토러스 다중컴퓨터를 위한 입출력 자원의 배치와 성능 분석*

Placement and Performance Analysis of I/O Resources for Torus Multicomputer

안중석** · 강준효*** · 김종현***
Joong Suk Ahn · Jun Hyo Kang · Jong Hyun Kim

Abstract

Performance bottleneck of parallel computer systems has mostly been I/O devices because of disparity between processor speed and I/O speed. Therefore I/O node placement strategy is required such that it can minimize the number of I/O nodes, I/O access time and I/O traffic in an interconnection network. In this paper, we propose an optimal distance-*k* embedding algorithm, and analyze its effect on system performance when this algorithm is applied to n x n torus architecture. We prove this algorithm is an efficient I/O node placement using software simulation. I/O node placement using the proposed algorithm shows the highest performance among other I/O node placements in all cases. It is because locations of I/O nodes are uniformly distributed in the whole network, resulting in reduced traffic in the interconnection network.

1. 서론

최근 컴퓨터 시스템들은 프로세서와 반도체 주기억장치 속도의 상승에 힘입어 성능이 계속 향상되고 있으며, 특히 다수의 프로세서들로 구성되는 병렬컴퓨터 시스템의 개발로 인하여 시스템 성능은 크게 향상되고 있다. 그러나 이러한 시스템들에서는 선형적인 성능 향상을 위하여 프로세서들 간의 상호 연결 방식, 기억장치 배치 방법, 캐쉬 일관성 유지 문제, 상호연결망의 성능 등 설계 시의 고려해야 할 사항이 많이 존재하고, 따라서 다양한 종류의 시스템 구조들이 제시되어 왔다[1,9,15]. 이러한 시스템들은 프로세서들 간의 기억장치 공유 여부에 따라 밀결합 시스템(tightly-coupled system)과 소결합 시스템(loosely-coupled system)으로 나

* 본 연구는 한국과학재단 핵심전문연구비(951-0910-072-2) 지원으로 수행되었음.
** 한컴기술연구소 개발실
*** 연세대학교 전산학과

눌 수 있는데, 최근에는 두 가지 구조가 혼합되어 많은 수의 프로세서들을 가진 대규모 병렬컴퓨터시스템으로 발전하고 있다. 그러한 시스템에서 최근에는 반도체 기술의 발전으로 하드웨어적인 라우팅을 사용하는 메쉬(mesh)나 토러스(torus) 구조가 많이 채택되고 있는 추세이며, 앞으로 대규모 병렬 컴퓨터에서는 *k*-ary *n*-cube 구조에 기반을 둔 상호연결망이 널리 사용될 전망이다 [12].

여러 개의 프로세서들이 하나의 시스템을 이루고 있는 병렬컴퓨터시스템에서는 액세스를 많이 받는 자원(resource)의 위치에 따라 상호연결망의 통신량이 달라지며, 그에 따라 통신 지연 시간이 많은 영향을 받게 된다. 시스템 성능에 영향을 주는 자원들 중에서 I/O 장치들은 빈번히 액세스를 받음에도 불구하고 프로세서의 속도 향상에 비해 속도가 느리게 발전함으로 인하여 전체 시스템 내에서 병목으로 작용하여 성능 향상에 큰 저해 요인이 되었음은 많은 연구를 통해 밝혀진 바 있다 [10]. 따라서 다중프로세서 시스템의 I/O 노드 배치 방법에 대한 연구의 중요성이 부각되고 있다.

하이퍼큐브를 사용한 초기의 시스템인 Intel iPSC/1에서 각 프로세서 노드들은 계산만 전담하였으며 모든 I/O 기능들은 호스트 컴퓨터(host computer)가 담당하였다. 개선된 모델인 iPSC/2 시스템에서는 I/O 성능을 향상시키기 위하여 다수의 디스크들을 사용하여 화일들을 디클러스터링(declustering)시켜 저장함으로써 디스크 액세스의 동시성을 높이는 방법을 사용하였다[15]. NCUBE 시스템에서는 8 개씩으로 이루어진 서브큐브(subcube) 마다 한 개씩의 I/O 프로세서를 배치하여 I/O 부담을 분산시켰다. 그러나 이러한 구조로 I/O 대역폭을 향상시키는 데에는 한계가 있었고, 따라서 대규모 응용들을 효율적으로 또는 고속으로 처리하는 것이 불가능하였다. 그러한 I/O 병목 현상을 해결하기 위하여 최근의 다중프로세서 시스템들에서는 모든 노드들 또는 일부분의 노드들이 I/O 제어 기능을 포함하고 있다. Intel Paragon 시스템에서는 모든 계산 노드들이 메쉬 네트워크를 통하여 연결되어 있고, I/O 노드들은 메쉬 구조의 우측과 좌측에 별도의 행(column)들을 추가하여 접속되어 있다[9].

이와 같이 시스템 내의 모든 노드들 또는 일부분이 계산 기능과 I/O 기능을 모두 가지도록 함으로써 I/O 대역폭이 크게 향상될 수 있었다. 그러나 Paragon 시스템과 같이 별도의 I/O 노드들을 기본적인 n x n 메쉬 구조의 외부에 추가하면 계산 노드들의 I/O 액세스 시간이 위치에 따라 크게 차이가 나게 된다. 또한 I/O 노드들이 모여 있기 때문에 그들과 인접한 노드들에는 다른 노드들로부터의 I/O 요구들을 처리하기 위한 I/O 통신들이 많이 통과하므로 통신량이 집중되어 일반적인 프로세서간 통신의 속도를 크게 저하시키는 문제가 발생한다. 여기에서 프로세서간 통신이란 다른 노드에 위치한 기억장치의 액세스 또는 프로세서간의 데이타 교환을 위한 통신을 의미한다. 따라서 가능한한 적은 수의 I/O 노드들을 사용하면서도 I/O 액세스 시간을 최소화시키고, 또한 상호연결망에서의 I/O 통신량을 최소화시킬 수 있는 최적의 I/O 구조를 설계할 필요가 있다.

Ghosh et al.[8]은 하이퍼큐브 다중프로세서 시스템에서 프로세서간 통신과 I/O 통신을 분리시키기 위하여 별도의 링크(link)들을 외부에 추가하여 I/O 노드들을 접속하는 방법을 제안하였다. 그러나 이 경우에는 I/O 링크와 접속되는 특정 노드들에는 추가되는 링크들을 위한 통신 채널들을 별도로 포함해야 하는 문제가 발생한다. Reddy and Banerjee[14]는 별도의 링크를 추가시키지 않고 기본적인 노드들 중의 몇 개를 선택하여 I/O 노드로서의 기능을 수행하도록 함으로써 I/O 접속을 위한 하드웨어 오버헤드를 최소화시키는 방법을 제안하였다. 이와 같이 기본적인 구조를 유지하면서 선택된 노드들만 I/O 기능을 가지게 하는 방식을 I/O 임베딩(I/O embedding)이라고 부른다. 이러한 방식에서 가장 중요한 설계 요소들은 I/O 노드의 최적 갯수와 위치의 결정이다.

토러스 구조는 단순하며 규칙적이고 확장성도 뛰어나기 때문에 앞에서 살펴본 바와 같이 최근에 개발된 대규모 병렬컴퓨터에서 많이 채택되고 있다. 그러나 토러스 구조를 기반으로 한 다중프로세서 시스템에서의 I/O 임베딩에 관한 연구는 수행된 바가 없다. 본 논문에서는 최적 거리-*k* 임베딩 알고리즘을 제시하고, 이것을 n x n 토러스 구조에 적용하여 시스템에 미치는 영향을 분석함으로써 이 알고리즘이 토러스 구조에서 최적의 I/O 임베딩 방식임을 실험적으로 증명하고자 한다.

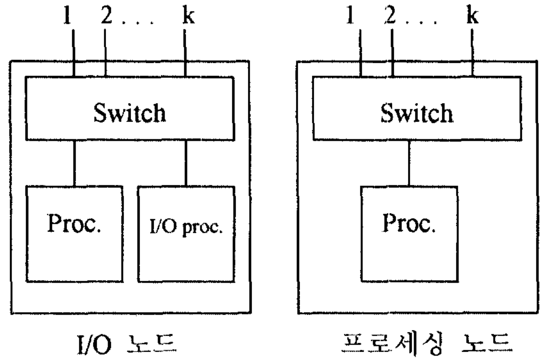
본 논문의 구성은 다음과 같다. 제2장에서는 I/O 임베딩과 이 분야에서의 이전 연구들에 대하여 살펴 본 뒤 최적 거리-k 임베딩 알고리즘을 제시하고, 제3장에서는 본 연구에서 사용한 소프트웨어 시뮬레이터에 관하여 설명한다. 제4장에서는 시뮬레이션 방법과 결과에 대하여 논하고, 마지막으로 제5장에서 본 연구에 대한 결론을 정리한다.

2. I/O 노드 배치 알고리즘

다중프로세서 컴퓨터시스템에서 I/O 장치들을 분산 배치하는 방법은 여러 연구팀들에 의하여 연구된 바가 있다. 본 절에서는 그와 관련된 이전의 연구들에 대하여 살펴본 뒤, 본 연구에서 제안하는 토러스 구조를 위한 최적 I/O 임베딩 알고리즘에 대하여 설명하고자 한다.

2.1 I/O 임베딩

I/O 임베딩이란 시스템의 기본적인 구조를 유지하면서 선택된 노드들만 I/O 기능을 가지도록 하는 방식을 말한다. 이와 같이 몇 개의 노드들만 I/O 기능을 가지도록 할 때에는 I/O 노드 수의 결정과 I/O 노드의 위치가 중요하게 되며, 이들을 결정하는 것을 I/O 임베딩 문제(I/O embedding problem)라고 한다[13]. I/O 노드의 수가 많게 되면 I/O 액세스가 빨리 이루어지는 장점이 있으나 반대로 각 노드의 효율이 떨어지는 단점이 있으며, I/O 노드들이 한쪽에 치우쳐 분포하게 되면 상호 연결망 내에서 hot-spot을 야기하게 된다. 이와 같은 I/O 임베딩을 위한 노드들의 구조를 살펴보면 〈그림 1〉과 같다.

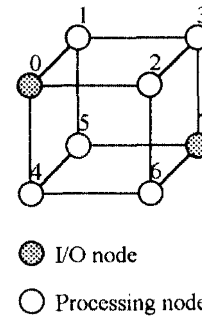


〈그림 1〉 노드의 구조

Reddy et al.[13]은 하이퍼큐브 혹은 2진 n-cube 시스템에서 완전 I/O 임베딩(perfect I/O embedding) 방식을 제안하고, 일반적인 k-regular 네트워크에서 I/O 임베딩 문제가 NP-complete임을 보였다. Reddy et al.은 완전 I/O 임베딩 조건을 다음과 같이 정의하고 있다.

정의 1: 시스템 내의 각 노드가 반드시 한 개의 I/O 노드와만 인접하도록 위치시키는 경우를 완전 I/O 임베딩이라고 한다.

즉, 모든 계산 노드들이 반드시 한 개의 I/O 노드와 인접(adjacent)해야 하며, 두 개 또는 그 이상의 I/O 노드들과 인접하지 않아야 한다. 그리고 I/O 노드들 사이의 최소 거리는 3이다. 그들의 연구 결과에 따르면 하이퍼큐브에서는 임의의 정수 i에 대하여 $n=2^i-1$인 n-차원 구조의 하이퍼큐브에서만 완전 I/O 임베딩이 존재하였다. 완전 I/O 임베딩의 경우를 예를 들어 살펴보면 위에서 제시된 조건에 의하여 3-차원(2^2-1)의 경우는 〈그림 2〉와 같다. 여기에서 보이는 바와 같이, 특정한 노드에 상관없이 서로 대칭되는 노드들에 존재하기만 하면 완전 I/O 임베딩 조건을 만족하는 것을 알 수 있다. 따라서 3-차원 하이퍼큐브는 완전 I/O 임베딩을 가지며, I/O 노드의 위치는 (0,7), (1,6), (2,5), (3,4) 등이 될 수 있다.



〈그림 2〉 하이퍼큐브에서의 완전 I/O 임베딩

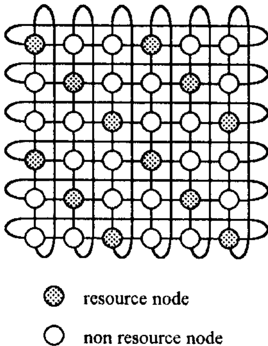
이러한 I/O 임베딩 문제는 I/O라는 특정한 자원이 아닌 일반적인 자원 할당 문제(resource allocation problem)로 확장될 수 있다. Ramanathan과 Chalasani[12]는 k-ary

n-cube 시스템을 위한 완전 j-인접 임베딩(perfect j-adjacency embedding) 조건을 정의하고, 완전 j-인접 임베딩이 존재하지 않을 경우를 위해 준완전 j-인접 임베딩(quasi-perfect j-adjacency embedding)을 위한 알고리즘을 제시하였다. 그들은 완전 j-인접 임베딩과 준완전 j-인접 임베딩을 다음과 같이 정의하고 있다.

정의 2: 자원을 가지고 있는 노드(resource node : 이하 자원 노드라 함)들은 서로 인접하지 않으며, 자원을 가지고 있지 않는 노드(non resource node : 이하 비자원 노드라 함)들은 자원을 가지고 있는 j 개의 노드들과 인접한 경우를 완전 j-인접 임베딩이라 한다.

정의 3: 자원 노드들은 서로 인접하지 않으며, 비자원 노드들은 j개 혹은 $(j+1)$개의 자원 노드들과 인접한 경우를 준완전 j-인접 임베딩이라 한다.

완전 j-인접 임베딩에서 비자원 노드는 j개의 자원 노드들과 인접해 있으므로 자원이 필요할 때마다 그 자원을 신속히 액세스할 수 있을 것이다. 〈그림 3〉은 6 x 6 토러스에서 완전 2-인접 임베딩의 예이다. 이 그림에서 보는 바와 같이, 자원을 가지고 있지 않은 노드들은 모두 두 개의 자원 노드들과 인접하고 있음을 알 수 있다.



〈그림 3〉 6-ary 2-cube에서 완전 2-인접 임베딩

정의 1은 계산 노드들이 반드시 하나의 I/O 노드와 인접하고 있는 경우만을 정의하고 있는데, 정의 2는 계산 노드들이 j개의 I/O 노드들과 인접하고 있는 경우를 정의하고 있으며, 정의 3에서는 완전 임베딩이 존재하지 않을 경우를 위한 차선책을 정의하고 있다. Ramanathan과 Chalasani는 [12]에서 k-ary n-cube 를 위한 완전 j-인접 임베딩 및 준완전 j-인접 임베딩의 존재 조건을 아래와 같이 제시하고 증명하였다.

정리 1: 다음의 두 조건이 모두 만족될 경우 k 2인 k-ary n-cube에서 완전 j-인접 임베딩(perfect j-adjacency embedding)이 존재한다.

(i) $k = (2n + j)i,\ i > 0$, i는 정수

(ii) $2n = ji,\ i > 0$, i는 정수

예를 들어, 〈그림 3〉은 $k = 6$이고 $n = 2$인 경우에 해당된다. 여기에 정리 1를 적용하여 완전 j-인접 임베딩이 존재함을 증명하여 보자. 우선 첫번째 조건은 $k = 6$은 $n = 2$, $j = 2$, $i = 1$인 경우이므로 만족한다. 또한 두번째의 조건도 만족됨을 알 수 있다. 따라서 6 x 6 토러스는 완전 2-인접 임베딩을 갖는다.

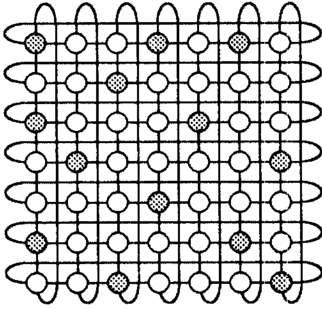
정리 2: $1 \le j \le n$, 또는 $j=(2n-1)$, 또는 $j=2n$인 경우에는 k 2인 k-ary n-cube 에서 준완전 j-인접 임베딩이 존재한다.

예를 들어, 7 x 7 토러스에 대한 준완전 2-인접 임베딩을 생각하여 보자. 정리 1의 조건이 만족되기 위하여는 j 혹은 i가 짝수이어야 하므로 홀수인 k에 대하여 완전 2-인접 임베딩은 존재하지 않는다. 따라서 7 x 7 토러스는 완전 2-인접 임베딩은 존재하지 않는다. 그러나 정리 2를 적용하여 보면 $1 \le 2 \le 2$인 경우이므로 〈그림 4〉와 같이 준완전 2-인접 임베딩이 존재할 수 있다.

여기에서 정의 1의 완전 임베딩은 완전 j-인접 임베딩의 특별한 경우인 완전 1-인접 임베딩인 것을 알 수 있다. $j > 1$인 경우에 완전 j-인접 임베딩은 완전 임베딩에 비해 많은 I/O 노드를 가지므로 성능 향상에는 도움이 되지만 그만큼 효율이 떨어지는 단점이 있다. 따라서 계산 노드가 I/O 노드를 반드시 인접할 필요가 없는 경우인 완전 r-임베딩에 대하여 Bose et al.[3]은 다음과 같이 정의했다.

정의 4: 비자원 노드는 거리 r 혹은 그 이내에 자원 노드를 가지며, 자원 노드들은 서로간에 적어도 $2r+1$ 이상의 거리에 존재할 때 r-임베딩이라 한다. 또한, 그 중에서 비자원 노드들이 정확하게 한 개만의 자원 노드와 r 혹은 그 이내의 거리를 가지고 인접할 때 완전 r-임베딩이라고 한다.

여기에서 자원이란 I/O나 공유 데이타 혹은 공유 코드를 의미한다. 그러나 본 연구의 목적은 I/O 노드의



〈그림 4〉 7-ary 2-cube에서 준완전 2-인접 임베딩

배치 방법이므로 위의 정리 2에서 자원을 가지고 있는 노드는 I/O 노드이고 자원을 가지지 않는 노드는 계산 노드이다(그림 1 참조). 물론 이러한 임베딩 방법들은 일반적인 자원의 배치 방법에도 적용될 수 있을 것이다.

2.2 토러스 구조를 위한 최적 I/O 임베딩

이 절에서는 토러스 구조에 완전 r-임베딩을 적용하고, 본 연구에서 사용된 크기의 토러스 구조에 대한 I/O 임베딩 결과를 살펴 보고자 한다. 먼저 n x n 토러스 구조에서 완전 r-임베딩의 특징을 나열하면 다음과 같다.

• 완전 r-임베딩 조건 : $n=(2r^2 + 2^r + 1) \times i$, ($i$는 임의의 정수)

• I/O 노드의 수 : $(2r^2 + 2^r + 1) \times i^2$

• 계산 노드 수에 대한 I/O 노드 수의 비율 : $\frac{1}{2r^2 + 2r + 1}$

• 하나의 I/O 노드로부터 거리 r에 존재하는 계산 노드의 수 : $4r$

• 하나의 I/O 노드로부터 거리 r 내에 존재하는 계산 노드의 수 : $2r^2 + 2r$

완전 r-임베딩은 모든 시스템 크기들에 대하여 존재하는 것이 아니기 때문에 본 연구에서는 모든 시스템 크기에 대하여 최적의 I/O 임베딩을 구하기 위해서 다음과 같은 알고리즘을 제안한다. 모든 크기의 시스템들에 대하여 일단 완전 I/O 임베딩을 위한 I/O 노드 배치를 찾아내고, 그 조건이 만족되지 않는 시스템 크기에 대하여는 다음과 같은 정책을 사용하여 배치를 결정한다.

• I/O 노드를 추가하지 않는다.

• r-임베딩 방식을 적용한다.

본 논문에서는 이 알고리즘을 최적 거리-k I/O 임베딩(optimal distance-k I/O embedding)이라고 부르기로 하며, 이 알고리즘은 〈그림 5〉와 같다. 이 알고리즘의 입력은 계산 노드와 I/O 노드 사이의 최대 거리인 k와 시스템의 크기를 나타내는 n, 그리고 완전 I/O 임베딩이 존재하지 않는 경우에 I/O 노드를 추가하는지의 여부(optimal_add / optimal_no) 등이다.

〈그림 6〉은 완전 I/O 임베딩이 존재하는 10 x 10 크기의 토러스 구조에서의 완전 I/O 노드 배치를 보여주고 있다.

완전 I/O 임베딩을 적용할 수 없는 크기에 대해서 앞에서 언급한 알고리즘을 이용하여 I/O 노드들을 거리(distance) = 1 로 최적 배치한 결과는 〈그림 7〉과 같다.

〈그림 7〉에서 ○는 계산 노드를 나타내며, ●는 I/O 노드를 나타낸다. ⊘는 optimal_add의 경우에 추가되는 I/O 노드를 나타낸다. 이와 같이 I/O 노드를 추가하는 경우에는 변(side)에 I/O 노드들이 추가되어 각 계산 노드들 중에 두 개 또는 그 이상의 I/O 노드와 인접하는 계산 노드들이 존재하게 된다. 본 연구에서는 변에 I/O 노드들을 추가하는 경우와 그렇지 않은 경우에 대한 성능을 모두 측정하고 비교하여, 제안한 알고리즘이 최적임을 보이고자 한다.

3. 소프트웨어 시뮬레이션

3.1 개요

본 장에서는 이 연구에서 사용한 소프트웨어 시뮬레이터에 대하여 서술한다. 다중프로세서 시스템은 다양한 프로세서의 동작, 수많은 프로세서간의 상호작용과 구조의 유연성 등의 이유로 분석적 모델을 이용한 성능분석이 아주 어렵다. 그리고 실제 시스템을 구성하는 것은 비용, 시간, 유연성 등의 문제가 있으므로 본 연구에서는 소프트웨어 시뮬레이션 기법을 사용하여 2장에서 제시한 I/O 노드의 배치 방법에 따른 시스템의

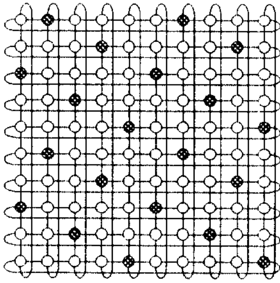
```
-Input
       k : distance
       n : √N  N= Number of nodes in the system
       method : optimal_add or optimal_no
              (perfect : either optimal_add or optimal_no)

-Procedure
 i := ⌈n/(2k^2 + 2k + 1)⌉;
 Ideal := (2k + 2k + 1) × i;
 curr_row := 0;
 curr_col := k;
 for j := 1 to n*i do
   for l := 0 to i-1 do
     row[curr] := curr_row;
     col[curr] := curr_col + (2k + 2k + 1) × l
     curr := curr + 1;
   curr_row = (curr_row + k) mod n;
   curr_col = (curr_col + k + 1) mod n;
 for l := 1 to n*i do
   if (method = optimal_no) then
     if (row[l] ≤ n AND col[l] ≤ n) then
       allocate (row[l], col[l]) node
   else if (method = optimal_add) then
     curr_row := row[l];
     curr_col := col[l];
     if (col[l] > n) then
       if (row[l] ≤ n) then
         if no iop within distance k from (row[l],n) then
           curr_col := n
         else skip allocation;
       if (row[l] > n ) then
         if no iop within distance k from (n, curr_col) then
           curr_row := n
         else skip allocation;
     allocate (curr_row, curr_col) node;
     row[l] := (row[l] + k) mod Ideal;
     col[l] := (col[l] + k + 1) mod Ideal;
```

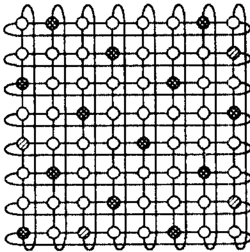
〈그림 5〉 최적 거리-k I/O임베딩을 위한 알고리즘

성능을 분석하였다. 그리고, 시뮬레이터를 새로 작성하여 사용하는 것은 비용이 많이 들고 시간이 많이 걸리며 시뮬레이터 자체를 검증해야 하는 오버헤드가 있으므로 본 연구에서는 그동안 다중프로세서 시스템의 시뮬레이터로 공개되고 검증된 시뮬레이터 중의 하나인 PROTEUS[5]를 수정하여 시뮬레이터로 사용하였다.

PROTEUS는 MIMD 다중프로세서 시스템의 개발 및 분석, 병렬 알고리즘의 개발 등을 위해 MIT 대학에서 개발된 실행-구동 시뮬레이터(execution-driven simulator)이다. PROTEUS는 각 프로세서의 명령어와 버스나 네트워크에 대한 액세스, 인터럽트 요구(interrupt request) 등을 사건 단위로 시뮬레이트한다. 그러나 사용자는 확장된 C 언어와 제공되는 시뮬레이터 콜(simulator call)을 사용하여 PROTEUS를 위한 병렬 프로그램을 작성



〈그림 6〉 10 x 10 토러스 구조에서의 완전 I/O 임베딩



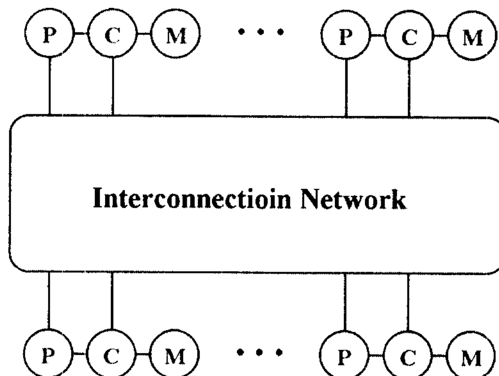
〈그림 7〉 8 x 8 토러스 구조를 위한 최적 I/O 임베딩

할 수 있다. 사용자 프로그램 중 다른 프로세서에의 개입이 필요하지 않은 부분은 표준 C 로 작성되고 C 컴파일러에 의해 호스트 컴퓨터의 명령어로 변환된다. 다른 프로세서 및 상호연결망에의 개입이 필요한 부분은 제공되는 시뮬레이터 콜에 의해 처리되고, 이것은 실제 다중프로세서 시스템의 비지역 명령어(nonlocal instruction)에 대응된다. 〈그림 8〉은 PROTEUS가 제공하는 기본적인 구조의 모델이다.

3.2 시뮬레이션 환경

본 절에서는 앞 절에서 설명한 시뮬레이터를 본 연구에 사용한 환경과 연구 목적에 알맞게 수정한 부분에 관하여 기술한다.

앞 절에서 설명한 시뮬레이터를 본 연구에 사용하기 위하여 시스템 구조는 분산 메모리를 가지며 프로세서들간의 네트워크는 직접 네트워크이며, 토폴로지는 토러스(torus)로 설정하였다. 시스템 클럭은 50 MHz로 설정하였으며, 라우팅 과정에서 통신 링크와 라우팅 칩에서 지연되는 시간은 각각 한 사이클 즉 20ns로 설정하였다. 따라서 한 통신 링크 당 최대 대역폭은 50MB/s이 된다. I/O 요구의 크기는 4K 및 1K 바이트로 정하였다. 본 연구는 I/O 노드들의 배치에 따른 I/O 통신 성능을 분석하는 것이므로 네트워크상의 통신량을 극대화시키기 위하여 디스크 드라이브의 동작은 시뮬레이션에 포함시키지 않았으며 이것은 모든 I/O 요구들이 디스크 캐쉬에 적중되는 경우에 해당한다. 그리고 시스템 네트워크는 I/O 뿐만 아니라 계산에 필요한 프로세서간 통신에도 사용이 되므로 본 연구에서는 I/O 요구(I/O request)와 프로세서간 요구(inter-PE request)를 모두 고려하였으며, 프로세서간 요구의 크기는 32바이



〈그림 8〉 PROTEUS의 기본 구조 모델

트임을 가정하였다. 시뮬레이션에 사용된 시스템 파라메타를 정리하면 〈표 1〉과 같다.

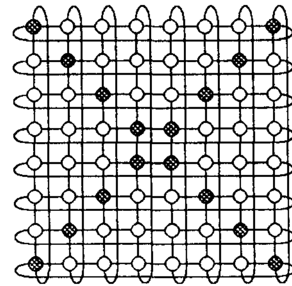
〈표 1〉 시스템 파라메타

parameter	Value
Number of processors	64, 100, 256
Network topology	n x n torus
Peak bandwidth	50MB/s per comm. link
Communication delay	20ns/link, 20ns/switch
Routing method	worm-hole, X-Y routing
Inter-PE request size	32byte
I/O request size	1024byte, 4096byte

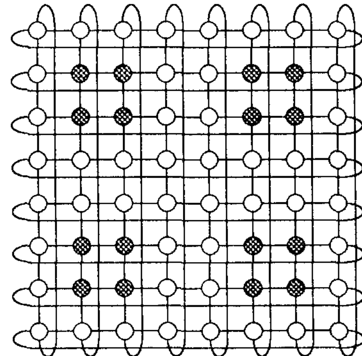
본 연구는 실제 시스템과 유사한 환경에서 I/O 노드의 배치 방법에 따른 성능을 분석할 필요가 있기 때문에 PROTEUS에서 이상적인 경우로 가정하고 작성된 부분들을 실제 시스템과 유사하게 수정할 필요가 있다. 특히 본 연구에서는 시스템 네트워크의 성능이 주된 관심 분야이기 때문에 이 부분에 많은 수정이 이루어졌다. PROTEUS의 라우팅에서 경로 설정은 X-Y 라우팅을 사용하고, 전송 제어로는 웜홀 라우팅을 사용하는데 이상적인 경우를 위하여 라우팅 칩이 무한대의 버퍼를 가지는 것으로 가정하였다. 또한 본 실험에서는 한 차원 내에서 교착 상태을 방지해주는 가상-채널 라우팅(virtual-channel routing)[6]을 사용하도록 시뮬레이터 엔진을 수정하였다. 가상-채널 라우팅에서 여러 개의 가상 채널들은 하나의 물리 채널을 공유하면서 시분할(timesharing) 방식으로 데이타 전송이 이루어지고, 교착 상태의 필요충분 조건인 사이클을 없애주는 역할을 한다. 본 실험에서 가상 채널의 수는 2로 설정하였다.

4. 시뮬레이션 및 결과 분석

이 장에서는 제3장에서 설명한 시뮬레이터를 이용하여 토러스 구조에 최적 I/O 임베딩과 완전 I/O 임베딩 방식을 각각 적용하였을 때의 성능을 측정하고 분석한 결과를 기술하였다. 성능 척도로는 네트워크 지연 시간(network latency time)을 사용하였다. 네트워크 지연 시간이란 하나의 패킷(packet)을 보내기 시작하여 그 패킷이 목적지에 도착할 때까지의 시간이다. 고려 대상으로 한 시스템 크기들은 8 x 8, 10 x 10, 16 x 16 등이다. 10 x 10 에서는 완전 I/O 임베딩이 존재하지만, 8 x 8 과 16 x 16 에서는 완전 I/O 임베딩이 존재하지 않기 때문에 최적 I/O 임베딩을 적용하였다. 최적 혹은 완전 I/O 임베딩과의 비교 대상으로 본 연구에서 고려한 I/O 배치 방법들은 〈그림 9〉와 같이 노드간 평균 거리를 최소화하면서 대칭성을 갖는 cross 형태의 배치 방법과 Lee와 Chen[11]이 hot-spot을 가지는 모델로 제시했던 submesh 배치 〈그림 10〉이다.



〈그림 9〉 8 x 8 토러스 구조에서 cross 형태의 I/O 노드 배치



〈그림 10〉 8 x 8 토러스 구조에서 submesh 형태의 I/O 노드 배치

시스템 내에 I/O 노드가 많이 존재할 수록 계산 노드와 I/O 노드 간의 평균 거리가 짧아져서 성능은 그만큼 향상될 것이지만, 하드웨어 비용이 높아지고 이용율은 떨어지게 된다. 따라서 최소의 I/O 노드들을 이용하여 최고의 성능을 얻을 최적의 배치 방법을 찾는 것이 중요하다. 〈표 2〉는 각 I/O 노드 배치 방법에 따

른 I/O 노드 수를 보여주고 있다.

〈표 2〉 I/O 노드 배치 방법에 따른 I/O 노드의 수

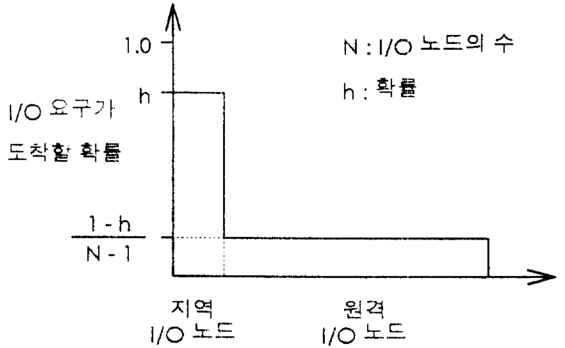
	submesh	cross	optimal_no	optimal_add	perfect
8 x 8	16	16	13	18	N/A
10 x 10	20	20	N/A	N/A	20

〈표 2〉에서 optimal_no 란 제2장에서 설명한 알고리즘에서 자원 노드와의 거리가 1이 넘는 경우에 I/O 노드를 추가하지 않는 경우를 의미하고, optimal_add는 I/O 노드를 추가하는 경우를 나타낸다. 또한 perfect는 완전 I/O 임베딩을 나타낸다.

시뮬레이션의 작업부하는 시스템 내의 각 계산 노드가 I/O 요구들을 발생하는 시간 간격과 목표 I/O 노드들의 번호에 대한 분포 함수에 근거하여 난수를 발생시켜 사용하였다. 여기서 목표 I/O 노드 번호란 계산 노드가 필요로 하는 데이타를 가지고 있는 I/O 노드를 말한다. I/O 요구간 간격은 지수 분포임을 가정했다 [8,14]. 목표 I/O 노드들의 분포는 위치에 따른 지역성이 존재하는 것으로 가정하였다. 즉, 계산 노드는 가장 가까운 I/O 노드로 더 많은 요구들을 보내며, 이러한 비율을 본 연구에서는 지역성의 정도(locality factor)라고 부른다. 예를 들어 지역성의 정도가 0.4라면, 임의의 계산 노드 i가 가장 인접한 I/O 노드 j로 요구를 보내게 되는 확률이 40 %이고, 나머지 60%는 그 외 다른 I/O 노드들로 균등한 비율로 요구들을 보낸다는 것을 의미한다. 지역성에 대한 액세스 분포는 〈그림 11〉과 같다. 여기에서 지역 I/O 노드(local I/O node)란 해당되는 계산 노드로부터 가장 가까이 있는 I/O 노드를 나타내고, 원격 I/O 노드(remote I/O node)는 그외의 I/O 노드들이다. 그리고 I/O 임베딩 방식이 프로세서간 통신에 미치는 영향도 분석하였는데, 프로세서간 통신은 모두 균등 액세스 분포를 가지는 것을 가정하였으며 프로세서간 요구의 크기는 32바이트로 가정하였다.

4.1 I/O 요구들만 존재하는 경우

이 절에서는 시스템 네트워크에서 프로세서간 통신과 I/O 통신이 서로 다른 시간에 발생함으로써 어느 한



〈그림 11〉 지역성에 의한 액세스 비율 분포

순간에 I/O 요구(I/O request)들만 존재하는 경우에 대하여 실험한 결과를 기술한다. 데이타를 I/O 노드에 분산 저장하는 방식에 따라 계산 노드와 I/O 노드 간의 거리가 달라질 수 있는데, 본 실험에서는 가장 가까운 I/O 노드에 필요한 데이타가 있을 확률을 0.5로 가정하였으며, 각 I/O 요구가 포함하는 데이타의 크기가 1 Kbyte와 4 Kbyte일 때의 성능을 각각 분석하였다.

4.1.1 I/O 요구의 크기가 1 Kbyte인 경우

〈그림 12〉는 8 x 8 토러스 구조에 대해 각 프로세서가 발생하는 요구의 요구간 시간을 10sec에서 270sec까지 변화시켰을 때의 네트워크 지연 시간을 측정한 결과를 보여주고 있다. 결과를 보면, optimal_add의 지연 시간이 가장 짧고 optimal_no, cross, submesh 순으로 길어지며, 포화점도 더 낮은 것으로 나타나 있다. 통신량이 적을 때에는 배치 방식들의 성능 간에 차이가 별로 나지 않지만 통신량이 많아질 수록 차이가 커지는 것을 알 수 있다. 특히 요구간 간격이 80sec인 경우에는 optimal_add가 cross나 submesh에 비해 10sec이상 네트워크 지연 시간이 짧은 것을 알 수 있다. optimal_add 의 경우는 I/O 노드의 수가 18개로서, cross나 submesh가 16개인 데 비하여 12.5% 가량 더 많지만 성능은 25%가량 더 높게 나타나므로 최적 I/O 임베딩 방식이 가격대 성능비 면에서도 더 우수하다는 것을 알 수 있다. 요구간 간격이 줄어들 수록 그 차이는 더 커지는 것을 알 수 있다. Optimal_no의 경우는 cross나 submesh에 비하여 I/O 노드의 수가 20%가량 적음에도

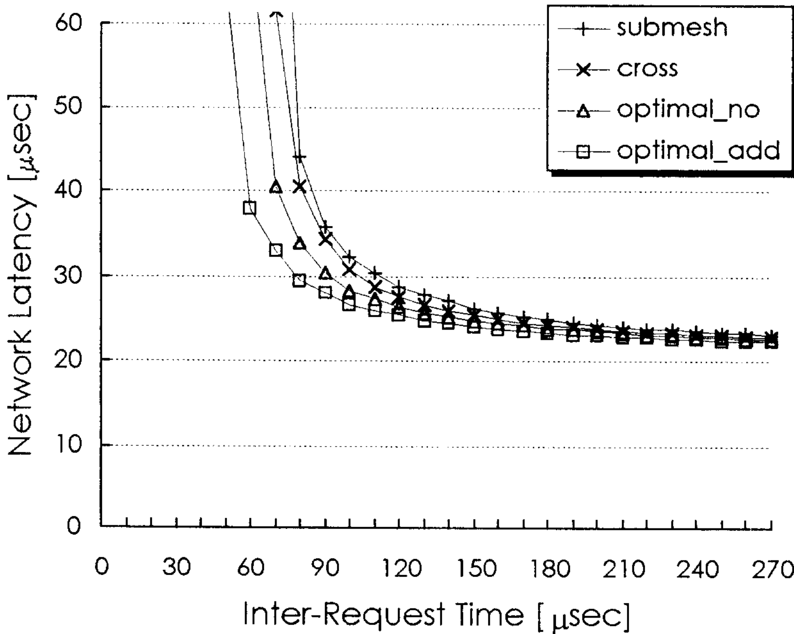
불구하고 항상 더 높은 성능을 보이고 있다. 전체적으로 보면, 최적 I/O 임베딩을 적용한 배치 방식들이 시스템의 성능 향상 및 가격대 성능비 개선에 도움을 많이 준다는 것을 알 수 있다.

〈그림 13〉은 10 x 10 토러스 구조에 완전 I/O 임베딩과 cross 및 submesh의 성능을 비교한 결과이다. 〈그림 13〉의 결과에 따르면, 완전 I/O 임베딩이 다른 두 가지 배치 방식들에 비해 성능이 월등히 우세한 것으로 나타났다. 예를 들어, 요구간 간격이 110sec인 경우에는 완전 I/O 임베딩이 submesh보다 25%이상 높은 성능을 보였으며, 이는 요구간 간격이 짧아질 수록 성능 차이가 더 커지는 것을 알 수 있다. 10 x 10 토러스 구조에서는 완전 I/O 임베딩이 존재하므로 모든 계산 노드는 반드시 하나의 I/O 노드를 인접하게 된다. 따라서 〈그림 13〉의 최적 I/O 임베딩의 경우보다 더 높은 성능을 보이는 것이다. 그리고 〈그림 13〉에서 submesh 배치가 cross 배치보다 더 좋은 성능을 보이는 데 반하여 〈그림 12〉에서는 cross 배치가 submesh 배치보다 나은 성능을 보이는 것은 10 x 10에서는 submesh가 cross 배치에 비해 계산 노드와 I/O 노드 간의 거리가 더 짧아지기 때문이다.

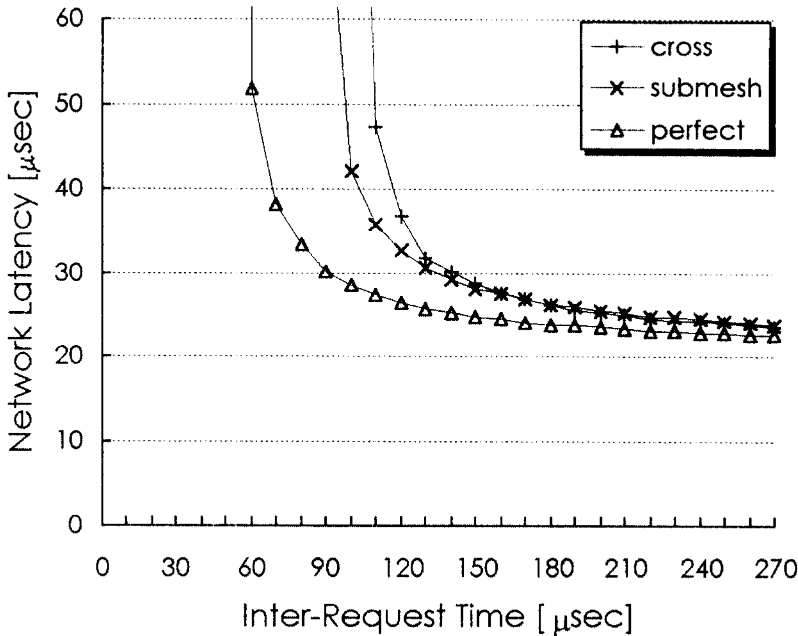
4.1.2 I/O 요구의 크기가 4Kbyte인 경우

이 절에서는 시스템 네트워크에 I/O 요구만 존재하고 I/O 요구의 크기가 4096바이트인 경우에 대하여 실험한 결과를 기술한다. 4.1절에서의 경우에 비해 I/O 요구의 크기가 4배이기 때문에 같은 요구간 시간 간격에 대해 통신량도 4배 커지게 된다. 실험 결과는 4.1절과 비슷하며, 전반적으로 4.1절의 경우보다 빨리 포화되는 특성을 가지는데, 그 이유는 통신량이 그만큼 더 많기 때문이다. 요구간 시간 간격은 0.1msec부터 2.0msec까지 변화시키면서 실험을 진행하였다.

〈그림 14〉는 8 x 8 토러스 구조를 대상으로 실험한 결과를 보여주고 있으며, 결과를 보면 optimal_add, optimal_no, cross, submesh 순으로 성능이 높게 나타났다. 이 순서는 4.1절의 〈그림 12〉의 결과와 동일하다.



〈그림 12〉 8 x 8 토러스에서 I/O 요구간 간격에 따른 네트워크 지연
(I/O only, I/O request size = 1024 byte)



〈그림 13〉 10 x 10 토러스에서 I/O 요구간 간격에 따른 네트워크 지연
(I/O only, I/O request size = 1024 byte)

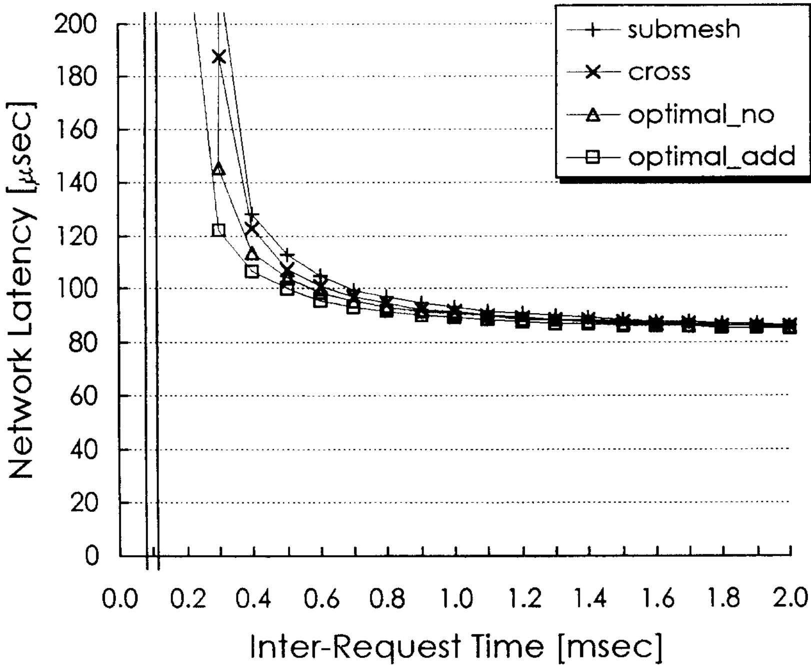
〈그림 12〉와 비교해 볼 때 각 배치 방법에 따른 차이의 절대값이 더 큰 것을 알 수 있는데, 이는 한 요구의 크기가 증가했으므로 전체적으로 통신시간이 기어졌기 때문이다. 그외의 특성은 〈그림 12〉의 결과와 동일하다. 따라서 I/O 노드 배치에 따른 성능이 I/O 요구의 크기와는 무관하다는 것을 알 수 있다.

〈그림 15〉는 10 x 10 토러스 구조에서 완전 I/O 임베딩을 cross, submesh 배치와 비교한 결과이다. 〈그림 15〉의 결과를 보면 완전 I/O 임베딩이 다른 두가지 배치 방식들에 비해 성능이 월등히 높게 나타나 있으며, 특히 요구간 시간이 0.4msec인 경우에는 submesh 배치 방식보다 완전 I/O 임배딩의 지연 시간이 32% 더 짧게 나타났다. 그외에, 통신량이 적을 때에는 cross 배치 방식이 submesh 배치 방식에 비해 지연 시간이 더 짧지만 통신량이 많을 때에는 오히려 cross 배치 방식이 submesh 배치 방식보다 성능이 더 낮고 더 빨리 포화되는 것을 알 수 있다.

지금까지의 결과를 정리하면, 완전 및 최적 I/O 임베딩이 모든 경우에 있어서 다른 I/O 노드 배치 방식에 비하여 더 우수한 성능을 보였으며, 성능이 I/O 요구의 크기와는 무관하다는 것을 알 수 있었다.

4.2 프로세서간 요구와 I/O 요구가 동시에 존재하는 경우

이 절에서는 I/O 요구 뿐만 아니라 프로세서간 요구도 시스템 내에 동시에 존재하는 경우에 대하여 실험한 결과를 기술한다. 시스템 네트워크에 I/O 요구와 프로세서간 요구가 동시에 존재하는 것은, 가상 메모리를 가지며 시분할 방식으로 여러 프로세스들을 처리하는 최근의 다중프로세서 시스템에서의 일반적인 작업부하 특성에 해당된다[7]. 본 실험에서 사용한 I/O 요구의 크기는 1024바이트이며, 프로세서간 요구의 크기는 32바이트이다. I/O 요구의 지역성은 0.5이며 프로세서 요구는 균등 분포 액세스(uniform distribution access)를 가지는 것으로 가정했다. 그리고 프로세서간 요구는 전체 요구의 90%이고 I/O 요구는 10%인 것으로 가정했다[8]. 실험은 요구간 간격을 20sec에서 40sec까지



〈그림 14〉 8 x 8 토러스에서 I/O 요구간 간격에 따른 네트워크 지연
(I/O only, I/O request size = 4096 byte)

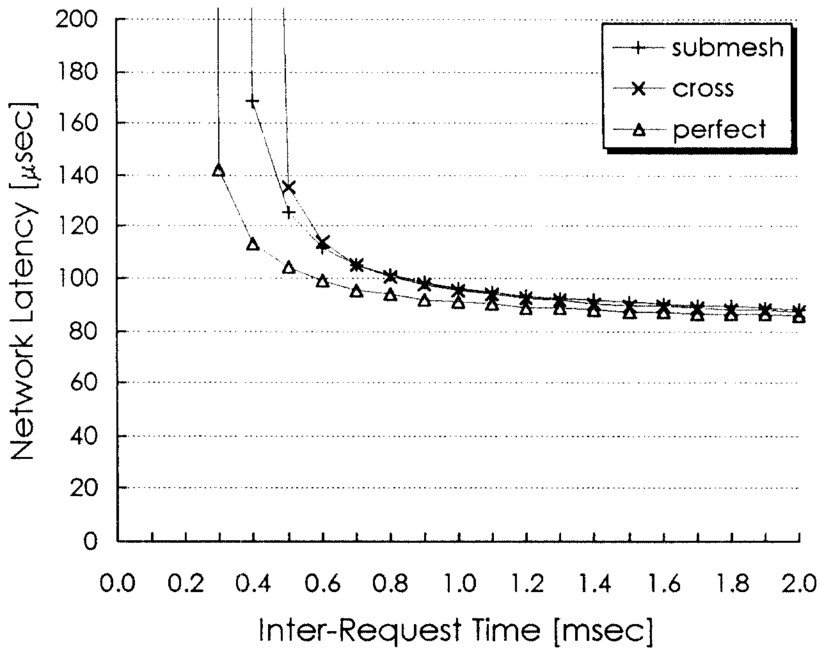
변화시키면서 수행하였다. 시스템의 크기는 16 x 16 으로 설정하였다.

〈그림 16〉은 프로세서간 요구가 존재할 때 요구간 시간의 변화에 따른 지연 시간을 측정한 결과를 보여주고 있다. 결과에 따르면, 프로세서간 요구가 존재할 때에도 완전 I/O 임베딩이 다른 배치 방법에 비하여 더 우수한 성능을 나타내는 것을 알 수 있다. 이 그림에서 cross와 optimal_add가 비슷한 성능을 나타내고 있지만 4.1절에서 기술한 바와 같이 optimal_add가 I/O 노드 수가 10% 가량 적으므로 그만큼 성능이 더 높다는 것을 알 수 있다. optimal_no의 경우cross보다 성능이 낮게 나타나는데, 그 이유는 cross 보다 I/O 노드의 수가 20% 가량 더 적기 때문이다. 그리고 프로세서간 요구가 존재할 때에는 배치 방법에 따른 차이가 더 커지는 것을 결과를 통해 확인할 수 있다.

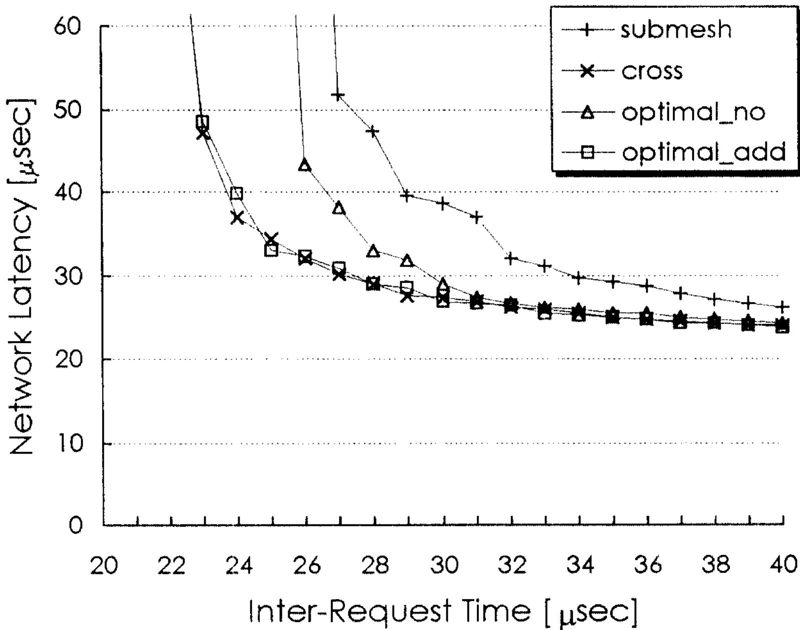
〈그림 17〉은 프로세서간 요구와 I/O 요구가 모두 존재할 때 요구간 시간의 변화에 따른 프로세서간 요구의 지연 시간을 보여주고 있다. 결과에 따르면, 최적 I/O 임베딩의 경우에 프로세서간 통신 지연도 가장 짧은 것을 알 수 있으며, 요구간 간격이 감소함에 따라 프로세서간 통신 지연 시간은 비교적 서서히 증가한다는 것을 알 수 있다. 이 결과로부터, 프로세서간 요구가 I/O 요구의 지연 시간에 미치는 영향보다는 I/O 요구가 프로세서간 요구의 지연 시간에 미치는 영향이 더 크다고 말할 수 있다. 그 이유는 I/O 요구의 발생율이 전체 요구의 10%라 하더라도 I/O 요구의 크기는 프로세서간 요구의 32배이므로 I/O 요구가 전체 통신량에서 차지하는 비율은 78%나 되기 때문이다.

5. 결 론

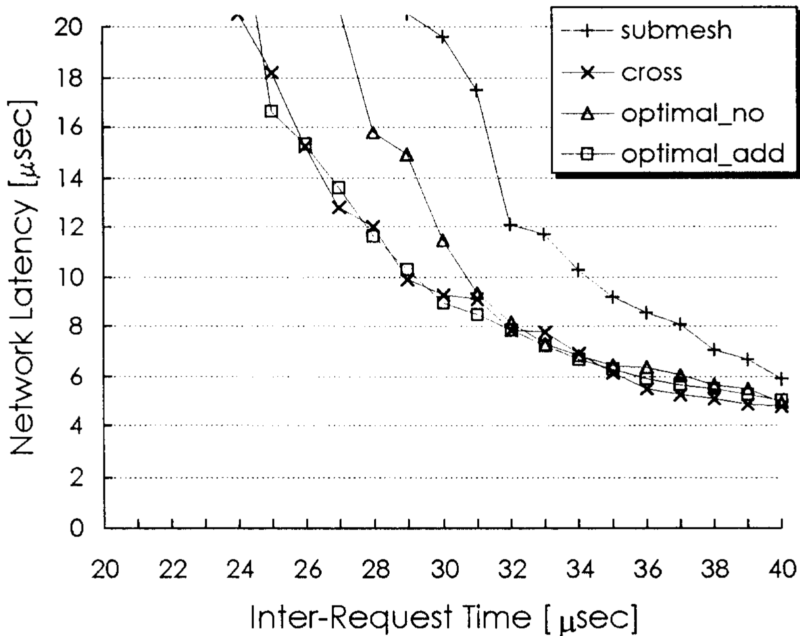
본 논문에서는 토러스 구조의 다중프로세서 시스템을 위한 I/O 노드 배치 방법을 제시하고, 그 방법이 시스템에 미치는 영향을 실험적으로 분석하여 다른 배치 방법들 보다 우수한 성능을 보인다는 것을 입증하였다. 시스템 모델로는 계산 노드와 I/O 노드를 가지며 가상-채널 웜홀 라우팅을 사용하는 n x n 토러스 구조로 설정하였다. I/O 요구의 크기는 1024 바이트와 4096



〈그림 15〉 10 x 10 토러스에서 I/O 요구간 간격에 따른 네트워크 지연
(I/O only, I/O request size = 4096 byte)



〈그림 16〉 16 x 16 토러스에서 요구간 간격의 변화에 따른I/O 요구의 지연 시간
(IPC & I/O, I/O request size = 1024 byte)



〈그림 17〉 16 x 16 토러스에서 요구간 간격에 따른프로세서간 요구의 지연 시간 (IPC & I/O, I/O request size = 1024 byte)

바이트의 경우를 고려하였으며, 프로세서간 요구도 존재하는 것으로 가정하였다. 실험 결과에 따르면, 모든 경우에 있어서 완전 I/O 임베딩 조건을 적용한 배치 방법이 가장 높은 성능을 보였으며, 최적 I/O 임베딩 방식이 cross 배치와 submesh 배치 보다 더 좋은 성능을 보이는 것을 알 수 있었다. 그 이유는 I/O 노드가 시스템 전체에 고르게 분포됨으로 해서 시스템 네트워크 상에서의 hot-spot을 감소시켰기 때문이다. 또한 16 x 16의 경우에는 submesh 배치를 제외하고는 모두 비슷한 성능을 보였지만 완전 I/O 임베딩 조건을 적용하고 I/O 노드를 추가하지 않은 optimal_no 방식이 I/O 노드 수가 가장 적으므로 최적의 I/O 배치 방식이라고 할 수 있다. 따라서 시스템의 크기가 커질 수록 I/O 노드를 추가하지 않는 경우가 최적의 I/O 노드 배치 방법이라고 할 수 있다. I/O 요구의 비율을 변화시킨 실험 결과에 따르면, I/O 요구의 발생 비율을 증가시킬 수록 최적 I/O 임베딩과 다른 배치 방법들과의 성능 차이가 더 커졌다. 따라서 I/O 비율이 높을 수록 최적 I/O 임베딩의 효과가 더 커진다고 말할 수 있다.

참고문헌

[1] A. Agarwal et al., "The MIT alewife machine: a large-scale distributed-memory multiprocessor," Scalable Shared-Memory Multiprocessors, Kluwer Academic Publishers, 1991.

[2] E. R. Berlekamp, Algebraic coding theory, New York: McGraw-Hill, 1968.

[3] B. Bose and et al, "Lee distance and topological properties of k-ary n-cubes," IEEE Transactions on Computers, Vol. 44, No. 8, pp. 1021-1030, August 1995.

[4] E. A. Brewer and C. N. Dellarocas, PROTEUS User Documentation, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, 0.5 edition, December 1992.

[5] S. Cannon, "Concurrent file system - Making highly parallel mass storage transparent," Proceedings. of Supercomputing '89, St. Petersberg, FL, May 1989.

[6] W. J. Dally and C. L. Seitz, "Deadlock-free message

routing in multiprocessor interconnection networks," IEEE Transactions on Computers, Vol. C-36, No. 5, May 1987.

[7] J. Ghosh and B. Agarwal, "Parallel I/O subsystems for hypercube multicomputers," Proceedings of the Fifth International Parallel Processing Symposium, pp. 381-384, May 1991.

[8] J. Ghosh and et al., "Performance evaluation of a parallel I/O subsystem for hypercube multicomputers," Journal of Parallel and Distributed Computing 17, pp. 90-106, 1993.

[9] K. Hwang, Advanced Computer Architecture, New York, NY: McGraw Hill, Inc., 1993.

[10] D. Jensen, "Disk I/O in high-performance computing systems," Ph.D Dissertation, University of Illinois, Urbana, Illinois, U.S.A., 1993.

[11] S. Y. Lee and C. M. Chen, "Optimal hot spot allocation on meshes for large-scale data-parallel algorithms," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 8, pp. 788-802, August 1995.

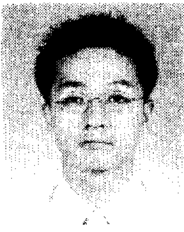
[12] P. Ramanathan and S. Chalasani, "Resource placement in k-ary n-cubes," '89 International Conference on Parallel Processing, pp. II-133 - II-140, 1992.

[13] A. L. N. Reddy, P. Banerjee, and S. G. Abraham, "I/O embeddings in hypercubes, " Proc. 1988 Int. Conference on Parallel Processing, pp. 318-338, August 1988.

[14] A. L. N. Reddy and P. Banerjee, "Design, analysis, and simulation of I/O architectures for hypercube multiprocessors," IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 2, pp. 140-151, April 1990.

[15] D. Reed and R. Fujimoto, Multicomputer Networks, Cambridge, MA: MIT Press, 1985.

● 저자소개 ●



안중석

1994년 연세대학교 문리대학 전산학과 졸업 (이학사)

1996년 연세대학교 대학원 전산학과 졸업 (이학석사)

1996년~현재 한컴기술연구소 연구원

관심분야 컴퓨터구조, 컴퓨터시뮬레이션, 성능분석

강준효

1995년 연세대학교 문리대학 전산학과 졸업 (이학사)

1997년 연세대학교 대학원 전산학과 졸업 (이학석사)

관심분야 컴퓨터구조, 병렬알고리즘, 성능분석

김종현
1976년 연세대학교 공과대학 전기공학과 졸업 (공학사)
1981년 연세대학교 대학원 전기공학과 졸업 (공학석사)
1988년 Arizona State University 컴퓨터공학과 졸업 (Ph.D)
1976~1982년 국방과학연구소 연구원
1988~1990년 한국전자통신연구소 실장
1990년~현재 연세대학교 전산학과 교수
관심분야 컴퓨터구조, 컴퓨터시뮬레이션, 성능분석