프로세스의 수와 실행시간에 따른 NOW의 성능 분석

Performance Analysis of a NOW According to the Number of Processes and Execution Time

조수현
금오공과대학교 컴퓨터공학과 대학원
김영학
금오공과대학교 컴퓨터공학부 교수

Soo-Hyun Cho
Dept. of Computer Engineering, Kumoh National Institute of Technology
Young-Hak Kim
Professor, Dept. of Computer Engineering, Kumoh National Institute of Technology

중심어 : NOW, MPI, 브로드캐스팅(Broadcasting), 부하분할(Load Sharing), 작업 프로세스(Work Process)

요 약

최근에 고비용 수퍼 컴퓨터를 대신하여 네트워크 상에 연결된 저가의 PC와 워크스테이션으로 구성된 NOW (Network of Workstations) 시스템이 널리 활용되고 있다. NOW에서 병렬처리를 위한 성능은 각 컴퓨터의 계산능력과 통신시간에 의존한다. 현재 NOW에서 병렬처리 성능을 향상하기 위한 다양한 방법들이 제안되고 있다. 그러나 기존의 결과들은 각 컴퓨터의 계산능력에 따른 작업 부하의 균형 관점에서 연구를 수행하였다.

만일 NOW에서 한 컴퓨터가 다수의 작업 프로세스를 가질 경우 메시지 패싱에 필요한 통신시간의 감소를 예측할 수 있다. 따라서 본 논문에서는 작업 프로세스 관점에서 성능 개선 요인을 분석하고, 작업 프로세스 수를 증가할 때 전체 성능에 미치는 영향을 실험적으로 평가한다. 또한 본 논문의 실험에 사용될 새로운 브로드캐스팅 방법을 제안한다. 본 논문에서는 실험적 평가를 위해 LAM/MPI를 사용한다.

Abstract

Recently, instead of a high-cost supercomputer, there have been widely used a NOW system that consists of low-cost PCs and workstations connected all over the network. In a NOW, performance for parallel processing depends on the computation power of each computer and communication time. Currently, a lot of methods have been proposed in order to increase the performance of parallel processing. However, the previous results have been studied in the view of balancing work load as the computation power of each computer.

If a computer has multiple work processes in a NOW, we can predict a decrease of communication time needed in message passing. Therefore, in this paper, we analyzes factors of improving the performance in the view of work processes, and evaluates experimently an effect on total performance as the number of work processes increases. Also, we propose a new broadcasting method to be used in experiment of this paper. This paper uses the LAM/MPI for an experimental evaluation.

I. 서론

최근에 컴퓨터 하드웨어 및 통신기술이 발전하면서 수퍼 컴퓨터를 대신하여 네트워크 상의 컴퓨터들을 이용한 병렬처리 기술이 활발히 연구되고 있다. 이러한 형태의 병렬처리 기술은 NOW로 알려져 있으며 PC 및 워크스테이션 등과 같은 저가의 컴퓨터들을 네트워크에 연결하여 서로의 계산 능력을 공유한다.

현재 MPI(Message Passing Interface)[1, 2], PVM(Parallel Virtual Machine)[3, 4], MPJ[5]와 같은 라이브러리가 개발되어 NOW에서 효과적인 병렬처리를 위해 사용되고 있다. NOW 시스템은 서로 다른 성능을 갖는 다양한 컴퓨터들이 네트워크 환경에 구성되어 있기 때문에, 전체 수행시간을 향상시키기 위한 여러 가지 요소가 존재한다. 대표적인 요소들은 다음과 같다.

- o 워크스테이션 자원의 유휴시간(Idle time)을 줄인다.
- o 워크스테이션 부하를 분할(Load Sharing)한다.
- o 전송하는 데이터의 크기가 클 경우 워크스테이션의 대기 시간(Latency)이 길어지므로 알맞은 크기의 데이터를 결정한다.
- o 병렬처리를 위해 효율적으로 데이터를 전송한다.
- o 네트워크에 연결된 워크스테이션은 성능의 차이를 보일 수 있으므로 그에 맞는 부하 분할과 전송 데이터 크기를 고려해야 한다.

NOW 환경에 가장 적합한 병렬처리 방법은 SPMD(Single Program Multiple Data) 형식을 들 수 있다. SPMD는 단일 프로그램이 NOW 시스템에 구성된 다수의 컴퓨터에서 서로 다른 자료를 가지고 수행하는 방법을 의미한다. 여기서 각 컴퓨터에서 수행되는 프로그램을 다른 표현으로 작업 프로세스(Work Process)라 한다. 후에 본 논문에서는 작업 프로세스란 용어를 주로 사용한다.

NOW에서 효율적인 병렬처리를 위해 현재 다양한 방법들이 제안되고 있다. 그러나 대부분의 기존 연구들은 효율적인 병렬처리를 위해 NOW에 참여하는 컴퓨터들의 계산능력과 부하의 균형관점에서 연구가 진행되었다[6, 7, 8]. 또한 네트워크 상에 발생되는 통신 시간을 줄이기 위해 전송되는 자료의 패킷단위를 조정하는 방법 등이 대부분이다[9, 10, 11, 15].

본 논문의 연구배경은 NOW에 참여하는 한 컴퓨터에 다수의 작업 프로세스를 생성할 경우, 전체적인 성능에 어떤 영향을 주게 될 것인가의 의문으로부터 시작되었다. 먼저 하나의 컴퓨터에 다수의 작업 프로세스를 생성할 경우 NOW에서 같은 프로세스의 브로드캐스팅 시간과 메시지 교환에 필요한 시간을 절약할 수 있다. 이유는 하나의 컴퓨터에 다수의 작업 프로세스가 수행되기 때문에 네트워크를 통한 메시지 교환이 이루어지지 않고, Unix 혹은 Linux 같은 운영체제에서는 fork 시스템 호출에 의해 쉽게 같은 프로세스를 생성할 수 있다.

그러나 한 컴퓨터의 계산능력과 부하문제가 존재하기 때문에 적정량 이상의 작업 프로세스를 하나의 컴퓨터에 생성할 경우 오히려 전체 성능을 저하할 가능성이 있다. 따라서 본 논문에서는 NOW에서 하나의 컴퓨터에 다수 개의 작업 프로세스의 생성이 전체 성능에 어떤 영향을 주게 되는지를 실험적으로 평가한다.

이러한 문제를 해결하기 위해 본 논문에서는 NOW에서 전체 수행시간 단축을 위해 다음과 같은 3가지 관점을 고려한다. 첫째, NOW에 참여하는 각 컴퓨터의 작업 프로세스들에 빠르게 메시지를 전송할 수 있는 새로운 브로드캐스팅 방법을 제안한다. 둘째, 프로세스들의 부하를 고려한 작업량 배분 및 통신 대기시간 단축 방법 등을 분석한다. 마지막으로 하나의 컴퓨터에 다수의 작업 프로세스를 생성할 경우 전체 수행시간에 미치는 영향을 실험적으로 분석한다.

본 논문에서는 실험적 평가를 위해 기존 네트워크 상에 연결된 컴퓨터들을 이용하여 NOW 환경을 구성하였으며, 현재 NOW에서 널리 사용되는 LAM/MPI 라이브러리를 이용하여 평가 프로그램을 구현하였다. 또한 평가를 위한 도구로 병렬처리에 대한 벤치마킹으로 널리 이용되는 행렬곱셈 문제를 수식 모델로 사용하였다. 브로드캐스팅 대한 실험적 평가에서는 작업프로세스 수와 데이터의 크기에 따라 기존방법과 제안된 방법간의 비교평가를 수행하였다.

또한 작업 프로세스의 수와 수행시간에 따른 성능 평가를 위해 다음과 같이 세 가지 부하분할 방법을 사용하였다. 첫 번째와 두 번째 방법은 각각 작업을 균등하게 배분하는 방법과 부하를 고려하여 작업을 배분하는 방법을 사용하였다. 세 번째는 통신 대기 시간을 줄이기 위해 일정시간 간격에 따라 연속적으로 작업을 분배하는 방법을 사용하였다.

그 결과로 본 논문에서 제안한 브로드캐스팅과 부하분할 방법에서는 작업 프로세스 수와 전송 데이터의 크기에 따라 기존 방식들보다 약 3.2%에서 4.2배의 성능이 향상되었다. 또한 작업 배분과정에서 네트워크 통신시간은 전체 성능에 영향을 주는 주요한 요인중의 하나이기 때문에, 이럴 경우 작업 프로세스 수를 증가하기 보다 적절한 프로세스의 수와 브로드캐스팅, 부하분할방법 등을 함께 고려했을 때 더 좋은 성능향상을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 연구를 개괄적으로 검토하고, 3장에서는 NOW에서 작업 프로세스 생성과 이와 관련된 기본적인 용어 정의와 새로운 브로드캐스팅 알고리즘을 제안한다. 4장에서는 작업 프로세스 관점에서 실험에 참여하게 될 부하 분할 방법 등을 설명한다. 5장은 실험결과 및 분석에 대해서, 끝으로 6장에서 결론을 설명한다.

II. 관련 연구

MPI는 NOW에서 사용되는 표준화된 메시지 전달 함수의 라이브러리로, 동기 및 비동기 방식의 점대점 통신함수와

집단화 통신함수를 제공한다. 일반적으로 널리 사용되는 라이브러리는 PVM, MPICH, LAM/MPI[12]가 있다. 본 논문에서는 미국 노트르담 대학에서 개발한 LAM/MPI를 실험적 평가를 위한 도구로 이용한다.

LAM/MPI에서 MPI_Bcast 함수는 참여하는 작업 프로세스 수에 따라 점대점 통신을 사용한다. 이 함수는 각 컴퓨터들에게 순차적으로 메시지를 전달하는 마스터/슬레이브 방법과 바이노미얼 트리(Binomial Tree) 방식으로 구현되었다. 이러한 함수는 근거리를 기반으로 구현되어 있기 때문에 실제 원거리 전송에는 접합하지 않다.

최근에 T.Kielmann 등에 의한 MAGPIE[13]가 제안되었는데 이는 원거리 집단화 통신을 효율적으로 수행하기 위해 개발된 라이브러리이다. 이 함수는 NOW에 참여하는 컴퓨터들을 일정한 비율로 그룹을 만든 후에 각 그룹의 조정자(Coordinators)를 통해 통신이 이루어진다. 처음 단계에서 마스터 컴퓨터는 각 그룹의 조정자들에게 점대점 통신을 수행하고, 다음에 각 그룹내의 조정자를 통해 이진 트리 방식으로 메시지를 전송한다.

A. Piotrowski는 NOW에서 효과적인 부하 분할에 따라 전체 성능을 개선하는 방법을 제안하였다[14]. 이 연구에서는 부하를 고정 분할 단위(Fixed granularity), 가변 분할 단위(Variable granularity), 적응 분할 단위(Adaptive granularity)로 분류하였다. 이 연구에서는 성능 비교를 위해 고정 분할 단위 알고리즘은 Send 알고리즘, 가변 분할 단위 알고리즘은 GSS(Guided Self-Scheduling) 알고리즘, Factoring 알고리즘, TSS(Trapezo- idal Self-Scheduling) 알고리즘을 이용하였다[15, 16]. 적응 분할 알고리즘에서는 컴퓨터의 반응시간을 이용하여 부하를 동적으로 분할하는 방법이 사용되었다.

연구 결과를 요약하면 대체적으로 슬레이브 컴퓨터에게 고정 크기의 데이터를 전송하여 처리하는 Send 알고리즘[15]과 아직 처리가 되지 않고 남아 있는 데이터의 양에 따라, 슬레이브 컴퓨터에 전송할 데이터의 양을 결정하는 GSS 알고리즘[16]이 다른 부하 분할 알고리즘보다 좋은 성능을 보인 것으로 나타났다. 하지만 여전히 마스터 컴퓨터와 슬레이브 컴퓨터들간의 일정 대기 시간과 유휴시간이 생기는 문제를 가지고 있다.

III. 기본개요 및 새로운 브로드캐스팅

1. 기본 개요

NOW에서 하나의 작업을 다수의 컴퓨터에서 병렬수행 하기 위해 SPMD 방법이 가장 널리 사용되고 있다. SPMD 방법은 마스터 역할을 하는 컴퓨터가 자신이 소유한 하나의 프로그램을 병렬환경에 참여하는 모든 컴퓨터에 전송한다. 그러면 NOW에 참여하는 각 컴퓨터는 서로 다른 자료를 사용하여 같은 프로그램을 병렬로 수행하여 주어진 작업을 효율적으로 처리할 수 있다.

본 논문에서는 SPMD 방법의 병렬처리 환경을 고려하며 다음에서 이와 관련된 개요를 간략하게 기술한다. 그림 1은 6개의 작업 프로세스가 생성되어 동일한 프로그램 코드를 수행하는 예를 보여준다. 0, 1, 2, ... 5는 각 작업 프로세스들의 번호를 의미한다. 각 컴퓨터에 생성된 작업 프로세스들에 대한 메시지 교환은 기 설치된 LAM/MPI 데몬에 의해 이루어진다.

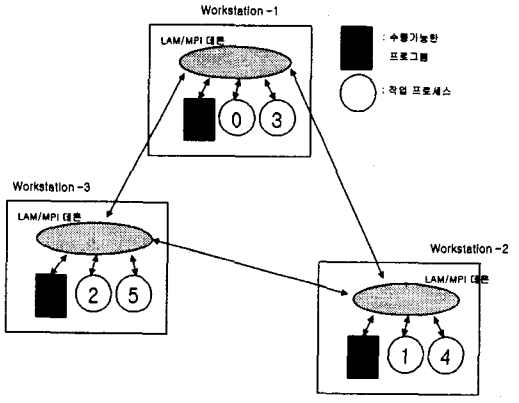

그림 1. 작업 프로세스의 생성 예

작업 프로세스의 고유번호는 NOW에 참여하는 컴퓨터의 수에 따라 0부터 시작하여 라운드 로빈 방식으로 생성된다. 예를 들어 사용자가 작업에 참여하는 프로세스 수의 값을 입력하면, 그림 1과 같이 workstation-1, workstation-2 방향으로 작업 프로세스들이 라운드 로빈에 의해 순차적으로 생성된다.

일반적으로 고유번호 0을 갖는 작업 프로세스가 마스터의 역할을 하게 되고, 나머지 번호를 가지는 작업 프로세스들은 슬레이브 프로세스가 된다. 마스터의 역할을 하는 작업 프로세스는 작업 분배 및 전반적인 조정역할을 수행하고, 나머지 작업 프로세스들에서 수행된 결과들을 취합하여 최종적인 결과 값을 계산하게 된다.

2. 새로운 브로드캐스팅 방법

기존 MPI에 구현된 브로드캐스팅은 점대점 통신을 기본으로 하는 마스터/슬레이브 방법과 이진트리를 기본으로 하여 전송속도를 개선하는 방법이 있다. 그러나 이러한 방법은 하나의 컴퓨터에 다수개의 작업 프로세스가 생성될 경우, 각 작업 프로세스에 매번 메시지를 전송하여 불필요한 통신시간을 소모하게 된다. 따라서 이 절에서는 하나의 컴퓨터에 다수개의 프로세스가 생성될 경우 좀 더 효율적으로 메시지를 브로드캐스팅하는 새로운 방법을 제안한다.

그림 2는 기존 MPI에 구현된 메시지 브로드캐스팅의 예를 보여준다. 여기서 전체 9개의 작업자 프로세스가 라운드 로빈 방식으로 생성되었고, 각 컴퓨터에는 3개의 작업 프로세스가 생성된 예를 보여준다. 그림에서 괄호 안의 번호는 마스터 프로세서가 나머지 프로세스에게 메시지를 점대점 형식으로 브로드캐스팅하는 예를 보여준다.

먼저 마스터/슬레이브 방법의 경우는 참여하는 컴퓨터 중에서 마스터 프로세스가 결정되고, 나머지 프로세스들은 슬레이브 프로세스(이하 슬레이브)들로 구성된다. 만일 각 컴퓨터의 성능과 부하를 무시한 정책이 사용된다면, 메시지 전송은 전체 작업량을 참여하는 슬레이브 수만큼 균등하게 분할되어 나머지 슬레이브들에게 순차적으로 전송한다. 각 슬레이브는 자신에 할당된 작업을 수행한 후 결과를 마스터에게 보내고 마스터는 각 결과 값들을 취합한다.

이 방법은 구현이 용이하고 단순한 전송방법이나 전송하는 시간과 수신하는 시간과의 차이가 큰 환경에서는 적합하지 못한 단점을 갖는다. 즉 메시지의 크기와 참여하는 컴퓨터의 수에 비례하여 전체 통신시간이 증가하게 된다.

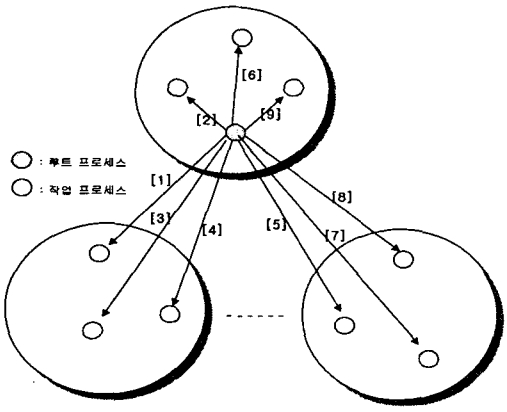

그림 2. 기존 브로드캐스팅 방법

트리 방법에서 마스터는 전송 메시지를 임의의 프로세스에게 보내고 현재 메시지를 받은 모든 프로세스가 그렇지 않은 프로세스에게 2배씩 확대하며 재귀적인 방법으로 메시지를 전달한다. 이 방법은 참여하는 컴퓨터 수의 로그(Log) 함수에 비례한 통신을 가지나, 하나의 컴퓨터에 다수의 작업 프로세스가 생성될 경우 중복 전달문제를 해결하지 못한다.

이러한 문제를 해결하기 위해 본 논문에서는 각 컴퓨터에 생성된 작업 프로세스 관점에서 브로드캐스팅 방법을 고려한다. 먼저 이러한 방법을 구현하기 위해 그림 3과 같이 각 컴퓨터는 자신이 생성한 작업 프로세스를 이진트리 개념으로 구성한다. 각 컴퓨터에서 루트 프로세스를 리더 프로세스로 정의한다. 기본적인 아이디어는 크게 두 단계로 구성된다. 첫 번째 단계에서는 마스터에 해당하는 마스터 프로세스가 각 컴퓨터의 리더 프로세스에게 점대점 방식으로 메시지를 전송한다. 다음에 각 컴퓨터의 리더 프로세스는 기 구성된 이진트리 개념을 이용하여 자신이 마스터로부터 받은 메시지를 전송한다.

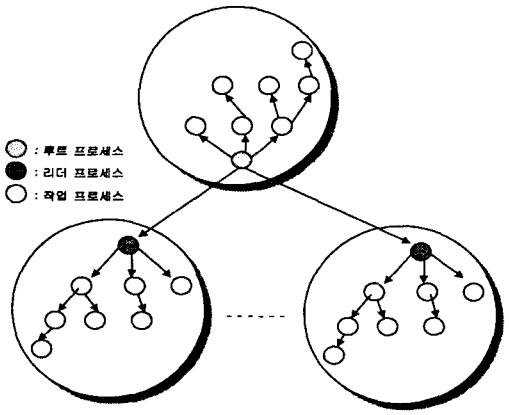

그림 3. 제안된 브로드캐스팅 방법

제안된 방법의 장점은 각 컴퓨터의 리더 프로세스간의 메시지 전송은 네크워크를 통하여 이루어지나, 각 컴퓨터내의 작업 프로세스간의 메시지 전송은 간단한 IPC(Inter Process Communication)를 통하여 이루어진다. 따라서 메시지 브로드캐스팅에 필요한 네트워크 통신시간은 작업 프로세스의 수에 비례하지 않고 참여하는 컴퓨터 수에 만 비례하게 된다.

즉 전체 컴퓨터의 수와 작업 프로세스의 수가 각각 N과 P일 경우 네트워크를 통한 브로드캐스팅은 N-1번이 이루어지고, 각 컴퓨터 내에서는 네트워크 통신이 없이 logK 단계를 수행하여 브로드캐스팅을 수행한다. 제안된 방법은 5장

에서 실험적 평가를 통하여 기존의 방법과 비교되고, 작업 프로세스의 수와 수행시간과의 관계를 분석하기 위한 기초적인 도구로 이용된다.

IV. 평가 모델과 부하분할

이 장에서는 본 논문의 주 과제인 작업 프로세스의 증가가 전체 수행시간에 어떤 영향을 주게 되는 지를 분석하기 위해, 후에 실험에 사용될 평가 모델과 몇 가지 부하 분할 방법을 소개한다.

1. 평가 모델

행렬 곱셈은 작업 단위의 분할이 간단하여 병렬처리 분야에서 성능 평가를 위해 가장 일반적으로 사용되는 방법 중 하나이다. 그림 4는 N×N 크기를 갖는 행렬 A와 B를 보여준다. 본 논문에서는 모든 작업 프로세스들에게 행렬 A를 전송하고, 행렬 B는 부하분할 정책에 따라 배분된다. 행렬 B에서 프로세스들로의 작업량을 결정하기 위해 각 작업 프로세스들로 전송할 작업량을 k, 전체 작업량을 n, 계산작업에 참여한 작업 프로세스의 수를 m이라 정의한다. 그래서 부하 분할에 있어 작업 프로세스들로의 작업량은 k=n/m에 의해 결정된다.

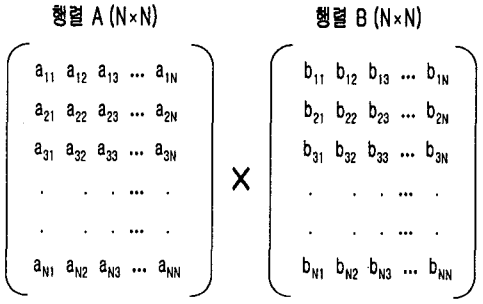

그림 4. 성능평가에 사용된 행렬 A, B

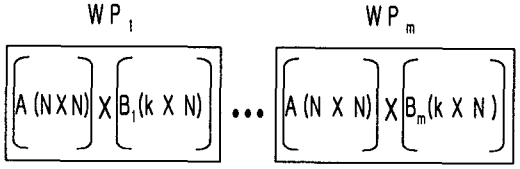

그림 5. 각 작업 프로세스에 할당된 작업량

결국 실제 각 작업 프로세스들은 SPMD방법으로, 전송된 행렬 A와 행렬 B에서 배분된 작업량 k를 수행한다. 그림 5는 전체 작업량을 참여하는 작업 프로세스들로 나누어 할당한 것을 보여준다. 각 작업 프로세스는 자신에게 할당된 작업량을 계산한 후, 마스터 프로세스로 전송한다. 그림 6은 마스터 프로세스에서 각 작업 프로세스에 의해 수행된 결과 값들을 취합한 형태를 보여준다.

본 논문에서는 위 방법을 참여한 프로세스 수만큼 균등하게 작업량을 배분하는 NLS 방법에 적용하여 평가를 하였다. 반면 LS, Pre-LS 방법에서는 프로세스 수에 따른 균등한 작업량 크기보다 작은 작업을 전송한다. 이들 방법들은 다음절에서 상세하게 설명된다. 예를 들어 전체 작업할 행렬식이 180*180이고, 프로세스수가 4개면, NLS 방법인 경우 마스터를 제외한 3개의 작업 프로세스들에게 60(180/3)개의 행이 전송된다. 하지만 LS, Pre-LS 방법에서는 6개의 행을 다음 작업으로 전송하여 평가한다. 이것은 작업 프로세스의 수를 증가하였을 때 마스터와 작업 프로세스간의 통신 대기시간과 유휴시간을 고려하기 위함이다. 이런 작업은 마스터와 작업 프로세스간에 각각 30회의 통신을 수행하도록 한다.

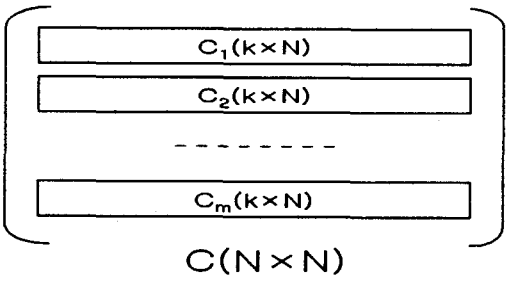

그림 6. 각 작업 프로세스들의 결과 값 통합

2. 부하분할

2.1. NLS(No Load Sharing) 방법

작업 프로세스의 부하를 고려하지 않고, 전체 작업량을 참여한 프로세스 수만큼 균등하게 배분하여 전송하는 방식이다. 언급된 평가모델에서와 같이 먼저, 마스터는 다른 슬레이브들에게 첫 번째 행렬식을 전송한다. 그런 다음 두 번째 행렬식에서의 일정비율(C/S)크기의 데이터를 각 슬레이브들에게 전송한다. 여기서 C는 두 번째 행렬식의 전체 행(Row)수를 말하며 S는 슬레이브 수를 나타낸다. 마스터가 계산수행에 참여할 경우에 S는 S+1이 된다.

각 슬레이브들은 자신의 계산작업을 수행한 후, 결과 값을 마스터로 전송한다. 마스터는 각 슬레이브들의 결과 값

을 취합함으로서 모든 계산 작업을 완료한다. NLS의 전체 수행시간은 참여하는 컴퓨터들의 성능에 좌우되며, 전송하는 데이터의 크기가 클 경우 마스터/슬레이브간의 통신 대기시간으로 성능저하의 원인이 된다.

2.2. LS(Load Sharing) 방법

이 방법은 그림 7과 같이 "Work Pool"을 기반으로 한다[17]. 수행할 작업은 큐(Queue)형태로 구성을 하고, 지정된 크기의 작업량(행수 6)을 슬레이브들에게 전송한다. 이는 빠른 응답을 보인 슬레이브들로 또 다른 작업을 부여해서, 통신 대기시간과 유휴시간을 단축하는 장점을 갖는다.

각 슬레이브는 자신이 수행할 작업을 마스터로부터 받아 수행하며 그 결과 값을 다시 마스터에게 전송한다. 마스터는 슬레이브들이 전송한 결과 값을 받아 처리하고 전송한 슬레이브들에게 다음 작업을 전송한다. 이런 작업은 전송할 행렬식의 전체 행수만큼 반복된다.

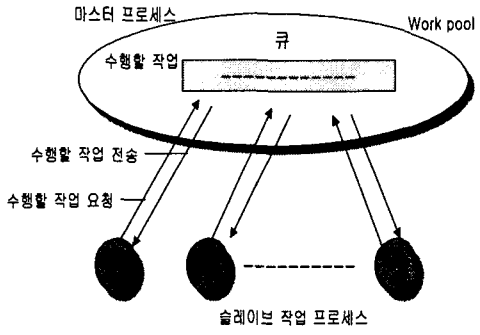

그림 7. Work Pool

이 방법은 NLS의 단점인 컴퓨터들의 성능 의존성을 해결할 수 있다. 하지만 각 슬레이브들에게 전송할 적절한 작업 크기 값을 고려해야 한다. 이는 전송 데이타의 크기가 작을 경우 오히려 많은 통신횟수가 발생하여 성능이 저하되기 때문이다. 또한 각 슬레이브들이 결과 값을 마스터에게 전송하고 다음 작업을 수행하기 위해서는 마스터로부터 작업을 전송 받아야만 하는데, 그로 인한 통신 대기시간이 발생한다.

2.3. Pre-LS 방법

LS 방법의 단점인 마스터와 슬레이브간의 통신 대기시간을 해결코자 그림 8과 같이 마스터는 슬레이브들로부터 결과 값 수신여부와 관계없이 일정시간 간격으로 슬레이브들에게 미리 다음 작업을 전송한다[18]. 즉 통신시간과 계산시간을 중복되게 함으로서, 자원간의 통신대기시간과 유휴시간을 최대한 줄여 NOW 환경에서 병렬 프로그램의 수행시간을 단축시킬 수 있다. 그밖에 슬레이브들로부터의 결과 값 수신방법 등은 Work Pool 기반의 LS방법과 동일하다.

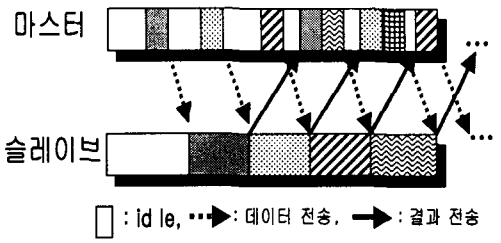

그림 8. Pre-LS 방법

2.4. 작업 프로세스 증가에 따른 Pre-LS 방법

이전 3가지 부하분할 방법은 각 컴퓨터에 하나의 작업 프로세스가 생성된 상태지만, 이 방법에서는 하나의 컴퓨터에 다수의 작업 프로세스를 생성한다. 전체적인 개념은 Pre-LS 방법과 동일하지만 작업 프로세스의 증가에 따라 전체 성능이 어떻게 변화하는지를 평가하기 위한 방법이다. 또한 전체 수행시간에서 작업 프로세스 수와 다른 요소(데이터 크기, 브로드캐스팅)들간의 관계에 대해서도 성능평가를 한다.

V. 실험결과 및 분석

본 장에서는 먼저 본 논문에서 제한된 새로운 브로드캐스팅 방법을 기존 방법과 비교한다. 다음에 제안된 브로드캐스팅을 사용하여 4장에서 설명한 여러 가지의 부하분할 방법들을 실험적인 환경에서 평가한다. 또한 이러한 실험을 기반으로 하여 하나의 컴퓨터에서 작업 프로세스의 수를 증가하였을 때, 전체 성능에 미치는 영향 등을 실험적으로 평가하고 그 결과를 분석한다.

1. 평가 방법

1.1. 브로드캐스팅

브로드캐스팅 전송방법은 크게 마스터/슬레이브, 트리방법이 있다. 본 논문에서는 작업 프로세스의 수를 증가하였을 때, 기존의 2가지 방법과 제안된 방법과의 성능평가를 통해 제안된 방법이 우수함을 보인다.

실험환경은 참여하는 컴퓨터 수(3대)를 고정시킨 상태에서 각 컴퓨터에 생성되는 작업 프로세스의 수와 전송 데이터의 크기를 증가시킨다. 이는 브로드캐스팅의 경우 작업 프로세스의 증가 및 전송 데이터 크기와의 관계를 알아보기 위함이다. 제안된 브로드캐스팅 전송 방법의 성능평가를 위해 실험 환경에 적합하게 새 라이브러리를 구현하여 평가를 수행하였다. 표 1은 브로드캐스팅 평가를 위한 환경을 보여준다.

표 1. 브로드캐스팅 평가를 위한 환경

컴퓨터수	전송 데이터 크기	작업 프로세스 수
3	32 * 32	6
		12
		18
		24
		30
	64 * 64	6
		12
		18
		24
		30
	128 * 128	6
		12
		18
		24
		30

1.2. 부하분할

먼저 표 2와 같은 상황에서 컴퓨터 수와 작업 프로세스의 수를 고정하여 NLS, LS, Pre-LS 방법의 성능 평가를 수행한다.

표 2. 부하분할 평가를 위한 환경

컴퓨터수	작업 프로세스 수	전송 데이터 크기
4	4	72 * 72
		144 * 144
		180 * 180

Pre-LS 방식에서 결과 값 수신여부와 관계없이 미리 전송되는 시간에 대해서는 다양하게 고려하지 않고 50ms를 기준으로 성능평가를 하였다. 물론 최적의 성능을 발휘하기 위해서는 적절한 데이터 전송시간을 결정해야 한다. 다양한 전송시간을 고려한 성능평가에 대해서는 향후에 논의한다.

1.3. 작업 프로세스 수

NOW에서는 컴퓨터와 컴퓨터들이 네트워크를 통해 자원을 공유한다. 그래서 컴퓨터들 내에 생성된 작업 프로세스들간의 효율적인 통신방법과 부하분할은 전체 성능을 향상시킨다. 본 논문에서는 실제 계산작업에 이전에 언급된 브로드캐스팅, 부하분할 방법들을 적용하여 작업 프로세스 수를 증가하였을 때 전체 수행시간에 어떤 영향을 주는지 실험을 통해 알아본다. 표 3은 컴퓨터 수와 전송 데이터의 크기를 고정시킨 상태에서 작업 프로세스 수를 증가하고 있다. 작업 프로세스의 수 1, 2, 3은 각 컴퓨터에서 생성되는 프로세스의 수를 나타낸다. 즉 1인 경우 전체 참여하는 컴퓨터 수가 4개이므로 전체 생성되는 프로세스의 수는 4개가 된다.

표 3. 작업 프로세스 수에 따른 성능평가를 위한 환경

컴퓨터수	전송 데이터 크기	작업 프로세스 수
4	72 * 72	1
		2
		3
	144 * 144	1
		2
		3
	180 * 180	1
		2
		3

2. 실험 환경

성능평가를 위한 실험환경은 다음과 같이 구성한다.

1. 운영체제 : Solaris 2.5.1
2. 소프트웨어 : LAM/MPI 6.3.2
3. CPU/Memory : SUN SPARC 110MHz, 32M

그림 9와 같이 실험에 참여한 컴퓨터들은 10Mbps 이더넷으로 연결된 9대의 동일한 환경의 같은 성능을 갖는 워크스테이션들이며 성능평가를 위해 다양한 크기의 행렬 곱셈계산을 수행하였다. 또한 9대의 컴퓨터 중에서 한 대가 마스터 컴퓨터가 되고 그 속에 마스터 프로세스가 위치하게 되며 나머지 컴퓨터(마스터 컴퓨터 포함)에 생성된 프로세스들은 작업 프로세스가 된다. 실험 결과에 사용된 값들은 10번의 수행 결과에 대한 평균값을 이용하였다.

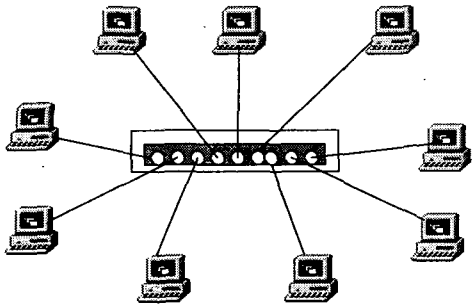

그림 9. NOW 구성

3. 실험 결과

3.1. 브로드캐스팅 결과 비교

일반적으로 브로드캐스팅 함수인 MPI_Bcast는 마스터와 슬레이브들 모두 이 함수를 호출하여 수행되었을 때 비로소 데이터 전송이 이루어진다. 즉 마스터/슬레이브들의 동기화가 이루어졌을 때 전송된다.

그림 10, 11, 12는 고정된 컴퓨터 수와 데이터 크기에서 작업 프로세스 수에 따른 브로드캐스팅 결과 비교를 보여준다. 본 논문에서 사용된 LAM/MPI의 마스터/슬레이브, 트리 방법은 생성된 작업 프로세스 수에 따라 구분된다. 그림 10은 데이터 크기가 32*32일 때 작업 프로세스의 증가에 따른 실험적인 결과를 보여준다. 프로세스의 수가 증가할수록 다른 방법에 비해 제안된 방법의 성능이 우수함을 알 수 있다. 특히 그림 12에서 데이터 크기가 128*128일때, 제안된 방법은 작업 프로세스 수가 6개 이하인 경우 별다른 차이가 없지만 작업 프로세스 수가 증가하여 30이 될 때 마스터/슬레이브 방법에 비해서 약 4.2배, 트리에 비해선 약 10% 성능향상을 보였다.

또 다른 성능차이의 원인은 브로드캐스팅 되는 데이터의 크기다. 그림 11, 12에서 작업 프로세스 수가 30, 데이터의 크기가 64*64에서 128*128로 증가했을 때, 마스터/슬레이브 방법과 제안된 방법간에는 약 3.2배에서 4.2배로 향상되었고, 트리방법과는 약 8.7%에서 10% 성능향상을 보였다. 결국, 본 논문에서 제안된 브로드캐스팅 방법이 기존의 방법보다 전송 데이터의 크기와 작업 프로세스 수가 증가할수록 많은 성능향상이 있음을 확인할 수 있다. 이는 각 컴퓨터별, 컴퓨터 내 작업 프로세스들간의 효율적인 전송으로 불필요한 통신 대기시간을 고려해준 결과이다. 마지막으로 컴퓨터 수에 따른 성능비교에서도 같은 성능차이를 확인할 수 있다.

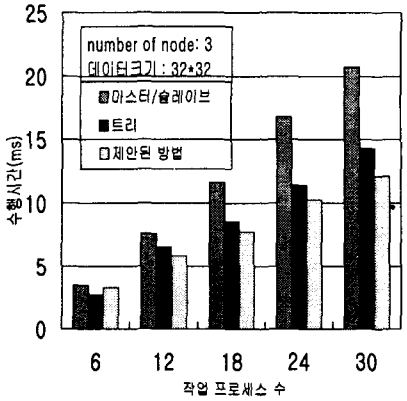

그림 10. 브로드캐스팅 결과 비교
(데이터크기: 32×32)

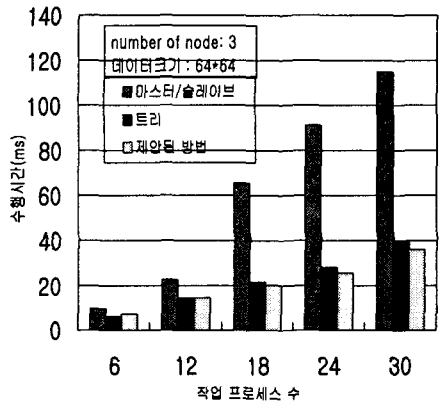

그림 11. 브로드캐스팅 결과 비교
(데이터크기: 64×64)

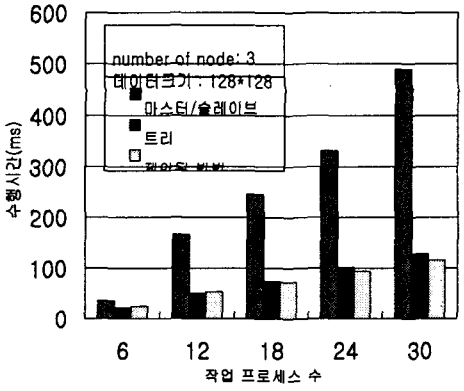

그림 12. 브로드캐스팅 결과 비교
(데이터크기: 128×128)

3.2 부하 분할 결과 비교

그림 13은 3가지 부하분할 방법에 작업 프로세스 수와 컴퓨터 수를 4로 고정한 후에, 전송 데이터의 크기를 증가하면서 실험한 결과를 보여준다. Pre-LS 방법은 데이터의 크기가 180*180일 때 NLS 방식에 비해서 약 3.7배 성능향상을 보였다. 이는 데이터를 처리하는 시간은 일정하다고 볼 때 전체 수행시간에 영향을 미치는 통신 대기시간을 단축시켜 이런 결과가 나온 것으로 판단된다.

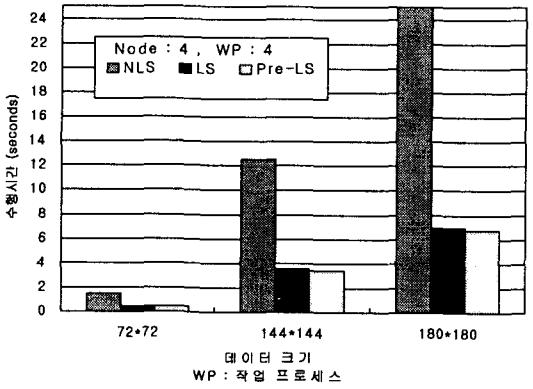

그림 13. 부하 분할 방법들간의 결과 비교

Pre-LS와 LS간의 성능비교에서도 약 3.2%차이를 보이는데 이것은 마스터가 작업할 데이터를 보내고 각 슬레이브는 처리한 결과를 마스터에게 전송한다. 그때 마스터와 슬레이브간에 통신대기 시간이 발생하기 때문이다.

결국, Pre-LS 방법은 작업 프로세스들간의 통신대기 시간과 유휴시간을 고려함으로서, 이전의 부하분할 방법보다 성능이 향상됨을 알 수 있었다. 또한 보다 나은 성능향상을 위해 Pre-LS 방법에서 주의할 점은 각 슬레이브의 결과 값 수신여부와 관계없이 미리 전송되는 일정시간 간격을 고려해야 한다.

3.3. 작업 프로세스 수에 따른 결과 비교

그림 14는 본 논문에서 제안한 브로드캐스팅 방법과 Pre-LS 부하분할 방법을 적용하여, 실제 마스터가 슬레이브들로 작업량을 배분하고 각 슬레이브들의 부분 결과 값을 최종적으로 취합한 시간이다. 또한 컴퓨터 수와 데이터 크기를 고정한 후에 각 컴퓨터에 생성되는 작업 프로세스 수를 증가하면서 실험을 하였다.

작업 프로세스 수와 전체 수행시간과의 관계에서 각 슬레이브에 전송된 데이터의 처리시간과 마스터로의 결과 값 전송시간은 일정하였다. 이는 마스터에서 각 슬레이브로 데이터를 전송하는 시간이 전체 수행시간을 좌우한다는 것을 의미한다. 또한 마스터가 각 슬레이브들에게 전송하는 작업 데이터의 크기에 따라 성능차이를 보였다.

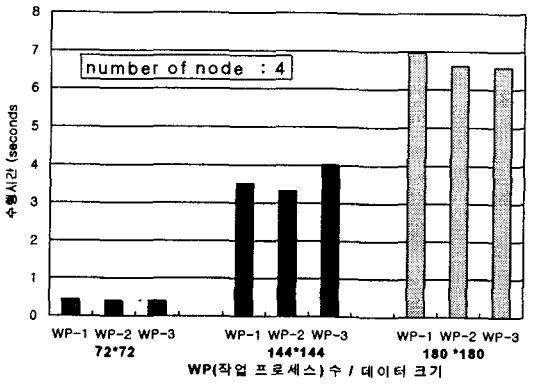

그림 14. 작업 프로세스 수와 수행시간 간의 결과 비교

실험 결과에서 데이터 크기가 144*144, 각 컴퓨터에 생성되는 작업 프로세스 수를 2에서 3으로 증가하였을 때, 약 17.4% 성능저하를 보였다. 이것은 NOW에서 작업을 계산할 때 작업 프로세스만을 증가하면 오히려 프로세스들의 계산시간보다 통신시간이 증가하기 때문이다. 즉 NOW에서 성능향상을 위해 작업 프로세스 수를 무한정 증가하기보다 브로드캐스팅, 부하분할방법 등을 함께 복합적으로 고려해야 한다. 또한 전체수행 시간에서 작업 프로세스들의 계산시간보다 네트워크를 통한 통신시간에 따라 전체성능이 좌우됨을 알 수 있었다.

VI. 결론

본 논문에서는 NOW에서 작업 프로세스 관점의 성능개선 요인(브로드캐스팅, 부하분할)을 분석하고, 작업 프로세스 수가 전체 성능향상에 어떤 영향을 주는지 실험적으로 평가하였다.

브로드캐스팅에서는 마스터/슬레이브, 트리, 제안된 방법에 전송되는 데이터의 크기와, 작업 프로세스 수를 증가하여 실험적으로 성능평가를 하였다. 제안된 방법이 기존 방법에 비해서 작업 데이터의 크기와 작업 프로세스 수가 적을 경우, 성능차이가 없었지만 작업 크기와 프로세스의 수가 증가할수록, 최대 약 4.2배 성능향상을 보였다. 이는 같

은 그룹의 프로세스들과 컴퓨터간의 전송방법을 달리 적용하여 불필요한 통신시간을 줄여주었기 때문이다.

부하분할 방법은 작업 프로세스들의 부하를 고려하고 미리 다음작업을 전송함으로서 통신 대기시간을 줄여 전체 수행시간을 단축시킬 수 있었다. Pre-LS 방식이 가장 좋은 성능을 보인 것은 그만큼 컴퓨터들의 유휴시간을 줄여 통신시간과 계산시간을 중복시킨 결과이다.

마지막으로 위의 두 가지 방법을 취합하여 전체 수행시간과 작업 프로세스 수와의 관계에서 프로세스 수를 증가하였을 때 오히려 성능이 저하되었다. 이것은 프로세스들의 계산시간보다 그들간의 통신시간이 증가하였기 때문이다. 그래서 성능향상을 위해서는 효율적인 브로드캐스팅, 부하분할방법과 데이터 크기를 함께 고려되어야 한다. 또한 NOW에서 프로세스들의 계산시간보다 통신시간이 전체 수행시간에 많은 영향을 주었다.

참 고 문 헌

[1] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", Mar. 1994.

[2] MPI Forum, MPI-2: Extensions to the Message-Passing Interface,http://www.mpi-forum.org/docs/mpi-20-html/mpi-2-report.html

[3] PVM : Parallel Virtual Machine, http://www.epm.ornl.gov/pvm/pvm_home.html

[4] Y. Cotronis and J. Dongarra, "Recent Advances in Parallel Virtual Machine and Message Passing Interface", 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, pp. 23-26, Sep. 2001.

[5] M. Baker and B. Carpenter, "MPJ: A Proposed Java Message Passing API and Environment for High Performance Computing", IPDPS Workshops, pp. 552-559, 2000.

[6] M. Bhandarkar, L. V. Kale, E. Sturler, and J. Hoeflinger, "Adaptive Load Balancing for MPI Programs", International Conference on Computational Science, pp. 108-117, 2001.

[7] S. Y. Lee and C. H. Cho, "Load Balancing for Minimizing Execution Time of a Target Job on a Network of Heterogeneous Workstations", JSSPP, pp. 174-186, 2000.

[8] A. Bevilacqua, "A Dynamic Load Balancing Method On A Heterogeneous Cluster Of Workstations", Informatica (Slovenia) 23(1), 1999.

[9] R. Rabenseifner and A. E. Koniges, "Effective Communication and File-I/O Bandwidth Benchmarks", Proceedings of the 8th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2001, pp. 24-35, Sep. 2001.

[10] K. Zielinski, M. Gajecki, and G. Czajkowski, "Parallel Programming Systems for LAN Distributed Computing", ICDCS, pp. 600-607, 1994.

[11] J. E. Moreira, D. Schouten, and C. D. Polychronopoulos, "The Performance Impact of Granularity Control and Functional Parallelism", LCPC, pp. 581-597, 1995.

[12] LAM/MPI Parallel Computing, http://www.mpi.nd.edu/-lam/

[13] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, "MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. In Proc. Symposium on Principles and Practice of Parallel Programming(PPoPP), pp. 131-140, Atlanta, GA, 1999.

[14] A. Piotrowski and S. Dandamudi, "A Comparative Study of Load Sharing on Network of Workstations", Proc. Int. Conf. Parallel and Distributed Computing System, New Orleans, Oct. 1997.

[15] A. Piotrowski and S. Dandamudi, "Performance of a Parallel Application on Network of Workstations", 11th Int. Symp. High Performance Computing systems, Winnipeg, pp. 429-440, Jul. 1997.

[16] C. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Computers", IEEE Trans. Computers, Vol.C-36, No.12, pp. 1425-1439, Dec. 1987.

[17] B. Wilkinson and C. M. Allen, "Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers", Prentice Hall, 1999.

[18] 구본근, "NOW 환경에서 개선된 고정 분할 단위 알고

리즘", 한국정보처리학회논문지, 제8-A권, 제2호, pp. 117-124, 2001.

조 수 현(Soo-Hyun Cho) 정회원

2000년 2월 : 금오공과대학교 컴퓨터공학과 졸업(공학사)
2002년 2월 : 금오공과대학교 컴퓨터공학과 대학원 졸업 (공학석사)
2002년 2월 ~ 현재 : 금오공과대학교 컴퓨터공학과 대학원 박사과정
<관심분야> : 병렬/분산처리, 이동 에이전트, 병렬 프로그래밍, Cluster Computing

김 영 학(Young-Hak Kim) 정회원

1984년 2월 : 금오공과대학교 전자 공학과 졸업(공학사)
1989년 2월 : 서강대학교 대학원 전자계산학과 졸업 (공학석사)
1997년 2월 : 서강대학교 대학원 전자계산학과 졸업 (공학박사)
1989년 ~ 1997년 : 해군사관학교 전산과학과 교수
1998년 ~ 1999년 : 여수대학교 멀티미디어학부 교수
1999년 ~ 현재 : 금오공과대학교 컴퓨터공학부 조교수
<관심분야> : 병렬 알고리즘, 분산 및 병렬처리 등