NOW 환경에서 개선된 고정 분할 단위 알고리즘

구 본 근†

요 약

NOW 환경에서 성능을 향상시키기 위해 부하 공유는 중요하다. 잘 알려져 있는 부하 공유 정책으로는 고정 분할 단위, 가변 분할 단위, 적응 분할 단위 정책이 있다. 가변 분할 단위 정책은 각종 매개변수에 따른 성능의 변동이 심하다. 하지만 고정 분할 단위 정책을 구현한 Send 알고리즘은 가변 분할 단위 정책의 알고리즘 보다 비슷하거나 조금 낮은 성능을 보이지만 가변 분할 단위 정책보다 매개변수에 따른 성능의 변화가 적다. 그러나, Send 알고리즘에서는 계산 시간과 통신 시간을 겹치지 않아 네트워크 상에서 발생하는 긴 대기 시간은 병렬 프로그램의 수행 시간에 많은 영향을 준다. 본 논문에서는 preSend 알고리즘을 제안한다. preSend 알고리즘은 Send 알고리즘과는 다르게 종속 노드로부터의 부분 결과 수신을 기다리지 않고 주 노드는 미리 종속 노드가 연산을 하는데 필요한 다음 데이터를 미리 전송함으로써 성능을 향상시킨다. 주 노드가 종속 노드에게 미리 다음 데이터를 전송하기 때문에 종속 노드는 다음 데이터를 주 노드로부터 수신하기 위한 휴지 시간이 없이 계속적으로 데이터를 처리할 수 있다. 즉, preSend 알고리즘은 계산 시간과 통신 시간을 겹치게 할 수 있다. 따라서, 네트워크 상의 긴 대기 시간의 영향을 감소시킬 수 있으며 NOW 환경에서 병렬 프로그램의 수행 시간을 감소시킬 수 있다. 두 알고리즘의 수행 시간을 비교하기 위해 우리는 320x320 행렬 곱셈을 이용하였다. 수행 시간을 비교한 결과는 preSend 알고리즘이 Send 알고리즘보다 짧은 수행 시간을 갖는다는 것을 보인다.

Refined fixed granularity algorithm on Networks of Workstations

Bon-Geun Goo†

ABSTRACT

At NOW (Networks Of Workstations), the load sharing is very important role for improving the performance. The known load sharing strategy is fixed-granularity, variable-granularity and adaptive-granularity. The variable-granularity algorithm is sensitive to the various parameters. But Send algorithm, which implements the fixed-granularity strategy, is robust to task granularity. And the performance difference between Send and variable-granularity algorithm is not substantial. But, in Send algorithm, the computing time and the communication time are not overlapped. Therefore, long latency time at the network has influence on the execution time of the parallel program. In this paper, we propose the preSend algorithm. In the preSend algorithm, the master node can send the data to the slave nodes in advance without the waiting for partial results from the slaves. As the master node sent the next data to the slaves in advance, the slave nodes can process the data without the idle time. As stated above, the preSend algorithm can overlap the computing time and the communication time. Therefore we reduce the influence of the long latency time at the network and the execution time of the parallel program on the NOW. To compare the execution time of two algorithms, we use the 320x320 matrix multiplication. The comparison results of execution times show that the preSend algorithm has the shorter execution time than the Send algorithm.

키워드 : NOW, 부하 공유, load sharing

1. 서 론

최근의 컴퓨터 관련 기술은 과거의 그 어느 때 보다 비약적으로 발전하고 있다. 특히 컴퓨터 하드웨어 기술은 프로세서의 성능을 비약적으로 향상 시켰으며, 이러한 고 성능의 프로세서는 많은 수치 연산이 필요한 과학, 기술 응용 분야뿐만 아니라 멀티미디어 응용 분야 등과 같이 다양한 형태의 대용량 데이터를 효율적으로 분석, 처리할 수 있다.

그러나, 현재의 컴퓨터 시스템에서 제공할 수 있는 계산 능력(computing power)보다 더 강력한 계산 능력을 필요로 하는 다양한 응용 분야가 있다[1]. 강력한 계산 능력이 요구되는 분야로는 과학적 또는 공학적인 문제에 대한 시뮬레이션, 기상 예보 시스템, DNA 구조 모델링 등이 있다. 이러한 응용 분야에서는 대용량의 데이터에 대한 대단히 많은 반복적인 계산을 통해 문제에 대한 해를 찾을 수 있다. 또한 해는 적절한 시간 내에 구해져야 한다. 이러한 요구를 만족

† 정 회 원 : 충주대학교 컴퓨터공학과 교수
논문접수 : 2000년 12월 7일, 심사완료 : 2001년 5월 8일

시키기 위해 많은 응용 분야에서는 병렬 시스템을 사용하여 문제에 대한 해를 구하고 있다.

병렬 시스템으로는 병렬 컴퓨터(*real* parallel computer)와 워크스테이션들의 네트워크(NOW, Networks of Workstations) 등이 있다. 병렬 컴퓨터는 전용의 상호연결 네트워크로 연결된 다수의 프로세서를 이용하여 강력한 계산 능력을 제공하는 고가의 시스템이다. 그러나, 병렬 컴퓨터가 강력한 계산 능력을 제공하지만 워크스테이션과 같이 많이 보급되어 있지 않아 상대적으로 유용성이 떨어진다.

NOW는 개인용 컴퓨터, 워크스테이션 등과 같은 비교적 저가의 컴퓨터들을 네트워크에 연결하여 서로의 계산 능력을 공유함으로서 강력한 계산 능력을 제공하는 기술이다. 즉, NOW는 네트워크에 연결되어 있는 많은 소규모의 컴퓨터들이 보유하고 있는 계산 능력과 메모리를 집합적으로 이용함으로써 병렬 컴퓨터를 모방할 수 있다[2, 3]. NOW 시스템과 병렬 시스템 사이의 차이점은 다음과 같다[2].

- 이기종 노드(node heterogeneity) : NOW를 구성하는 워크스테이션-노드(node)-은 하드웨어 또는 운영체계가 같을 필요가 없다. 즉, NOW의 각 노드는 종류가 다른 노드들로 구성될 수 있지만 병렬 컴퓨터는 같은 종류의 노드들로 구성된다.
- 노드에서의 부하(load on nodes) : NOW의 각 노드는 NOW의 한 노드로서 이용되는 것과 동시에 일반적인 응용에도 사용되므로 각 노드에서의 부하는 매우 다양하며, 예측하기가 쉽지 않다. 하지만, 병렬 컴퓨터의 경우는 모든 노드가 하나의 스케줄러에 의해 제어되기 때문에 각 노드에서의 부하 차이가 심하지는 않다.
- 네트워크상의 부하(load on network) : NOW의 네트워크는 병렬 처리와 관련한 메시지 전송뿐만 아니라 일반적인 처리를 위한 메시지 전송을 위해서도 사용되므로 네트워크의 부하를 미리 예측하기가 어렵다. 하지만, 병렬 컴퓨터는 노드간의 메시지 전송을 위해 전용의 상호 연결 네트워크(interconnection network)를 이용하므로 네트워크가 병렬 처리와 관련한 메시지 전송만을 담당하고 있기 때문에 네트워크의 부하를 미리 예측하기가 어렵지 않다.
- 노드의 고장(node failure) 관리 : NOW는 각 노드의 관리자가 개별적으로 그 노드를 관리하므로 노드의 고장에 대한 고려가 있어야 한다.

이러한 특징을 갖는 NOW 시스템에서 부하 공유(load sharing)은 NOW 상에서 수행되는 병렬 프로그램의 성능에 많은 영향을 준다. 그러나, 병렬 컴퓨터에서의 부하 공유와 관련한 연구에 비해 NOW에서의 부하 공유와 관련한 연구는 많지 않다[2]. 또한, 병렬 컴퓨터와는 달리 NOW는 각 노드의 성능의 차이, 메시지 전송을 위한 상대적으로 긴 메시지 전송 시간 등이 NOW 성능에 영향을 미치므로 효율적인 부하 공유는 NOW의 성능을 향상시킬 수 있다.

본 논문에서는 효율적인 부하 공유를 위해 부하 분할 단위가 고정된 고정 분할 단위(fixed granularity) 알고리즘인 Send 알고리즘을 개선한 preSend 알고리즘을 제시한다. Send 알고리즘은 부하 공유를 담당하는 주 노드(master node)가 종속 노드(slave node)에게 분할 단위만큼의 데이터를 전송하며, 그 데이터를 수신한 종속 노드는 정해진 연산을 수행한 후 그 결과를 다시 주 노드에게 전송한다. 종속 노드로부터 결과를 수신한 주 노드는 다음 분할 단위만큼의 데이터를 다시 종속 노드에게 전송하여 연산을 수행할 수 있도록 하고 있다. 이때, 주 노드는 종속 노드로부터 결과가 전송될 때까지 기다려야 한다. NOW에서는 데이터 전송을 위해 일반적인 네트워크를 사용하므로 상대적으로 긴 메시지 전송 시간은 병렬 프로그램의 수행 시간에 많은 영향을 준다.

본 논문에서 제시하는 preSend 알고리즘은 주 노드가 종속 노드로부터의 결과를 기다리지 않고 종속 노드가 연산을 수행하고 있는 동안에 다음 분할 단위만큼의 데이터를 미리 보내는 알고리즘이다. 따라서, 종속 노드는 현재의 분할 단위에 대한 연산을 마치고, 결과를 전송한 후 주 노드로부터 다음 분할 단위를 전송 받기를 기다리지 않고 이미 도착해 있는 분할 단위를 이용하여 즉시 연산을 수행할 수 있다. 즉, preSend 알고리즘은 계산 시간(computing time)과 통신 시간(communication time)을 겹치도록 하여 전체적인 수행 시간을 감소시키고자 하는 개선된 고정 분할 단위 알고리즘이다. preSend 알고리즘이 Send 알고리즘 보다 개선되었음을 보이기 위해 많은 응용 분야에서 이용되는 연산인 행렬 곱셈을 이용하였다. 본 논문에서는 두 개의 320x320 행렬 곱셈을 preSend 알고리즘과 Send 알고리즘을 이용하여 수행하였을 때 수행 시간을 비교하고, 각 알고리즘을 적용하여 병렬 프로그램을 수행했을 때의 수행 시간을 일반화하여 분석한다.

본 논문의 구성은 다음과 같다. 제2장에서는 부하 공유와 관련한 기존의 연구들을 소개하며, 제3장에서는 preSend 알고리즘에 대해 기술한다. 제4장에서는 preSend 알고리즘과 Send 알고리즘을 비교하기 위한 실험 환경 및 결과에 대해 기술하고, 제5장에서 결론을 내린다.

2. 관련 연구

부하 공유와 관련한 연구로는 J. Jacob의 노드의 계산 비

용과 통신 비용을 고려하여 태스크를 분산(spreading)하거나, 축소(shrinking)하는 모델을 제시하였다[8]. 이 모델에서는 노드 간의 통신 링크를 'Problem Edge Class'로 분류하였으며, 이 edge class를 바탕으로 태스크를 분산 또는 축소시킬 지를 판단하며, 각 통신 링크의 edge class를 잘못 예측하였을 경우에 성능상의 변화를 비교, 분석하였다. 이 연구의 결과에 의하면 태스크 매핑-나아가서는 부하 공유-에 통신 링크의 'Problem edge class'를 정확히 예측하는 것이 많은 영향을 줌을 알 수 있다.

NOW를 위한 부하 공유와 관련된 연구로는 A. Piotrowski의 각 부하 공유 알고리즘에 대한 비교 연구가 있다[2]. 이 연구에서는 부하 공유 알고리즘을 고정 분할 단위(fixed granularity), 가변 분할 단위(variable granularity), 적응 분할 단위(adaptive granularity)로 분류하였으며, 비교 연구에서는 고정 분할 단위 알고리즘은 Send 알고리즘을, 가변 분할 단위 알고리즘은 GSS(Guided Self-Scheduling) 알고리즘, Factoring 알고리즘, TSS(Trapezoidal Self-Scheduling) 알고리즘을 이용하여 각 알고리즘을 비교 분석하였다[2]. 또, 적응 분할 단위 알고리즘에서는 노드의 반응 시간(response time)을 이용하여 부하를 동적으로 분할하였다.

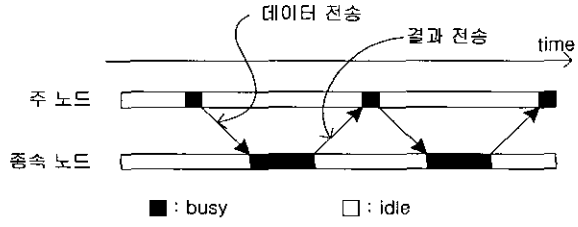
Send 알고리즘[4]은 데이터에 대한 연산을 수행할 각 종속 노드에게 고정 크기의 데이터를 전송하여 처리하는 간단한 부하 공유 알고리즘이다. GSS 알고리즘[5]은 전체 데이터 중에서 아직 처리가 되지 않고 남아 있는 데이터의 양에 따라 태스크의 크기 즉, 종속 노드에게 전송할 데이터의 양을 결정하는 알고리즘이며, Factoring 알고리즘[6]은 종속 노드가 처리하였던 이전의 데이터 양에 대한 고정된 크기의 비율로 다음에 처리해야 하는 데이터 양을 결정하는 방법이다. 반면, TSS 알고리즘[7]은 각 종속 노드가 처리해야하는 데이터의 양을 감소시키는 방법이다. 종속 노드의 반응 시간을 이용하여 노드가 처리해야 하는 데이터의 양을 결정하는 것으로 반응 시간이 늦은 노드에게는 적은 양의 데이터를 전송하여 처리하고, 반응 시간이 빠른 노드에게는 많은 양의 데이터를 전송하여 처리하는 알고리즘이다.

[2]에 의하면 대체적으로 Send 알고리즘과 GSS 알고리즘이 다른 부하 공유 알고리즘보다 좋은 성능을 보인 것으로 나타났다. 또, 가장 성능이 높은 노드를 각 노드에게 부하를 공유시키는 주 노드로서 동작을 할 때보다는 성능이 비교적 낮은 노드를 데이터에 대한 연산에는 참여하지 않고 부하 공유를 조정하는 주 노드로 동작을 시킬 때가 더 좋은 성능을 보였다. 그러나, GSS 알고리즘의 성능은 매개변수에 많은 영향을 받지만, Send 알고리즘은 GSS 알고리즘과의 성능 차는 크지 않으면서 매개변수에 따른 성능의 변화가 심하지 않은 강건한(robust) 알고리즘이다.

3. 부하공유 알고리즘

3.1 Send 알고리즘

앞의 비교 연구에서 대체적으로 좋은 성능을 보인 Send 알고리즘은 부하 공유를 담당하는 주 노드가 종속 노드에게 분할 단위만큼의 데이터를 전송한 후, 그 종속 노드로부터 결과를 전송 받을 때까지 대기하며, 종속 노드로부터 결과를 전송 받으면 다음 부하 분할 단위만큼의 데이터를 전송한다. 종속 노드는 주 노드로부터 수신한 데이터를 이용하여 정해진 연산을 수행한 후 결과를 주 노드에게 전송한다. 그리고, 주 노드로부터 연산을 수행할 때 사용할 다음 데이터를 수신할 때까지 대기해야 한다. (그림 1)은 Send 알고리즘에 따른 주 노드와 종속 노드의 동작을 나타내고 있다.

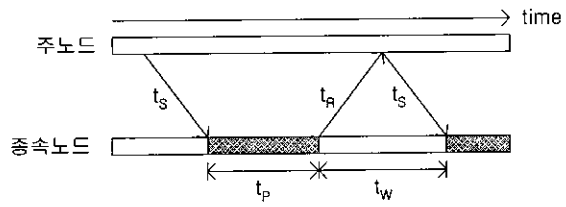


(그림 1) Send 알고리즘에서 주 노드와 종속 노드

(그림 1)에서 보인 것과 같이 주 노드는 데이터를 종속 노드에게 전송한 후 종속 노드로부터의 결과가 수신될 때까지 휴지(idle) 상태를 유지한다. 물론 하나의 주 노드가 다수의 종속 노드에게 데이터를 전송하지만, 데이터 전송을 위해 사용하는 네트워크에서의 상대적으로 긴 메시지 전송 시간으로 인해 주 노드의 휴지 상태는 존재한다. 이러한 상황은 종속 노드에게서도 발생한다. 종속 노드가 데이터를 이용한 연산을 하기 위해서는 주 노드로부터 데이터를 수신해야 하는데 네트워크에서의 긴 메시지 전송 시간 때문에 데이터 수신이 완료되는 시간동안 종속 노드는 휴지 상태를 유지한다. 또, 데이터에 대한 연산 수행을 완료한 후 그 결과를 주 노드에게 전송해야 연산에 사용할 다음 데이터를 주 노드로부터 수신할 수 있기 때문에 종속 노드는 결과를 전송한 후 다음 데이터를 수신할 동안 역시 휴지 상태를 유지한다.

(그림 2)는 Send 알고리즘에서 주 노드와 종속 노드의 상호 동작과 관련한 시간 관계를 나타내고 있다.

Send 알고리즘에서는 주 노드가 종속 노드에게 연산에 사용할 고정된 분할 단위의 데이터를 전송하고, 이 데이터를 수신한 종속 노드는 정해진 연산을 수행한 후, 부분 결과



(그림 2) Send 알고리즘에서 주 노드와 종속 노드의 동작

를 다시 주 노드에게 전송하는 상호 동작을 반복함으로써 전체 데이터에 대한 처리를 수행한다. 이때, 주 노드가 종속 노드에게 연산에 사용할 데이터를 전송하는 시간을 t_S, 수신된 데이터를 이용하여 종속 노드에서 연산을 수행하는 시간을 t_P, 종속 노드가 연산을 완료한 후 주 노드에게 부분 결과를 전송하는 시간을 t_R이라고 하자. 그리고, 노드에서 처리하는 고정 분할 단위 데이터의 수를 n이라고 할 때 병렬 프로그램의 수행 시간 t_{Send}는 다음과 같다. 물론, 수행 시간에는 주 노드에서 발생하는 각종 오버헤드 즉, 전체 데이터를 디스크 등과 같은 보조 기억 장치에서 읽어 오는 시간, 종속 노드로부터 수신한 부분 결과를 합병하는 시간 등을 제외한 주 노드와 종속 노드간의 상호 동작만을 고려한다. 이때, 전체 고정 분할 단위 데이터의 수가 N이고, 종속 노드의 수가 p일 때 $n = \lceil N/p \rceil$ 이다.

$$t_{Send} = n(t_S + t_P + t_R) \tag{1}$$

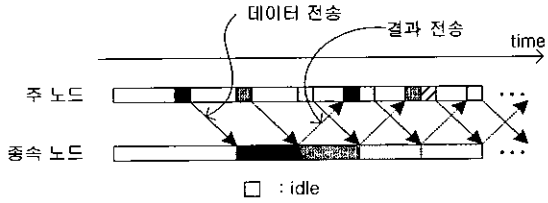
(그림 2)에서 종속 노드가 데이터를 이용하여 연산을 수행한 후 부분 결과를 전송하고, 다음 연산을 위한 데이터를 주 노드로부터 수신할 때까지 휴지 상태를 유지하는 시간을 t_W라고 하면, 위의 식 (1)은 다음 식 (2)와 같다.

$$\begin{aligned} t_{Send} &= n(t_S + t_P + t_R) \\ &= t_S + (n-1)(t_P + t_W) + t_P + t_R \\ &= t_S + nt_P + (n-1)t_W + t_R \end{aligned} \tag{2}$$

3.2 preSend 알고리즘

본 논문에서는 종속 노드의 계산 시간과 통신 시간을 겹치게 함으로써 t_W를 최소화하여 NOW에서의 프로그램 수행 시간을 감소시켜 NOW의 성능을 향상시키는 preSend 알고리즘을 제안한다. preSend 알고리즘은 Send 알고리즘과 같이 고정 분할 단위 알고리즘이다. 즉, preSend 알고리즘은 주 노드가 종속 노드에게 고정된 크기의 분할 단위만큼의 데이터를 전송을 한다. 그러나, preSend 알고리즘이 Send 알고리즘에서처럼 종속 노드로부터 결과를 수신할 때까지 주 노드가 휴지 상태를 유지하는 것은 아니며, 종속 노드에게 분할 단위만큼의 데이터를 전송한 후 적당한 시간이 지나면 종속 노드로부터의 결과 수신 여부와 관계없이 다음 0데이터를 미리 전송을 한다.

즉, preSend 알고리즘은 계산 시간과 통신 시간을 겹치게 하여 전체적인 수행 시간을 감소시키고자 한다. (그림 3)은 preSend 알고리즘을 이용할 경우에 주 노드와 종속 노드의 동작을 나타내고 있다.



(그림 3) preSend 알고리즘에서 주 노드와 종속 노드

(그림 3)에서 보인 것과 같이 주 노드는 데이터를 종속 노드에게 전송한 후 적당한 시간이 지나면 종속 노드로부터의 결과 수신과 관계없이 다음 데이터를 종속 노드에게 다시 전송한다. (그림 3)은 종속 노드가 이전에 수신한 데이터를 이용하여 연산을 수행한 후 결과를 주 노드에게 전송을 한 직후, 주 노드가 미리 전송한 다음 데이터가 종속 노드에게 수신되어 다음 데이터의 수신을 위해 휴지 상태로 가지 않고 즉시 다음 데이터를 이용하여 연산을 수행하는 이상적인 상황을 나타내고 있다. 이상적인 상황이 아닌 실질적인 상황에서도 역시 주 노드가 미리 다음 데이터를 종속 노드에게 전송하는 이 알고리즘을 이용할 경우에 종속 노드의 휴지 시간을 감소시킬 수 있다.

본 논문에서 제시하는 preSend 알고리즘 중에서 주 노드가 수행하는 알고리즘은 다음과 같다.

```
read data from disk ;
spawn the slave processes & store the process ID into the
process array ;
while all partial results are not gathered by master
      check the receiving buffer ; /* non-blocking receive */
      if there exists the message in buffer
          receive the partial result from slave ;
          merge the partial result to the result ;
      wait during the arbitrary time period ;
      send the data to slave process ;
kill all slave processes ;
print the result ;
```

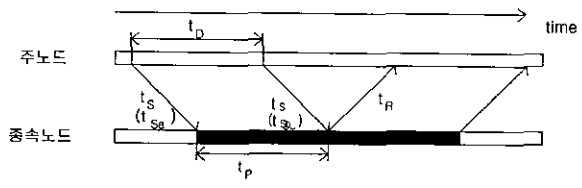
(주 노드에서 수행하는 부하 공유 알고리즘)

본 논문에서 제시하는 preSend 알고리즘 중에서 종속 노드가 수행하는 알고리즘은 다음과 같다.

```
while true
      wait the data from the master process ;
      process the data & make the partial result ;
      send the result to master ;
```

(종속 노드에서의 수행)

(그림 4)는 preSend 알고리즘에서 주 노드와 종속 노드의 동작과 관련한 시간 관계를 나타내고 있다.



(그림 4) preSend 알고리즘에서 주 노드와 종속 노드의 동작

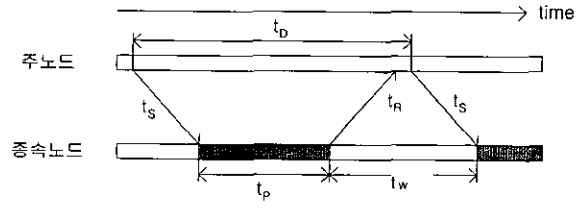
preSend 알고리즘에서 각 종속 노드에서 처리하는 고정 분할 단위 데이터의 수를 n이라고 할 때 병렬 프로그램의 수행 시간은 다음과 같다.

$$t_{preSend(t_D \approx t_P)} = t_S + nt_P + t_R \quad (3)$$

식 (3)은 식 (2)에서 $t_W = 0$인 경우 즉, 종속 노드의 계산 시간과 통신 시간이 이상적으로 겹쳐진 경우를 나타내고 있다. 이때, 계산 시간과 통신 시간을 이상적으로 겹치게 하기 위해서는 (그림 4)에서 $t_{Sa} \approx t_{Sb}$일 때 주 노드가 종속 노드에게 전송하는 연속된 두 데이터 전송 간의 지연 시간 t_D는 t_P이어야 한다.

하지만, t_S가 항상 같지는 않다. 즉, t_S는 네트워크를 통해 데이터를 전송하는 시간이므로 네트워크 상태에 따라 (그림 4)의 t_{Sa}와 t_{Sb}가 같지 않을 수 있으며, 종속 노드의 계산 시간 t_P도 수행하는 연산의 종류에 따라 일정하지 않다.

$t_D \gg t_P$인 경우 즉, 종속 노드의 연산 시간보다 주 노드에서의 연속된 두 데이터 전송간의 지연 시간이 긴 경우에 주 노드와 종속 노드간의 동작과 관련한 시간 관계는 (그림 5)과 같다.



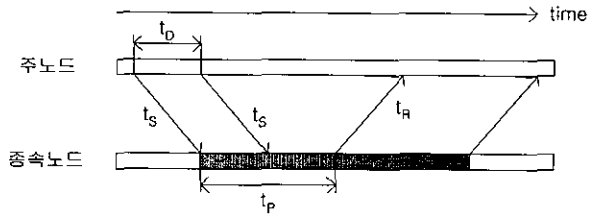
(그림 5) $t_D \gg t_P$인 경우 주 노드와 종속 노드의 동작

이 경우 병렬 프로그램의 수행 시간은 다음과 같다.

$$\begin{aligned} t_{preSend(t_D \gg t_P)} &= t_S + (n-1)(t_P + t_W) + t_P + t_R \\ &= t_S + nt_P + (n-1)t_W + t_R \end{aligned} \quad (4)$$

식 (4)에 의하면 t_D에 따라 preSend 알고리즘의 수행 시간이 Send 알고리즘의 수행 시간과 비슷하거나 길어진다.

$t_D \ll t_P$인 경우 즉, 주 노드에서 연속된 두 데이터 전송 간의 지연 시간이 아주 짧은 경우에 주 노드와 종속 노드 간의 동작과 관련한 시간 관계는 (그림 6)과 같다.



(그림 6) $t_D \ll t_P$인 경우 주 노드와 종속 노드의 동작

이 경우 병렬 프로그램의 수행 시간은 앞의 식 (3)처럼 아래와 같다.

$$t_{preSend(t_D \ll t_P)} = t_S + nt_P + t_R \quad (5)$$

하지만, 주 노드의 데이터 전송과 종속 노드들의 부분 결과 전송이 특정 시간대에 집중될 수 있어 네트워크 상의 충돌 현상, 재 전송 등으로 인해 식 (3)의 t_S와 t_R보다 식 (5)의 t_S와 t_R이 상당히 증가할 수 있다. 따라서, $t_{preSend(t_D \ll t_P)}$는 $t_{preSend(t_D \approx t_P)}$ 보다 길어진다.

지금까지의 분석에 의하면 주 노드가 종속 노드에게 연속된 두 데이터를 전송할 때 두 데이터 전송 간의 시간 간격 t_D가 preSend 알고리즘의 수행 시간에 영향을 준다. 특히, t_D가 종속 노드의 계산 시간인 t_P에 근접해 있을 때 알고리즘 수행 시간이 가장 짧을 수 있음을 알 수 있다.

4. 실험 환경 및 결과

본 논문에서 제시하는 preSend 알고리즘이 Send 알고리즘보다 효율적임을 보이기 위해 동일한 환경에서 preSend 알고리즘을 구현한 경우와 Send 알고리즘을 구현한 경우 그리고, GSS 알고리즘을 구현한 경우에 병렬 프로그램의 수행 시간을 비교하였다. 수행 시간의 비교를 위해 많은 응용 분야에서 이용되는 연산인 행렬 곱셈을 이용하였다. 본 논문에서는 두 개의 320x320 행렬을 곱셈하는 병렬 프로그램을 PVM(Parallel Virtual Machine) 라이브러리를 이용하여 작성, 수행하였다.

PVM은 NOW를 구성하는 노드에 주 프로세스(master process) 또는 종속 프로세스(slave processes)를 생성시키는 함수와 각 노드들 간의 메시지 전송을 위한 함수들로 구성된 C 언어와 Fortran 언어 라이브러리 및 몇 가지 유

틸리티들로 구성되어 있다. 이러한 PVM 루틴을 이용하여 병렬 컴퓨터 뿐만 아니라 NOW에서 병렬 프로그램을 작성하고, 수행할 수 있다.

본 논문의 실험에서는 PVM과 C 언어 프로그램을 이용하여 두 개의 320x320 행렬을 곱하는 프로그램을 주 노드에서 수행되는 프로그램과 종속 노드에서 수행되는 프로그램으로 나누어 각각 구현하였다. 주 노드에서 수행되는 프로그램은 부하를 분할하고 전송하는 역할을 하며, preSend 알고리즘과 Send 알고리즘을 구현한 프로그램이 수행된다. 종속 노드에서 수행되는 프로그램은 주 노드로부터 수신한 데이터를 이용하여 행렬 곱셈을 수행하고, 그 결과를 주 노드에게 전송하는 것으로 주 노드에서 수행되는 preSend 알고리즘과 Send 알고리즘을 구분하지 않고 동일한 프로그램을 사용하여 수행을 하였다.

작성한 병렬 프로그램을 수행하기 위해 우리는 NOW 환경을 Sun Sparc 호환 워크스테이션인 현대전자의 Axil 240 기종(64MB의 주기억장치) 다섯 대를 이용하여 구성하였다. 각 워크스테이션에서 수행되는 운영체제로는 Solaris 2.5를 이용하였으며, C 언어 컴파일러는 공개용 gcc 컴파일러를 이용하였다. 또 워크스테이션들은 10Mbps의 대역폭을 갖는 허브를 이용하여 연결되었다. 다섯 대의 워크스테이션 중에서 한 대는 주 노드로서 preSend 또는 Send 알고리즘을 구현한 프로그램을 수행하며, 나머지 네 대의 워크스테이션은 종속 노드로서 주 노드로부터 수신한 행렬 데이터를 이용하여 행렬 곱을 연산한 후 주 노드에게 결과를 전송하는 프로그램을 수행한다.

실험 프로그램에서 사용한 고정된 분할 단위로는 320x320 행렬에서 한 개의 열(column)을 사용하였다. 즉, 주 노드에서 종속 노드에게 전송하는 데이터로는 한 개의 열을 구성하는 320개의 데이터 즉, 열 벡터(column vector)이며, 종속 노드는 주 노드로부터 수신한 열 벡터를 이용하여 행렬 곱셈 연산을 수행한 후, 그 결과를 주 노드에게 전송한다. 따라서, 주 노드와 종 노드 사이의 데이터 전송은 행렬 곱셈을 위한 열 벡터 전송을 위해 320회, 종속 노드에서 수행한 행렬 곱셈의 부분 결과(partial result)를 전송하기 위한 320회이다. GSS 알고리즘에서 분할 단위는 '남아있는 열 벡터의 수 / 종속노드의 수'를 이용하여 결정하였다.

다음의 <표 1>은 두 개의 320 x 320 행렬 곱셈 연산을 수행할 때 preSend 알고리즘, Send 알고리즘, GSS 알고리즘을 이용하여 구현한 병렬 프로그램의 수행 시간을 나타낸 것이다. 병렬 프로그램의 수행 시간은 주 노드가 첫 번째 열 벡터를 종속 노드에게 전송한 시간부터 종속 노드로부터의 부분 결과들을 수신하여 두 개의 320 x 320 행렬 곱셈의 최종 결과를 얻기까지의 시간을 이용하였다. 수행 시간은 각 알고리즘을 구현한 - 또는, 매개변수를 달리한 - 병렬 프로그램들을 수행 시간대와 순서를 달리하여 10회 수행한 후 그 수행 시간의 평균을 이용하였다.

〈표 1〉 수행 시간

알 고 리 즘	수 행 시 간(초)
GSS	31.46
Send	28.45
preSend (delay = 0)	28.51
preSend (delay = 50)	27.87
preSend (delay = 100)	26.98
preSend (delay = 150)	27.42
preSend (delay = 200)	27.18
preSend (delay = 250)	27.26
preSend (delay = 300)	29.42
preSend (delay = 400)	38.79

<표 1>에서 GSS와 Send는 각각 GSS 알고리즘과 Send 알고리즘을 이용하여 구현한 병렬 프로그램의 수행 시간이며, preSend(delay = *n*)는 preSend 알고리즘을 구현한 병렬 프로그램의 수행 시간이다. preSend 알고리즘을 수행할 때의 매개변수 *n*은 주 노드가 종속 노드에게 고정 분할 단위만큼의 데이터를 전송할 때 연속된 두 전송 사이의 시간 간격을 나타내고 있으며, 본 논문에서는 밀리 초 단위의 시간 간격을 사용하였다.

<표 1>의 preSend(delay = 0)는 preSend 알고리즘에서 주 노드가 연속된 두 개의 열 벡터를 종속 노드에게 전송할 때 두 전송 사이에 시간 간격을 두지 않고 연속적으로 보내는 경우이다. 이 경우 320x320 행렬 곱셈을 수행한 두 알고리즘의 수행 시간이 많은 경우에 비슷하였다. 이것은 각 종속 노드가 행렬 곱셈에 사용할 열 벡터의 전송이 알고리즘 수행 초기에 집중됨으로 인해 발생하는 네트워크 상의 충돌 현상 및 재전송 과정에서 발생하는 시간적 손실이 원인인 것으로 판단된다. 또한, NOW를 동 기종의 워크스테이션들로 구성하였으므로 주 노드로부터 수신한 열 벡터를 이용하여 행렬 곱셈을 수행한 종속 노드들이 행렬 곱셈의 부분 결과를 주 노드에게 전송하는 것도 비슷한 시기에 집중되어 있다. 따라서, 종속 노드가 부분 결과를 주 노드에게 전송할 때 발생할 수 있는 네트워크 상의 충돌 현상 및 재 전송 과정 역시 알고리즘 수행 시간에 많은 영향을 준다.

그리고, preSend(delay = 300)과 preSend(delay = 400)의 수행 시간은 *Send* 알고리즘의 수행 시간과 비슷하거나 긴 결과를 얻었다. 이것은 주 노드가 종속 노드에게 연속된 두 개의 열 벡터를 전송할 때 두 전송간의 시간 간격이 너무 길어 종속 노드가 행렬 곱셈을 수행한 뒤 그 부분 결과를 주 노드에게 전송한 후까지도 다음 행렬 곱셈 연산에 사용할 열 벡터가 종속 노드에 도착하지 않았기 때문에 종속 노드는 다음 부분 결과를 위한 연산을 즉시 수행하지 못하

고 주 노드가 전송한 열 벡터가 종속 노드에 도착할 때까지 대기해야 하기 때문이다.

하지만 주 노드에서 전송하는 연속된 두 열 벡터의 전송 시간 간격이 50밀리 초에서 250밀리 초 사이인 경우에 preSend 알고리즘의 수행 시간은 Send 알고리즘 수행 시간보다 짧았다. 이것은 preSend 알고리즘이 계산 시간과 통신 시간을 겹치게 함으로써 병렬 프로그램의 수행 시간을 감소시킨 것으로 판단된다. 본 논문에서의 실험 결과에 의하면 Send 알고리즘보다 preSend 알고리즘을 이용하여 구현한 병렬 프로그램의 수행 시간이 최대 약 5%정도 감소되었다.

GSS와 같은 가변 분할 알고리즘의 경우 [2]에 의하면 Send와 비슷하거나 조금 나은 성능을 보이는 것으로 기술하였지만, 본 논문의 실험에 의하면 Send 알고리즘의 경우보다 오히려 좋지 못한 성능을 보이고 있다. 이것은 본 논문에서의 실험 환경과 [2]에서의 실험 환경과 구성이 동일하지 않은 점이 가장 큰 요인이라 판단되며, 또한 GSS와 같은 가변 분할 알고리즘이 여러 요인에 의해 성능 상의 변화가 심할 수 있음을 나타내고 있다.

5. 결 론

값이 저렴한 다수의 워크스테이션을 네트워크로 연결하여 병렬 시스템을 구성한 NOW는 각 노드간의 데이터를 전송하기 위한 전용의 네트워크가 없고, 각 노드가 이 기종으로 구성될 수 있다는 등의 특성을 가지고 있기 때문에 각 노드에 부하를 공유하는 정책이 전체 시스템의 성능에 많은 영향을 준다. 부하 공유 정책으로는 고정 분할 단위, 가변 분할 단위, 적응 분할 단위 정책이 있다. 이들 정책에 대한 관련 연구에서 고정 분할 정책을 구현한 Send 알고리즘이 가변 분할 단위 정책을 구현한 GSS, Factoring, TSS 알고리즘, 적응 분할 정책을 구현한 알고리즘보다 대체적으로 비슷하거나 조금 낮은 성능을 보이고 있지만, 여러 요인에 따른 성능의 변화가 상대적으로 심하지 않아 강건한 알고리즘이다.

Send 알고리즘은 주 노드가 종속 노드에게 처리할 데이터를 전송하며, 종속 노드로부터 결과를 수신한 후 처리할 다음 데이터를 다시 종속 노드에게 전송한다. 이때, 네트워크에서 발생하는 긴 전달 지연으로 인해 각 노드에서는 많은 시간을 휴지 상태로 유지하며, 또한 전체 수행 시간에서 많은 손실을 준다.

본 논문에서 Send 알고리즘이 갖는 이러한 문제점을 해결하기 위해 노드에서의 계산 시간과 통신 시간을 겹치게 함으로써 각 노드에서 휴지 상태를 유지하는 시간을 감소시켜 NOW의 성능을 향상시키고자 하는 preSend 알고리즘을 제안하였다. preSend 알고리즘은 주 노드가 종속 노드에게 분할 단위만큼의 데이터를 전송한 후 적당한 시간이 지나면 종속 노드로부터의 부분 결과 수신과 상관없이 다음 데이터를 미리 전송을 하는 것이다.

preSend 알고리즘이 Send 알고리즘보다 개선되었음을 보이기 위해 다섯 대의 Sun Sparc 호환 워크스테이션을 10 Mbps의 대역폭을 갖는 네트워크에 연결시켜 구성한 NOW에서 두 개의 320 x 320 행렬을 곱셈하는 병렬 프로그램을 작성하여 그 수행 시간을 비교하였다.

두 개의 320 x 320 행렬 곱셈의 수행 시간을 비교한 결과는 주 노드에서 종속 노드에게 전송하는 연속된 두 고정 분할 단위 데이터 전송간의 시간 간격 t_D가 50~250 밀리 초의 범위에 있으면 Send 알고리즘을 적용한 경우보다 preSend 알고리즘을 적용한 경우가 병렬 프로그램의 수행 시간이 최대 약 5%정도 감소되었다. 이 결과를 일반화하면 두 전송간의 시간 간격 t_D가 preSend 알고리즘을 적용한 병렬 프로그램의 수행 시간에 많은 영향을 주며, 특히 종속 노드에서의 계산 시간 t_P에 근접할수록 병렬 프로그램의 수행 시간이 짧아질 수 있음을 알 수 있다.

앞으로의 연구 과제는 preSend 알고리즘에서 주 노드에서의 두 데이터 전송 간의 시간 간격 t_D를 동적으로 결정할 수 있는 방법에 대한 연구가 계속되어야 한다. 또한, 이 기종의 워크스테이션들로 NOW를 구성했을 경우에 t_D를 결정할 수 있는 방안에 대한 연구도 필요하다.

참 고 문 헌

[1] B. Wilkinson and M. Allen, "PARALLEL PROGRAMMING : Technique and Applications Using Networked Workstations and Parallel Computers," Prentice Hall, 1999.

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

[3] T. Anderson, D. Culler, D. Patterson and the NOW team, "A Case for NOW(Networks of Workstations)," IEEE Micro, 15(2), pp.54-64, Feb. 1995.

[4] 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, July. 1997.

[5] 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.

[6] S. Hummel, E. Schonberg and L. Flynn, "Factoring : A Method for Scheduling Parallel Loops," Comm. ACM, Vol.35, No.8, pp.90-101, Aug. 1992.

[7] T. Tzen and L. Ni, "Dynamic Loop Scheduling for Shared-Memory Multiprocessors," IEEE Trans. Parallel Dist. Syst., Vol.4, No.1, pp.87-98, Jan. 1993.

[8] J. Jacob and S. Lee, "Task Spreading and Shrinking on Multiprocessor Systems and Networks of Workstations," IEEE Trans. Parallel and Distributed systems, Vol.10, No.10, pp.1082-1101, Oct. 1999.

구 본 근

e-mail : bggoo@gukwon.chungju.ac.kr

1991년 인제대학교 전산학과 졸업(학사)

1993년 부산외국어대학교 대학원 컴퓨터공학과(공학석사)

1998년 경북대학교 대학원 컴퓨터공학과 (공학박사)

1998년~2000년 충주대학교 컴퓨터공학과 전임강사

2000년~현재 충주대학교 컴퓨터공학과 조교수

관심분야 : 컴퓨터구조, 병렬컴퓨터 등