종속성 정보를 기반으로 원시 입력 프로그램의 병렬화에 필요한 여러 가지 병렬화 작업을 수행한다. 프로그램의 병렬화에 있어 그 대상은 루프 구조이다. 루프 구조는 크기가 원시 입력 프로그램에서 차지하는 비중이 작더라도, 그 수행이 반복되기 때문에 전체 프로그램의 성능에 많은 영향을 미친다[11]. 루프 구조를 병렬화하는 방법은 크게 벡터화와 좁은 의미의 병렬화로 분류할 수 있는데, 병렬화할 때는 병렬 컴퓨터의 특성에 따라 효율적인 것을 택하여 병렬화한다.

일반적으로 데이터 흐름 분석은 프로그램에서 사용된 변수들의 정의-사용 관계와 사용-정의 관계를 기반으로 한다[1]. 본 논문에서는 기존에 사용되던 변수들의 정의-사용 관계와 사용-정의 관계를 확장하여 외부 노출 정보를 표현하는 기법을 사용하여, 데이터 흐름

분석 정보가 다음 단계의 데이터 종속성 분석에 효과적으로 사용될 수 있도록 하였다.

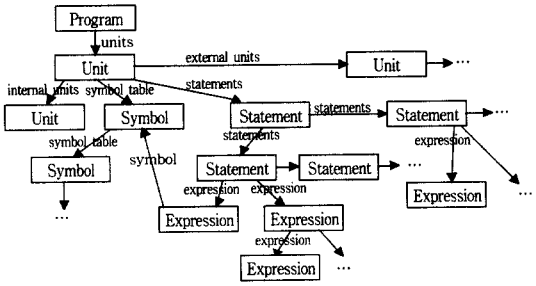
데이터 종속성 분석 단계에서는 데이터 흐름 분석 단계에서 작성된 데이터 흐름 정보를 기반으로 원시 입력 프로그램의 병렬화 작업에 필요한 데이터 종속성 정보를 작성한다.

3.1 어휘 분석과 구문 분석

이 과정은 일반적인 컴파일러에서도 처음에 실행되는 단계로서, 원시 입력 프로그램을 이후의 단계에서 작업하는데 편리한 구조인 추상구문트리를 생성하는 단계이다. 이 단계에서는 기본적으로 프로그램의 문법적인 요소들을 구별해 내는 어휘 분석을 거쳐서, 이들이 문법적으로 오류가 없는 것을 확인하고 프로그램의 문법적 구조를 트리 형태로 변환하는 구문 분석으로 이루어져 있다.

본 논문에서 사용하고 있는 추상구문트리는 (그림 2)와 같이 나타낼 수 있다. 사용하고 있는 노드로는 UnitNode, StmtNode, ExprNode, SymbNode 4가지이며, 이들을 이용하여 프로그램을 계층적으로 표현한다.

UnitNode는 메인 프로그램이나 서브 프로그램과 같은 모듈을 표현한다. 한 프로그램이 여러 개의 모듈로 구성되어 있으면, 포인터를 이용해서 리스트와 같이 연결된다.



(그림 2) 추상구문트리의 계층적 구조

StmtNode는 모듈을 구성하고 있는 문장들을 나타낸다. 모듈은 변수 선언문과 같은 비실행문과 치환문과 같은 실행문이 있는데, 이들 모두가 StmtNode로 표현된다. 한 모듈 안에 있는 문장들은 포인터를 이용해서 리스트와 같이 연결되어 있는데, 모듈에서 맨 처음에 나타나는 문장의 StmtNode는 UnitNode가 포인트하고 있다. 또한 선언문을 통해서 선언된 변수들은 SymbNode를 사용해서 변수들의 특성을 표현하고, 이들 또한 포인터를 사용해서 리스트와 같이 연결하고, UnitNode가 맨 처음에 위치한 SymbNode는 포인트하도록 하여, 이들 SymbNode들은 모듈의 심볼테이블로 사용된다. WHILE문, DO문, IF문같이 문장의 하부 구조가 문장이 될 수도 있는 경우에 StmtNode는 하부 구조로서의 StmtNode를 포인트하기도 한다.

ExprNode는 문장을 구성하는 요소들인 수식이나 연산자, 또는 이들의 나열을 표현하는데 사용된다. 수식에는 연산자가 있고 하나 또는 두 개의 피연산자가 있다. 이들을 2진트리 형태로 표현하고, 이들의 루트 노드를 StmtNode가 포인트하고 있다.

3.2 데이터 흐름 분석

데이터 흐름 분석 단계에서는 프로그램 수행 시에 데이터가 사용되고 정의되는 과정을 분석하여 데이터 흐름 정보를 얻는다. 오늘날 대부분의 프로그램은 순차, 분기, 반복을 특징으로 하는 구조적 프로그램 형태로 대부분 작성되므로, 본 논문은 구조적 프로그램의 기본 단위인 단순 치환문, 순차 구조, 분기 구조, 반복 구조를 대상으로 데이터 흐름 정보를 분석한다[1].

데이터 흐름 정보는 데이터 종속성 정보를 분석하는데 이용되며, 데이터 종속성 정보에는 흐름종속, 역종속, 출력종속 등 세 가지가 있다. 그러나, 일반적으로 이용되는 정의-사용 관계에 의한 데이터 흐름 정보는 흐름종속을, 사용-정의 관계에 의한 데이터 흐름정보는 역종속을 표현하기에 용이하여 출력종속 정보를 분석하기 위해서는 별도의 정보가 필요하다. 본 논문에서는 세 가지 형태의 종속성 정보를 종합적으로 분석하기 위해 블록 단위로 데이터 흐름 정보를 분석한다.

프로그램이 수행될 때 일정한 위치에서 흐름이 시작되고 한 위치에서 끝나는 일련의 문장들을 묶은 것을 블록이라고 한다. 블록은 단 하나의 문장이 될 수도 있고, 순환적으로 내부에서 블록의 흐름으로 구성될 수 있고, 여러 갈래로 분기가 이루어지기도 하고, 일련의 블록들이 반복되어 수행되기도 한다. 각 블록은 같은 레벨의 블록과 연결되고, 내부 블록은 다른 블록에 의해 직접적인 영향을 받지 않는다. 프로그램은 문장들이 각각 블록으로 나누어진 블록들의 순차적인 수행 형태로 나타난다.

본 논문에서는 데이터 흐름 정보를 분석하기 위해, [1]에서 사용하고 있는 데이터 흐름 분석 방법을 확장

하여, 블록 외부에 영향을 미칠 수 있는 외부노출(outward exposed) 정보를 이용하고 있다. 외부노출 정보는 데이터가 흐르는 방향을 고려하여, 전방노출(outpreward exposed) 정보와 후방노출(out-postward exposed) 정보로 구분한다. 한 블록의 전방노출 정보는 이전 블록의 후방노출 정보와 종속관계가 있을 수 있는 정보이며, 후방노출 정보는 다음 블록의 전방노출 정보와 종속관계가 있을 수 있는 정보이다. 이들 외부노출 정보는, 정의-사용 관계 및 사용-정의 관계 분석을 위해, 다시 외부로 노출되는 정의와 사용 정보로 구분한다.

프로그램은 블록의 흐름으로 볼 수 있으므로, 블록간의 연관성을 고려하여 정보를 유지하고 갱신하기 위해, 제어의 흐름에 따라 블록으로 입력되는(In) 정보와 다음 블록으로 출력되는(Out) 정보를 생성하여, 데이터 흐름 정보를 분석한다. 본 환경에서 사용되는 모든 데이터 흐름 정보는 <표 1>과 같으며 본 논문의 나머지 부분에서는 약어를 사용하기로 한다.

〈표 1〉 확장된 데이터 흐름 정보

약 어	의 미	범 위
InPreDef	입력 전방노출 정의	이전 블록까지의 외부노출 정보
InPreUse	입력 전방노출 사용	
InPostDef	입력 후방노출 정의	
InPostUse	입력 후방노출 사용	
PreDef	전방노출 정의	현재 블록의 외부노출 정보
PreUse	전방노출 사용	
PostDef	후방노출 정의	
PostUse	후방노출 사용	
OutPreDef	출력 전방노출 정의	현재 블록까지의 외부노출 정보
OutPreUse	출력 전방노출 사용	
OutPostDef	출력 후방노출 정의	
OutPostDef	출력 후방노출 사용	

현재 분석하고 있는 블록의 전방노출 정보에서, PreDef는 이 블록에서 정의되는 변수들로서 각 변수 정의 중에서 처음으로 정의되는 것만을 말하며 이전 블록까지에서 정의된 변수를 무효화할 수 있고, PreUse는 이 블록에서 정의되기 전에 사용되는 변수로서 이전 블록까지에서 정의된 변수를 사용하는 것이다. 또한 후방노출 정보에서, PostDef는 이 블록에서 정의되는 변수들로서 각 변수 정의 중에서 마지막으로 정의되는 것만을 말하며 다음 블록에 영향을 미칠 수 있고, PostUse는 이 블록에서 정의한 후에 사용되는 변수와 정의하지 않고 사용되는 변수를 말한다.

현재 분석하고 있는 블록의 입력 외부노출 정보는 이전 블록까지를 하나의 블록으로 보고 분석된 외부노출 정보인데, 입력 외부노출 정보와 현재 분석하고 있는 블록의 외부노출 정보를 결합하여, 출력 외부노출 정보를 생성하게 되는데, 출력 외부노출 정보는 현재 블록까지의 모든 블록을 하나의 블록으로 보고 분석한 외부노출 정보가 된다. 이 출력 외부노출 정보가 다음 블록으로 넘겨져서, 다음 블록의 외부노출 정보를 분석하는 방식으로 프로그램의 모든 블록을 분석하게 된다.

예를 들어, <표 2>에서 3개의 단순 치환문의 경우에 각 문장을 블록으로 보고 외부노출 정보를 분석해 보았다. 표에서 각 문장은 번호로 표현하고, 변수는 동일한 변수가 여러 곳에 나오기 때문에 이들을 구별하기 위해서 첨자를 사용했다. 그러므로 첨자가 다르더라도 변수 이름이 같으면 동일한 변수이다.

〈표 2〉 외부노출 정보 분석의 예

문장	문 장	외부노출 정보
1	$A_1 = A_2 + B_1$	InPreDef=ϕ, InPreUse=ϕ InPostDef=ϕ, InPostUse=ϕ
		PreDef={[1,A_1]}, PreUse={[1,A_2],[1,B_1]}, PostDef={[1,A_1]}, PostUse={[1,B_1]}
		OutPreDef={[1,A_1]}, OutPreUse={[1,A_2],[1,B_1]}, OutPostDef={[1,A_1]}, OutPostUse={[1,B_1]}
2	$B_2 = A_3 + C_1$	InPreDef={[1,A_1]}, InPreUse={[1,A_2],[1,B_1]} InPostDef={[1,A_1]}, InPostUse={[1,B_1]}
		PreDef={[2,B_2]}, PreUse={[2,A_3],[2,C_1]}, PostDef={[2,B_2]}, PostUse={[2,A_3],[2,C_1]}
		OutPreDef={[1,A_1],[2,B_2]}, OutPreUse={[1,A_2],[1,B_1],[2,C_1]}, OutPostDef={[1,A_1],[2,B_2]}, OutPostUse={[2,A_3],[2,C_1]}
3	$B_3 = B_4 + C_2$	InPreDef={[1,A_1],[2,B_2]}, InPreUse={[1,A_2],[1,B_1],[2,C_1]} InPostDef={[1,A_1],[2,B_2]}, InPostUse={[2,A_3],[2,C_1]}
		PreDef={[3,B_3]}, PreUse={[3,B_4],[3,C_2]}, PostDef={[3,B_3]}, PostUse={[3,C_2]}
		OutPreDef={[1,A_1],[2,B_2]}, OutPreUse={[1,A_2],[1,B_1],[2,C_1],[3,C_2]}, OutPostDef={[1,A_1],[3,B_3]}, OutPostUse={[2,A_3],[2,C_1],[3,C_2]}

분기 구조와 반복 구조에 대해서는 연결된 순차 구조와 비슷한 방법으로 데이터 흐름 분석을 할 수 있다. 분기 구조에서는 프로그램 실행시 조건에 따라 실행 여부가 결정되므로, 프로그램을 정적으로 분석하는 시스템에서는 정확한 데이터 흐름 분석이 용이하지 않다. 보통 이런 경우에는 보수적인 관점에서 분석하는 것이 일반적이나, 병렬 프로그래밍 환경의 일차적인 구현에 목표를 두고 있는 본 논문에서는 특별히 고려할 것이 많은 분기 구조는 고려하지 않고 있다.

본 논문에서 이용한 데이터 흐름 분석은 분석의 기

본 단위라고 할 수 있는 블록을 기준으로 데이터 흐름을 분석하고 각 블록마다 자신의 데이터 흐름 정보와 더불어 현재 블록에 이르는 유효한 데이터 흐름 정보와 현재 블록까지의 데이터 흐름 정보를 관리한다. 그리고, 블록의 최소 단위가 문장이므로, 문장 노드 구조에 데이터 흐름 정보를 포함시켰다. 이렇게 함으로써, 프로그램을 수정하였을 때, 데이터 흐름 정보는 전체적으로 다시 분석되지 않고, 수정한 블록부터 데이터 흐름 정보를 분석하고, 이후의 블록에 대한 데이터 흐름 정보가 수정 이전의 데이터 흐름 정보와 차이가 없으면 데이터 흐름 분석 작업을 마무리함으로써, 변경된 부분만을 고려한 부분적인 데이터 흐름 분석의 수행이 가능하다.

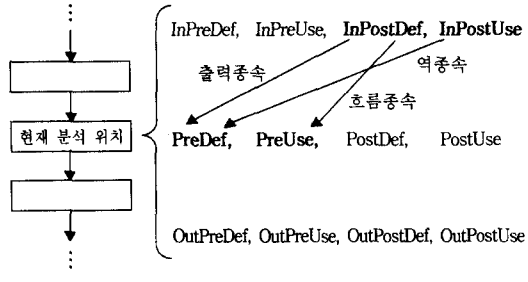
3.3 데이터 종속성 분석

프로그램에서 참조하는 변수 값을 사용 또는 정의하는 문자의 위치와 순서에 따라 여러 가지 형태의 종속성 정보가 나타날 수 있다. 이러한 종속성 정보에는 흐름종속, 역종속, 출력종속이 있다. 흐름종속은 변수의 정의-사용 관계에 있는 문장 사이에 존재하는 종속관계를, 역종속은 사용-정의 관계에 있는 문장 사이의 관계를, 출력종속은 정의-정의 관계에 있는 문장 사이의 관계를 나타낸다[3, 8, 14].

다음과 같은 예에서 보면, 흐름종속 $S_1 \delta S_2$, 역종속 $S_2 \delta^a S_3$, 출력종속 $S_1 \delta^o S_3$ 이 존재한다. 물론, 이러한 정보는 데이터 흐름 분석을 통해 얻어진 데이터 흐름 정보를 기반으로 알 수 있다.

S_1 : A = ...
S_2 : ... = A
S_3 : A =

본 논문에서는 데이터 흐름 정보 분석에서 얻은 외부노출 정보를 이용해서 용이하게 데이터 종속성 분석을 하고 있다. 데이터 종속성 분석에 이용되는 데이터 흐름 정보는 입력 외부노출 정보와 현재 분석 대상이 되는 블록의 외부노출 정보이다. 흐름종속은 InPostDef 집합과 PreUse 집합의 교집합이 되는 변수들 사이에 존재하고, 역종속은 InPostUse 집합과 PreDef 집합의 교집합이 되는 변수들 사이에 존재하고, 출력종속은 InPostDef 집합과 PreDef 집합의 교집합이 되는 변수들 사이에 존재한다. 이들 종속관계는 입력 외부노출 정보에 있는 변수가 종속관계의 소스가 되고, 블록의 외부노출 정보에 있는 변수가 종속관계의 싱크가 된다. (그림 3)은 데이터 흐름 정보와 데이터 종속성 정보와의 관계를 나타낸다. 이 관계를 <표 2>에 사용된 세 개의 문장에 적용해 보면, 흐름종속은 $[1,A_1] \rightarrow [2,A_3]$과 $[2,B_2] \rightarrow [3,B_4]$, 역종속은 $[1,B_1] \rightarrow [2,B_2]$, 그리고 출력종속은 $[2,B_2] \rightarrow [3,B_3]$이 존재함을 알 수 있다.



(그림 3) 데이터 흐름 정보와 데이터 종속성 정보와의 관계

데이터 종속성 분석은 스칼라 변수에 대한 분석과 배열에 대한 분석이 있다. 스칼라 변수에 대한 분석은 데이터 흐름 분석에서 분석된 데이터 흐름 정보를 이용하여 쉽게 얻을 수 있다. 그러나, 배열에 대한 분석은 훨씬 복잡한 과정을 거친다.

루프에서의 종속성에는 루프 반복간 종속성(loop-carried dependence)과 루프 반복내 종속성(loop-independent dependence)이 있다[3]. 다음과 같은 경우를 생각해 보자.

```
        DO I = 2, N
S1 :      A(i) =
S2 :           = A(i)
S3 :           = A(i-1)
        END DO
```

S_1과 S_2 사이에는 루프 구조가 반복 수행되는 과정에서 하나의 반복 내에서만 흐름종속이 존재하는데, 이와 같은 종속성을 반복내 종속성이라고 한다. S_1과 S_3 사이에는 루프 반복간 흐름종속이 존재하는데, 루프의 한 반복을 수행할 때 S_1에서 정의된 것을 다음 반복을 수행할 때 S_3에서 사용한다. 이와 같이 루프의 반복과 반복 사이에 존재하는 종속성을 루프 반복간 종속성이라고 한다.

종속성 정보를 이렇게 구분하는 것은 루프구조의 병렬화 변환에 종속성 정보가 이용되기 때문이다. 루프

구조에서 루프 반복내 종속성만 존재한다면, 모든 루프 반복들을 병렬로 수행할 수 있다. 하지만, 루프 반복간 종속성이 존재한다면, 단순한 병렬 수행은 불가능 하지만 특별한 변환 방법을 사용한다면 제한적인 병렬 수행이 가능하다.

루프 구조에서 루프 반복간 종속성이 존재하는 경우, 병렬화가 가능한 방법을 찾기 위해서 자세한 종속성 정보가 필요하다. 이를 위해서, 배열 변수들의 첨자식 사이의 관계를 분석하게 된다. 이런 종속성 분석에 많은 연구결과가 있었지만, 본 환경에서는 첨자식의 유형에 따라 사용되는 Separability 테스트, Gcd 테스트, 그리고 Banerjee 테스트 등을 이용하여 종속성 분석을 수행한다[8, 14].

중첩된 루프의 데이터 종속성 정보는, 종속관계에 있는 루프 반복들의 루프 제어변수의 값들을 벡터로 표현하면, 종속성은 방향벡터(direction vector)나 거리벡터(distance vector)로 표현된다. 종속관계에 있는 두 배열의 제어변수의 값들을 각각 벡터 $\alpha=(\alpha_1, \cdots, \alpha_n)$와 $\beta=(\beta_1, \cdots, \beta_n)$라고 하면, 거리벡터는 $D=\beta-\alpha=(d_1, \cdots, d_n)$ $d_i=\beta_i-\alpha_i$로 표현된다. 방향벡터는 $d=(d_1, \cdots, d_n)$, $d_i='<'$ if $\alpha_i<\beta_i$ $d_i='='$ if $\alpha_i=\beta_i$, $d_i='>'$ if $\alpha_i>\beta_i$로 표현된다. 이들은 병렬화 변환의 종류에 따라 적절히 이용된다. 예를 들어, 아래와 같은 프로그램에서, 배열요소 A(i+1, j, k-1)와 A(i, j, k) 사이에 흐름 종속이 존재하는데, 거리벡터는 (1, 0, -1)이며, 방향벡터는 (<, =, >)이다. 데이터 종속성 정보를 표현하기 위한 이 두 가지 방법들은 루프 구조의 병렬화 작업을 수행하는 과정에서 유용하게 이용되는데, 거리벡터가 일정한 경우에는 위의 예제와 같이 거리벡터와 방향벡터를 정확하게 구할 수 있지만, 거리벡터가 일정하지 않은 경우에는 방향벡터만을 이용하여 종속성 정보를 표현할 수밖에 없다[9].

```
DO i = 1, n
  DO j = 1, m
    DO k = 1, l
      A(i+1, j, k-1) = A(i, j, k) + C
    END DO
  END DO
END DO
```

데이터 종속성 정보가 가지고 있는 잘 알려진 특성은 다음과 같다. 데이터 종속성 정보를 방향벡터로 표현하였을 때, 모든 벡터요소가 '='이든가, 아니면 처음으로 '='이 아닌 벡터요소는 '<'이다[6]. 모든 벡터요소가 '='이면 종속성이 루프 반복내 종속성임을 의미한다. 처음부터 i-1번째 벡터요소가 '='이고, i번째 벡터요소가 '<'라면, i번째 루프에서 루프 반복간 종속성임을 의미한다. 이때, i번째에서 이미 실행 순서에 따른 종속성이 나타났으므로, i번째 이후에 나타나는 벡터요소는 '=', '<', '>' 중에서 어느 것이라도 나타날 수 있다[14].

본 환경에서는 데이터 종속성 분석의 결과를 추상구문트리의 문장 노드가 데이터 종속성 정보를 가지고 있도록 하였다. 이것은 병렬화 변환이 문장 단위로 이루어지기 때문에, 병렬화 과정에서 문장 단위의 데이터 종속성 정보에 대한 참조 및 갱신을 용이하게 하기 위함이다. 또한, 본 환경에서는 데이터 흐름 분석과 마찬가지로 사용자의 의도에 따라 전체적인 혹은 부분적인 데이터 종속성 분석을 수행할 수 있다. 일반적으로 원시 입력 프로그램에 대한 병렬화 수행 과정에 있어서 그 대상은 루프 구조이다. 그러므로, 루프 구조에 대한 병렬화를 위해서는 전체적인 데이터 종속성 분석보다는 부분적인 데이터 종속성 분석이 더 효율적이다. 이러한 점을 고려하여, 데이터 흐름 분석과 마찬가지로 데이터 종속성 분석 작업 역시 전체적인 분석과 부분적인 분석 두 가지 모두를 지원하도록 하고 있다.

3.4 루프 구조의 병렬화

일반적으로 프로그램의 성능에 가장 큰 영향을 끼치는 부분은 루프 구조이다. 루프 구조는 비록 원시 프로그램 수준에서는 전체적인 코드 중에서 지극히 적은 분량이라도 반복 수행을 하기 때문에 프로그램 작업 시간 면에서는 대부분의 시간을 소비하고 있는 부분이다. 따라서, 대부분의 병렬 프로그래밍 환경에서 병렬화 과정의 의미는 루프 구조에 대한 변환 작업을 의미하는 것이 보통이다. 본 논문에서는 이런 이유로 병렬화의 대상을 원시 입력 프로그램의 루프 구조로 제한한다.

원시 입력 프로그램을 변환하는 과정에서 가장 중요한 것은 프로그램의 의미가 변하지 않아야 된다는 점이다. 이것을 위해서는, 원시 입력 프로그램을 병렬화하는 과정에서 데이터 종속성 정보를 이용한다. 루프 구조를 병렬화하는 과정은 병렬 컴퓨터의 특성을 효율적으로 활용할 수 있고 병렬화가 가능한 루프가 있어

야만 한다. 그러나, 이런 조건을 만족하지 않는 경우에는 루프 구조 변환 방법들을 이용하여 원하는 병렬화된 루프 구조를 얻을 수 있다. 루프 구조의 변환에는 루프 교환, 루프 분리, 루프 융합 등이 있다[14].

루프 구조의 병렬화는 데이터 종속성 분석을 통해 얻은 데이터 종속성 정보를 활용하여 이루어진다. 루프 구조를 병렬화해도 원시 입력 프로그램의 의미가 달라져서는 안되므로, 병렬화로 인해서 데이터 종속성 정보의 특성이 지켜지지 않으면 안 된다.

단순한 병렬화에 이용되는 데이터 종속성 정보는 방향벡터이다. 방향벡터를 보고 병렬 수행 가능성과 다른 변환 가능성을 파악하게 된다. 방향벡터에서 벡터 요소가 '='이면 해당 루프에서는 루프 반복간 종속성이 없기 때문에 모든 루프 반복을 병렬로 수행할 수가 있다. 또한 '='이 아닌 첫 번째 벡터요소가 '<'이라는 조건만 만족되면, 벡터요소들끼리 순서를 바꾸어도 관계없다. 이런 특성을 이용하여 효율적인 병렬 프로그램을 생성할 수가 있는 것이다[3, 9, 14].

병렬화에는 미세단위 병렬화(fine grain parallelization)와 대단위 병렬화(coarse grain parallelization)로 크게 나눌 수가 있다. 벡터처리기를 가지고 있는 병렬 컴퓨터에서는 벡터처리를 위해 미세단위 병렬화를 지향하고, 독립된 컴퓨터들을 연결하여 구성한 병렬 컴퓨터에서는 대단위 병렬화를 지향하여 병렬화의 효율을 높인다. 미세단위 병렬화를 위해서는 방향벡터에서 마지막 벡터요소가 '='이 되도록 변환하고, 대단위 병렬화를 위해서는 앞부분의 벡터요소가 '='이 되도록 변환한다.

병렬화 변환에는 많은 방법이 있지만, 이곳에서는 대표적인 것 몇 가지만 살펴보기로 한다. 병렬화 대상으로 하고 있는 루프 구조가 다중 루프일 경우, 미세단위 병렬화나 대단위 병렬화를 위해서 루프의 순서를 변경할 수가 있다. 예를 들어 다음과 같은 다중 루프를 보자.

```
DO i = 1, N
   DO j = 1, N
      DO k = 1, N
         A(i, j, k) = A(i-1, j, k+1) + ...
      END DO
   END DO
END DO
```

위의 루프에 존재하는 데이터 종속성의 거리벡터는 (1, 0, -1)이고, 방향벡터는 (<, =, >)이다. 루프 j에 대한 벡터요소가 '='이므로, 루프 j는 병렬수행이 가능하다. 따라서 미세단위 병렬화를 위해서는 루프 j와 루프 k의 위치를 교환해서 방향벡터가 (<, >, =)이 되도록 하면 벡터화가 가능하게 된다. 그러나 대단위 병렬화를 위해서는 루프 i와 루프 j의 위치를 교환해서 방향벡터가 (=, <, >)이 되도록 해서, 많은 부분이 병렬로 수행할 수 있도록 변환한다.

루프 융합은 대단위 병렬화를 목적으로 하는 경우 사용된다. 이 방법은 이웃하고 있는 두 루프를 하나의 루프로 변환하는 것으로 루프 변수의 영역이 동일해야 하며, 루프 융합이 이루어진 후 원시 입력 프로그램의 데이터 종속성 정보가 변하지 않고 유지되어야 한다.

루프 분리는 하나의 루프 안에 있는 모든 문장들이 종속 관계에 있어서 그 상태로는 병렬화를 할 수 없을 때 사용하는 변환 방법으로, 데이터 종속성 정보에 따라 좀 더 작은 단위의 그룹으로 나누어 두 개 이상의 루프에 분할하는 것으로, 미세단위 병렬화를 목적으로 주로 사용된다. 이 방법에서는 먼저 데이터 종속성 분석을 통해 구성된 데이터 종속성 정보를 표현한 그래프에서 강연결 요소들을 찾은 후 이들에 대해 위상 정렬을 수행하고 각 강연결 요소들을 순서대로 하나의 루프로 만들어 준다.

현재 나와있는 많은 병렬 컴퓨터들은 벡터 처리기를 가지고 있다. 이러한 병렬 컴퓨터들은 벡터 명령을 이용하여 벡터화가 이루어진 데이터들을 빠르게 처리할 수 있다. 벡터화는 배열에 대해서 일어나는데, A(1 : 100 : 3)과 같이 벡터처리를 대상이 되는 배열 요소의 시작 인덱스, 마지막 인덱스, 그리고 스트라이드로 표현한다.

4. 병렬 프로그래밍 사용자 환경

본 장에서는 본 논문에서 설계하고 구현한 병렬 프로그래밍 환경의 사용자 환경을 설명하고, 시나리오를 기반으로 각각의 기능들을 알아본다.

4.1 사용자 환경

본 병렬 프로그래밍 환경은 편집기를 기반으로 한 사용자 환경을 가지고 있다. 본 사용자 환경은 사용자에게 X Window를 기반으로 하는 그래픽 사용자 인터페이스를 제공한다. 이것은 사용자와 병렬 프로그래밍 환경 사이의 상호 작용을 용이하게 하기 위함이다. 사

용자와 병렬 프로그래밍 환경 사이에서 이루어지는 상호 작용은 병렬 컴퓨터의 하드웨어적인 특성과 병렬화의 어려움으로 인하여 병렬 프로그램 작성에 곤란을 겪고 있는 사용자들이 병렬 프로그램을 용이하게 작성할 수 있도록 도와준다. 이렇게 함으로써, 프로그래머가 의도한 대로 병렬화된 병렬 프로그램 작성이 가능하다.

본 사용자 환경은 기능적인 측면에서 다음과 같은 세 가지 기능을 가진다. 하나는 사용자의 프로그램 작성을 도와주는 편집 기능이며, 다른 하나는 데이터 종속성 정보와 같은 정보들을 사용자에게 보여주는 정보 제공 기능이며, 마지막으로 제공되는 기능은 병렬화 기능이다.

편집 기능은 사용자가 편집기를 이용하여 프로그램을 작성할 때 병렬 프로그래밍 환경이 사용자에게 제공해 주는 기능들을 의미한다. <표 3>의 주 메뉴 부분의 <File>과 <Edit>에서 제공하는 기능들이 이에 해당된다. 이 기능은 일반적으로 이용되는 편집기의 기능과 역할이 비슷하여 사용자들이 쉽게 사용할 수가 있다.

〈표 3〉 사용자 환경의 메뉴 이름과 기능

주 메뉴	부 메뉴		기 능
File	New		새로운 파일을 작성
	Open		기존에 작성된 파일을 열기
	Save		작성한 파일을 저장
	Save as		작성한 파일을 새로운 이름으로 저장
	Quit		프로그램의 종료
Edit	Copy		편집 내용 복사하기
	Cut		편집 내용 잘라내기
	Paste		편집 내용 붙여넣기
	Clear		편집 내용 지우기
Syntax	Parsing		해당 파일에 대한 문법 검사
Analysis	DFA	DFA	해당 파일에 대한 전체적인 데이터 흐름 분석
		Blocked DFA	해당 파일에 대한 부분적인 데이터 흐름 분석
	DDA	DDA	해당 파일에 대한 전체적인 데이터 종속성 분석
		Blocked DDA	해당 파일에 대한 부분적인 데이터 종속성 분석
Loop Trans	Loop interchange		루프 교환
	Loop distribution		루프 분리
	Loop fusion		루프 융합
	Stmt Reordering		문장 교환
	Vectorization		루프 벡터화
	Unvectorization		벡터 문장의 펼침
	Concurrentization		루프 병렬화
Option	Print DFA		데이터 흐름 정보 자세히 보기
	Print DDA		데이터 종속성 정보 자세히 보기
Pretty	indentation		작성한 프로그램에 대한 Indentation 자동 설정

정보 제공 기능은 본 병렬 프로그래밍 환경이 프로그램 대상 언어로 지정한 HPF에 대한 문법 검사와 데이터 흐름 분석, 데이터 종속성 분석 기능 등을 수행하여 사용자에게 제공할 수 있는 기능을 말한다. <표 3>의 주 메뉴 부분에서 <Syntax>, <Analysis>, <Option>, 그리고 <Pretty> 라는 메뉴가 이에 해당된다. 이 기능은 병렬 프로그래밍 환경이 사용자에게 제공해 주어야 할 여러 가지 정보들을 사용자의 요구에 따라 제공해 주는 것이다. 사용자는 병렬 프로그래밍 환경으로부터 원시 입력 프로그램에 대한 병렬화를 수행하는 과정에서 필요한 기본적인 데이터 종속성 정보들을 제공받고, 데이터 흐름 정보와 좀 더 자세한 데이터 종속성 정보들도 필요하다면 제공받을 수 있다. 이를 위해서 프로그램의 병렬화 작업을 수행할 때 사용자가 데이터 종속성 정보를 좀 더 쉽게 알아볼 수 있도록 데이터 종속성 정보를 보여주는 대화창을 별도로 제공하고 있다. 사용자는 데이터 종속성 정보를 알기 위해 데이터 종속성 분석 명령을 내리면, 데이터 종속성 정보가 이 대화창을 통해 사용자에게 보여진다. 사용자는 이 대화창을 통해 모든 데이터 종속성 정보를 알아볼 수 있으며, 사용자의 필요에 따라 좀 더 자세한 정보도 볼 수 있도록 하고 있다.

주 메뉴 <Syntax>에는 <Parsing>이라는 기능이 제공되는데, 이는 편집 창에 있는 프로그램에 대해서 파싱을 실행하여 추상구문트리를 생성해 준다. 주 메뉴 <Analysis>에는 <DFA>와 <DDA>가 있는데, 각각 데이터 흐름 분석과 데이터 종속성 분석을 실행한다. 이들 기능은 프로그램 전체에 대해서 또는 설정된 블록에 대해서 분석할 수 있도록 더 자세한 하부 메뉴를 가지고 있다. 또한 <Pretty>는 편집 창에 있는 프로그램의 구조를 잘 표현하도록 인덴테이션을 자동으로 설정해 주어, 프로그램의 오류를 쉽게 알 수 있는 기능이다.

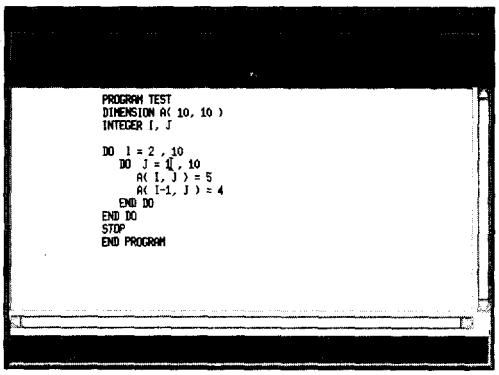
병렬화 기능은 본 병렬 프로그래밍 환경이 제공하는 7가지의 루프 구조 변환들을 수행하는 것을 의미한다. <표 3>의 주 메뉴 부분에서 <Loop Trans>라는 메뉴가 이에 해당된다. 현재로서는 루프 교환, 루프 분리, 루프 융합, 문장 교환, 루프의 벡터화, 벡터 문장의 펼침, 루프의 병렬화 기능을 수행하도록 하부 메뉴를 가지고 있다.

4.2 시나리오

사용자 환경을 이용하여 병렬 프로그래밍 과정을 살

펴보기 위해서 루프 교환을 수행하는 과정을 예로 들면 다음과 같다.

사용자는 병렬 프로그래밍 환경을 시작하고, 병렬 프로그램의 작성을 위해 메뉴에서 <File>과 <Open>을 차례로 선택해서 파일 열기 명령을 수행한다. 그러면 파일 선택을 위한 대화상자가 나타난다. 이 대화상자에서 편집하기 원하는 파일을 선택한다. 그러면 (그림 4)와 같이 선택된 파일의 내용이 보여지는 편집기 창이 나타난다. 사용자는 필요에 따라 프로그램을 수정해서 저장할 수가 있다. (그림 4)에서는 sample0.f라는 프로그램이 나타나 있다.

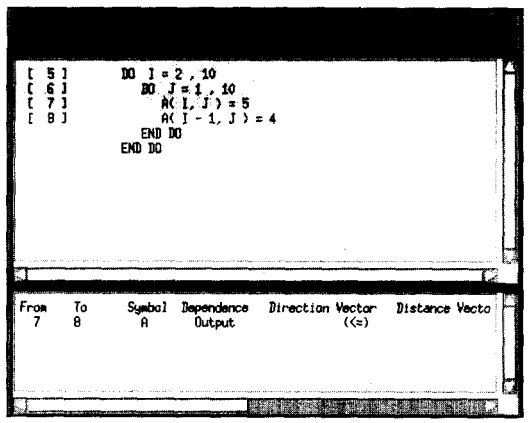


(그림 4) 편집기 창의 모습

이 프로그램을 병렬화하기 위해 사용자는 우선 어휘분석과 구문분석을 수행해야 하는데, 이를 위해 메뉴에서 <Syntax>와 <Parsing>을 선택한다. 어휘분석과 구문분석의 수행이 완료되면, 추상구문트리가 내부적으로 생성된다.

그후 사용자는 데이터 흐름 분석을 수행한다. 데이터 흐름 분석은 메뉴에서 <Analysis>-<DFA>-<DFA>를 선택하여 프로그램 전체에 대한 분석을 할 수도 있고, 병렬화 변환 대상으로 생각하고 있는 루프를 먼저 선택한 후, 메뉴에서 <Analysis>-<DFA>-<Blocked DFA>를 선택하여, 선택된 루프 구조에 대해서만 분석할 수도 있다. 데이터 흐름 분석이 완료되면 데이터 흐름 정보가 추상구문트리에 추가되어 저장되어있게 된다.

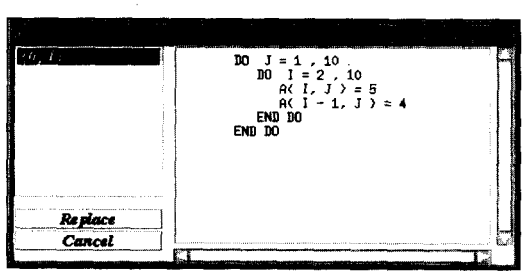
다음으로 데이터 종속성 분석을 수행해야 한다. 데이터 종속성 분석도 데이터 흐름 분석과 마찬가지로 프로그램 전체에 대하여 분석할 수도 있고, 선택된 루프 구조에 대해서만 분석할 수도 있다. 데이터 종속성 분석을 수행하면 데이터 종속성 정보를 보여주는 창에 분석 결과가 나타나게 된다. 예로 보여주는 시나리오에서는 루프 변수가 I인 DO 루프를 블록으로 선택하여 분석하도록 하였고, 그 결과가 (그림 5)에 나타나 있다. 그림에 의하면 배열 A가 문장 7번에서 8번으로 출력종속 관계에 있고, 방향 벡터는 (<, =)임이 나타나 있다.



(그림 5) 데이터 종속성 분석 결과

사용자는 이와 같은 종속성에 관한 정보를 보고 루프 변환 방법 중에서 루프 교환 가능성을 판단한다. 예에서는 루프 J가 병렬화가 가능하므로, 만약 벡터화가 목적이라면 루프 J를 벡터화하면 되고, 대단위 병렬화가 목적이라면 가능한 루프 I와 루프 J를 교환하여 방향벡터가 (=, <)이 되도록 하여, 루프 J를 병렬화 하면 된다.

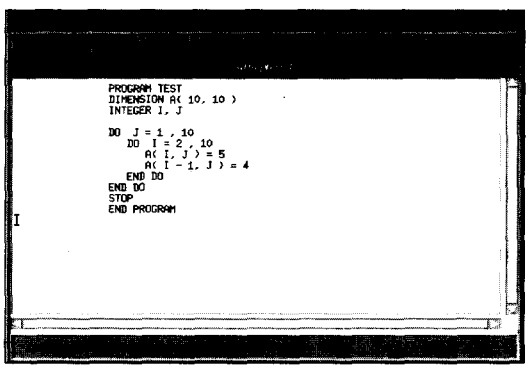
(그림 6)은 메뉴에서 <Loop Trans>-<Loop interchange>를 선택하여 루프 교환 명령을 수행한 결과를 보여주고 있는데, 이 창에서, <Replace>를 선택하여 정말로 프로그램을 변환할 것인지, 아니면 <Cancel>을 선택하여 취소할 것인지를 선택할 수 있도록 한다. 그



(그림 6) 루프 교환 대화 상자

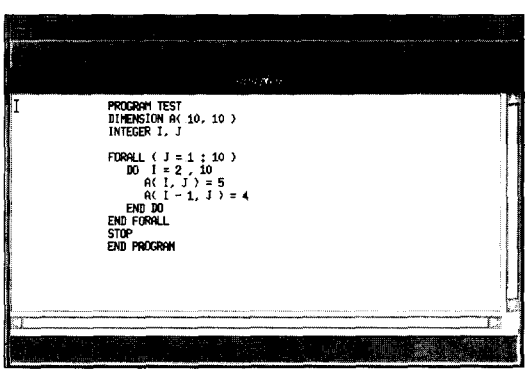
리고, 루프 교환을 위한 대화 상자에서 자신이 원하는 루프 교환을 선택하여 실행한다.

(그림 7)은 (그림 6)에서 <Replace>를 선택하여 루프 교환을 수행한 결과가 편집기에 보여진 모습을 나타내고 있다. 이와 같이 루프가 교환된 후에는 종속성 정보의 방향벡터가 (=, <)이 되어, 바깥쪽 루프를 병렬화할 수가 있게 된다.



(그림 7) 루프 교환 수행 결과

(그림 8)은 (그림 7)에서 <Loop Trans>-<Concurrentization>을 선택하여 바깥쪽 루프를 병렬화하여 FORALL로 바뀐 모습을 보여주고 있다.



(그림 8) 병렬화 수행 결과

5. 결론 및 향후 연구 방향

본 논문에서는 일반적인 병렬 프로그래밍 환경에 대해 소개하고, 설계하고 구현한 병렬 프로그래밍 환경에 대해 기술하였다. 병렬 프로그래밍 환경을 설계하고 구현하는데 있어서, 병렬 컴퓨터의 하드웨어적인 특성을 고려한 병렬 프로그램의 작성을 위해, 사용자와의 상호작용을 고려하여야 한다. 이러한 특성을 가지는 병렬 프로그래밍 환경을 설계하고 구현하기 위해 X-window를 기반으로 하는 그래픽 사용자 인터페이스 환경을 구축하고, 사용자는 병렬 프로그래밍 환경에서 제공하는 여러 가지 기능들을 용이하게 수행할 수 있다.

본 논문에서 설계하고 구현한 병렬 프로그래밍 환경에서는 원시 입력 프로그램의 병렬화 과정에서 이용되는 데이터 흐름 정보와 데이터 종속성 정보를 추상구문트리의 문장 노드를 확장하여 저장한다. 이것은 원시 입력 프로그램의 병렬화 과정에서 데이터 흐름 정보와 데이터 종속성 정보를 효율적으로 접근할 수 있도록 해 준다. 또한, 원시 입력 프로그램에 대한 수정이 이루어진 경우, 데이터 흐름 정보와 데이터 종속성 정보의 갱신이 효율적으로 이루어지도록 한다.

본 논문에서 설계하고 구현한 병렬 프로그래밍 환경에서는 데이터 흐름 분석 단계에서 외부노출 정보를 이용하여 다음 단계인 데이터 종속성 분석을 용이하게 수행할 수 있도록 하였다. 데이터 종속성 분석에서는 잘 알려진 Separability 테스트, Gcd 테스트, 그리고 Banerjee 테스트 등을 이용하였으며, 그 결과를 이용하여 루프 교환, 루프 융합 등의 루프 변환 작업들을 수행한다.

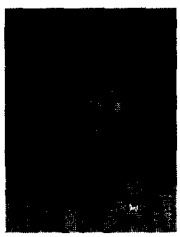
향후 연구 방향으로는 기존에 작성된 데이터 흐름 분석과 데이터 종속성 분석의 효율을 극대화하고, 본 병렬 프로그래밍 환경이 운영될 병렬 컴퓨터의 하드웨어적인 특성을 고려하여 병렬 디버거와 성능 분석기, 그리고 데이터 매핑 지원 도구 등의 기능들이 첨가되어야 하겠다. 그렇게 되면 하드웨어의 특성에 따라 좀 더 최적화된 병렬 프로그램 작성이 가능하게 될 것이다.

참 고 문 헌

[1] A. V. Aho, R. Sethi and J. D. Ullman, "Compilers : Principles, Techniques, and Tools," Addison-Wesley, 1986.

[2] F. Allen, M. Burke, P. Chales, R. Sytron and J. Ferrante, "An Overview of the PTRAN Analysis System for Multiprocessing," Journal of Parallel and Distributed Computing, Vol.5, No.5, pp.617-640, Oct. 1988.

[3] J. R. Allen and K. Kennedy, "Advanced Compilation for Vector and Parallel Computers," Morgan Kaufman Publishers, Inc., 1994.
[4] J. R. Allen and K. Kennedy, "A Parallel Programming Environment," IEEE Software, Vol.2, No.4, pp.21-29, Jul. 1985.
[5] J. R. Allen and K. Kennedy, "Automatic Translation of Fortran Programs to Vector Form," ACM Transactions on Programming Languages and Systems Vol.9, No.4, pp.491-542, Oct. 1987.
[6] U. Banerjee, "Dependence Analysis for SuperComputing," Kluwer Academic, 1988.
[7] V. Guarna Jr., D. Gannon, et. al., "Faust : An Integrated Environment for Parallel Programming," IEEE Software, pp.20-27, Jul. 1989.
[8] S. Hiranandani, K. Kennedy, C. W. Tseng and S. Warren, "The D Editor : A New Interactive Parallel Programming Tool," Technical Reports, CRPC, 1991.
[9] K. Kennedy and K. S. McKinley, "Analysis and Transformation in the ParaScope Editor," Technical Report CRPC-TR 90106, Rice University, Dec. 1990.
[10] K. Kennedy, K. S. McKinley and C. W. Tseng, "Interactive Parallel Programming Using the Parascope Editor," Technical Report, CRPC-TR 90096, Oct., 1990.
[11] D. E. Knuth, "An Empirical Study of Fortran Programs," Software Practice and Experience, 1, pp. 105-133, 1971.
[12] C. D. Polychornopoulos, M. R. Haghighat and C. L. Lee, "The Structure of Parafrase-2 : An Avanced Parallelizing Complier for C and Fortran," Center for Supercomputing Research and Development, University of Illinis at Urbana-Champaign.
[13] K. Smith and W. F. Appelbe, "PATAN Interactive Fortran Parallelizing Assistant Tool," Georgia Institute of Technology.
[14] H. Zima and B. Chapman, "Supercompliers for Parallel and Vector Computers," Addison-Wesley, 1991.



유 정 목

e-mail : jmyoo@cs.cnu.ac.kr
1992년~1996년 충남대학교 컴퓨터과학과 졸업(학사)
1996년~1998년 충남대학교 컴퓨터과학과 졸업(석사)
1998년~현재 충남대학교 컴퓨터과학과 박사과정(휴학중)
1998년~현재 공군 장교 복무
관심분야 : 병렬프로그래밍 환경, 병렬처리, 분산처리

이 동 희

e-mail : middle@postech.ac.kr
1991년~1995년 충남대학교 전산학과 졸업(학사)
1995년~1997년 충남대학교 전산학과 졸업(석사)
1997년~1998년 현대전자산업(주) 통신연구소 연구원
1998년~현재 포항공대 공정산업의 지능자동화연구센터 연구원
2000년~현재 아이시스텍(주) 소프트웨어개발팀장
관심분야 : 병렬프로그래밍 환경, 병렬처리, 분산처리

이 만 호

e-mail : mhlee@cs.chungnam.ac.kr
1971년~1975년 서울대학교 공과대학 응용수학과 졸업(학사)
1975년~1977년 한국과학기술원 전산학과 졸업(석사)
1984년~1991년 인디아나대학교 전산학과 졸업(박사)
1977년~1980년 국방과학연구소 연구원
1980년~1984년 충남대학교 계산통계학과 교수
1991년~1999년 충남대학교 컴퓨터과학과 교수
1999년~현재 충남대학교 정보통신공학부 교수
관심분야 : 병렬프로그래밍 환경, 병렬컴파일러, 정보검색, 디지털도서관