Non-Strict 프로그램 조건식의 향상된 스레드 분할

조 선 문。, 양 창 모 *, 유 원 희
인하대학교 전자계산공학과
* 청주교육대학교
e-mail:g1991270@inhavision.inha.ac.kr

The Enhanced Thread Partitioning of Conditional Expressions of Non-Strict Programs

Sun-Moon Jo。·Chang-Mo Yang *·Weon-Hee Yoo
Department of Computer Science and Engineering, Inha University
* Chungju National University of Education

요약

다중스레드 병렬기계(multithreaded parallel machine)를 위하여 함수 프로그램을 번역할 때 스레드 분할이란 수행 순서를 번역시간에 알 수 있어 정적 스케줄링이 가능한 프로그램의 부분을 식별하여 스레드로 모으는 작업을 말한다.

조건식에서 연산의 수행 순서는 판단식 -> 참실행식 또는 판단식 -> 거짓실행식이므로 번역시간에는 수행순서를 결정할 수 없다. 따라서 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 등의 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.

1. 서론

일반적으로 비평가인자 함수 언어(non-strict functional language)는 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[4,5]. 그러나 이러한 비평가인자 함수 프로그램을 기존의 폰 노이만(von Neumann)형 병렬기계에서 수행할 때 비평가인자로 인하여 미세수준(fine grain)의 동적 스케줄링이나 동기화가 필요하여 기존의 병렬 기계에서의 구현이 어렵다[1,6]. 이러한 언어를 병렬기계를 위하여 번역할 때, 프로그램의 지역성을 살리기 위하여 프로그램을 순차적으로 분할(partitioning)하는 과정이 필요하다.

스레드 분할이란 수행 순서를 번역시간에 알 수 있어 정적 스케줄링이 가능한 프로그램의 부분을 식별하여 스레드로 모으는 작업이다[3,6,7]. 스레드 분할의 기본 원칙으로 병렬성의 극대화, 스레드 길이의 최대화, 동기화의 최소화, 교착상태 회피, 자원 활용도의 극대화 등이 있다[2]. 이런 측면에서 볼 때 스레드 분할은 병렬성, 동기화 비용, 그리고 순차 수행의 효율성간의 흥정(trade-off)을 의미한다. 병렬 기계에서 비평가인자 함수 언어를 구현할 때는 자료 전달, 동기화와 비평가인자 의미로 인하여 스레드의 크기가 제한을 받는다. 따라서 다중스레드 모델을 위하여 비평가인자 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[6,7].

반복 분할, 분리제약 분할, 분리집합 분할 등 기존의 분할 알고리즘은 조건식을 판단식, 참실행식, 거짓실행식을

* 본 논문의 연구는 1999년도 정보통신부 대학 기초연구지원사업에 의해 수행되었음.

* 본 논문의 연구는 1999년도 정보통신부 대학 기초연구지원사업에 의해 수행되었음.

기본 블록으로 나누고 각각에 대하여 지역 분할을 적용하기 때문에, 동기화 횟수와 스레드간의 문맥 전환 횟수가 커서, 처리비용이 많이 들어 실제 스레드 수행의 성능이 저하된다.

이와 같은 단점을 해결하기 위해 본 논문에서는 비평가인자 분석 방법을 이용하여 조건식의 세 가지 기본 블록을 하나 또는 두 개의 기본 블록으로 병합함으로서 스레드 분할을 향상시키는 방법을 제안한다.

2. 스레드 분할 제약조건

다중스레드 모델을 위하여 비평가인자 프로그램을 스레드로 분할할 때 고려하여야 하는 조건에 대하여 설명한다. 스레드를 정의하면 다음과 같다[6,8].

1) 어떤 문맥에서도 스레드 내에서 명령의 올바른 수행순서를 번역시간에 결정할 수 있어야 한다.
2) 한번 스레드의 첫 명령이 수행되면 나머지 명령들은 번역시간에 결정된 순서대로 중단없이 수행되어야 한다.

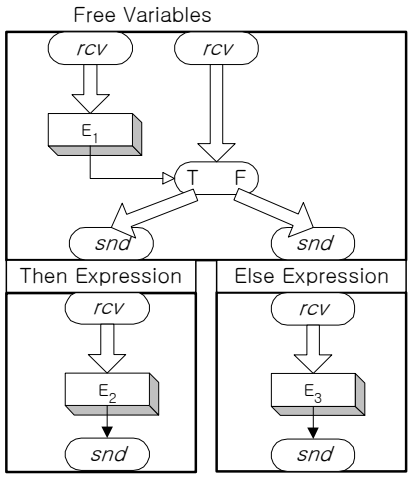
비평가인자 프로그램에서 연산은 주어진 문맥에 따라 수행 순서가 달라질 수 있고, 이 연산들은 수행시간에 동적으로 스케줄링되어야 하므로 스레드의 첫번째 조건으로 인하여 여러 개의 스레드가 만들어질 수 있다. 동적으로 스케줄링되어야 하는 연산들을 식별하기 위해서는 연산들의 종속 관계를 구하여야 한다. 종속 관계는 스레드내의 연산 사이에 존재하는 직접 종속과 서로 다른 스레드에 존재하는 연산간에 존재하는 간접 종속으로 나눌 수 있다. 또한 번역시간에 파악할 수 있는 상시 종속과 비평가인자 의미로 인하여 번역시간에 확정할 수 없고 수행 시간에 결정되는 잠재 종속이 있다[3,7]. 간접 종속에는 동기화가 필요한 I-구조의 I-load 연산이나 함수 호출연산과 요구 결과를 사용하는 연산사이에 발생하는 상시 간접종속과 non-strictness로 인하여 특정 문맥에서만 발생하는 잠재 간접 종속이 있다.

3. 조건식의 스레드 분할

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

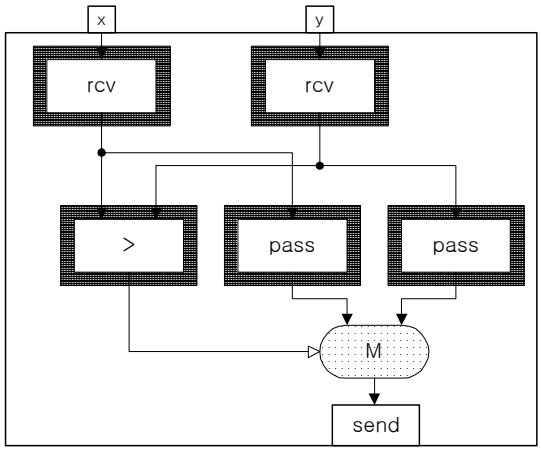
if E_1 then E_2 else E_3

조건식의 데이터플로우 그래프는 [그림 1]과 같다.



[그림 1] 조건식의 데이터플로우 그래프

다음과 같은 최대 값을 구하는 함수 max를 보자.



[그림 2] 함수 max의 스레드 분할

func max(x, y) = if x > y then x else 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 2]와 같이 세 개의 스레드로 분할된다.

함수 max에 대하여 판단식, 참실행식, 거짓실행식을 구성하는 세 개의 기본 블록을 하나의 기본 블록으로 가정하고 분리 집합 분할[9]을 적용하면 세 연산은 간접 종속을 갖지 않으므로 하나의 스레드로 분할할 수 있다.

이와 같은 방법을 사용하려면 앞서 제시한 스레드의 정의 2번을 약간 수정하여 스레드 내에서의 분기를 허용하여야 한다.

스레드내에서의 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 없고, 판단식, 참실행식,

거짓실행식을 구성하는 기본 블록사이에 간접 종속이 존재하지 않는 경우에만 병합할 수 있다.

```
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 3] 기본 블록을 병합할 수 있는 조건식의 예

[그림 3]의 ①와 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.

기본 블록간에 간접 종속이 존재하는가를 판단하는 방법은 사용하는 기존의 스레드 분할 방법을 이용한다. 기본 블록간에 간접 종속이 없다면 이들을 병합하고 지역 스레드 분할을 수행한다.

이와 같은 방법으로 조건식을 스레드 분할하는 알고리즘은 다음과 같다. 다음 알고리즘에서 *G(E)*는 식 *E*의 데이터플로우 그래프를 의미하고, *T(E)*는 *E*의 스레드 집합을 의미한다.

[알고리즘] 조건식의 스레드 분할 알고리즘

입력 : G(if E_1 then E_2 else E_3)

출력 : T(if E_1 then E_2 else E_3)

1. $G(E_1)$, $G(E_2)$, $G(E_3)$의 참여집합과 종속 집합을 구한다.
2. $G(E_1)$, $G(E_2)$, $G(E_3)$의 간접 종속 관계를 판단한다.
   - 2.1 $G(E_1)$, $G(E_2)$, $G(E_3)$가 간접 종속을 갖지 않는다면, $G(E_1)$, $G(E_2)$, $G(E_3)$을 병합하여 하나의 기본 블록으로 나타낸다.
   - 2.2 $G(E_1)$, $G(E_2)$가 간접 종속을 갖지 않는다면, $G(E_1)$, $G(E_2)$를 병합하여 하나의 기본 블록으로 나타내고, $G(E_3)$를 하나의 기본 블록으로 나타낸다.
   - 2.3 $G(E_1)$, $G(E_3)$가 간접 종속을 갖지 않는다면, $G(E_1)$, $G(E_3)$를 병합하여 하나의 기본 블록으로 나타내고, $G(E_2)$를 하나의 기본 블록으로 나타낸다.
   - 2.4 $G(E_2)$, $G(E_3)$가 간접 종속을 갖지 않는다면, $G(E_2)$, $G(E_3)$를 병합하여 하나의 기본 블록으로 나타내고, $G(E_1)$을 하나의 기본 블록으로 나타낸다.
3. 각 기본 블록에 지역 스레드 분할을 적용하여 T(if E_1 then E_2 else E_3)를 구한다.

4. 평가 및 결론

비평가함수 언어로부터 스레드 생성에 있어서 본 논문이 제안한 방법과 기존의 분리 집합 분할 방법을 비교하기 위해 정수 덧셈과 피보나치 탐색 등의 비교를 통하여 성능평가를 실시하였다. 이들은 비평가인자에서 조건식을 포함하고 있어서 조건식을 처리하는데 명백한 차이를 보이고 있어서 스레드 길이와 스레드 개수를 비교하는데 적합하다.

[표 1]은 조건식에서 두 방법간의 스레드 분할 특성을 비교하였다. 본 논문의 제안한 기법이 분리 집합 분할 방법에 비해서 보다 긴 스레드 길이와 보다 적은 스레드 개수를 갖는다는 것을 보여주고 있다. 분리 집합 분할을 이용할 때는 스레드 제약 조건으로 인하여 작은 스레드가 많이 발생하였다.

본 논문에서는 비평가인자 프로그램을 분석하여 조건식의 스레드 분할 방법에서 스레드 분할 제약 조건 때문에 병합할 수 없는 스레드를 더 큰 스레드로 분할하는 알고리즘을 제안하여 스레드의 크기를 크게 만들어 비평가인자 함수에서 수행 성능을 향상시킴으로서 처리비용의 절약되었다.

[표 1] 스레드 특성 비교

		정수 덧셈	피보나치 탐색
스레드 길이	기존 방식	5.4	6
	제안한 방식	7.1	10
스레드 개수	기존 방식	25	16
	제안한 방식	13	9

참 고 문 헌

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

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

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

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

[5] J. Farbrian and S. C. Wray, "Code Generation Techniques for Functional Languages," ACM Conf. on Lisp and Functional Programming, pp.94-104, 1986.

[6] 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 Symp. on Principles

of Programming Language, pp.259-271, 1995.
[7] 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.
[8] K. R. Traub, Implementation of Non-strict Functional Programming Languages, Research Monographs in Parallel and Distributed Computing, MIT Press, 1991.
[9] 양창모, 유원희, “분리집합을 이용한 Nonstrict 프로그램의 스레드 분할”, 한국정보과학회 논문지, 제 22 권 제 8호, pp.891-899, 1997년 8월.