융통성 있는 스레드 분할 시스템 설계와 평가

Design and Evaluation of Flexible Thread Partitioning System

조 선 문*
Sun-Moon Jo

요 약

다중스레드 모델은 긴 메모리 참조 지체 시간과 동기화의 문제점을 해결할 수 있다는 점에서 대규모 병렬 시스템에 매우 효과적이다. 다중스레드 병렬기계를 위하여 Non-Strict 함수 프로그램을 번역할 때 가장 중요한 것은 순차적으로 수행될 수 있는 부분을 찾아내어 스레드로 분할하는 것이다. 기존의 분할 알고리즘은 조건식의 판단식, 참실행식, 거짓실행식을 기본 블록으로 나누고 각각에 대하여 지역 분할을 적용한다. 이러한 제약은 스레드의 정의를 약간 수정하여 스레드 내에서의 분기를 허용한다면 좀더 좋은 분할을 얻을 수 있다. 스레드내에서의 분기는 병렬성을 감소시키거나 동기화의 횟수를 증가시키거나 또는 교착상태를 발생시키는 등 스레드 분할의 기본 원칙을 어기지 않으며 오히려 스레드 길이를 증가시키거나 동기화 횟수를 줄이는 장점을 가질 수 있다. 본 논문에서는 조건식의 세 가지 기본 블록을 하나 또는 두 개의 기본 블록으로 병합함으로서 스레드 분할을 향상시키는 방법을 제안한다.

Abstract

Multithreaded model is an effective parallel system in that it can reduce the long memory reference latency time and solve the synchronization problems. When compiling the non-strict functional programs for the multithreaded parallel machine, the most important thing is to find a set of sequentially executable instructions and to partitions them into threads. The existing partitioning algorithm partitions the condition of conditional expression, true expression and false expression into the basic blocks and apply local partitioning to these basic blocks. We can do the better partitioning if we modify the definition of the thread and allow the branching within the thread. The branching within the thread do not reduce the parallelism, do not increase the number of synchronization and do not violate the basic rule of the thread partitioning. On the contrary, it can lengthen the thread and reduce the number of synchronization. In the paper, we enhance the method of the partition of threads by combining the three basic blocks into one or two blocks.

☞ Keyword : Data Flow, Multithreaded, Functional Language, Dependence Set Partitioning, 데이터 플로우, 다중스레드, 함수언어, 종속 집합 분할

1. 서 론

컴퓨터 설계에서는 성능 향상을 위한 방법으로서 병렬처리 기술이 널리 사용되고 있다. 그러나 수많은 상용 칩의 프로세서를 탑재한 병렬 컴퓨터는 증가하는 메모리 참조의 지체 시간과 동기화 부담으로 인한 성능 체증의 문제점을 안고 있기 때문에, 현재는 다중스레드 모델(Multithreaded model)로 이러한 문제점을 해결하고자 하는 연구가 진행되고 있다.

다중스레드 모델은 데이터플로우 모델의 단점을 해결하기 위한 방안으로 프로그램에 내재한 병렬성을 최대한 활용하는 데이터플로우 모델과 지역성을 이용하여 순차 코드를 효율적으로 수행할 수 있는 폰 노이만 모델의 혼합형 계산 모델이다[1]. 다중스레드 모델의 기본 정책은 수행 가능한 스레드들 사이에서의 빠른 문맥 전환을 이용하여 동기화나 원격자료 접근 등에서 발생하는 지연을 은폐함으로서 처리기의 활용도를 향상시키는 것이다. 다중스레드 모델에서 프로그램은 함

* 정 회 원 : 배재대학교 IT교육 교수
sunmoon@pcu.ac.kr
[2006/11/25 투고 - 2006/12/27 심사 - 2007/04/04 심사완료]

수, 루프 등이 단위가 되는 코드 블록으로 구성되며, 각 코드블록은 번역시간에 수행 순서를 결정할 수 있는 명령의 모임인 스레드로 분할된다.

Non-Strict 함수 언어는 내재된 병렬성을 사용할 수 있으며, 원소의 값이 정의되지 않은 자료구조를 사용하던가 인자가 평가되지 않은 함수의 결과를 돌려주는 등의 유연성으로 인하여 프로그래머에게 높은 수준의 표현력을 제공한다[5]. 그러나 이러한 Non-Strict 함수 프로그램을 기존의 폰 노이만형 병렬기계에서 수행할 때, Non-Strict로 인하여 미세수준(fine grain)의 동적 스케줄링이나 동기화가 필요하기 때문에 기존의 병렬 기계에서의 구현이 어렵다[3]. 이러한 언어를 Threaded model를 사용하는 병렬기계를 위하여 번역할 때, 프로그램의 지역성을 살리기 위하여 프로그램을 순차적으로 분할하는 과정이 필요하다. 병렬 기계에서 Non-Strict 함수 언어를 구현할 때, 자료 전달, 동기화와 Non-Strict 의미로 인하여 스레드의 크기가 제한을 받는다. 따라서 다중스레드 모델을 위하여 Non-Strict 함수 프로그램을 스레드로 분할할 때 스레드 분할의 목적은 단순히 스레드의 크기를 최대화하여 동기화의 발생 횟수와 스레드 전환 횟수를 최소화함으로써 동적 동기화 비용을 감소시키고 프로그램의 수행 성능을 높이고자 하는 것이다[2].

다중스레드 모델을 위하여 함수 프로그램을 스레드로 분할하는 방법은 크게 하향식과 상향식 두 가지 방법으로 나뉜다. 전통적인 다중처리 시스템을 위하여 C와 같은 명령형 언어로 작성된 프로그램을 번역할 때 사용되는 하향식 방법은 프로그램으로부터 자료 종속 그래프(Data Dependence Graph)를 생성하고 이로부터 직접 코드를 생성한다. 이 방법은 태스크간의 통신 부담을 최소화할 만큼 태스크를 크게, 병렬성을 얻을 수 있을 만큼 태스크를 적당히 작도록 하는 것이 목표이다. 그러나 이 방법은 프로그램 중 연산의 수행 순서를 번역시간에 결정할 수 있는 언어에 적합한 방법이다. 이 방법이 다중스레드 코드 생성에 사용된 예는 SISAL 컴파일러이다[3], [9], [13].

상향식 스레드 분할 방법으로는 종속 집합 분할(dependence set partitioning)[1], 요구 집합 분할(demand set partitioning)[7], 종속 집합 분할과 요구 집합 분할을 반복적으로 적용하는 반복 분할(iterated partitioning)[6], 분리 제약 분할[separation constraint partitioning)[8] 등이 있다. 기존의 분할 알고리즘은 조건식을 판단식(predicate), 참실행식(then-arm), 거짓실행식(then-arm)을 기본 블록으로 나누고 각각에 대하여 지역 분할을 적용하기 때문에, 동기화 횟수와 스레드간의 문맥 전환 횟수가 커서, 처리하는 비용이 많이 들기 때문에 스레드 수행의 성능이 저하된다.

본 논문에서는 Non-Strict 함수 프로그램에서 조건식의 세 가지 기본 블록을 하나 또는 두 개의 기본 블록으로 병합함으로서 다중스레드 모델에서 스레드의 문맥 전환 횟수와 동기화 횟수를 감소하여 프로그램의 수행 성능을 향상시키는 방법을 제안한다. 또한 벤치마크에 대한 시뮬레이션을 통하여 제안된 스레드 코드의 생성과 비교한다.

2. 조건식의 Non-Strict 의미와 기존의 문제점

그림 1에서 간단한 조건식 (a)와 (b)를 보면, 조건식은 판단식(E_1), 참수행식(E_2), 그리고 거짓수행식(E_3)으로 구성된다. E_1은 결과가 논리값인 식이고 E_2와 E_3은 결과 값의 자료형이 동일한 임의의 식이다. 조건식은 문장이 아니므로 참수행식과 거짓수행식이 모두 존재하여야 한다.

```
(a) if  E1  then  E2  else  E3
(b) if  X == 0  then  a + b  else  a - b
```

(그림 1) 간단한 조건식 예

Non-Strict 의미를 사용하는 방법에는 그래프 감축을 이용한 지연 평가와 leninet 평가라는 두 가지가 있다. Haskell과 같은 언어를 구현할 때 사용되는 지연 평가는 가능한 순간까지 연산의 평가를 미룬다. 프로그램의 결과를 얻을 수 있는 평가 순서가 존재한다면 지연 평가는 항상 결과를 얻을 수 있는 장점이 있다[14]. Id와 같은 언어를 구현할 때 사용되는 lenient 평가는 가능한 한 연산의 평가를 늦추는 지연 평가와 달리 연산의 평가에 필요한 값이 사용 가능할 때까지 연산의 평가를 늦춘다[12], [14].

```
let   p = (n ==0);
      a  = if  p then 3 else d;
      b  = if  p then c else 4;
      c = a + 2; d = b+1; e = c *d
in   e
```

(그림 2) Non-Strict 예

그림 2와 같이 조건식에서의 Non-Strict 의미를 보여주고 있다. 그림 2에서 만약 인자가 모두 사용할 수 있는 경우에 함수나 식을 평가할 수 있으면, p가 거짓인 경우 a는 d에 종속되며, d는 b에, b는 c에, c는 a에 종속되어 교착 상태가 발생한다. 또한 p가 참일 때 b는 c에, c는 a에, a는 d에, d는 b에 종속되어 교착 상태가 일어난다. 그러나 Non-Strict 하게되면 p가 참일 때 a = 3, c = 5, b = 5, d = 6의 순서로 계산되어 e = 30이 되고, p가 거짓일 때 b = 4, d = 5, a = 5, c = 7의 순서로 계산되어 e = 35가 된다[14].

다중스레드 컴퓨터에서 스레드가 효과적으로 실행되기 위해서는 스레드 분할시 여러 사항을 고려해야 한다. 스레드 분할시 고려 사항은 병렬성의 극대화, 스레드 길이의 최대화, 동기화 비용의 최소화, 교착 상태의 회피, 자원 활용도의 극대화 등이 있다[1]. 다중스레드 모델의 Non-Strict 프로그램을 스레드로 분할할 때 고려하여야 하는 조건에 대하여 스레드를 정의하면 다음과 같다[8], [11].

첫 번째, 어떤 문맥에서도 스레드 내에서 명령의 올바른 수행 순서를 번역시간에 결정할 수 있어야 한다.

두 번째, 한번 스레드의 첫 명령이 수행되면 그 스레드를 구성하는 나머지 명령어들은 번역시간에 결정된 순서대로 중단 없이 수행되어야 한다.

명령형 언어로 작성된 프로그램과 달리 Non-Strict 함수 언어로 작성된 프로그램에서 연산의 평가 순서는 구문적으로 명시되지 않으므로 Non-Strict 함수 프로그램으로부터 스레드를 얻는 것은 간단한 일이 아니다. 연산의 평가 순서를 결정하기 위하여 컴파일러는 모든 자료 종속을 파악하여야 한다. Non-Strict 함수 프로그램에서 연산은 주어진 문맥에 따라 수행 순서가 달라질 수 있다. 그러므로 연산들은 수행시간에 동적으로 스케줄링되어야 하므로 스레드의 첫 번째 조건으로 인하여 여러 개의 스레드가 만들어진다. 첫 번째 정의를 만족하도록 스레드를 분할하려면 동적으로 스케줄링되어야 하는 연산들은 하나의 스레드로 묶으면 안 된다. 동적으로 스케줄링되어야 하는 연산들을 식별하기 위해서는 연산들의 종속 관계를 구하여야 한다. 종속 관계는 스레드내의 연산사이에 존재하는 직접 종속과 서로 다른 스레드에 존재하는 연산간에 존재하는 간접 종속으로 나눌 수 있다[11].

두 번째 요구사항에 따른 분할 조건으로 인하여 동기화가 필요한 자료 접근, 원격자료 접근 연산, 함수 호출, 루프 호출 등 변역시간에 수행 시간을 예측할 수 없는 연산들과 그 결과를 이용하는 연산을 서로 다른 스레드에 위치시켜야 한다[13].

기존의 연구로서, Nikhil은 데이터플로우 프로그램 그래프내 노드들을 결합하지 않았다. 즉, 그래프내 각 노드는 한 스레드를 의미한다[4]. 따라서 스레드의 크기는 매우 작으며, 이는 상당히 많은 Fork와 Join 연산을 발생한다. 또한 그의 스레

드 분할 기법에서는 스레드의 병합이 고려되지 않았다.

Schauser의 분할 기법에서 데이터플로우 그래프는 종속 집합이나 요구 종속 집합을 이용하여 수행하였다[8]. 여기서 종속 집합은 레이블과 병합 노드가 새로운 분할의 출발점으로 인식되는 입력 노드로 다루고 있고, 레이블은 스위치 노드의 각 출력 아크에 삽입되어 분할의 제약 조건을 나타낸다. 따라서 Schauser의 분할 기법에서는 조건문의 판단식, 참실행식 부분과 거짓실행식 부분은 각각 서로 다른 스레드에 속하게 된다. 가령, 조건문을 포함하는 데이터플로우 그래프의 간접 종속이 없는 스레드가 포함되어 있거나, 또한 긴 지체 시간을 요하는 명령어가 포함되어 있지 않더라도, 그래프는 적어도 다음과 같이 4개로 분할된다. 스위치와 그 상위 노드를 포함한 스레드 부분, 참실행식 부분, 거짓실행식 부분, 해당 병합 노드와 그 하위 노드를 포함한 스레드 부분이다. 따라서 분할 기법은 작은 크기의 스레드를 상당히 많이 생성할 수 있으므로 동기화 횟수와 스레드간의 문맥 전환 횟수가 커서, 처리하는 비용이 많이 들어 스레드 수행의 성능이 저하된다. Hoch는 제어 벡터를 이용하여 조건문의 참실행식 부분과 거짓실행식 부분을 결합하고자 하나, 여기에서는 branch 문이 추가로 발생하는 문제점이 있으며, 추가되는 branch 명령어는 조건문의 중첩 수준과 함께 증가한다[10].

3. 스레드 분할 시스템 설계

3.1 조건식의 스레드 기법

본 논문에서는 새롭게 도입한 스위치 노드를 이용하여 동일 스레드내 동일 제어 조건을 갖는 여러 스위치 노드를 각각 결합하여 branch 명령어의 수를 감소시키며, 또한 간접 종속관계를 이용하여 조건식의 세 가지 기본 블록을 하나 또는 두 개의 기본 블록으로 병합함으로서 조건식의 스레드 분할을 향상시키는 알고리즘을 제안한다.

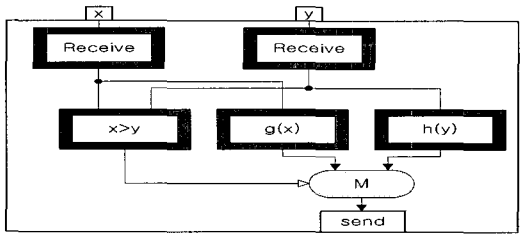
조건식은 다음과 같이 판단식(E_1), 참수행식(E_2), 그리고 거짓수행식(E_3)으로 구성된다.

if E_1 then E_2 else E_3

다음과 같은 함수 f(x, y)를 보면, func f(x, y) = if x>y then g(x) else h(y);

이 함수에 기존의 스레드 분할 방법을 적용하면 판단식, 참실행식, 거짓실행식이 기본 블록이 되고 각 기본 블록은 매개변수를 전달받는 스레드와 결과를 전달하는 스레드를 제외하고 하나의 스레드로 분할되어 그림 3과 같이 세 개의 스레드로 분할된다.

함수 f(x, y)에 대하여 판단식, 참실행식, 거짓실행식을 구성하는 세 개의 기본 블록을 하나의 기본 블록으로 가정하고 분리 집합 분할을 적용하면 세 연산은 간접 종속을 갖지 않으므로 하나의 스레드로 분할할 수 있다.



(그림 3) 함수 f(x, y)의 스레드 분할

그림 4와 같이 지역 분기를 이용한 분할 알고리즘을 제안한다. 스레드내에서 분기는 병렬성을 감소시키거나 동기화의 횟수를 증가시키거나 또는 교착상태를 발생시키는 등의 스레드 분할의 기본 원칙을 어기지 않으며 오히려 스레드 길이를 증가시키거나 동기화 횟수를 줄이는 장점을 가질 수 있다. 하지만 스레드의 정의를 변경한다고 해서 모든 조건식을 하나의 기본 블록으로 병합할 수 있는 것은 아니고, 판단식, 참실행식, 거짓실행식을 구성하는 기본 블록사이에 간접 종속이 존재하지 않는 경우에만 병합할 수 있다. 조건식에서 부분식간에 간접 종속이 생기는 경우는

h(x, y, z, w) = if x=y then z+1 else w-1 와 같이 세 부분이 모두 간접종속을 가지므로 당연히 세 개의 기본 블록 스레드로 분할된다.

입력 : 데이터플로우 그래프
출력 : 스레드 그래프

1. 그래프의 노드를 위상적 순서로 방문하면서 각 노드들을 모아 하나의 분할을 만든다.
2. 동일한 입력 종속 집합을 갖는 노드들을 모아 하나의 분할을 만든다.
3. I-fetch 노드 중에서 결합 가능한 노드를 결정한다.
4. 노드 u의 참여 집합 P(u)와 노드 v의 분리 집합 S(v)와의 교집합을 검사한다.
5. 결합 가능한 I-fetch 노드의 출력 아크와 연결된 분할들을 찾아 이들을 하나의 분할로 통합한다.
6. 1 단계에서 5단계까지 동일한 방법으로 각 분할 스레드를 생성한다.

(그림 4) 지역 분기를 이용한 분할 알고리즘

그림 5는 간접 종속이 없으므로 기본 블록을 병합할 수 있는 조건식의 예들이다.

① (a, b) = if (x = y) then (x+y, x-y) else (x+y, x/y)
② (a, b) = if (x = y) then (x+y, x-y) else (x+w, x-w)
③ (a, b) = if (u = w) then (x+y, x-y) else (x*y, x-y)

(그림 5) 기본 블록을 병합할 수 있는 조건식

그림 5의 ①과 같은 예는 판단식, 참실행식, 거짓실행식을 구성하는 기본 블록사이에 간접종속이 존재하지 않으므로 세 기본 블록을 하나의 기본 블록으로 병합할 수 있다.

②와 같은 예는 판단식, 참실행식을 구성하는 기본 블록사이에 간접종속이 존재하지 않으므로 이들을 병합하여 하나의 기본 블록으로 병합할 수 있다.

③과 같은 예는 참실행식, 거짓실행식을 구성하는 기본 블록사이에 간접종속이 존재하지 않으므로 이들 두 기본 블록을 하나의 기본 블록으로 병합할 수 있다.

입력 : G(if E1 then E2 else E3)
출력 : T(if E1 then E2 else E3)

1. G(E1), G(E2), G(E3)의 참여집합과 종속 집합을 구한다.
2. G(E1), G(E2), G(E3)의 간접 종속 관계를 판단한다.
   2.1 G(E1), G(E2), G(E3)가 간접 종속을 갖지 않는다면, G(E1), G(E2), G(E3)를 병합하여 하나의 기본 블록으로 나타낸다.
   2.2 G(E1), G(E2)가 간접 종속을 갖지 않는다면, G(E1), G(E2)를 병합하여 하나의 기본 블록으로 나타내고, G(E3)를 하나의 기본 블록으로 나타낸다.
   2.3 G(E1), G(E3)가 간접 종속을 갖지 않는다면, G(E1), G(E3)를 병합하여 하나의 기본 블록으로 나타내고, G(E2)를 하나의 기본 블록으로 나타낸다.
   2.4 G(E2), G(E3)가 간접 종속을 갖지 않는다면, G(E2), G(E3)를 병합하여 하나의 기본 블록으로 나타내고, G(E1)를 하나의 기본 블록으로 나타낸다.
3. 각 기본 블록에 지역 스레드 분할을 적용하여 T(if E1 then E2 else E3)를 구한다.

(그림 6) 조건식의 스레드 분할 알고리즘

기본 블록간에 간접 종속이 존재하는가를 판단하는 방법은 사용하는 기존 스레드 분할 방법을 이용한다. 기본 블록간에 간접 종속이 없다면 이들을 병합하고 지역 스레드 분할을 수행한다. 그림 6에서 G(E)는 식 E의 데이터플로우 그래프를 의미하고, T(E)는 E의 스레드 집합을 의미한다.

3.2 조건식의 스레드 분할 기법의 정확성

조건식의 스레드 기법에서 스레드 분할의 정확성을 증명하는 것은 스레드들을 분할된 그래프가 사이클을 갖고 있지 않으면 스레드내에 간접종속이 존재하지 않는다고 할 수 있다. 따라서 조건식의 스레드 분할 알고리즘에 의하여 분할된 스레드의 각 노드들이 간접종속을 갖지 않음을 보이는 것이 조건식의 스레드 분할을 증명하는 것이다. 조건식의 스레드 분할에서 노드 u의 참여집합을 P라 하고, 노드 v의 분리 집합을 S라고 하자. 다음 조건을 만족하면 u와 v를 병합할 수 있다.

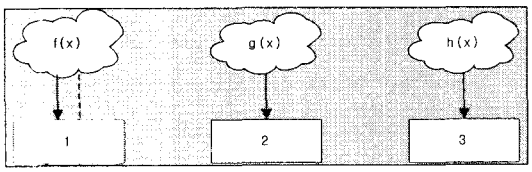
첫째, $P(u) \not\subset P(v)$라면 u와 v사이에 간접종속이 존재하므로 병합할 수 없다.

둘째, $S(u) \cap P(v) \neq \varnothing$이면 u와 v사이에 간

접종속이 존재하므로 병합할 수 없다.

셋째, P(u) ⊂ P(v)이고 S(u) ∩ P(v) = ∅ 이면 u와 v사이에 간접종속이 존재하지 않으므로 병합할 수 있다.

조건식에 대한 정확성을 보이는 예로서 그림 5의 식을 고려하였다. 여기서 X = (x1, x2, x3, ···, xn)이라 하고, 식 f(X) then g(X) else h(X)의 데이터플로우 그래프가 그림 7과 같이 스레드로 분할될 때, 결과를 저장하는 노드를 그림 7과 같이 노드 1, 노드 2, 노드 3 이라고 표현한다.



(그림 7) 결과를 저장하는 노드

따라서, 그림 5의 ①번에서

P(1) = {x, y}　　S(1) = ∅

P(2) = {x, y}　　S(2) = ∅

P(3) = {x, y}　　S(3) = ∅

P(1) ∩ S(2) = ∅이고 P(2) ∩ S(1)=∅이므로 노드 1과 2는 병합된다. 노드 1과 2를 병합한 후 각 노드는 다음과 같다.

P(1) = {x, y}　　S(1) = ∅

P(3) = {x, y}　　S(3) = ∅

P(1) ∩ S(3) = ∅ 이고 P(3) ∩ S(1) = ∅ 이므로 노드 1과 노드3은 병합된다. 그러므로 판단식, 참실행식, 거짓실행식을 한 스레드로 병합할 수 있다.

그림 5의 ②번에서

P(1) = {x, y}　　S(1) = {w}

P(2) = {x, y}　　S(2) = {w}

P(3) = {x, w}　　S(3) = {y}

P(1) ∩ S(2) = ∅ 이고 P(2) ∩ S(1) = ∅ 이므로 노드 1과 2는 병합된다. 노드 1과 2를 병합한 후 각 노드는 다음과 같다.

P(1) = {x, y}　　S(1) = {w}

P(3) = {x, w}　　S(3) = {y}

P(1) ∩ S(3) = {y} 이고 P(3) ∩ S(1) = {w} 이므로 노드 1과 3은 간접종속이 존재하므로 병합할 수 없다. 그러므로 판단식, 참실행식을 병합하고 거짓실행식은 간접종속이 있으므로 병합할 수 없다.

그림 5의 ③번에서

P(1) = {u, w}　　S(1) = {x, y}

P(2) = {x, y}　　S(2) = ∅

P(3) = {x, y}　　S(3) = ∅

P(1) ∩ S(2) = ∅ 이고 P(2) ∩ S(1)={x, y} 이므로 노드 1과 2는 병합할 수 없다.

P(2) ∩ S(3) = ∅ 이고 P(3) ∩ S(2) = ∅ 이므로 노드 2와 3은 병합된다.

P(1) ∩ S(3) = ∅ 이고 P(3) ∩ S(1) = {x, y} 이므로 노드 1과 3은 병합할 수 없다. 따라서 참실행실과 거짓실행식은 병합되고, 판단식과는 병합할 수 없다.

4. 실험 결과

본 논문에서 몇 가지 벤치마크 프로그램의 시뮬레이션을 통해 조건식의 스레드 분할의 효과를 비교하였다. 실험 도구로는 펜티엄 1.2GHz, 메모리 256MB를 가진 PC에서 수행하였으며, 사용한 소프트웨어는 C언어를 이용하였다.

본 논문에서 제안하는 알고리즘은 Non-Strict 의미를 갖는 함수 언어를 대상으로 하므로 Haskell과 Id 같은 Non-Strict 의미를 갖는 모든 함수 언어에 대하여 적용할 수 있다. 하지만 Haskell과 Id 같은 함수 언어의 모든 특징을 갖는 복잡한 Non-Strict 함수 언어는 구현하기 어렵기 때문에 실험을 단순하게 하기 위하여 Non-Strict 함수 언어의 모든 특징을 갖는 Non-Strict 함수 언어대신 Non-Strict 함수 프로그램을 스레드로 분할할 때 고려되어야 할 특징을 사용하였다. 또한 I-구조를 이용하는 구조형 자료와 조건식, 함수의 재

귀 호출, 루프식 등의 제어 구조를 제공하며 Non-Strict 함수 언어의 하나인 패턴 부합을 조건식으로 표현한다. 추상 자료형이나 다형 시스템은 사용하지 않으며 모든 자료형이 번역시간에 결정되는 정적 자료형 검사를 사용한다.

실험 평가 대상으로는 Hoch와 Schauser 있으나, Hoch의 기법은 제어 벡터를 사용하여 조건식의 참실행식과 거짓실행식을 합병하고자 하는 것 이외에는 Schauser의 기법과 동일하다. 따라서 본 논문에서는 Schauser의 기법만을 고려하여 평가하였다.

Non-Strict 함수 언어로부터 스레드 생성에 있어서 본 논문이 제안한 방법과 기존의 기법간의 비교를 위해서 SUM, 이진 탐색 등의 벤치마크를 사용하여 성능평가를 실시하였다. 이들은 Non-Strict에서 조건식을 포함하고 있어서 조건식을 처리하는데 명백한 차이를 보이고 있어 스레드 개수, 스레드 길이, 동기화 계수기를 비교하는데 적합하였다. 사용된 SUM은 병렬 계산 방식으로 1부터 65,535 사이의 모든 정수를 더하는 프로그램이 수행되었으며, 이진 탐색은 10,000개의 원소에 대해서 수행되었다.

표 1은 두 기법간의 스레드의 특성을 비교하였다. 본 논문의 기법이 Schauser의 기법에 비해서 보다 적은 스레드의 개수와 동기화 계수기를 갖으며, 보다 긴 스레드 길이를 갖는다는 것을 보여주고 있다.

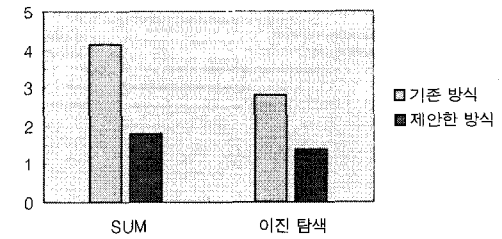
(표 1) 스레드 비교

		SUM	이진탐색
스레드 길이	기존 방식	5	7
	제안한 방식	7	12
동기화 계수기 (Time)	기존 방식	1.6	1.6
	제안한 방식	1.2	1.3

표 2는 벤치마크에 대해서 동적으로 측정된 명령어의 상대적 빈도수를 비교하였다. 표 2에서 동기화는 스레드간에 동기화를 수행하는 명령어, ALU는 산술, 논리 연산을 수행하는 명령어, 적재와 저장은 스레드 경계선 상에서 수행되는 프레임으로부터의 값의 적재와 저장을 수행하는 명령어, branch는 제어 분기를 수행하는 명령어의 그룹이다. SUM과 이진 탐색에 대해서 branch 명령어의 개수가 본 논문의 기법에서 각각 감소하고 있는 것은 스위치 결합 때문이다. 표 2는 SUM에서 두 기법간의 동기화 명령어 빈도수의 비율이 다른 벤치마크에 비해서 크다.

(표 2) 명령어 빈도수의 비교

		동기화	ALU	적재/저장	Branch
SUM	기존 방식	329649	131577	594906	197244
	제안 방식	2300	131577	461220	145241
이진탐색	기존 방식	278	184	673	151
	제안 방식	42	184	420	73



(그림 8) 명령어의 상대적 빈도수

그림 8은 두 기법간의 명령어의 상대적 빈도수를 보여주고 있다. 제어와 branch 명령어가 기존의 기법에서 더 많이 수행되는데, 이러한 명령어는 스레드 형성 기법에 크게 영향을 받는다. 그림 6에서 SUM에서 제어와 branch 명령어 빈도수의 비율 차이를 확인할 수 있다. 이는 표 2에서 동기화 명령어 빈도수의 관점에서 볼 수 있다. 그러므로 SUM에서 두 기법간의 동기화 명령어 빈도수의 비율은 크며, 동기화 명령어는 메시지에 의해 처리되기 때문에 다른 명령어에 비해서 많은 실행 시간이 필요하다.

5. 결론 및 향후 과제

Non-Strict 함수 프로그램의 스레드 분할은 동일한 스레드에 존재할 수 있는 연산들을 찾아 나가는 것으로 간접 종속이 존재하지 않는 연산을 찾아 하나의 스레드로 병합해 나가는 과정이었다. 기존의 분할 알고리즘은 조건식의 판단식, 참실행식, 거짓실행식을 기본 블록으로 나누고 각각에 대하여 지역 분할을 적용하기 때문에 작은 크기의 스레드가 많이 발생하므로 문맥 전환 횟수와 동기화가 많이 발생하였다.

본 논문에서는 Non-Strict 함수 프로그램에서 조건식의 세 가지 기본 블록을 하나 또는 두 개의 기본 블록으로 병합함으로서 다중스레드 모델에서 스레드의 문맥 전환 횟수와 동기화 횟수를 감소하여 프로그램의 수행 성능을 향상시키는 방법을 제안하여, 조건식의 스레드 분할 알고리즘의 정확성을 보였다. Non-Strict 함수 프로그램에서 스레드의 크기가 증가한다는 것은 수행시간에 스레드간의 문맥 전환의 횟수와 동적 동기화 횟수가 줄어든다는 것을 의미하므로 문맥 전환의 횟수의 감소는 수행 성능에 큰 영향을 미치는 요소 중 하나인 통신량과 동기화 비용의 감소를 뜻하므로 프로그램의 수행 성능이 향상되었다.

향후 연구 과제로는 Non-Strict 함수 프로그램의 수행 성능을 더욱 향상시키기 위해 조건식의 결합의 크기에 따른 성능의 차이가 있으므로 스레드 분할을 수행할 때 시작 노드를 선택하는 방법에 대한 연구가 요구된다.

참 고 문 헌

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

[2] G. Lindstrom, "Static Evaluation of Functional Programs," Symp. on Compiler Construction, SIGPLAN Notices Vol.21, No.7, pp.196-206, 1996.

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

[4] R. S Nikhil, "A Multithreaded Implementation of ID using P-RISc Graphs", Int'l Proc. of 6th Annual Workshop on Languages and Compilers for Parallel Computing, 1993.

[5] J. F. Anthony and G. H. Peter, Functional Programming, Addison-Wesley, 1987.

[6] E.H. Rho, H.H. Kim, D.J. Hwang, and S.Y. Han, "Effects of Data Bundling in Non-Strict Data Structures", Proc. of Parallel Architectures and Compilation Techniques '95, 1995, pp140-148

[7] K. E. Schauser, D. E. Culler, and T. von Eicken, "Compiler-Controlled Multithreading for Lenient Parallel Languages," 5th ACM Conf. on Functional Programming Languages and Computer Architecture, LNCS Vol.523, pp.50-72, 1991.

[8] 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.

[9] B. Shankar and A. P. W. Bohm, and W. A. Najjar, "Top-Down Thread Generation for Sisal," Proceedings of SISAL '93, 1993.

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

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

[12] Rober Jonathan Ennals, "Adaptive Evaluation of Non-Strict Programs," PhD thesis, University of Cambridge, 2004.

[13] Guang R. Gao, Lubomir Bic, Jean-Luc, "Advanced Topics in Dataflow Computing and Multithreading," IEEE Computer Society Pr, 1995

[14] Simon Thompson, "Haskell: The Craft of Functional Programming," Addison Wesley Publishing Company, 1999

◐ 저 자 소 개 ◑

조 선 문(Sun-Moon Jo)
1995년 충주대학교 컴퓨터공학과 졸업(학사)
2001년 인하대학교 대학원 전자계산공학과 졸업(석사)
2007년 인하대학교 대학원 컴퓨터정보공학과 박사
관심분야 : 병렬처리, 프로그래밍 언어, 보안, XML 접근제어
E-mail : sunmoon@pcu.ac.kr