논문 2007-44CI-4-3

이기종 클러스터 시스템에서 Cilk와 MPI 특성 비교

(Comparing Cilk and MPI on a heterogeneous cluster system)

이 규 호*, 김 준 성**

(Kyuho Lee and JunSeong Kim)

요 약

최근 수년간의 급속한 기술의 발전과 대량생산 체제의 영향으로 개인용 컴퓨터와 간단한 네트워크 장비를 이용한 클러스터 시스템 구현이 용이해졌으나, 개인용 컴퓨터의 교체 주기가 짧아짐에 따라 시스템 구성을 자유롭게 할 수 있는 클러스터 시스템의 이기종화를 초래하였다. 이기종 클러스터 시스템을 이용하여 구축된 병렬처리 시스템의 경우 그 성능을 효율적으로 사용하기 위해서는 각 노드의 성능을 고려한 작업 관리가 필요하다. 본 연구에서는 이기종 클러스터 시스템에서 MPI와 Cilk 병렬처리 시스템의 특성을 성능측면에서의 speedup과 활용도측면에서의 프로그램 코드의 복잡도를 정량적으로 살펴보았다. 실험에 따르면 작은 데이터를 이용하는 경우 Cilk가, 큰 데이터를 이용하거나 정규화된 데이터 교환 형태를 갖는 경우 MPI가 더 좋은 성능을 보였으며 코드 복잡도의 경우 Cilk가 간결한 프로그래밍 스타일을 제공함을 보였다.

Abstract

Recently cluster system built from personal computers and network devices are easily and economically constructed. Rapid technological change discloses new processors on the market yielding cluster systems heterogeneity. A parallel system in heterogeneous environment needs work managers for utilizing the full power of the heterogeneous cluster system. In this paper, we compare MPI and Cilk in a heterogeneous cluster system in terms of performance and code complexity. Experimental results show that Cilk is better than MPI with small sizes of data transfers while MPI outperforms Cilk with big sizes of data transfers. Also, We find that Cilk requires less programming efforts to write a parallel program.

Keywords : heterogeneous cluster system, MPI, Cilk, parallel programming, code complexity

I. 서 론

기술의 발전에 따라 개인용 컴퓨터와 네트워크의 성능은 비약적으로 발전해 왔으며, 대량생산 체제의 영향으로 가격은 저렴해 지고, 보급은 폭발적으로 증가하고 있다. 이렇게 대량 생산되고 보급되는 개인용 컴퓨터와 네트워크 장비들은 저가의 가격대를 형성하며 쉽게 접근할 수 있지만, 메인프레임이나 슈퍼컴퓨터로 일컬어지는 고성능 시스템들은 아직 고가의 가격대를 형성하고 있으며, 쉽게 접근할 수 없다. 이런 상황에서 좋은 성능과 저렴한 가격을 가지고 있는 개인용 컴퓨터들을 이용하여 저비용의 고성능 병렬처리 시스템을 구축할 수 있다는 것은 매우 유용한 것임에 틀림없다. 클러스터 시스템은 이와 같이 개인용 컴퓨터 또는 워크스테이션과 간단한 네트워크 장비들로 구축된 분산 병렬 컴퓨팅 환경이다.[4][7] 급속한 기술 발전은 더 좋은 성능의 프로세서를 지속적으로 생산할 수 있도록 하였으며, 이것은 프로세서의 교체 주기를 짧게 하여 클러스터 시스템의 이기종화를 가져왔다.

이기종으로 구성된 클러스터 시스템을 이용한 병렬처리 시스템의 효율적인 활용을 위해서는 클러스터 시스템을 구성하는 개별 컴퓨터들의 성능을 고려할 필요가 있다.[4~6] 즉, 클러스터 시스템을 구성하는 각 컴퓨

* 학생회원, ** 정회원, 중앙대학교 전자전기공학부
(School of Electrical and Electronics Engineering, Chung-Ang University)

※ 이 논문은 2006년도 중앙대학교 학술연구비(일반연구비) 지원에 의한 것임
접수일자: 2007년4월6일, 수정완료일: 2007년6월21일

터들의 상대적인 성능에 따른 개별 컴퓨터에서 수행하는 작업의 크기 또는 양을 조절하는 것이다. 각 컴퓨터의 성능을 고려하지 않는 경우 상대적으로 낮은 성능을 가진 컴퓨터가 전체 클러스터 시스템의 성능을 좌우하게 되므로 그 성능을 최대한 활용할 수 없다.

본 논문에서는 이기종 클러스터 시스템에서, 쓰레드를 생성하고 교환하여 병렬처리를 수행하는 Cilk와 메시지 교환을 통해 병렬처리를 수행하는 MPI의 성능 및 코드 복잡도를 비교한다. Cilk와 MPI의 특성을 비교하기 위하여 생산년도와 성능이 다른 프로세서를 가진 개인용 컴퓨터 16대와 100Mbps로 동작하는 네트워크 스위치로 구성된 이기종 클러스터 시스템을 구축하여 실험한다. 본 논문의 구성은 다음과 같다. Ⅱ장에서는 Cilk와 MPI 병렬처리 시스템에 대해 간략하게 설명한다. Ⅲ장에서는 두 병렬처리 시스템의 비교를 위한 실험 환경과 실험에 사용된 벤치마크 프로그램 및 평가 방법을 설명한다. Ⅳ장에서는 실험 결과를 분석하고, Ⅴ장에서 결론을 내린다.

Ⅱ. 병렬 처리 시스템

1. Cilk Multithreaded 시스템

Cilk는 쓰레드를 이용한 병렬처리를 위해 개발된 프로그래밍 언어와 그 언어를 이용하여 작성된 프로그램을 실행할 수 있는 환경을 제공하는 runtime 시스템을 포함한다.[2][6] Cilk는 일반적인 병렬처리 프로그램의 개발을 위해 디자인되었지만 dataflow 모델을 기반으로 하여 병렬적인 데이터 형태 혹은 메시지 교환 형태로 작성하기 어려운 병렬처리 프로그램을 개발하는데 더욱 효과적이다. Cilk runtime 시스템에서 제공하는 dataflow 모델은 쓰레드를 생성하고, 생성된 쓰레드에서 필요한 데이터의 준비가 완료되면, 자동적으로 쓰레드와 데이터를 대기하고 있는 다른 프로세서에서 가져가 처리하도록 함으로써 병렬처리를 수행한다. Cilk 언어는 기본 C(ANSI C) 언어에 예약어를 추가한 형태로 runtime 시스템에서 데이터 관리, 메시지 관리, 계산 및 동기화 관리 등을 처리할 수 있도록 구성되어 있다. 이렇게 함으로서 C 언어를 알고 있는 프로그래머는 쉽게 Cilk 프로그래밍 언어를 사용할 수 있고, 병렬처리 프로그램 개발에 있어 쓰레드의 관리 및 교환 등 다른 부분에 관심을 가질 필요 없이 알고리즘상의 병렬성에 집중할 수 있도록 한다.

2. Message Passing Interface(MPI) 표준

MPI는 메시지 교환을 통해 병렬처리를 수행하는 프로그램을 개발하기 위한 라이브러리 표준이다.[1][4][7] 대학, 연구소 등 40여개의 단체가 참여하고 있는 MPI 포럼에서 개발되어 유지되고 있으며 병렬처리 시스템 개발에 가장 많이 사용되고 있다. 본 연구에서는 MPI 표준을 구현한 라이브러리들 중에서 이식성에 중점을 두고 구현된 라이브러리인 MPICH를 이용한다.[3] MPI 라이브러리를 이용해 개발된 병렬처리 프로그램은 send와 receive 명령어를 통해 각 컴퓨터 간에 메시지를 교환함으로서 병렬처리를 수행한다. 라이브러리에서 기본적으로 이용할 수 있는 함수는 MPI_Init, MPI_Send, MPI_Recv, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize 의 6개이며, 필요에 따라 추가적으로 125가지의 다양한 함수를 이용할 수 있다. MPI 라이브러리를 이용해 개발된 프로그램의 수행은 기본적인 메시지 교환 환경만이 요구되며 클러스터 시스템을 구성하는 개별 시스템의 상대적인 성능을 반영하는 등의 작업 관리 시스템은 필수요소가 아니다. 따라서 프로그래머가 직접 작업 관리 시스템을 설계하고 구현할 필요가 있다.

본 실험에서는 Cilk와 MPI의 서로 다른 병렬처리 시스템의 공정한 비교를 위하여 테스트 프로그램의 작성에 있어 Cilk와 MPI 각각의 특성을 유지하면서 각기 다른 시스템의 장점 반영이 가능한 경우 이를 활용하도록 하였다: MPI의 경우, 제공되는 특정 작업 관리 방식이 없기 때문에 Cilk의 dataflow 모델을 모방한 형태의 간단한 작업관리자를 구현하여 활용하였다. Cilk의 경우, 수 Kbytes 이상의 큰 데이터를 포함하는 쓰레드에 대한 생성/교환을 보장하지 못하기 때문에 큰 데이터를 이용하는 프로그램들에 대한 지원으로 MPI의 메시지 교환 방식을 모방한 형태의 데이터 전송 루틴을 구현하여 활용하였다.

Ⅲ. 실험 환경 및 평가 방법

1. 이기종 클러스터 테스트 베드

이기종 클러스터 시스템을 구축하기 위해 생산년도와 성능이 다른 16대의 x86 기반의 개인용 컴퓨터를 준비하였다. 각 컴퓨터는 최소한 128Mbyte의 메모리를 가지도록 구성하였으며, 모든 컴퓨터는 100Mbps로 동작하는 Nortel BayStack 70-24T 이더넷 스위치로 연결하였다. 표 1은 클러스터 시스템을 구성하는 개별 컴퓨터의 사양을 보여준다. 각 컴퓨터의 이름은 직접 작업

표 1. 실험에 사용된 개별 컴퓨터 사양
Table 1. Characteristics of each PC for the testbed.

이름	클럭주파수 (MHz)	메모리 (Mbytes)	상대적 성능	운영체제 (Linux)
ant1~6	400(P-II)	128	1.0	v7.3
ant7~8	450(P-III)	128	1.07~1.10	v7.3
ant9~10	800(P-III)	128	1.94~2.20	v9.0
ant11~16	800(P-III)	128	1.74~2.20	v7.3

을 수행하는 개체이므로 일개미를 의미하는 ant로 명명하였고, 상대적 성능은 ant1 컴퓨터의 성능을 기준으로 각 컴퓨터의 성능을 측정하여 정리한 것이다. 그밖에 각 컴퓨터의 프로세서 동작 주파수와 메모리 크기, 운영체제의 버전을 정리하였다.

실험에서 사용된 벤치마크 프로그램들은 페이징이 필요할 만큼 크기가 큰 프로그램들이 아니다. 벤치마크 프로그램들은 초기 프로그램 자체 로딩을 위한 디스크 접근을 제외하고 어떤 디스크 접근도 없다. 그러므로 디스크 성능은 고려하지 않는다.

2. 테스트 벤치마크 프로그램

Cilk와 MPI의 상대적 비교를 위하여 다양한 형태의 계산 및 통신 방식을 가지는 5가지 테스트 프로그램을 MPI 라이브러리를 활용한 C 언어와 Cilk 언어로 각각 작성하였다. Fibonacci, Traveling Salesman Problem, n-Queens 의 경우 재귀호출 알고리즘을 이용하여 생성되는 태스크*의 수를 쉽게 조절할 수 있도록 하였다. Matrix Multiplication, Laplace Equation 의 경우 매트릭스를 나누어 계산을 수행하는 방식으로, 매트릭스를 분배하는 방식을 조절함으로서 생성되는 태스크의 수를 쉽게 조절 할 수 있다. 재귀호출의 깊이와 매트릭스의 크기는 클러스터 시스템을 구성하는 16대의 컴퓨터에서 모두 작업을 수행할 수 있을 만큼 충분한 수의 태스크를 생성할 수 있도록 설정하였다.

- Fibonacci: 피보나치 수열을 계산하는 알고리즘으로서 다음과 같은 계산 방식을 가진다.

$$f(n)=\begin{cases} f(n-1)+f(n-2) & , \ n \geqq 2 \\ 1 & , \ n=1 \\ 0 & , \ n=0 \end{cases}$$

재귀호출 형태로 구현하고 n=43 으로 설정하여 실험한다.

- Traveling Salesman Problem (TSP): n개의 도시를 중복없이 모두 방문하는데 소요되는 최소 비용 또는 시간을 산출하기 위한 알고리즘으로 O(n!) 만큼의 계산량을 가진다. 재귀호출 형태로 구현하고 도시의 수 n=11 로 설정하여 실험한다.
- n-Queens(NQ): n×n 체스판 위에 n개의 queen을 안전하게 놓을 수 있는 방법이 몇 가지인가를 탐색하는 알고리즘이다. 이를 해결하기 위해서는 O(n^n) 만큼의 과도한 검색을 필요로 한다. 재귀호출 형태로 구현하고 체스판의 크기를 n×n=13×13 으로 설정하여 실험한다.
- Matrix Multiplication(MM): 두개의 행렬에 대하여 그 곱을 계산하는 알고리즘이다. 실험에 사용된 알고리즘은 하나의 행렬을 모든 노드에 전송하고 다른 하나의 행렬을 행 또는 열에 따라 나누어 각 노드에 계산을 맡기는 형태이다. 실험에서는 매트릭스 크기를 n×n=1100×1100으로 설정하였다.
- Laplace Equation(Laplace): Laplace 방정식을 이산적으로 계산하기 위한 알고리즘으로서 매트릭스를 나누어 서로 일부분을 교환하여 계산함으로써 문제를 해결한다. 이 벤치마크 프로그램의 경우 다른 네 개의 벤치마크 프로그램에 비해서 네트워크 오버헤드가 중요한 판단 요소가 된다. 실험에서는 매트릭스의 크기를 n×n=1000×1000으로 설정하였다.

이들 벤치마크 프로그램들은 실제 계산 또는 검색을 수행하는 코어 부분과 그 외의 오버헤드 부분으로 분리하여 작성되었다. 코어 부분은, 각 벤치마크 프로그램에 대하여, Cilk와 MPI에 공통된 코드를 동일한 최적화 옵션으로 컴파일하여 사용하였다. 오버헤드 부분은 태스크의 생성 및 분배 루틴과 각 컴퓨터의 성능을 측정하기 위한 테스트 루틴을 포함한다. MPI의 경우, 큐를 이용한 간단한 형태의 작업관리자를 구현하여 사용하였으며, 각 벤치마크 프로그램의 특성을 가장 확실하게 가지는 코어 코드를 활용한 테스트 루틴을 통하여 개별

† 태스크란 프로그램 수행에 있어 구분되는 개별적인 단위로, Cilk에서는 생성 및 교환의 대상이 되는 쓰레드를, MPI에서는 개별적인 작업(work descriptor)을 의미한다.

컴퓨터의 성능을 고려한 작업 분배가 이루어지도록 하였다.

3. 평가 방법

Cilk와 MPI 병렬처리 시스템의 비교를 위해 성능 측면에서는 speedup을, 활용도 측면에서는 코드 복잡도를 이용한다. Speedup은 병렬처리 시스템을 구성하는 컴퓨터의 수를 1개에서 16개까지 증가시켜가면서 단일 시스템에서의 수행시간에 대한 다수의 시스템으로 구성된 클러스터 시스템에서의 수행시간을 비율로 보여준다. 성능 측면의 비교에서 speedup을 주요 판단 자료로 이용하고 추가적으로 수행 시간을 참고하여 전체적인 성능을 판단한다.

코드 복잡도는 각 벤치마크 프로그램에 대하여 코어 알고리즘과 클러스터 환경에서의 병렬처리를 위해 추가적으로 필요한 명령어인 태스크 생성 및 분배 루틴 등을 포함한 그 외의 부분이 차지하는 라인수를 이용하여 평가한다.[8~9] 코드 복잡도의 경우 가장 간단한 비교 기준은 프로그램의 전체 코드 라인수이지만 전체 라인수는 프로그래머의 프로그래밍 스타일 등에 의해서 차이를 보일 수 있으므로 핵심적인 루틴들의 라인수를 이용하여 평가한다. 이 기준이 비록 정확한 코드 복잡도를 판단하기에는 부족하지만 쉽게 이해할 수 있는 기준이며 개략적인 코드의 복잡도를 판단하기에는 충분하다.

IV. 실험 결과

Cilk와 MPI 두 병렬처리 시스템의 공정한 성능 비교를 위해 실험 진행시 영향을 줄 수 있는 다른 프로그램의 수행을 모두 제거하였으며, Cilk에 대한 실험을 할 때는 MPI와 관련된 사항을 모두 제거하고, MPI에 대한 실험을 할 때는 Cilk와 관련된 사항을 모두 제거하여 외부의 영향을 최소화하였다. 실행 시간 측정은 프로그램의 코어 부분과 태스크의 생성, 데이터 교환을 위한 부분만을 측정하였다. 프로그램의 환경 및 데이터 초기화에 필요한 시간 등은 실행시간 측정에 포함하지 않았다. 더불어 OS 자체의 프로세스 스케줄링 등 랜덤하게 발생할 수 있는 영향을 최소화하기 위해 하나의 벤치마크 프로그램과 한 단계의 시스템에 대해 다섯 번의 동일한 실험을 반복하여 그 평균을 취하였다.

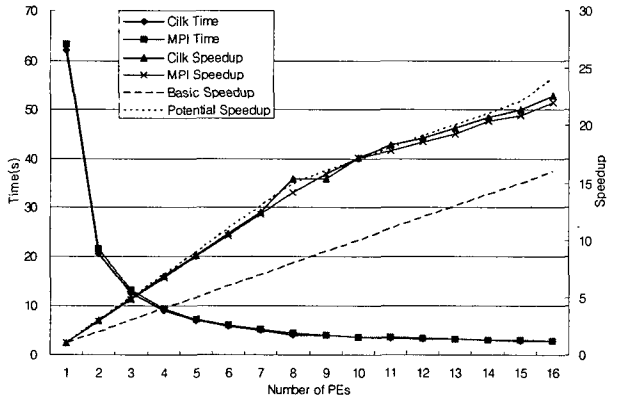

그림 1. Fibonacci의 실행 시간과 speedup
Fig. 1. Fibonacci: Times and speedup.

1. 성능 비교

그림 1~5는 이기종 클러스터 시스템에서의 각 벤치마크 프로그램별 성능 측정 결과를 보여준다. 각 그래프에는 이해를 돕기 위하여 이론적인 speedup을 같이 표시하였다. 클러스터 시스템 구축에 사용된 모든 컴퓨터의 성능이 동일하다고 가정할 경우에 대한 이론적인 speedup은 'Basic Speedup'으로 표현한 반면에 표 1에 보인 것과 같이 클러스터 시스템을 구성하는 각 컴퓨터의 상대적인 성능을 고려한 이론적인 speedup은 'Potential Speedup'으로 표시하였다. 각 그래프에서 X축은 클러스터 시스템을 구성하는 컴퓨터의 수를 나타내고, 왼쪽의 주 Y축은 초 단위의 실행 시간을, 오른쪽의 보조 Y축은 speedup 정도를 나타낸다.

그림 1은 Fibonacci의 실험 결과이다. MPI와 Cilk가 거의 비슷한 성능을 보여주고 있고, 클러스터 시스템을 구성하는 각 컴퓨터의 성능을 고려한 이론적인 speedup인 Potential Speedup에도 근접한 결과를 보여주고 있

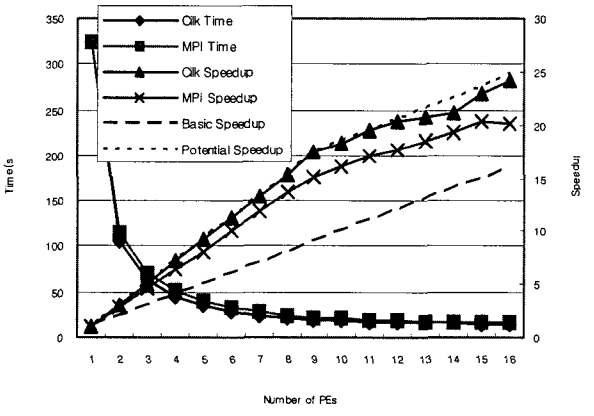

그림 2. Traveling Salesman Problem의 실행 시간과 speedup
Fig. 2. Traveling Salesman Problem: Times and speedup.

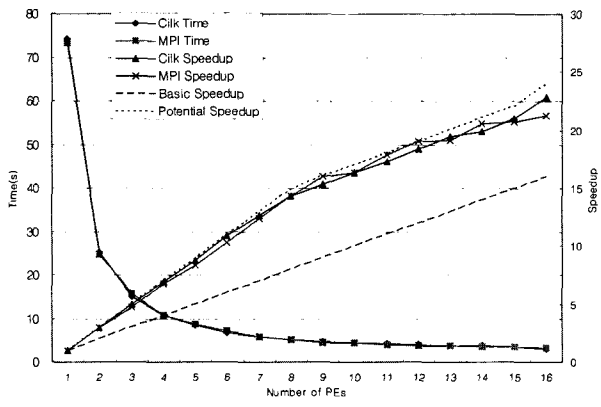

그림 3. n-Queens의 실행 시간과 speedup
Fig. 3. n-Queens: Times and speedup.

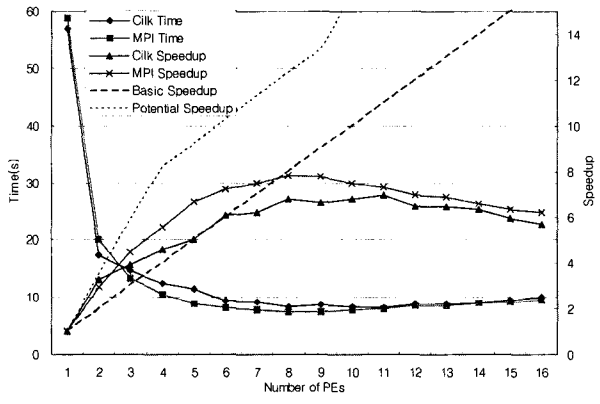

그림 4. Matrix Multiplication의 실행 시간과 speedup
Fig. 4. Matrix Multiplication: Times and speedup.

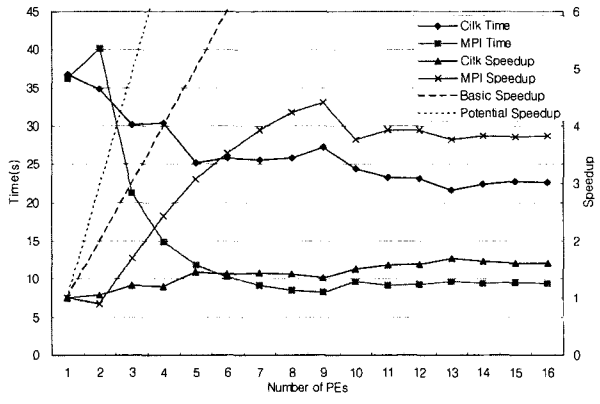

그림 5. Laplace Equation의 실행 시간과 speedup
Fig. 5. Laplace Equation: Times and speedup.

다. 한대의 컴퓨터에서 실행 시간을 보면 다소 차이가 있음을 볼 수 있는데, Cilk의 경우, 쓰레드가 생성되면서 동시에 계산을 진행할 수 있는 구조로 되어있으나 MPI의 경우, 모든 작업을 생성한 후에 계산을 진행하도록 되어 있기 때문이다. 이것은 나머지 벤치마크 프로그램들에서도 동일하게 해당되는 사항이다.

그림 2는 TSP의 실험 결과를 보여준다. Cilk의 경우 Potential Speedup에 거의 근접한 결과를 보여주고 있지만, MPI의 경우는 그보다 조금 못한 결과를 보여주고 있다. 실행 시간 그래프를 살펴보면 큰 차이가 없어 보일 수도 있으나, 주 Y축의 최대값이 300초대로 실질적인 차이는 5~10초 정도로 상당히 큰 차이를 보인다. 이러한 차이는 네트워크 오버헤드의 영향으로 발생한다. Cilk는 쓰레드의 생성된 순서와 관계없이 계산이 끝나면 결과가 수집되는 반면, MPI는 작업이 순서대로 생성되고 결과가 순서대로 수집이 되기 때문에 추가적인 대기시간이 필요하게 된다. 모든 노드에서 계산이 동시에 끝났다고 하더라도 순서대로 계산 결과를 수집하기 때문에 대기하는 노드가 발생되고 추가적으로 시간이 소요된다.

그림 3은 NQ의 결과를 보여준다. 그래프에서 확인할 수 있듯이 Fibonacci 벤치마크 프로그램과 비슷하게 MPI와 Cilk가 거의 같은 성능을 나타내고 있다. NQ는 TSP와 비슷한 알고리즘을 가지고 있지만 TSP와 달리 MPI와 Cilk의 speedup 차이가 거의 없다. NQ의 각 태스크는 알고리즘상 서로 다른 계산량을 가지게 된다. 또한 다음 단계의 태스크를 생성하기 전 추가적인 계산을 하도록 되어있기 때문에 Cilk에서 쓰레드를 생성하는 과정에서 더 많은 시간이 소요된다. MPI의 경우는 이 계산을 태스크 생성과정에서 일괄적으로 진행하기 때문에 성능면에서 큰 영향을 미치지 않는다. 따라서 Cilk의 성능이 TSP 경우에서 보다 조금 떨어져 MPI와 비슷한 결과를 보인다.

그림 4에는 MM의 결과를 보여준다. MM의 경우는 앞선 벤치마크 프로그램에서와 달리 병렬처리를 통한 성능 향상 정도가 사용되는 컴퓨터의 수에 따라 Basic Speedup에도 미치지 못하는 것을 볼 수 있다. 이는 계산에 소요되는 시간과 데이터 전송에 소요되는 시간의 상관관계에 의한 것이다. 6~7대 보다 적은 수의 컴퓨터를 이용할 경우 데이터를 분배하여 계산함으로서 얻어지는 소요시간 감소에 비해 데이터 전송에 따른 소요시간 증가의 정도가 작아 클러스터 시스템을 구성하는 개별 컴퓨터의 수에 비례하여 성능 향상이 됨을 볼 수 있다. 하지만, 8대 이상의 컴퓨터를 이용하게 되면 데이터를 분배하여 계산함으로서 얻어지는 소요시간 감소 효과보다 데이터 전송으로 인한 소요시간 증가의 정도가 더 커지게 되어 클러스터 시스템을 구성하는 개별 컴퓨터 수의 증가가 오히려 전체 성능의 저하로 나타난다. MM의 경우 간단하고 정규화되어 있는 데이터 전

송 형태를 가지므로 예측대로 메시지 교환 형식을 필요에 따라 쉽게 적용할 수 있는 MPI가 성능 우위를 보인다. Cilk는 계산 시간과 데이터전송 시간의 균형이 맞지 않는 경우 - 사용되는 컴퓨터의 수가 극히 적어 데이터 전송이 거의 불필요해지는 경우 ($n \leq 3$) 혹은 사용되는 컴퓨터의 수가 많아 계산에 소요되는 시간보다 데이터 전송에 소요되는 시간이 상대적으로 많아지는 경우 ($n \geq 11$) - 에 dataflow 모델에 기반한 작업관리시스템의 효율적인 스케줄링의 활용으로 MPI와 비슷한 성능을 보인다.

그림 5는 Laplace의 결과 그래프이다. MM과 같이 크기가 큰 매트릭스를 이용하는 벤치마크 프로그램으로서 MPI가 Cilk보다 좋은 성능을 보이고 있다. 그러나 Laplace에서의 성능 차이는 MM의 경우에 비해 더욱 크며, 이는 알고리즘의 차이에 기인하는 것이다. Laplace의 알고리즘은 각 노드들이 매트릭스의 일부분을 나누어 가지고, 나누어 가진 매트릭스의 일부분을 다시 인접 노드들 간에 교환하며 계산하는 방식이다. 따라서 네트워크를 통한 데이터 교환으로 발생하는 오버헤드가 실행시간의 주요 결정 요소가 된다. Laplace의 경우 MM과 달리 프로그램이 수행되는 동안 지속적으로 데이터 교환이 발생하게 되므로 Cilk와 MPI의 차이는 더욱 극명하게 나타나며 결과는 그래프에서 볼 수 있듯이 Cilk보다 MPI가 2배 이상 좋은 성능을 보인다.

2. 코드 복잡도 비교

표 2는 각 벤치마크 프로그램들의 코드 복잡도를 라인수로 종합함으로써 병렬처리를 위한 프로그래밍에 요구되는 인적 물적 노력의 일면을 가늠하고자 한다. 각 벤치마크 프로그램의 코드 복잡도를 실제 계산을 수행하는 코어 (Core) 부분과 병렬처리 시스템의 초기화 및 설정, 태스크의 생성 및 분배 등을 포함하는 오버헤드 (OH) 부분으로 구분하였다. 더불어, 각 작업에 반드시 필요한 기본 코드의 라인수 만을 고려함으로써 프로그래머의 개인적인 코딩 스타일의 영향을 최소화하였다.

전체 코드 복잡도(Total)를 비교하면 MPI가 Cilk에 비해 월등히 많은 라인을 사용하고 있다는 것을 알 수 있다. 이는 오버헤드부분의 비교에서 특히 그대로 나타난다. Cilk의 경우 runtime 시스템에서 쓰레드의 관리 기능 등을 기본적으로 제공하여 프로그래머의 프로그래밍 작업의 수고를 덜게 된다. 반면에, MPI의 경우 기본적으로 제공되는 작업관리시스템이 없으므로 이기종 클러스터 시스템의 효율적 활용을 위해서는 태스크의 분배를 위한 추가적인 기능들이 필요하다. MPI를 위한 효율적이고 포괄적인 작업관리 기능을 라이브러리 형태로 제공하는 방식으로 오버헤드 부분에서 오는 불이익은 쉽게 제거될 수 있으나 그러한 작업관리시스템에 대한 절대적인 기준은 설정되어 있지 않다. 코어부분만을 비교하면, Cilk의 코어부분에 쓰레드의 생성을 위한 추가적인 라인이 포함되어 있기 때문에 Cilk가 MPI 보다 조금 더 많은 라인을 사용하고 있는 것을 볼 수 있다.

표 2. 벤치마크 프로그램들의 코드 복잡도 비교
Table 2. Code Complexities.

벤치마크	Cilk			MPI		
	Total	= Core	+ OH	Total	= Core	+ OH
Fibonacci	13	10	3	71	7	64
TSP	32	23	9	91	18	73
NQ	28	26	2	85	19	66
MM	36	9	27	71	9	62
Laplace	59	17	42	33	15	18

전체적으로 runtime 시스템에서 제공하는 작업관리 시스템을 기반으로 하는 Cilk를 활용한 병렬처리 프로그래밍이 MPI 보다 손쉽고 간단한 프로그래밍 스타일을 제공한다고 할 수 있다. 더불어, Cilk 프로그램은 클러스터 시스템의 구성 환경에 대하여 매우 유연하다는 장점을 갖는다. 즉, 클러스터 시스템을 구성하는 개별 컴퓨터의 추가와 제거 및 교체에 따른 설정 변화에 무관한 프로그램의 수행이 가능하다. 이에 반해 MPI는 클러스터 시스템을 구성하는 개별 컴퓨터의 변경에 따라 실행 환경의 설정을 변경해야 하고, 프로그램 실행 명령을 입력할 때에도 이용할 컴퓨터의 수를 명시해야 하므로, 클러스터 시스템의 손쉬운 활용면에서 상대적으로 그 유연성이 떨어진다.

VI. 결 론

클러스터 시스템은 널리 사용되고 있는 개인용 컴퓨터와 네트워크 장비로 쉽고 값싸게 구축될 수 있다. 기술의 급속한 발전으로 개인용 컴퓨터의 성능이 지속적으로 증가하고, 클러스터 시스템을 구성하는 컴퓨터들을 추가 또는 교체함으로서 전체 클러스터 시스템의 이기종화를 야기했다. 본 논문에서는 이기종 클러스터 환경에서 MPI와 Cilk의 두 병렬처리 시스템에 대하여 성능과 코드 복잡도를 비교하였다.

실험을 통한 테스트 프로그램의 성능 평가 결과를 종

합하면, 작은 데이터를 이용하거나 dataflow 모델에 기반한 작업관리시스템의 효율적 적용이 가능한 경우, Cilk와 MPI의 차이가 작기는 하지만 Cilk가 더 좋은 성능을 보였다. 반면에 큰 데이터를 이용하거나 간단하고 정규화되어 데이터 교환 형태의 예측이 가능한 경우 MPI가 더 좋은 성능을 보여주었고, 그 성능 차이는 크게 나타났다.

코드 복잡도 및 클러스터 시스템의 활용도 측면에서 보면 Cilk가 MPI보다 손쉽고 유연하다는 것을 알 수 있다. MPI는 클러스터 시스템을 구성하는 개별 컴퓨터의 성능을 반영할 수 있는 포괄적이고 효율적인 작업관리시스템을 라이브러리화하여 제공함으로써 보다 쉬운 병렬처리 프로그램 개발을 할 수 있도록 개선될 수 있다. 또한, Matrix Multiplication과 Laplace Equation 벤치마크 프로그램에서 볼 수 있듯이 Cilk는 데이터 교환에 의존적인 프로그램들에 대한 개선의 여지가 있다.

참 고 문 헌

[1] *M. Snir, "MPI: The complete reference", MIT Press, MA: Cambridge, USA, 1996.*

[2] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: an efficient multithreaded runtime system," in PPoPP'95, Santa Barbara, 1995.

[3] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," Parallel Computing, vol. 22, no. 6, pp. 789 - 828, Sep 1996.

[4] L. T. Yang and M. Guo, "High-Performance Computing: Paradigm and Infrastructure", John Wiley & Sons, 2006.

[5] Y. K. Kwok, "On exploiting heterogeneity for cluster based parallel multithreading using task duplication", Journal of Supercomputing, 2003.

[6] S. Baek, K. Lee, J. Kim and J. Morris, "Heterogeneous Network of Workstations", Lecture Note on Computing Science, Vol. 3189, Springer-Verilag, pp. 426-439, 2004.

[7] R. Buyya, "High-Performance Cluster Computing: Architectures and Systems", Prentice Hall PTR, 1999.

[8] K. Lee and J. Kim, "Performance and Complexity of Prallel Programming", Joint Conference on Communications and Information, pp75, 2005.

[9] S. VanderWiel, D. Nathanson and D. J. Lilja, "Complexity and Performance in Parallel Programming Languages", International Workshop on High-Level Parallel Programming Models and Supportive Environments, pp. 3-12, April 1997.

저 자 소 개

이 규 호(정회원)

2004년 중앙대학교 전자전기 공학부 학사 졸업.

2006년 중앙대학교 전자전기 공학부 석사 졸업.

<주관심분야 : 병렬처리 시스템, 시스템 성능 분석, 시스템 소프트웨어>

김 준 성(정회원)

1991년 중앙대학교 전자공학과 학사 졸업.

1993년 중앙대학교 대학원 전자공학과 석사 졸업.

1998년 미국 미네소타대학교 전기공학과 박사 졸업.

2002년~현재 중앙대학교 전자전기공학부 부교수

<주관심분야 : 컴퓨터 구조, 병렬처리 시스템, 홈네트워크, SoC 구조, 시스템 성능 분석>