한정된 프로세서 환경에서 페이지 실행시간 동기화를 이용한 효율적인 다중 결합

(Efficient Multiple Joins using the Synchronization of Page Execution Time in Limited Processors Environments)

이 규 옥 † 원 영 선 †† 홍 만 표 †††
(Kyu-Ock Lee) (Young-Sun Weon) (Man-Pyo Hong)

요 약 관계형 데이타베이스 시스템에서 결합 연산자는 데이타베이스 질의를 구성하는 연산자들 중 가장 많은 처리시간을 요구한다. 따라서 이러한 결합 연산자를 효율적으로 처리하기 위해 많은 병렬 알고리즘들이 소개되었다. 그 중 다중 해쉬 결합 질의의 처리를 위해 할당 트리를 이용한 방법이 가장 우수한 것으로 알려져 왔다. 그러나 이 방법은 할당 트리의 각 노드에서 필연적인 지연이 발생되는 데 이는 튜플-시험 단계에서 외부 릴레이션을 디스크로부터 페이지 단위로 읽는 비용과 이미 읽는 페이지에 대한 해쉬 결합 비용간의 실행시간 차이에 의해 발생하게 된다. 이는 페이지 실행시간 동기화 기법을 이용하여 할당 트리 한 노드에서의 실행시간을 줄일 수 있었다. 본 논문에서는 한 노드에서의 성능 개선 효과를 할당 트리 전체로 확장하여 전체 다중 해쉬 결합의 성능 분석을 수행하였으며 한정된 프로세서 환경 하에서 입력 릴레이션 수와 할당된 프로세서 수와의 관계에 따른 효율적인 다중 해쉬 결합 알고리즘을 제안하였다. 그리고 분석적 비용 모형을 세워 기존 방식과의 다양한 성능 분석을 통해 비용 모형의 타당성을 입증하였다.

Abstract *In the relational database systems, the join operation is one of the most time-consuming query operations. Many parallel join algorithms have been developed to reduce the execution time. Multiple hash join algorithm using allocation tree is one of the most efficient ones. However, it may have some delay on the processing each node of allocation tree, which is occurred in tuple-probing phase by the difference between one page reading time of outer relation and the processing time of already read one. This delay problem was solved by using the concept of 'synchronization of page execution time' which we had proposed. In this paper, the effects of the performance improvements in each node of the allocation tree are extended to the whole allocation tree and the performance evaluation about that is processed. In addition, we propose an efficient algorithm for multiple hash joins in limited number of processor environments according to the relationship between the number of input relations in the allocation tree and the number of processors allocated to the tree. Finally, we analyze the performance by building the analytical cost model and verify the validity of it by various performance comparison with previous method.*

1. 서 론

최근 데이타베이스의 규모가 대용량화되고 데이타베이스 연산도 매우 복잡해짐에 따라 이 분야에 대한 연구가 더욱 활발히 이루어지고 있다. 관계형 데이타베이스 연산 중에서 결합 연산은 다른 연산에 비해 월등히 많은 비용을 요구하는 연산으로서 데이타베이스의 규모가 커지고 복잡도가 높아짐에 따라 그 비용은 더욱 증가하게 된다. 그에 대한 해결책은 병렬성(parallelism)을 최대한 이용하는 것이라고 할 수 있다. 이와 같은 복잡한 다중 결합 질의를 효율적으로 처리하기 위하여 병렬성을 이용한 많은 연구가 수행되었다. 병렬성은 하나의 결합 연산

† 정 회 원 : 한국기계연구원 연구원
kolee@kimm.re.kr
†† 비 회 원 : 아주대학교 정보및컴퓨터공학부
ysweon@madang.ajou.ac.kr
††† 종신회원 : 아주대학교 정보및컴퓨터공학부 교수
mphong@madang.ajou.ac.kr
논문접수 : 2000년 10월 11일
심사완료 : 2001년 8월 2일

에 대해서 다수의 프로세서들이 병렬로 작업을 수행하는 연산자내 병렬성(intra-operator parallelism)과 다중 질의의 병렬 수행을 위한 연산자간 병렬성(inter-operator parallelism)으로 나눌 수 있다. 일반적으로 질의 계획(query plan)은 결합 순차 트리(join sequence tree)라 불리는 연산 트리로 변환되는데, 이 트리의 말단 노드는 입력 릴레이션을, 중간 노드는 두 자식 노드에 할당된 두 릴레이션의 결합으로부터 생성된 결과 릴레이션을 나타낸다. 연산 트리는 그 형태에 따라 좌향(left-deep) 트리, 우향(right-deep) 트리 그리고 부쉬(bushy) 트리로 분류되며, 좌향 트리와 우향 트리를 선형 실행 트리(linear execution trees) 또는 순차 결합 순서(sequential join sequences) 라고 부른다.

다양한 결합 방법 중에서 해쉬 결합은 다른 결합 방법에 비해 우수한 성능을 보이며 특히, 다수의 해쉬 결합이 우향 트리 형태로 구성된 다중 해쉬 결합일 경우, 파이프라인으로 처리할 수 있다[1,2,3,4,5]. 하나의 파이프라인으로 구성된 다중 해쉬 결합은 여러 단계로 이루어져 있으며, 각각의 단계는 여러 개의 프로세서에 의해서 병렬로 실행될 수 있는 하나의 결합 연산으로 이루어져 있다.

최근까지는 주로 선형 실행 트리에 대한 연구가 진행된 반면, 하드웨어의 급속한 발전과 점차 복잡해지는 질의로 인해 현재에는 부쉬 트리에 대한 연구가 활발히 이루어지고 있다. 또한 연산자간 병렬화와 연산자내 병렬화의 통합적 접근이라든지 결합 순서 스케줄링 및 프로세서 할당 등을 다루는 연구들이 제안되었다[6]. 최근의 연구결과들 중에서 다중 해쉬 결합 질의 처리를 위한 효율적인 방법으로 할당 트리를 이용한 정적 프로세서 할당 방법이 제안되었는데, 이 방법은 파이프라인이 가능한 부쉬 트리의 노드들을 그룹핑하여 할당 트리를 생성하고 기본 릴레이션의 초기 정보, 즉 릴레이션의 카디널리티와 속성 값의 도메인 카디널리티를 이용하여 상향식(bottom-up) 방법으로 누적 실행 비용을 계산 한 후 다시 하향식(top-down) 방법에 의해 프로세서를 할당하는 방법이다[7]. 특히, 이 방법은 동기실행시간(synchronous execution time) 개념을 이용해 할당 트리 내에서 동일한 수준의 노드들이 가급적 동시에 실행이 완료 되도록 함으로서 전체적으로 지연시간을 줄이고자 하였다[6]. 그러나 이 방법에서는 할당 트리의 각 노드에 할당된 프로세서들에 대해서는 각 노드 내에서의 해쉬 결합들을 수행하는 비용 모형에 대한 충분한 분석 및 평가가 고려되지 않음으로서 각 노드 내에서 필연적으로 발생될 수 있는 지연시간에 대한 처리를 할 수 없었다. 이 문제는 페이지 실행시간 동기화를 이용한 파이프라인 해쉬 결합 알고리즘을 제안하여 해결하였으며[9,10], 본 논문에서는 이 페이지 실행시간 동기화 기법을 할당 트리의 한 노드로부터 할당 트리 전체로 확장하였으며, 분석적인 비용 모형을 세워 한정된 프로세서 환경 하에서 입력 릴레이션 수와 할당된 프로세서 수와의 관계에 따른 효율적인 다중 해쉬 결합 방법을 제안하였다. 또한 이 비용 모형을 토대로 그 결과를 기존의 방법과 비교하였다.

논문의 구성은, 2장에서 관련 연구로서 할당 트리 한 노드의 실행을 페이지 실행시간 동기화 기법을 이용해 수행하는 내용을 살펴보고, 3장에서는 페이지 실행시간 동기화 기법을 할당 트리 전체로 확장한 내용과 다양한 환경에서 효율적인 다중 해쉬 결합을 수행할 수 있는 방법에 대해 살펴본다. 4장에서는 제안한 방법에 대한 성능 평가를 위한 비용 모형을 제시한다. 5장에서는 제시한 비용 모형을 통해 다양한 방법으로 성능을 평가를 하며 기존의 방법과 비교한다. 마지막으로 6장에서는 결론으로서 본 논문의 결과를 정리한다.

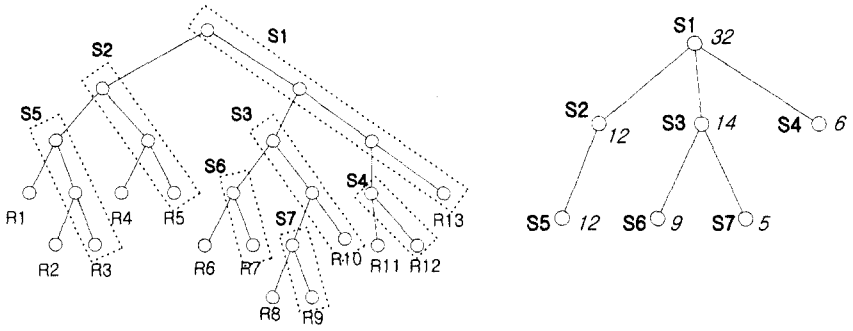
2. 페이지 실행시간 동기화를 이용한 다중 파이프라인 해쉬 결합

2.1 할당 트리를 이용한 프로세서 할당 기법

부쉬 트리로 표현된 다중 해쉬 결합 질의를 처리하기 위하여 부쉬 트리에서 파이프라인이 가능한 노드들을 그룹핑하여 할당 트리로 변환한 후 질의를 처리하는 방법이 제안되었다. 부쉬 트리 형태로 표현된 다중 해쉬 결합 질의를 할당 트리의 형태로 변환하고 각 노드에 프로세서를 할당하는 방법에 대해 살펴본다.

할당 트리를 생성하는 방법은 먼저 부쉬 트리로 표현된 질의에서 파이프라인이 가능한 노드들을 찾는다. 그리고 파이프라인이 가능한 노드들을 그룹핑하여 할당 트리의 한 노드로 변환한다. 그림 1은 부쉬 트리에서 파이프라인이 가능한 노드들을 식별하여 할당 트리로 변환하는 과정을 보여준다. (a)의 각각의 파이프라인의 그룹들은 (b)의 한 노드로 변환됨을 알 수 있다.

부쉬 트리를 할당 트리로 변환한 후, 상향식(bottom-up) 방법에 의해 누적실행비용(cumulative execution cost)을 계산하고 다시 하향식(top-down) 방법에 의해 각 노드에 프로세서를 할당하게 된다. 이 방법은 하나의 부(parent) 노드의 실행 전까지 모든 자식(child) 노드들이 거의 동시에 실행을 마칠 수 있도록 동기실행시간(synchronous execution time) 개념을 이용하며, 결합 연산을 수행하기 전에 단지 기본 릴레이션들의 카디



(a) (b)

그림 1 부쉬 트리의 할당트리 변환

널리티와 속성값이 가질 수 있는 도메인의 카디널리티만을 고려하여 누적 실행 비용을 계산하고, 그 값에 의해 프로세서를 할당하는 프로세서 할당 기법을 이용한다.

이러한 프로세서 할당 방법의 실행 과정은 다음과 같다.

step 1 : 하나의 결합 순서 휴리스틱을 이용하여 부쉬 실행 트리를 결정한다.

step 2 : 부쉬 트리로부터 파이프라인이 가능한 릴레이션들을 그룹핑하여 할당 트리로 변환한다.

step 3 : 상향식 방법에 의해 할당 트리 각 노드의 누적 실행 비용을 계산한다.

step 4 : 하향식 방법에 의해 할당 트리 각 노드에 프로세서를 할당한다.

이 프로세서 할당 기법의 문제점은 기본 릴레이션의 카디널리티와 속성값이 가질 수 있는 도메인의 카디널리티 만을 고려하여 누적 실행 비용을 계산하고 그 값에 따라 프로세서들을 할당하기 때문에 각각의 결합 결과로 생성되는 중간 릴레이션(intermediate relations)들의 크기를 전혀 예측할 수 없고 그로 인해 각 노드에 프로세서를 할당을 하기 위한 보다 정확한 workload를 예측하기 어렵다는데 있다. 따라서, 이러한 문제점을 해결하기 위한 연구들도 활발히 진행되고 있다.

할당 트리의 한 노드는 그림 2와 같은 하나 이상의 해쉬 결합 질의를 파이프라인 방식으로 처리할 수 있는 우향 트리로 구성되어 있다. 여기서 내부 릴레이션들(R)에 대한 해쉬-표의 구축은 동시에 이루어질 수 있으며 모든 해쉬-표의 구축이 완료되면 외부 릴레이션(S)은 디스크로부터 페이지 단위로 입력되어 해쉬-표를 구축할 때 사용했던 동일한 해쉬 함수를 적용하여 한 튜플씩 해쉬-표에서 비교한다. 비교한 결과, 일치하는 튜플은 결합되어 상위-수준의 결합으로 흘러가며 일치하지 않는 튜플은 무시하게 된다. 일치하여 상위-수준의 결합으로 흘러온 튜플들은 같은 방식으로 처리되어 최상위-수준의 결합에 도달하게 되며 여기서 결합에 성공한 튜플들은 최종적으로 페이지 단위로 디스크에 저장하게 된다.

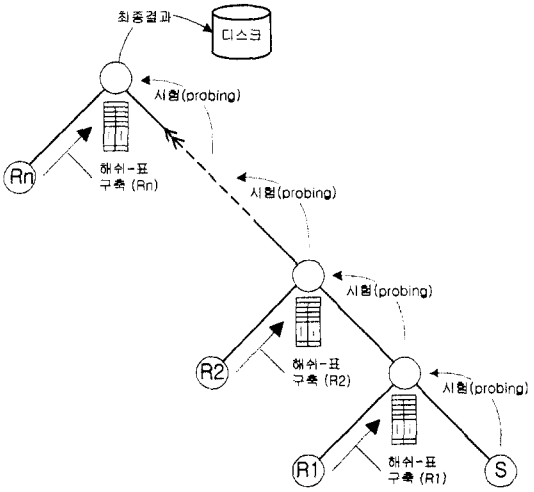


그림 2 할당 트리의 한 노드

할당 트리 한 노드에서의 해쉬 결합의 위치 및 각 해쉬 결합 내에서의 기능에 따라 각 프로세서들이 처리해야할 비용은 많은 차이를 갖게 되며 이는 프로세서들의 불가피한 지연을 야기할 수 있으며 나아가 전체 시스템의 성능을 저하시킬 수 있다. 이 같은 문제점을 방지하기 위해서는 정확하고도 보다 정밀한 비용 모형을 세우고 다양한 성능 분석을 통해 효과적인 프로세서의 할당 방법을 마련하는 것이 필수적이다.

할당 트리의 한 노드, 즉 우향 트리의 해쉬 결합 실행 절차를 할당된 프로세서들의 측면에서 나타내면 그림 3과 같다.

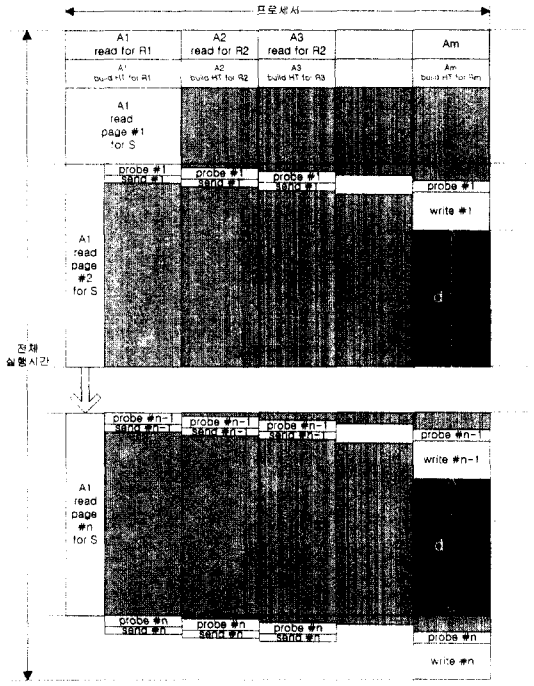


그림 3 할당 트리의 한 노드의 해쉬 결합 실행 절차

여기서는 한 노드에 할당된 프로세서의 수가 그 노드 내의 내부 릴레이션(R)의 수 보다 많거나 같고, 각 프로세서는 I/O 전담 프로세서가 있으며, 프로세서들 사이의 통신지연시간(communication latency)은 없다고 가정한다. 또한 프로세서 구조는 분산 메모리와 공유 디스크를 갖는 다중 프로세서 구조라고 가정한다.

먼저, 할당 트리의 한 노드에 할당된 프로세서(A1, .., Am) 전체는 그 노드에 속한 모든 내부 릴레이션들(R)을 디스크로부터 읽어서 각각의 해쉬-표를 구축한다. 구축이 완료되면, 할당된 프로세서들 중 하나의 프로세서(A1)는 외부 릴레이션(S)을 디스크로부터 페이지 단위로 읽어서 해쉬 결합을 진행하여 그 결과를 상위 해쉬 결합에 할당된 프로세서들에 전송한다. 이 같은 해쉬 결합은 상향식으로 이루어져 그 결과가 최상위 해쉬 결합에 할당된 프로세서들에 전송되며 최상위 해쉬 결합에 할당된 프로세서들은 최종적으로 결합을 수행한 후 그 결과를 디스크에 저장하게 된다. 이 작업은 외부 릴레이션(S)의 크기를 페이지 크기로 나눈 수 즉, n번 파이프라인 방식으로 실행되며 마지막으로 최종적으로 입력된 페이지에 대한 해쉬 결합을 상기의 절차로 수행하게 되면 할당 트리 한 노드의 처리가 완료된다.

그러나 그림 3에서 보는 바와 같이 외부 릴레이션(S)의 한 페이지를 디스크로부터 읽는 시간에 비해 이미 읽은 한 페이지에 대한 해쉬 결합 실행시간이 현저히 차이(그림 3의 d)가 나는 것을 볼 수 있다[9]. 이 차이는 트리의 깊이(depth)가 클수록 더욱 커지며 따라서, 결합에 참여하는 프로세서들의 지연시간이 길어지고 그 영향이 전체 질의 실행시간에 미치게 된다. 또한 일반적으로 질의 최적화 과정을 통해 내부 릴레이션의 크기보다 외부 릴레이션의 크기가 상대적으로 크기 때문에 외부 릴레이션(S)의 크기가 커질수록 전체 질의 실행시간에 미치는 영향은 더욱 커지게 된다.

따라서 외부 릴레이션(S)의 한 페이지를 디스크로부터 읽는 시간과 이미 읽은 한 페이지에 대한 해쉬 결합 실행시간과의 차이를 줄임으로서 할당 트리 내의 한 노드에서의 실행시간을 최소화할 수 있는 페이지 실행시간 동기화 기법이 제안되었다.

2.2 페이지 실행시간의 동기화

할당 트리의 한 노드 즉, 우향 트리로 표현된 다중 해쉬 결합을 파이프라인 형식으로 실행함에 있어 프로세서들의 지연시간을 최소화하고 나아가 전체 시스템의 성능을 향상시키기 위해서는 프로세서가 튜플-시험 단계에서 외부 릴레이션(S)을 디스크로부터 한 페이지 읽는 비용과 이미 읽은 페이지에 대한 처리시간 사이의 차이를 최대한 줄여야 한다. 이러한 차이를 줄이기 위해서는 외부 릴레이션(S)의 한 페이지를 읽는 시간에 다수의 페이지를 다수의 프로세서에서 동시에 읽도록 하고 그 시간 내에 이미 읽은 다수의 페이지들을 처리하도록 함으로서 페이지들을 동시에 읽는 시간과 이미 읽은 페이지들에 대한 처리가 가급적 동시에 완료될 수 있도록 하는 것이다. 이는 위에서 언급한 차이에 인한 프로세서들의 지연을 최소화할 수 있게 되는데 이 개념을 '페이지 실행시간 동기화' 라고 하였다[9,10].

페이지 실행시간 동기화는 외부 릴레이션(S)의 한 페이지를 읽는데 소요되는 시간을 이미 읽은 외부 릴레이션(S)의 한 페이지에 대한 해쉬 결합 실행시간으로 나누고, 그 값을 k라고 할 때, 가능하다면 k개의 프로세서를 외부 릴레이션(S)의 페이지를 디스크로부터 읽는 작업에 할당하는 방법이다. 따라서, k개의 프로세서들은 외부 릴레이션(S)에서 k개의 페이지를 동시에 읽을 수 있으며, 읽혀진 k개의 페이지들에 대한 해쉬 결합 실행

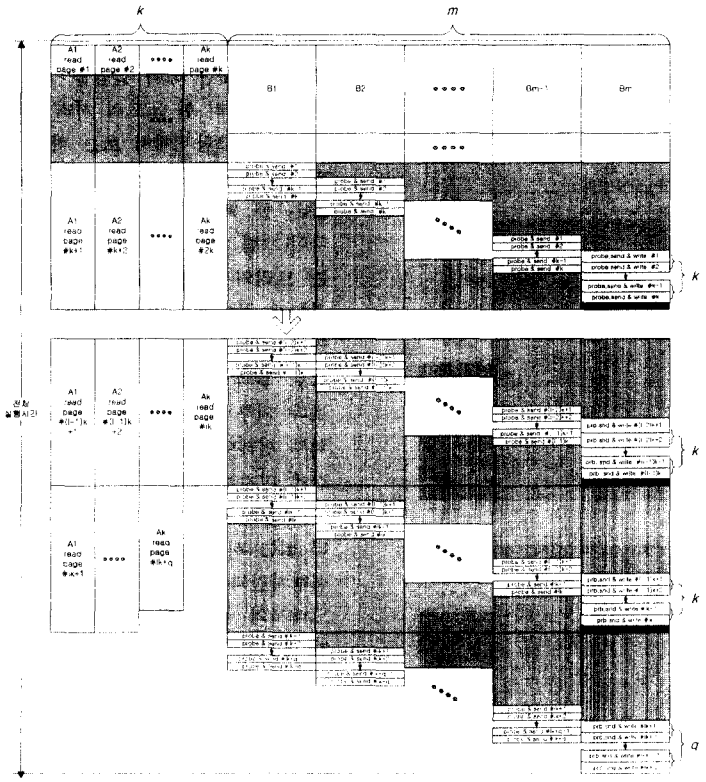


그림 4 페이지 실행시간 동기화를 이용한 다중 해쉬 결합 실행 과정

은 나머지 프로세서들에서 파이프라인 방식으로 실행하도록 한다. 결국, k개 단위의 페이지 입력과 이미 읽은 k개 페이지에 대한 다중 해쉬 결합 작업이 파이프라인 방식으로 처리될 수 있다. 따라서, 전체 실행시간에 많은 영향을 미치고 있는 외부 릴레이션(S)의 페이지 단위 입력 비용을 k개의 프로세서에서 동시에 실행할 수 있도록 함으로서 질의에 대한 전체 실행시간을 줄일 수 있다. 단, 여기서의 전제는 할당 트리 한 노드에 할당된 프로세서의 수는 그 노드의 내부 릴레이션(R)의 수보다는 커야 하고, 하나의 해쉬 결합에는 적어도 하나 이상의 프로세서를 할당하여야 한다는 것을 전제로 한다. 또한 프로세서 구조는 분산 메모리와 공유 디스크를 갖는 다중 프로세서 구조라고 가정한다.

그림 4는 할당 트리의 한 노드에서 페이지 실행시간 동기화를 이용한 다중 해쉬 결합 의 실행 과정을 보여준다. 여기서, k는 외부 릴레이션(S)의 페이지를 읽는 작업에 할당된 프로세서의 수이며, m은 해쉬 결합을 수행하는 프로세서의 수로서, 그 노드에 포함된 내부 릴레이션(R)의 수 즉, 해쉬 결합의 수보다는 크거나 같아야 한다. 따라서, 그 노드에 할당된 전체 프로세서의 수를 N이라고 할 때, $N=k+m$ 이 된다. k값은 그림 4에서의 d 값을 줄일 수 있는 최적의 값이며 프로세서 수가 충분하다고 더 큰 값을 k값으로 부여한다고 해도 해쉬 결합 쪽에서 그 시간 안에 k개의 페이지를 처리할 수 없으므로 페이지를 읽는 프로세서들이 오히려 지연될 수 있다. 그러므로 충분한 프로세서들이 할당되었을 경우에는 k개의 프로세서를 외부 릴레이션(S)을 페이지 단위로 읽는데 할당하고, 나머지 프로세서들은 모두 해쉬 결합을 수행하는 데 할당함으로서 내부 릴레이션들을 읽어서 해쉬-표를 생성하는데 소요되는 시간을 줄이도록 한다.

3. 할당 트리의 실행

할당 트리로 표현된 다중 해쉬 결합의 실행은 각 노

드에 할당된 프로세서들에 의해 실행된다. 트리 전체는 동기실행시간 개념에 입각해서 실행되며 각 노드의 실행시간이 짧아질수록 전체 트리의 실행시간이 빨라질 수 있다. 본 연구에서는 할당 트리의 각 노드의 실행 그리고 트리 전체의 실행에 있어서 입력 릴레이션의 수와 할당된 프로세서 수와의 관계를 분석하였다.

3.1 할당 트리 한 노드의 실행

할당 트리의 실행을 위해 할당 트리의 각 노드에는 누적실행비용에 따라 프로세서들이 할당된다. 하지만 그 프로세서의 수는 그 노드에서 결합될 입력 릴레이션의 수보다 적거나, 같거나 또는 많을 수 있다. 단일 프로세서의 경우는 순차 실행 밖에 될 수 없으며, 할당된 프로세서의 수가 입력 릴레이션의 수보다 적거나 같을 경우에는 페이지 실행시간 동기화 기법을 사용할 수 없으므로 기존 방식처럼 모든 내부 릴레이션을 프로세서들이 동시에 디스크로부터 페이지 단위로 읽어 해쉬-표를 구축한 다음, 마찬가지로 프로세서들이 외부 릴레이션을 동시에 페이지 단위로 읽어서 튜플-시험 단계를 통해 결합을 수행하도록 한다.

할당 트리의 한 노드 즉, 우향트리로 표현된 다중 해쉬 결합의 실행의 경우, 그 노드에 할당된 프로세서의 수가 그 노드에서 결합될 입력 릴레이션의 수보다 적거나 같을 경우에는 기존 실행 방식의 성능이 우수하였고, 클 경우에는 페이지 실행시간 동기화 기법을 이용한 실행 방법이 우수하였다. 자세한 성능 분석 결과는 5장에서 언급한다.

3.2 할당 트리 전체 실행

할당 트리로 표현된 다중 해쉬 결합은 누적실행비용에 따라 할당된 프로세서들에 의해 실행된다. 하지만 단일 프로세서 환경이라든지 또는 할당 트리에 포함된 입력 릴레이션 수에 비해 시스템 전체의 프로세서 수가 적을 경우에는 기존 방식대로의 프로세서 할당이 불가능하며 따라서 다음과 같은 다양한 방법으로의 실행이 가능하다.

1) 순차 실행

2) 누적실행비용을 고려한 부분 순차 실행

본 연구에서는 이러한 경우에 2가지 방법에 의해 할당 트리를 실행하고 나아가 프로세서의 수가 할당 트리에 포함된 전체 입력 릴레이션 수보다 클 경우에는 페이지 실행시간 동기화 기법까지 적용하여 실행 성능을 분석하였다. 성능 분석결과는 단일 프로세서 시스템인 경우에는 순차 실행 방법이 유일한 방법이므로 성능이 동일하다고 할 수 있으며 프로세서의 수가 하나 이상이고 입력 릴레이션의 수와 동일할 때까지는 누적실행비용을 고려한 부분 순차 실행 방법이 순차 실행 보다 성능이 우수함을 보였다. 그리고 프로세서의 수가 입력 릴레이션의 수보다 클 경우에는 그 차이가 크면 클수록 페이지 실행시간 동기화 기법을 이용한 실행 방법이 가장 성능이 우수함을 보였다. 자세한 성능 분석 결과는 5장에서 언급한다.

4. 비용 모형

본 장에서는 할당 트리의 실행을 위한 다양한 비용 모형을 제시한다. 먼저 할당 트리의 임의의 한 노드에서 프로세서 수와 입력 릴레이션 수와의 관계에 따른 비용 모형, 그리고 할당 트리 전체의 실행 방법에 따른 각각의 분석적 비용 모형을 제시한다. 할당 트리 전체를 실행하는 방법은 첫째, 할당 트리의 각 노드를 순서에 맞게 순차적으로 실행하는 방법이며 이때 할당 트리에 할당된 전체의 프로세서들을 실행되는 노드에 전부 할당한다. 둘째는 기존의 방법과 같이 누적실행비용에 따라 프로세서들을 할당 트리에 분배하고 병렬 실행이 가능한 노드들을 병렬로 실행하는 방법이다. 마지막으로 페이지 실행시간 동기화를 이용한 방법이다. 할당 트리 전체를 실행함에 있어서도 전체 프로세서의 수와 입력 릴레이션과의 관계에 따른 비용 모형도 고려하였다. 본 연구에서 제시한 비용 모형의 기본 가정은 다음과 같다.

(1) 컴퓨터 구조는 분산 메모리와 공유 디스크를 갖

표 1 비용 모형과 성능 분석을 위한 매개변수와 특성 값

매개변수	의 미	특성 값
N_P	한 노드에 할당된 전체 프로세서 수	
$\|\|R\|\|$	내부 릴레이션(R)의 카디널리티	
$\|\|S\|\|$	외부 릴레이션(S)의 카디널리티	
P_{speed}	프로세서 처리 속도	3MIPS
t_{pio}	페이지 당 디스크 서비스 시간	20ms
I_{read}	디스크로부터 한 페이지 읽는데 필요한 명령어 수	5,000
I_{write}	디스크에 한 페이지 쓰는데 필요한 명령어 수	5,000
I_{build}	해쉬 표 구축 단계에서 한 튜플 처리를 위한 명령어 수	100
I_{probe}	튜플 시험 단계에서 한 튜플 처리를 위한 명령어 수	200
I_{send}	한 튜플 전송에 필요한 명령어 수	100
P_{size}	페이지 당 튜플 수	40tuples
ρ_i	한 노드 내의 i 번째 결합 단계에서의 결합률	
d	외부 릴레이션(S) 한페이지 읽는 시간과 한 페이지 결합 시간과의 차이	
k	d값을 줄이기 위해 할당한 외부 릴레이션(S) 입력 전담 프로세서 수	
B_P	$N_P - (k$ + 한 노드 내의 해쉬결합 수)	

는 다중 프로세서 구조이다.

(2) 각각의 프로세서는 하나의 내부 릴레이션에 대한 해쉬-표를 저장하기에 충분한 크기를 메모리를 갖는다.

(3) 프로세서들 사이에 통신지연시간(communication latency)은 없다.

(4) 다수의 프로세서들이 한 릴레이션의 다수의 페이지를 동시에 읽을 수 있는 디스크 구조를 갖는다.

본 연구의 비용 모형을 위한 매개변수 및 환경 특성 값은 표 1과 같다.

먼저, 한 노드에서의 비용 모형을 세우고, 다음 전체 할당 트리에 대한 분석적 비용 모형을 세운다.

4.1 한 노드의 해쉬 결합

할당 트리 한 노드(S)를 누적실행비용을 통해 할당된 프로세서들을 이용해 실행할 때의 결합 비용을 $Join_{time}(S)$라 할 때, $Join_{time}(S)$는 (1)식과 같다.

$$Join_{time}(S) = \frac{\sum_{i=1}^{m} \frac{\| R_i \|}{P_{size}} \cdot \left(\frac{I_{read} + I_{build}}{P_{speed}} + t_{pio}\right)}{N_P} + \frac{\|S\|}{P_{size}} \cdot \left(\frac{I_{read}}{P_{speed}} + t_{pio}\right) + \sum_{i=1}^{m-1} \frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1} \cdot \rho_i} \cdot I_{probe} + I_{send}}{P_{speed}} + \left(P_{size}^{m-1} \cdot \frac{I_{probe}}{P_{speed}} + \frac{\frac{P_{size}^{m}}{P_{size}} \cdot I_{write}}{P_{speed}} + \frac{P_{size}^{m}}{P_{size}} \cdot t_{pio}\right) \quad (1)$$

$Join_{time}(S)$는 3개 항목의 합으로 구성되는데, 첫 번째 항목은 한 노드 내의 모든 내부 릴레이션들을 디스크로부터 읽어 들여 각각의 해쉬-표를 구축하는 비용이고, 둘째는, 외부 릴레이션을 디스크로부터 페이지 단위로 읽어 들이는 시간으로서 이 시간에는 읽어 들인 페이지들에 대한 해쉬 결합 실행시간이 포함된다. 셋째는, 마지막으로 읽어 들인 페이지에 대한 해쉬 결합 실행 비용이다.

또한, 할당 트리 한 노드를 페이지 실행시간 동기화 기법을 이용해 실행할 때의 비용을 $Join_{time}^{SyncPET}(S)$라 할 때, $Join_{time}^{SyncPET}(S)$는 (2)식과 같다.

$$Join_{time}^{SyncPET}(S) = \frac{\sum_{i=1}^{m} \frac{\| R_i \|}{P_{size}} \cdot \left(\frac{I_{read} + I_{build}}{P_{speed}} + t_{pio}\right)}{B_P} + \frac{\left(\frac{\|S\|}{P_{size}} - 1\right)}{k} \cdot \left(\frac{I_{read}}{P_{speed}} + t_{pio}\right) + \sum_{i=1}^{m-1} \frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1} \cdot \rho_i} \cdot I_{probe} + I_{send}}{P_{speed}} + \sum_{i=1}^{a} \left(P_{size}^{m-1} \cdot \frac{I_{probe}}{P_{speed}} + \frac{\frac{P_{size}^{m}}{P_{size}} \cdot I_{write}}{P_{speed}} + \frac{P_{size}^{m}}{P_{size}} \cdot t_{pio}\right) \quad (2)$$

여기서, B_P는 내부 릴레이션(R)들을 디스크로부터 읽어서 해쉬-표를 구축하고 페이지 단위로 해쉬 결합을 수행하는 프로세서의 수를 말하며, 그 수는 $N_P-(k+depth)$가 된다. 만약, B_P가 0보다 작거나 같다면, B_P는 바로 트리의 $depth$가 되고, 만약 0보다 크다면, B_P는 $B_P+depth$가 된다. 또한 k는 외부 릴레이션을 디스크로부터 한 페이지 읽는데 소요되는 비용을 이미 읽은 한 페이지 해쉬 결합 비용으로 나눈 값이다.

4.2 할당 트리 전체 실행

할당 트리를 실행하는 방법은 전체 노드를 순차적으로 전체 프로세서를 이용하여 실행하는 방법, 누적실행비용을 통해 할당된 프로세서로 병렬 실행하는 방법, 그리고 페이지 실행시간 동기화를 이용한 방법을 비교 분석하고자 한다.

할당 트리(T) 전체 실행 비용을 $Total_{cost}(T)$라고 하고 T에 포함된 임의의 노드를 S_x라고 할 때, 순차 실행의 경우 비용은 다음 (3)식과 같다.

$$Total_{cost}(T) = \sum_{x \in T} Join_{time}(S_x) \quad (3)$$

그리고, 할당 트리의 실행을 누적실행비용을 통해 할당된 프로세서로 실행할 경우의 실행 비용은 (4)식과 같다.

$$Total_{cost}(T) = \max_{p \in Set_{node}} \left(\sum_{x \in p} Join_{time}(S_x) \right) \quad (4)$$

마지막으로 페이지 실행시간 동기화를 이용한 실행 비용은 (5)식과 같다.

$$Total_{cost}(T) = \max_{p \in Set_{node}} \left(\sum_{x \in p} Join_{time}^{SyncPET}(S_x) \right) \quad (5)$$

5. 성능 분석

본 장에서는 4장에서 설정한 비용 모형들을 근거로 할당 트리의 한 노드 및 전체 트리에 대한 성능을 입력 릴레이션 수와 할당된 프로세서 수의 관계 측면에서 면밀히 분석 평가하였다. 성능 분석을 위한 매개 변수는 표 1과 같다. 동일한 환경에서의 성능분석을 위해 내부 릴레이션들(R)과 외부 릴레이션(S)의 카디널리티를 각각 2,000,000과 3,000,000 튜플 정도로 설정하였고, 할당

트리의 각 노드에 포함된 각각의 해쉬 결합에서의 결합률은 50% 정도로 하여 결합에 이용하였다.

5.1 한 노드의 실행 성능

할당 트리 한 노드의 실행은 성능 분석 결과에서 그 노드에 할당된 프로세서의 수에 따라 실행 방법을 유동적으로 할 필요가 있다. 그림 5에서 그림 8은 한 노드를 구성하고 있는 우향트리의 깊이(depth) 즉, 다중 해쉬 결합의 수와 할당된 프로세서 수와의 관계에서 한 노드의 실행 성능을 기존방식과 페이지 실행시간 동기화 기법을 이용한 방법을 비교한 결과다.

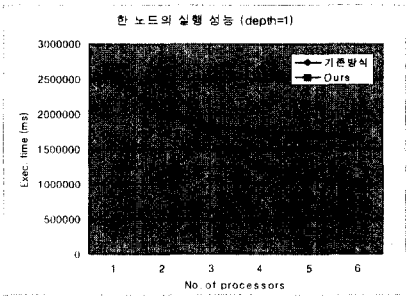


그림 5 깊이가 1인 경우

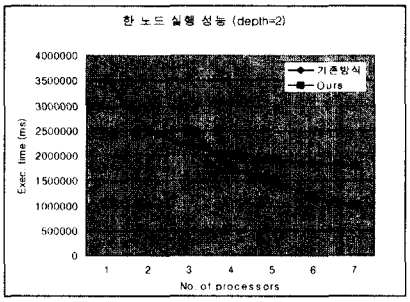


그림 6 깊이가 2인 경우

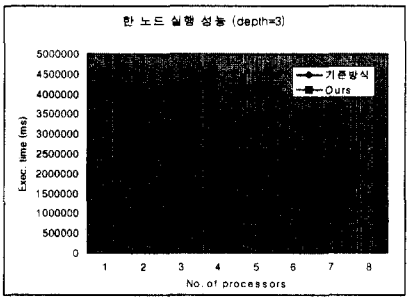


그림 7 깊이가 3인 경우

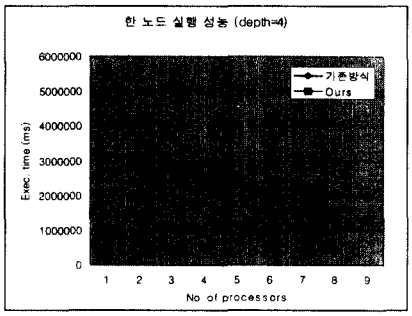


그림 8 깊이가 4인 경우

그림 5의 경우, 우향트리의 깊이가 1일 때는 내부 릴레이션(R) 1개와 외부 릴레이션(S) 1개 모두 2개의 입력 릴레이션으로 구성되어 있는 하나의 해쉬 결합이라고 할 수 있다. 깊이가 2인 경우는 내부 릴레이션 2개와 한 개의 외부 릴레이션으로 구성된 2개의 해쉬 결합이 우향 트리 형태로 된 다중 해쉬 결합이다. 그리고 깊이가 3과 4인 경우는 각각 내부 릴레이션이 3개와 4개 그리고 외부 릴레이션이 한 개로 구성된 다중 해쉬 결합이다.

그림 5에서 그림 8의 결과에서 보는 바와 같이 한 노드를 구성하고 있는 해쉬 결합의 수에 관계없이 그 노드에 할당된 프로세서의 수가 그 노드에 포함된 입력 릴레이션 수보다 작거나 같을 경우에는 기존 방식으로 수행하는 것이 좋지만 프로세서 수가 많을 경우에는 페이지 실행시간 동기화를 이용한 다중 해쉬 결합의 실행이 우수하며, 프로세서 수가 많아지면 질수록 그 성능의 차이는 더욱 커지게 된다.

5.2 할당 트리 전체의 실행 성능

할당 트리 전체의 성능 분석을 위해 그림 1에 표현된 다중 해쉬 결합 질의를 대상으로 하였다. 그림 1에서 보는 바와 같이 이 다중 해쉬 결합은 13개의 입력 릴레이션으로 구성된 부쉬 트리로 표현되고 7개의 노드로 구성된 할당 트리로 변환된다. 이 다중 해쉬 결합 질의를 첫째, 할당 트리의 각 노드를 실행 순서에 맞게 순차적으로 실행하고 이때 할당 트리에 할당된 전체의 프로세서들을 실행되는 노드에 전부 할당하도록 하였으며, 둘째는 기존의 방법과 같이 누적실행비용에 따라 프로세서들을 할당 트리에 분배하고 병렬 실행이 가능한 노드들을 병렬로 실행하였다. 마지막으로 페이지 실행시간 동기화를 이용하여 실행시키고 이들 각각의 성능을 분석하였다. 그 결과는 그림 9와 같다.

그림 9에서의 결과는 누적실행비용을 통해 할당된 프

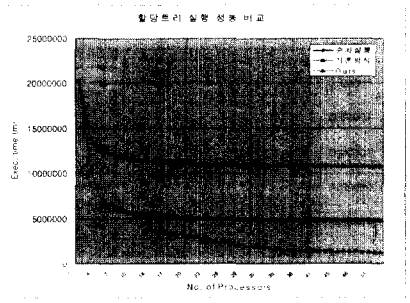


그림 9 할당 트리 전체 실행 성능 비교

로세서들을 이용한 기존의 실행 방법이 순차실행 방법보다 우수함을 보인다. 하지만 할당 트리 전체의 입력 릴레이션의 수 보다 프로세서의 수가 커지면서부터는 페이지 실행시간 동기화를 이용한 다중 해쉬 결합의 실행 성능이 가장 우수한 결과를 보였다.

질의는 최적화 과정을 통해 외부 릴레이션의 크기가 내부 릴레이션 보다 일반적으로 크게 된다. 따라서, 그림 10은 할당 트리 전체에 할당된 프로세서의 수를 32개로 고정시킨 다음 외부 릴레이션들의 크기를 500,000 튜플에서 3,000,000 튜플로 증가시키면서 기존 방식과 페이지 실행시간 동기화를 이용한 방식을 전체 할당 트리의 실행 측면에서 성능 분석한 결과다.

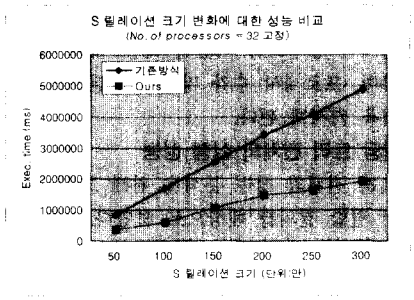


그림 10 S 릴레이션 크기 변화에 대한 성능 분석

그림 10에서 보는 바와 같이 외부 릴레이션들(S)의 크기가 커질수록 기존 방식의 경우 할당 트리 전체 실행시간이 비례적으로 증가하는 반면, 페이지 실행시간 동기화를 이용한 방식은 미세한 증가 추이를 보임을 알 수 있다. 이것은 페이지 실행시간 동기화를 이용한 방식이 외부 릴레이션(S)의 크기에 거의 독립적이라는 점이며 외부 릴레이션(S)의 크기가 커질수록 기존 방식에 비해 더욱 우수한 성능을 보인다는 것을 의미한다.

6. 결 론

복잡한 다중 해쉬 결합 질의에 대한 처리를 할당 트리를 이용하여 처리하는 것은 가장 이상적인 방법이라고 할 수 있다. 하지만 다중 파이프라인 해쉬 결합을 실행하기 위해 할당 트리의 한 노드에 할당된 프로세서들이 기능별로 상대적인 비용의 차이를 보이게 되고, 이로 인한 불가피한 프로세서들의 지연이 다중 해쉬 결합 질의의 성능을 저하시켰다. 이 문제는 페이지 실행시간 동기화를 이용해 할당 트리 각 노드에서의 성능을 개선할 수 있었다.

본 연구에서는 페이지 실행시간 동기화를 통해 확보한 할당 트리 각 노드의 성능 개선 효과를 할당 트리 전체로 확장하여 전체 다중 해쉬 결합 질의의 성능을 분석하였으며, 또한 한정된 프로세서 환경 하에서 입력 릴레이션 수와 할당된 프로세서 수와의 관계에 따른 효율적인 다중 해쉬 결합 알고리즘을 제안하였다.

제안한 알고리즘과 기존 방식과의 성능을 비교 분석한 결과 첫째, 할당 트리 한 노드의 효율적인 수행을 위해서는 한 노드에 포함된 입력 릴레이션의 수가 그 노드에 할당된 프로세서 수 보다 적거나 같을 경우에는 기존의 방식대로 실행하고, 많을 경우에는 페이지 실행시간 동기화를 이용하여 실행하는 것이 우수한 성능을 보였다. 이 현상은 프로세서의 수가 증가할수록 그 효과는 더욱 커졌다. 둘째, 할당 트리 전체를 실행함에 있어서도 순차 실행보다 누적실행비용을 통해 할당된 프로세서를 이용한 병렬 실행 방법이 우수하였으며 할당 트리 전체에 포함된 입력 릴레이션 수보다 프로세서 수가 클 경우에는 페이지 실행시간 동기화를 이용한 실행 방법이 가장 우수한 성능을 보였다. 또한 할당 트리에 포함된 해쉬 결합의 수에 관계없이 전체적인 질의 처리 성능이 기존 방식에 비해 우수하였고, 특히 페이지 실행시간 동기화를 이용한 실행 방법이 외부 릴레이션(S)의 크기에 독립적이면서 외부 릴레이션(S)의 크기가 커질수록 그 성능이 기존 방식에 비해 더욱 좋아짐을 보였다.

결과적으로 다중 해쉬 결합에 포함된 입력 릴레이션 수와 할당된 프로세서 수와의 관계에 따라 효율적인 실행 방법을 유연하게 선택하여 실행할 수 있다는데 의미가 있다.

향후 연구과제로서는 해쉬 필터(hash filter)와의 연계(interleave)를 통한 다중 파이프라인 해쉬 결합 알고리즘의 확장, 그리고 특정 프로세서 구조에서의 다중 해쉬 결합 알고리즘 개발도 과제로서 의미가 있을 것이다.

참 고 문 헌

[1] Hui-I Hsiao, Ming-Syan Chen, and Philip S. Yu, "On Parallel Execution of Multiple Pipelined Hash Jois," *Proc. ACM SIGMOD*, pp.185-196, May 1994.

[2] Ming-Syan Chen, Ming-Ling Lo, Philip S. Yu, and Honesty C. Young, "Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins," *18th International Conference on VLDB*, pp.15-26, August 1992.

[3] Ming-Syan Chen, Mingling Lo, Philip S. Yu, and Honesty C. Young, "Applying Segmented Right-Deep Trees to Pipelining Hash Joins," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 7, No. 4, August 1995.

[4] Donovan A. Schneider and D.J. DeWitt, "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines," *Proceedings of the 16th VLDB Conference*, pp.469-480, August 1990.

[5] Mingling Lo, Ming-Syan Chen, C. V. Ravishankar, and Philip S. Yu, "On Optimal Processor Allocation to Support Pipelined Hash Joins," *Proc. ACM SIGMOD*, pp.69-78, May 1993.

[6] Ming-Syan Chen, P.S. Yu, and K.L. Wu, "Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries," *Proc. 8th International Conf. Data Engineering*, pp.58-67, Feb. 1992.

[7] Hui-I Hsiao, Ming-Syan Chen, "Parallel Execution of Hash Joins in Parallel Databases," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, pp.872-883, Aug. 1997.

[8] D.J.DeWitt, and J. Gray, "Parallel Database System: The Future of High Performance Database System," *Comm. of ACM*, pp.85-98, June 1992.

[9] 이규옥, 원영선, 홍만표, "페이지 실행시간 동기화 기법을 이용한 다중 파이프라인 해쉬 결합", 정보과학회 논문지:시스템 및 이론, 제27권, 제7호, pp. 639-649, 2000.

[10] Kyuock Lee, Youngsun Weon, and Manpyo Hong, "Multiple Pipelined Hash Joins Using Synchronization of Page Execution Time," *Int'l Conf. PDPTA'2000*, Vol. V, pp. 2863-2869, June, 26-29, 2000.

이 규 옥

1985년 충남대학교 계산통계학과 졸업. 1987년 충남대학교 계산통계학과 석사. 2001년 아주대학교 컴퓨터공학과 박사. 1987년 ~ 1989년 한국통력자원연구소 위촉연구원. 1989년 ~ 현재 한국기계연구원 선임연구원. 관심분야는 병렬데이타베이스, 병렬알고리즘, 컴퓨터그래픽스, 시스템 통합임.

원 영 선

1993년 경기대학교 전자계산학과 이학사. 1995년 경기대학교 전자계산학과 이학석사. 2001년 아주대학교 컴퓨터공학과 박사. 2001년 ~ 현재 충남대학교 정보통신공학부 BK 전임교수. 관심분야는 병렬처리, 병렬 데이타베이스, 병렬 알고리즘.

홍 만 표

서울대학교 자연과학대학 계산통계학과(1981). 서울대학교 자연과학대학 계산통계학과 석사(1983). 서울대학교 자연과학대학 계산통계학과 박사(1991). 1983년 ~ 1985년 울산공과대학 전자계산학과 전임강사. 1985년 ~ 현재 아주대학교 정보 및 컴퓨터공학부 교수. 1993년 ~ 1994년 미네소타대학 전자공학과 교환교수. 관심분야는 병렬처리