병렬성 및 지역성 증진을 위한 컴파일러 최적화

김 진 미[†] · 변 석 우[†] · 표 창 우[††] · 이 만 호[†††]

요 약

본 논문에서는 순차 언어로 작성된 프로그램을 '병렬화'와 '지역성 향상'을 목적으로 변형시키는 최적화 기법에 대해서 논의한다. 의존성과 지역성을 고려하여 순차 프로그램의 루프 구조를 분석하고, 루프 분산과 루프 병합 기법을 적용하여 프로그램을 변형시킨다. 이 변형된 프로그램은 쉽게 '굵은 단위'의 병렬성과 지역성이 향상된 형태의 쓰레드 프로그램으로 표현될 수 있다. 따라서 이 변형 기법은 최적화/자동병렬화 컴파일러 구현에 유용하게 응용될 수 있다. 4개의 SPARC 프로세서를 장착한 Solaris 시스템에서 이 기법을 SPEC95 프로그램에 적용하여 시험한 결과 순차프로그램과는 20~62%, 기존의 SUIF 병렬화 컴파일러와는 3~12% 정도의 수행 시간이 개선되는 효과를 얻게 되었다.

Compiler Optimization for Parallelism and Locality Improvement

Jin-Mee Kim[†] · Sug-Woo Byun[†] · Chang-Woo Pyo[††] · Mann-Ho Lee[†††]

ABSTRACT

In this paper, we study on the transformation technique of sequential programs for the purpose of 'exploiting parallelism' and 'improving locality'. Based on the analysis of loop procedures of sequential programs with the factor of dependency and locality, two transformation techniques of loop distribution and loop fusion are applied to them. Transformed programs can be easily expressed as a parallel program with thread notation, having coarse-grain parallelism and improved locality. This means that those transformations can be useful tools for optimizing and automatic-parallelizing compiler construction. Application of those techniques to SPEC95 on a Solaris machine with four SPARC processors shows an improvement of execution time.

1. 서 론

병렬 프로그래밍 환경을 지원하기 위하여 이루어지고 있는 연구 방법으로는 기존의 순차 언어로 작성된 프로그램을 자동으로 병렬화 하는 방법, 기존의 순차 언어에 병렬 구문을 추가하여 병렬성 표현을 지원하는 방법 그리고 새로운 병렬 언어를 개발하는 방법이 있다. 그 중에서 자동 병렬화 방법은 기존의 순차 프로그램을 바꾸지 않으므로 기존에 사용하던 소프트웨어를 새로 개발하는 비용을 절약할 수 있고, 순차 프로그램에 익숙해져 있던 프로그래머가 시스템의 병렬 기능을 효과적으로 이용할 수 있는 이점이 있다.

최근에 연구되어 널리 사용하고 있는 병렬성과 지역성 증진을 위한 자동 병렬화 프로그램 변환은 프로그램의 수행 시간에 가장 많은 영향을 주는 루프 단위의 의존 및 지역 관계 분석에 기초하고 있다[1,10]. 루프 내에서의 명령어와 데이터의 병렬성 및 지역성을 최대한 이용하여야 고성능 프로세서의 성능을 좌우하는 파이프라인, 다량의 레지스터, 캐시 메모리가 효과

† 정 회 원 : 한국전자통신연구원
†† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
††† 종신회원 : 충남대학교 컴퓨터과학과 교수
논문접수 : 1998년 7월 15일, 심사완료 : 1998년 11월 5일

적으로 사용될 수 있다[2,4].

본 논문에서는 자동 병렬화 번역기의 구현으로 병렬화를 위한 코드 변환을 수행한 후 쓰레드를 이용한 최적화된 코드 생성을 통하여 프로그램의 효율성을 높이는 방안을 제안하였으며, 구현된 번역기의 성능에 관한 내용을 기술하였다. 이는 병렬 컴퓨터에서의 프로그램 개발 생산성을 높일 수 있고 컴파일러의 관련 기술 발전에 도움을 줄 수 있다. 개발된 병렬화 번역기는 기본 컴파일러로 선정한 SUIF 컴파일러를 사용하여 순차 언어인 FORTRAN 및 C 프로그램을 입력으로 받아 병렬성 및 지역성 분석을 수행하고, 분석 정보를 최대한 이용하여 구현한 루프 분산 및 루프 병합의 프로그램 변환을 수행한 후, 쓰레드 실행시간 라이브러리를 이용한 병렬성을 명시한 C 프로그램을 출력한다. 루프 분산의 목적은 루프 몸체에 나타난 문장들을 분리하여 분리된 문장 간의 의존성을 제거하고 지역성 정보를 기반으로 하여 새로운 병렬 루프를 만들고자 하는 것이며, 루프 병합은 병렬 루프의 계산량을 증가 시키고자 하는 것이다.

본 논문의 2장에서는 병렬화 번역기를 구현하기 위해 필요한 프로그램 분석 방법 및 프로그램 변환의 내용에 대해 기술하고, 3장에서는 병렬화 번역기의 설계 및 구현에 대한 내용을 설명하였다. 4장에서는 병렬화 번역기의 각 단계에 대한 실험의 결과를 보여주고 분석하며, 마지막 5장에서는 병렬화 번역기의 결과 및 향후 계획에 대해서 다루고 있다.

2. 프로그램 분석

프로그램 분석 단계에서는 컴파일러의 전단부 과정을 수행하고 중간 코드를 생성 하게 되면 중간 코드를 입력으로 자료 의존성, 제어 의존성 및 지역성 등을 분석하여 그 정보를 추가한 중간 코드를 생성 한다. 본 병렬화 번역기에서는 스탠포드 대학에서 개발한 SUIF[6]에서 사용된 의존성 및 지역성 정보를 이용하여 루프 변환을 적용하였다.

2.1 의존성 분석

병렬화 번역기는 의존성 분석을 통하여 프로그램의 문장 사이에 가해지는 실행 순서 상의 제약 조건을 추출한다. 의존성은 루프 회전 사이에 존재 할 수 있고, 루프 회전 내에 존재할 수 있으며 이에 관계하는 변수의 읽기 및 쓰기 정보에 따라 입력 의존성, 출력 의존성, 흐름 의존성, 반 흐름 의존성으로 나뉘어 진다. 컴파일러는 프로그램 내에 표현된 자료 의존성 관계들을 분석하여 문장 사이에 가해지는 순서, 두개 이상의 연산에 가해지는 순서, 혹은 루프의 반복 사이에 가해지는 순서를 재배열 함으로서 효율성을 증진시킬 수가 있다. 의존성 분석은 실행 순서상의 제약 조건을 추출하므로 재배열된 프로그램이 수행되는 결과는 변환 전의 프로그램이 수행되는 결과와 같아야 한다. 따라서 자료 의존성을 정확히 파악하는 일은 매우 중요한 일이다.

2.2 지역성 분석

대부분의 병렬성을 위한 개선 변환이 지역성 개선을 위해서 사용 될 수 있다. 메모리 계층 구조 상에서 컴퓨터 성능을 향상 시키기 위해 컴파일러에 의한 프로그램 변환을 통하여 지역성을 향상시킴으로써 시스템의 속도 향상에 기여하는 캐시의 성능을 높일 수가 있다.

프로그램이 실행될 때에는 명령어와 데이터 접근이 넓은 범위에 걸쳐 분포되어 발생하지 않으며, 특정 부분에 집중되는데 이를 지역성의 원칙이라고 한다[3]. 지역성이란 접근되는 메모리 요소들의 시간 또는 공간적인 분포를 말하는 것으로 시간 지역성과 공간 지역성으로 나눌 수 있다. 시간 지역성이란, 캐시에 저장된 어떤 변수가 다시 재참조 되는 현상을 말하고, 공간 지역성이란 같은 캐시 라인이 다시 참조되는 현상을 말한다. 따라서 시간 지역성은 공간 지역성의 특별한 경우라고 생각 할 수 있다. 지역성 측정을 위한 척도로서 재사용 거리를 재사용 벡터 공간(reuse vector space) 등을 이용하여 적용한다.

재사용은 루프에서 서로 다른 회전이 동일한 메모리 위치를 참조함을 의미한다. 루프의 지역성을 증진시키기 위하여 재사용이 많이 발생하도록 루프를 변형하여야 하며 이를 위하여 루프에 존재하는 재사용 형태를 알아야 한다. 재사용에 대한 정보는 벡터 형태로 표현되며 이를 재사용 벡터라고 부른다. 특히 SUIF 에서는 같은 캐시 라인을 사용하는 그룹 공간 지역성 개선을 위하여 균일 생성 집합(uniformly generated set)에 기초한 공간 지역성을 검사하는 방법을 취한다. 즉, 문장 사이의 지역성 정보로 균일 생성 집합이 이용되어 하나의 균일 생성 집합에 포함된 배열 변수들은 유

사한 첨자식을 가지고 있고 상수 거리 만큼 떨어져 있게 된다. 루프에서 하나의 배열에 대한 참조가 {A[i], A[i+1], A[i-1]}과 같을 경우 배열의 인덱스 함수는 단지 상수항 만이 다르며 이러한 참조를 균일 생성 참조(uniformly generated reference) 라고 한다[8, 9]. 이 개념들은 M. Lam의 균일 생성 집합과 Gannon 의 참조창(reference window)에서 사용되었다.

<정의> 중첩 루프의 회전 공간이 n 차원이고, 배열 A가 d차원이라 하자. 인덱스 함수 $\vec{f}(\vec{i})$와 $\vec{g}(\vec{i})$가 다음의 조건을 만족 할 때, 두 배열 참조 $A[\vec{f}(\vec{i})]$와 $A[\vec{g}(\vec{i})]$는 균일 생성 참조라고 한다.

조건 : $\vec{f}(\vec{i})$와 $\vec{g}(\vec{i})$는 각각 $\vec{f}(\vec{i}) = \overrightarrow{Hi} + \overrightarrow{C_f}$와 $\vec{g}(\vec{i}) = \overrightarrow{Hi} + \overrightarrow{C_g}$이다. 여기서, $H: Z^n \rightarrow Z^d$는 $d \times n$의 행렬로 표현된 선형 변환(linear transformation)이고 $\overrightarrow{C_f}$, $\overrightarrow{C_g}$는 Z^d에 속하는 상수(constant) 벡터이다.

두 문장 사이에 지역성이 있다는 것은 문장 구성 변수들 중 같은 균일 생성 집합에 포함될 수 있는 변수가 존재한다는 것을 말한다.

2. 병렬화 번역기의 설계 및 구현

본 병렬화 컴파일러의 개발은 SUIF 컴파일러 시스템을 기반으로 이루어졌다. SUIF을 대상으로 한 이유는 중간 언어를 조작하기에 편리한 많은 라이브러리들을 제공하고, 중간 언어에 대한 접근 함수 뿐만 아니라 다양한 변환 함수를 제공함으로서 여러 가지 최적화 기술 및 병렬화 기술을 구현 할 수 있으며, SUIF 컴파일러 시스템을 사용하는 그룹들과의 연구 교류를 쉽게 할 수 있도록 하기 때문이다. 본 장에서는 SUIF 컴파일러에 대한 설명과, 병렬성 및 지역성 증진을 위해 선정된 루프 분산 및 병합의 구현 내용을 기술한다. 그리고 병렬화 번역기 시스템의 구조 및 실행 환경에 대하여 설명한다.

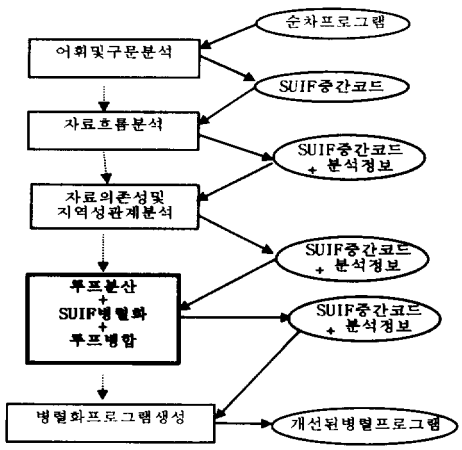
2.1 SUIF 컴파일러 시스템[6]

SUIF는 컴파일러의 단계를 중간 코드를 기반으로 한 모듈들로 구분하여, 새로운 모듈을 추가하여 확장하거나 시험하기에 적합하도록 만들어진 연구용 컴파일러 시스템이라고 할 수 있다. 이 중간 표현 언어에 대해서 많은 라이브러리가 구축되어 있고 SUIF 시스템은 컴파일러 기반 구조를 제공함으로서 임의의 최적화 기술이나 병렬화 기술들을 구현하여 그 유용성을 쉽게 시험할 수 있도록 설계되었다. 이러한 점으로 인해 많은 컴파일러 관련 개발자들이 SUIF의 중간 코드를 이용하여 연구를 진행하고 있다. SUIF 컴파일러 시스템은 중간 언어와 중간 언어에 관한 다수의 라이브러리, C 언어에 대한 전단부 과정, 그리고 SUIF 언어를 C 언어로 변환하는 과정들을 포함하고 있다.

본 병렬화 번역기 시스템에서는 SUIF에 구현되어 있는 단위 모듈 변환 및 루프 타일링[7]을 사용하며, 병렬성 및 지역성 증진을 위해 루프 분산 및 병합을 구현하여 적용하였다. SUIF에서 사용된 단위 모듈 변환이란 단위 모듈 행렬로 표시되는 변환으로 루프 기울임, 루프 반전, 루프 교환의 조합을 말하며 본래의 루프를 완전 변환 가능(fully permutable) 한 루프로 변경한다. 완전 변환 가능한 조건은 루프 타일링 및 루프 분산 등의 선결 조건이 될 수 있다.

2.2 병렬화 번역기 시스템

본 연구에서 개발한 병렬화 번역기는 FORTRAN이나 C 의 순차 프로그램을 컴파일하여 실행 시간 라이브러리를 포함하는 개선된 C 병렬 프로그램을 생성한다. 다음 (그림 1)은 병렬화 번역기의 개략 구조이다. 전단부에 해당하는 어휘 및 구문 분석, 자료 분석부에 해당하는 자료 흐름 분석 및 자료 의존성 관계 분석은 기존의 SUIF에서 제공하는 기능을 사용하였다



(그림 1) 병렬화 번역기 구성도
(Fig. 1) parallelizing translator structure

프로그램 변환에 해당하는 병렬화 프로그램 생성 부분은 지역성 및 병렬성을 고려한 루프 분산 및 루프 병합 모듈을 구현하여 SUIF의 병렬화 부분과 연결되어 사용하도록 하였다. 병렬화 코드 생성은 SUIF의 실행 시간 라이브러리를 사용하여 솔라리스 쓰레드를 이용할 수 있게 수정하였다. 병렬화 번역기는 FORTRAN 프로그램이나 C 프로그램을 입력으로 하여 쓰레드를 사용하는 실행 가능한 코드로 변환한다. 본 연구에서 개발한 병렬화 번역기는 기존의 SUIF 모듈에서 루프 분산이나 루프 병합 모듈을 첨가함으로써 시스템의 병렬성을 높일 수가 있다.

루프 분산은 병렬 수행 시스템에서 루프의 병렬성을 얻기 위해 사용하거나, 루프 교환 등의 변환을 수행하기 위한 선행 변환에 이용되고 있다. 또한 루프 분산은 데이터 지역성 향상을 위하여도 응용이 가능하므로 의존성 분석 및 지역성 정보를 이용한 루프 분산은 병렬성 및 지역성을 증진시켜 종합적인 성능 향상에 효과를 얻을 수 있다. 개발된 루프 분산은 변수 재사용 정보를 도입하여 재사용 관계에 있는 문장은 같은 루프 내포체에 배치하도록 하였다. 이는 병렬 루프를 생성할 때 재사용 관계에 있는 문장들을 같은 병렬 루프 내에 두게 함으로써 병렬 루프의 계산량을 증가시키고 지역성이 개선되므로 전체 성능 개선에 효과를 가져올 수 있기 때문이다. 루프 병합의 경우에는 서로 다른 반복문을 하나로 합쳐 하나의 반복 내에 수행되는 작업을 크게 하여 쓰레드를 지원하는 공유 메모리 병렬 시스템의 성능 향상을 가져올 수 있으며 같은 자료를 접근하는 두 개의 루프를 병합하는 경우 시간적 지역성을 높이는 효과가 있으므로 캐시 메모리나 가상 메모리의 성능을 높일 수 있다.

(1) 루프 분산

루프 분산은 단일 루프를 여러 루프로 나누는 것을 말한다. 루프 분산의 목적은 루프 몸체에 나타난 문장들을 분리하여 분리된 문장 간의 의존성을 제거함으로써 새로운 병렬 루프를 만들고자 하는 것이다. 이러한 루프 분산은 반복간의 의존성을 서로 다른 반복문과 반복문 사이의 의존성으로 바꾸어 병렬 수행이 가능한 반복문으로 만들 수 있다. 또한 분할된 루프는 몸체를 작게 하여 캐시 지역성을 높일 수 있고 접근하는 배열 수를 감소시켜 메모리 사용을 줄일 수 있다. 그리고 루프 교환과 같은 다른 변형 기법 전에 이용할 수도 있다.

루프 분산 구현을 위해서는 문장들 사이의 의존성과 지역성 정보는 그래프를 이용하여 나타낸다[12]. 그래프의 노드는 루프내의 문장에 대응하고, 에지는 의존성 또는 지역성을 나타낸다. 문장 사이의 의존성 여부는 문장을 구성하는 변수들(배열)의 의존성 백터를 조사하여 구한다. 스칼라 변수에 관해서는 변수의 심벌 정보를 이용하여 같은 이름의 스칼라 변수 들이 있는 문장은 하나의 루프 내에 유지되도록 하였다. 의존성 그래프에서 사이클이 존재하지 않으면 각각의 노드를 분리하여 루프를 분산할 수 있다. 이 경우 각각의 노드는 문장이나 내포된 루프가 된다. 분리된 루프들은 의존성 그래프에서 위상 정렬(topological sort) 순으로 순서를 정한다. 루프 분산은 의존성 그래프에서 SCC(Strongly Connected Component)를 찾아 단일 루프에 각 SCC 별로 루프의 몸체를 구성하게 된다. 의존성 그래프에서 노드가 단일 SCC이면 루프 분산이 발생하지 않는다. 컴파일러는 분산 전에 스칼라 확장을 행하므로 스칼라에 관한 반 의존성과 출력 의존성과의 관계를 고려하지 않는다. 루프 분산 방법의 구현 절차는 다음과 같다.

① 루프 내의 각 문장에 대해서 의존성 그래프의 노드를 하나씩 만들고, 문장들 간의 의존 및 지역 관계를 의존성 그래프의 에지로 표현한다.
② 의존성 그래프에서 SCC를 찾아서 그 구성 요소(component)를 노드로 하는 축약된 의존성 그래프를 만든다. 축약된 그래프에서는 사이클이 존재하지 않는다.
③ 축약된 의존성 그래프에서 위상 정렬(topological sort)을 하여 노드간의 에지를 기준으로 전방향 의존성을 찾고, 노드에 해당하는 문장들로 이루어진 새로운 루프를 만든다.

(2) 루프 병합

루프 병합은 이웃하는 두개의 루프가 같은 반복 수행 공간을 가지고 있을 때, 서로 다른 반복문을 하나로 합쳐 하나의 반복 내에 수행되는 작업을 크게 하는 것을 말한다. 루프 병합은 원래 루프 자체의 비교 및 분기에 따르는 오버헤드를 줄이기 위하여 제안되었으나 그 이외에도 같은 자료를 접근하는 두 개의 루프를 병합하는 경우 시간적 지역성을 높이는 효과가 있으므

로 캐시 메모리나 가상 메모리의 성능을 높일 수 있다. 그리고 루프 본체의 크기를 크게 함으로 공통 수식 제거, 명령어 스케줄링과 같은 스칼라 최적화의 범위를 넓게 해준다. 루프 병합은 병합 전의 모든 의존성이 병합 후에도 바뀌지 않았을 경우에 적용 가능하다. 이는 변환된 루프가 사전적으로 음의 벡터를 유발하지 않으면 적법하다.

본 연구에서의 루프 병합의 목적은 병렬 루프의 계산량을 증가 시키는 것이다. 루프 타일링 및 루프 분산에 의해 새로운 병렬 루프를 만들고 병렬 루프인 doall 루프를 대상으로 루프 병합을 실행한다.

do 루프의 경우에는 순차 병합 방해 의존성(serial-fusion preventing dependence)이 없을 때 루프 병합이 가능하지만 doall 루프의 경우에는 병렬 병합 방해 의존성(parallel-fusion preventing dependence)이 없어야 루프 병합이 가능하게 된다. 따라서 do 루프는 순차적으로 루프가 수행되므로 렉시코그래피컬한 순서에 반하는 의존성만 없으면 되지만, doall 루프는 루프의 모든 인스턴스가 동시에 수행 될 수 있으므로 어떤 의존성도 없어야 루프 병합이 가능하다. 순차 병합 방해 의존성 및 병렬 병합 방해 의존성의 정의[11]는 다음과 같다.

<정의> 순차 병합 방해 의존성

어떤 의존성 S δ_{α} S´가 순차 병합 방해 의존성이라 함은 어떤 두 반복 벡터에 대하여 i∈[S]와 i∈[S´]에 대해 그 수행 인스턴스 S(i)와 S(i´)가 의존성을 가지는 반복 벡터가 $i_c > i'_c$라는 조건하에 존재하여야 한다.

<정의> 병렬 병합 방해 의존성

어떤 의존성 S δ_{α} S´가 병렬 병합 방해 의존성이라 함은 어떤 두 반복 벡터에 대하여 i∈[S]와 i∈[S´]에 대해 그 수행 인스턴스 S(i)와 S(i´)가 의존성을 가지는 반복 벡터가 $i_c \neq i'_c$라는 조건하에 존재하여야 한다.

두 개의 doall 루프를 받아서 병렬 병합을 시도하는 루프 병합 방법의 구현 절차는 다음과 같다.

① 병렬화 가능한 루프이면서 하한 및 상한의 경계 값과 스텝이 동일한 식으로 이루어진 인접한 루프를 찾는다.

② 두 병렬 루프의 몸체를 합한 새로운 루프를 만들어 새로운 루프가 병렬 가능한 루프인지를 의존성 검사를 통해 확인한다.

③ 병렬화가 가능하지 않은 루프라면 첫번째 병렬 루프를 제외하고, 인접한 두 번째와 세 번째 루프에 대해 동일한 방법을 적용한다. 병렬화가 가능한 경우에는 새로 만든 병렬 루프와 세 번째 병렬 루프에 대해서 동일한 방법을 적용한다.

④ 루프 병합을 적용할 두 병렬 루프의 인덱스가 다른 경우에는 루프가 수행된 후에 수정된 인덱스들의 값을 하한 경계 값으로 설정하도록 코드를 생성한다.

(3) 병렬 코드 생성

병렬 처리의 효과는 프로그램에서 병렬 처리를 위하여 사용하는 단위에 따라서 그 성능이 크게 좌우된다. 기존의 유닉스에서 사용되는 프로세스는 단일 주소 공간을 사용하는 하나의 순차적인 수행 흐름으로 프로그램 실행에 필요한 코드 메모리, 정적 메모리, 힙 메모리, 스택 메모리 및 프로세스 관리를 위한 정보를 가지고 있으며 다중처리기에서 범용의 병렬 프로그램을 작성하기에는 매우 비효율적이다. 이러한 단점을 보완할 수 있는 개념이 쓰레드이며 이는 기존의 전통적인 유닉스의 프로세스 개념 중에서 수행의 흐름만을 가지며, 프로세스의 코드 메모리, 정적 메모리, 힙 메모리 등은 다른 쓰레드들과 공유하고 서로 다른 수행점 관리를 위하여 스택 및 쓰레드 관리용 정보만을 단독으로 가지고 있다. 따라서 쓰레드의 생성, 수행 및 소멸 등에서 병렬 수행을 제어하는데 드는 비용이 기존의 프로세스에 비하여 적으므로 효과적인 병렬 처리를 할 수 있게 된다.

본 병렬화 컴파일러의 코드 생성은 쓰레드 라이브러리를 사용하는 실행시간 라이브러리를 지원하며 번역된 코드는 실행시간 라이브러리가 사용된 C 프로그램이 된다. 본 구현에서는 기존의 SUIF에서 지원하는 실행시간 라이브러리 모듈에 솔라리스 쓰레드를 지원하는 모듈을 첨가하였다.

3. 실험 결과

본 병렬화 번역기를 위해 구현된 루프 분산 및 루프 병합을 기존의 SUIF 컴파일러의 단계에 포함시켜 실험하였다. 실험은 다음 항목을 대상으로 각각을 SPEC95

벤치마크 프로그램에 적용시켜 수행 시간을 측정한 후 성능 향상을 비교하였다.

- 병렬화 하지 않은 SUIF
- 기존 SUIF 병렬화
- 지역성을 고려하지 않은 루프 분산 후 병렬화
- 지역성을 고려한 루프 분산 후 병렬화
- 루프 병합 병렬화
- 지역성을 고려하지 않은 루프 분산 후 루프 병합 병렬화
- 지역성을 고려한 루프 분산 후 루프 병합 병렬화

본 실험은 SPEC95 실수형 벤치마크 프로그램들에 대해서 루프 변환의 개수와 실행 시간을 측정하였다. 각 벤치마크 프로그램들은 포트란 프로그램 들이며 SUIF 번역기 시스템을 통하여 C 프로그램으로 변환된다. 변환된 C 프로그램들은 4개 프로세서를 장착하고 운영 체제로써 Solaris 2.4를 사용하며 GNU[5]의 gcc 및 g++ 컴파일러와 솔라리스 쓰레드를 사용한 환경에서 측정되었다.

구현된 루프 분산 및 병합에 대하여 변형된 루프의 수는 <표 1>과 같다.

<표 1>에서 tomcatv의 경우 SUIF 과정 중에서 루프 분산 모듈만을 첨가하였을 경우 3개의 루프에 대해 루프 분산이 발생하였으며, 루프 분산에서도 지역성을 고려하였을 경우 1개의 루틴에 대하여 루프 분산이 발생하였다. 그리고 기존 SUIF 코드에서 루프 병합의 조건을 만족하는 루프는 발생하지 않았고, 루프 분산 후 루프 병합을 수행하는 루프는 3개임을 알 수 있다. 여기서 [(2:3)]의 의미는 두 루프가 연속적으로 병렬 병합의 조건을 만족하여 루프 병합을 가능하게 하는 루프의 수가 3개임을 나타낸다. 예로 [(6:4)]의 의미는 6개의 루프가 연속적으로 병렬 병합의 조건을 만족하여 루프 병합을 가능하게 하는 루프의 수가 4개임을 뜻한다. 다른 시험 프로그램의 루프 변환 수도 동일한 의미로 해석할 수 있다.

다음 <표 2>는 실행 시간을 측정하여 나타낸 표이다. SPEC95 벤치마크의 경우 SUIF의 병렬화 과정에서 제대로 컴파일이 되지 않는 프로그램들이 다수 있었다. 현재는 제대로 컴파일이 되어 수행이 가능한 프로그램들만 한정하여 실행 시간을 측정하였다. 나머지 프로그램들에 대해서는 추후 수정 후 측정할 계획이다. <표 2>에서 실행 시간을 비교하여 보면 전반적으로 루프 분산 후 루프 병합을 수행 한 경우에 성능이 우수함을 알 수 있다.

〈표 2〉 프로그램 실행시간 측정
〈Table 2〉 program execution time

단위 : sec

	①	②	③	④	⑤	⑥	⑦
tomcatv	3514	2894	2896	2875	2869	***2830***	2865
swim	1694	650	759	634	663	802	***627***
hydro2d	3017	2149	2157	2155	2154	2153	***2142***
mgrid	7749	6008	6049	6007	6009	***5316***	6001
turb3d	7004	5802	5843	5842	5843	***5621***	5835

① 병렬화 하지 않은 SUIF
② 기존 SUIF 병렬화
③ 루프 분산 후 병렬화
④ 지역성을 고려한 루프 분산 후 병렬화
⑤ 루프 병합 병렬화
⑥ 루프 분산 후 루프 병합 병렬화
⑦ 지역성을 고려한 루프 분산 후 루프 병합 병렬화

〈표 1〉 프로그램 루프 변형 수
〈Table 1〉 number of loop transformation

프로그램	루프 분산 수	지역성을 고려한 루프분산 수	루프 병합 수	루프분산 후 루프병합 수	지역성을 고려한 루프분산 후 루프병합 수
tomcatv	3	1	0	3 [(2:3)]	1 [(2:1)]
swim	15	11	0	15 [(2:4)(3:4)(4: 3)(6:4)]	12 [(2:3)(3:5)(4: 2)(6:2)]
su2cor	29	12	0	36 [(2:15)(3:2)(4:15)(5:1)(8:3)]	17 [(2:14)(3:1)(4:1)(8:1)]
hydro2d	24	21	0	24 [(2:10)(3:2)(4:10)(8:2)]	20 [(2:7)(3:1)(4:12)]
mgrid	4	0	0	3 [(2:3)]	0
applu	18	0	0	12 [(2:6)(5:6)]	0
turb3d	10	5	0	3 [(2:2)(4:1)]	0

<표 2>를 바탕으로 기존 순차 프로그램, SUIF에서 사용한 병렬화 프로그램, 본 연구에서 구현한 루프 변형을 적용한 프로그램으로 분류하여 비교하면 (그림 2)와 같다.

	tomca	swim	hydro	mgrid	turb3d
순차프로그램	3514	1694	3017	7749	7004
SUIF병렬화	2894	650	2149	6008	5802
루프변형	2830	627	2142	5316	5621

(그림 2) 실행시간 비교
(Fig. 2) comparison of execution time

대부분의 프로그램이 순차프로그램보다는 병렬화 프로그램에서의 실행 시간 성능이 많이 향상됨을 보였고, 기존 SUIF에서 사용한 병렬화 과정에 본 연구에서 구현한 루프 변형, 즉 루프 분산 후 루프 병합을 적용한 경우 보다 나은 성능 향상을 보였다. mgrid의 경우 <표 1>에서 볼 수 있듯이 지역성을 고려한 루프 변형이 발생하지 않았으므로 지역성을 고려하지 않은 루프 분산 후 루프 변형을 적용 할 경우 가장 좋은 성능 향상이 있었다. turb3d의 경우에는 지역성을 고려한 루프 분산 후에 루프 병합 루프가 발견되지 않았으며, 일반 루프 분산 후 루프 병합 시에 성능이 우수하였다.

실험 결과에서 알 수 있듯이 SUIF에서 기본적으로 사용된 병렬화 결과와 비교할 때 루프 분산만을 적용하였을 경우에는 다수의 병렬 루프를 생성 할 수는 있으나 반복내에서 수행되는 작업을 적게 하므로 병렬 루프의 계산량은 감소함을 보였으나, 루프 분산에서도 지역성을 고려할 경우에는 성능 향상이 있음을 알 수 있다. 루프 병합만을 적용하였을 경우에도 병렬 루프의 계산량이 증가함을 알 수 있다. 루프 분산 후 루프 병합을 적용할 경우 최적화 된 결과를 얻을 수 있음을 알 수 있다. 실험 결과에서는 순차프로그램과는 20~62%, 기존의 SUIF 병렬화 컴파일러와는 3~12% 정도의 수행 시간이 개선되는 효과를 얻게 되었다.

4. 결 론

본 논문에서는 병렬성 및 지역성을 향상시키기 위한 루프 변형의 방법으로 루프 분산 및 루프 병합을 선정하여 구현한 내용을 기술하였으며, 기존 SUIF에서 구현된 프로그램 변형 방법과 실행 시간 성능을 비교하였다. 루프 분산의 목적은 루프 몸체에 나타난 문장들을 분리하여 분리된 문장 간의 의존성을 제거함으로써 새로운 병렬 루프를 만들고자 하는 것이며, 루프 병합은 병렬 루프의 계산량을 증가 시키고자 하는 것이다. 개발된 루프 분산은 의존성 정보 외에도 지역성 향상을 실험하기 위하여 변수 재사용 정보를 도입하여 재사용 관계에 있는 문장은 같은 루프 내포체에 배치하도록 하였다. 이는 병렬 루프를 생성할 때 재사용 관계에 있는 문장들을 같은 병렬 루프 내에 두게 함으로써 병렬 루프의 계산량을 증가 시키고 지역성이 개선되므로 전체 성능 개선에 효과를 가져올 수 있기 때문이다. 루프 병합의 경우에는 서로 다른 반복문을 하나로 합쳐 하나의 반복 내에 수행되는 작업을 크게 하여 쓰레드를 지원하는 공유 메모리 병렬 시스템의 성능 향상을 가져올 수 있으며 같은 자료를 접근하는 두 개의 루프를 병합하는 경우 시간적 지역성을 높이는 효과가 있으므로 캐시 메모리나 가상 메모리의 성능을 높일 수 있었다.

실험 결과 루프 분산은 다수의 병렬 루프를 생성하였으며, 루프 병합의 경우에는 예상한 바와 같이 병렬 루프의 계산량을 증가 시켜 주었다. 벤치마크 프로그램의 실행 시간을 비교하여 보면 전반적으로 루프 분산 후 루프 병합을 수행 한 경우에 성능이 우수함을 알 수 있다.

성능 향상 실험 결과를 토대로 각 루틴별로 성능 향상을 측정하여 측정된 루틴들 중심으로 성능 향상 및 감소에 대하여 분석할 계획이다.

참 고 문 헌

[1] S. Carr, K. S. McKinley and C. Tseng, "Compiler optimizations for improving data locality," *Six International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

[2] D. Gannon, W, Jalby and K. Gallivan, "Strategies for cache and local memory management by global program transformation," In *Supercomputing : 1st International Conference*, number 297 in lecture Notes in Computer Science, pages 229-254, Athens, Greece, 1987. Berlin : Springer

Verlag.

[3] L. Hennessy and D. A. Patterson, *Computer Architecture : A Quantitative Approach*, pp.635-693, Morgan Kaufmann Publishers. Inc., San francisco, California, 1996.

[4] K. Hwang, *Advanced Computer Architecture : Prallelism, Scalability, Programability*, pp.3-96, McGraw-Hill, Singapore, 1993.

[5] R. M. Stallman, *Using and Porting GNU CC*, Free Software Foundation, Inc., Cambridge, Massachusetts, 1989.

[6] Stanford Compiler Group, The SUIF Library, SUIF Compiler Documentation Set, 1994.

[7] M. E. Wolf, Improving locality and parallelism in nested loops, Ph.D. Dissertation CSL-TR-92-538, Stanford University, Dept. Computer Science, August 1992.

[8] M. E. Wolf and M. S. Lam, "A data locality optimizing algorithm," In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30-44, Toronto, June 1991.

[9] M. E. Wolf and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," *IEEE Transactions in Parallel and Distributed Systems*, 2(4) : 452-471, October 1991.

[10] M. Wolfe, *High Performance Compilers for Parallel Computing*, Addison-Wesley, Redwood City, CA, 1996.

[11] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computere*, Addison-Wesley, 1991.

[12] 김상훈, 표창우, 윤석한, 루프 분산에 의한 데이터 지역성 개선, 한국정보과학회 추계 학술발표 논문집, 제24권 2, pp.359-362, 1997.

김 진 미

e-mail : jkim@computer.etri.re.kr

1988년 부산대학교 계산통계학과 졸업(학사)

1997년~현재 충남대학교 컴퓨터과학과 석사과정 재학중

1988년~현재 한국전자통신연구원 선임연구원

관심분야 : 병렬컴파일러, 컴파일러최적화, 프로그래밍 환경

변 석 우

e-mail : swbyun@etri.re.kr

1980년 숭실대학교 전자계산학과 졸업(학사)

1982년 숭실대학교 대학원 전자계산학과 졸업(석사)

1989년~1994년 영국 University of East Anglia(전산학 박사)

1982년~현재 한국전자통신연구원 책임연구원

관심분야 : 프로그래밍언어 이론 및 응용, 정형 방법론

표 창 우

e-mail : pyo@cs.hongik.ac.kr

1980년 서울대학교 전자공학과 졸업(학사)

1982년 서울대학교 대학원 컴퓨터공학과 졸업(석사)

1989년 University of Illinois at Urbana-Champaign 전산학 박사

1990년~1991년 US Army Construction Engineering Research Laboratory(미국 육군 건설 기술 연구소) 연구원

1991년~홍익대학교 공과대학 조교수

1997년~현재 홍익대학교 공과대학 부교수

관심분야 : (대분류)컴파일러, 프로그램 변환, 마이크로프로세서

(소분류)데이터 지역성 최적화, 레지스터 할당, 내장 프로세서를 위한 코드 발생기

이 만 호

e-mail : mhlee@cs.chungnam.ac.kr

1977년~1980년 국방과학연구소 연구원

1980년~1982년 충남대학교 계산통계학과 전임강사

1982년~1984년 충남대학교 계산통계학과 조교수

1991년 Indiana University PhD 취득

1991년~1994년 충남대학교 전산학과 조교수

1994년~현재 충남대학교 컴퓨터과학과 부교수

관심분야 : 병렬컴파일러, 병렬처리, 전자도서관