논문 2009-46CI-6-7

DWM: 이기종 클러스터 시스템의 동적 자원 관리자

(A Dynamic Work Manager for Heterogeneous Cluster Systems)

박 종 현*, 김 준 성**

(JongHyun Park and JunSeong Kim)

요 약

고속 네트워크를 통해 연결된 다수의 컴퓨터를 호환성 있는 통신 라이브러리를 활용하여 병렬처리를 수행하는 클러스터 컴퓨팅은 가격대 성능비에서 장점을 가지며 다양한 응용분야에서 실용화되고 있다. 이기종 클러스터 환경에서는 클러스터 시스템을 구성하는 개별 노드의 성능이 직접적인 영향을 주기 때문에 효율적인 자원 관리가 매우 중요하다. 본 논문에서는 DWM(Dynamic Work Manager)라 불리우는 동적 자원 관리자를 제안한다. DWM은 성능이 다른 이기종 클러스터 시스템에서 각 노드의 자원을 충분하게 활용할 수 있도록 설계되었다. DWM을 사용하여 다양한 벤치마크 프로그램을 클러스터 시스템에서 실행시켜 봄으로써 DWM의 성능과 프로그램의 복잡도를 측정한다. 실험 결과로부터 DWM의 사용은 이기종 클러스터 시스템의 성능을 효율적으로 활용하는 동시에 MPI 병렬처리 프로그램의 작성을 용이하게 함을 알 수 있다.

Abstract

Inexpensive high performance computer systems combined with high speed networks and machine independent communication libraries have made cluster computing a viable option for parallel applications. In a heterogeneous cluster environment, efficient resource management is critically important since the computing power of the individual computer system is a significant performance factor when executing applications in parallel. This paper presents a dynamic task manager, called DWM (dynamic work manager). It makes a heterogeneous cluster system fully utilize the different computing power of its individual computer system. We measure the performance of DWM in a heterogeneous cluster environment with several kernel-level benchmark programs and their programming complexity quantitatively. From the experiments, we found that DWM provides competitive performance with a notable reduction in programming effort.

Keywords : Parallel processing, Cluster computing, Message Passing Interface, Dynamic Work Manager, Programming complexity

I. 서 론

클러스터 컴퓨팅이란 하나의 큰 문제를 여러 개의 태스크로 나누어 네트워크로 연결된 여러 개의 프로세서를 이용하여 한꺼번에 수행함으로써 해결하는 방식이다. 클러스터 시스템은 높은 가용성, 확장성과 더불어 우수한 성능을 제공하여 과학계산 분야를 포함하여 다양한 분야에서 널리 활용되고 있다[2, 5]. 최근 수년간의 기술 발전으로 개인용 컴퓨터들의 성능은 급격히 증가되었고, 어떤 방식으로든 네트워크에 연결되게 되었다. 이러한 발전은 클러스터 시스템을 쉽게 구축할 수 있게 했지만, 개인용 컴퓨터의 교체주기 또한 짧게 함으로써 클러스터 시스템의 이기종화를 초래하였다. 이렇게 이기종화된 클러스터 시스템의 성능을 최대한 이용하기 위해서는 클러스터를 구성하는 개별 컴퓨터들의 성능을 고려한 태스크 관리자가 필요하게 된다[8~11]. 개별 컴퓨터들의 성능을 고려하지 않은 경우 클러스터 시스템 전체의 성능은 가장 낮은 성능을 가진 컴퓨터에 좌우되기 때문이다.

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

※ 이 논문은 2008년도 중앙대학교 학술연구비 지원에 의한 것임

접수일자: 2009년8월11일, 수정완료일: 2009년11월2일

본 논문에서는 이기종 클러스터 시스템을 구성하는 개별 컴퓨터들의 성능을 고려한 동적 자원 관리자로서 DWM (Dynamic Work Manager)을 제안한다. DWM은 사용자에게 라이브러리 형태로 제공되며 태스크의 처리와 메시지의 전달을 위한 별도의 쓰레드로 이루어져 있다. 또한 DWM을 이용함에 있어서 이기종 클러스터 시스템의 성능을 효율적으로 활용하고 있는지 검증하기 위해 테스트 프로그램을 작성하고 실험을 통하여 확인한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 MPI 라이브러리와 선행연구 결과물인 WPM(Work Packet Manager)과 더불어 DWM의 내부 구조 및 라이브러리로 제공되는 함수들을 간략히 설명한다. Ⅲ장에서는 DWM의 성능 측정을 위해 간단한 벤치마크 프로그램을 통한 실험을 진행하고 그 결과를 분석한다. 마지막으로 Ⅳ장에서는 본 논문의 결론과 전반적인 내용에 대한 요약을 기술한다.

Ⅱ. 본 론

1. MPI 와 WPM (Work Packet Manager)

MPI(Message Passing Interface)는 클러스터 환경을 구축하는데 가장 많이 사용되는 라이브러리 표준으로 send와 receive 명령어를 통해 클러스터를 구성하는 개별 컴퓨터 간에 메시지를 교환함으로써 병렬처리를 수행한다[1, 3]. MPI 라이브러리를 이용하여 개발된 프로그램의 수행은 기본적인 메시지 교환 환경만이 제공되며, 특별히 준비된 태스크 분배 및 관리 환경은 없다. 따라서 프로그래머가 직접 태스크를 관리하는 루틴을 설계하고 구현하여 이용해야 한다. 본 연구에서는 MPI 표준을 구현한 라이브러리들 중에서 이식성에 중점을 두고 구현된 라이브러리인 MPICH를 이용한다[7].

클러스터 환경을 위한 병렬처리 프로그램을 작성하는 경우 각 개별 시스템에 동일한 양의 작업을 할당하는 방식은 프로그래밍을 용이하게 한다. 하지만, 이기종 클러스터 시스템에서 이와 같은 단순한 작업 할당은 자원을 낭비하는 결과를 초래한다. 이기종 클러스터 환경에서 클러스터를 구성하는 개별 컴퓨터간의 작업 분할 방식은 성능에 직접적인 영향을 미친다. WPM (Work Packet Manager)는 MPI를 위한 정적 자원 관리 시스템이다[12]. 이는 클러스터를 구성하는 개별 컴퓨터의 성능을 고려한 작업 분배를 위해 테스트 루틴의 수행을 통한 각 컴퓨터의 상대적인 성능 평가 단계를 포함한다. 테스트 루틴은 수행하고자 하는 프로그램의 특성을 반영한다. 평가 결과를 이용하여 클러스터 내의 개별 컴퓨터가 처리할 수 있는 범위 내에서 태스크를 할당함으로써 WPM을 통한 프로그램의 성능은 해당 클러스터 시스템에서 기대될 수 있는 이상적인 성능의 기준이 되며 본 논문에서 제안하는 DWM의 성능에 대한 비교 대상으로 활용한다.

2. DWM: Dynamic Work Manager

DWM은 MPI 표준에 태스크의 동적 관리와 교환을 위한 dataflow 모델 기반의 루틴을 라이브러리 형태로 추가하여 구현되었다. DWM은 처리를 위해 대기하는 태스크들을 위한 큐와 처리가 완료된 태스크들을 위한 큐로 구성된다. 클러스터 컴퓨팅 환경에서 프로그램 실행 중 생성된 태스크는 모두 DWM이 관리하게 되며 DWM 내부의 큐에 저장된다. DWM은 저장된 태스크를 slave 컴퓨터들에게 분배하고 수행이 끝난 태스크 결과는 DWM에 다시 전송된다.

그림 1은 DWM을 사용하는 병렬처리 프로그램의 실행 과정을 나타낸다. 우선적으로 병렬처리를 위한 기본적인 환경 설정을 하고, DWM을 초기화한다. 다음으로 태스크를 생성하고, 생성된 태스크를 DWM이 관리할 수 있도록 master 내부의 DWM이 관리하는 큐에

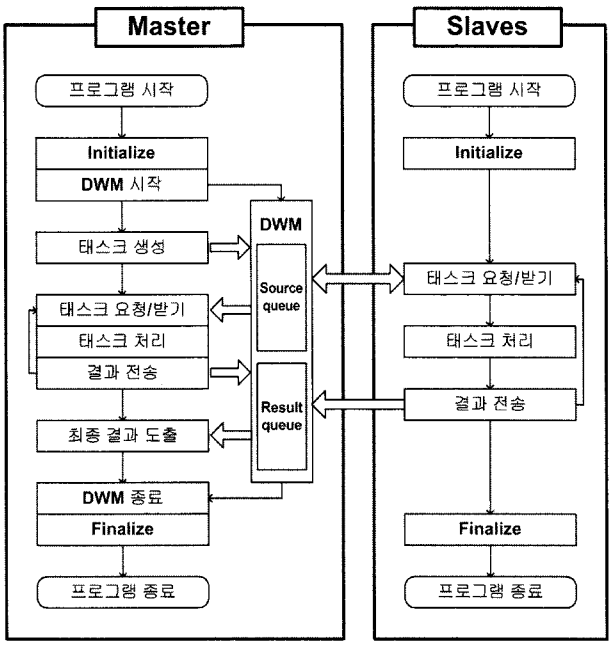


그림 1. DWM을 이용한 병렬처리 프로그램의 흐름도
Fig. 1. Program flow for parallel processing with DWM.

저장한다. DWM은 slave 컴퓨터들에게 태스크 분배를 시작하며, 수행이 끝난 태스크는 결과를 master 컴퓨터에게 다시 전송한다. DWM은 slave들로부터 전송되어오는 결과들을 큐에 저장하며 master 컴퓨터에서 최종 결과를 도출하기 위한 루틴을 수행한다. 최종 결과가 도출되면 DWM과 병렬처리를 수행하기 위한 기본적인 환경을 종료하고 프로그램이 종료된다. DWM의 초기화 과정에서 master 와 slave 컴퓨터 각각은 태스크를 분배하고 관리하는 쓰레드를 수행시킨다. 이 쓰레드는 개별 컴퓨터에 할당된 태스크의 종료와 더불어 대기 중인 태스크의 존재 여부를 확인하는 절차를 수행하며 필요에 따라 새로운 태스크를 할당받도록 한다. 때문에 이기종 컴퓨터들로 구성된 클러스터 시스템에서 컴퓨터의 성능이 좋을수록 더 많은 태스크를 수행하게 되어 전체적으로 클러스터 시스템 성능의 효율적 활용이 가능하다.

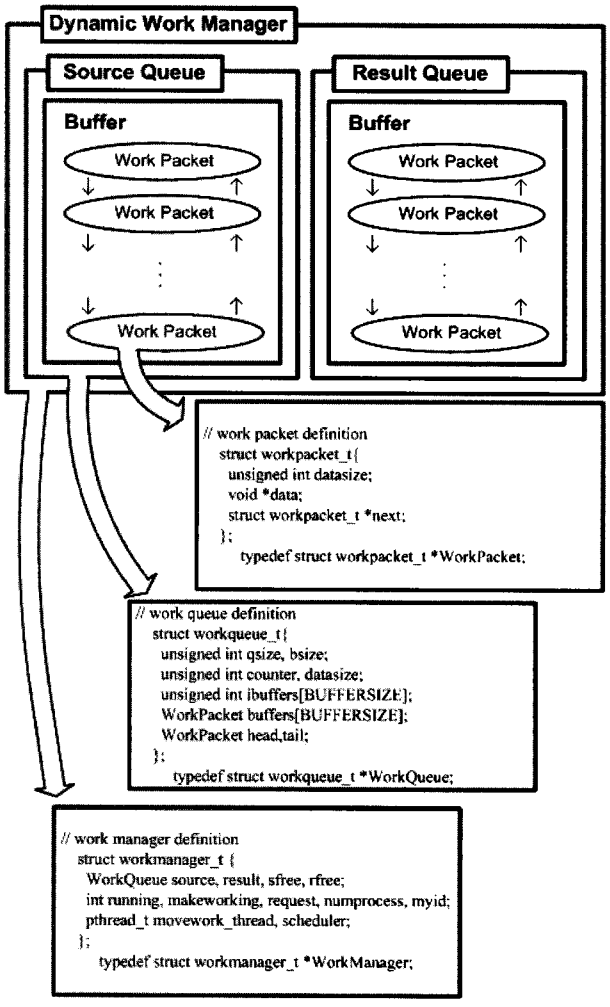


그림 2. DWM의 내부구조
Fig. 2. The structure of DWM.

그림 2는 DWM의 내부 구조와 관련 구조체의 정의를 보여준다. WorkManager는 source queue와 result queue 두 개의 WorkQueue가 linked-list 형태로 구성되고, 각 WorkQueue는 생성되는 태스크의 수에 따라 내부에 WorkPacket들을 가진다. WorkPacket은 데이터를 저장하는 공간으로 데이터 크기를 나타내는 datasize 변수와 실제 데이터가 저장된 주소를 가지는 인스턴스 변수인 data 등으로 구성된다. WorkPacket에 따라 메

표 1. DWM 함수 원형
Table 1. DWM function prototype.

```
// WorkManager descriptor 생성 및 해제
WorkManager consWorkManager
    (unsigned int srcdatasize, unsigned int rstdatasize);
void destroyWorkManager (WorkManager wm);
// WorkManager 생성 및 종료
void runWorkManager (WorkManager wm);
void stopWorkManager (WorkManager wm);
// 태스크를 동적으로 관리하는 관리자 본체
void *master_scheduler (void *arg);
// Source 큐 및 Result 큐에 태스크 저장
void addWorkSource (WorkManager wm, void *data);
void addWorkResult (WorkManager wm, void *data);
// Source 큐 및 Result 큐로부터 태스크를 획득
WorkPacket getWorkSource (WorkManager wm);
WorkPacket getWorkResult (WorkManager wm);

// WorkQueue descriptor 생성 및 해제
WorkQueue consWorkQueue (unsigned int datasize, int type);
void destroyWorkQueue (WorkQueue wq, int type);
// WorkQueue 내부 buffer 생성 및 해제
void consBuffers (WorkQueue wq);
void destroyBuffers (WorkQueue wq);
// WorkQueue에 저장된 WorkPacket의 수를 반환
unsigned int getWorkQueueLength (WorkQueue wq);
// WorkQueue에 data 저장
void addWorkPacket (WorkQueue wq, void *data);
// WorkQueue에서 태스크를 획득
WorkPacket getWorkPacket (WorkQueue wq);
// WorkQueue의 버퍼에서 WorkPacket을 이동
void moveWorkPacketFromBuffers (WorkQueue wq);

// WorkPacket descriptor 생성 및 해제
WorkPacket consWorkPacket (unsigned int datasize);
void destroyWorkPacket (WorkPacket wp);
// WorkPacket의 크기 반환 (data 크기 포함)
unsigned int getWorkPacketSize (unsigned int datasize);
// WorkPacket의 크기 반환 (data 크기 배제)
unsigned int getWorkPacketHeaderSize (void);
```

모리 공간이 할당 혹은 해제되기 때문에 상대적으로 많은 시간을 소비하게 된다. 이렇게 소비되는 시간을 최소화하기 위해 WorkQueue에는 미리 생성된 WorkPacket들인 Buffer가 존재한다. source queue에 저장된 각각의 WorkPacket은 DWM 내부에 구현된 쓰레드에 의해서 각 컴퓨터들로 분배되며 도출된 결과값은 master 컴퓨터로 전송되어 result queue에 저장된다. 표 1 은 WorkManager, WorkPacket, WorkQueue를 위한 각종 함수들의 원형을 보여준다.

Ⅲ. 실 험

1. 벤치마크 프로그램 및 실험 환경

본 논문에서 사용되는 벤치마크는 다양한 형태의 계산 및 통신 방식을 가지는 커널 수준의 프로그램들이다. 각 벤치마크에 대한 설명을 표 2 에 종합하였다. Fibonacci(FIB), Traveling Salesman Problem(TSP), n-Queens(NQ)의 경우 재귀호출 알고리즘을 이용하여 생성되는 태스크의 수를 쉽게 조절할 수 있도록 하였다. Matrix Multiplication (MM)의 경우 행렬을 나누어 계산을 수행하며 행렬 분배 방식에 따라 생성되는 태스크의 수를 조절하도록 하였다. 재귀호출의 깊이와 행렬의 크기는 클러스터 환경에서 작업을 수행하기 위해 충분한 수의 태스크를 생성할 수 있도록 설정하였다.

MPI 라이브러리를 이용한 클러스터 환경으로서, 생산년도와 성능이 다른 6대의 x86 기반의 개인용 컴퓨터로 이기종 클러스터 시스템을 구축하였다. 클러스터를 구성하는 각 컴퓨터는 최소한 128Mbytes의 메모리를 가지도록 설정했으며, 모든 컴퓨터는 100Mbps로 동작하는 Nortel BayStack 70-24T 이더넷 스위치로 연결하였다. 표 3에 클러스터 시스템을 구성하는 개별 컴퓨터의 사양을 정리하였다. 각 컴퓨터의 상대적 성능은 ant1 컴퓨터의 성능을 기준으로 측정하여 정리한 것이다. 그 밖에 각 컴퓨터의 프로세서 동작 주파수와 메모리 크기, 운영체제의 버전을 정리하였다.

표 2. 테스트 벤치마크 프로그램
Table 2. Test benchmark program.

	크기	설 명
FIB	n=43	피보나치 수열을 계산하는 알고리즘 $F(n)=F(n-1)+F(n-2)$
TSP	n=13	n개의 도시를 중복없이 모두 방문하는데 소요되는 비용을 산출하는 알고리즘
NQ	n=14 (n*n)	n×n 체스판 위에 n개의 queen을 안전하게 놓을 수 있는 방법을 탐색하는 알고리즘
MM	n=1100 (n*n)	두 정방행렬의 곱을 계산하는 알고리즘

표 3. 이기종 클러스터 시스템
Table 3. Heterogeneous cluster system.

이름	클럭주파수 (MHz)	메모리 (Mbytes)	상대적 성능	운영체제 (Linux)
ant1	400(P-Ⅱ)	128	1.0	RedHat v7.3
ant2	400(P-Ⅱ)	128	1.0	RedHat v7.3
ant3	450(P-Ⅲ)	128	1.08	RedHat v7.3
ant4	800(P-Ⅲ)	128	2.07	RedHat v9.0
ant5	800(P-Ⅲ)	128	1.97	RedHat v7.3
ant6	2400(P-Ⅳ)	512	6.42	RedHat v9.0

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

2. 성능 평가 및 실험 방법

실험의 성능 평가 기준으로 두 가지를 활용한다. 첫 번째는 프로그램의 실행 시간이다. 클러스터 환경에서 병렬처리 프로그램의 성능을 비교하기 위한 기준으로서의 실행 시간은 다양한 변수로 인해 잘못된 비교를 야기할 수 있다. 본 논문에서는 동일한 벤치마크 프로그램에 대하여 태스크 수에 따른 실행 시간의 변화와 클러스터를 구성하는 컴퓨터의 수에 따른 실행 시간의 변화를 고려하였다. 실행 시간 측정은 프로그램의 코어 부분과 태스크의 생성, 데이터 교환을 위한 부분만을 고려하며 병렬처리 환경 및 데이터 초기화에 필요한 시간 등은 포함하지 않았다. 공정한 성능 비교를 위해서 실험 진행시 영향을 미칠 수 있는 요소를 최대한 제거하여 실험이 진행되는 동안에는 다른 프로그램이 수행되지 않도록 하였으며, 동일한 실험을 다섯 번 반복하여 그 결과들의 평균을 취하였다.

두 번째는 프로그램의 복잡도이다. DWM과 WPM으로 작성된 병렬처리 프로그램의 복잡도를 정량적으로 비교하기 위해 Non-Commented Lines of Code (NCLOC)와 Cyclomatic Complexity (CC)를 사용한다[5~6]. 프로그램의 복잡도는 가장 간단하게 프로그램의

총 라인 수를 고려할 수 있으나, 이는 프로그래머의 프로그래밍 스타일에 따라 차이를 보일 수 있으므로, 빈 라인, 주석 라인, 변수 선언 라인을 제외한 라인 수를 의미하는 NCLOC를 사용한다. 주어진 두 가지 형태의 프로그램에서 더 작은 NCLOC를 가지는 프로그램이 더 간단한 해법이며 운영 및 유지가 쉽다. NCLOC는 프로그램 복잡도의 일면을 보이지만 프로그램의 내용에 대한 반영은 제한적이다. CC는 프로그램의 흐름에서 선형의 독립적인 패스의 수를 계산함으로써 프로그래밍 언어의 형식에 독립적인 알고리즘 수준의 복잡도를 반영함으로써 NCLOC를 보완한다. CC는 프로그램 흐름도를 이용하여 계산결과를 하나의 숫자 (CC = E - N + p, E는 그래프의 엣지 수, N은 그래프의 노드 수, p는 연결된 컴포넌트의 수)로 나타낸다. 주어진 두 가지 형태의 프로그램에서 더 작은 CC를 가지는 프로그램이 더 이해하기 쉽고 간결하다.

3. 실험 결과 및 분석

가. 벤치마크 프로그램의 실행 시간 비교

그림 2 는 DWM을 적용한 네 가지 벤치마크 프로그램에 대하여 클러스터 시스템을 구성하는 컴퓨터 수에 따른 실행 시간을, 그림 3 은 생성되는 태스크 수에 따른 실행 시간을 측정한 결과이다. 그래프의 X축은 클러스터 시스템을 구성하는 컴퓨터의 수 혹은 태스크의 수를, 왼쪽의 주 Y축은 초 단위의 실행 시간을, 오른쪽의 보조 Y축은 컴퓨터 그리고 태스크가 증가함에 따라 얻을 수 있는 Speed Up 정도를 나타낸다. 비교 대상으로서 이상적인 성능으로 기대되는 WPM을 적용한 벤치마크 프로그램의 결과를 함께 도시하였다.

전체적으로 DWM의 성능 패적이 WPM의 성능 패적을 충실히 따르고 있는 사실로부터 DWM이 병렬처리를 수행하는데 관여하는 컴퓨터의 수는 물론 생성되는 태스크의 수와 무관하게 이기종 클러스터 시스템의 성능을 효율적으로 활용하고 있음을 확인할 수 있다. 더불어, DWM을 이용한 경우, 해당 클러스터 시스템에서 이상적인 성능으로 기대되는 WPM을 적용한 경우 보다 조금 낮은 성능을 보여주고 있으나 그 차이는 매우 미미하다는 것을 알 수 있다. DWM을 적용한 MPI 병렬처리 프로그램에서는 DWM이 모든 태스크의 관리 및 교환을 담당한다. 동적인 태스크 분배와 더불어 계산과 메시지 교환의 중첩을 위해 추가된 쓰레드 루틴으

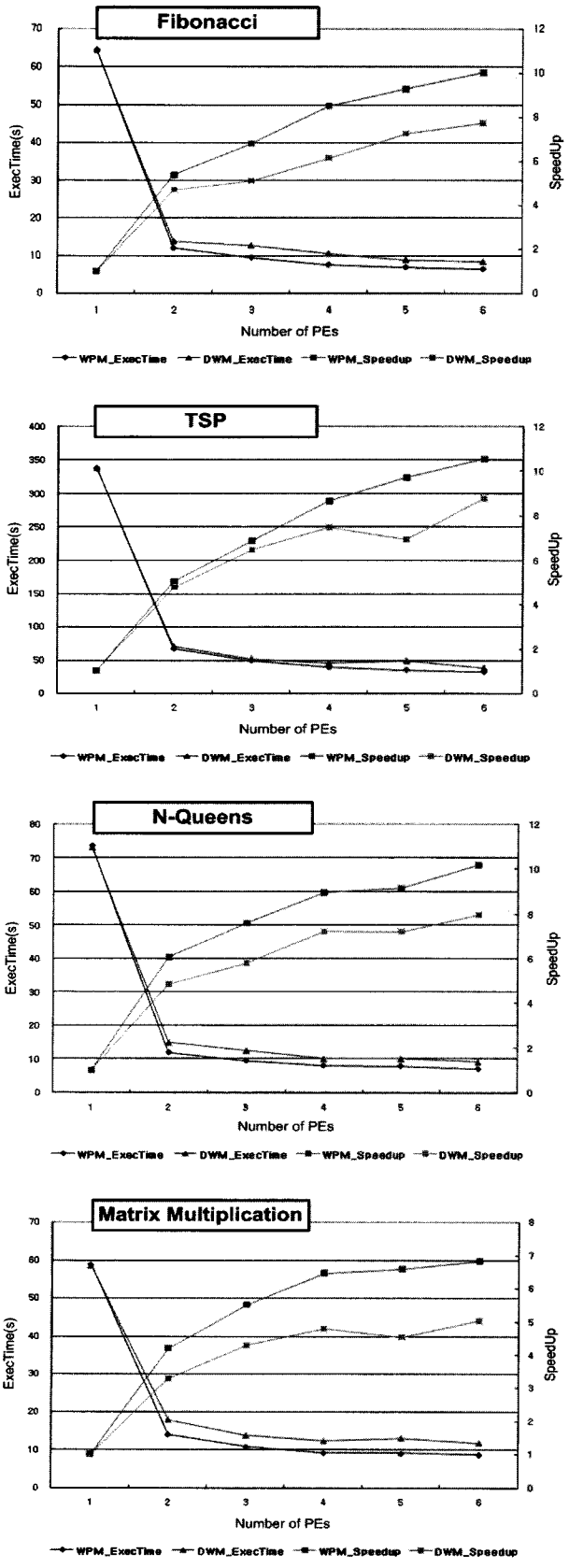


그림 3. DWM의 컴퓨터 수에 따른 성능
Fig. 3. Performance of DWM on number of processors.

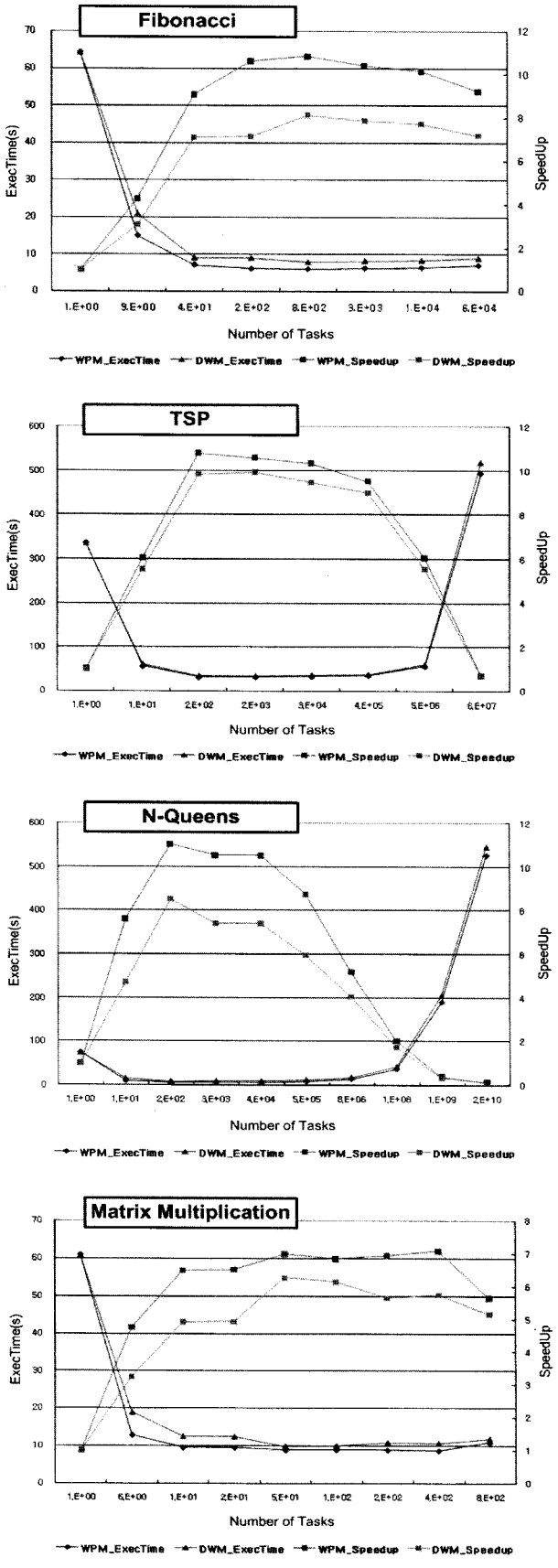


그림 4. DWM의 태스크 수에 따른 성능
Fig. 4. Performance of DWM on number of tasks.

로 인해 DWM은 WPM에 비해 부가적인 오버헤드가 존재하게 된다.

나. 벤치마크 프로그램의 복잡도 비교

그림 4는 WPM과 DWM을 이용한 MPI 벤치마크 프로그램들의 복잡도를 NCLOC와 CC를 사용하여 나타낸 평가 결과이다. 또한 상대적인 비교를 위해서 WPM의 복잡도를 기준으로한 DWM의 복잡도를 각각의 막대그래프 상단에 수치로 나타내었다. 앞서 언급했듯이 NCLOC, CC 모두 작을수록 프로그램 이해도가 높다.

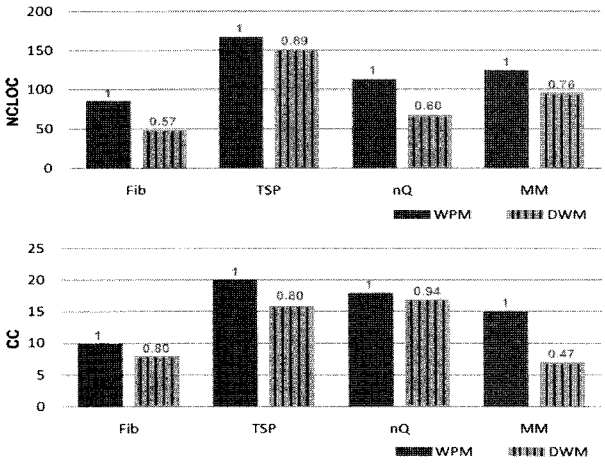


그림 5. 프로그램 복잡도 평가
Fig. 5. Programming complexity.

프로그램의 복잡도를 비교하면, WPM에 비해 DWM을 사용함으로써 NCLOC의 경우 11~43%의 복잡도 감소 효과가, CC의 경우 6~53%의 복잡도 감소 효과를 볼 수 있다. 이는 병렬처리 프로그래밍 및 코드의 유지관리에 있어 실질적인 수고의 감소가 눈에 뜨일 정도로 크게 작용할 것으로 기대된다. 응용 프로그램에서는 태스크의 생성과 계산을 위한 부분에만 집중하면 되기 때문에 간단한 프로그래밍이 가능하게 된다. 결론적으로 DWM을 라이브러리화 함으로써 이기종 클러스터 시스템의 성능을 효율적으로 활용하는 프로그램을 보다 손쉽게 작성할 수 있게 된다.

Ⅳ. 결 론

클러스터 시스템은 널리 사용되고 있는 개인용 컴퓨터와 네트워크 장비로 쉽게 구축될 수 있다. 기술의 급속한 발전으로 개인용 컴퓨터의 성능이 지속적으로 증

가하고, 클러스터 시스템을 구성하는 컴퓨터들을 추가 또는 교체함으로써 전체 클러스터 시스템의 이기종화를 야기했다. 이기종화된 클러스터 시스템에서 수행할 프로그램들은 클러스터 시스템의 성능을 최대한 활용하기 위한 태스크 관리가 필요하다. 본 논문에서는 클러스터 환경에서 병렬처리 시스템 구축에 가장 많이 사용되고 있는 MPI를 위한 동적 자원 관리자인 DWM (Dynamic Work Manager)을 라이브러리 형태로 제안하고 구현하였다. 간단한 벤치마크 프로그램들을 이용한 실험 결과로 부터 DWM의 사용은 이기종 클러스터 시스템의 성능을 효율적으로 활용하는 동시에 MPI 병렬처리 프로그램의 작성을 용이하게 함을 알 수 있다.

참 고 문 헌

[1] Sinr, M : MPI -The complete reference, MIT Press, 1996.

[2] Buyya, R : High Performance Cluster Computing - Architecture and Systems, Prentice Hall PTH, 1999.

[3] Baek, S., Lee, K., Kim, J., Morriss, J. : Heterogeneous Network of Workstations, Lecture Note on Computing Science, Vol. 3189, Springer -Verilag, 426-439, 2004.

[4] Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H., Zhou, Y. : An efficient multithreaded runtime system, PPoPP'95, Santa Barbara, 1995.

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

[6] Grady, R.: Successfully Applying Software Metrics, Computer, Vol. 27, 18-26, 1994.

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

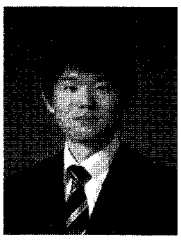
[9] Aversa, R., Mazzocca, N., Villano, U.: A case study of application analytical modeling in heterogeneous computing environment, The Journal of Supercomputing, Vol. 24(1), 5-24, 2003.

[10] Khokhar, A. A., Prasanna, V. K., Shaaban, M. E., Wang, C. L.: Heterogeneous Computing: Challenges and Opportunities, Computer, 18-27, June 1993.

[11] Kim, J., Lilja, D. J.: Performance-Based Path Determination for Interprocessor Communication in Distributed Computing Systems, IEEE Transactions on Parallel and Distributed Systems, 316-327, March 1999.

[12] 이규호, 김준성, 이기종 시스템으로 구성된 클러스터 시스템을 위한 MPI Work Packet Manager, IEEK, Dec 2005.

저 자 소 개



박 종 현(학생회원)
2008년 중앙대학교 전자전기 공학부 학사 졸업
2008년~현재 동대학원 석사과정
<주관심분야 : 컴퓨터구조, 병렬처리시스템>

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 : 컴퓨터구조, 병렬처리시스템, 시스템성능 분석, 임베디드 시스템 설계>