다중 프로그램 환경에 적합한 이중 연결 CC-NUMA 시스템

서 효 중†

요 약

다중 프로세서 시스템에서 여러 개의 프로그램이 동시에 수행될 경우의 프로그램 수행 성능은, 각 프로세스를 어떠한 물리적 위치의 프로세서에 할당하여 수행하는가에 따라 다르게 나타난다. 일반적으로 시 공간적으로 인접한 프로세서에 동일 프로그램의 프로세스를 할당할 경우 프로세스간 통신비용이 절감되므로 가장 효율적인 결과를 얻을 수 있다. 그러나 프로세스를 할당하는 운영체제는 이와 같은 친화성을 고려하기 위하여 부가적인 처리를 필요로 하며, 실제 수행시 각 프로그램은 독립적으로 수행되므로, 여러 프로그램으로부터 발생한 프로세스를 할당하는 방법은 많은 계산을 필요로 한다. 이중 링 구조의 CC-NUMA 시스템의 경우 특히 다수의 공유 메모리 접근에 의한 많은 트랜잭션이 발생하며, 연결망 부하의 불균등에 따른 병목 현상을 나타내므로, 프로세스의 할당 정책에 따라서 큰 성능 차이를 나타내게 된다. 본 논문은 균일한 연결망 부하특성을 나타내며, 프로세스 할당 정책을 필요로 하지 않는 CC-NUMA 시스템을 제시한다. 논문에서 제시하는 구조는 이중 링 구조와 동일한 연결망 비용을 나타내며, 건너뛰 연결을 이용한 균등한 부하 분배를 수행함으로써 프로세스 할당 정책의 유무와 무관한 성능을 보인다. 프로그램 구동 시뮬레이션을 통한 검증 결과 제시한 시스템은 이중 링 구조의 CC-NUMA 시스템에 비하여 1.5배의 성능 개선을 나타냈다.

A dual-link CC-NUMA System Tolerant to the Multiprogramming Environment

Hyo-Joong Suh†

ABSTRACT

Under the multiprogrammed situation, the performance of multiprocessor system is affected by the process allocation policy of the operating systems. The lowest communication cost can be achieved when the related processes positioned to the adjacent processors. While the effective allocation is quite difficult to the real situation, and the processing of the allocation policy consumes some computation time. The dual-ring CC-NUMA systems exhibit a quite performance difference according to the process allocation policy due to a lot of unbalanced memory transactions on the interconnection networks. In this paper, I propose a load balanced dual-link CC-NUMA system that does not requires the processes allocation policy. By the program-driven simulation results, the proposed system shows no remarkable difference according to the allocation policy while the dual-ring systems shows 10% performance improvement by the process allocation. In addition, the proposed system outperforms the dual-ring systems about 1.5 times.

키워드 : 다중 프로그램 환경(Multiprogramming Environment), 프로세스 할당 정책(Process Allocation), 이중 연결 CC-NUMA 시스템 (Dual-link Interconnection Network CC-NUMA Systems)

1. 서 론

다중 프로세서 시스템에서 동시에 여러 개의 프로그램이 수행될 때, 동일한 프로그램으로부터 생성된 여러 프로세스는 인접한 프로세서에 할당될 때 높은 성능을 나타낸다. 이러한 이유는 동일 프로그램에 속한 프로세스 간의 통신이 많이 발생하기 때문이며, 프로세스의 인접한 시간적 공간적 할당을 통하여 프로세스간 트랜잭션 전송 지연을 줄일 수 있기 때문이다[1]. 특히 Data General의 AViiON 시스템[2], 서울대학교의 PANDA 시스템[3]과 같은 고속의 SCI(Scalable Coherent Interface)[4] 점대 점 연결을 이용한 CC-NUMA(Cache Coherent Non Uniform Memory Access) 시스템[5]은, CC-NUMA 고유의 지역성 도출에 의한 성능 및 확장성 개선에 더하여 단위 연결의 대역폭 및 지연을 개선한 것이다. 하지만 점대 점 연결을 이용한 구조는 통신하고자 하는 프로세서의 물리적 거리에 따라 비례하여 전송 경로의 길이가 길어지며, 따라서 프로세스의 할당 위치에 따라 병렬 프로그램 수행 시간이 달라지므로 지역성을

※ 본 연구는 2004년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음.
† 정 회 원 : 가톨릭대학교 컴퓨터정보공학부 교수
논문접수 : 2003년 10월 11일, 심사완료 : 2004년 6월 9일

높게 활용하여야 하는 CC-NUMA 시스템의 성능에 영향을 준다[6].

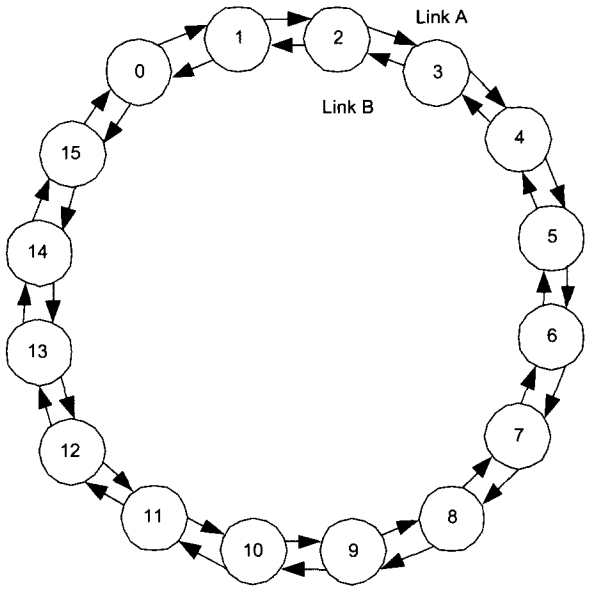
프로세스 할당 정책은 이러한 인접성을 이용하여 효율적인 프로그램 수행 성능을 도출하기 위한 것이다. 갱 스케줄링(gang scheduling) 기법[7]은 시간분할형(time-multiplexing) 스케줄링 방법으로, 각 프로그램에게 배타적인 시간을 할당하여 할당된 시간 만료와 함께 프로그램에 배정된 모든 프로세서를 재할당하는 방법을 사용한 것으로, 재할당 스케줄링 순서는 초기에 정적으로 결정된다. 2-단계 프로세서 집합(2-level processor set) 스케줄링[8]은 공간 공유형(space-sharing) 스케줄링 정책으로서, 스케줄러는 각 프로그램에 프로세서 풀을 배치하고 각 프로그램의 프로세스는 배치된 프로세서 풀 내에서 프로세서를 할당한다. 그러나 여러 개의 병렬 프로그램이 동적으로 시작되는 환경에서는 이와 같은 스케줄링을 적용하기 힘들며, 동적 환경에 대해 적절한 방법을 도출하기 어렵다.

본 논문은 이중 연결 구조를 이용한 CC-NUMA 시스템에서 공간적 지역성이 전체 시스템에 균등히 분배되도록 연결망을 구성하여, 다중 프로그램 운영 체제에서 부가적인 스케줄링을 적용하지 않아도 성능 저하를 발생하지 않는 구조를 제시하고, 스케줄링 정책을 적용한 이중 링 구조의 CC-NUMA 시스템에 대해 우수한 성능을 보임을 제시한 것이다. 본 논문의 구성은 다음과 같다. 2장에서 본 논문에서 제안하는 연결 구조와 다중 프로그램의 할당 방법을 설명하고, 3장에서 다중 프로그램을 위한 시뮬레이션 도구와 환경을 서술하며, 4장에서 시뮬레이션 결과를 제시하고, 5장에서 결론을 맺는다.

2. 이중 연결 CC-NUMA 시스템과 프로세서 할당

2.1 반대 방향 이중 링 구조

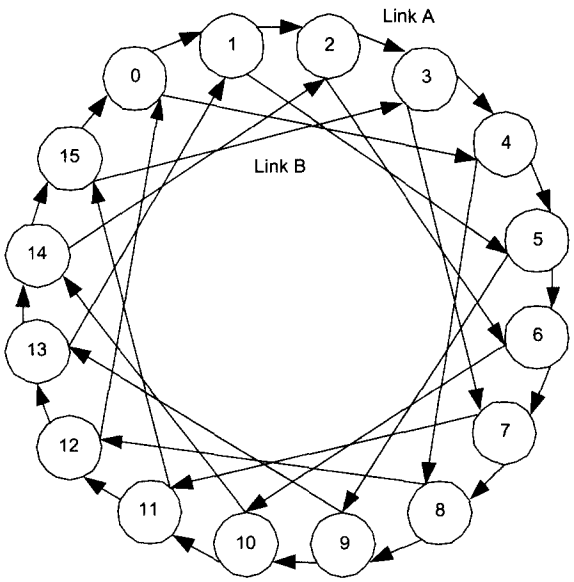
(그림 1)은 비교 대상으로 삼은 점 대 점 연결을 이용하여 이중 링 구조로 구성된 시스템이다. 일반적으로 각 노드는 한 개 내지 네 개의 정도의 프로세서로 구성되어 있으며, 각 노드는 서로 점 대 점 연결을 통하여 링 형태로 연결되어 있다. 두 개의 링크는 서로 반대 방향으로 연결되어 전송할 목적 노드의 위치에 따라서 가까운 거리 방향의 링크를 사용하도록 구성된다. SCI 링크 등을 이용한 단일 링 구조 시스템은 IBM의 NUMA-Q[9], Debois의 Express Ring [10], PANDA 시스템 등이 있으며, Data General의 AViiON 시스템과 PANDA-II 시스템에서 (그림 1)과 같은 이중 링 구조로 확장되었다. 이중 링 구조는 크로스바 스위치 등에 비하여 높은 성능을 나타냄이 알려져 있으며[11], 최근 고성능 시스템에서 채용되고 있다. 각 노드는 두 개의 연결 링크와 링크 제어기를 가지고 있다.



(그림 1) 반대방향 이중 링 구조의 시스템(16노드)

2.2 건너뜀을 갖는 이중 연결 구조

(그림 2)는 본 논문에서 제안하는 건너뜀을 갖는 이중 연결 구조의 CC-NUMA 시스템이다. 링크 A는 기존의 링 구조와 동일하게 연결되며, 링크 B는 링크 A와 동일한 방향으로 각 노드 간에 일정 수만큼 건너뜀을 가지고 연결된다.



(그림 2) 건너뜀을 갖는 구조의 시스템(16노드, 4 건너뜀)

2.3 트랜잭션의 처리

CC-NUMA 구조에서 트랜잭션은 캐시 일관성 유지 방법에 따라 다른 형태로 생성된다. 링 상에 발생하는 트랜잭션은 요청과 응답으로 분리된다. 스누핑에 기반한 일관성 유지 기법을 사용하는 경우, 요청은 모든 노드에 대하여 방송 형태로 전송된다. 반면 디렉토리 방식의 경우 Full-map 디렉토리 방식과 Chain 디렉토리 방식에 따라 달라지며, 해당되는 주소의 메모리에 공유된 모든 노드에 대한 정보를 유지하는 Full-map 디렉토리의 경우 트랜잭션은 해당되는 주소의 메모리를 가지고 있는 홈 노드로 우선 요청되고, 그 노드의 디렉토리 정보에 의하여 트랜잭션이 해당 주소를 캐싱하고 있는 노드에게로 전달되며, Chain 디렉토리 방식은 홈 노드로 트랜잭션이 요청되고 트랜잭션은 홈 노드로부터 해당되는 메모리를 가지고 있는 다음 노드로 전달되고, 다음 노드는 공유하고 있는 또 다른 노드에 대한 링크를 가지고 있으므로 요구된 트랜잭션에 따라 다시 다음 노드로 요청을 전달된다[12].

디렉토리 방식의 일관성 유지의 경우 주로 일대 일 방식의 트랜잭션이 사용되며, 스누핑에 비하여 여러 횟수의 트랜잭션 단계가 발생하므로, 링 구조와 같이 방송에 적합한 형태의 시스템에서는 스누핑에 의한 일관성 유지 방법이 보다 높은 성능을 나타낸다[11].

본 논문에서 설정한 일관성 유지 방법은 스누핑 방식이므로, 각 트랜잭션은 다음의 <표 1>과 같은 종류가 발생하며, 각 트랜잭션에 대하여 방송 구조의 요청 트랜잭션의 경우 링크 A와 B를 통하여 동시에 생성시켜 병행적인 방송 처리를 할 수 있도록 하였고, 일대 일 전송의 경우 가장 짧은 단계를 가지는 경로로 트랜잭션이 전달되도록 하였다.

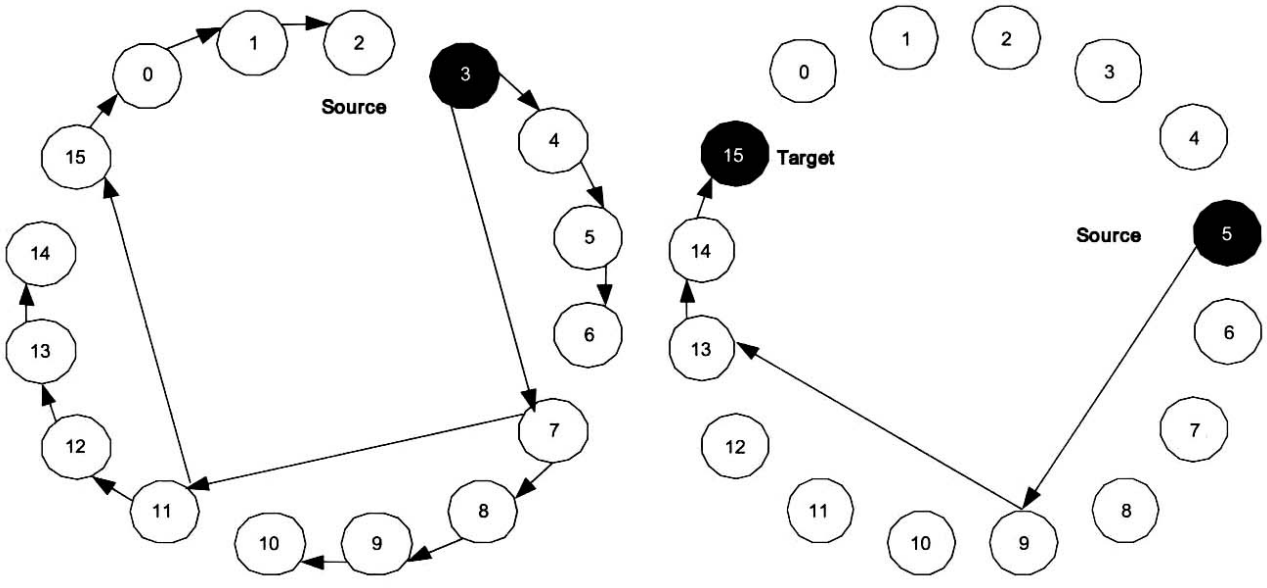
〈표 1〉 트랜잭션의 종류와 유형

트랜잭션 이름	트랜잭션 종류
읽기/쓰기 요청	방송 트랜잭션
무효화	
읽기/쓰기 응답	일 대 일 트랜잭션
되쓰기	

다음 (그림 3)은 16노드 구조에서 각각 방송 트랜잭션과 일대 일 전송 트랜잭션의 전달 경로를 나타낸 것이다.

반대방향 이중 링 구조에서 방송 트랜잭션은 모든 노드에 전달되어야 하므로, 노드의 수와 동일한 단계의 경로를 가지게 되며, 본 논문에서 제안한 구조는 링크 A와 B를 통하여 동시에 트랜잭션을 전달하므로, 보다 짧은 지연을 나타낸다.

일대 일 트랜잭션의 경우, 이중 링에서 링의 정 반대에 있는 노드간에 트랜잭션 전송이 일어날 때 가장 긴 시간 지연이 나타나며, 총 노드의 개수를 N개라 할 때, 전송 단계는 $N/2$가 된다. 반면, 논문에서 제안한 구조의 경우, 링크의 회전방향으로 자신의 직전 노드에게 일대 일 트랜잭션을 전송할 때 가장 긴 경로가 설정된다. 이 때의 전송 경로 단계는 노드의 수를 N, 건너뜀의 수를 I라 할 때, $N/I + I - 2$ 이다. 따라서 본 논문에서 제안한 구조는 방송 트랜



(a) 방송 트랜잭션의 전달 경로 (b) 일대 일 트랜잭션의 전달 경로

(그림 3) 방송 트랜잭션과 일대 일 트랜잭션의 전달 경로(16노드, 4 건너뜀)

잭션에서 어느 정도 지연 개선을 나타내게 되며, 일대 일 전송의 경우 건너뜀 수를 가장 작은 값인 $I=2$, 가장 큰 값인 $I=N/2$로 할 때 양방향 이중 링과 동일한 최대 지연을 나타낸다. I가 최대와 최소 사이의 건너뜀 값으로 사용될 경우, 이보다 적은 지연을 나타내게 된다. 건너뜀 수의 적절한 선택은 방송 트랜잭션과 일대 일 트랜잭션의 전송 경로 길이에 의하여 선택하여야 하며, 반대방향 이중 링 구조에 비하여 방송 트랜잭션의 전송 경로 길이의 개선은 $N-(N/I+I)+1$ 이고, 일대 일 트랜잭션의 평균 전송 경로 길이의 개선은 $N(N-4N/I+6)/8$ 이 된다.

2.4 다중 프로그램 환경에서의 프로세스 할당

병렬 프로그램이 수행될 경우, 동일 프로그램에서 발생한 프로세스는 상호간 공유 데이터를 가지게 된다. 공유 데이터는 프로세스간에 양방향 접근이 일어나게 되므로, 공유 데이터가 병렬 프로세스를 수행하는 프로세서 집합과 가까이 배치될 때 높은 성능을 나타낸다. 반대방향 이중 링의 경우 운영 체제는 동일 프로그램의 프로세스들을 인접 노드로 할당할 경우 가장 높은 성능을 기대할 수 있으며, 이러한 친화성(affinity)을 고려하지 않을 경우, 분산된 지역 메모리를 통한 지역성으로 성능 향상을 꾀하는 CC-NUMA 시스템에서 특히 큰 성능 저하가 일어난다[13]. 논문에서 제안한 건너뜀 링크를 갖는 구조의 경우 일대 일 전송 경로의 최대 단계수가 반대방향 이중 링에 비하여 적으므로, 친화성을 고려하지 않은 할당이 일어나도 성능 저하의 정도는 보다 적어진다.

본 논문에서 실험한 할당 기법은 복수 개의 프로그램 수행에서 각 프로그램이 요구하는 프로세서 수의 총합이 전체 시스템에 존재하는 모든 프로세서의 수보다 적은 경우를 대상으로 하였다. 이 경우는 복수 개의 프로그램에 존재하는 모든 프로세스가 각각의 프로세서에 할당될 수 있는 경우이다. 따라서 운영 체제는 프로세스를 스케줄할 때, 물리적으로 인접한 프로세서에 프로세스를 할당하는 것으로 시간적/공간적 지역성을 얻을 수 있다. 이러한 형태로 프로세스가 수행되면 스케줄에 따른 캐시 교체 등이 발생하지 않으므로, 단지 프로세스의 시간적/공간적 트랜잭션 전송 지연에 따른 성능만이 관계되며, 다양한 스케줄링 정책에서 목적하고자 하는 지역성을 최대한 활용할 수 있다.

3. 시뮬레이션 방법

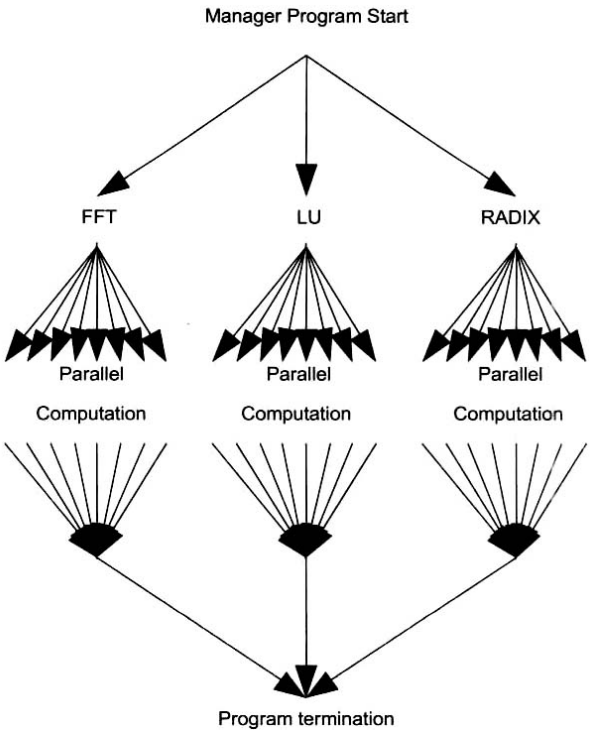
여러 프로세서를 사용하는 시스템의 시뮬레이션 도구로 Augmint를 사용하였다[14]. Augmint는 x86 기계의 프로그램을 추적할 수 있도록 구성된 다중 프로세서 시뮬레이터로, 프로그램의 메모리 접근을 추적한다. 시뮬레이터 프로그램은 전단부와 후단부로 나뉘는데, 전단부는 프로세서로부터 발생하는 메모리 접근을 생성하고, 후단부는 모든 메모리 접근 경로 및 네트워크를 구현한다.

성능 평가에 사용한 프로그램은 다중 프로세서 성능측정에 다수 사용되는 SPLASH-2 벤치마크[15] 프로그램 중 다음 <표 2>와 같은 세 개의 프로그램을 사용하였다.

〈표 2〉 시뮬레이션에 이용된 프로그램 및 부하

프로그램	부 하
FFT	-m14 -p8 -n2048 -l5
LU	-n128 -p8 -b16
RADIX	-p8 -n9000 -r1024 -m2097152

세 개의 벤치마크는 각각 별개의 단위 프로그램이고, 시뮬레이터에서 이러한 프로그램의 동시 수행을 지원하지 않으므로, 벤치마크 프로그램의 관리 프로그램을 작성하였으며, 관리 프로그램에서 각 프로그램을 시작하도록 수정하였다. 다음 (그림 4)는 이러한 수행 구조를 나타낸 것이다.



(그림 4) 다중 프로그램의 수행 형태

세 프로그램의 부하는 동일하지 않고 각 프로그램은 병렬 계산을 위한 준비 및 초기화 단계가 각각 존재한다. 따라서 각 프로그램을 병렬로 수행한다 하여도, 실제 병렬 계산 부분이 겹쳐 수행되도록 하지 않으면 목적한 다중 프로그램 환경이 제대로 시험되지 않으므로 각 프로그램의 초기화 및 병렬 계산을 위한 준비를 완료한 후, 각 프로그램이 동시에 계산을 시작하도록 하였다.

각 프로그램의 수행 시간은 차이가 있으므로, 종료 시간을 각각 측정하고, 전체 프로그램의 수행 시간 또한 측정하였다. 전체 프로그램의 수행 시간은 세 프로그램 중 가장 오래 걸린 프로그램이 종료된 시간이므로, 전체 프로그램 수행 시간은 다중 프로그램 수행 성능에 비례하지 않는다.

시뮬레이션을 위하여 사용한 시스템은 32노드를 가지도록 하였으며, 각 노드 내에는 한 개의 프로세서를 가지고 있는 형태를 가정하였고, 앞서 제시한 방송 트랜잭션과 일대 일 트랜잭션 전송 경로에 대하여 가장 높은 값을 가지는 8의 건너뜀 수를 갖도록 설정하였다. 기타 변수는 다음 <표 3>과 같이 최신 프로세서의 속도 및 링크의 속도를 기준으로 하였다.

〈표 3〉 실험 대상 시스템 환경

항 목	값
프로세서 클럭 속도	1GHz
시스템의 프로세서 수	32개
프로세서 당 캐시	128Kbyte, 4way, LRU
노드 개수	32개
노드 내 버스 속도	266MHz
노드 당 원격 캐시	512Kbyte, 8way, LRU
링크 전송 대역폭	1Gbyte/s

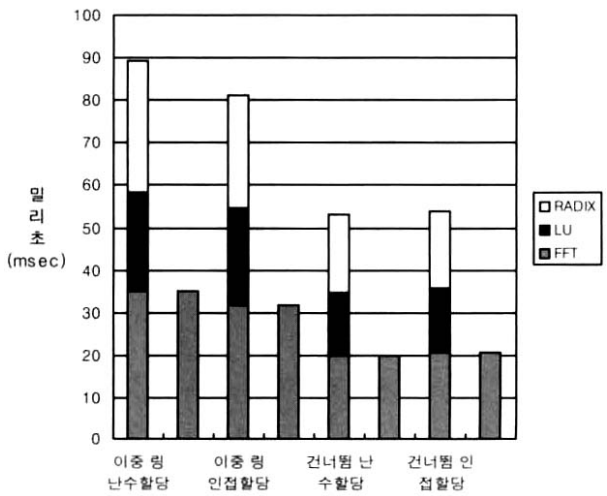
각 구조에 사용한 프로세서 할당 방법은 두 가지를 적용하였다. 첫 번째는 프로세스를 시스템 상의 프로세서에 난수적으로 할당하여 링크 A 또는 B의 진행 경로상 인접한 위치와 무관하게 배치되도록 한 것이고, 두 번째는 서로 다른 프로그램으로부터 생성된 프로세스를 연결상 멀리 배치하고, 동일 프로그램에서 생성되는 프로세스는 링크 A 또는 B 경로상의 인접 위치에 배치되도록 할당한 것이다. 첫 번째 방법은 운영 체제에서 스케줄을 위한 정책이 없는 경우에 해당되며, 두 번째 방법은 운영 체제에서 스케줄러를 통하여 프로세스간의 연관을 가지고 프로세서에 배치한 경우이다. 두 번째 경우에 운영 체제의 스케줄 정책 수행을 위한 부가적인 처리 시간은 시뮬레이션 시간에 포함하지 않았다.

4. 시뮬레이션 결과

4.1 프로그램 수행 시간

(그림 5)는 각 프로그램의 수행시간의 합이다. 실제로는 세 프로그램은 동시에 수행하므로 세 프로그램이 동시에 시작해서 종료되기까지의 시간은 각 항목의 오른쪽 막대로 표시하였다.

이중 링 난수 할당은 반대방향 이중 링 구조에서 난수를 발생시켜 프로세서에 할당하였으므로 스케줄링이 없이 할당한 경우에 해당되며, 이중 링 인접 할당은 동일한 프로그램으로부터 발생한 프로세스를 인접한 노드에 위치시킨 것이므로 친화성에 따른 스케줄링을 수행한 경우에 해당된다. 건너뜀 난수 할당과 건너뜀 인접 할당은 본 논문에서 제안한 구조에서, 스케줄링을 적용하지 않는 경우와, 인접성에 의한 스케줄링을 적용하여 링크 A 및 B의 경로상 가까운 단계에 있는 프로세서에 할당한 경우이다.



(그림 5) 각 프로그램 수행 시간의 합과 전체 프로그램 수행 시간

각 축의 오른쪽 작은 막대는 전체 프로그램이 수행된 시간인데, 세 프로그램 중 FFT가 가장 오랜 시간이 걸렸기 때문에 전체 프로그램의 종료 시간은 FFT 시간과 동일한 결과를 나타냈다.

이중 링 구조의 경우 그림에서 나타나듯이 스케줄링을 적용하지 않을 경우 10%의 성능 저하를 나타낸다. 반면 논문에서 제안한 건너뜀 구조의 경우, 난수할당과 인접할당은

유의한 성능 차이를 나타내지 않았으며, 이는 건너뜀 구조로 인하여 낮아진 트랜잭션 지연 차이와 균등하게 분배된 연결망 부하에 의한 것이다.

4.2 트랜잭션 처리 지연 시간

다음 <표 4>는 각 프로세서로부터 발생한 노드간 트랜잭션에서 요청이 시작될 때부터 응답이 완료될 때까지 걸린 평균 지연 시간이다. 프로세서로부터 읽기 또는 쓰기에 의해 노드간 트랜잭션이 발생되는 것은 캐시 실패에 의한 노드간 데이터 전송과 다른 노드의 캐시를 무효화 하기 위한 일관성 관련 트랜잭션이 발생한 경우이므로, 이러한 지연이 적을수록 프로그램의 수행 시간이 빨라지게 되고, 보다 짧은 트랜잭션의 처리 시간을 나타낸다.

〈표 4〉 트랜잭션 처리 지연 시간

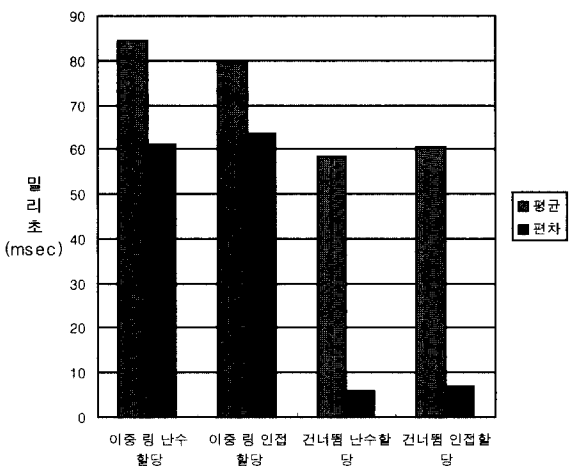
구 조	프로세서 할당정책	사이클
반대방향 이중 링	난 수	2376
	인접할당	2291
8 노드 건너뜀	난 수	1246
	인접할당	1259

표에서 나타난 것과 같이 프로세스 할당 정책의 유무에 따른 지연 시간의 차이는 반대방향 이중 링 구조에서 뚜렷이 나타났으며, 논문에서 제안한 건너뜀 링크 구조의 경우 훨씬 작은 차이를 보여주었다. 또한 논문에서 제안한 구조는 반대방향 이중 링 구조에 비하여 상당히 적은 지연을 보여주고 있음을 알 수 있다.

4.3 링크 점유 정도

반대방향 이중 링 구조에 비해 논문에서 제안한 건너뜀을 갖는 구조에서 보다 짧은 노드간 전송 거리를 나타냄은 앞서 제시된 바 있다. 그러나 실제 프로그램이 수행되는 상황에서의 트랜잭션 지연은 전송 거리에 더불어 링크에 대한 경쟁이 중요한 역할을 하게 된다. 이러한 경쟁은 프로그램 수행 시에 동적으로 나타나며, 특정 링크에 트랜잭션의 전송이 집중되는 현상이 나타나게 될 경우, 노드간의 물리적 경로에 따른 지연에 비하여 링크에 대한 경쟁으로 인한 지연이 더 크게 나타날 수 있으며, 이 경우에 프로세스의 인접 배치에 따른 이득은 상당히 감소될 수 있다.

(그림 6)은 전체 프로그램의 수행 시간 동안 트랜잭션에 의해 발생한 각 링크의 점유 시간에 대한 평균 및 편차를 계산한 것이다.



(그림 6) 링크 점유 시간의 평균 및 편차

반대방향 이중 링 구조의 경우, 건너뜀을 갖는 경우에 비하여 높은 편차를 나타내며 이는 상호연결망이 균일하게 사용되지 못함을 나타낸다. 이중 링 구조에서 각 프로세스를 서로 가까운 위치의 프로세서에 할당할 경우, 인접한 노드 사이에 트랜잭션이 다수 발생하게 되므로, 특정 링크에 대한 경쟁이 인접한 노드 배치에 따른 성능 이득을 저하시킬 수 있다. (그림 6)의 반대방향 이중 링 구조와 같이 큰 편차를 나타낸 경우 링크의 사용이 특정 위치에 집중되었음을 나타낸다. 반면 건너뜀 연결을 갖는 구조는 링크 점유간에 상당히 낮은 편차를 보이며, 이러한 성질은 전체 링크가 비교적 균일하게 사용되고 이에 따라서 보다 낮은 경쟁이 나타났음을 의미한다. 결국 건너뜀 경로가 트랜잭션을 균일하게 분배시키는 역할을 하고 있음을 알 수 있다. 트랜잭션이 특정 링크에 집중되었을 경우, 링크에 대한 재시도가 발생하게 되며, <표 5>는 트랜잭션이 링크에 대한 경쟁에 의하여 재시도된 횟수이다.

〈표 5〉 트랜잭션이 경쟁에 의하여 재시도된 횟수

구 조	프로세서 할당정책	재시도 횟수
반대방향 이중 링	난 수	73968
	인접할당	78044
8 노드 건너뜀	난 수	7012
	인접할당	7423

인접할당 정책을 사용하였을 때 두 경우 모두 높은 재시도 횟수가 나타났으며, (그림 6)에서 인접 할당이 보다 높은 편차로 나타난 결과와 부합한다. 또한 건너뜀을 갖는 구

조의 경우, 반대방향 이중 링 구조에 비하여 10% 미만의 재시도 횟수로 효율적인 연결망 사용이 이루어졌음을 알 수 있다.

5. 결 론

운영 체제의 일반적인 형태인 다중 사용자 및 다중 프로그램 환경에서, 연관된 프로세스를 인접한 프로세서에 할당하여 상호연결망의 경로에 따른 지연을 줄이기 위한 노력은 다중 프로세서 시스템의 프로그램 수행 능력을 높이기 위하여 시도되었다. 이중 링 구조의 CC-NUMA 시스템과 같이 분산된 메모리 구조로 물리적 거리에 의한 트랜잭션 지연 시간의 현저한 차이를 나타내는 시스템의 경우, 스케줄링 정책에 의하여 어느 정도의 성능향상을 기대할 수 있는 반면 프로세스의 할당 정책을 관리하기 위한 부가적 처리가 발생되는 단점이 존재하였다.

본 연구는 이중 연결을 갖는 CC-NUMA 시스템에서 건너뜀 연결 구조를 갖도록 배치하여 균등한 부하 분배 및 트랜잭션 처리 지연의 감소를 꾀하였으며, 실험 결과, 반대방향 이중 링 구조의 경우, 친화성에 의한 프로세스 할당에 따라 10%의 성능 개선을 나타내었으나, 건너뜀을 갖는 구조의 경우, 유의한 성능 차이를 나타내지 않았다. 이러한 이유는 건너뜀을 갖는 구조가 반대방향 이중 링 구조에 비하여 짧은 노드간 경로를 가지고 있고, 트랜잭션의 발생에 의하여 발생하는 링크의 점유가 전체 링크에 골고루 분산되어 나타나는 특징을 가지고 있으며, 분산된 트랜잭션 부하로 낮은 트랜잭션 재시도 횟수를 나타내어 프로세스의 노드 할당에 따라서 지연의 차이가 없기 때문이다.

결과적으로 본 논문에서 제시한 건너뜀 경로를 가지는 이중 연결 구조의 CC-NUMA 시스템을 구성할 경우, 운영체제 수준의 친화성을 고려한 프로세스 할당 정책이 필요하지 않으며, 다중 프로세서 시스템에서 보다 간단하고 하드웨어 독립적인 운영 제제의 구현과 함께 높은 성능의 트랜잭션 처리능력을 얻을 수 있음이 확인되었다.

참 고 문 헌

[1] A. Gupta, A. Tucker and S. Urushibara, "The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications," In Proc. of SIGMETRICS, pp.120-132, 1991.

[2] http://www.dg.com/.

[3] http://panda.snu.ac.kr/nrl/.

[4] IEEE Computer Society, IEEE Standard for Scalable Coherent Interface(SCI), Institute of Electrical and Electronics Engineers, Aug., 1993.

[5] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz and Monica S. Lam, "The Stanford Dash multiprocessor," Computer, Vol.25, No.3, pp.63-79, Mar., 1992.

[6] G. Agarwal, Lim, Kranz and Kubiatowicz, "APRIL : A processor architecture for multiprocessing," Proc. of the 17th Annual International Symp. on Computer Architecture, pp. 104-114, May, 1990.

[7] Y. Zhang, H. Franke, J. E. Moreira and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," Proc. of International Parallel and Distributed Processing Symp., pp.133-144, May, 2000.

[8] D. L. Black, "Scheduling support for concurrency and parallelism in the mach operating system," IEEE Trans. on Computer, pp.35-43, May, 1990.

[9] Tom Lovett and Russel Clapp, "STiNG : A CC-NUMA Computer System for the Commercial Marketplace," Proc. of the 23th International Symp. on Computer Architecture, pp.308-317, May, 1996.

[10] L. Barroso and M. Dubois, "The Performance of Cache-Coherent Ring-based Multiprocessors," Proc. of the 20th International Symp. on Computer Architecture, pp.268-277, May, 1993.

[11] Hitoshi Oi and N. Ranganathan, "A Comparative Study of Bidirectional Ring and Crossbar Interconnection Networks," Proc. of the 1998 International Conf. on Parallel and Distributed Processing Techniques and Applications, pp.883-890, Jul., 1998.

[12] S. Gupta and S. Abraham, "A distributed directory cache coherence scheme and its effects on network performance," Journal of High Performance Computing, Vol.2, No.1, pp. 3-16, Nov./Dec., 1995.

[13] E. P. Markatos and T. J. LeBlanc, "Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors," IEEE Trans. on Parallel and Distributed Sys-

tems, Vol.5, No.4, pp.379-400, Apr., 1994.

[14] A-T. Nguyen, M. Michael, A. Sharma and J. Torrellaz, "The Augmint multiprocessor simulation toolkit for Intel x86 architecture," Proc. of the IEEE International Conf. on Computer Design, Oct., 1996.

[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. "Methodological considerations and characterization of the SPLASH-2 parallel application suite," Proc. of the 22th Annual International Symp. on Computer Architecture, pp.24-36, 1995.

서 효 중

e-mail : hjsuh@catholic.ac.kr

1991년 서울대학교 이학사

1994년 서울대학교 공학석사(컴퓨터공학)

2000년 서울대학교 공학박사(컴퓨터공학)

2002년 지씨티 리서치 선임연구원

2003년~현재 서울대학교 컴퓨터연구소 객원연구원

2003년~현재 가톨릭대학교 컴퓨터정보공학부 전임강사

관심분야 : 컴퓨터 구조, 병렬처리 시스템, 내장형시스템, 클러스터 시스템