고정 그리드를 이용한 병렬 공간 조인의 태스크 할당에 관한 연구

김 진 덕[†] · 서 영 덕[††] · 홍 봉 희[†††]

요 약

공간 조인은 두개의 데이터 집합으로부터 공간적인 조건을 만족하는 두 객체 쌍의 집합을 구하는 것으로 비용이 매우 큰 연산자이다. 지난 수년동안 공간 조인의 순차 수행 시간은 많이 향상되었지만, 그 응답시간은 사용자의 요구를 만족시키지 못하고 있다. 따라서 최근 병렬 시스템을 이용하여 이러한 문제를 해결하려는 연구가 진행되고 있다. 그렇지만 프로세서의 수가 증가할수록 병렬 처리에 의한 프로세서의 효율성은 급격히 떨어진다. 이것은 병렬 공간 조인을 수행할 경우 순차 공간 조인 보 다 디스크 병목 현상과 메시지 전송 오버헤드가 심하게 발생하기 때문이다. 이 논문에서는 공유 디스크 구조에서 다중 프로세서의 디스크 동시 접근으로 인한 병목 현상을 완화하고, 메시지 전송을 최소화하기 위한 태스크 할당 방법을 제안한다. 제안한 태스크 할당 방법을 두 가지 공간 조인 기법에 각각 적용하여 디스크 접근 횟수와 메시지 전송 횟수의 감소 효과를 실험으로 평가한다. MIMD 구조 및 공유디스크 방식의 병렬 시스템에서의 다양한 실험에서 이 논문에서 제안한 준동적 태스크 할당 방법이 정적 할당과 동적 할당 방법에 비해 우수함을 보였다.

A Study on Task Allocation of Parallel Spatial Joins using Fixed Grids

Jin-Deog Kim[†] · Young-Duk Seo[††] · Bong-Hee Hong[†††]

ABSTRACT

The most expensive spatial operation in spatial databases is a spatial join which computes a combined table of which tuple consists of two tuples of the two tables satisfying a spatial predicate. Although the execution time of sequential processing of a spatial join has been so far considerably improved, the response time is not tolerable because of not meeting the requirements of interactive users. It is usually appropriate to use parallel processing to improve the performance of spatial join processing. However, as the number of processors increases, the efficiency of each processor decreases rapidly because of the disk bottleneck and the overhead of message passing. This paper proposes the method of task allocation to soften the disk bottleneck caused by accessing the shared disk at the same time, and to minimize message passing among processors. In order to evaluate the performance of the proposed method in terms of the number of disk accesses and message passing, we conduct experiments on the two kinds of parallel spatial join algorithms. The experimental tests on the MIMD parallel machine with shared disks show that the proposed semi-dynamic task allocation method outperforms the static and dynamic task allocation methods.

키워드 : 공간데이터베이스(Spatial Database), 병렬공간조인(Parallel Spatial Join), 공간색인(Spatial Index), 고정그리드(Fixed Grid)

1. 서 론

공간 데이터베이스에서 주로 사용되는 공간 연산은 영역 질의와 공간 조인으로서 공간 데이터 처리 시에 많은 연산이 요구되며, 연산의 복잡도(complexity)가 매우 큰 연산자이다. 특히 공간 조인은 많은 처리 시간이 필요한 연산이다. 공간조인은 단일주사(single scan) 질의인 점 질의(point query)나 영역 질의(region query)와는 달리 다중주사(multiple scan) 질의이기 때문에 객체의 수가 증가함에 따라 연산 시간이 급격히 증가한다[2]. 그러므로 빠른 질의 처리를 요구하는 응용에서 공간 조인의 효율적 처리방안이 필요하다. 그래서 공간 색인을 이용한 공간 조인과 병렬 처리에 관한 연구에 많은 관심이 집중되고 있다. 병렬 시스템을 이용한 공간 조인에서 CPU 연산시간의 단축효과는 뛰어나지만 동시 다발적인 디스크 접근으로 인한 디스크 병목 현상과 프로세서 수의 증가에 따른 과도한 메시지 전송 횟수는 전체적인 병렬 공간 조인의 성능 향상에 있어서 걸림돌로 작용한다.

공간 조인을 위한 공간 색인 기법은 크게 단일할당 공간 색인과 다중할당 공간 색인과 같은 두 가지로 분류된다[14]. 단일 할당은 주로 R-tree와 같은 객체 중심의 공간

† 정 회 원 : 동의대학교 컴퓨터공학과 교수
†† 준 회 원 : 부산대학교 대학원 컴퓨터공학과
††† 정 회 원 : 부산대학교 컴퓨터공학과 교수
논문접수 : 2001년 4월 3일, 심사완료 : 2001년 6월 13일

색인이며, 다중할당은 Quad tree와 같은 영역 중심 정규분할 공간색인에 주로 적용된다. 각 색인을 이용한 공간 조인의 특성은 다음과 같다.

첫째, 단일할당 공간 색인을 이용한 공간 조인은 단일 할당, 다중 조인(Single Assign, Multiple Join : ***SAMJ***)[14]의 특징을 가지고 있다. 단일 할당 공간 색인에서 하나의 객체는 단 하나의 버켓에만 할당된다. 이 방법은 공간 객체를 완전히 포함하도록 객체를 중심으로 데이터 공간을 분할하는 비정규(irregular) 영역 겹침 기법으로 R-tree, R*-tree 등의 공간 색인 기법[2, 3]이 대표적인 예이다. 이와 같은 객체 할당인 경우 공간 조인 시 R 데이터 집합(레이어)의 하나의 셀과 S 데이터 집합의 여러 셀간의 조인을 수행해야 한다.

둘째, 다중 할당 공간 색인을 이용한 공간 조인은 다중 할당, 단일 조인(Multiple Assign, Single Join : ***MASJ***)[14]의 특징을 가지고 있다. 다중 할당 공간 색인은 하나의 객체를 여러 버켓에 할당한다. 이 색인은 공간 객체의 위치와 크기와는 독립적으로 데이터 공간을 분할하고 분할 경계선상에 위치하는 공간 객체에 대해서는 객체를 포함하는 모든 버켓에서 각각 공간 객체에 대한 포인터를 갖는다. 이 색인은 데이터 집합 영역을 정규 분할(regular decomposition)한다. 대표적인 공간 색인의 예로는 Quad tree[5]가 있다. 이와 같은 객체 할당인 경우 공간 조인 시 R의 하나의 셀은 같은 지역을 의미하는 S의 오직 하나의 셀과 공간 조인을 수행하면 된다.

공간 조인 기법 SAMJ와 MASJ는 각각 단점이 있다[14]. 일반적으로 단일 할당 색인을 이용한 공간 조인(SAMJ)은 색인이 겹침 영역 기법이므로 다중 조인이 발생하므로 여과 단계의 후보 객체 쌍을 구하기 위한 CPU 연산과 디스크 접근 횟수가 다중 할당 색인을 이용한 공간 조인 보다 많다. 이와 같은 경우 병렬 처리시 CPU 연산 시간은 프로세서의 수에 비례해 단축되지만, 디스크 접근 시간은 프로세서의 수가 증가하면 디스크 병목현상으로 인해 병렬 처리 이득을 보기 어려워진다.

이와는 반대로 다중 할당 색인을 이용한 공간 조인은 객체의 중복이 발생하고 공간 조인의 결과도 중복이 되므로 이를 제거하는 추가 연산이 필요하다. 중복 제거 작업을 위해서는 각 Slave 프로세서간에 많은 메시지 전송이 필요하지만, 프로세서의 수가 증가하면 할수록 처리비용이 증가하게 된다. 그러나 이러한 단점은 병렬 처리시의 태스크 할당 방법에 따라 완화될 수 있다.

병렬 처리에서 연산의 전체 수행 시간은 가장 느린 프로세서에 의해 좌우된다. 그러므로 각 Slave 프로세서의 수행 시간의 편차를 최소화하는 것이 병렬 처리에 의한 이득을 극대화하는 길이다. 이는 각 Slave프로세서에 부과되는 부하의 평준화를 통해 가능해진다. 그렇지만 공간 데이터 베이스 시스템에서는 태스크의 지역성을 최대한 반영하지 못하는 태스크 할당일 경우 전술한 디스크 접근의 중복 및 메시지 전송 횟수가 증가한다.

그래서 이 논문에서는 두 가지 대표적인 공간 조인 기법의 단점을 동시에 해결할 수 있는 태스크 할당 방법을 제시하고자 한다. 이 논문에서 새로 제시하는 준동적(semi-dynamic) 태스크 할당방법은 공간 데이터의 지역성과 동적 부하 평준화를 동시에 고려하여 태스크 중 일부는 서로 인접한 태스크를 그룹으로 묶어 정적(static)으로 할당하고, 나머지 태스크는 태스크의 크기 순으로 동적(dynamic)으로 할당한다. 이 방법은 정적 태스크 할당 방법과 동적 태스크 할당 방법의 장점을 적절히 활용할 수 있다. 그리고 제안한 태스크 할당 방법을 적용하기 위한 두 가지 병렬 공간 조인 기법은 고정 그리드를 기반으로 하였다. 구체적인 병렬 공간 조인 기법으로는 첫째, 단일 할당 고정 그리드를 이용한 병렬 공간 조인(Parallel Spatial Join using Single-Assignment fixed grid : **PSJ_SA**), 둘째, 다중 할당 고정 그리드를 이용한 병렬 공간 조인(Parallel Spatial Join using Multiple-Assignment fixed grid : **PSJ_MA**)이다.

성능 평가를 위한 실험 데이터는 Sequoia 2000 벤치마크 데이터(실제 데이터)[15]를 사용한다. 실험은 두 가지 공간 조인 기법에 다양한 버퍼의 크기, 그리드 해상도, 프로세서 수를 적용하였다. 구체적인 실험평가 항목으로는 다양한 버퍼의 크기별 디스크 접근 횟수 감축 효과, 중복 제거를 위한 메시지 전송회수의 감축 효과 등이다. 실험 평가를 위한 병렬 처리시스템으로서 MIMD 구조 및 공유 디스크(shared disk) 병렬 처리 시스템인 SP2를 이용한다.

이 논문의 구성은 다음과 같다. 2장에서는 공간 조인의 관련연구 및 논문의 연구 방향을 기술하고, 3장에서는 공간 조인의 정의와 특성 및 각 색인에 따른 공간 조인 기법을 설명한다. 4장에서는 태스크의 할당 방법과 부하 평준화에 대해 설명한다. 5장에서는 다양한 실험평가의 결과를 살펴보고, 각 태스크 할당 방법의 장단점을 분석한다. 6장에서 결론을 맺는다.

2. 관련 연구

지금까지의 공간 조인에 관한 연구를 대분류하면 우선 Z-order를 적용한 연구[11], 해쉬 조인에 관한 연구[12], 변환 기법을 활용한 공간 조인에 관한 연구 등 매우 다양하다. 그렇지만 최근에 연구된 대부분의 공간 조인 기법들은 조인의 대상이 되는 두 데이터 집합이 공간 색인화된 경우에 집중되고 있다. 주로 연구된 공간 색인 방법으로는 R-tree 및 R*-tree[2, 3], PMR Quad tree[5], Grid

File, Seeded tree[10], 그리고 Filter tree[8] 등이 있다. 전술한 바와 같이 공간 조인에 관련된 공간 색인은 크게 단일할당 공간 색인[2, 3, 10]과 다중할당 공간 색인[5, 14]으로 대분된다. 또한 공간 데이터 집합 중 한 쪽 또는 두 쪽 모두 공간 색인이 생성되어 있지 않은 경우의 공간 조인 기법에 관한 연구[1, 8, 10]도 활발하게 진행되었다. 이들 연구는 주로 다중 공간 조인의 처리 기법을 제시하고 있다. 그리고 최근 다중 프로세서를 이용한 병렬 공간 조인[3, 5, 14]에 관한 연구도 증가하고 있다. 또한 공간 조인의 성능을 미리 예측하는 비용모델을 제시한 연구[6, 13]도 진행되었다.

다중 프로세서를 이용한 병렬 공간 조인에 대한 연구로써 관련연구 [3, 5, 14, 16, 17]이 있다. 관련연구 [3]에서는 단일 할당 공간 색인 방식인 R*-tree상에서 공유 가상 메모리(shared-virtual memory)를 가지는 병렬 시스템에서의 공간 조인을 위한 방법들을 제시하고 있다. 이 연구에서는 정적 태스크 할당, 정적 라운드로빈 태스크 할당, 동적 태스크 할당으로 나누어 처리하는 방법을 소개하고 있다. 그러나 이들 태스크 할당 방법들은 태스크를 R*-tree의 중간 노드에서 겹치는 한 쌍의 노드를 하나의 태스크로 간주하고, 태스크를 할당하는 문제를 중심으로 연구를 진행했다. 이 때 서로 동일 R-tree 노드를 다른 프로세서에 할당하고 있는 데 이는 디스크 병목 현상을 일으키는 요인이 된다.

관련연구 [5]에서는 단일할당 공간 색인을 이용한 것으로서 비겹침-정규 데이터 공간 분할 방식을 사용하는 PMR Quad tree에서 병렬 조인을 위한 방법에 대해서 다루고 있다. 이 연구에서는SIMD array processor 환경에서 데이터-병렬 모델(data parallel model)을 기초로 한 병렬 공간 조인 방법을 제시하고 있다. 그러나 제시한 실험 환경인 SIMD는 특수 목적 응용에 주로 사용되는 시스템이다. 그러므로 최근 보급이 확대되고 있는 MIMD 환경에서 공간 조인 기법을 제시하는 것이 필요하다. 또한 이 방법은 노드의 병합과정에서 병합된 사분면들의 데이터들을 주기억장치 내에 적재하여 처리한다. 만약 두 트리의 구조 중 한쪽은 트리의 깊이가 매우 크고 반대쪽은 매우 작은 극단적인 경우에는 많은 병합과정이 일어난다. 그러므로 주기억장치 내의 버퍼가 충분하지 않을 경우 여러 번의 페이지 대체(replacement)가 일어나 전체적인 수행속도가 느려진다.

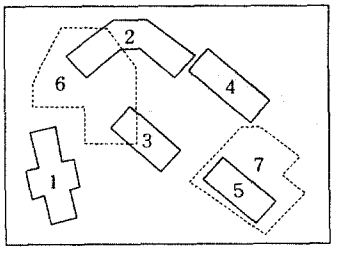
관련 연구 [14]에서는 병렬 공간 조인을 위한 데이터 분할 방법, 즉 단일 할당 공간 분할과 다중 할당 공간 분할의 장단점에 대해 언급하고 있다. 또한 다중 할당 공간 분할 방법을 이용한 공간 조인의 성능 평가를 다양한 프로세서 수와 다양한 그리드 해상도에 대해 적용하였다. 그렇지만 이 연구에서는 단일 할당 공간 분할과 다중 할당 공간 분할 간의 직접적인 성능 평가를 하지 않았고, 실험 또한 병렬 시스템에서 수행된 것이 아니라 단일 프로세서 시스템에서 모의 실험을 통해 성능 평가를 하고 있다. 그리고 공간 인접성을 반영하기 위한 태스크 할당 방법을 다루지 않는다.

관련 연구 [17]에서는 R-tree를 이용한 병렬 공간 조인에서 태스크 할당 방법과 객체 캐싱 방법을 제시하여 성능 향상을 도모하는 방법을 제시하였다. 이 연구에서는 공유 디스크 구조에서 다중 프로세서의 디스크 동시 접근으로 인한 병목 현상을 완화하고, 프로세서간의 메시지 전송을 최소화하기 위한 태스크 생성 방법, 태스크 할당 방법을 제시했다. 또한 성능 저하 요소의 분석과 디스크 접근 시간을 줄이기 위한 객체 캐쉬 방법과 시공간 지역성을 이용한 태스크 생성 및 할당 방법을 제시했다. 그러나 이 연구에서는 정제 단계의 디스크 접근 횟수 감축을 위한 방법으로 버퍼의 크기 및 메시지 전송횟수를 고려하지 않고 있다.

3. 공간 조인 기법

이 논문에서 사용하는 공간 조인[16]이란 두 공간 데이터 집합 R과 S에서 R의 *i*번째 컬럼과 S의 *j*번째 컬럼이 공간 속성일 때 두 객체 집합의 cartesian product중에서 공간적인 조건(spatial predicate)을 만족하는 두 객체 쌍의 객체 식별자(id) 집합을 구하는 것이다[2]. 공간적인 조건으로서는 교차(intersect), 포함(containment) 관계 등이 있다. 이 논문에서 적용하는 공간 조건은 교차 관계이다. 일반적으로 지리 정보 시스템에서는 데이터 집합 R과 S는 지도상의 같은 위치를 나타내는 레이어 들이다.

예를 들어 "오염지역 내에 존재하는 빌딩을 검색하라"라는 예에서 연산에 필요한 객체 집합은 빌딩정보를 나타낸 Building레이어와 오염지역을 나타내는 Pollution레이어이다. (그림 1)에서 실선은 빌딩을 표현하는 데이터 집합이고, 점선은 오염지역을 나타내는 데이터 집합이다. 각 집합은 공간 데이터로서 다각형 정보를 갖고 있다. 그리고 공간적인 조건으로는 교차 조건이 사용되었다. 결국 이 질의의 공간 조인은 빌딩과 오염지역이 교차하는 객체에 대한 쌍을 구하는 것이다. (그림 1)에서 공간 조인의 결과로서 세 개의 객체 쌍들의 집합{(2,6), (3,6), (5,7)}이 생성된다.



(그림 1) 공간 조인 연산의 예

기존의 데이터베이스의 병렬 처리에 관한 연구로서 DIRECT 시스템[4]은 관계형 데이터베이스의 조인을 병렬로 처리하고자 하였다. 이 시스템은 데이터 병렬 처리법을 이용한다. 데이터 분할 방법은 하나의 튜플은 하나의 버켓에 할당되고, 하나의 R 버켓은 단 하나의 S버켓과 조인을 수행하는 단일 할당, 단일 조인(Single Assign, Single Join : SASJ)이라 할 수 있다. 각 R과 S 버켓의 쌍을 태스크라 하면 병렬 수행을 위해 각 태스크가 하나의 프로세서에 할당된다. 데이터 집합 R과 S가 각각 A와 B개의 데이터 페이지로 구성될 때 이론적인 병렬처리 시간 이득은 MAX(A, B)이 된다.

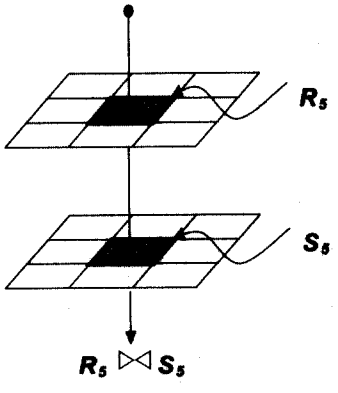
그러나 관계형 데이터베이스 시스템에서 적용되는 병렬 조인 기법을 공간 데이터베이스 시스템에는 그대로 적용하기가 어렵다. 왜냐하면 문자 데이터와는 달리 공간 데이터는 2차원의 공간 상에 존재하므로 순서화가 불가능하고, 공간 데이터가 크기(extent)를 가져 단일 할당, 단일 조인이 불가능하기 때문이다[14]. 그러므로 병렬 처리 공간 조인 연산을 수행하기 위해서는 단일 할당과 다중 조인 또는 다중 할당과 단일 조인 기법이 사용되어야 한다.

이 논문에서 적용하는 공간 조인 기법은 다음과 같다. 단일 할당 고정 그리드를 이용한 병렬 공간 조인과 다중 할당 고정 그리드를 이용한 병렬 공간 조인이 그것들이다. 각 기법을 간단히 PSJ_SA, PSJ_MA로 명명한다.

3.1 PSJ_MA(다중 할당 고정 그리드를 이용한 병렬공간조인)

다중할당 고정 그리드를 공간 색인으로 이용하는 PSJ_MA는 다음과 같은 특성이 있다.

- **다중 할당** : 먼저, 데이터 집합 공간을 일정한 면적을 갖도록 X,Y 좌표축으로 N등분(fixed grid)한다. 각 분할된 버켓을 그리드 셀이라 한다. 이 때 각 그리드 셀들은 균일한 크기(*regular extent*)를 가지며, 각 그리드 셀간에는 겹침이 존재하지 않는다. 그 뒤, 데이터 집합의 객체들을 객체의 일부분이라도 포함하는 여러 그리드 셀에 중복 참조(reference)된다.

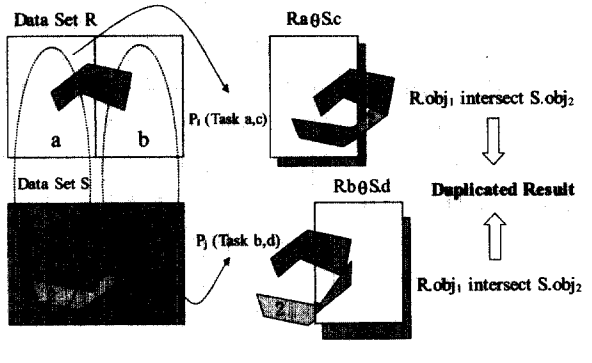


(그림 2) 단일 조인

- **단일 조인** : PSJ_MA에서 하나의 태스크는 (그림 2)처럼 같은 위치의 셀 쌍의 공간 조인 작업을 의미한다. 병렬 수행을 위해 각 태스크가 하나의 프로세서에 할당된다. 프로세서 개수가 P로 유한하다면 평균 n/P개의 태스크를 각 프로세서가 수행해야 하고, 부하평준화 알고리즘이 요구된다. 병렬 처리를 위한 구체적인 태스크의 생성 및 할당 과정은 4장에서 자세히 다룬다.

PSJ_MA는 단일 조인이므로 여과 단계에서 각 그리드 셀은 오직 한번씩만 검색하면 된다. 따라서 공간 색인의 검색 범위가 좁혀지는 장점이 있다. 그러나 이 방식의 단점은 데이터를 중복해서 참조하기 때문에 조인의 결과가 중복된다[14]는 것이다. 결과의 중복이 발생하는 경우의 예를 들면 (그림 3)에서 R의 a셀과 S의 c셀간의 조인 작업이 하나의 태스크이고, b셀과 d셀간의 조인 작업이 하나의 태스크 이다. 태스크(a, c)가 P_i 프로세서에 할당되고, 태스크(b, d)가 P_j에 할당된다면 각 프로세서는 동일한 결과를 반환하게 되어 결과가 중복된다. 이 때 P_i와 P_j가 동일한 프로세서가 아니면 메시지 전송에 의해 중복이 제거되어야 하지만, 동일한 프로세서이면 메시지 전송 없이 간단히 중복 제거된다[14]. 이를 위한 태스크 할당방법은 4장에서 자세히 다룬다.

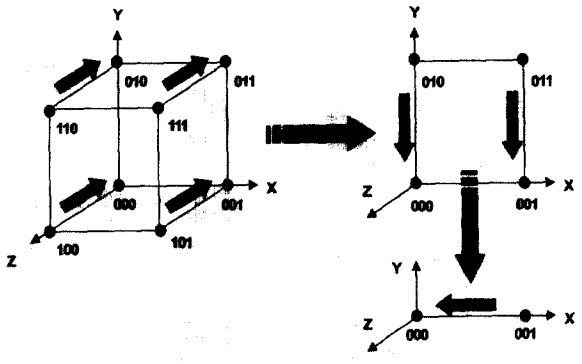
PSJ_MA는 여과 단계를 수행한 후 생성된 후보 객체쌍의 중복을 정제 단계를 수행하기 전에 제거한다. 왜냐하면 일반적으로 정제 단계의 비용이 매우 크기 때문에 중복된 정제 단계를 피하기 위해 정제 단계를 수행하기 전에 중복된 후보 객체쌍을 제거하는 것이다.



(그림 3) 결과의 중복 예제

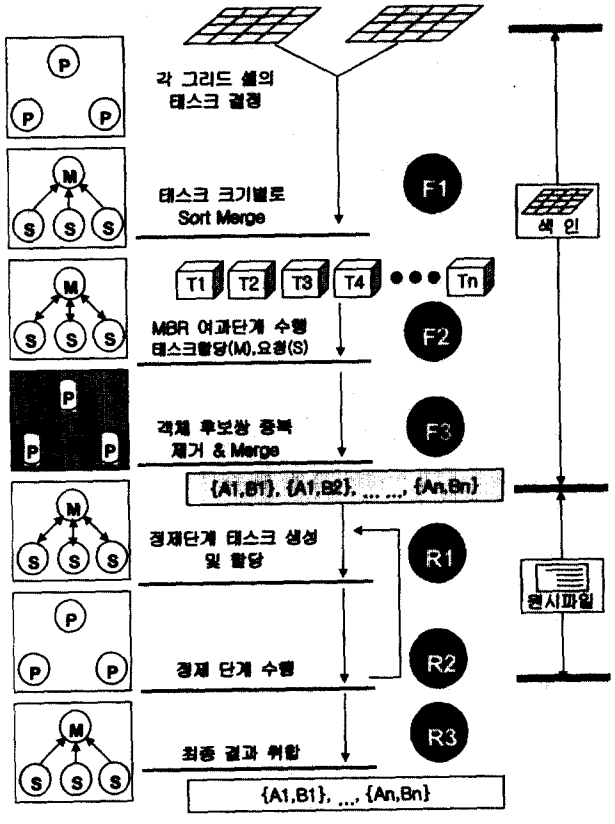
일반적으로 공간 조인의 결과로 나온 객체 식별자의 쌍의 개수가 상당히 크기 때문에 중복제거에 많은 시간이 소요된다. 이 논문에서는 중복 제거를 위해 Hyper-Cube Network형태로 구성된 프로세서들간의 Sort Merge방법[12]을 이용하여 병렬성을 높였다. 이 방법은 프로세서의 개수가 P개일 때 $\text{Log}_2\ P$ 단계 만에 중복이 제거된다. 예를 들어 (그림 4)처럼 8개의 프로세서(0-7번 프로세서)가 공간 조인

을 위해 사용되었다면 3단계만에 중복이 제거된다.



(그림 4) Sort Merge

최초 각 프로세서에서 후보 객체쌍들의 중복을 제거하면서 정렬을 한다. 그 뒤 1단계에서 Z축을 기준으로 프로세서들이 후보 객체쌍들을 전송하면서 중복을 제거한다. 2, 3단계 역시 각각 Y,X축을 기준으로 Merge를 하게 된다. 최종적으로 0번 프로세서에는 중복이 완전히 제거된 후보 객체쌍만 남게 된다.



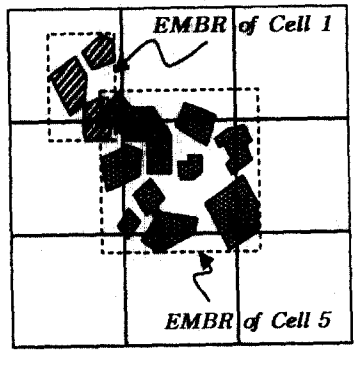
(그림 5) PSJ_MA의 수행 단계

(그림 5)는 PSJ_MA의 각 수행 단계를 그림으로 도식화한 것이다. 이 기법은 병렬 여과 (F1~F3)및 병렬 정제(R1~R3)의 2단계로 수행된다. 각 단계는 또한 태스크 생성, 태스크 수행, 결과 취합의 3단계로 세분화 된다. (그림 5)의 왼쪽 상자는 각 프로세서의 상태를 나타내는 것으로서, P는 모든 프로세서가 독립적으로 동작하는 상태이고, M은 Master 프로세서, S는 Slave 프로세서 상태를 나타낸다. 프로세서 사이의 화살표는 메시지의 전송상태를 표시한 것이다. F3상태의 메시지 전송과정을 살펴보면, 중복 제거를 위해 각 프로세서간의 메시지 전송이 있음을 그림으로 보여준다. 그리고 (그림 5)의 오른쪽은 각 단계의 디스크 검색 대상을 나타낸 것이다. PSJ_MA는 여과 단계인 F1~F3의 상태에서는 색인 파일만을 읽고, 정제 단계인 R1~R3는 원시 데이터 파일만을 읽는다. 즉, 여과 과정과 정제단계가 완전히 분리되어 있다. PSJ_MA의 여과 단계에서 디스크 접근 횟수를 줄이기 위해 버퍼를 사용한다. 버퍼의 한 슬롯(slot)의 크기는 1Kbytes이고 전체 버퍼 크기는 2K부터 256K까지 다양하게 적용한다. 그리고 페이지 교체(page replacement)는 LRU(Least Recently Used) 방식을 택한다.

3.2 PSJ_SA(단일 할당 고정 그리드를 이용한 병렬공간조인)

단일 할당 고정 그리드를 공간 색인으로 이용하는 PSJ_SA 기법은 다음과 같은 특성이 있다.

- **단일 할당** : 먼저, 다중 할당 고정 그리드처럼 데이터 집합 공간을 일정한 면적을 갖도록 X,Y 좌표축으로 등분한다. 그러므로 최초의 각 그리드 셀들은 균일한 크기를 가지며, 각 그리드 셀간에는 겹침이 존재하지 않는다. 그 뒤, 데이터 집합의 객체들을 그 객체의 MBR의 중심점을 포함하는 그리드 셀에만 포함시킨다. 즉 모든 객체는 각각 오직 하나의 그리드 셀에만 포함된다. 단일 할당 고정 그리드에서 모든 객체가 각 그리드 셀에 할당된 후 각 셀의 범위는 포함된 객체들을 모두 감싸는 최소 경계사각형으로 재조정된다. 이 논문에서는 재조정된 범위를 그리드 셀의 확장 MBR (*EMBR* : Extended MBR)이라 한다. (그림 6)에서 점선 사각형은 셀 1, 5의 확장 MBR을 각각 나타낸 것으로서 각 셀은 불규칙적인 크기(*irregular extents*)를

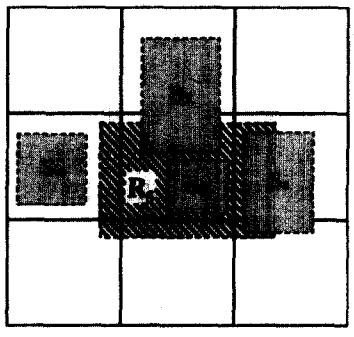


(그림 6) 단일 할당 고정 그리드와 확장 MBR

갖고 셀간의 겹침이 발생한다. EMBR은 단일 할당 고정 그리드에서만 정의된다.

- **다중 조인** : 단일 할당 고정 그리드의 각 셀은 불규칙한 범위를 가지므로, 하나의 셀 R_i의 EMBR은 다수개의 셀S_j의 EMBR과 겹칠 수 있다. 그러므로 PSJ_SA는 하나의 셀 R_i에 대해 다수개의 셀 S_j와 다중 조인을 수행해야 한다. 다시말해, 하나의 셀 Ri와 여러 개의 대응 그리드 간의 조인 작업이 하나의 태스크가 된다. (그림 7)에서 3개의 조인쌍(join pair) {R5,S2}, {R5,S5}, {R5,S6}이 하나의 태스크로 정의되며, 한 프로세서에서 수행된다.

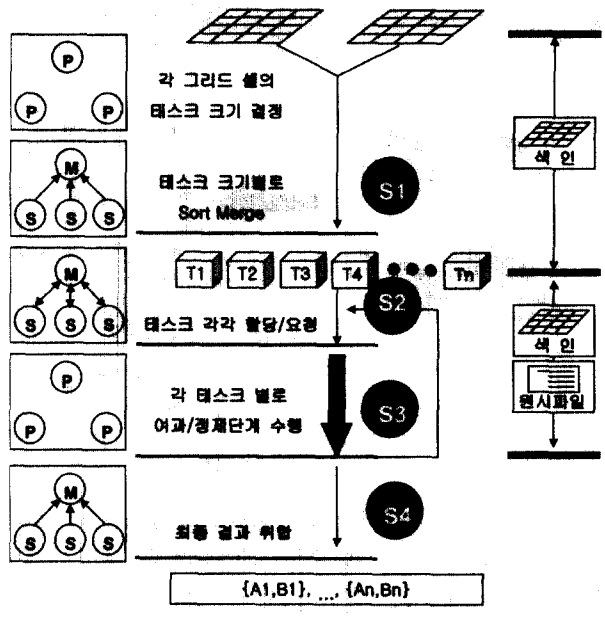
이 논문에서는 한 셀 R_i의 EMBR과 겹치는 EMBR을 가진 모든 S의 셀 들을 대응 그리드(*corresponding grid*)라 한다. (그림 7)에서 R_5의 EMBR은 S_2, S_5, S_5의 EMBR과 겹친다. 그러므로 (그림 7)에서 S_2, S_5, S_6는 R_5의 대응 그리드들이다. (그림 7)에서 데이터 집합 R을 기준으로 프로세서를 할당할 때 R의 5번 셀에 하나의 프로세서가 할당되면 S의 3개의 셀들과 차례대로 여과, 정제단계를 수행한다.



(그림 7) 단일 할당 고정 그리드 대응 그리드 결정

PSJ_SA의 장점으로는 공간 조인의 결과값이 중복되지 않으므로 중복을 제거하기위한 작업이 불필요하다는 것이다. 반면 하나의 셀이 다수의 대응 그리드를 가진다. 그래서 하나의 셀에 대해 두 번 이상 디스크 검색이 이루어질 수 있어 보다 많은 공간 색인 검색이 요구되고 MBR 여과 단계의 CPU 연산량이 증가하는 단점이 있다. 이러한 경우 CPU 연산량은 다중 프로세서를 이용함으로써 해결되지만, 디스크 접근 횟수의 증가는 병목 현상을 초래하는 경향이 있다.

(그림 8)은 PSJ_SA의 각 수행 단계를 그림으로 도식화한 것이다. 이 기법은 PSJ_MA와 달리 하나의 태스크 내에서 병렬 여과와 병렬 정제단계를 연속적으로 수행한다. 구체적인 수행단계는 태스크 생성(S1), 태스크 할당(S2), 대응 그리드 결정 및 태스크 수행(S3), 결과 취합(S4)으로 나눠진다.



(그림 8) 단일할당-다중조인 수행 단계

4. 태스크 생성, 할당 및 수행

병렬 처리에서 연산의 전체 수행 시간은 가장 느린 프로세서에 의해 좌우된다. 그러므로 각 Slave 프로세서의 수행 시간의 편차를 최소화 하는 것이 병렬 처리에 의한 이득을 극대화하는 길이다. 이는 각 Slave프로세서에 부과되는 부하의 평준화를 통해 가능해진다. 그래서 이 논문에서는 부하 평준화(load balancing)를 위해 다음과 같은 3가지 태스크 할당(정적, 동적, 준동적) 방법을 도입한다. 특히 새롭게 제시한 준동적 태스크 할당 방법은 정적 할당 방법과 동적 할당 방법의 장점을 적절히 활용할 수 있다.

4.1 정적 태스크 할당

부하 평준화를 위해 각 그리드 셀의 작업량을 미리 추정하여 태스크를 균등하게 분배한 뒤 실행 전에 한꺼번에 각 프로세서에 태스크들을 할당하는 정적 태스크 할당 방법을 고려해 볼 수 있다. (그림 9)는 태스크가 11개이고 프로세서가 4개일 때의 정적 태스크 할당방법을 보여주고 있다. 구체적인 수행 단계는 다음과 같다.

1) 프로세서는 미리 각 태스크들의 부하 크기를 추정하고, 크기별로 내림차순하여 리스트(*main task list*)로 구성한다. 각 태스크의 부하 크기는 색인 파일의 헤더 정보(각 셀의 객체 개수)를 기초로 추정이 가능하며 그 수식은 아래와 같다.

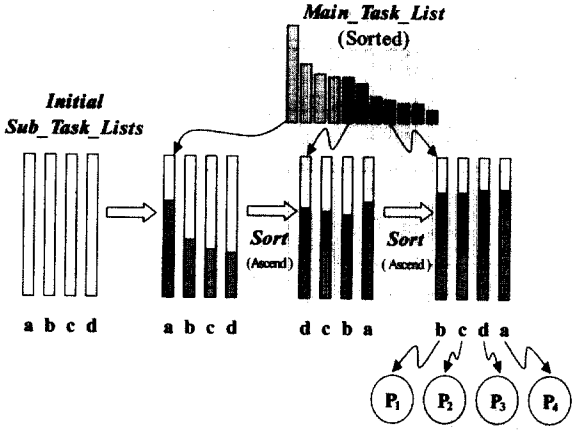
$$T_i = \sum_{j=1}^{k} |R_i| * |S_j|$$

Ti : workload of a task i (i = 1~n)
k : the number of corresponding grids
|Ri| : the number of objects of grid cell Ri
|Sj| : the number of objects of grid cell Sj
(where Sj are the corresponding grid of Ri)

2) Master 프로세서는 *main task list*로부터 태스크를 순서대로 각 *sub task list*에 삽입한다. (그림 9)의 '*a*', '*b*', '*c*', 그리고 '*d*'가 각각 *sub task list*들이다. *sub task list*의 개수는 프로세서의 개수와 동일하다. 그렇지만 다음과 같은 경우는 새로운 태스크를 *sub task list*에 삽입하지 아니한다. (그림 9)의 3번째 단계처럼 만약 한 *sub task list*의 삽입전의 부하 크기 합이 다른 *sub task list*의 삽입후의 부하 크기 합보다 크다면 삽입을 생략한다.

3) 각 *sub task lists* 들을 부하 크기 합에 따라 오름차순으로 정렬한다.

4) *main task list*의 모든 task들이 삽입되었다면, Master 프로세서는 각 *sub task list*를 Slave 프로세서에게 하나씩 할당한다. 그렇지 않다면, Master 프로세서는 단계 2부터 다시 진행한다.



(그림 9) 정적 태스크 할당

비록 정적 태스크 할당 방법은 각 프로세서에 태스크의 부하 크기가 공평하게 분배되었다 할지라도, 각 프로세서에서의 수행 종료 시간은 많은 차이가 있을 것이다. 왜냐하면 공간 데이터베이스 시스템에서 다각형의 점의 개수가 매우 가변적이며, 같은 크기라도 조건 검사 연산의 시간 또한 그 위치에 따라 크게 차이가 있기 때문이다. 또한 현실적으로 각 프로세서마다 동일한 성능을 보이지 않으므로 정적 부하 평준화로 태스크를 할당하면 각 Slave 프로세서의 태스크 수행종료 시간에서 많은 차이를 보이게 된다.

4.2 동적 태스크 할당

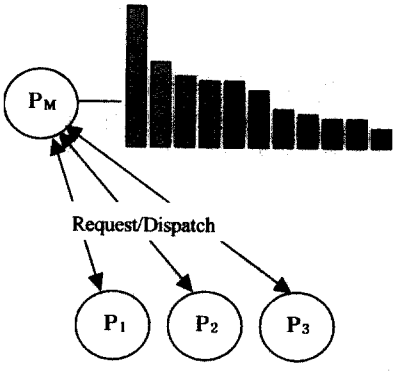
정적 태스크 할당 방법의 문제를 해결하기 위해 동적 태스크 할당 방법을 고려해 볼 수 있다. 그래서 이 논문에서는 태스크의 크기를 고려하여 동적으로 부하 평준화를 이루는 동적 태스크 할당 방법을 도입하였다. 구체적인 수행 단계는 다음과 같다.

1) 정적 태스크 할당 방법의 단계 1과 동일
2) 최초 Master는 각 Slave 프로세서에게 *task list*의 태스크를 순서대로 하나씩 할당한다. Slave 프로세서가 주어진 태스크를 종료함과 동시에 Master 프로세서에게 새로운 태스크를 요구한다. 요구를 받은 Master 프로세서는 *task list*의 다음 task를 할당한다.

동적 태스크 할당의 경우 각 프로세서의 수행 종료 시간의 거의 일치하게 되어 전체적인 수행 시간은 단축될 수 있다. 이것을 의사코드(pseudo code)로 나타내면 아래와 같다. 동적 태스크 할당 방법은 (알고리즘 1)에서 보는 바와 같이 Master 프로세서와 Slave 프로세서간에는 request/dispatch를 위한 많은 메시지 전송이 필요하다. (그림 10)은 동적 태스크 할당 방법의 request/dispatch과정을 그림으로 나타낸 것이다. P_m은 Master 프로세서를 의미하고, P_i는 각 Slave 프로세서를 의미한다.

```
IF( MASTER PROCESSOR )
{     Calculate the Size of each task ;
      Task_List = Sort Task and make LIST ;
      For( i = 1~#of Task)
      {
            Receive_Messge(ANY_SLAVE, PROC_ID) ;
            Send_Message(PROC_ID, Task_List[i++]) ;
      }
      For(i=2~N)Send_Message(PROC[i],NULL) ;
}
IF( SLAVE PROCESSOR )
{     While(1)
      {
            Send_Message(MASTER, PROC_ID) ;
                  /* Request Task */
            Receive_Message(MASTER, task) ;
            If( task = NULL ) break ;
            Results += Execute_Task
}     }
```

(알고리즘 1) 동적 태스크 할당



(그림 10) 동적 태스크 할당

4.3 준동적 태스크 할당

전술한 동적 태스크 할당 방법은 단지 태스크의 크기

만을 고려하였다. 그래서 각 태스크의 지역성을 전혀 반영하지 못해 인접한 두개의 태스크가 서로 다른 프로세서에 할당되어 수행될 가능성이 매우 높다. 그 결과 PSJ_SA의 경우 중복된 디스크 입출력이 발생할 수 있고, PSJ_MA의 경우 결과의 중복이 프로세서간에 일어날 확률이 커져 메시지 전송량이 증가한다. 그러므로 지역적으로 인접한 두개 이상의 태스크는 가능한 하나의 프로세서에서 수행되는 것이 바람직하다. 그래서 이 논문에서는 태스크의 크기와 인접성을 고려한 태스크 할당 방법을 새로 제시한다.

전체 태스크를 크게 두 부분으로 나누어 한 부분을 먼저 정적으로 인접성을 고려하여 태스크를 할당한다. 그 뒤 나머지 부분은 동적인 부하 평준화 방법으로 크기를 고려하여 태스크를 할당한다. 이를 이 논문에서는 준동적 태스크 할당 방법이라 정의한다.

준동적 태스크 할당 방법은 인접성을 고려한 정적 태스크 할당으로 중복 디스크 검색 및 과도한 메시지 전송 등을 막을 수 있고, 동적 태스크 할당으로 부하 평준화가 용이해 진다는 점을 이용하는 것이다. 정적 태스크 할당의 단위는 서로 인접한 n개의 태스크 그룹이고, 동적 태스크 할당의 단위는 단순 태스크이다. 인접도의 척도는 Hilbert Curve[9]의 차이값이다.

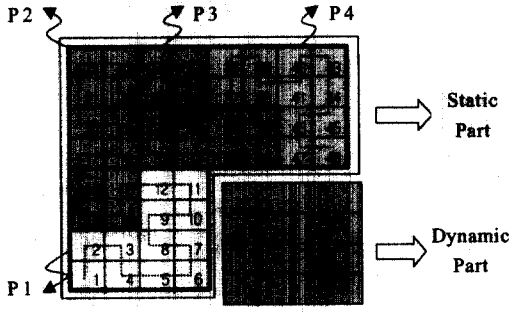
이론적으로는 정적 할당을 위한 태스크의 비율을 최대한 높이는 것이 성능 향상에는 좋지만 일정 비율을 초과할 경우 부하 평준화가 이루어지지 않아 전체적인 수행 성능이 오히려 떨어질 수 있다. 정적 할당을 위한 태스크의 비율과 동적 할당을 위한 태스크의 비율은 데이터의 특성과 프로세서의 수에 따라 달라진다. 특히 데이터의 특성 중 분포상태와 많은 관련이 있다. 데이터의 분포상태가 거의 균일한 경우에는 정적 할당을 위한 태스크의 비율이 상대적으로 높아진다. 반면, 데이터의 분포 상태의 Skewed일 경우에는 그 비율을 줄여야 한다. <표 1>은 프로세서의 개수가 8개일 때 데이터의 분포 특성에 따른 정적 태스크 할당 비율을 실험을 통해 확인한 것이다. 분포 상태는 데이터의 밀도 표면(Density Surface)을 통해 확인할 수 있다. 프로세서의 개수가 증가하면 정적 태스크 할당 비율이 높아지고, 프로세서의 개수가 감소하면 정적 태스크 할당 비율 또한 감소한다.

〈표 1〉 정적 태스크 할당의 비율

	정적 태스크 할당 비율
균 일 분 포	85%
Gaussian 분 포	80%
불균일 분포(실제 데이터)	70%

(그림 11)의 예를 이용하여 준동적 태스크 할당 방법을 설명하고자 한다. (그림 11)에서 태스크의 3/4(64개의 태스크 중 48개)을 정적으로 태스크를 할당하고, 나머지는 동적으로 태스크를 할당한다고 가정하자. 그리고 프로세서의 수가 4개라고 가정하면, 정적 태스크 할당을 위한 태스크 그룹은 각각 12개의 단순태스크로 구성된다. 먼저, (그림 11)에 나타나듯 Hilbert Curve 번호 1부터 12까지는 프로세서 1에 한꺼번에 할당되고, 13~24는 프로세서 2에, 25~26은 프로세서 3에, 37~48은 프로세서 4에 각각 할당된다.

그렇지만 정적 부하 평준화이므로 각 Slave의 수행 완료 시간은 일정하지 않다. 이를 보완하기 위해 (그림 11)의 오른쪽 하단에 있는 전체 태스크 중 1/4을 동적 부하 평준화 방법으로 태스크를 할당하게 된다. 즉, 어느 Slave 프로세서라도 할당된 그룹 태스크를 완료한 프로세서부터 동적 부하 평준화 방법으로 태스크를 할당 받게 되므로 최종 종료 시점은 거의 일치하게 된다. 즉, 정적 부하 수행시간과 동적 부하 수행시간은 제각기 다르지만 각 프로세서의 전체 수행 종료 시간은 거의 일치한다.



(그림 11) 준동적 태스크 할당

5. 실험 평가

이 논문에서는 IBM SP2를 이용하여 전술한 병렬 공간 조인 기법에 따른 태스크 할당 방법의 성능을 평가한다. 실험 데이터로 실제 데이터 및 인위 데이터의 두 가지 유형을 사용하고, 프로세서의 수는 16개까지, 그리드 해상도는 16부터 256까지, 버퍼의 크기는 2Kbytes부터 256Kbytes까지 다양하게 실험하였다. 이 논문에서 사용하는 실험 데이터는 Sequoia 2000 Benchmark 데이터[15]로서 실제 데이터(real data)이고 불균일 분포 상태의 다각형 정보이다. 각 다각형을 구성하는 점들의 좌표 값까지 가지고 있다. <표 2>에 각 데이터의 특성을 정리하였다.

〈표 2〉 Sequoia 데이터의 특성

#of obj.(R)	#of obj.(S)	Density(R)	Density(S)
41814	37793	0.39	0.42

5.1 페이지 단위 디스크 검색 횟수

일반적으로 하나의 태스크를 구성하는 R_i와 S_j 셀들은

한 개 이상의 페이지를 갖는다. 공간 조인이 다중 주사 질의 이므로 하나의 페이지를 여러 번 읽어야 한다. 그러므로 버퍼가 있다면 디스크 접근 횟수를 줄일 수 있다. 이와 같이 하나의 태스크를 수행할 때 발생하는 버퍼 효과를 이 논문에서는 **지역 버퍼 효과**(local buffer effect)라 정의한다. 태스크 할당 방법에 관계없이 지역 버퍼 효과를 얻을 수 있다.

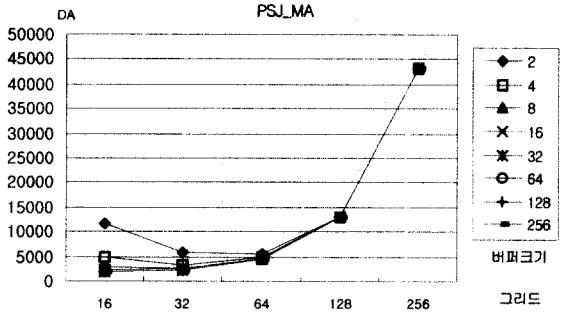
PSJ_SA인 경우 대응 그리드가 존재하고, 이웃한 두 태스크의 대응 그리드들간에는 공통되는 그리드 셀이 존재한다. 그러므로 이전에 수행했던 태스크들에 의해 버퍼 hit되는 경우가 발생하고, 이로 인해 디스크 접근 비용의 감소로 이어진다. 이와 같이 태스크들간의 지역성에 의해 발생하는 버퍼 효과를 **전역 버퍼 효과**(global buffer effect)라 정의한다.

PSJ_MA는 이전 태스크의 수행으로 얻어지는 전역버퍼 효과는 없다. 왜냐하면, PSJ_MA는 이웃 그리드 셀과의 겹침이 없고 연관성이 전혀 없기 때문이다. 반면에 PSJ_SA에 대해 지역성을 고려한 준동적 태스크 할당 방법을 적용할 경우, 정적으로 부여되는 태스크들 간에는 지역성이 존재한다. 이는 PSJ_SA인 경우 전역 버퍼 효과로 인해 디스크 접근 비용의 감소로 이어진다.

(그림 12)는 PSJ_MA의 디스크 접근 횟수를 버퍼별로 그래프화 한 것이다. (그림 12)는 다음과 같은 두 가지 사항으로 요약할 수 있다.

우선, 태스크 할당 방법에 관계 없이 디스크 검색 횟수는 일치한다. 즉, 태스크의 지역성을 반영하는 준동적 태스크 할당 방법이라 할 지라도 전역 버퍼 효과가 전혀 없음을 보여준다. 이것은 PSJ_MA가 단일 조인의 특성을 가지므로 주변 그리드와의 연관성이 전혀 없는 데이터 병렬(Data Parallel)방식이기 때문이다.

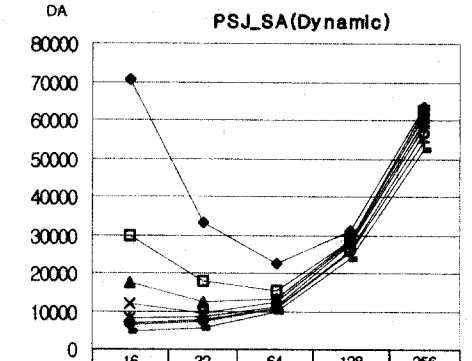
둘째, 그리드 해상도가 낮을 경우 지역 버퍼 효과에 의해 버퍼의 크기가 증가함에 따라 디스크 접근 횟수가 상당히 감소함을 보여주고 있다. 반면, 그리드 해상도가 256일 경우 버퍼의 효과가 크지 않았다. 이것은 그리드 해상도가 낮을 때 하나의 그리드 셀은 여러 개의 디스크 페이지를 차지하기 때문이다. 그리고 그리드 해상도가 높을 때는 대부분 하나의 그리드 셀이 한 개의 디스크 페이지로 구성되기 때문에 지역버퍼 효과는 거의 없음을 알 수 있다.



(그림 12) PSJ_MA의 디스크 접근 횟수

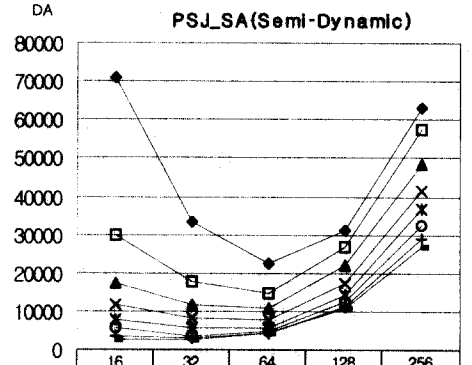
(그림 13)은 PSJ_SA에 각각 동적 태스크 할당 방법과 준동적 태스크 할당 방법을 적용했을 경우의 디스크 접근 횟수를 버퍼별로 그래프화 한 것이다. (그림 13)은 다음과 같은 두 가지 사항으로 요약할 수 있다.

우선, (그림 13)의 (a), (b) 그래프 모두 버퍼의 크기가 클수록 디스크 접근 횟수가 상당히 감소함을 알 수 있다. PSJ_MA와 같이 그리드 해상도가 낮을 경우 디스크 접근 횟수의 감축 효과가 더 크다. 이것은 PSJ_SA가 다중조인 방식으로서 주변 그리드 셀과의 연관성이 존재하므로 버퍼



	16	32	64	128	256
2	70763	33111	22447	31218	63572
4	29673	17664	15493	28715	62538
8	17268	12233	13046	28358	61146
16	11935	9291	12311	27932	59693
32	8107	8342	10940	27447	58202
64	6839	7719	10762	26001	56525
128	6188	7372	10502	25625	54655
256	4762	5526	9830	24012	52425

(a) 동적 태스크 할당



	16	32	64	128	256
2	70760	33105	22421	31083	63208
4	29662	17553	14911	26803	57604
8	17205	11857	11013	22140	48310
16	11794	8188	7976	17488	41674
32	7660	5497	5714	14476	36812
64	5451	3374	4871	12479	32557
128	3546	2844	4514	11446	29034
256	2675	2701	4424	10865	26903

(b) 준동적 태스크 할당

(그림 13) PSJ_SA의 디스크 접근 횟수

의 크기와 전역 버퍼의 효과는 비례하고, 그리드 해상도가 낮을 경우의 디스크 접근 횟수의 감축효과는 지역 버퍼 효과 때문이다.

둘째, 동적 태스크 할당 방법에 비해 이 논문에서 제안한 준동적 태스크 할당 방법을 적용하면 상당한 디스크 접근 횟수의 감축 효과가 있다. 이는 준동적 태스크 할당 방법이 태스크의 지역성을 반영함으로써 전역 버퍼 효과가 나타나기 때문이다. 이러한 전역 버퍼 효과는 그리드 해상도가 높을 경우 보다 많이 나타난다. (그림 13)에서 (a)와 (b)의 차이값은 모두 전역 버퍼 효과이다. 그리고 준동적 태스크 할당 방법은 상대적으로 지역 버퍼 효과보다 전역 버퍼의 효과가 극대화됨을 알 수 있다.

PSJ_MA의 실험 결과(그림 12)와 PSJ_SA의 실험 결과(그림 13)를 비교하면, 전반적으로 PSJ_MA의 디스크 접근 횟수가 PSJ_SA의 그것보다 적지만 그리드 해상도가 매우 높고, 버퍼의 크기가 매우 클 때 준동적 태스크 할당 방법을 적용하면 PSJ_SA의 디스크 접근 횟수가 오히려 적다. 그 이유는 Sequoia 데이터의 밀도가 커, PSJ_MA의 경우 그리드 해상도가 높아지면 객체의 중복이 매우 많아지기 때문이다. 또 다른 이유로는 그리드 해상도가 256일 때 PSJ_SA의 준동적 태스크 할당은 지역성을 근거로 한 전역 버퍼 효과가 매우 크기 때문이다. 그러나 한가지 주의할 것은 위와 같은 경우 PSJ_SA의 디스크 접근 횟수가 PSJ_MA의 그것보다 적다고 해서 여과 시간에서도 그대로 나타나는 것은 아니라는 점이다. 여과 시간은 MBR 비교 연산 횟수, 조인 그리드 쌍의 개수, 디스크 Seek 횟수 등 여러 가지 변수의 조합이기 때문이다.

그리고 (그림 12)와 (그림 13)에서 공통적으로 버퍼의 크기가 클 경우 그리드 해상도 16에서 가장 적은 디스크 접근 횟수를 기록하지만, 최고의 성능을 나타내는 최적의 그리드 해상도는 32이다. 이는 전술한 바와 같이 전체 수행시간은 MBR 비교 연산 횟수, 조인 그리드 쌍의 개수, 디스크 Seek 횟수, 메시지 전송 횟수 등 여러 가지 변수의 조합이기 때문이다. 최적의 그리드 해상도는 공간 활용률과도 관련이 있다.

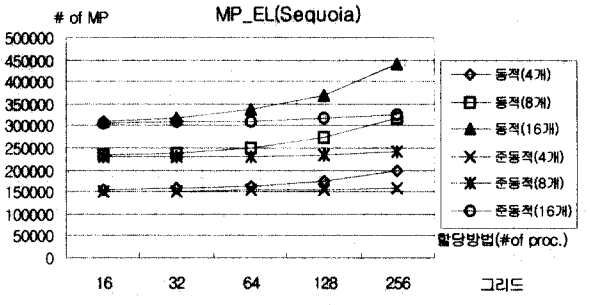
5.2 후보 객체쌍의 중복 제거를 위한 메시지 전송 횟수

후보 객체쌍의 중복 제거 과정은 PSJ_MA에서만 존재한다. (그림 14)는 중복 제거를 위한 메시지 전송횟수를 프로세서 개수와 그리드 해상도를 기준으로 도식화한 것이다. (그림 14)는 다음과 같은 세가지 사항으로 요약된다.

첫째, 이 논문에서 제안한 준동적 태스크 할당 방법은 그리드 해상도에 관계없이 메시지 전송횟수가 거의 일정하며, 그리드 해상도가 높을 경우 동적 태스크 할당 방법에 비해 메시지 전송 횟수가 현격히 줄어듦을 알 수 있다. 이것은 준동적 태스크 할당 방법을 적용하면 중복되는 후보 객체쌍이 프로세서 내에서 거의 제거되기 때문이다. 반면, 그리드 해상도가 높을 경우 동적 태스크 할당에서 이웃한 그리드 셀이 서로 다른 프로세서에 할당될 확률이 높아지기 때문이다.

둘째, 그리드 해상도가 매우 낮을 경우 두 태스크 할당 방법간의 성능차이는 거의 없다. 왜냐하면 그리드 해상도가 낮을 경우 객체의 중복이 거의 없고, 이로 인해 결과의 중복 자체도 거의 없으므로 메시지 전송 횟수에 있어서의 차이가 거의 없기 때문이다. 실험 결과 그리드 해상도가 16일 경우에는 객체의 중복률이 5%정도이지만 그리드 해상도가 256일 경우에는 150%에 달한다.

셋째, 프로세서의 개수가 증가할수록 메시지 전송량은 증가한다. 이것은 프로세서 개수가 증가할수록 중복되는 후보 객체쌍이 분산될 확률이 높아지고, 또한 Sort Merge 방법의 단계가 증가하기 때문이다. 전술한 바와 같이 프로세서의 개수가 P일 때 $\text{Log}_2 P$ 단계가 필요하다.



(그림 14) 후보 객체쌍의 중복 제거를 위한 메시지 전송 횟수

5.3 태스크 할당 방법의 성능 비교

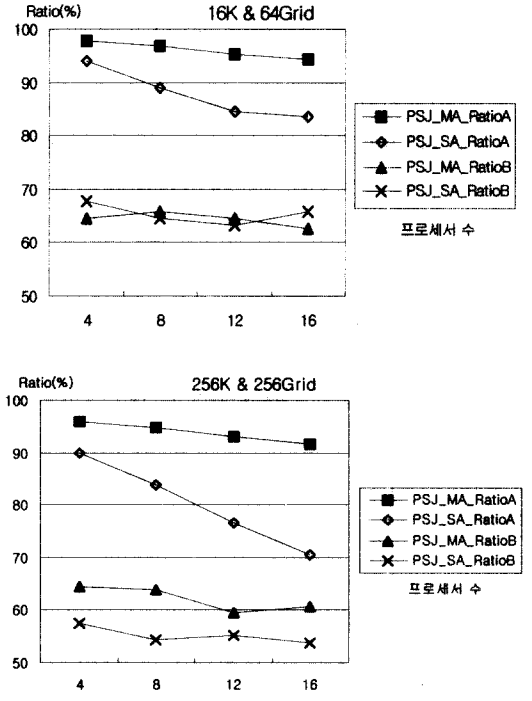
(그림 15)의 (a), (b)는 전술한 3가지 태스크 할당 방법(정적, 동적, 준동적)의 성능 차이를 간단히 나타낸 것이다. 아래의 *RatioA*는 동적 태스크할당 대비 준동적 태스크 할당 방법의 성능을 나타내기 위한 비율이고, *RatioB*는 정적 태스크할당 대비 준동적 태스크 할당 방법의 성능을 나타내기 위한 비율이다. (a)그래프는 그리드 해상도가 64이고, 버퍼의 크기는 16Kbytes일 때의 성능 비교 그래프이며, (b)는 그리드 해상도가 256이고, 버퍼의 크기가 256.Kbytes일 때의 성능 비교 그래프이다.

$$RatioA = \frac{(\text{exec.time at semi} - \text{dynamic allocation})}{(\text{exec.time at dynamic allocation})} \text{ X } 100$$

$$RatioB = \frac{(\text{exec.time at semi} - \text{dynamic allocation})}{(\text{exec.time at dynamic allocation})} \text{ X } 100$$

(그림 15)의 Y축의 모든 값들은 100이하이다. 즉, 준동적 태스크 할당 방법이 항상 동적 또는 정적 태스크 할

당 방법에 비해 좋은 성능을 보임을 의미한다. 또한 각 그래프에서 프로세서의 개수가 많을수록 준동적 태스크 할당 방법이 좋은 성능을 보인다. 그리고 (a)와 (b)를 비교 종합하면 준동적 태스크 할당 방법은 버퍼의 크기가 크고, 그리드 해상도가 높을수록 보다 좋은 성능을 보임을 알 수 있다.

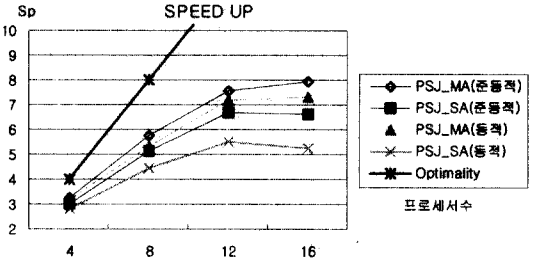


(그림 15) 태스크 할당 방법의 성능 평가

그리고 (그림 15)의 *RatioA*에서 PSJ_SA가 준동적 태스크 할당 방법을 사용할 경우 PSJ_MA의 그것보다 많은 시간 단축 효과가 있었다. 이는 PSJ_SA의 디스크의 중복 검색을 줄이는 효과가 PSJ_MA의 메시지 전송 오버헤드를 줄이는 효과 보다 더 크기 때문이다. 특히 프로세서의 개수가 많을수록 발생하기 쉬운 디스크 병목 현상을 준동적 태스크 할당 방법이 완화시키는 역할을 한다. 이는 (그림 13)에 나타난 바와 같이 준동적 태스크 할당 방법이 디스크 접근 횟수를 감소시키고, 디스크 접근 횟수의 감소는 디스크 병목현상의 완화로 이어지기 때문이다.

반면, 정적 태스크 할당방법의 수행시간은 그 자체에 많은 편차가 발생하여 (그림 15)에서 *RatioB*의 그래프상의 수치는 큰 의미를 갖지는 않지만, 준동적 태스크 할당 방법이 정적 태스크 할당 방법에 비해 월등히 좋은 성능을 보임을 알 수 있다. 실제로 정적 태스크 할당의 수행 시간은 상황에 따라 엄청난 차이를 보였다.

(그림 16)은 Sequoia 데이터를 이용하여 병렬 공간 조인을 수행했을 경우의 병렬 처리에 의한 Speed Up을 정리한 것이다. 병렬 처리에 의한 Speed Up은 $S_p = \frac{T_1}{T_p}$으로 정의된다. 여기서 T_1은 순차 수행시의 실행 시간이고, T_P는 P개의 프로세서로 병렬 처리시의 실행시간을 의미한다. 태스크 할당은 동적 태스크 할당과 준동적 태스크 할당 방법을 각각 적용하였으며, 그리드 해상도는 64이고 64Kbytes의 버퍼를 사용했을 경우이다. (그림 16)에서 병렬 처리에 의한 Speed Up은 프로세서가 증가함에 따라 꾸준히 증가하는 추세를 보이다가 프로세서가 12개 이상이면 거의 포화 상태에 이른다. 그리고 PSJ_MA가 병렬 처리에 의한 Speed Up 효과가 큰 반면, 준동적 태스크 할당 방법의 의한 성능향상은 전술한 바와 같이 PSJ_SA가 보다 크다. 특히, PSJ_SA일 경우에는 프로세서의 개수가 많을 경우 동적 태스크 할당 방법을 적용하면 디스크 병목 현상으로 오히려 Speed Up 지수가 떨어지는 반면 준동적 태스크 할당 방법을 적용하면 디스크 병목 현상이 다소 완화됨을 알 수 있다. 또한 준동적 태스크 할당이 PSJ_MA에서는 메시지 전송 횟수의 감소로 이어져 Speed Up 지수가 올라감을 알 수 있다.



(그림 16) 병렬 처리의 Speed Up 효과

5.4 실험 결과 분석

5.4.1 디스크 접근 횟수

준동적으로 할당된 태스크 그룹을 수행하기 위해 소요되는 디스크 접근 횟수는 전역 버퍼 효과로 버퍼 hit에 의해 절약되는 디스크 접근 횟수만큼 감축되는 효과가 있다. DA_{buffer_effect}를 '전역 버퍼 효과로 인해 절약되는 디스크 접근 횟수'라 할 때 식 (34)와 같이 표현될 수 있다. 수식 34의 $DA_{save}(i)$는 i단계 전의 태스크의 수행이 현재 태스크 수행에 있어서 얼마만큼의 디스크 접근 횟수를 감축하느냐를 표현한 것이다. k가 1일 때는 바로 이전의 태스크 수행으로 버퍼에 존재하는 페이지들에 의한 감축 효과이고, k가 i-1일 때는 첫 태스크의 수행으로 인한 전역 버퍼 효과로 얻어지는 감축효과를 나타낸 것이다. 그러므로 지역적으로 가까운 태스크 들을 보다 많이, 그리고 연속적으로 동일한 프로세서에 할당할 경우 전역버퍼의 효과는 크다고 볼 수 있다. 식 (1)을 보면 첫 번째 태스크를 수행할 경우 DA_{buffer_effect}는 전혀 없다. 두 번째 태스크를 수행할 경우 바로 직전의 태스크 수행으로 인한 DA_{buffer_effect}가 있다. 마찬가지로 세 번째 태스크를 수행할 경우 이전 두개의 태스크 수행이 DA_{buffer_effect}

에 반영된다.

$$DA_{buffer\_effect}(i) = \sum_{k=1}^{i-1} DA_{save}(k)$$

k번째 전의 태스크 수행이 현재 태스크 수행 시 디스크 접근 감축효과를 나타내는 $DA_{save}(k)$ 수식은 다음과 같이 표현된다.

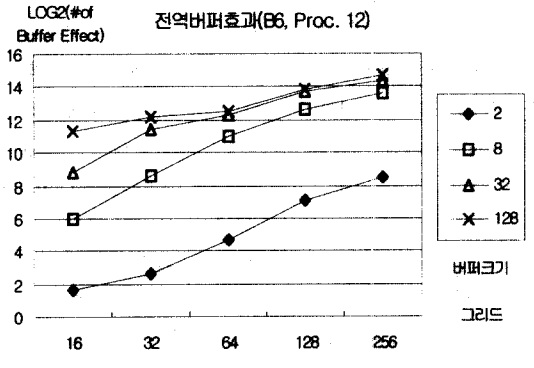
$$DA_{save}(k) = f1(k) * f2(k) * f3(k) * f4(k)$$

- $f1(k)$ = 현재 셀로부터 k번째 전의 그리드의 배타적인 대응 그리드 들의 페이지 개수
- $f2(k)$ = Hilbert Curve Number의 차가 k인 두 그리드 셀의 대응 그리드간의 배타적인 교집합 비율
- $f3(k)$ = 현재 셀로부터 k번째 전의 그리드의 대응 그리드 들이 버퍼에 남아있을 확률
- $f4(k)$ = 현재 셀로부터 k번째 전의 그리드의 대응그리드 들의 Replace를 고려한 Hit확률

이 때 $f1(k)$의 값은 그리드 해상도와 양의 상관관계가 있다. 왜냐하면 그리드 해상도가 증가할수록 대응 그리드의 개수가 증가하기 때문이다. 그리고 $f2(k)$값은 준동적 태스크 할당 방법에서 인접도의 척도가 Hilbert Curve의 특성으로 인해 실험 결과 <표 3>과 같이 가정할 수 있는 고정값이다. <표 3>에서 k가 9이상일 때는 $f2(k)$의 값이 0.001에 근접하였다. 다시 말해, 9번 전에 수행된 태스크는 현재의 태스크 수행에 있어서 디스크 접근의 감축효과가 거의 없다는 것이다. 그리고 $f3(k)$와 $f4(k)$값은 버퍼의 크기에 비례한다. 그러므로 $DA_{save}(k)$값은 그리드 해상도와 버퍼의 크기에 비례한다고 볼 수 있다. (그림 17)은 전역 버퍼 효과가 그리드 해상도와 버퍼의 크기에 비례함을 보여주는 그래프이다.

〈표 3〉 Hilbert Curve에 의한 $f2(k)$ 의 값

k	1	2	3	4	5	6	7	8	9
$f2(k)$	0.667	0.000	0.066	0.020	0.020	0.004	0.001	0.001	0.001



(그림 17) 준동적 할당을 적용한 PSJ_SA의 전역 버퍼 효과

5.4.2 후보 객체쌍의 중복제거를 위한 메시지 전송 횟수

PSJ_MA는 다중 할당이므로 객체의 중복이 발생하고, 이에 따라 여과 단계 후 후보 객체쌍의 중복이 생긴다. 따라서 이들의 제거를 위한 과정이 필요하고, 이 과정에서 많은 메시지 전송량이 필요하다. 2장에서 제시한 바와 같이 이 논문에서는 중복 제거를 위해 Sort Merge 방법을 사용한다. 이 절에서는 프로세서의 수를 함수로 하는 각 단계별 메시지 전송량 및 전체 전송량을 추정하는 비용 수식을 제시하고자 한다.

여과 단계를 마친 후 전체 중복 후보 객체쌍의 개수를 CP_{red}라 할 때 정적 또는 동적 태스크 할당일 경우 메시지 전송 없이 프로세서 내에서 자체적으로 제거되는 중복량은 CP_{red}/P이다. 그러므로 Sort Merge 방법의 1단계에서 전송될 메시지 양은 $CP_{red} - \frac{CP_{red}}{P} = \frac{CP_{red}(P-1)}{P}$이다. 이는 지역성을 전혀 반영하지 않은 상태에서 각 프로세서에 태스크를 할당했을 경우이다.

그렇지만 준동적 태스크 할당처럼 각 태스크의 지역성을 반영한다면 프로세서 내에서 자체적으로 제거되는 중복량이 $\delta * CP_{red}/P$이다. 그래서 1단계에서 전송될 메시지 양은 $\frac{CP_{red}(P-\delta)}{P}$이다. 단 $1 <= \delta <= P$이다. 즉, δ는 태스크 할당시의 지역성 반영의 척도이다. 인접한 태스크를 동일 프로세서에 할당했을 경우 의 값은 증가한다. 이상적인 경우 P에 근접하게 되어 1단계 전송 메시지 양은 0에 가깝게 된다. 그 결과 각 단계에서의 메시지 전송량이 줄어들고, 전체적인 메시지 전송량이 대폭 줄어든다.

6. 결론 및 향후 연구 과제

공간 조인은 대상이 되는 데이터 집합의 개수가 증가할수록 많은 디스크 접근 횟수와 CPU 연산이 필요하다. 이러한 문제점을 해결하려는 노력의 일환으로 병렬 처리 시스템을 이용하고자 하는 연구가 진행되고 있다. 그러나 병렬 시스템을 이용한 공간 조인에서 CPU 연산시간의 단축효과는 뛰어나지만 동시 다발적인 디스크 접근으로 인한 디스크 병목 현상과 프로세서 수의 증가에 따른 과도한 메시지 전송 횟수는 전체적인 병렬 공간 조인의 성능 향상에 있어서 걸림돌로 작용한다.

이 논문에서는 대표적인 두 가지 공간 조인 기법(MASJ, SAMJ)을 병렬 시스템을 이용하여 처리하고자 할 때 발생하는 문제점을 해결하는 태스크 할당 방법을 제시하였다. 이 논문에서 새롭게 제시한 준동적 태스크 할당 방법은 공간 데이터의 지역성과 동적 부하 평준화를 동시에 고려하여 태스크 중 일부는 서로 인접한 태스크를 그룹으로 묶어 정적(static)으로 할당하고, 나머지 태스크는 태

스크의 크기 순으로 동적(dynamic)으로 할당한다. 그리고 제안한 태스크 할당 방법을 적용하기 위한 두 가지 병렬 공간 조인 기법(PSJ_MA, PSJ_SA)은 고정 그리드를 기반으로 하였다.

실제 데이터를 이용한 실험 결과 이 논문에서 제시한 준동적 할당 방법을 적용할 경우 기존의 정적 또는 동적 태스크 할당 방법에 비해 좋은 성능을 보였다. 구체적으로 준동적 태스크 할당 방법을 적용한 PSJ_MA는 디스크 접근 횟수의 감소로 성능 향상이 있었고, PSJ_SA는 결과의 중복 제거를 위한 메시지 전송 횟수의 감소로 성능 향상이 있었다. 또한 그리드 해상도가 높고 버퍼의 크기가 클수록 준동적 태스크 할당 방법이 보다 좋은 성능을 보임을 실험으로 알 수 있었다.

준동적 태스크 할당 방법을 적용한 두 공간 조인 기법의 병렬 처리에 의한 Speed Up은 PSJ_MA가 높게 나왔지만, 동적 또는 정적 태스크 할당 방법 대비 준동적 태스크 할당 방법의 성능 향상율은 PSJ_SA가 보다 높다. 이것은 PSJ_SA의 디스크의 중복 검색을 줄이는 효과가 PSJ_MA의 메시지 전송 오버헤드를 줄이는 효과 보다 더 크기 때문이다.

이 논문의 연구 결과는 병렬 처리에 의한 CPU 연산 시간의 단축 효과 뿐만 아니라 디스크 입출력 시간과 프로세서간의 메시지 전송시간을 동시에 줄일 수 있는 태스크 할당 방법을 약간의 변형을 통해 대표적인 두 가지 공간 조인 기법에 적용할 수 있다. 앞으로 주기억 장치를 기반으로 하는 공간 조인을 위한 공간 색인에 관한 연구를 진행하고자 한다.

참 고 문 헌

[1] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. S. Vitter, "*Scalable Sweeping Based Spatial Join*," Proc. of Int. Conf. on VLDB, pp.570-581, 1998.

[2] T. Brinkhoff, H. P. Kriegel, R.Schneider, B. Seeger, "*Efficient Processing of Spatial Joins Using R-trees*," Proc. of Int. Conf. on Management of Data, ACM SIGMOD, pp.237-246, 1993.

[3] T. Brinkhoff, H. P. Kriegel, B. Seeger, "*Parallel Processing of Spatial Joins Using R-trees*," Proc. of Int. Conf. on Data Engineering, pp.258-265, 1996.

[4] D. J. DeWitt, "*DIRECT-A Multiprocessor Organization for Supporting Relational Database Management System*," IEEE Trans. on Computers, pp.395-406, 1979.

[5] E. G. Hoel, H. Samet, "*Data-Parallel Spatial Join Algorithms*," Proc. of Int. Conf. on Parallel Processing, pp.227-234, 1994.

[6] Y. W. Huang, N. Jing, E. A. Rundensteiner, "*A Cost Model for Estimating the Performance of Spatial Joins Using R-trees*," Proc. of Int. Conf. on SSDBM, pp.30-38, 1997.

[7] J. D. Kim, B. H. Hong, "*Parallel Spatial Join Algorithms using Grid Files*," Proc. of Int. Symp. on DANTE'99, pp. 127-135, 1999.

[8] N. Koudas, K. C. Sevcik, "*Size Separation Spatial Join*," Proc. of Int. Conf. on Management of Data, ACM SIGMOD, pp.324-335, 1997.

[9] R. Laurini, D. Thompson, "*Fundamentals of Spatial Information Systems*," Academic Press, 1992.

[10] M. L. Lo, C. V. Ravishankar, "*Spatial Joins Using Seeded Trees*," Proc. of Int. Conf. on Management of Data, ACM SIGMOD, pp.209-220, 1994.

[11] J. A. Orestein, "*Redundancy in spatial databases*," Proc. of Int. Conf. on Management of Data, ACM SIGMOD, pp. 294-305, 1989.

[12] J. M. Patel, D. J. Dewitt, "*Partition based spatial merge join*," Proc. of Int. Conf. on Management of Data, ACM SIGMOD, pp.259-270, 1996.

[13] Y. Theodoridis, E. Stefanakis, T. Sellis, "*Cost Models for Join Queries in Spatial Databases*," Proc. of Int. Conf. on Data Engineering, pp.476-483, 1998.

[14] X. Zhou, D. J. Abel, David Truffet, "*Data Partitioning for Parallel Spatial Join Processing*," Proc. of Int. Conf. on SSD, pp.178-196, 1997.

[15] http : //epoch.cs.berkeley.edu : 8000/sequoia/benchmark/polygon/, Sequoia 2000 FTP server home page.

[16] 김진덕, 홍봉희, "단일/다중 할당 공간 색인에서 병렬 공간 조인의 성능 평가", 한국정보과학회논문지, 제26권 제6호, pp.763-779, 1999.

[17] 서영덕, 김진덕, 홍봉희, "병렬 공간 조인을 위한 객체 캐쉬 기반 태스크 생성 및 할당", 한국정보과학회논문지, 제26권 제10호, pp.1178-1192, 1999.

김 진 덕

email : jdk@dongeui.ac.kr

1993년 부산대학교 컴퓨터공학과 졸업 (학사)

1995년 부산대학교 대학원 컴퓨터공학과 졸업(석사)

2000년 부산대학교 대학원 컴퓨터공학과 졸업(박사)

1998년~2001년 부산정보대학 정보통신계열 전임강사

2001년~현재 동의대학교 컴퓨터공학과 전임강사

관심분야 : 객체지향 DB, 지리정보시스템, 공간 질의어, 공간 색인

서 영 덕

email : ydseo@hyowon.cc.pusan.ac.kr

1997년 부산대학교 컴퓨터공학과 졸업 (학사)

1999년 부산대학교 대학원 컴퓨터공학과 졸업(석사)

1999년~현재 부산대학교 대학원 컴퓨터공학과(박사과정)

관심분야 : 지리정보시스템, 병렬 처리, 공간 색인

홍 봉 희

email : bhhong@hyowon.cc.pusan.ac.kr

1982년 서울대학교 컴퓨터공학과 졸업 (학사)

1984년 서울대학교 대학원 컴퓨터공학과 졸업(석사)

1988년 서울대학교 대학원 컴퓨터공학과 졸업(박사)

1987년~현재 부산대학교 컴퓨터공학과 교수

관심분야 : 공학DB, 객체지향 DB, GIS, 분산공간 DB, 개방형 GIS