동기화 명령을 가지는 내포 병렬 루프 프로그램의 수행중 접근이상 탐지를 위한 레이블링

배 상 현, 전 용 기, 배 종 민

(한국통신 서울통신운용연구단, 경상대 전자계산학과, 경상대 전자계산학과)

A Labeling for on-the-fly Detection of Access Anomalies in Nested Parallel Loop Programs with Synchronization Operations

Sang-Hyun Bae, Yong-Kee Jun, Jong-Min Bae

(Korea Telecom, GyeongSang National Univ., GyeongSang National Univ.)

요 약 공유 메모리 병렬 프로그램의 주요 문제의 하나는 공유 변수에 접근하는 비 결정적 수행이다. 본 연구에서는 공유 메모리 병렬 프로그램의 접근이상(access anomaly)을 탐지하는 방법들중 수행중 탐지 기법을 보인다. 수행중 접근이상 탐지 기법은 접근이상이 존재하면 적어도 하나는 탐지할 수 있는 장점을 가지고 있다. 수행중 탐지 기법인 English-Hebrew Labeling은 동기화 명령을 가지는 내포 병렬 루프 프로그램에서 적용될 수 있는 레이블링 기법으로 레이블링에 많은 저장장소를 필요로 하는 단점을 가지고 있었다. 본 연구에서는 새로운 레이블링 방법을 소개하고, 기존의 English-Hebrew Labeling과 최악의 경우에 기억 장소 복잡도의 측면과 시간 복잡도의 측면에서 효율성을 비교, 분석하게 된다.

1. 서 론

본 연구에서 대상으로 하는 병렬프로그램은 병렬처리의 단위로 루프구조가 일반적이다. 이런 병렬프로그램은 동일한 입력에 대해서 매 수행시마다 다른 결과를 나타낼 수 있는 비결정적(nondeterministic) 수행이 생길 수 있는데, 이런 의도되지 않은 비결정적 수행의 원인은 공유 변수에 대해서 적어도 하나의 쓰기가 있는 접근들이 서로간에 보장된 순서 없이 발생하기 때문이다. 이를 자료경쟁(data race) 또는 접근이상(access anomaly)[2, 4]라고 하는데, 병렬프로그램을 코딩 및 디버깅하는데 어려움을 주는 요소이다.[7]

병렬 프로그램에서 접근이상을 탐지하는 기법으로는, 탐지 정보를 수집하는 시기에 따라서, 정적 분석(static analysis)[8], 사후 분석(post-morterm analysis)[9], 수행중 탐지(on-the-fly detection)[2] 기법이 있다.

본 연구에서는 동기화가 나타나는 비내포 및 내포 병렬 루프 프로그램에서 수행중 탐지 기법을 사용하여, 접근이상 탐지를 위한 효율적인 병행성 정보 부여를 위한 레이블링 기법으로 English-Hebrew Labeling[3,4]을 간단히 소개하고, 이를 개선한 새로운 레이블링 기법으로 Reduced English-Hebrew Labeling을 소개한다. Task Recycling[4]은 English-Hebrew(EH) Labeling보다 병렬명령(parallel operation)에서 병렬성 정보를 유지하는데 효과적이지 못하다. 하지만, EH 레이블은 많은 저장장소를 요구하기 때문에 수행이나 공간면에서 Task Recycling 보다 못하다[4]. 따라서, 본 연구는 EH 레이블링을 개선하여 필요로 하는 저장장소와 수행시간을 줄였다.

2. 연구 배경

병렬 프로그램의 수행중 이상탐지를 위해 각 공유변수에 대한 접근역사(access history)를 이용한다. 각 접근역사 내에는 해당 공유변수를 접근한 명령들이 소속한 각 블록의 병행성 정보가 수행 중에 저장된다. 이러한 각 블록들의 병행성 정보를 제공하기 위해서는 각 블록마다 수행중에 레이블을 부여해야 한다. 병행 관계를 검사하기 위해 부여되는 레이블링 기법으로 English-Hebrew Labeling, Task Recycling, Offset-Span Labeling[5], 그리고 Nest Region Labeling[2] 등이 있다.

EH 레이블링에서 테그(tag)[4]는 English(E) 레이블과 Hebrew(H) 레이블로 쌍으로 이루어져 있고, 개념적으로 E 레이블은 POEG[4,10]에서 왼쪽으로부터 오른쪽(left-to-right)방향 순서의 수로 만들어진다. 이 레이블은 그래프 전체를 거치지 않고, 즉시 실시간으로 발생되어져야 한다. 이 레이블은 수들의 스트링이며 사전식 순서(lexicographical order)를 가진다. 부모블록 p를 가진 doall

에서 자손블록 $c_0 \ldots c_m$는 E 레이블로

doall: $E(tag_c) \leftarrow E(tag_p) \mid 1$ (1)

이 되며, 이기서 | 는 추가 연산자이다.

마찬가지로, 부모 p의 레이블은 동기화 명령(coordination operation)에 의해 만들이진 자손블록 c에서 E 레이블로

coordination: $E(tag_c) \leftarrow E(tag_p) \mid 1$ (2)

이 되며, 부모 $p_0 \ldots p_m$을 가진 endall 에서 자손블록 c의 E 레이블은 다음과 같이 된다.

endall: $E(tag_c) \leftarrow \max(E(tag_{p_i}))$ (3)

H 레이블의 경우 오른쪽에서 외쪽(right-to-left) 순서로 대칭적으로 만들어진다. 한가지 다른 차이는 doall 명령의 i 번 째 자손은 $j + 1 - i$를 가진다

EH 레이블링에서 레이블의 길이는 doall과 동기화 명령의 수에 의해 늘어난다. 즉, 식(3)에 의해 합류전인 $E(tag_{p_i})$와 비교해서 최대값(max)을 취하게 되므로, 식(1)의 생성시 늘어난 길이가 줄어들지 않는다. 동기화를 나타내는 식(2)에서도 레이블 길이는 늘어나서 과도한 저장장소를 소요로 한다[4].

3. Reduced English-Hebrew Labeling(REH)

EH의 저장장소를 줄이기 위한 REH 레이블링은 fork시 식(1)과 같고, 합류시에는 새로운 삭제 연산자 $\ll$ 을 사용하며, 식(4)과 같이 부모 스래드 $p_0, , , p_m$의 E 레이블들 중에서 가장 큰 $E(tag_{p_i})$이 합류시에 스트링의 마지막 수를 삭제한 새로운 태그의 마지막 수에 1를 증가시킨다.

endall: $E(tag_c) \leftarrow max\,(E(tag_{p_i})) \ll i + 1$ (4)

예를 들면, REH 레이블링을 나타내는 그림 1에서 블록 B_3, B_4, B_5의 E 레이블 최대값인 113은 합류 후에 마지막 수인 3이 없이진 태그 11의 마지막 수에 대해 1이 더해져서 12가 되었다 이 식은 합류전의 레이블인 11의 길이와 등일함을 의미한다.

동기화 명령에서 부모 블록 p의 태그는 동기화 명령 후에 블록 c의 레이블로는 다음과 같이 된다

coordination: $E(tag_c) \leftarrow E(tag_p) + 1$ (5)

예를 들면, 그림 1에서 B_6의 <12,13>은 동기화 명령에서 마지막 수에 각각 1이 더해져서 <13,14>가 되었다.

REH 레이블링에서는 태그가 정수형 스트링으로 저장된다고 가정할 때, 식 $L(E(tag_{i_K}) - E(tag_{i_k}))$는 tag_j와 tag_i에서 각각 K 번째 수를 마이너스하는 식이며, tag_j와 tag_i 의 길이가 다를 경우 길이가 짧은 tag의 길이까지만 계산이 된다.

tag_i 와 tag_j 가 다음 두 조건 중 하나를 만족하면 순차이다

$E(tag_i) = E(tag_j)$ (6)

$E(tag_i) < E(tag_j)$ and $H(tag_i) < H(tag_j)$,

$L(E(tag_{j_i}) - E(tag_{i_i})) = L(H(tag_{j_i}) - H(tag_{i_i}))$ (7)

예를들면, 그림 1에서 블록 B_{10}의 <13,14>와 블록 B_9의 <123,111>과의 비교는 식 (7)에 의해 13 > 123 과 14 > 111이고, E 레이블에서 최초의 다른 수인 두 번째 수 3과 2는 3 - 2 = 1 이고, H 레이블에서 두 번째 4와 1은 4 - 1 = 3이 되어서 1과 3으로 두 값이 같지 않기 때문에 병행 수행함을 알 수 있다. 블록 B_{10}의 <13,14>와 B_2의 <12,11>과의 관계는 병행수행이지만 <13,14>의 동기화 리스트 포인터 값이 B_7이기 때문에 <121,113>과 <12,11>의 비교는 순차이기 때문에 순차 수행된다.

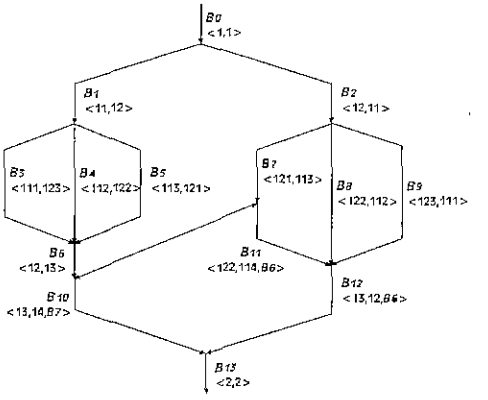

그림 1 Reduced English-Hebrew Labeling

4. 분 석

본 장에서는, 기존의 기법인 EH 레이블링과 새로 제시된 REH 레이블링에 의한 수행중 이상탐지 기법의 효율성을 비교, 분석한다 수행중 감시 기법의 효율성은 오류를 수정할 병렬 프로그램에 대해 Dinning과 Schonberg의 표시법[6]으로 다음의 요소들에 의존한다

V 감시하는 최대 공유변수의 수

T: 병렬 프로그램의 논리적 최대병렬성

N: fork-join의 최대 네포깊이

S: 순서화된 최대 동기화 수

F: 순서화된 쓰레드들의 최대 생성회수

수행중 감시 기법의 효율성을 분석할 적절한 척도는 기억장소 복잡도의 측면과 시간 복잡도의 측면이다. 기억장소 복잡도 측면은 접근역사의 크기와 수행블럭들에 대해 유지하는 레이블의 크기에 의존한다. 시간 복잡도 측면은 레이

블 생성과 접근사건 발생시 병행성 검사와 접근역사 유지를 위한 시간에 의존한다. 기존의 기법들에 대한 효율성은 기존 연구[1, 2, 5]에 잘 분석되어 있다.

만약 변수가 각 쓰레드에서 억세스되면 T 쓰레드 이름만큼의 수의 레이블들을 가진 접근 역사가 있다 따라서 기억장소 복잡도는 VT가 되며, 어떤 쓰레드에 대한 EH 레이블의 크기는 N의 fork-join의 내포 깊이에 비례한다. 하지만 레이블은 join에서 줄어들지 않고 fork가 생길 때마다 늘어나므로 최악의 경우 F 길이가 더 늘어나고, 동기화 명령에서도 최악의 경우 S가 더 늘어나서, 결국 N 보다 더 큰 (N+S+F) 가 된다. 그러나, REH 레이블인 경우 최악의 레이블 길이는 N이 된다 만약 접근 역사가 레이블에 대한 포인터로 저장될 경우 각 레이블은 길이 N을 가지고 있고, 각 레이블들에 대한 VT의 독특한 포인터 일 수 있다 따라서, Garbage Collection이 사용될 경우 최대 공간은 O(VTN)가 된다.

REH 레이블인 경우, 접근 이상을 포함하는 변수에 각각 접근하는지 증명하기 위한 최악의 시간은 O(TN)가 필요하다. 왜냐하면, 변수의 접근 역사에서 T번 비교되고, 각각의 비교는 O(N) 시간이 걸리기 때문이다. 물론 EH 레이블인 경우 레이블 길이가 내포깊이 보다 커지므로 N보다 훨씬 많은 (N+S+F)가 된다.

따라서, 기억장소 복잡도의 측면과 시간 복잡도의 측면에서 본 EH 레이블링 기법과 REH 레이블링 기법의 효율성을 비교한 것은 (표 1)과 같다 이것은 REH 레이블링 기법이 훨씬 적은 기억장소와 접근시간을 갖기 때문에 효율적이다

표 1. 최악의 시간과 저장장소의 요구 비교

Labeling		EH Labeling	REH Labeling
Space		O(VT(N+S+F))	O(VTN)
Time	Thread Creation & Termination	O(T)	O(T)
	Per-access	O(T(N+S+F))	O(TN)

5. 결 론

공유변수를 갖는 다중 처리 시스템에서 병렬 프로그램은 공유변수에 병렬로 접근하는 블록들 간에 발생하는 이상들 때문에 비결정적인 수행을 초래할 수 있으므로 순차 프로그램에 비해 오류 수정 작업이 어렵다.

레이블은 공유변수에 대한 접근이 발생할 때 병행성 검사를 위한 정보이다. 레이블링 기법의 효율성은 레이블을 위한 시간적인 부담과 공간적인 부담에 의존적이다. EH 레이블은 동기화 명령이 있는 내포 병렬 루프 프로그램에 적용될 수 있는 효과적인 기법이다. EH 레이블이 이점을 가지고 있음에도 불구하고, 레이블링에 기억장소를 많이 차지하는 단점을 가지고 있었다. 본 연구에서는 이런 단점을 개선한 효율적인 REH 레이블링을 소개하였다. 특히, 이 레이블링은 실제 컴퓨터 시스템에서 쉽게 구현될 수 있는 구조를 가지고 있다.

병렬 프로그램의 오류 수정을 위한 향후 과제는 효율적이고 일반적인 구조에 적용될 수 있는 레이블링 기법에 관한 연구와 실용적인 측면에서의 접근이상 탐지를 위한 연구가 필요하다.

참 고 문 헌

[1] 김혜정, 전용기, 고건, "임의적 동기화를 가지는 비내포 병렬 루프의 접근이상을 위한 효율적 탐지," 한국 정보 과학회 학술 발표 논문집, 20(1) 693-696, April 1993.

[2] Jun, Y., and K. Koh, "On-the-fly Detection of Access Anomalies in Nested Parallel Loops," 3rd Workshop on Parallel and Distributed Debugging, pp. 107-117, ACM, May 1993

[3] Nudler, I, and L. Rudolph. "Tools for the Efficient Development of Efficient Parallel Programs," 1st Israeli Conference on Computer System Engineering, IEEE, 1988.

[4] Dinning, A., and E. Schonberg, "An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection," 2nd Symp. on Principles and Practice of Parallel Programming, pp. 1-10, ACM, March 1990.

[5] Mellor-Crummey, J M, "On-the-fly Detection of Data Races for Programs with Nested Fork-Join Parallelism," Supercomputing '91, pp. 24-33, ACM/IEEE, Nov. 1991.

[6] Dinning, A, and E. Schonberg, "An Evaluation of Monitoring Algorithms for Access Anomaly Detection," Ultracomputer Note 168, Courant Institute, New York University. July 1989

[7] McDowell, C E., and D. P. Helmbold, "Debugging Concurrent Programs," Computing Surveys, 21(4) 593-622, ACM, Dec. 1989.

[8] Emrath, P A., and D. A. Padua, "Automatic Detection of Nondeterminacy in Parallel Programs," 1st Workshop on Parallel and Distributed Debugging, pp 89-99, ACM, May 1988.

[9] Emrath, P A, S. Ghosh, and D. A. Padua, "Detecting Nondeterminacy in Parallel Programs," Software, 9(1): 69-77, IEEE, Jan. 1992.

[10] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, 21(7). 558-565, ACM, July 1978.