■ '99 정보과학 논문경진대회 수상작

다중 공간 조인의 병렬 처리
(Parallel Processing of Multi-Way Spatial Join)

류 우 석 † 홍 봉 희 ††
(WooSeok Ryu) (BongHee Hong)

요 약 GIS에서 사용하는 다중 공간 조인은 두 개 이상의 공간 조인이 중첩된 표현이다. 이는 공간 조인에 비해 보다 많은 수행 시간을 필요로 하는데 이를 빠르게 처리하기 위한 병렬화 알고리즘에 대한 연구가 없었다.

이 논문에서는 다중 공간 조인을 다중 공간 여과와 다중 공간 정제로 나누어서 병렬화한다. 그리고, 정제 단계에서 효율적인 정제 수행을 위해 2단계 실행 방법을 제시하는데, 첫번째가 다중 공간 여과의 결과인 후보 객체 테이블에서 발생하는 객체 및 연산의 중복을 제거하기 위한 그래프 생성이고, 두번째가 그래프의 분할에 의한 병렬 정제이다. 그래프에 의한 정제가 그렇지 않은 방법에 비해 매우 높은 성능 향상을 보였으며 병렬 정제를 위한 태스크 생성 방법은 객체를 정점으로 표현하는 그래프에서의 중복 최소화 분할방법이 가장 좋은 성능을 나타내었다.

Abstract Multi-way spatial join is a nested expression of two or more spatial joins. It costs much to process multi-way spatial join, but there have not still reported the scheme of parallel processing of multi-way spatial join.

In this paper, parallel processing of multi-way spatial join consists of parallel multi-way spatial filter and parallel spatial refinement. Parallel spatial refinement is executed by the following two steps. The first is the generation of a graph used for reducing duplication of both spatial objects and spatial operations from pairs candidate object table that are the results of multi-way spatial filter. The second is the parallel spatial refinement using that graph. Refinement using the graph is proved to be more efficient than the others. In task creation for parallel refinement, minimum duplication partitioning of the Spatial_Object_On_Node graph shows best performance.

1. 서 론

GIS에서 사용하는 공간 질의는 방대한 양의 기하 데이타 및 공간 연산의 복잡성으로 인하여 기존의 관계형 데이타베이스 또는 객체 지향형 데이타베이스에서의 질의보다 훨씬 많은 계산 시간을 필요로 하며 또 공간 연산도 매우 다양하다. 대표적인 공간 연산 중 하나인 공간 조인은 두 공간 데이타 집합에 대해 포함(within), 교차(cross), 겹침(overlap)등의 특정 공간 조건을 만족하는 객체 쌍의 집합을 찾아내는 연산이다. 공간 조인은 두 데이타 집합에 대한 다중 주사(multi-scan)를 필요로 하므로 점 질의(point query)나 영역 질의(region query)에 비해 많은 수행 시간을 필요로 한다. 그래서 공간 조인을 효율적으로 수행하기 위해 지금까지 공간 색인의 연구[1], 조인 방법에 대한 연구[2] 그리고 공간 조인의 병렬화 연구[3, 4]가 진행되었다.

다중 공간 조인은 공간 조인보다 더욱 복잡한 공간 질의이다. 다중 공간 조인은 이원 공간 조인의 중첩(nested)된 표현으로써 여러 공간 조인이 포함된 공간 질의이다. 여기서의 중첩이란 서로 다른 공간 조인이 동일한 데이타 집합에 의해 연결되는 경우를 말하는데, (A Θ_1 B)와 (B Θ_2 C)에서 B가 두 조인에 걸쳐 있으므로 이를 (A Θ_1 B Θ_2 C)라 부른다. 즉 다중 공간 조인은 다음의 조건을 만족한다.

$$A\ \Theta_1\ B\ \Theta_2\ C \equiv (A\ \Theta_1\ B) \cap (B\ \Theta_2\ C)$$

† 비 회 원 : 부산대학교 컴퓨터공학과
wsryu@hyowon.cc.pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@hywon.cc.pusan.ac.kr

예를 들어 주거구역에 포함된 공장 중에서 재개발 구역에 포함되는 공장을 출력하라와 같은 공간 질의는 주거구역과 공장건물에 대한 공간 조인에 추가로 공장건물과 재개발구역의 공간 조인을 필요로 한다. 즉 주거구역, 공장건물, 재개발구역에 대한 다중 공간 조인을 필요로 하는 질의이다. 이와 같은 다중 공간 조인은 복잡한 공간 질의에서 많이 사용되지만, 이를 효율적으로 수행하기 위한 연구는 거의 없다.

이 논문에서는 불필요한 정제 대상을 줄이고 정제 시간을 최소화 하기 위해 우선적으로 다중 공간 여과를 수행하고 이후에 다중 공간 여과의 결과에 대해서 한꺼번에 정제를 수행하는 방법을 먼저 제안한다. 그리고, 다중 공간 여과의 결과인 후보 객체 테이블(Candidate Object Table)에서 정제 연산 쌍을 정의한 레코드에서 객체 및 연산의 중복이 많이 발생하게 되는데 이 중복으로 인하여 정제 단계에서 많은 계산의 낭비가 발생하는 문제점이 있으므로 정제 연산을 수행할 공간 객체 및 정제 연산의 중복을 제거하여 최적의 정제를 수행하기 위한 그래프 모델링 방법인 연산을 정점으로 표현(G_{on})하는 방법과 객체를 정점으로 표현(G_{oe})하는 방법을 제안하고 각각에 대한 정제 알고리즘을 비교한다.

다중 공간 조인의 병렬화 방법으로는 다중 공간 여과와 정제 각각에 독립적으로 병렬화를 적용하여서 다중 공간 조인의 병렬화를 달성하는데, 다중 공간 여과는 정제에 비해 매우 적은 시간을 차지한다. 그러므로 이 논문에서는 정제 단계에서의 효율적으로 수행을 위한 병렬 알고리즘을 제시하고 성능 평가를 수행한다. 성능평가로는 우선 이원 공간 조인을 연속적으로 수행하는 것과 다중 공간 여과와 정제에 의한 수행방법을 우선적으로 비교하며, 그래프 생성 방법인 G_{on}과 G_{oe}에 의한 정제 성능도 함께 비교한다. 그리고, G_{on}과 G_{oe}에서의 병렬화 알고리즘과 병렬 정제 성능을 비교 측정한다.

이 논문의 구성은 다음과 같다. 2장에서는 기존의 다중 조인에 대한 연구 및 공간 조인에 대한 관련연구를 소개한다. 3장에서는 이 논문에서 제시하는 병렬 다중 공간 조인의 방법론을 설명하며, 4장에서는 효율적인 정제 수행을 위한 그래프 알고리즘을 제안하고 이를 병렬 수행하기 위한 태스크 생성 방법을 설명한다. 5장에서는 병렬 다중 공간 여과 및 정제의 성능을 실험을 통해 비교 평가한다. 마지막으로 6장에서는 이 논문의 결론 및 향후 연구에 대하여 기술한다.

2. 관련연구

이 논문에서 제시하고자 하는 병렬 다중 공간 조인에 대한 관련 연구는 비 공간 데이타베이스, 즉 관계형 데이타베이스와 객체 지향 데이타베이스에서의 다중 조인 및 다중 조인의 병렬화 연구와 공간 데이타베이스에서의 공간 조인 및 병렬 공간 조인 연구로 크게 나눌 수 있다. 하지만 이 논문과 직접적인 연관이 있는 병렬 다중 공간 조인에 대한 기존의 연구는 전혀 없다. 2.1에서는 비 공간 데이타베이스에서의 관련 연구에 대해서 설명하고 2.2에서는 공간 데이타베이스에서의 관련 연구에 대해 언급한다.

2.1 비공간 데이타베이스에서의 병렬 다중 조인 연구

비공간 데이타베이스에서의 병렬 다중 조인에 대한 연구는 80년대 후반에서부터 90년대까지 많은 연구가 진행되었다. [6]에서는 병렬 질의 최적화를 위한 2단계 접근 방법을 제안하였는데 첫번째 단계는 조인 순서의 결정으로 이는 bush-tree및 left-deep tree, right-deep tree중에서 실행 비용에 의해 가장 성능이 우수한 질의 트리를 비용 모델에 의해 선택하는 단계이다. 그리고, 선택된 질의 트리에 대해 프로세서 할당을 통한 병렬화 수행을 하는 것이 두 번째 단계이다. 관계형 데이타베이스에서는 질의 수행에 대한 실행 비용 예측이 비교적 간단하나 복잡한 공간 데이타를 다루는 공간 데이타베이스에서는 실행 비용 예측이 어려우므로 실행 비용에 의한 질의 트리 선택은 공간 데이타베이스에서는 적용하기 어렵다.

조인의 병렬화는 크게 두 가지로 나뉘어 연구가 진행되었는데 초반에는 하나의 조인 연산을 병렬화 하는 연산 내 병렬화(intra-operation parallelism)에 대한 연구 [7, 8, 9, 10]가 주로 수행되었으며 이후에는 다중 조인 및 복잡한 질의에서 보다 나은 성능을 발휘하기 위한 연산간 병렬화(inter-operation parallelism)가 [11, 12] 등에 의해서 진행되었다. 대부분의 연구에서는 시뮬레이션을 통해 성능평가를 수행하여 논문의 당위성을 증명하였는데 [13]는 다중 조인에서의 여러 가지 조인 방법 및 질의 트리 별로 실질적인 성능평가를 통해 비교적 객관적인 결론을 도출하였으며 여기서는 bush-tree에서의 전체 병렬화(full parallel)에 의한 수행이 가장 성능이 좋은 것으로 평가되고 있다. 그리고, 선형 트리(linear-tree)에서의 파이프라인 병렬 조인 기법은 pipelining delay가 심해서 성능이 좋지 않은 것으로 평가되고 있다.

기존의 데이타베이스에서 사용하는 다중 조인의 병렬화는 병렬 다중 공간 여과에도 적용할 수 있다. 하지만 비용 모델에 의한 조인 순서 결정은 공간 데이타의 복잡성으로 인해 바로 적용하기 어려우며, 다중 조인의 병

렬화를 공간 데이타에 적용할 때 발생하는 공간 조인의 특징은 추가로 고려할 필요가 있다. 즉 공간 조인은 여과와 정제를 분리함으로 얻는 이점을 고려할 필요가 있다. 따라서 이 논문은 그 동안 다루어지지 않은 다중 공간 여과의 병렬화, 다중 공간 정제의 병렬화 알고리즘을 새로 제시하고 이를 입증하고자 한다.

2.2 공간 데이타베이스에서의 연구

공간 데이타베이스에서의 공간 조인에 대한 연구는 공간 색인과 이에 대한 조인 기법에 대한 연구와 공간 조인의 병렬화 연구로 나뉘어진다. 공간 색인은 객체를 기준으로 분할하는 R-tree 계열과 영역을 기준으로 분할하는 Quad-tree 및 Grid-file로 나뉜다. 공간 조인에서는 대부분 R-tree계열의 공간 색인을 사용하고 있는데 [2]에서는 R*-tree를 사용하여 공간 색인 단계에서 공간 객체의 근사치(MBR, Minimum Boundary Rectangle)를 이용하여 공간 여과를 먼저 수행한 다음에 후에 후보 객체 쌍에 대하여 정제를 수행하고 있다. 여과와 정제의 분리를 통해 불필요한 기하 데이타 비교를 줄여서 수행시간을 단축할 수 있다. R-tree와 유사한 공간 색인으로 Seeded-Tree가 [1]에서 연구되었는데 이는 공간 연산의 결과에 대해 다시 공간 조인을 수행할 때 공간 색인을 빠르게 재생성하기 위해 고안된 색인으로써 여러 공간 연산이 중첩되어 있을 때 효율적이나 두 대상 공간 데이타 중에서 하나의 공간 데이타에는 공간 색인이 존재하고 있음을 가정하고 있다.

공간 조인의 병렬화는 [3, 4]에서 연구되었다. [3]은 R*-tree 공간 색인을 이용하고 있으며 병렬 공간 조인에서의 태스크 생성 방법 및 할당에 대한 몇 가지 방법론을 제시하고 있으나 그래프 생성에 의한 태스크 생성이 아닌 휴리스틱 태스크 할당 방법을 제시하고 있다. 그리고, [4]는 PMR Quad-tree 공간 색인을 이용하고 있으며, 범용적으로 사용하지 않는 SIMD array processor 시스템에서의 병렬 공간 조인 방법을 제안하고 있다.

공간 데이타베이스에서의 다중 공간 조인은 최근에 [5, 14, 15]에서 연구되고 있다. [5]에서는 공간 색인으로서 2P-tree를 사용하고 있는데, 이는 R-tree와 비슷하지만 공간 객체의 근사치로서 MBR대신 MBS (Minimum Bounding Strip)에 의한 근사치를 사용하고 있다. 여기서는 공간 조인에서의 여과와 정제의 분리를 다중 공간 조인에도 적용하는 것이 효율적이라는 사실을 성능평가와 함께 제시하였지만 정제 단계에서의 구체적인 알고리즘은 제시되어 있지 않으며, 후보 객체 테이블에서의 데이타 및 연산의 중복에 대해서는 언급이 없어 중복성의 심각성을 간과하였다. [14, 15]에서는 공간 색인으로서 R-tree를 사용하였는데 [14]에서는 질의 트리로서 left-deep tree를 선택하여 파이프라인 방법을 적용하는 알고리즘을 제안하였으며 이에 대한 비용 모델을 제시하였다. 그리고 [15]은 다중 조인에 CSP(Constraint Satisfaction Problem)을 적용한 효율적인 알고리즘 및 비용 모델을 제시하였다. 그러나 두 연구 모두 여과와 정제를 분리하여 고려하지 않으므로 불필요한 정제를 수행할 수 있다. 지금까지 다중 공간 조인의 병렬화 연구 사례는 전혀 발표되지 않았는데 앞으로 다중 공간 조인 수행 방법에 병렬 다중 공간 조인 알고리즘을 적용하여 새로운 다중 공간 조인의 병렬 실행 방법을 만들고 검증할 필요가 있다.

3. 다중 공간 조인

그림 1은 이 논문에서 사용하는 다중 공간 조인 알고리즘을 설명하기 위한 예제 데이타인데 이후의 예제들은 그림 1의 가상 데이타를 기준으로 한다. 이 데이타는 A, B, C 세 데이타 집합에 대한 다중 공간 조인으로써 공간 데이타는 모두 다각형 데이타(Polygon Data)이며 공간 조인 연산은 겹침 연산(overlap operation)을 사용하는 것을 가정한다. 그리고, 다중 공간 조인은 다중 공간 조인의 대상이 되는 데이타 집합 중 첫번째 데이타 집합과 마지막 데이타 집합이 서로 다른 비 재귀 다중 공간 조인(non-recursive multi-way spatial join)을 가정한다. 3.1장에서는 다중 공간 조인의 특징을 설명하고 3.2장에서는 여과와 정제를 분리하는 다중 공간 조인 방법에 대하여 구체적으로 설명한다.

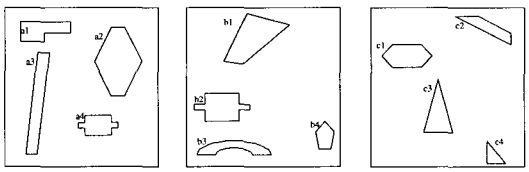


그림 1 (A Θ_1 B Θ_2 C)을 위한 예제 데이타

3.1 다중 공간 조인의 특징

다중 조인의 특징은 여러 조인이 중첩되어 있으므로 한 공간 조인의 결과가 다른 조인의 입력이 되어 수행되는 점이다. 그러므로 조인의 결과는 다른 데이타 집합과의 조인에 의해 더욱 제한이 가해지게 된다. 공간 조인은 기하 데이타의 비교로 많은 계산 비용을 필요로 하는데 다중 공간 조인에서 여과와 정제의 분리 없이 한꺼번에 수행하면 불필요한 기하 데이타 비교까지 수

행하게 되므로 계산 시간의 불필요한 낭비가 발생하게 된다. 그림 2는 그림 1의 공간 객체에 대한 MBR을 겹쳐서 표현한 것으로 (a3, b3)는 서로 MBR이 겹치나 b3은 데이타 집합 C의 어느 객체와도 겹치지 않으므로 실제 기하 데이타에 대한 비교가 필요 없이 바로 제거할 수 있다. (b4, c4)도 이와 마찬가지로 바로 제거 가능하다. 즉, 공간 조인과 달리 다중 공간 조인에서는 (a3, b3), (b4, c4)와 같이 MBR비교는 만족하지만 정제는 불필요한 객체 쌍들이 존재할 수 있다.

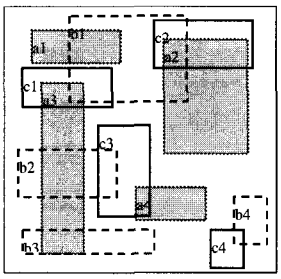


그림 2 그림 1의 데이타에 대한 MBR의 겹침

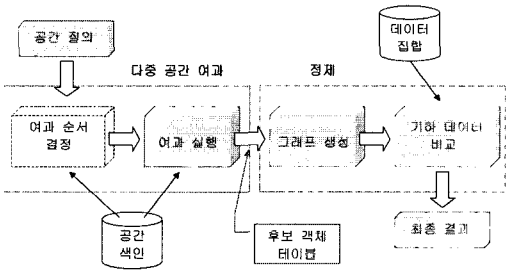


그림 3 다중 공간 조인의 전체적 흐름도

3.2 다중 공간 조인의 수행

그림 3은 다중 공간 조인의 전체적인 흐름도이다. 이 논문에서 제시하는 다중 공간 조인은 그림 3과 같이 크게 2단계로 나누어 수행한다. 첫번째 단계는 다중 공간 여과(Multi-way Spatial Filter) 단계로서 실제 기하 데이타가 아닌 기하 데이타의 근사치를 가지고 다중 공간 조인을 수행하는 것이다. 근사치를 사용한 다중 공간 여과를 우선적으로 수행함으로써 3.1절에서 언급한 다중 공간 조인의 특징을 이용하여 비교적 적은 비용으로도 많은 제한을 가할 수 있는 장점이 있다. 하지만 다중 공간 여과의 수행은 근사치에 대한 비교이므로 기존의 관계형 데이타베이스 또는 객체 지향 데이타베이스에서의 속성 조인에 의한 다중 조인 방법과 크게 다르지 않는 특징이 있다. 다중 공간 조인의 두 번째 단계는 다중 공간 여과의 결과에 대해 실제 정제 연산을 수행하는 단계로서 대부분의 수행 시간을 정제 단계에서 소모하게 된다. 3.2.1과 3.2.2에서는 다중 공간 여과와 후보 객체 테이블에 대하여 각각 설명하고 4장에서 정제 방법을 기술한다.

3.2.1 다중 공간 여과

다중 공간 여과는 실제 기하 데이타 대신에 기하 데이타의 근사치를 이용하여 다중 공간 조인을 수행하여 정제가 필요한 후보 객체 테이블을 생성하는 작업이다. 다중 공간 여과의 실행은 다중 조인과 마찬가지로 우선 다중 공간 여과의 실행에 필요한 조인 순서의 결정과 결정된 조인 순서에 따라 조인을 실행하는 두 단계로 다시 나뉘어진다. 조인 순서의 결정은 bush-tree, left-deep tree등의 질의 트리 중에서 비용 모델을 통하여 가장 최적의 수행 시간을 나타내는 질의 트리를 선택하는 단계이다. 그리고, 선택된 질의 트리에 대하여 실제 여과를 수행하게 된다.

다중 공간 여과는 관계형 데이타베이스에서의 다중 조인과 크게 다르지 않다. 차이점은 첫째, 데이타의 관점에서 볼 때 관계형 데이타베이스에서는 1차원 속성값의 비교, 또는 식별자에 대한 동일 조인(eq-join)에 의해 조인을 수행하는데 반해 공간 데이타베이스에서의 다중 공간 여과는 공간 객체의 근사치, 즉 MBR등의 2차원 데이타간의 관계성 비교를 통해 여과를 수행한다. 그리고, 색인의 관점에서 볼 때 관계형 데이타베이스에서는 속성 값의 크기에 따라 생성하는 B-tree 등의 1차원 색인에 의한 조인을 수행하고, 공간 데이타베이스에서는 공간 데이타의 위치에 따라 생성하는 R-tree, Quad-tree, Grid-file등의 공간 색인에 의해 공간 데이타를 접근하여 공간 여과를 수행한다.

다중 공간 여과는 사실 다중 공간 조인의 전체적인 수행 시간에서 볼 때 매우 적은 시간을 소모한다. 수치상으로도 다중 공간 여과의 수행시간이 전체의 10%미만일 정도로 다중 공간 여과는 다중 공간 조인에서 매우 적은 위치를 차지하고 있다. 그리고, 다중 공간 조인의 알고리즘도 비공간 데이타베이스에서의 다중 조인과 크게 다르지 않다. 그러므로 다중 공간 여과는 다중 공간 조인에서 크게 중요한 문제가 아니며, 더구나 병렬화의 중요한 이슈가 되지 못한다.

3.2.2 후보 객체 테이블

다중 공간 여과의 결과로서 후보 객체 테이블이 생성되는데, 이 테이블의 레코드 각각은 후보 객체 리스트(Candidate Object List)로써 정제작업, 즉 실제 기하 데이타 비교가 필요한 객체 쌍들의 묶음이다. 후보 객체

테이블은 다음과 같이 정의된다.

정의 : (A Θ_1 B Θ_2 C)에 대한 후보 객체 테이블은
MBR_{ai} Θ_1 MBR_{bj}, MBR_{bj} Θ_2 MBR_{ck}를 만족하는 (ai, bj, ck)의 집합이다.
단, ai ∈ A , bj ∈ B, ck ∈ C이고, MBR_a Θ MBR_b 는 a와 b에 대한 MBR 조인이다.

R1 : a1	b1	c1
R2 : a1	b1	c2
R3 : a2	b1	c1
R4 : a2	b1	c2
R5 : a3	b1	c1
R6 : a3	b1	c2
R7 : a3	b2	c2
R8 : a3	b2	c3

그림 4 그림 1의 (A Θ_1 B Θ_2 C) 에 대한 후보 객체 테이블 (Candidate Object Table)

그림 2에 대한 후보 객체 테이블은 그림 4와 같이 표현된다. 그림 4의 R1 레코드에 있는 데이타는 (a1, b1, c1)인데, 이는 (a1 Θ_1 b1)과 (b1 Θ_2 c1)의 수행을 의미하며 두 연산 모두 만족해야 최종 결과로서 선택된다. 그러나 그림 4의 R1과 R2에서 (a1, b1)의 중복과 같이 객체, 연산이 발생하며 공간 색인의 종류에 따라 다중 공간 여과단계에서 레코드의 중복도 발생하게 된다. 그러므로 중복의 고려 없이 레코드별로 정제 수행을 하게 되면 이러한 중복되는 연산들을 두 번 이상 실행하는 문제점이 발생하게 된다. 이 중복성은 데이타 집합이 복잡해질수록 많이 발생하게 되며, 실제 데이타에서도 상당히 많이 발생한다. 후보 객체 테이블에서의 연산의 중복의 정도를 나타내는 연산의 중복률은 연산의 중복 회수와 정제가 필요한 연산의 회수간의 비율로 정의되며 이는 아래 수식과 같이 정의하였다.

$$Duplication\ Rate = \frac{\sum_{i=1}^{Nc-1} C_{(i,i+1)} - \sum_{i=1}^{Nc-1} CU_{(i,i+1)}}{\sum_{i=1}^{Nc-1} CU_{(i,i+1)}} = \frac{(Nc-1)*C_{table} - \sum_{i=1}^{Nc-1} CU_{(i,i+1)}}{\sum_{i=1}^{Nc-1} CU_{(i,i+1)}}$$

N_c : 테이블에서의 열의 수
$C_{(i,i+1)}$: i 와 i+1 번째 열에 포함된 객체 쌍의 총 수 (중복 포함)
$CU_{(i,i+1)}$: $C_{(i,i+1)}$에서 중복되는 객체 쌍을 제거한 후에 남은 객체 쌍의 수
C_{table} : 테이블의 레코드 개수 ($C_{table} = C_{(i,i+1)}$)

위 중복률을 표현하는 수식에서 분모는 전체 테이블에서 정제를 필요로 하는 연산의 회수이다. 분자는 연산의 중복 표현된 회수인데, 이는 테이블의 모든 연산의 회수에서 정제를 필요로 하는 연산의 회수를 뺀 값이다. 중복률이 0이면 후보 객체 테이블은 중복이 전혀 발생하지 않으며 1이면 100%의 중복률을 나타낸다. 이 중복률은 데이타 집합간의 밀집도 및 기하 데이타의 크기 등에 따라 아주 다양한 값을 가지는 특징이 있다. 그림 4의 후보 객체 테이블은 정제가 필요한 연산의 수가 (a1,b1), (a2,b1), (a3, b1), (a3, b2), (b1, c1), (b1, c2), (b2, c3)의 7개이고, 실제 표현된 연산의 수는 8*2, 즉 16개 이므로 (16-7) / 7 = 1.28의 중복률을 가진다.

4. 다중 공간 조인의 병렬 정제

실제 데이타에 대한 다중 공간 조인에서 후보 객체 테이블에서의 데이타 또는 연산의 중복은 정제 단계에서 불필요한 디스크 접근 및 CPU 계산의 낭비를 유발하므로 효율적인 정제를 위해서는 중복의 제거가 반드시 필요하다. 이 논문에서는 정제 단계를 두 단계로 나누어 수행하는데 첫번째 단계가 바로 후보 객체 테이블의 중복을 제거하기 위해 후보 객체 테이블을 그래프로 재구성하는 단계인데, 이 단계에서 모든 연산의 중복이 제거되어 최적의 정제 수행이 가능해진다. 그리고, 두 번째 단계가 그래프를 탐색하며 실제 기하 비교를 수행하는 단계이다. 4.1장에서 그래프 생성 방법 및 그래프에서의 순차 탐색 방법에 대해 기술하고 4.2에서 정제의 병렬화 방법을 다루고 있다.

4.1 후보 객체 테이블에서의 그래프 생성

이 장에서는 후보 객체 테이블의 데이타에 대해 그래프를 생성하는 두 가지 모델링 방법을 제시하고 있다. 여기서 사용하는 그래프 G는 정점 N과 간선 E 및 정점의 집합 NC로 구성되는데, NC는 동일한 데이타 집합의 객체 또는 객체 쌍을 가지는 정점들의 집합으로 정의된다. 후보 객체 테이블에서 이 그래프를 생성하는 두 가지 방법의 차이점은 정점 및 간선에 포함되는 정보의 차이인데, 첫번째는 각각의 연산을 정점으로 표현하는 방법이고 두 번째 방법은 객체를 정점으로 표현하는 방법이다.

4.1.1 연산을 정점으로 표현 (Operation on Node)

이 방법은 연산을 정점으로 표현하고 정점에 포함된 같은 객체가 포함된 정점들을 간선으로 표현하는 방법이다. 연산을 정점으로 표현하는 그래프 G_{on}및 이를 구성하는 정점과 간선의 정의는 다음과 같다.

- 정점 $N_{a\,b)}$
  MBR 조인을 만족하는 a와 b의 쌍, 단 $a \in A$, $b \in B$이며 A와 B는 공간 조인을 필요로 하는 데이타 집합.
- $N_{(a\,b1)}$와 $N_{(b2\,c)}$를 연결하는 간선 E
  b1 = b2 을 만족하는 정점간의 연결, 단 $a \in A$, b1,b2 $\in B$, $c \in C$ 이며 A와 B, B와 C는 공간 조인을 필요로 하는 데이타 집합.
- 그래프 G_{on}
  후보 객체 테이블을 N(a b)와 E로 표현한 그래프
- $NC_{(A\,B)}$
  N(a b)의 집합, 단 $a \in A$, $b \in B$

그림 4를 G_{on}으로 표현할 때 레코드 (a1, b1, c1)은 $N_{(a1\ b1)}$과 $N_{(b1\ c1)}$의 두 정점과 간선으로 표현된다. 또 (a1, b1, c2)는 $N_{(a1\ b1)}$과 $N_{(b1\ c2)}$로 분리되어 저장되는데 $N_{(a1\ b1)}$은 이미 이전 레코드에서 생성되어 있으므로 $N_{(b1\ c2)}$만 새로 생성하고 이전의 $N_{(a1\ b1)}$과 간선을 통해 연결하게 된다. 즉, 이 그래프 표현 방법은 동일한 연산이 두 번 이상 중복되어 표현되지 않고 오직 한번만 표현되므로 연산의 중복을 완전히 제거하는 장점을 지니고 있다. 그러나 각각의 정점에 b1과 같이 동일한 객체가 포함될 수 있으므로 객체의 중복은 해결하지 못하므로 동일한 객체를 읽기 위해 디스크 접근을 여러 번 해야 하는 문제가 발생할 수 있다. 그림 4에 대한 G_{on}모델링은 그림 5와 같다.

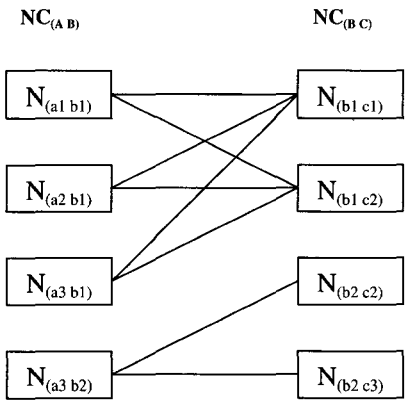


그림 5 그림 4의 후보 객체 테이블에 대한 G_{on} 표현

Gon의 생성은 후보 객체 테이블에서 한 레코드씩 읽으면서 그래프에 정점과 간선을 삽입하는 방법으로 이루어진다. 정점을 삽입할 때마다 동일한 정점이 존재하는지를 탐색해야 하는데, 탐색 시간은 정점의 개수에 비례하게 되므로 정점의 수가 많아지면 그래프의 생성 시간이 매우 길어지게 된다. 그러므로 정점들에 대해 색인을 생성함으로써 정점 삽입을 위한 탐색 시간을 줄여서 그래프 생성 시간을 줄일 수 있다. 이를 위한 색인으로 이 논문에서는 해쉬 색인(hash index)을 사용하고 있으며 구현 알고리즘도 복잡하지 않다.

G_{on}에서의 정제는 그래프에서 $NC_{(A\ B)}$에 포함된 정점들을 기준점으로 하여 그래프를 탐색하면서 기하 비교를 수행한다. $NC_{(A\ B)}$에 포함된 정점은 해쉬 테이블의 슬롯을 순차 탐색하면서 접근하는데, 이 정점에서의 행 기준 탐색(depth-first traversal) 을 통하여 그래프 탐색을 수행하게 된다.

4.1.2 객체를 정점으로 표현 (Object on Node)

이 방법은 객체를 정점으로 표현하고 기하 비교를 위한 객체 쌍은 정점 간의 간선을 사용해서 표현하는 방법이다. 객체를 정점으로 표현하는 G_{oe} 및 이에 대한 정점과 간선의 정의는 다음과 같다.

- 정점 : N_o
  후보 객체 테이블에 포함된 객체 o를 가지는 정점
- 간선 : E(s e)
  Ns와 Ne를 연결하는 간선, 이때 객체 s와 e는 MBR조인을 만족한다. 단, $s \in A$, $e \in B$이며 A와 B는 공간 조인을 필요로 하는 데이타 집합.
- 그래프 Goe
  후보 객체 테이블을 Na와 E(s e)로 표현한 그래프
- NCi
  No의 집합, 단 $o \in$ (후보 객체 테이블의 i 번째 column 의 집합)

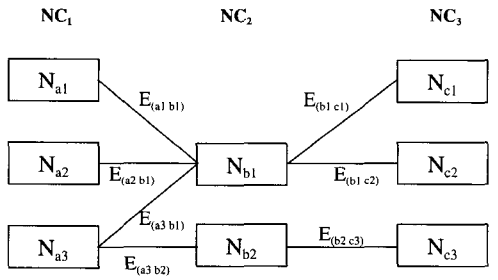


그림 6 그림 4의 후보 객체 테이블에 대한 G_{oe} 표현

그림 4를 G_{oe}로 표현할 때 레코드 (a1, b1, c1)은 N_{a1}, N_{b1}, N_{c1}의 세 정점과 $E_{(a1\ b1)}$, $E_{(b1\ c1)}$의 두 간선으로 표현된다. 이 방법은 연산이 간선에 포함되어 표현되므로 연산의 중복을 제거하고 있으며 또한 객체가 오직 하나의 정점에 저장되므로 객체의 중복도 제거하는 장점을 가진다. 그림 4에 대한 완전한 G_{oe} 표현은 그림 6

과 같다.

G_{oe}의 생성 또한 G_{on}과 마찬가지로 후보 객체 테이블에서 레코드 별로 삽입되는데, G_{on}에서와 마찬가지로 첫 번째 열의 노드는 색인을 사용하여 빠르게 그래프를 생성할 수 있다. G_{oe}의 생성 알고리즘 및 순차 탐색 알고리즘은 G_{on}과 상당히 비슷하다.

4.2 그래프에 대한 병렬 정제

앞에서 생성한 G_{on} 또는 G_{oe}에서의 병렬화 수행은 그래프 분할을 통한 태스크 생성과 프로세서 할당에 의한 정제 수행의 단계를 거친다. 그러나 그래프는 비교적 복잡하게 얽혀 있으므로 그래프 생성 방법에 따라 동일한 연산을 여러 프로세서가 맡아서 수행할 수 있다. 이는 곧 정제 시간의 낭비가 되므로 태스크 간의 중복이 최소화 되도록 태스크를 생성하는 그래프 분할 알고리즘이 필요하다. 이 장에서는 빠른 시간에 중복을 제거하여 효율적인 수행을 유도하는 중복 최소화 분할 방법을 제안하고 5장에서는 중복 최소화 방법의 수행 성능으로 디스크 접근 시간 및 CPU 연산 시간을 비교 평가한다. 그리고, 추가적으로 그래프의 분할을 통한 방법이 아닌 그래프의 제거에 의한 정제 방법을 4.2.2에서 제안한다.

4.2.1 중복 최소화 분할

그래프를 분할하는 가장 좋은 방법은 그래프에서 연결 요소(connected component)들을 추출해 내는 것이다. 추출된 연결 요소는 다른 연결 요소와 연결 간선 수가 최소화되었다고 가정한다. 연결 요소 간에 연결 간선 수가 많으면 연결 요소를 태스크로 할당할 경우에 중복 정제 비용이 큰 문제가 있다. 병렬 처리에서 각각의 연결 요소들을 태스크로 생성하면 태스크 간의 중복이 전혀 발생하지 않는다. 그러나 연결 요소들의 크기는 상당히 다양하며 수행 속도도 서로 상당히 다르다. 그리고, 대상 데이타가 밀집된 데이타일수록 연결 요소들의 크기가 커져서 태스크의 부하 균등화가 어려워지는 문제가 발생하며, 또한 그래프에서 연결 요소들을 추출하는 시간도 상당한 비중을 차지한다. 그림 7은 예제 그래프로서 이는 2개의 연결 요소를 가지는데 이는 2개의 태스크로 병렬 수행하는 것이 최적이다. 그러나 프로세서가 3개 이상일 때에는 2개의 프로세서에만 태스크가 할당되는 문제점이 있다.

위 문제를 해결하기 위하여 이 논문에서는 그래프 전체에 대해서 연결 요소를 생성하는 대신 그래프에서의 임의의 두 정점과 간선을 제외하고 나머지에 대해서 연결 요소를 생성하여 분할을 수행하는데, 이 연결 요소를 부분 연결 요소(sub-connected component)라고 한다. 부분 연결 요소는 다음과 같이 정의된다.

• 부분 연결 요소 (sub-connected component)

임의의 두 정점에 대하여 서로간에 경로가 존재하면 이를 부분 연결 요소라 한다. 단, 경로에 포함 된 정점 N_c에 대하여 $\forall N_c \in (NC_1 \cup NC_2)$ 를 만족해야 한다.

이는 그래프 전체에 대해 연결 요소를 고려하지 않고 그래프의 일부분, 즉 NC_1과 NC_2에 있는 정점들에 한해서만 연결 요소를 고려함으로써 빠른 시간에 그래프를 분할할 수 있는데, 그래프에서의 연결요소 및 부분 연결 요소는 그림 7과 같다.

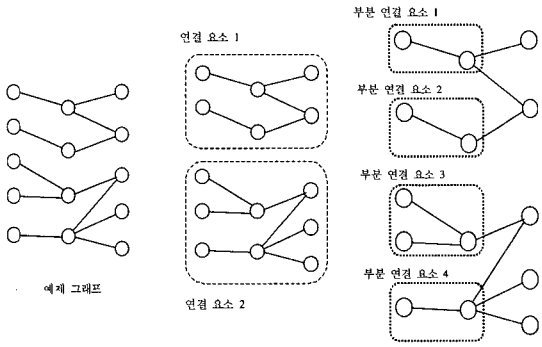


그림 7 예제 그래프에서의 연결 요소 및 부분 연결 요소

위 부분 연결 요소를 하나의 태스크로 정의하여 그래프를 분할할 수 있는데, G_{on}을 여기에 적용하면 그림 5에서 부분 연결 요소는 다음과 같이 나뉘게 된다.

Group1 : $N_{(a1,\ b1)}$, $N_{(a2,\ b1)}$, $N_{(a3,\ b1)}$, $N_{(b1,\ c1)}$, $N_{(b1,\ c2)}$

Group2 : $N_{(a3,\ b2)}$, $N_{(b2,\ c2)}$, $N_{(b2,\ c3)}$

그리고 그림 6의 G_{oe}를 여기에 적용할 때는 N_{a1}, N_{a2}, N_{a3}, N_{b1}, N_{b2}가 준 연결 요소로 묶이게 되며 이를 수행할 시 그래프를 탐색하여 N_{c1}, N_{c2}, N_{c3}도 함께 수행된다. 3개의 데이타 집합에 대한 다중 조인에서는 이 휴리스틱 방법으로 태스크 간의 중복을 완전히 제거할 수 있다. 하지만 4개 이상의 데이타 집합에 대한 다중 조인에서는 중복이 발생하게 되며 그래프의 형태에 따라 부분 연결 요소 간의 중복의 정도가 달라지게 된다. 그러나 실제 사용되는 대부분의 다중 공간 조인은 4개 이하의 데이타집합에 대한 다중 조인인데 이 때에는 부분 연결 요소에 의한 분할방법이 연결 요소에 의한 분할방법과 크게 차이가 없으면서도 더욱 빠르게 분할할 수 있으므로 최적에 가까운 성능을 나타낼 수 있다.

G_{on}에서 부분 연결 요소를 찾아서 분할하는 알고리즘은 매우 간단하다. G_{on}에서의 부분 연결 요소는 다음과

같은 특징을 가지고 있다.

정리 1. G_{on}의 부분 연결 요소

G_{on}의 부분 연결 요소에 있는 정점 중 NC_1에 포함된 임의의 정점 $N_{(a\ b)}$, $N_{(c\ d)}$에 대하여 항상 b=d를 만족한다.

G_{on}에서는 NC_1에 포함된 임의의 정점 $N_{(a\ b)}$와 연결되어 있는 NC_2의 모든 노드 $N_{(c\ d)}$에 대해서 b = c를 만족하는 특성이 있다. 그리고, 이를 만족하는 $N_{(c\ d)}$에 대해 간선이 존재하는 NC_1의 임의의 노드 $N_{(a'\ b')}$에 대해서도 b′ = c를 만족하게 된다. 즉, $N_{(a\ b)}$와 $N_{(a'\ b')}$는 $N_{(c\ d)}$를 통해서 경로가 존재하므로 이는 부분 연결 요소인데, 이때 항상 b = b′ = c를 만족하므로 정리 1이 증명된다. 정리 1의 역도 참으로서 NC_1의 임의의 노드 $N_{(a\ b)}$와 $N_{(c\ d)}$에서 b = d를 만족하면 이 노드들은 항상 부분 연결 요소이다. 그러므로 G_{on}에서 부분 연결 요소 간에는 첫번째 열의 두 번째 객체만 동일하면 되므로 이는 첫번째 열의 노드를 삽입할 때 사용하는 해쉬 함수의 키로서 두 번째 객체의 식별자를 사용하면 부분 연결 요소에서 NC_1에 포함된 정점은 항상 동일한 해쉬 슬롯에 저장되게 된다. 즉, 해쉬 함수를 사용하여 부분 연결 요소를 쉽게 추출할 수 있다.

하지만 G_{oe}에서의 부분 연결 요소 추출은 G_{on}처럼 해쉬 함수만으로는 해결할 수 없다. 이는 실제 그래프 탐색을 통해 이루어지는데 이는 NC_1의 정점에서는 Flink를 탐색하고, NC_2의 정점에서는 Blink를 탐색하면서 재귀 호출(recursive call)을 통해 부분 연결 요소를 추출한다. 이는 다음과 같은 알고리즘에 의해 생성된다.

INPUT Na : Na ∈ NC_1을 만족하는 G_{oe}의 임의의 노드
OUTPUT S : NC_1에 포함되면서 Na와 준 연결 요소인 노드들의 집합, 초기값은 ∅

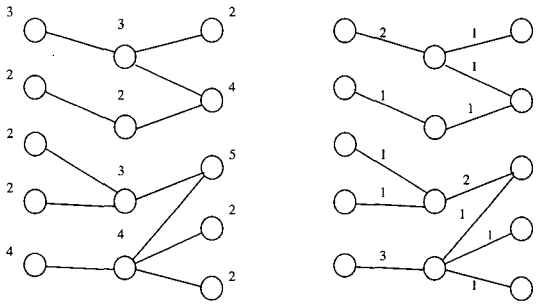
ALGORITHM Make_SCC_Goe
Begin
  S = S ∪ Na
  For all Nb *such that Nb ∈ C2 & E(a b) is exist*
    For all Na' *such that Na' ∈ C1 & E(a' b) is exist*
      If Na' ∉ S then
        Make_SCC_Goe (Na', S)
End.

알고리즘 1 G_{oe}에서의 부분 연결 요소 추출 알고리즘

병렬 수행에서 그래프의 생성은 모든 프로세서에 의해 동시에 이루어지는데 반해 태스크의 생성은 마스터에서만 이루어진다. 이때 마스터 프로세서는 부하 균등화를 위한 태스크의 할당 순서 결정을 해야 하는데, 순서 결정을 위한 태스크의 실행 비용은 G_{on}에서는 태스크에 포함된 정점의 개수이고, G_{oe}에서는 태스크에 포함된 간선의 개수로 산정한다. 정제를 위해 마스터에서 전송하는 태스크는 부분 연결 요소에서 NC_1에 포함되는 정점의 집합인데, 다른 프로세서들은 이 정점들만 가지고 스스로 그래프를 탐색하면서 정제를 수행한다. 이렇게 모든 프로세서가 그래프를 가지므로 부하 평준화에서 전송되는 데이타의 양을 줄일 수 있다.

4.2.2 그래프 제거에 의한 병렬 정제

그래프 제거에 의한 병렬 정제는 그래프의 분할 없이 기하 비교를 통해 그래프를 줄여 가면서 정제를 수행하는 방법이다. 정제의 순서를 결정하기 위해서는 정점 또는 간선에 대한 가중치가 필요한데, 그림 8과 같이 그래프 표현 방법에 따라 가중치가 정점에 또는 간선에 부여된다. Gon에서는 정점에 가중치를 부여하는데 그 값은 정점의 연산 결과를 필요로 하는 정점의 개수로 정의한다. 즉, 그림 8 가)의 왼쪽 가장 위의 정점은 3개의 정점에 영향을 미치므로 가중치가 3이 된다. 그리고, Goe에서는 간선에 가중치를 부여하는데 간선에 포함된 연산의 결과를 필요로 하는 간선의 개수로 가중치를 정의한다.



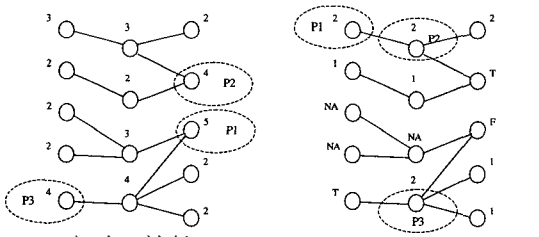
가) G_m에서의 가중치 나) G_m에서의 가중치

그림 8 그림 7의 예제 그래프에 대한 가중치 부여

그래프 제거에 의한 정제는 높은 가중치(weights)를 가진 연산을 먼저 수행하면서 그 결과를 그래프에 다시 반영하여 연관된 연산들의 가중치를 재 조정하면서 모든 연산이 종료될 때까지 반복적으로 프로세서에 연산을 할당하는 방법이다. 이때 각각의 프로세서에 할당되는 태스크의 단위는 연산, 즉 두 객체 식별자가 된다. 병렬 수행 방법으로 마스터 프로세서는 가장 가중치가 높은 연산을 휴지 프로세서(idle processor)에 할당하고 결과를 받아서 실시간으로 그래프를 재조정하며 모든

연산의 수행이 종료될 때까지 이를 반복하게 된다. 이때 다른 프로세서들은 마스터 프로세서에게 정제할 연산을 요청하고 연산을 받으면 정제 수행, 결과값 전송을 반복하게 된다.

그림 9는 그림 8의 가) G_{on} 그래프에 대한 병렬 정제인데 여기에서는 프로세서가 4개인 것으로 가정하고 있다. 하나는 마스터 프로세서이고 나머지 3개가 정제를 수행하는 프로세서로서 P1~P3의 이름을 가지고 있다. 그림 9의 가)는 정제 단계에서의 초기 프로세서 할당으로서 가중치가 높은 연산의 순서대로 3개의 프로세서가 할당되어서 각각 정제를 수행하게 된다. 정제의 결과는 두번째 단계인 그림 9 나)에 반영되는데 P2, P3는 참이고 P1에서 수행한 연산이 거짓이라고 가정한다면 그림 9 나)에서 정제가 더 이상 불필요한 연산이 NA로 표시가 되어 불필요한 정제를 수행할 필요가 없다. 이런 방법으로 할당 및 정제를 계속하면 그림 9 라)에서 하나의 레코드가 선택이 되고 그림 9 라)의 단계에서 P1을 마지막으로 정제가 완료된다. 이 방법은 객체를 정점으로 표현하는 그래프 생성 방법에서도 동일하게 적용될 수 있다.



가) 초기 프로세서 할당 나) 2번째 프로세서 할당

다) 3번째 프로세서 할당 라) 4번째 프로세서 할당

그림 9 그림 8의 가)에 대한 프로세서 할당 및 정제

이 방법은 그래프의 제거단계에서 정제가 불필요한 연산을 그래프 분할에 의한 정제에서보다 더욱 완벽하게 제거할 수 있으며, 그래프 분할에 의한 정제와는 달리 프로세서간 중복 계산은 전혀 발생하지 않는 장점이 있다. 그러나, 그래프는 마스터 프로세서에서만 유지하게 되므로 정제시 프로세서간 전송할 데이타 양이 많아지는 문제가 있으며 또한 그래프 재조정에 따른 마스터 프로세서의 계산량이 많아져서 그만큼 다른 프로세서들의 휴지 상태(idle state)가 빈번하게 발생할 수 있다. 이는 그래프가 복잡하고 프로세서가 많을수록 더욱 심각해지므로 병렬 수행시 좋은 성능을 나타내기 어렵다. 5장의 실험평가에서는 그래프 분할에 의한 정제방법의 성능을 실험을 통해 비교 평가한다.

5. 실험 평가

5.1 성능 평가 환경

5.1.1 Machine Architecture

이 논문에서는 다중 공간 조인의 성능을 평가하기 위해 두가지 시스템을 사용하였다. 하나는 범용적으로 사용하는 UNIX시스템인 Sun Sparc 20로서 80M의 주 메모리를 탑재하고 있으며 운영체제는 Solaris 2.5.1이다. 이 시스템은 여러 다중 공간 조인 알고리즘의 성능을 비교하는데 사용된다. 그리고, 병렬화 성능을 비교하기 위해 사용한 시스템은 MIMD 구조 및 공유 디스크 구조를 가지는 Parsytec사의 CC-16 병렬 컴퓨터로 powerPC 604 CPU을 16개 장착하고 있으며 4개의 프로세서가 하나의 라우터(router)에 고속 네트워크로 맞물려 있다

CC16에서는 엔트리 노드(entry node)라는 하나의 프로세서가 디스크 입출력이나 콘솔(console) 입출력을 담당하고 있는데 데이타는 전역 디스크(global disk)에 저장되어 있으며 엔트리 노드에 의해서 관리되고 있다. 엔트리 노드(Entry Node)를 제외한 나머지 노드들은 전역 디스크를 통해 데이타에 접근한다. 그리고, 엔트리 노드를 제외한 나머지 노드는 64M의 메모리를 가지며 엔트리 노드는 128M의 주 메모리를 가지고 있다. 노드간의 통신 방법은 메시지 전달(message-passing) 방식을 사용한다.

5.1.2 대상 데이타

이 논문에서는 Sequoia 2000 Storage benchmark [16]에 사용되는 다각형 데이타를 성능 평가를 위한 대상 데이타로 사용한다. 이는 총 79,607 개의 다각형으로 이루어져 있으며 한 다각형을 이루는 점의 개수는 수개에서 수천 개로 그 복잡성이 매우 다양하다. 그리고, 데이타 집합들은 urban layer, Agricultrual Land, Rangeland, Forest Land, Water, Island등의 다양한 layer로 나뉘어져 있으며 각각이 최고 20000여 개의 다각형을 가지고 있다. 이 데이타는 그림 10와 같다.

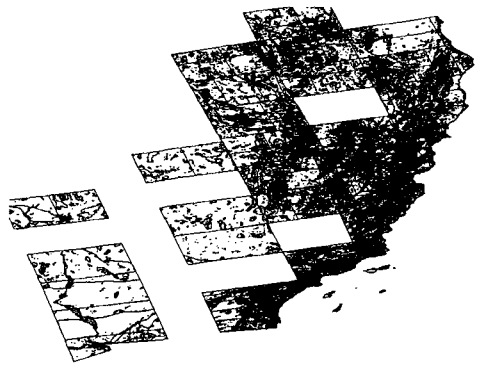


그림 10 성능평가에 사용할 Sequoia 다각형 데이타

성능평가에 사용할 데이타 집합은 그림 10의 데이타 중 6개의 layer를 추려내고 후보 객체 테이블에서의 연산의 중복률과 데이타 집합의 크기에 따라 4개의 layer를 가지는 12가지의 조인 순서를 무작위로 추출하였다. 성능평가에 사용할 10가지 데이타에 대한 표는 표 1에 나타나 있다.

표 1 병렬 다중 공간 조인을 위한 대상 데이타

	S1	S2	S3	S4	S5
데이타 집합A	7060	5291	3651	5291	3651
데이타 집합B	3651	4388	4388	3177	5291
데이타 집합C	4388	3651	3177	4388	3424
데이타 집합D	3424	3424	5291	3424	3177
후보객체 테이블의 크기	1152	2774	3218	4524	7445
연산의 중복률	0.53	0.98	1.02	2.03	3

	L1	L2	L3	L4	L5
데이타 집합A	11865	9663	11836	9680	11836
데이타 집합B	10237	10237	10237	9663	9680
데이타 집합C	5178	11867	9663	11836	9663
데이타 집합D	11836	5178	9880	5178	10237
후보객체 테이블의 크기	26115	72448	170760	345726	467698
연산의 중복률	1.63	5.17	10.29	15.53	20.64

S1 ~ S5은 단일 프로세서에서의 다중 공간 조인 성능을 평가하기 위한 비교적 적은 크기의 데이타로서 중복률이 50%에서 300%까지 되도록 5가지를 선택하였다. 그리고 L1 ~ L5은 병렬 알고리즘의 성능을 평가하기 위한 비교적 큰 크기의 데이타로서 L1은 163%의 중복률을 가지며 L5은 2000%의 중복률을 가진다. 일반적으로 복잡한 데이타일수록 중복률이 높아지는데 L1에서 L5로 갈수록 후보 객체 테이블의 레코드 개수가 늘어나는 것도 이 때문이다. 이 10가지 데이타를 사용하여 다중 공간 조인 알고리즘들의 성능 및 병렬 다중 공간 방법들의 성능 비교를 수행한다.

5.2 다중 공간 조인의 구간별 실행 시간

다중 공간 조인의 수행에서는 일반적인 단일 공간 조인에 비해 정제 시간의 비율이 훨씬 높게 차지한다. 정제 시간이 전체 수행 시간의 99%를 차지하며 그 중에서도 실제 기하 비교, 즉 CPU 연산 시간이 전체 수행 시간의 95% 이상을 차지한다. 이 결과에도 알 수 있듯이 다중 공간 조인의 핵심적인 성능 개선 방법은 실제 기하 데이타 비교 회수를 최대한 줄이는 방법의 고안이라 할 수 있다.

5.3 다중 공간 조인 알고리즘의 비교

이 장에서는 하나의 프로세서로 다중 공간 조인을 실행 할 때의 알고리즘별 수행 속도를 비교한다. 알고리즘의 비교는 S1 ~ S5 데이타에 대하여 비교 수행하였는데 5가지의 다중 공간 조인 방법에 대한 수행 시간을 비교하고 있다. 각각의 알고리즘에 대한 구체적인 기술은 표 2에 나타나 있다.

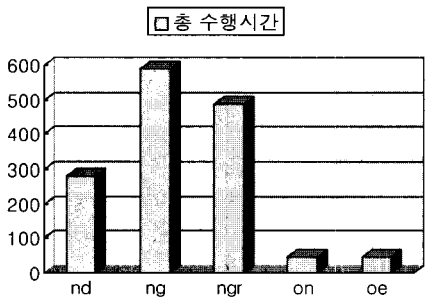
표 2 다중 공간 조인 알고리즘의 비교

ND	여과와 정제를 분리하지 않고 여과를 수행하면서 바로 정제 수행 (Non_Divide)
NG	다중 공간 여과를 먼저 수행하고 그 결과인 후보 객체 테이블에서 레코드 별로 정제를 수행 (Non_Graph)
NGR	다중 공간 여과를 먼저 수행하고 그 결과인 후보 객체 테이블에서 레코드의 중복을 제거한 후 레코드 별로 정제를 수행 (Non_Graph_with_delete_Record_duplication)
ON	다중 공간 여과를 수행하고 후보 객체 테이블에서 G_{on} 그래프를 생성한 후 그래프를 따라 정제를 수행 (Operation_to_Node_graph_generation)
OE	다중 공간 여과를 수행하고 후보 객체 테이블에서 G_{oe} 그래프를 생성한 후 그래프를 따라 정제를 수행 (Operation_to_Edge_graph_generation)

5가지 수행 알고리즘의 수행 시간은 그림 11과 그림 12에 나타나 있다. 그림 11은 S4에 대한 수행 시간을 구간별로 나타낸 것이고, 그림 12은 총 5가지의 데이타 집합에 대한 수행 성능의 비교로서 ND의 수행 속도를 100으로 할 때의 상대 속도에 비교 그래프이다. 일반적으로 여과와 정제를 분리하여 수행하는 것이 그렇지 않

은 것보다 성능이 우수하다고 알고 있으나 이는 정제를 어떤 방법으로 수행하는가에 따라 큰 차이가 있다. ND와 NG, NGR은 비교하면 이는 중복률에 따라서 속도차가 나게 된다. 즉, 그림 12을 볼 때 중복률이 1 이하인 S1와 S2에서는 NGR이 ND보다 성능이 우수하나 중복률이 1이상일 때는 중복 연산의 과부하 때문에 오히려 ND가 성능이 우수하다. 후보 객체 테이블에서는 모든 종류의 중복성, 즉 객체, 연산의 중복이 제거되지 않고 모두 중복 표현되므로 ND에서 한번 처리할 연산도 후보 객체 테이블에서는 여러 번 표현되게 된다. 그러므로 후보 객체 테이블에서 레코드별로 바로 정제를 하면 중복이 발생하는 객체와 연산은 중복되는 회수만큼 처리해야 하는 문제가 발생하므로 중복 연산의 과부하가 발생한다. 그리고 이 논문에서는 다중 공간 여과에서 다중 할당에 의한 그리드 파일을 색인으로 사용하므로 후보 객체 테이블에서 레코드 자체의 중복도 적지 않게 발생한다. NGR이 이 레코드 자체의 중복을 제거한 방법인데 레코드 중복의 제거만으로도 큰 성능 향상을 꾀할 수 있다. S1와 S3을 비교하면 S3이 중복률이 높으므로 ND에 비해 NG, NGR이 더욱 성능이 나빠지게 되며 ND와 NGR의 성능 비는 중복률에 비례하는 사실을 알 수 있다.

후보 객체 테이블에서 중복성을 제거하기 위해서 이 논문에서 제안한 두가지 그래프 생성에 의한 정제 성능은 ND 방법보다 성능이 높은 것으로 나타났다. 이 그



	총시간	여과	레코드 중복제거	그래프	정제	정제 - 디스크	정제 - CPU
nd	280.87	2.43			278.44	4.6	273.84
ng	588.85	0.75			588.1	12.47	575.42
ngr	489.3	0.79	3.15		485.36	9.37	475.77
on	45.45	1.3		0.1	44.05	1.48	42.55
oe	46.24	0.8		0.08	45.36	1.39	43.92

그림 11 S3 데이타에 대한 다중 공간 조인 방법별 수행 시간의 비교(단위 : 초)

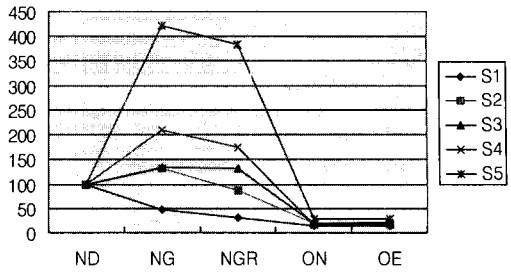


그림 12 데이타 집합별 수행 속도 비교 (ND를 100으로 환산할때)

래프 생성 방법은 후보 객체 테이블에서의 중복을 모두 제거한 것이므로 중복률과는 관계없이 ND에 비해 5배 이상의 성능향상을 나타낸다. 즉 그래프 생성에 의한 정제는 후보 객체 테이블의 중복률과는 전혀 무관하며 데이타 중복률이 높을수록 그래프에 의한 방법이 그래프를 사용하지 않은 방법보다 우수하다는 사실을 알 수 있다.

5.4 태스크 생성 방법별 정제 성능

5장에서 그래프에서의 연산의 중복을 최소화 하기 위한 중복 최소화 분할을 제안하였는데, 이 장에서는 이 성능을 비교하기 위해 중복 최소화 분할과 중복에 대한 고려 없이 분할을 수행하는 단순 분할을 그래프 모델링 방법과 함께 비교한다. 여기서 사용하는 단순 분할은 중복의 구분 없이 NC_1에 포함된 정점들을 프로세서의 개수에 따라 바로 나누고, 이를 태스크의 크기에 따라 정렬하는 방법이다.

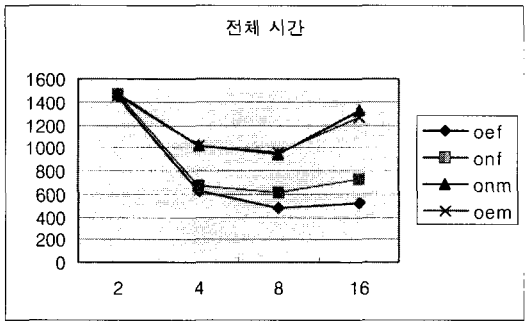
그림 18은 네가지 태스크 생성 방법에 대한 정제 수행 속도를 나타낸 그래프이다. 대상이 되는 네가지 태스크 생성 방법은 ONM, ONF, OEM, OEF인데, ONM은 연산을 정점으로 표현할 때의 단순 분할 방법이고, ONF는 연산을 정점으로 표현할 때의 중복 최소화 분할, OEM는 객체를 정점으로 표현할 때의 단순 분할, OEF는 객체를 정점으로 표현할 때의 중복 최소화 분할이다. 4가지 방법을 표로 표현하면 표 3과 같다.

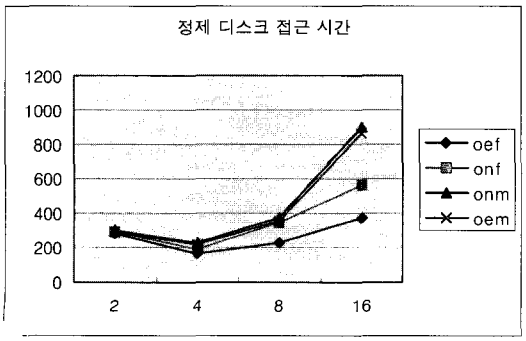
표 3 태스크 생성 방법 비교

태스크 생성 방법	그래프 표현 방법	그래프 분할 방법
ONM	연산을 정점으로 표현	단순 분할
ONF	연산을 정점으로 표현	중복 최소화 분할
OEM	객체를 정점으로 표현	단순 분할
OEF	객체를 정점으로 표현	중복 최소화 분할

그림 13을 보면 정제의 CPU계산 시간은 프로세서에 반비례하지만 디스크 접근 시간은 프로세서가 많아질수록 증가하며 ONM이나 OEM 방법은 디스크 접근 시간이 급격히 증가한다. 그 이유는 전역 디스크에 대한 부하 외에도 좋지 못한 분할 방법에 의한 중복 디스크 접근이 상당히 많이 발생하기 때문이다. 이는 전체 수행 시간에 큰 영향을 미치는데 단순 분할 방법은 중복 최소화 분할에 비해 데이타의 중복이 매우 높으며 이런 중복 때문에 프로세서가 많아질 경우에 디스크 부하를 유발하여 오히려 성능이 떨어뜨리는 결과를 초래한다.

전체 수행 시간을 비교하면 결과적인 성능의 비교는 첫째 단순 분할 방법에 비해 중복 최소화 분할이 월등이 우수하며, 둘째 객체를 정점으로 표현한 방법이 연산을 정점으로 표현한 방법보다 우수하다는 사실을 알 수 있다. 그리고, 객체를 정점으로 표현한 방법이 더욱 우수한 이유는 연산을 정점으로 표현한 방법에 비해 객체를 중복해서 읽는 경우가 적기 때문이다. 연산을 정점으로 표현할 때의 최적 분할 방법도 결국 16개의 프로세서를 사용할 때에는 디스크에 대한 중복 접근으로 인해 성능이 더 나빠짐을 알 수 있다.





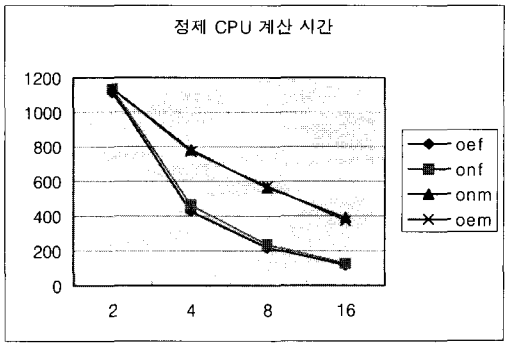


그림 13 L4데이타에 대한 태스크 생성 방법별 정제 속도의 비교(단위 : 초)

6. 결론 및 향후연구

이 논문에서는 다중 공간 조인을 실행할 때 연속적으로 공간 조인을 수행하는 방법보다 이를 다중 공간 여과와 정제로 분리하여 여과를 우선적으로 한꺼번에 수행한 후에 일괄적으로 정제를 수행하는 것이 더욱 우수함을 증명하였다. 정제 단계에서는 후보 객체 테이블에서 레코드 별로 정제를 수행하는 것이 매우 비효율적이며 이를 해결하기 위해 두가지 그래프 생성 방법을 제시하였다. 그래프 생성에 의한 정제는 후보 객체 테이블의 중복률과는 전혀 무관하며 중복률이 높을수록 그래프를 사용하지 않은 방법보다 월등히 우수하였다. 그리고 다중 공간 조인의 병렬화는 여과와 정제 각각에 병렬화를 적용하였는데 성능평가 결과 객체를 정점으로 표현하는 그래프 생성 방법이 객체를 중복해서 읽는 경우가 적으므로 연산을 정점으로 표현하는 그래프 생성 방법보다 우수하였으며, 병렬 정제를 위한 태스크 생성 방법은 단순 분할 방법보다 부분 연결 요소에 의한 중복 최소화 분할 방법이 더욱 우수한 성능을 발휘하였다.

향후연구로는 공간 조인 외에 영역 질의 및 다른 여러 질의가 포함된 더욱 복잡한 공간 경로 표현식의 병렬화 수행 방법을 연구하고, 병렬 수행에서의 비용 예측(cost estimation)을 통해 최종적으로 공간 질의 단계에서의 최적화 병렬 알고리즘을 도출하는 것이다.

참 고 문 헌

[1] M.L. Lo, C.V. Ravishankar, The Design and Implementation of Seeded Trees : An efficient Method for Spatial Joins, IEEE Transactions of Knowledgement and Data Engineering, Vol 10 No 1

pp 136-152, 1998.
[2] T. Brinkhoff, H.P. Kriegel, B. Seeger, Efficient Processing of Spatial Joins Using R-Trees, Proc. ACM SIGMOD Int. Conf. pp 237-246, 1993.
[3] T. Brinkhoff, H.P. Kriegel, B. Seeger, Parallel Processing of Spatial Joins Using R-Trees, Proc. 12th IEEE Data Engineering pp258-265, 1996.
[4] Erik g. Hoel, Hanan Samet, Data-Parallel Spatial Join algorithms, International Conference on Parallel Processing, 1994, pp227-234.
[5] H. Veenhof Processing Multi-Way Spatial Joins, CTIT Ph.D-thesis series no. 97-15, ISBN 90-365-1002-3, September 1997.
[6] W. Hong and M. Stonebraker. Optimization of Parallel query Execution Plans in XPRS, Proceedings of the 1st Conference on Parallel and Distributed Information Systems, pp218-225, December 1991.
[7] D.J. DeWitt and R. Gerber, Multiprocessor Hash-Based Join Algorithms, Proc. 11th Int'l conf. Very large Data Bases, pp. 151-162, August 1985.
[8] H. Lu, K.L. Tan, and M.-C Shan, Hash-Based join Algorihtms for Multiprocessor Computers with Shared Memory Proc. 16th Int'l Conf. Very Large Data Bases, pp. 198-209, August 1990.
[9] J. Richardson, H. Lu, and K. Mikkilineni, Design and Evaluation of Parallel Pipelined Join Algorithms Proc ACM SIGMOD pp. 399-409, May 1987.
[10] D. Schneider and D.J. DeWitt, A Performance Evaluation of Four Parallel Join Algorihtms in a Shared-Nothing Multiprocessor Environment, Proc. ACM SIGMOD pp. 110-121, 1989.
[11] H. Lu, M.-C Shan, K.L Tan, Optimization of Multi-Way Join Queries for Parallel Execution, Proceedings of 17th International Conference on Very Large Data Bases, pp. 549-560. September, 1991.
[12] M.S Chen, P.S. Yu, Optimization of Parallel E1) xecution for Multi-Join Queries, IEEE Transactions on Knowledge and Data Engineering, pp 416-428, June 1996.
[13] A.N. Wilschut, J. Flokstra, P.M.G. Apers Parallel Evaluation of multi-join queries, Proc. ACM SIGMOD, pp. 115-125, 1995.
[14] N. Mamoulis, D. Papadias Integration of Spatial Join Algorithms for Processing Multiple Inputs, Proc. ACM SIGMOD, pp. 1-12, June 1999
[15] D. Papadias, N, Mamoulis, Y. Theodoridis Processing and Optimization of Multiway Spatial Joins Using R-trees, Proceedings of 18th Symposium on Principles of Database Systems, pp.44-55, March 1999.
[16] M. stonebraker, J. Frew, K. Gardels, J. Meredith The Sequoia 2000 Storage Benchmark Proc. ACM SIGMOD, pp. 2-11, May 1993.

류 우 석
1997년 2월 부산대학교 컴퓨터공학과 졸업. 1999년 2월 부산대학교 대학원 컴퓨터공학과 졸업. 1999년 1월 ~ 현재 (주) 아키정보기술 근무. 관심분야는 component GIS, 인터넷 GIS, High Performance GIS.

홍 봉 희
정보과학회논문지 : 데이타베이스 제 27 권 제 1 호 참조