비평가인자 함수 프로그램의 스레드 분할 향상을 위한 자료형 분리 집합 분할 알고리즘

양 창 모† · 주 형 석†† · 유 원 희†††

요 약

비평가인자 함수 언어는 비평가인자 여의로 인하여 기존의 von Neumann 형 병렬기계에서 효율적인 수행을 어렵게 하는 미세수준의 동적 스케줄링과 동기화가 필요하다. 비평가인자 함수 프로그램을 번역할 때 순차적으로 수행될 부분을 찾아 이들을 하나의 스케줄링 단위로 병합하는 과정이 중요하다. 이러한 과정을 스레드 분할이라 한다.

본 논문에서는 비평가인자 함수 프로그램을 스레드로 분할하는 자료형 분리집합 분할이라는 스레드 분할 알고리즘을 제안한다. 자료형 분리 집합 분할 알고리즘은 자료형을 비교할 수 없는 입력명과 출력명사이에는 잠재 간접 종속이 존재할 수 없다는 사실을 이용하여 스레드 분할을 수행한다. 이 방법을 사용하면 기존의 스레드 분할 방법에서 실패하는 스레드의 병합이 가능하며, 기존의 분할 알고리즘보다 더 큰 스레드를 생성할 수 있다.

Typed Separation Set Partitioning for Thread Partitioning of Non-strict Functional Programs

Changmo Yang† · Hyungseok Joo†† · Weonhee Yoo†††

ABSTRACT

The semantics of non-strict functional languages require fine-grain dynamic scheduling and synchronization, making an efficient implementation on conventional parallel machine difficult. In compiling these languages, the most important step is to extract the sequentially executable portions of a program and to group them into a scheduling unit. This process is called partitioning.

In this paper, we propose Typed Separation Set Partitioning algorithm for partitioning non-strict functional programs into threads using type information of input names and output names. Any input cannot be indirectly dependent on outputs whose types are incompatible to those of inputs. This algorithm can generate the longer threads than other partitioning algorithms can do.

1. 서 론

비평가인자 함수 언어(non-strict functional lang-uages)는 내재된 병렬성을 사용할 수 있으며, 원소의 값이 정의되지 않은 자료구조를 사용한다던가 인자가 평가되지 않은 함수의 결과를 돌려주는 등의 유연성으로 인하여 프로그래머에게 높은 수준의 표현력을 제공한다. 하지만 비평가인자 함수 프로그램을 기존의 von Neumann 형 병렬 기계에서 수행할 때 비평가인자 의

※ 본 논문의 연구는 정보통신부 1998년도 대학기초연구 지원사업의 지원에 의하여 수행됨.
† 정 회 원 : 청주교육대학교 전임강사
†† 정 회 원 : 유한대학 전자계산과 부교수
††† 종신회원 : 인하대학교 전자계산공학과 교수
논문접수 : 1997년 12월 29일, 심사완료 : 1998년 5월 29일

미(non-strict semantics)로 인하여 미세한 수준의 동적 스케줄링이나 동기화가 필요하다.[2, 12]. 이러한 언어를 병렬기계를 위하여 번역할 때, 프로그램의 지역성을 이용하여 동적 스케줄링과 동기화의 횟수를 줄이도록 프로그램을 순차코드로 분할하는 과정(partitioning)이 중요하다.

비평가인자 함수 프로그램을 병렬기계에서 수행할 때 동적 스케줄링은 두 가지 이유로 발생하게 된다. 첫째, 비평가인자 의미로 인하여 모든 연산의 수행 순서를 번역시간에 결정하는 것이 불가능하다. 함수의 연산이 수행되는 순서는 인자의 값뿐만 아니라 그 함수가 호출되는 동적 문맥에 따라 달라지기 때문이다. 둘째 오랜 지연시간(latency time)을 갖는 처리기간 메시지 전달과 I-구조(I-structure)[3] 접근과 같은 동기화가 필요한 연산으로 인하여 동적 스케줄링이 필요하다. 상용처리기에서의 동적 스케줄링은 비용이 많이 들기 때문에 그래프 감축 기계(graph reduction machine)[12], 데이터플로우 기계(dataflow machine)[2], 그리고 다중스레드 기계(multithreaded machine)[4, 5, 6, 8, 9, 10, 11]등에서 구현되어 왔다.

다중스레드 모델은 von Neumann 모델과 데이터플로우 모델의 장점을 혼합한 모델로 확장형 병렬 기계(scalable parallel machine)를 구성하는데 적당한 실행 모델이다. 다중스레드 모델을 위하여 비평가인자 함수 프로그램을 번역할 때 중요한 것은 순차적으로 수행될 수 있는 명령의 집합을 찾아내어 스레드로 분할하고, 동적 스케줄링은 이들 스레드 사이에서만 발생하도록 하는 것이다. 분할이란 수행순서를 번역시간에 알 수 있어 정적으로 스케줄링할 수 있는 프로그램의 부분을 식별하여 스레드로 모으는 작업이다[14, 15, 16, 17, 18]. 병렬기계에서 비평가인자 함수 언어를 구현할 때는 동기화와 비평가인자 의미로 인하여 스레드의 크기가 제한을 받기 때문에 분할의 최종 목표는 스레드의 크기를 가능한 한 크게 만들어 스레드간의 문맥전환 횟수를 최소화하는 것이다[12, 13, 15]. 스레드의 크기를 크게 함으로써 내부 병렬성은 저하되지만 스레드내의 정적 스케줄링과 스레드에 의하여 생성되어 다른 스레드로 전달되는 값의 수가 감소함에 따라 동적 동기화 비용이 감소하게 된다[9, 14].

비평가인자 함수 프로그램을 순차 스레드로 분할하는 대표적인 방법으로는 종속집합 분할(dependence set partitioning)[8], 요구집합 분할(demand set partitioning)[7, 13], 분리제약 분할(separation constraint partitioning)[14], 분리집합 분할(separation set partitioning)[17, 18] 등이 있다. 후에 Schauser등은 종속집합 분할과 요구집합 분할을 반복적으로 적용하는 반복분할(iterated partitioning)과 반복분할을 함수의 호출 관계로 확장한 방법인 전역분할(global partitioning)을 제안하였다[16].

기존의 스레드 분할 알고리즘들은 잠재 종속이 존재하지 않는 노드들을 하나의 스레드로 병합할 때 입력과 출력의 이름만을 고려하여 스레드로 분할하였기 때문에 스레드 분할의 능력이 제한된다. 본 논문에서는 이러한 기존의 분할방법의 문제점을 해결하기 위하여 입력과 출력의 이름뿐만 아니라 입력과 출력의 자료형을 고려하여 서로 비교할 수 없는 자료형을 갖는 노드들간에는 잠재 종속이 존재하지 않는다고 판단하여 하나의 스레드로 병합하는 자료형 분리집합 분할(Typed Separation Set Partitioning) 알고리즘을 제안한다. 이 알고리즘은 분리집합 분할에 자료형 정보를 추가하여 확장한 것으로 스레드 분할에 자료형을 이용하는 방법은 보고된 적이 없는 것으로 알고 있다. 이 방법을 사용하면 기존의 분할 방법에서 실패하는 스레드의 병합이 가능하며, 기존의 분할 알고리즘보다 더 큰 스레드를 생성할 수 있다.

2. 스레드 분할 제약조건

이 장에서는 다중스레드 모델을 위하여 비평가인자 함수 프로그램을 스레드로 분할할 때 고려하여야 하는 조건에 대하여 설명한다. 먼저 스레드를 정의하면 다음과 같다[14, 15].

1) 어떤 문맥에서도 스레드 안에 있는 명령의 올바른 수행순서를 번역시간에 결정할 수 있어야 한다.

2) 한번 스레드의 첫 명령이 수행되면 나머지 명령들은 번역시간에 결정된 순서대로 중단 없이 수행되어야 한다.

비평가인자 함수 프로그램에서 연산은 주어진 문맥에 따라 실행순서가 달라질 수 있고, 이 스레드들은 실행시간에 동적으로 스케줄링되어야 하므로 첫 번째 조건으로 인하여 여러 개의 스레드가 만들어질 수 있다. 따라서 첫 번째 정의를 만족하도록 스레드를 분할하려면, 동적으로 스케줄링되어야 하는 연산들은 하나의 스레드로 묶으면 안 된다.

동적으로 스케줄링되어야 하는 연산들을 구분하기 위해서는 연산들의 종속관계를 구하여야 한다. 연산들의 종속관계는 번역시간에 파악할 수 있는 상시종속(certain dependence)과 번역시간에 확정할 수 없는 잠재종속(potential dependence)이 있다[15, 16]. 또한 종속관계에 있는 연산들의 위치에 따라 종속관계를 스레드 내에 존재하는 직접종속과 서로 다른 스레드간에 존재하는 간접종속으로 구분한다. 간접종속에는 동기화가 필요한 I-구조의 *I-fetch* 연산이나 함수호출 연산과 요구결과를 사용하는 연산사이에 발생하는 상시 간접종속과 비평가인자 의미로 인하여 특정 문맥에서만 발생하는 잠재 간접종속이 있다.

스레드 정의의 두 번째 요구사항에 따른 분할 조건으로 인하여 원격자료 접근 연산, 함수호출, 루프호출 등 번역시간에 그 실행시간을 예측할 수 없는 연산을 나타내는 노드들과 그 결과를 이용하는 노드를 서로 다른 스레드에 위치시켜야 한다. 이와 같이 분할 처리되는 연산과 그 결과를 사용하는 연산사이에는 상시 간접종속이 존재한다. 따라서 비평가인자 함수 프로그램의 스레드 분할이란 간접종속이 존재하지 않는 노드를 찾아 하나의 스레드로 병합해 나가는 과정이다.

3. 자료형 정보를 이용한 스레드 분할

이 절에서는 기본 블록의 입력과 기본 블록을 구성하는 노드들의 자료형 정보를 고려하여 스레드 분할을 수행하는 자료형 분리집합 분할 알고리즘을 제안한다. 기존의 스레드 분할 알고리즘은 기본 블록의 모든 입력과 출력이 잠재 간접 종속을 갖는다고 가정하여 스레드 분할을 수행하였다. 하지만 기본 블록의 입력과 출력의 자료형을 고려한다면 잠재 간접 종속 관계에 있지 않은 입력과 출력의 쌍이 존재할 수 있으며 이 정보를 이용한다면 스레드의 분할을 향상시킬 수 있다.

3.1 분리집합 분할

자료형 분리집합 분할은 분리집합 분할의 확장이므로 먼저 분리집합 분할에 대하여 간단히 살펴본다. 데이터플로우 그래프에서 간접종속은 그래프의 입력과 출력간에 발생되며, 간접종속을 갖는 입력의 자손노드들과 출력의 조상노드들간에도 간접종속이 존재한다. 이러한 사실을 이용하여 간접종속관계를 갖지 않는 노드들을 판별하는 방법은 출력 값을 얻기 위하여 명시적으로 사용되지 않는 입력의 일부를 사용하는 노드와 그렇지 않는 노드를 구별하는 것이다. 이를 위하여 모든 노드에 대하여 결과를 얻기 위해 사용하는 입력의 집합인 참여집합(participation set)과 결과를 얻기 위하여 명시적으로 사용되지 않는 입력의 집합인 분리집합(separation set)이 필요하다[17, 18]. 분리집합은 동일한 스레드에 존재할 수 없는 노드를 분리하기 위하여 사용된다. 참여집합은 [정의 1]과 같이 정의된다. 참여집합은 종속집합과 거의 동일하며, 차이점은 그래프의 노드뿐만 아니라 출력명에 대해서도 정의된다는 것이다.

[정의 1] 참여집합

출력명 또는 노드 u의 참여집합 $P(u)$는 u가 사용하는 모든 입력명을 원소로 갖는 집합이다.

$$P(u)=\begin{cases}\{u\}, & u\in In\\ \bigcup_{v\in Anc(u)} P(v), & u\in N\bigcup Out\end{cases} \quad (1)$$

여기서 In은 입력명의 집합, Out은 출력명의 집합, N은 노드의 집합이며, $Anc(u)$는 u의 조상노드의 집합이다.

노드간의 간접종속관계를 구하는 방법은 다음과 같다. 노드 u와 v의 참여집합을 각각 $P(u)$와 $P(v)$라 하고 $u \not\rightarrow v$는 u에서 v로 잠재 간접종속이 존재하지 않음을 나타낸다고 할 때,

$$P(u)\subset P(v) \Rightarrow u \not\rightarrow v \quad (2)$$

이다.

식 (1)은 노드 u의 결과를 만드는데 필요한 입력의 집합이 노드 v의 결과를 만드는데 필요한 입력의 집합의 부분집합이라면 u에서 v로의 잠재 간접종속이 존재하지 않을 의미한다. 만일 노드 u와 v가 식 (1)의 전제를 만족하지 않는다면, u의 결과가 $P(v)-P(u)$의 부분집합일 수도 있으며, 반대로 v의 결과가 $P(u)-P(v)$의 부분집합일 수도 있다. 즉, u의 결과가 v의 입력일 수도 있고, v의 결과가 u의 입력일 수도 있다는 의미이다.

참여집합은 두 노드간에 간접종속이 존재하는가를 판단하는데 사용할 수 있지만 올바른 스레드 분할을 참여집합만으로 얻을 수는 없다. 한 노드가 둘 이상의 후속노드를 가지고, 후속노드들간에 간접종속관계가 존재할 때 병합하려는 두 노드간의 간접종속 여부뿐만 아니

라 자손노드와의 간접종속여부도 고려하여야 한다. 이를 위하여 노드 u와 u의 자손노드가 결과를 만드는데 사용되지 않는 입력의 집합이 필요하다. 이를 분리집합이라 한다.

[정의 2] 분리집합

출력명 또는 노드 u의 분리집합 $S(u)$는 u가 사용하지 않는 모든 입력명을 원소로 갖는 집합이다.

$$S(u) = \begin{cases} \bigcup_{v \in Dsc(u)} S(v), & u \in N \\ In - P(u), & u \in Out \end{cases} \quad (3)$$

u가 내부노드이면 u의 자손노드의 분리집합의 합집합이고, u가 출력명이면 u의 참여집합의 여집합이다. 여기서 $Dsc(u)$는 u의 자손노드의 집합이다.

노드 u는 다음과 같은 식 (4)를 만족하는 후속 노드 v와 같은 스레드 내에 존재할 수 없다.

$$S(u) \cap P(v) \neq \emptyset \quad (4)$$

식 (4)는 u와 u의 자손노드가 간접 종속을 갖는 입력을 사용하는 노드와 병합되지 않음을 나타낸다.

3.2 자료형 분리집합 분할

본 논문에서 사용하는 언어에서 변수, 함수 등 모든 이름은 자료형을 갖고 정적으로 자료형 검사가 이루어진다고 가정한다. 기본 자료형에는 정수, 실수, 문자, 논리값이 있으며, 구조 자료형(structured data type)으로는 기본 자료형의 값을 원소로 갖는 배열이 있다.

(그림 1)은 정수 a, b, c를 입력 인자로 하여 논리값 x와 실수 y의 튜플을 결과로 생성하는 함수로 (그림 2)와 같은 데이터플로우 그래프로 변환된다. 입력명, 출력명과 그들의 자료형은 (이름, 자료형) 형태의 순서쌍으로 표현하였다. *int*는 정수형, *real*은 실수형, *char*는 문자형, *bool*은 논리형을 각각 나타낸다. 이외에 배열의 자료형은 〈*Arr*, 원소의 자료형〉의 형식으로 표현한다.

```
func foo(int a, b, c) returns (bool, real)
type bool x; real y = =
  let x <- (a + 1) = (b * 2);
      y <- (b * 2) / (c - 1);
  in  (x, y);
```

(그림 1) 비평가인자 함수 프로그램
(Fig. 1) Non-strict Functional Program

자료형 정보를 고려하지 않는다면 (그림 2)의 (a)와 같은 데이터플로우 그래프는 3개의 스레드로 분할된다. 그 이유는 입력명 a와 출력명 y 그리고 입력명 c와 출력명 x사이에 잠재 간접 종속이 존재하기 때문이다. 하지만 입력명 a의 자료형이 정수형이고, 출력명 y의 자료형이 실수형이라면 a와 y사이에는 잠재 간접 종속이 존재하지 않으며, 입력명 c의 자료형이 정수형이고 출력명 x의 자료형이 논리형인 경우에도 역시 서로 잠재 간접 종속이 존재하지 않을 것이다. 따라서 (그림 2)의 (a)와 같은 데이터플로우 그래프는 (그림 2)의 (b)와 같이 하나의 스레드로 분할될 수 있다.

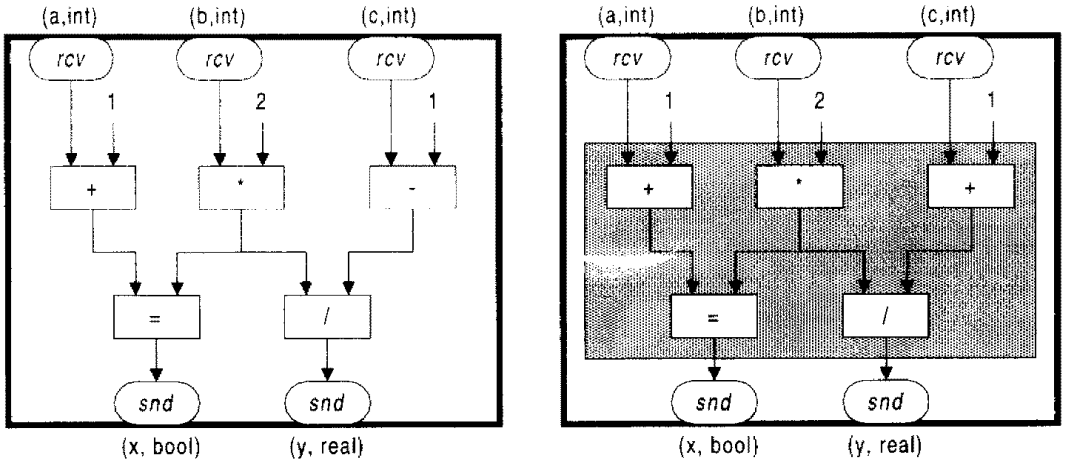
노드가 자신의 자료형과 비교할 수 없는 자료형(incompatible data type)을 가진 출력명과 잠재 간접 종속이 없음을 알고, 이를 스레드 분할에 이용하기 위하여 자료형 분리집합(typed separation set)을 정의한다. 자료형 분리집합은 분리집합의 부분 집합이다.

입력명, 출력명, 그리고 노드의 자료형을 나타내기 위하여 데이터플로우 그래프를 구성하는 입력명의 집합, 출력명의 집합, 그리고 노드의 집합을 [정의 3]과 같이 정의한다.

[정의 3] 자료형을 갖는 데이터플로우 그래프의 구성 요소

자료형을 갖는 데이터플로우 그래프를 구성하는 요소는 다음과 같다.

- Σ_{io} : alphabets which represent input names and output names,
- $In \subset \Sigma_{io}$, $Out \subset \Sigma_{io}$
- O : a set of nodes
- T : a set of types, { ***int***, ***bool***, ***real***, ***char***, ***int_array***, ***bool_array***, ***real_array***, ***char_array*** }
- $Tin = In \times T$, a set of tuple (ν_i, τ_i), where $\nu_i \in In$ and $\tau_i \in T$
- $Tout = Out \times T$, a set of tuple (ν_o, τ_o), where $\nu_o \in Out$ and $\tau_o \in T$
- $TN = O \times T$, a set of tuples (o, τ_n), where $o \in O$ and $\tau_n \in T$
- E : a set of arcs e, where $e \in (In \cup TN) \times (Out \cup TN)$



(그림 2) 자료형을 갖는 데이터플로우 그래프와 분할
(Fig. 2) Dataflow graph with type information and its partitioning

[정의 3]에서 Σ_{IO}는 입력명과 출력명을 나타내는 알파벳이고 O은 노드의 집합이다. T는 자료형의 집합으로 정수형, 논리형, 실수형, 문자형, 그리고 그들의 배열형을 원소로 갖는다. Tin은 입력의 집합으로 입력명 v_i와 입력명의 자료형 τ_i의 순서쌍을 원소로 갖는다. $Tout$은 출력의 집합, TN은 노드의 집합으로 Tin과 비슷하게 정의된다. $n = (x, \textbf{\textit{int}}) \in Tin$일 때 $x = nameof(n)$ 그리고 $\textbf{\textit{int}} = typeof(n)$으로 지정된다. $Tout$과 TN의 원소도 비슷하게 지정된다.

자료형 분리집합은 "노드의 분리집합"과 "노드의 출력명의 자료형과 비교할 수 없는 자료형을 갖는 입력명의 집합"의 차집합이다. 자료형 분리집합을 정의하기 전에 노드의 자료형과 비교할 수 없는 자료형을 가진 입력의 집합(set of inputs with incompatible type)을 정의하자.

[정의 4] 비교 불가능한 자료형을 가진 입력의 집합

노드 u의 자료형과 비교할 수 없는 자료형을 갖는 입력명의 집합 $TIS(u)$는 다음과 같이 정의된다.

$$TIS(u) = \bigcup_{w \in S(u)} \{ v \}$$
$$such\ that\ incompatible(typeof(v), typeof(w)) = true \quad (6)$$
, where $w \in Dsc(u) \cap Out$

[정의 4]는 노드 u의 분리집합의 원소들 가운데 출력명의 자료형과 비교할 수 없는 자료형을 가진 입력명의 집합을 정의한다. [정의 4]에서 $Dsc(u) \cap Out$는 u의 자손인 출력명이다. $typeof(u)$는 입력명, 출력명, 또는 노드인 u의 자료형을 돌려주는 함수이며, $incompatible(typeof(v), typeof(w))$는 입력명 v의 자료형과 출력명 w의 자료형을 비교하여 비교 가능하면 거짓을, 비교할 수 없으면 참을 돌려주는 함수이다. 함수 $incompatible$의 정의는 적용되는 언어에 따라 달라질 수 있다. 본 논문에서는 스레드 분할의 효과를 높이기 위하여 엄격한 자료형 시스템(strongly typed system)을 사용하였다. 자료형 시스템에 대해서는 [1] 등에 소개되어 있다.

출력명 또는 노드 u의 자료형 분리집합 $TS(u)$는 [정의 5]와 같이 분리집합 $S(u)$와 [정의 4]에서 정의된 집합 $TIS(u)$의 차집합으로 정의된다.

[정의 5] 자료형 분리집합

출력명 또는 노드 u의 자료형 분리집합 $TS(u)$는 다음과 같이 정의된다.

$$TS(u) = S(u) - TIS(u) \quad (7)$$

(그림 2)의 (a)와 같은 데이터플로우 그래프의 노드

눈의 자료형 분리집합을 구하여 보자. 데이터플로우 그래프를 구성하는 입력명, 출력명, 그리고 노드의 자료형을 갖는 집합은 다음과 같다.

Tin = {(a, ***int***), (b, ***int***), (c, ***int***)}

$Tout$ = {(x, ***bool***), (y, ***real***)}

TN = {(+, ***int***), (*, ***int***), (-, ***int***), (=, ***bool***), (/, ***real***)}

노드 u에 대하여 $O(u) = Dsc(u) \cap Out$라 할 때 각 노드 u의 $S(u)$와 $O(u)$는 다음과 같다.

$S(+) = \{c\}$ $O(+) = \{x\}$

$S(*) = \{a, c\}$ $O(*) = \{x, y\}$

$S(-) = \{a\}$ $O(-) = \{y\}$

$S(=) = \{c\}$ $O(=) = \{x\}$

$S(/) = \{a\}$ $O(/) = \{y\}$

이로부터 $TIS(u)$와 $TS(u)$를 구하면 다음과 같다.

$TIS(+) = \{c\}$ $TS(+) = S(+) - TIS(+) = \{c\} - \{c\} = \emptyset$

$TIS(*) = \{a, c\}$ $TS(*) = S(*) - TIS(*) = \{a, c\} - \{a, c\} = \emptyset$

$TIS(-) = \{a\}$ $TS(-) = S(-) - TIS(-) = \{a\} - \{a\} = \emptyset$

$TIS(=) = \{c\}$ $TS(=) = S(=) - TIS(=) = \{c\} - \{c\} = \emptyset$

$TIS(/) = \{a\}$ $TS(/) = S(/) - TIS(/) = \{a\} - \{a\} = \emptyset$

모든 노드의 자료형 분리집합이 공집합이라는 사실은 모든 노드의 출력이 잠재 간접 종속을 갖는 입력명이 없다는 의미이므로 (그림 2)의 (a)와 같은 그래프의 모든 노드는 (그림 2)의 (b)와 같이 하나의 스레드로 병합될 수 있다.

[알고리즘 1] 자료형 분리집합 분할 알고리즘

Inputs Dataflow Graph $G = (N, E, In, Out)$

Outputs Thread Graph TG

Temporaries Participation Set of node u, $P(u)$
Separation Set of node u, $S(u)$
Typed Separation Set of node u, $TS(u)$

Procedure

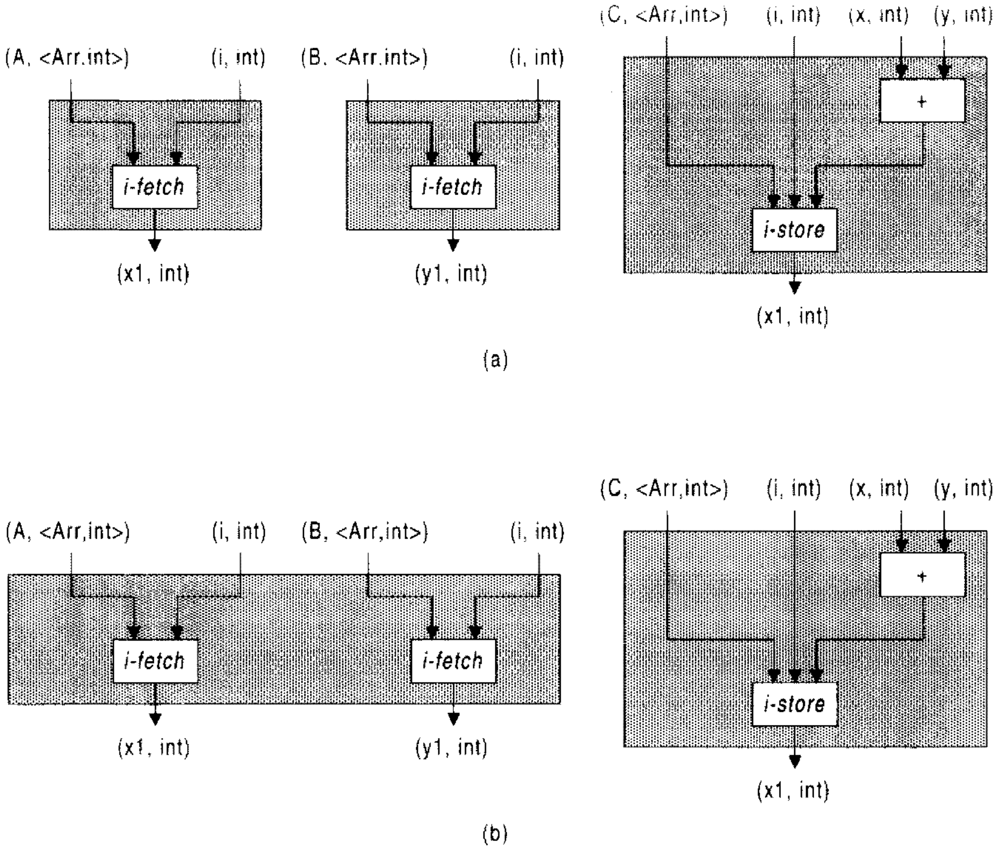
1 Identify all certain indirect dependences into potential indirect dependences;
2 Compute $P(u)$, $S(u)$ for all nodes $u \in N$;
3 Compute $TS(u)$ for all nodes $u \in N$;
4 **Repeat**
5 Select a node $u \in N$ in topological order;
6 **Repeat**
7 Select a node v such that $u \neq v$;
8 **if** ($P(u) \cap TS(v) = \emptyset$) **and** ($P(v) \cap TS(u) = \emptyset$) **then**
9 Merge v into u;
10 Remove v;
11 $P(u) = P(u) \cup P(v)$;
12 $TS(u) = TS(u) \cup TS(v)$;
end-if;
13 **until** no more node such that $u \neq v$;
14 **until** no more node can be merged
End-Procedure

[알고리즘 1]은 자료형 분리집합을 이용하여 비평가인자 함수 프로그램을 스레드로 분할하는 알고리즘이다. 단계 1은 잠재 간접 종속과 상시 간접 종속을 통합하는 과정이다. 이를 수행함으로써 반복 분할에서 필요한 부분할[16]을 피하거나 분리제약 분할의 높은 수행비용[14]을 감소시킬 수 있다. 단계 2와 3에서 스레드 분할에 필요한 노드들의 참여집합, 분리집합, 그리고 자료형 분리집합을 계산한다. 다음 노드를 위상순서(topological order)로 운행하면서 간접 종속이 존재하지 않는 노드들을 병합한다. 노드의 병합 여부를 결정하는 조건으로 식 (7)을 사용한다.

$$(P(u) \cap SS(v) = \emptyset) \wedge (P(v) \cap SS(u) = \emptyset) \quad (7)$$

```
func vec_add(int[] A, B; int n) returns int[];
type int i; int[10] C ==
  for i from 1 to n {
    C[i] <- A[i] + B[i] }
  final C;
```

(그림 3) 배열의 덧셈 프로그램
(Fig.3) Array Addition Program



(그림 4) 배열 합을 구하는 데이터플로우 그래프
(Fig. 4) Dataflow graph which adds two arrays

자료형 분리집합 분할 알고리즘은 기본 블록의 입력명과 출력명의 자료형 정보를 이용하여 출력명과 간접 잠재 종속을 가질 수 없는 입력명들을 분리집합에서 제거함으로써 비평가인자 함수 프로그램의 스레드 분할을 향상시키는 알고리즘이다.

[알고리즘 1]의 복잡도는 자료형 분리집합을 구하는 과정으로 인하여 $O(n^3)$이다. 자료형 분리집합을 구하는 과정은 [정의 4]에서 보이는 것과 같이 모든 노드의 분리집합에 속한 입력명의 자료형과 출력명의 자료형을 비교하여야 한다. 입력명의 수, 출력명의 수 그리고 노드의 수를 n개라 할 때 최악의 경우 분리집합에 속하는 입력명의 수는 n개이므로 자료형 분리집합을 구하는 과정의 시간 복잡도는 $O(n^3)$이 된다. 나머지 부분의 시간 복잡도는 모두 $O(n^2)$이다.

(그림 3)은 두 배열 A와 B의 합을 구하여 C에 저장하는 프로그램이다. 이 프로그램에서 루프 몸체를 데이터플로우 그래프로 표현하면 (그림 4)와 같다. (그림 3)과 같은 비평가인자 함수 프로그램의 루프 몸체에 분리집합 분할 알고리즘을 적용하면 (그림 4)의 (a)와 같이 3개의 스레드로 분할된다. 하지만 $A[i]$를 채취하는 노드의 출력과 $B[i]$를 채취하는 노드의 입력과는 자료형을 고려할 경우 잠재 간접 종속이 존재하지 않으므로 두 노드는 (그림 4)의 (b)와 같이 하나의 스레드로 병합할 수 있다.

이제 자료형 분리집합을 이용하여 스레드로 분할하는 방법을 (그림 4)의 그래프를 통하여 알아보자. $A[i]$를 채취하는 노드와 $B[i]$를 채취하는 노드 두 노드의 결과의 합을 구하는 노드, 그리고 결과를 $C[i]$에 저장하는 노드를 각각 1, 2, 3, 4라 할 때 노드들의 참여집합과 자료형 분리집합은 다음과 같다.

$P(1) = \{A, i\}$ $TS(1) = \{x, y\}$
$P(2) = \{B, i\}$ $TS(2) = \{x, y\}$
$P(3) = \{x, y\}$ $TS(3) = \{A, B\}$

$P(4) = \{C, x, y, i\}$ $TS(4) = \{A, B\}$

노드 1과 2를 병합하자. $P(1) \cap TS(2) = \emptyset$이므로 노드 1과 2는 노드 1로 병합 가능하게 되며 노드 2는 제거된다. 노드 1과 2를 노드 1로 병합한 후 각 노드의 참여집합과 자료형 분리집합은 다음과 같다.

$P(1) = \{A, B, i\}$ $TS(1) = \{x, y\}$

$P(3) = \{x, y\}$ $TS(3) = \{A, B\}$

$P(4) = \{C, x, y, i\}$ $TS(4) = \{A, B\}$

노드 1과 3을 병합하자. $P(1) \cap TS(3) = \{B\}$이므로 1과 3을 병합할 수 없다. 노드 1과 4도 같은 이유로 병합할 수 없다.

다음 노드 3과 4를 병합하자. $P(3) \cap TS(4) = \emptyset$이므로 노드 3과 4는 노드 3으로 병합 가능하게 되며 노드 4는 제거된다. 노드 3과 4를 노드 3으로 병합한 후 각 노드의 참여집합과 자료형 분리집합은 다음과 같다.

$P(1) = \{A, B, I\}$ $TS(1) = \{x, y\}$

$P(3) = \{C, x, y, I\}$ $TS(3) = \{A, B, C, x, y\}$

노드 1과 3을 병합하자. $P(1) \cap TS(3) = \{A, B\}$이므로 1과 3을 병합할 수 없다. 최종적으로 (그림 3)의 비평가인자 함수 프로그램은 (그림 4)의 (b)와 같이 분할된다.

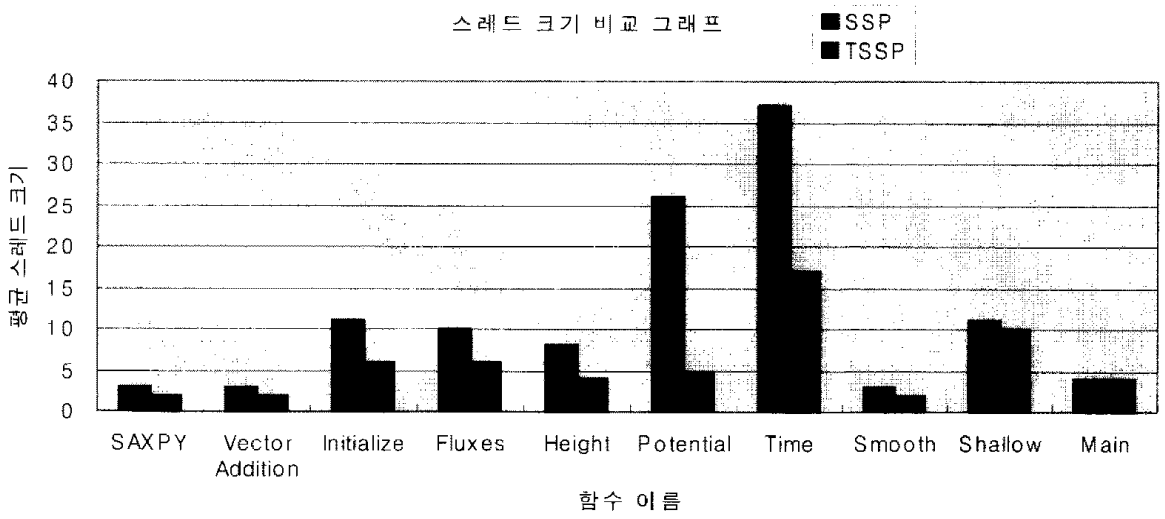
4. 실 험

이 장에서는 자료형 분리집합 분할과 분리집합 분할 알고리즘을 적용하여 생성된 스레드의 길이를 비교함으로써 자료형 분리집합 분할 알고리즘의 우수성을 보인다. 실험에 사용된 프로그램은 X = a*X + Y를 계산하는 SAXPY 프로그램, 배열의 합을 구하는 프로그램 그리고 간단한 일기예보 프로그램인 shallow라는 프로그램이다. SAXPY 프로그램과 배열 합 계산 프로그램은 루프의 몸체만을 사용하였으며, 8개의 함수로 구성되어 있는 shallow 프로그램은 프로그램 전체를 대상으로 하였다. 실험 대상 프로그램은 모두 많은 배열 연산을 포함한다.

(그림 5)의 실험 결과에서 보듯이 자료형 분리집합 분할 알고리즘은 분리집합 알고리즘보다 모든 실험 대상에 대하여 더 큰 스레드를 생성한다. 분리집합 분할 알고리즘과 다른 스레드 분할 알고리즘의 비교는 [16]에 제시되어 있다. 자료형 분리집합 분할은 그의 특성상 여러 자료형이 존재하는 프로그램 특히, 많은 배열 연산이 존재하는 프로그램에 뛰어난 성능을 보인다.

5. 결 론

본 논문에서는 비평가인자 함수 프로그램을 스레드로 분할하는 자료형 분리집합 분할 알고리즘을 제안하였다. 자료형 분리집합 분할은 입력과 출력의 이름만을 고려하여 스레드 분할을 수행하는 기존의 스레드 분할 알고리즘과는 달리 입력과 출력의 자료형 정보를 사용한다. 입력과 출력의 자료형 정보를 사용함으로써 간접



(그림 5) 분리집합 분할과 자료형 분리집합 분할의 스레드 크기 비교 그래프
(Fig. 5) Comparison of Separation Set Partitioning and Typed Separation Set Partitioning

종속이 존재하지 않는 입력과 출력의 쌍을 더 많이 찾아낼 수 있으며 이로부터 더 큰 스레드의 분할을 얻을 수 있다.

본 논문에서 제안한 스레드 분할 알고리즘은 하나의 기본 블록을 스레드로 분할하는 지역분할 방법이므로 함수 호출까지 고려한 전역 스레드 분할방법에 대해서도 연구가 진행되어야 한다.

스레드 분할 알고리즘의 유용성을 평가하는 방법으로는 스레드의 평균 크기, 동적으로 나타나는 명령들의 분포 그리고 각 분할 방법에 대한 프로그램의 수행 시간을 계산하는 방법 등이 있다[9, 14]. 본 논문에서는 개념적인 다중스레드 모델을 가정하였기 때문에 스레드로 분할된 비평가인자 함수 프로그램의 동적 특성을 분석하지 못하고 단지 분할된 스레드의 평균 크기 및 스레드를 분할하는데 방문되는 노드의 수만을 비교하였다.

또한 계산의 입자 크기가 증가됨에 따라 한 스레드가 수행되기 위하여 필요한 모든 입력을 기다리는 지연시간이 적은 입자 크기를 갖는 스레드에 비하여 증가될 수도 있다. Najjar등의 연구[14]에 따르면 프로그램에 존재하는 모든 병렬성을 활용할 만큼 충분한 수의 처리기가 존재하지 않는다면 입력을 기다리는 지연시간은 동적 동기화 비용 감소로 인하여 상쇄된다. 하지만 입력을 기다리는 지연시간도 프로그램의 수행 성능에 영향을 미치는 요소 중의 하나이므로 이에 대한 분석이 필요할 것이다.

이상과 같은 이유로 본 논문에서 제안된 스레드 분할 알고리즘이 비평가인자 함수 프로그램의 수행 성능 향상에 미치는 영향을 자세히 분석하기 위해서는 다중스레드 모델의 구현과 다중스레드 모델을 위한 코드 생성에 관한 연구가 필요하다.

참 고 문 헌

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

[2] Arvind and R. A. Iannucci, *Two Fundamental Issues in Multiprocessors: The Dataflow Solutions*, MIT/LCS/TR-226-6, Laboratory for Computer Science, MIT, 1987.

[3] Arvind, R. S. Nikhil, and K. Pingali, "I-Structures: Data Structures for Parallel Computing," *ACM Transactions on Programming Languages and Systems*, Vol.11, No.4, pp.598-632, 1990.

[4] D. E. Culler, S. C. Goldstein, K. E. Schauser, and T. von Eicken, "TAM - A Compiler-Controlled Threaded Abstract Machine," *Journal of Parallel and Distributed Computing*, Vol.18, No.3, pp.347-370, 1993.

[5] J. -L. Gaudiot and L. Bic(editors), *Advanced Topics in Data-flow Computing*, Prentice-Hall, 1991.

[6] V. G. Grafe and J. E. Hoch, "The Epsilon-2 Multiprocessor System," *Journal of Parallel and Distributed Computing*, Vol.10, No.4, pp.309-318, 1990.

[7] J. E. Hoch, D. M. Davenport, V. G. Grafe, and K. M. Steele, "Compile-time Partitioning of a Nonstrict Language into Sequential Threads," *Proceedings of 3rd IEEE Symposium on Parallel and Distributed Processing*, pp.180-189, 1991.

[8] R. A. Iannucci, *Parallel Machines: Parallel Machine Languages The Emergence of Hybrid Dataflow Computer Architectures*, Kluwer Academic Publishers, 1990.

[9] W. A. Najjar, L. Roh, and A. P. W. Bohm, "An Evaluation of Medium-Grain Dataflow Code," *Int'l Journal of Parallel Programming, Vol.22, No.3*, pp.209-242, 1994.

[10] R. S. Nikhil and Arvind, "Can dataflow subsume von Neumann computing?," *Proceedings of 16th Annual International Symposium on Computer Architecture*, pp.262-272, 1989.

[11] R. S. Nikhil, G. M. Papadopoulos, and Arvind, "*T: A Multithreaded Massively Parallel Architecture," *Proceedings of 19th Annual International Symposium on Computer Architecture*, pp.156-167, 1992.

[12] S. L. Peyton Jones, "Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-Machine," *Journal*

of Functional Programming, April, 1992.

[13] K. E. Schauser, D. E. Culler, and T. von Eicken, "Compiler-Controlled Multithreading for Lenient Parallel Languages," *Proceedings of Symposium on Functional Programming Languages and Computer Architectures*, LNCS Vol.523, pp.50-72, 1991.

[14] K. E. Schauser, D. E. Culler, and S. C. Goldstein, "Separation Constraint Partitioning - A New Algorithm for Partitioning Non-strict Programs into Sequential Threads," *21th ACM Symposium on Principles of Programming Language*, pp.259-271, 1995.

[15] K. R. Traub, *Implementation of Non-strict Functional Programming Languages*, Research Monographs in Parallel and Distributed Computing, MIT Press, 1991.

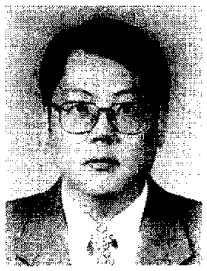
[16] K. R. Traub, D. E. Culler, and K. E. Schauser, "Global Analysis for Partitioning Non-strict Programs into Sequential Threads," *Conference on Lisp and Functional Programming*, pp.324-334, 1992.

[17] C. M. Yang, H. S. Joo, and W. H. Yoo, "Separation Set Partitioning: Algorithm to Partition Non-strict Programs into Sequential Threads," 30th Hawaiian International Conference on System Science, pp.626-627, 1997.

[18] 양창모, 유원희, "분리집합을 이용한 Nonstrict 프로그램의 스레드 분할," 정보과학회논문지(B), 24권 8호, pp.891-899, 1997년 8월.

양 창 모

1985년 인하대학교 전자계산학과(이학사)
1988년 인하대학교 대학원 전자계산학과(이학석사)
1997년 인하대학교 대학원 전자계산학과(공학박사)
1990년~1998년 동명전문대학 전자계산과 부교수
1998년~현재 청주교육대학교 전임강사
관심분야 : 프로그래밍 언어(함수언어), 계산 모델, 병렬 처리



주 형 석

1985년 인하대학교 전자계산학과(이학사)
1987년 인하대학교 대학원 전자계산학과(이학석사)
1997년 인하대학교 대학원 전자계산학과(공학박사)
1987년~현재 유한전문대학 전자계산과 부교수
관심분야 : 프로그래밍 언어(함수언어, 객체지향언어), 계산 모델, 병렬 시스템 등임.

유 원 희

1975년 서울대학교 공과대학 응용수학과(이학사)
1978년 서울대학교 대학원 계산학전공(이학석사)
1985년 서울대학교 대학원 계산학전공(이학박사)
1979년~현재 인하대학교 공과대학 전자계산공학과 교수
관심분야 : 프로그래밍 언어(함수언어), 컴파일러, 계산 모델, 병렬 처리, 실시간 시스템