DOI: 10.3745/KIPSTA.2010.17A.2.073

멀티태스킹에 의한 병행 점진 평가 방법

한 정 란†

요 약

하드웨어의 성능이 향상됨에 따라 멀티태스킹 방식으로 병행 처리하는 연구가 다양하게 진행되고 있다. 프로그램 개발 단계에서 프로그램을 수정할 경우, 전체 프로그램을 다시 평가하는 대신 수정한 부분과 그 부분에 영향 받는 부분만을 다시 평가하는 방법이 점진 평가인데, 여러 프로세서에서 병렬로 처리하는 대신 자바언어의 멀티쓰레딩 기능을 활용하여 점진 평가의 효율성을 증대시키는 연구가 필요하다.

본 논문에서는 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하여 객체 지향언어인 자바 언어에서 병행 점진 평가를 동시에 효율적으로 수행할 수 있는 병행 점진 평가 알고리즘을 제시하고 그 알고리즘의 정확성을 증명한다. 실험을 통해 병행 점진 평가 방법의 효율성을 분석한다.

키워드 : 병행 점진 평가, 종속 차트, 병행 점진 평가 알고리즘, 멀티태스킹, 멀티쓰레딩

A Concurrent Incremental Evaluation Technique Using Multitasking

Han, junglan†

ABSTRACT

As the power of hardware has improved, there have been numerous researches in processing concurrently using multitasking method. The incremental evaluation is the evaluation method of reevaluating only affected parts instead of reevaluating overall program when the program has been changed. It is necessary to do more studies that improve the efficiency of concurrent incremental evaluation to do multitasking using multi-threading of Java not to do in parallel using multiprocessor.

In this paper, the dependency in the dependency chart is based on the attribute that describes the real value of the variable that directly affects the semantics, thereby doing efficient evaluation. So using the dependency, this paper presents the concurrent incremental evaluation algorithm for Java Languages and proves its correctness, analyzing the efficiency of concurrent incremental evaluation by the simulation.

Keywords : Concurrent Incremental Evaluation, Dependency Chart, Concurrent Incremental Evaluation Algorithm, multitasking, Multi-Threading

1. 서 론

프로그램 개발 단계에서 프로그램의 일부분이 수정될 때마다 전체 프로그램을 다시 평가하는 것은 프로그램 개발을 위해 소요되는 시간과 공간적인 측면에서 비효율적이다. 점진 평가는 전체 프로그램을 다시 평가하는 대신 수정한 부분과 그 부분에 영향 받는 부분만을 다시 평가하는 방법이다. 프로그램 개발 환경의 실행 효율성 측면에서 고려해 볼 때 점진 평가 방법이 효율적인 평가 방법이라 할 수 있다. 점진 평가의 효율성을 증대시키기 위해, 여러 프로세서에서 병렬로 처리하는 대신 하드웨어 성능의 향상으로 멀티태스킹 방식에 의해 멀티쓰레딩 기능을 활용하는 연구가 필요하다. 자바의 멀티쓰레딩 기능을 활용하면 병행(concurrency) 점진 평가를 쉽게 수행할 수 있다.

점진 평가를 수행하기 위해, 수정된 부분과 그 부분에 영향을 받아 변경되는 부분을 찾아내어 이 부분만을 다시 평가해야하고 이를 위해 점진 속성 평가 방법[1, 2]을 사용한다. 프로그램에서 실행되는 결과는 변수의 값을 계산하고 저장하여 출력하는 것이고 변수의 값을 나타내는 속성에 대한 종속성(dependency)만 관리하면 훨씬 효율적인 점진 평가를 수행할 수 있다[2]. 반면, 기존의 병행 점진 속성 평가 방법에서 사용된 종속 그래프(dependency graph)의 경우 속성 문법(attribute grammar)에 기반을 두어 프로그램의 의미 구조를 나타내기 위해 프로그램에 소속된 모든 속성에

† 종신회원 : 협성대학교 경영정보학과 부교수
논문접수 : 2009년 10월 27일
수 정 일 : 1차 2010년 2월 25일
심사완료 : 2010년 3월 22일

대해 종속성을 나타내어 종속 그래프가 상당히 복잡하고 수정이 일어날 때마다 모든 속성에 대해 변화를 파급(propagation)시켜야 하므로 비효율적이라 할 수 있다. 객체의 특성에 맞게 동시에 적절하게 평가하는 병행 점진 평가 방법에 대한 연구가 필요하고 자바 언어에 대해 병행 점진 평가를 효율적으로 수행할 수 있는 방법이 제시되어야 하는데 이러한 연구들은 미미한 실정이고 동적 의미를 형식적으로 명세한 연구들이 진행되고 있다[2-4].

본 논문에서는 종속 차트(dependency chart)[1, 2]를 사용하여 객체 지향언어인 자바언어에서 병행 점진 평가를 동시에 수행할 수 있는 알고리즘을 제시한다. 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하고 병행 점진 평가를 수행하기 위해 필요한 변화를 추적하는 과정이 효율적으로 병행 수행된다. 특히, 자바의 멀티쓰레딩 기능을 활용하여 멀티태스킹 방식으로 병행 점진 평가를 수행한다. 2장에서는 관련 연구들을 기술하고 3장에서는 병행 점진 평가를 위해 멀티태스킹이 가능한 경우를 분석한다. 4장에서는 동시에 점진 평가를 수행하는 병행 점진 평가 알고리즘을 제시하고 알고리즘의 정확성을 증명한다. 5장에서는 실험을 통해 제시된 병행 점진 평가의 효율성을 분석한다.

2. 관련 연구

점진 평가를 수행하는 많은 연구들이 진행되었고 최근에 발표된 점진 평가 방법에는 복잡한 변화 파급 과정을 단순하게 하기 위해서 동적 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성들을 중심으로 속성들 간의 종속성을 나타내어 종속 차트를 작성하고 작성된 차트와 속성 값을 고려하여 효과적인 평가를 수행했다[2].

여러 프로세서에서 병렬적(parallel)으로 수행하는 기존의 연구[7-9]들이 있는데 속성을 병렬적으로 평가하는 기존의 방법은 종속 그래프를 구성하는 동적(dynamic) 평가 방법을 사용하여 속성의 평가를 쉽게 병렬적으로 수행한다.

Henk[7]의 방법에서는 속성 평가자를 정적이고 동적인 방법을 결합하여 병행의 점진적인 평가자를 설계하여 크기가 동적으로 변하는 종속 그래프인 모델을 사용하여 값이 변경된 속성에 대한 변화를 이웃한 속성에 전달하는 변화 파급 과정을 수행하게 된다.

Hans와 Willy[8]의 경우 종속 그래프를 구성하지 않는 정적(static) 평가와 병렬 평가가 쉬운 동적 평가를 결합하여 각 평가의 장점을 취하고 트리를 일종의 부트리인 작은 영역들로 나누어서 각 영역별로 병렬적으로 종속 그래프를 구성해서 평가하고 있다.

Matthijs[9]의 방법에서는 병렬 속성 평가자(Attribute Evaluator)를 구성하는 여러 가지 방법을 제시하여 병행 의미 분석(concurrent semantic analysis)과 병렬 속성 평가에 대한 다른 연구들을 일반화시킨 방법을 사용하고 속성 계산을 여러 프로세서에 분산시키는 다양한 방법을 소개하고 있다.

Fábio Pasini[10]는 병행 시스템을 잘 표현할 수 있는 형식적이고 시각적인 언어인 Object-Based Graph Grammars (OBGG)을 사용하여 병렬 애플리케이션을 명세하였다. 샘플 병렬 애플리케이션을 OBGG로 모델링하고 이 모델의 기능적 특성을 모델 체킹으로 증명하였다.

그 외, 컴파일 방법으로 구현된 점진 시스템들 중 ALOE 에디터[11]나 Gandalf 프로젝트[12]에서는 언어의 추상 구문과 관련하여 액션 루틴을 작성하여 점진 컴파일러를 구현하는데 구문 구동(syntax-directed) 에디터에서 구문 트리를 경유하면서 변화가 생기면 관련된 노드에 대한 액션 루틴을 호출하여 변경된 부분을 다시 평가하게 된다.

기존의 연구들에서는 객체의 특성에 맞게 동시에 적절하게 평가하는 병행 점진 평가 방법을 수행한 연구들이 없었고 특히, 자바 언어에 대해 병행 점진 평가를 효율적으로 수행할 수 있는 방법에 대한 연구가 필요하다. 기존의 연구에서는 수정된 부분과 그 부분에 의해 영향받는 부분을 찾아내기 위해 변화를 파급시키는 변화 파급 과정이 필요하지만 본 연구에서는 속성들 간에 종속성을 나타내기 위해 종속 차트[1, 2]를 사용하여 변화를 파급시키는 과정이 없이 영향 받는 부분을 신속하게 추적할 수 있다. 본 논문에서는 변수의 값을 나타내는 속성을 중심으로 속성간의 종속성을 고려하여 객체 지향 언어인 자바를 점진 평가 하는데 여러 프로세서로 분산하여 평가하는 방법대신 자바의 멀티쓰레딩 기능을 활용하여 한 프로세서에서 멀티태스킹 방식으로 병행 점진 평가하는 방법을 제시하고자 한다.

3. 멀티태스킹 분석

병행 점진 평가를 수행하려면 종속 차트를 사용하는데, 종속 차트는 종속성을 나타내기 위한 유향(directed) 그래프로 소속클래스, 객체 리스트, 소속메서드, CON, FLAG, 종속링크로 구성된다[2]. 각 변수의 종속성을 나타내기 위해 선언된 소속클래스와 특정 클래스형을 갖는 객체들의 리스트를 나타내고 그 변수가 속한 소속메서드이름을 표시하고, 영향을 주는 변수를 종속링크로 연결한다[2]. CON은 조건속성을 나타내고 반복문의 조건식에 있는 속성이나 조건문의 조건식에 있는 속성이고, 조건속성의 값은 FLAG로 표시하는데 조건속성 값이 참인 경우 T, 거짓인 경우 F 로 표시한다[2].

프로그램에서 사용된 변수들의 값의 변화를 추적하여 계산하고 그 값을 출력하는 과정을 거치면 프로그램에서 올바른 실행 결과를 얻을 수 있다. 기존의 점진 평가[1, 2]에서는 변수 값을 수정하는 명령문이 나오면 그 변수 값에 영향받아 변하게 될 변수들을 종속차트의 종속링크를 통해 그 변수에 영향받는 변수들을 추적하여 영향받는 명령문들을 다시 평가한다. 따라서, 변수 값이 변할 때 종속링크를 찾아 변수 값이 변하는 명령문을 다시 실행하여 점진 평가를 수행하게 된다.

하드웨어 성능의 향상으로 멀티태스킹을 활용하면 점진 평가를 좀 더 효율적으로 수행할 수 있다. 효율적인 병행 점진 평가를 수행하기 위해, 점진 평가를 수행하는 과정 중 멀티태스킹이 가능한 부분을 찾아 분석하는 과정이 필요하다. (그림 1)의 자바 프로그램을 통해 멀티태스킹이 가능한 경우를 분석하는데 종속 차트 중 종속 링크부분만을 표시하여 분석한다.

Multithread 객체 thr1과 thr2 는 객체 속성 변수로 a, b, c를 갖고 지역변수로 x, y, z를 갖고 있다. 각 변수의 종속성을 표현한 종속 링크만을 갖는 종속차트는 (그림 2)와 같다.

(그림 1)의 ⑥번 명령문이 먼저 실행된 후 ⑦번과 ⑧번 명령문이 실행되므로 (그림 2)와 같은 종속 링크가 만들어지게 된다. (그림 2)에서 볼 수 있듯이 두 개의 링크 그래프로 구성되고 각 링크 그래프에 소속된 변수 값이 변경될 때 각 변수들을 평가하는 과정은 멀티태스킹이 가능하다. 두 링크 그래프 안에 소속된 변수들은 종속성이 없어 동시에 작업해도 오류없이 정확한 값을 갖게 된다. 왼쪽 그래프의 경우에도 c의 상위 정점(vertex)에 있는 두 부분그래프에 소속된 변수들의 값이 변하는 경우에도 멀티태스킹이 가능함을 알 수 있다. a와 x의 경우 각 변수에 대해 동시에 병행 점진 평가를 수행해도 각 변수의 값에 영향을 주지 않기 때문이다. a와 y의 경우도 병행 점진 평가가 가능한 변수들이다.

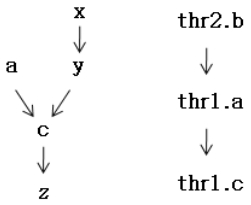
(그림 3)에서처럼 일반화를 시켜 멀티태스킹 가능한 경우를 분석하면 다음과 같다.

(그림 3)에서 ①과 ②의 두 부분그래프가 있고 각 그래프에 소속된 변수의 경우 멀티태스킹이 가능하고 각 그래프 안에 ③, ④, ⑤, ⑥ 부분 안에 속한 변수의 경우 멀티태스킹이 가능하다. 점진 평가할 때 멀티태스킹에 대한 정보가 필요하므로 종속차트를 만들 때 멀티태스킹 정보를 추가한다.

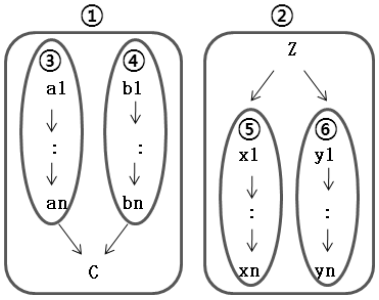
멀티태스킹이 가능한 지 효과적으로 검사하기 위해서, 종

```
public class Multithread
 {int a = 5, b = 10, c=1;       --- ①
   ...
  void example() {
   int x = 3, y, z;              --- ②
   y = x + 10;                    --- ③
   if (a < 5) c = y * a;         --- ④
   z = c * 5;                     --- ⑤
    ...
  }
  ...
  public static void main(String args[])
  { Multithread thr1, thr2;
    thr1 = new Multithread ();
    thr2 = new Multithread();
    ...
    thr1.example();              --- ⑥
    thr1.a = thr2.b * 2;         --- ⑦
    thr1.c = thr1.a;             --- ⑧
    ...
  } }
```

(그림 1) 자바 프로그램



(그림 2) 종속차트의 종속링크



(그림 3) 멀티태스킹 종속 차트

속차트를 구성할 때 추가의 정보를 넣어야 한다. (그림 3)에서처럼 같은 ③, ④, ⑤, ⑥ 부분이나 ①과 ② 영역 안에 속해 있는 정점들에게는 동일한 이름의 멀티태스킹 이름을 부여하는 방법을 사용하여 멀티태스킹 가능성을 표시한다. a1 정점처럼 상위 정점이 없이 시작하여 an에 이르는 정점들은 멀티태스킹의 가능성을 나타내는 정보를 표시할 때 같은 이름인 m1 으로 표시한다. 마찬가지로 b1 부터 bn에 이르는 정점들도 같은 이름인 m2로 표시한다. 만일 ① 안에 ③, ④처럼 영역이 겹쳐지는 경우에는 맨 마지막에 부여된 이름으로 멀티태스킹 이름을 결정한다. z 정점에서 시작되어 서로 다른 링크로 분리되는 x1에서 xn, y1에서 yn에 이르는 정점들도 서로 다른 이름을 부여해서 멀티태스킹 가능성을 표시한다. 만일 a1 값이 변경되었을 때 b1 값에 영향을 주지 않고, 동시에 b1 값이 변경되었을 때 a1 값에 영향을 주지 않아 서로 독립적인 특성을 갖고 있어, 두 변수가 변경된 경우 각 변수에 대해 점진 평가를 수행할 때 멀티태스킹이 가능하다. 즉, m1 안에 소속된 변수들과 m2 안에 소속된 변수들은 동시에 병행 점진 평가를 수행해도 올바른 결과 값을 가질 수 있는 독립적인(independent) 특성을 갖는 변수들이다. 반면 m1 안에 속해 있는 변수들은 순차적으로 평가해야 올바른 값을 얻을 수 있는 종속적인(dependent) 변수들이다. 즉, 임의의 i에 대해, ai 변수와 bi 변수는 멀티태스킹이 가능한 반면 임의의 i와 j에 대해, ai와 aj는 순차적으로 평가해야 올바른 값을 갖게 된다. 따라서, 임의의 변수들이 서로 다른 멀티태스킹 이름을 갖는 경우 그 변수에 대해서는 멀티태스킹 평가가 가능함을 알 수 있다. 독립적인 변수와 멀티태스킹 변수를 다음과 정의한다.

[정의 1] 독립적인 변수

어떤 변수 A의 값이 변경될 때 다른 변수 B의 값이 변

경되지 않고 각 변수 값이 서로에게 영향을 주지 않을 경우 두 변수 A와 B는 독립적인 변수라 정의한다.

[정의 2] 멀티태스킹 변수

어떤 변수 A와 B가 독립적인 변수일 경우 두 변수는 값에 영향을 주지 않으므로 동시에 병행 점진 평가를 수행해도 정확한 값을 갖고 이러한 변수를 멀티태스킹 변수라 정의한다.

4. 멀티태스킹 점진 평가 알고리즘

점진 평가를 수행하려면, 변수 값을 수정하는 명령문이 나올 경우 종속차트의 종속링크를 통해 그 변수 값에 영향받아 변하게 될 변수들을 깊이우선탐색(DFS)으로 추적하여 영향 받는 명령문들을 다시 평가한다. 점진 평가를 수행하는데 사용되는 용어들을 정의하면 다음과 같다.

[정의 3] change-set

값이 변경된 변수에 의존하는 종속적인 변수들로 변경된 변수에 영향받는 변수들의 집합이 change-set이다.

[정의 4] evaluated-set

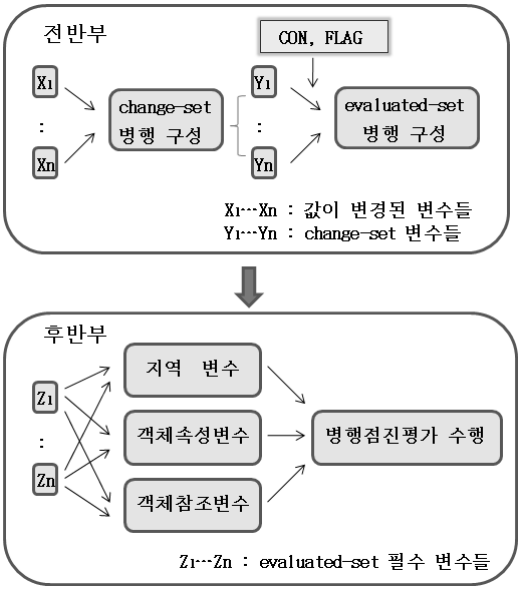
change-set에 속한 변수들 중 조건속성의 값에 따라 프로그램 수행 상 실행될 필요가 있는 변수들의 집합이 evaluated-set이다.

[정의 5] 필수(essential) 변수

change-set에 속한 변수들 중 조건속성의 값에 따라 프로그램 수행 상 실행될 필요가 있는 명령문에 속한 변수들을 필수(essential) 변수라 한다. 즉 evaluated-set에 속해 있는 변수들이다.

병행 점진 평가를 수행하는 과정은 (그림 4)에서처럼 두 단계로 진행된다. (그림 4)에서 CON은 조건속성을 나타내는 것이고 FLAG는 조건속성의 값을 표시하는 것이다. 첫 번째 단계인 전반부에서는 병행점진 평가에서 중요한 필수 변수를 찾아내는 과정으로 evaluated-set을 구성한다. 두 번째 단계인 후반부에서는 객체를 처리하는 부분으로 evaluated-set에 속한 필수변수의 유형별로 병행점진 평가를 수행한다.

먼저 전반부의 수행 과정을 살펴보면 변수 값이 변할 때 점진 평가를 수행하기 위해 종속링크를 찾아 가는 것이 중요하다. 만일 여러 변수들에 변화가 생겼을 경우 각 변수들에 대해 동시에 종속 링크를 추적하여 각 변수의 후속자 노드가 영향받는 변수들이 되고 이들 변수들로 change-set를 구성한다. change-set에 속한 변수들 중에서 프로그램이 실제로 실행될 때 반드시 실행되는 명령문에 소속된 변수들로 다시 평가해야 하는데, 이 필수 변수들이 evaluated-set에 들어간다. 만일 (그림 1)에서 x 값이 2로 변경되면 change-set은 {y, c, z}가 되고 ④번 if문의 조건속성이 거짓이므로



(그림 4) 병행 점진 평가 과정

c = y * a; 명령문이 실행되지 않아 evaluated-set은 {y, z}가 된다.

후반부에서는 evaluated-set에 속한 변수들을 다시 평가하게 되는데 독립적인 속성을 갖는 경우 멀티태스킹이 가능하므로 동시에 병행 평가하고 종속적인 특성을 갖는 변수들은 순차적으로 평가하게 된다. 평가하는 방법은 종속차트에 있는 정보를 검색하여 그 변수들이 속해있는 명령문들을 평가하게 된다. 따라서, 모든 변수들을 순차적으로 평가하는 것보다 변수의 특성에 따라 동시에 병행 점진 평가를 수행할 수 있으므로 효율적인 병행 점진 평가를 수행할 수 있다.

자바에서 점진 평가로 고려할 수 있는 경우는 지역 변수의 변화, 객체 속성 변수의 변화, 객체 이름의 변경, 클래스 이름이나 메서드 이름의 변경 등이다[2]. 멀티태스킹의 효율성을 높이기 위해 지역변수와 객체속성 변수의 변화를 제외한 나머지 경우의 변화에 대해서는 미리 변경될 사항을 종속차트에 조정하는 것이 더 바람직하다[4]. (그림 1)의 ⑦번 명령문 thr1.a = thr2.b * 2; 에서 thr1.a 대신 thr2.a로 객체 이름을 변경하면 해당 객체의 속성 변수에 변화가 생기게 되고 종속 링크도 변경해서 객체 thr2의 a값이 바뀌도록 종속차트를 조정한다. 클래스이름이나 메서드이름이 변경될 경우 해당 클래스나 메서드를 종속차트에서 찾아 그 클래스나 메서드에 소속되어 있는 변수들의 클래스이름이나 메서드이름을 변경하게 된다. 병행 점진 평가 알고리즘을 기술하기 위해 아래 세 가지 표기법을 정의한다[1].

(1) 순차 수행 : C_1 , C_2, … , C_n 이 순서대로 수행되는 경우 {: C_1 , C_2, … , C_n }
(2) 무조건 병행 수행 : 각 C_1 , C_2, … , C_n이 동시에 수행 {|| C_1 , C_2, … , C_n }
(3) 조건 병행 수행
{? $G_1 \rightarrow C_1$, $G_2 \rightarrow C_2$, … , $G_n \rightarrow C_n$ }

```
Algorithm MultiTasking(VT, DC, X);
 / let VT be the variable table in the DC /
 / let X be changed variables/
 / x1, x2, ... ,xn ∈ X /
 initialize mark flag;
{: {? x1 = variable in VT →
   {: P1 := dependency link in DC,
      if not (mark of P1) then
      {|| insert P1' successor using DFS into CS,
          mark of P1 := true } },
   x2 = variable in VT →
   {: P2 := dependency link in DC,
      if not (mark of P2) then
      {|| insert P2' successor using DFS into CS,
          mark of P2 := true } },
     ...
    xn = variable in VT →
    {: Pn := dependency link in DC,
      if not (mark of Pn) then
      {|| insert Pn' successor using DFS into CS,
          mark of Pn := true } } },
   / a1, a2, ... an ∈ CS /
{|| f1=a1's FLAG in DC, f2=a2's FLAG in DC, ... fn=an's FLAG in DC },
{||
   {? f1=G or T → if (a1's con.val) then insert a1 into ES,
      f1=F → if not ( a1's con.val) then insert a1 into ES,
      f1=I → insert a1 into ES },
   {? f2=G or T → if a2's con.val then insert a2 into ES,
      f2=F → if not ( a2's con.val) then insert a2 into ES,
      f2=I → insert a2 into ES },
          ...
   {? fn=G or T → if an's  con.val then insert an into ES,
      fn=F → if not ( an's  con.val) then insert an into ES,
      fn=I → insert an into ES } },
/ let b1, b2, ... bn ∈ ES, b1, b2, ... bn is dependent attribute /
{:  for each bi ∈ ES do
      if bi is local variable then evaluate bi
      else if bi is object member variable then
        find object of its methods call
        each associated object member variable evaluates
      else if bi is object name then
        each associated object member variable evaluates
      endif
    endFor; },
/ let c1, c2, ... cn ∈ ES, c1, c2, ... cn is independent attribute /
{||
   {? c1 is local variable → evaluate c1
      c1 is object member variable →
         find object of its methods call
         each associated object member variable evaluates
       c1 is object name →
         each associated object member variable evaluates },
    {? c2 is local variable →  evaluate c2
      c2 is object member variable →
         find object of its methods call
         each associated object member variable evaluates
       c2 is object name →
         each associated object member variable evaluates },
      ...
    {? cn is local variable → evaluate cn
      cn is object member variable →
         find object of its methods call
         each associated object member variable evaluates
       cn is object name →
         each associated object member variable evaluates }
   } }
```

(그림 5) 병행 점진 평가 알고리즘

G_1 이 참이면 C_1 을 수행하고 G_2 가 참이면 C_2 를 수행하고 G_n 이 참이면 C_n를 수행하는데 각각을 동시에 비교하면서 병행 수행한다.

(그림 4)의 전반부와 후반부에서 수행하는 과정을 알고리즘으로 기술하는데 전반부에서 change-set과 evaluated-set을 병행으로 구성하고 후반부에서 evaluated-set에 속한 필수 속성들에 대해 멀티태스킹 가능성에 따라 병행 수행하거나 순차 수행하여 평가하게 된다. VT는 DC(dependency chart)에 있는 변수들을 쉽게 검색하기 위해 변수들의 이름으로 구성된 색인테이블이다. con.val 은 조건 속성의 값을 나타내는 것이고, change-set은 CS로, evaluated-set은 ES로 표기한다.

위의 알고리즘의 정확성을 증명하려면 알고리즘이 종료된 후 프로그램 내의 모든 변수가 정확한 값을 갖는 지 조사해 보면 된다. 병행 점진 평가의 경우, 자바의 객체와 관련된 명령문들에서 동시에 병행 점진 평가를 수행했을 때 올바른 값을 갖는지 증명하면 된다. 먼저, 클래스 이름이나 메서드 이름이 변경된 경우는 종속차트에서 클래스 이름이나 메서드 이름을 찾아서 변수의 소속 정보를 변경하면 올바른 값을 갖게 된다. evaluated-set에 속한 변수들 중에 지역 변수나 객체 속성변수나 객체 이름일 경우가 있고 각 경우에서 객체에 따른 병행 점진 평가가 올바른지 다음 정리를 통해 증명한다.

[정리 1] evaluated-set에 속한 모든 지역변수들을 멀티태스킹을 고려하여 동시에 병행 점진 평가를 수행했을 때 지역변수들은 정확한 값을 갖는다.

[증명] evaluated-set에 속한 지역변수들 중 $x_1, x_2, \ldots x_n$ 는 멀티태스킹이 가능한 독립적인 변수라 가정하고 $y_1, y_2, \ldots, y_n$ 는 종속적인 변수라 가정하자. 각 x_i와 각 y_i는 변경된 변수에 영향 받는 변수이고 다시 평가해야하는 필수적인 변수이다. 각 x_i와 각 y_i는 변경된 변수들에 영향받는 변수들이므로 이러한 변수들은 값이 변하게 되는 변수들이고 다시 평가해야만 정확한 값을 갖게 된다. 종속차트의 종속링크를 통해 DFS로 추적된 변수이므로 올바른 추적 경로를 통해 발견된 변수이다. 따라서, 종속차트의 라인정보를 따라 각 x_i와 각 y_i가 포함된 명령문들 $S_1, S_2, \ldots S_n$을 찾을 수 있고, 각 Si 명령문을 다시 평가하면 그 명령문에 소속된 각 x_i와 각 y_i는 정확한 값을 갖게 된다. 각 x_i는 독립적인 변수이기 때문에 <정의 2>에 의해 멀티태스킹으로 평가 가능한 변수들이므로 동시에 병행 평가하더라도 올바른 값을 갖는 변수이다. 따라서, 각 x_i에 대해 병행 점진 평가를 수행하면 정확한 값을 갖게 된다. 각 y_i는 멀티태스킹이 불가능한 종속적인 변수이므로 순차적으로 점진 평가하면 정확한 값을 갖는다. ■

[정리 2] 객체 속성변수가 변경되고 그 객체 속성변수가 evaluated-set에 속해서 동시에 병행 점진 평가를 수행한 후

그 변수는 정확한 값을 갖는다.

[증명] 객체 속성변수가 변경되고 그 객체 속성변수가 evaluated-set에 속했다는 것은 변경된 값에 영향 받고 다시 평가해야하는 필수적인 객체 속성 변수라는 사실을 알 수 있다. 이러한 객체 속성변수가 소속된 객체를 찾아 다시 평가된 값을 변경해 줘야 각 객체는 정확한 객체 속성변수 값을 갖게 된다. 객체 속성변수들에 대해서 동시에 병행 점진 평가를 수행했을 때 정확한 값을 갖는 지 증명하면 된다.

evaluated-set에 속한 객체 속성변수 중에는 독립적인 변수와 종속적인 변수가 있다. x_1, x_2, ... x_n를 멀티태스킹이 가능한 독립적인 객체 속성변수라 가정하고 임의의 x_i 가 소속된 객체를 XX_i라 가정하자. 종속차트의 VT에서 x_i를 찾을 수 있고 종속차트의 라인 정보에 따라 x_i가 속한 명령문을 찾을 수 있다. x_i가 소속된 명령문을 다시 평가해야 x_i는 올바른 값을 갖게 되고 x_i가 소속된 객체 XX_i를 종속차트에서 찾을 수 있다. x_i는 독립적인 변수이므로 <정의 2>에 의해 멀티태스킹이 가능하고 동시에 병행 점진 평가 과정을 거쳐 평가하더라도 순차적으로 평가한 것과 같이 올바른 값을 갖게 된다. 각 x_i에 대해 병행 점진 평가를 수행하여 다시 계산된 x_i 값은 객체 XX_i에 속한 객체 속성변수가 변경되도록 저장하고 다시 평가된 값이 심벌 테이블에 들어간다. 따라서 객체 속성변수가 변경되어 멀티태스킹 과정으로 병행 점진 평가를 수행하더라도 각 x_i 는 정확한 객체 속성변수 값을 갖게 된다.

y_1, y_2, ... y_n를 멀티태스킹이 불가능한 종속적인 객체 속성변수라 가정하고 임의의 y_i 가 소속된 객체를 YY_i라 가정하자. 종속차트의 VT에서 y_i를 찾을 수 있고 종속차트의 라인 정보에 따라 y_i가 속한 명령문을 찾을 수 있고 점진 평가과정을 거쳐 y_i가 소속된 명령문을 다시 평가하게 된다. y_i 값은 객체 YY_i에 속한 객체 속성변수가 변경되도록 저장하여 올바른 값을 갖게 된다. 따라서 각 객체에 소속된 종속적인 변수와 독립적인 변수 모두 정확한 객체 속성변수 값을 갖게 된다. ■

[정리 3] 객체 이름이 변경되고 evaluated-set에 속했을 때 동시에 병행 점진 평가를 수행한 후 각 객체 속성 변수들은 정확한 값을 갖는다.

[증명] 종속적인 객체인 경우는 병행 점진 평가가 불가능하므로 순차적으로 점진 평가하고 독립적인 객체인 경우 동시에 멀티태스킹으로 병행 점진 평가를 수행했을 때 각 객체 속성 변수들은 정확한 값을 갖는다는 것을 증명하면 된다.

독립적인 속성을 갖는 객체 이름이 변경되어 evaluated-set에 속하면 각 객체 속성변수들은 종속차트에 변경된 객체로 소속 정보가 바뀌게 되고 종속 링크도 변경이 된다. 임의의 독립적인 XX를 evaluated-set에 속한 객체라 가정하고 그 객체에 속한 객체 속성변수를 x_1, x_2, ... x_n 이라 가정하자. 각 x_i 는 종속 차트에 있는 정보에 따라 작성되고 링크를 좇아가서 객체 이름이 변경된다. 각 x_i는 값이 변하게 될 영향 받는 필수적인 객체 속성변수들이고 이 변수들을 종속차트에서 찾을 수 있고 종속차트의 라인 정보에 의해 각 x_i가 속한 해당 명령문을 찾을 수 있다. 각 x_i는 독립적인 변수이므로 <정의 2>에 의해 멀티태스킹으로 병행 점진 평가를 수행해도 올바른 값을 갖는다. 즉, 각 x_i가 소속된 명령문들을 동시에 다시 실행하더라도 독립적인 변수이므로 올바른 값을 갖게 된다. 각 x_i의 명령문들은 종속차트를 통해 찾을 수 있고 이들 명령문에 대해 동시에 병행 점진 평가를 수행하면 각 x_i는 올바른 값을 갖게 된다.

임의의 종속적인 YY를 evaluated-set에 속한 객체라 가정하고 그 객체에 속한 객체 속성변수를 y_1, y_2, ... y_n 이라 가정하자. 각 y_i 는 종속 차트에 있는 정보에 따라 작성되고 링크를 좇아가서 객체 이름이 변경된다. 각 y_i는 종속차트에서 찾을 수 있고 종속차트의 라인 정보에 의해 각 y_i가 속한 해당 명령문을 찾아 점진 평가를 순차적으로 수행하여 각 y_i는 올바른 값을 갖는다. 따라서 독립적인 객체 속성변수와 종속적인 객체 속성변수는 정확한 값을 갖게 된다. ■

<정리 1>, <정리 2>, <정리 3>에 의해 evaluated-set에 속한 변수들 중 독립적인 속성을 갖는 변수들은 동시에 병행 점진 평가를 수행하고 종속적인 변수들은 순차적으로 점진 평가를 수행했을 때 각 변수들은 정확한 값을 갖는다는 것을 알 수 있고 제시된 알고리즘은 정확하다는 사실을 확인할 수 있다.

5. 병행 점진 평가의 효율성 실험

자바 언어에 대해서 병행 점진 평가를 수행하는 연구가 없어 점진 평가를 수행했을 때와 본 연구에서 제시한 병행 점진 평가 방법으로 수행했을 때를 비교하는데 윈도우 환경에서 자바 언어로 두 단계의 모의실험을 수행하여 병행 점진 평가의 효율성을 분석한다. 첫 번째 단계에서는 자바로 작성된 네 가지 유형의 프로그램에 대해 병행 점진 평가의 효율성을 분석한다. 두 번째 단계에서는 병행 수행의 효율성을 높일 수 있는 요인으로 작용하는 독립적인 변수의 비율에 따라 병행 수행 평가의 효율성을 분석한다. 병행 평가를 수행하기 위해 자바의 Thread 클래스의 확장 클래스인 subThread 클래스를 만들고 여러 subThread 객체를 생성한 후 동시에 각 subThread 객체의 실행을 시작하는 start() 메소드를 수행하여 병행 평가하게 된다.

첫 번째 모의실험에서, 합계와 평균을 계산하는 프로그램(P1), 최대값과 최소값을 찾는 프로그램(P2), 급료를 계산하는 프로그램(P3), 배열로 선형 리스트를 구현하는 프로그램(P4)에 대해 실험하고 네 가지 명령문인 배정문, 조건문, 반복문, 메서드 호출문에 대해 변수 값을 수정하였고 그 수정으로 점진 평가하는 명령문의 비율과 병행 점진 평가를 수행하는 명령문의 비율을 비교하여 수정하는 명령문 유형에 따라 병행 점진 평가의 효율성을 분석한다[4].

〈표 1〉 프로그램 점진 평가 결과

(단위:%)

프로그램 유형	배정문		조건문		반복문		호출문		평균	
	점진	병행	점진	병행	점진	병행	점진	병행	점진	병행
P1	30	21	12	10	31	24	35	23	27	19
P1	89	62	3	2	90	68	91	59	68	48
P2	33	24	32	26	38	29	41	27	36	27
P2	93	66	94	75	95	72	96	62	95	69
P3	6	4	7	4	76	58	71	40	40	27
P3	8	6	13	19	88	67	82	47	48	32
P4	9	5	17	9	88	58	81	46	49	30
P4	5	3	9	5	89	59	83	47	47	28
평균	34	24	23	18	74	54	73	44	51	35

<표 1>에서 네 가지 프로그램 유형에 대해 각 유형별로 2가지 형태의 프로그램을 사용하는데 소량의 자료를 사용한 경우와 대량의 자료를 사용한 경우이다. 프로그램 유형별로 비교했을 때 P4 프로그램의 경우 모든 수정 유형을 평균적으로 분석해 볼 때 점진 평가보다 1.7배 빠르게 실행되어 다른 프로그램 유형보다 더 효율적이다. P4의 경우 여러 수정 유형에서 다른 프로그램들보다 독립적인 변수들이 많고 그 변수들에 대해 수정될 경우 병행 점진 평가가 가능하므로 다른 프로그램 유형보다 효율적이라는 사실을 알 수 있다. 네 가지 명령문 유형의 병행 점진 평가의 수정 중 호출문의 경우 멀티태스킹의 가능성이 높은 독립적인 변수들의 수정 가능성이 높아서 병행 점진 평가가 점진 평가보다 1.6배 빠르게 수행되고 다른 수정 유형보다 효율성이 높음을 알 수 있다[4].

평균적으로 고려해볼 때 독립적인 변수들의 수정 가능성이 높을 경우 점진 평가보다 병행 점진 평가를 수행했을 때 평가 효율성이 높음을 알 수 있고 점진 평가하는 것을 100%로 놓았을 때 병행 점진 평가를 수행했을 때 점진 평가의 60-70% 만 실행하여 1.45배 빠르게 실행되어 점진 평가보다 효율적인 병행 점진 평가를 수행하게 된다[4].

두 번째 모의실험에서, 병행 평가하는 경우의 효율성을 비교하기 위해 병행 처리가 가능한 독립적인 변수의 비율 유형을 V1, V2, V3의 세 가지로 나타내고 각각 25%, 50%, 75%로 지정한다. 모의실험을 간단히 하기 위해 다음과 같이 가정한다. 첫째, 변화된 변수에 영향받는 명령문의 수를 넣어 준다. 둘째, 조건문의 경우 각 조건을 나타나는 부분의 명령문 수는 동일하다. 셋째, 독립적인 변수는 배정문, 조건문, 반복문, 호출문에 동일한 비율로 속해 있다.

<표 2>에 점진 평가한 것과 세 쓰레드(Thread) 클래스의 객체(**T3**)로 병행 수행한 결과와 다섯 쓰레드 클래스의 객체(**T5**)로 병행 수행한 결과가 나와 있다. evaluated-set을 순차적으로 점진 수행하는 것보다 세 쓰레드클래스의 객체로 병행 수행하는 것이 1.4 ~ 1.5배 더 빠르고 다섯 쓰레드 클래스의 객체로 병행 수행하면 1.6배 더 빠르다는 것을 알 수 있다. 독립적인 비율이 가장 높은 V3의 평가 시간이 가장 빠르다는 사실에서 독립적인 변수가 많을수록 병행 수행 시간을 단축할 수 있음을 확인할 수 있다.

〈표 2〉 변수유형별 병행점진평가

(단위:μs)

명령문수	점진	T3			T5		
		V1	V2	V3	V1	V2	V3
50	28	24	18	15	23	17	12
100	56	48	38	28	45	33	21
500	286	238	190	142	229	171	114
1000	571	477	381	286	458	343	229
2000	1145	954	763	571	917	687	458
3000	1718	1431	1146	857	1375	1030	685
4000	2288	1908	1526	1143	1834	1375	916
5000	2860	2385	1907	1429	2292	1719	1145

6. 결 론

본 논문에서는 병행 점진 평가를 수행하기 위해 종속 차트를 사용하여 객체 지향언어인 자바 언어에서 멀티태스킹 방식으로 병행 점진 평가를 수행하는 방법을 제시하였다. 프로그램의 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하여 변화를 추적하므로 효율적인 병행 점진 평가를 수행할 수 있다. 자바 언어에서 병행 점진 평가를 수행하는 알고리즘을 제시하고 그 알고리즘의 정확성을 증명하였고 두 단계의 모의실험을 통해 병행 점진 평가가 점진 평가를 수행하는 것보다 더 효율적이라는 사실을 확인하였다.

첫 번째 단계에서, 네 가지 프로그램 유형에 대해서 실험하였고 네 가지 명령문인 배정문, 조건문, 반복문, 메서드 호출문에 대해 수정한 후 다시 평가될 명령문 수를 중심으로 병행 점진 평가의 효율성을 검토해 보았다. 실험 결과 멀티태스킹의 가능성이 높은 독립적인 변수들의 수정 가능성이 높은 경우 병행 점진 평가를 수행하는 것이 점진 평가를 수행하는 것보다 더 효율적이었다. 평균적으로 고려해볼 때 점진 평가하는 것을 100%로 놓았을 때 병행 점진 평가를 수행했을 때 점진 평가의 60-70% 만 실행하여 병행 점진 평가가 1.45배 빠르게 수행되어 병행 점진 평가가 점진 평가 보다 더 효율적으로 수행된다는 사실을 확인할 수 있다.

두 번째 단계에서 병행 처리가 가능한 독립적인 변수의 비율 유형에 따라 병행 평가를 수행하였고 독립적인 변수의 비율이 높을수록 실행 효율성이 높았다. 평균적으로 1.4 ~ 1.6 배 빠르게 수행되었고 세 쓰레드 객체로 수행할 때보다 다섯 쓰레드 객체로 수행한 경우 더 효율적으로 수행되었다.

참 고 문 헌

[1] 한정란 “작용 식 기반 점진 해석” Ph. D Thesis 이화여대 1999.
[2] 한정란 “확장된 종속차트를 사용한 효율적인 점진 평가 방법”,

인터넷정보학회 논문지, 제10권 2호, pp.75-84, 2009.
[3] 한정란, “객체 지향 언어를 위한 의미 명세”, 인터넷정보학회 논문지, 제8권 5호, pp.35-43, 2007.
[4] 한정란, “멀티쓰레딩을 활용한 병행 점진 평가”, 추계학술발표 논문집, 한국정보처리학회, 2009.
[5] 한정란, 최성, “작용 식 기반 통합 점진 해석 시스템 구축”, 정보처리학회 논문지 제11권 3호, pp.149-156, 2004.
[6] David A. Watt and Deryck F. Brown, “Formalizing the Dynamic Semantics of Java”, 2006.
[7] Alblas, Henk, “Concurrent Incremental Attribute Evaluation,” Lecture Notes in Computer Science #461, Springer-Verlag, pp.343-358, 1990.
[8] Boehm, Hans Juergen and Willy Iwaenepoel, “Parallel Attribute Grammars Evaluation,” The 7th International Conference on Distributed Computing Systems, IEEE, Spetember, pp.347-354.
[9] Kuiper, Matthijs F. and S. Doaitse Swierstra, “Parallel Attribute Evaluation: Structure of Evaluators and Detection of Parallelism,” Lecture Notes in Computer Science #461, Springer-Verlag, pp.61-75, 1990.
[10] Fábio Pasini, Fernando L. Dotti,Code Generation for Parallel Applications Modelled with Object-Based Graph Grammars, Electronic Notes in Theoretical Computer Science, Volume 184, pp. 113-131, 2007.
[11] Raul Medina Rora and David S. Notkim “ALOE users′ and implementers′ guide” Carnegie-mellon Computer Science Depart. Research Report CS-81-145.
[12] A. N. Habermann “The Gandalf Research project” Computer Science research Review Carnegie-Mellon University.

한 정 란

e-mail : jlhan@uhs.ac.kr
1985년 이화여자대학교 전자계산학과(학사)
1987년 이화여자대학교 컴퓨터공학과(이학석사)
1999년 이화여자대학교 컴퓨터공학과(공학박사)
1999년～현 재 협성대학교 경영정보학과 부교수
관심분야 : 형식언어론, 점진컴파일러, 전자상거래, XML등