PARAFRASE II 분석을 통한 병렬 컴파일러에 관한 연구

이 상헌°, 박 두순
순천향대학교 공과대학 컴퓨터학부

A Study on Parallel Compiler Using PARAFRASE II Analysis

Sang-Heon Lee°, Doo-Soon Park
Dept. of computer science, College of Engineering
Soonchunhyang University

요 약

정보화 시대를 맞이해서 다량의 자료와 다양한 형태의 자료를 빠르게 처리함이 요구됨에 따라서 좀더 성능이 좋은 컴퓨터를 요구하였으며, 이러한 요구사항에 충족하기 위하여 병렬 컴퓨터가 등장하였다. 그러나, 병렬 컴퓨터에서 수행되는 병렬 프로그램을 사용자가 작성하기란 용이한 작업이 아니다. 사용자에게 편의를 제공하고, 기존의 프로그램들을 병렬 컴퓨터에서 직접 수행시키기 위하여 병렬 컴파일러가 등장하였다. 이러한 병렬 컴파일러를 만드는데는 상당한 시간과 노력이 요구된다. 본 논문에서는 PARAFRASE-2 병렬 컴파일러의 분석을 통하여 병렬 컴파일러의 구조를 살펴보고 간단한 병렬 컴파일러를 설계, 구현한다.

1. 서 론

현대 사회가 점차 정보화 사회로 바뀌어 감에 따라, 예전의 고속의 수치 연산 처리 목적의 컴퓨터에서 점차 통신 수단의 고속화를 뒷받침할 수 있고, 다양한 형태의 정보 즉, 음성, 영상, 문자 등을 신속히 처리할 수 있는 고성능의 컴퓨터를 요구하게 되었다. 이러한 목적에 충당하는 컴퓨터가 다양한 형태로 등장하였으며, 그 하나의 형태가 병렬 컴퓨터이다. 이런 병렬 처리 컴퓨터의 성능을 극대화시키기 위해서는 사용자가 문제에 따른 최대의 병렬 프로그램으로 작성해야한다.

그러나, 일반적으로 병렬 컴퓨터를 위한 병렬 프로그램을 작성하는 것은 일반 순차 프로그램을 작성하는 것 보다 훨씬 복잡한 작업과 많은 시간이 요구된다. 병렬 프로그램을 작성하기 위해서는 작성자는 병렬로 수행이 가능한 작업들을 추출해 내고 적절한 시기에 동기화를 시켜 주어야 하며 자료의 공유로 인한 부담을 감소하기 위하여 자료의 분할이나, 중복을 고려하여야 한다. 또한 이러한 작업들은 특정 기계의 구조에 의존적인 특성을 가지고 있다.

이러한 병렬 프로그램의 작성을 쉽게 하기 위하

여 개발된 것이 병렬 컴파일러이다. 병렬 컴파일러는 프로그래머가 작성한 일반 순차 프로그램을 자동으로 병렬 프로그램으로 바꿔주는 일종의 전처리기(preprocessor)이다. 이러한 병렬 컴파일러의 목적은 프로그래머가 부담하는 작업량을 최소화시키고, 병렬 프로그램 작성의 효율을 극대화하는데 있다.

병렬화 컴파일러는 C나 Fortran과 같은 순차 프로그램 입력에 대하여 다양한 분석과 코드 변환 과정을 거쳐, 병렬 언어 프로그램이나 Message Passing Interface(MPI)[6], 병렬 쓰레드(thread)와 같은 병렬 라이브러리를 호출하는 프로그램을 생성해 낸다.

이러한 재구성 컴파일러는 다음과 같은 유용성을 가진다. 첫째, 이미 개발된 순차 프로그램을 다시 작성할 필요 없이 쉽게 병렬 프로그램으로 바꿀 수 있어서, 새로운 병렬 컴퓨터에서 신속하게 사용할 수 있게 한다. 둘째, 프로그래머가 병렬 프로그램 대신에 순차 프로그램을 작성하도록 함으로써 기존의 순차 프로그램에 익숙해져 있던 프로그래머가 병렬 컴퓨터를 위한 프로그램을 쉽게 작성할 수 있게 된다. 마지막으로, 병렬 컴퓨터를 위한 프로그램은 실행시킬 컴퓨터 구조에 따라 병렬 수행 구조(primitive)나, 고려 사항이 다르기 때문에, 재구성 컴파일러를 사용함으로써 프로그램의 이식성(portability)도 높일 수 있다[14].

지금까지 재구성 컴파일러의 연구는 메시지 교환(message passing) 방식의 병렬 프로그램 생성보다는, 공유 메모리(shared memory)방식의 병렬 컴퓨터에서 유용한 병렬 반복문(loop)생성이나 벡터 연산의 생성을 중심으로 수행되어 왔다. 그 결과 Illlinois 대학의 Parafrase[3], Polaris[2, 15], IBM의 Ptran[14], Bonn대학의 SuperB[1], Stanford 대학의 Suif[9, 17], Pacific Sierra의 VAST, Rice 대학의 PFC[1], Kuck and Association의 KAP[8] 등의 다양한 병렬화 컴파일러가 발표되었다.

그러나 이러한 병렬 컴파일러의 제작은 많은 시간과 노력이 요구되며, 상당히 어려운 작업이다. 그러므로, 기존의 병렬 컴파일러를 분석하여 해당 소스 부분을 갱신하는 것이 새로운 병렬 컴파일러를 만드는데 최선의 방법중의 하나라고 할 수 있다.

본 논문의 목적은 Illinois 대학에서 만든 Parafrase-2를 분석하여, 병렬 컴파일러의 구조를 알아내고, 간단한 병렬 컴파일러를 설계·구현한다.

2. 병렬 컴파일러종류[11]

2.1.1.Parafrase[1,3,16]

Parafrase는 Univ. of Illinois at Urbana Champaign에서 개발된 최초의 병렬화 컴파일러로서 많은 병렬화 컴파일러에 영향을 주었다.

Parafrase의 입력은 Fortran 프로그램이며 출력은 확장된 병렬 Fortran 프로그램과 변환에 대한 오류, 성능 예상 등의 기록이 된다. 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.

Parafrase의 후속 컴파일러인 Parafrase2는 병렬화 정보가 추가된 추상 구문 트리를 내부 구조로 하여 C, Fortran, Pascal등의 다양한 언어를 지원하는 병렬화 컴파일러이다. Parafrase2는 변수값 추적 분석(symbolic analysis)[7]을 이용한 의존성 분석, 유도 변수 검출과 프로시저간 분석 등이 보강되었다.

2.1.2. PFC[1]

PFC는 Rice 대학에서 개발한 벡터화 컴파일러(vectorizing compiler)이다. 중간 표현으로 추상 구문 트리가 사용되며 Fortran66또는 Fortran77 프로그램을 입력으로 받아 프로시저간 분석, 전처리 코드 변환, 의존성 분석, 벡터 코드 생성의 네 가지 과정을 거쳐 Fortran90 프로그램을 출력한다.

PFC는 Parafrase에 비하여 상대적으로 적은 수의 분석 및 변환 알고리즘을 적용하고 있으며, 각각의 패스가 내부 자료 구조에 한꺼번에 적용되기 때문에 각각의 패스가 독립적으로 적용되는 Parafrase에 비하여 실행속도가 빠른 특징을 갖는다.

2.1.3. SUIF[9,10]

SUIF은 Stanford 대학에서 개발한 컴파일러 개발 환경이다. SUIF에서는 중간 표현으로서의 SUIF언어

와 Fortran-to-C변환기, C-to-SUIF변환기, SUIF-to-C변환기 등의 개발 도구 및 SUIF 중간 표현을 조작할 수 있는 라이브러리, 각종 프로그램 변환 라이브러리를 제공하고 있다. SUIF을 이용하여 프로그램 최적화기와 병렬화 컴파일러, MIPS 목적 코드 생성기 등이 개발된 상태이며 또한 다양한 최적화 컴파일러들이 여러 대학에서 개발되고 있다.

SUIF 병렬화 컴파일러는 Fortran과 C를 입력으로 받아 쓰레드 라이브러리를 호출하는 C프로그램을 생성한다. 코드 변환부에서는 Wolf등이 제안한 유니모듈러 transformation을 사용하여 병렬 반복문을 생성하는 것이 특징이며, 반복문 분할과 병합은 지원하지 않는다.

2.1.4. Polaris[2, 15]

Polaris는 UIUC에서 개발된 최신의 병렬화 컴파일러중의 하나이다.

Polaris의 개발은 Perfect Benchmark의 분석기반으로 하여 이루어졌는데, 기존의 실용적인 응용들이 실제로 효과적인 병렬화가 가능함을 확인하고 이를 자동으로 수행하기 위하여 필요한 변환 및 분석 기법을 구현하는 접근법을 취하였다.

Polaris에서 강조한 분석 및 변환 기법으로는 프로시저간 분석 시에 강력한 프로시저 확장(inline expansion)을 사용한 것과 변수값 추적 분석을 사용한 비선형 첨자의 의존성 분석, 유도 변수와 축약식의 대치, 스칼라 및 배열 지역화(privatization), 수행시간(run time)의 프로그램 분석을 통한 병렬 수행 등이 있다.

3. Parafrase-2 분석

Parafrase-2는 크게 Pass별로 구성되어 있으며, 그 Pass는 여러 파일들이 관련되어있다. 각 Pass에 속한 파일들을 찾아내고, 각 Pass가 실행시 처음 호출하는 함수를 찾아본다. 그 다음 그 함수와 연관된 함수들을 찾는다.

본 논문에서 사용한 Parafrase-2는 1995년 8월 29일에 release된 version 5.0.5를 사용하였다.

Parafrase-2는 아래와 같은 명령으로 순차프로그램을 병렬프로그램으로 바꾸어준다.

p2fpp [-p] [pass_file] [source_file]

[-p]는 패스들을 패스들이 저장된 파일로부터 입력받는다는 옵션이며, [pass_file]은 pass들이 저장된 파일이다. [source_file]는 병렬프로그램으로 바꿀 순차처리 프로그램이다. pass file의 예를 보면 아래와 같다.

```
fixup
callgraph /dev/null
sumfcn -d0
libsum -d0
param_alias
donest -d0
depend -d0
flow
constant
induction -d000
dotodoall -d0
codegen -d0 test.out
```

다음 부절에서는 이런 패스들의 특징과 각 패스들이 속한 파일들과 각 패스의 초기 호출 함수를 살펴본다.

3.1. callgraph

간단한 callgraph를 만든다. 아직까지 이 callgraph 패스는 절차상의 인수들로 여겨지지 않는다.

이 패스가 존재하는 파일은 src/analyses/call_graph/callgraph.c이며, 초기 호출 함수는 call_callgraph()이다.

3.2. sumfcn

sumfcn은 Parafrase2의 프로시저들의 참조 정보를 분석하고 요약하는 역할을 하는 pass이다. 이 pass가 존재하는 디렉토리와 파일은 src/analyses/interprocedure/procedure_reference_info/sumfcn.c이며, 초기 호출 함수는 call_sumfcn()이다.

3.3. libsum

이 pass는 어떤 소스 코드가 이용되지 않는지의 정보를 살핀다. 관련 정보는 사용자가 쉽게 갱신할 수 있게 끔 text 파일로 저장되어있다.

이 패스의 소스가 존재하는 디렉토리는 src/analyses/interprocedure/read_library_summary/libsum.c이며, 초기 호출 함수는 call_libsum()이다.

3.4. param_alias

이 pass는 formal parameter로 인해 생기는 aliasing 정보를 만들고 심볼 테이블에 저장한다. 또한, 전역변수까지 고려한다.

이 패스의 소스가 존재하는 디렉토리 src/analyses/interprocedure/parameter_alias/alias.c 이며, 초기 호출함수는 call_param_alias()이다.

3.5. donest

이 패스는 각 문장의 중첩 단계(nesting level)를 생성한다. 각 문장에 대하여 T_NEST는 각 문장이 중첩된 Do loop의 목록을 가리키기 위하여 갱신된다. 이 정보는 나중에 loop induction variable을 확인하기 위하여 depend pass에서 사용된다. 에러 메시지와 출력은 파일로 저장되며, 그 파일의 이름은 각 패스의 argument로 쓰인다.

이 패스의 소스 파일이 존재하는 경로는 src/misc/control/interface.c이며, 초기 호출 함수는 call_donest_pass()이다.

3.6. depend

이 패스는 각 문장의 In/Out 집합을 생성한다. C 언어는 변수 이름을 마음대로 가질 수 있기 때문에 각 문장에 대한 In/Out 집합은 꽤 복잡할 수 있다. In/Out 집합의 요소들은 변수들(심볼 테이블을 가리키는 포인터들)과 변수들이 될 수 있는 조건의 리스트들이다. In/Out 집합들이 연산될 수 없을 때마다 일반적인 (universal) In/Out 집합으로 간주된다. 이것은 이 문장이 어떤 메모리의 위치에도 읽고, 쓸 수 있다는 것을 의미한다. 에러 메시지와 출력은 이 패스에 인수로써 정해진 파일이름으로 저장된다.

이 패스의 소스 파일이 존재하는 경로는 src/misc/control/interface.c이며, 초기 호출함수는 call_depend()이다.

3.7. flow

이 패스는 프로그램의 flow graph를 생성한다. 만약 -d 옵션을 통해 디버그 플래그가 양수 값을 가지면, flow graph와 그에 상응하는 연결하는 배열과 flow graph의 dominator tree에 관한 정보가 출력될 것이다.

3.8. constant

이 패스는 프로그램의 전역에 상수를 전파한다. 이것은 constant folding과 constant propagation의 작업을 한다. 식에 존재하는 변수들의 값이 상수가 아니더라도 결국은 상수로 되면 상징적으로 전파가 이루어진다.

src/transformations/constant_propagation/constant.c이며, 초기 호출 함수는 constant()이다.

3.9. induction

이 패스는 근본적으로 Parafrase-2의 심볼 분석을 한다. 이 패스는 프로그램 내에서 루프 첨자 변수와 루프 invariant라는 용어로써 표현 가능한 모든 induction 식을 인식한다.

이 패스의 소스 파일이 존재하는 경로는 src/transformations/induction_elimination이며, 초기 호출함수는 induction.c안에 존재하는 induction()이다.

3.10. dotodoall

이 패스는 builddep에서 생성된 정보를 사용한다. 프로그램 내에 있는 모든 반복문을 검사하여, 반복문들이 병렬화될 수 있는지 없는지를 검사한다. 만약 하나의 반복문이 병렬화될 수 없다면, 병렬화를 막는 모든 종속관계들이 지적된다. 출력은 정해진 출력파일에 저장된다.

이 패스의 소스 파일이 존재하는 경로는 /src/misc/control이며, 초기 호출함수는 interface.c 안에 존재하는 call_dotodoall()이다.

3.11. codegen

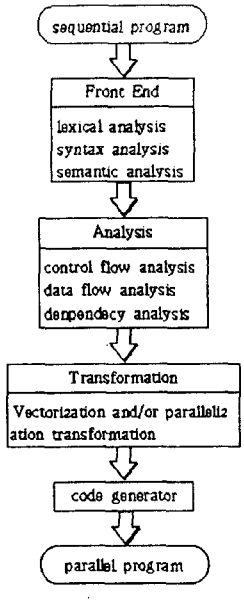
이 패스는 병렬 코드를 화일 또는 표준 출력장치로 출력하는 패스이다. syntax tree에 존재하는 모든 것을 파일 또는 표준 출력장치로 출력한다. 이때, 파일로 출력하고자 하면 옵션을 파일이름을 주면 된다. 즉, 실행시킬 패스들이 저장되어 있는 화일에 codegen parallel.c 라는 명령으로 parallel.c라는 옵션을 붙이면 parallel.c의 화일에 출력이 된다. 만약, 옵션을 생략할 경우에는 표준 출력장치로 출력을 하게된다.

코드가 Fortran인 경우는 출력할 때 라인 번호를 출력할 수가 있는데, -l 옵션을 붙이면 된다. 그러면 출력할 때, 73-80칼럼에 라인 번호가 추가된다.

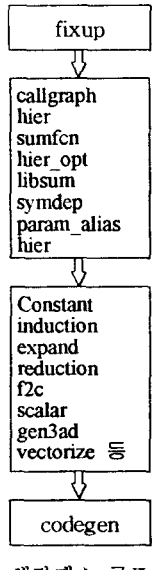
이 패스가 수행시 처음 호출하는 파일은 src/misc/control/interface.c이며, 초기 호출함수는 call_codegen()이다.

4. 간단한 병렬 컴파일러의 설계 및 구현

3절에서 분석한 방법에 의하면 병렬컴파일러 구조는 다음 그림과 같다.



병렬 컴파일러 구조



해당패스 구조

이중에서 다른 pass들은 그냥 사용하고 병렬성을 이용하여 SUIF에서 사용하는 중첩된 루프에서 병렬코드인 DOALL문장을 중첩된 루프의 바깥으로 끌어내기 위하여 CODEGEN pass에서 DOALL문장을 중첩된 루프의 바깥으로 끌어내기 위한 코드는 다음과 같다.

```
int call_codegen(p2_module, s)
module_t *p2_module;
char *s;
{
module_t *modls;
        int         lineflag;
        char *getnext();
int flag;
char ch;
char *t;
.....
DB0(5,"BEGIN Loop InterchangeWn");
traverse_modules(P2_module,FALSE,NULL,test_all_for_swap);
DB0(5,"END Loop InterchangeWn");
.....
return 0;
}
```

5. 결 론

본 논문에서는 병렬화 컴파일러인 Parafrase-2에 대한 원시코드를 분석하였으며 이를 바탕으로 하여 병렬화 컴파일러의 구조를 살펴보았고 간단한 병렬화 컴파일러도 구현하였다.

이러한 병렬화 컴파일러는 향후 작업의 대용량, 고속처리의 필요성을 볼 때 병렬컴퓨터의 일반화와 더불어 컴파일러 개발의 중심 기술이 될 것으로 판단된다.

참 고 문 헌

[1] Barbara Chapman and Hans Zima, Supercompilers for Parallel and Vector Computers, Addison-Wesley, 1990

[2] Bil Blume et. al., "Polaris : The Next Generation in Parallelizing Compilers," Proceedings of the Workshop on Languages and Compilers for Parallel Computing, pp. 10.1-10.18, 1994.

[3] C. D. Polychronopoulos et al., "The

Structure of Parafrase-2 : An Advanced Parallelizing Compiler for C and Fortran," In Proceedings of '89 International Conference on Parallel Processing, pp.II-39~48, IEEE Computer Society.

[4] Frances Allen, Michael Burke, Philippe Charles, Rob Crypton and Jeanne Ferrante, "Overview of the Ptran Analysis System." Journal of Parallel and Distributed Computing, Vol. 5, No. 5, pp. 617-640, 1988.

[5]Hans P. Zima and Barbara Mary Chapman, "Compiling for distributed-memory systems," Proceedings of the IEEE, Vol. 81, No. 2, pp. 264-286, 1993

[6]Message Passing Interface Forum, MPI : A Message Passing Interface Standrd, final report version 1.0 edition, 1995.

[7] Mohammad R. Haghighat and Constantine D. Polycronopoulos, "Symbolic Program Analysis and Optimization for Parallelizing compilers," CSRD Report 1237, UIUC, 1992.

[8] Robert H. Kuhn, Bruce Leasure, and Sanjiv M. Shah, "The KAP Parallelizer for DEC Fortran and DEC C Programs," Digital Technical Journal, Vol. 6, No. 3, pp. 57-70, 1994.

[9] Robert P. Wilson, Robert S. French, christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W. K. Tjiang, Shih-Wei Liao, C.W. Tseng, Mary W. Hall, Monica S. Lam, and John L. Hennessy, "SUIF : An infrastructure for research on parallelizing and optimizing compilers," ACM SIGPLAN Notices, Vol. 29, No. 12, pp. 31-37, 1994.

[10] Jong-Deok Choi, Michael Burke and Paul Carini, "Efficient Flow-Sensitive Interprocedural Computation of Pointer Induced Analysis and Side Effects." In Proc. of ACM Symposium on Principles of Programming Languages, pp. 232-246, 1993.

[11] 안준선, 최광훈, 김성훈, 한태숙, 최광무, "병렬 컴파일러의 소개", 1994,6. 정보과학회지 12권 제5호

[12] 병렬화 트랜스레이터 개발에 관한 연구, 최종 연구보고서(연구수행기관 : 한국과학기술원 전산학과), 한국전자통신연구소, 1996.

[13] 병렬화 트렌스레이터와 노드간 연결망의 검증 및 성능 평가 시스템 개발, 최종 연구 보고서(연구수행기관 : 한국과학기술원 전산학과), 한국전자통신연구소, 1995.

[14] 최종덕, "프로그램 병렬화를 위한 고급 컴파일러 기술", '91 컴퓨터 과학 하계 세미나 자료집, 인공지능연구센터, 한국과학기술원, 1991.

[15] Polaris Home Page, http://csrd.uiuc.edu/polaris/polaris.html

[16] Parafrase2 Home Page, http://csrd.uiuc.edu//parafrase2

[17] SUIF Home Page, http://suif.stanford.edu