□기술해설□

PVM/MPI를 이용한 병렬 프로그래밍

연세대학교 김신덕* · 양성봉* · 변혜란* · 맹혜선*

목 차

1. 서 론

컴퓨터를 응용하는 여러 분야에서 다양하고 복잡한 문제들을 신속하고 동시에 처리하기 위한 병렬 수행 알고리즘과 병렬 처리 시스템 구조 분야에서의 연구 성과들에 힘입어, 병렬 분산 처리 기술은 대규모 계산이 요구되는 문제들의 효과적인 해결책으로 제시되고 있다 [1]. 특히 최근에는 이러한 병렬 처리 기술의 응용 분야가 점차로 보편화 그리고 대규모화되고 있으며, 이와 비례하여 복잡도도 증가하고 있다. 이러한 응용 문제들을 신속하고 효과적으로 처리하기 위해서는 다양한 종류의 컴퓨터 시스템이 요구되어진다. 기존의 동종 머신 (homogeneous machine) 환경 하에서는 이러한 요구를 충족시킬 수 없기 때문에 다양한 기종의 시스템으로 구성되는 새로운 환경이 필요하게 되었다. 즉, 프로그램을 구성하고 있는 서로 다른 기능을 가지는 구성 요소들은 상호간의 통신 요구와 자신만의 고유한 요구 조건을 가지게 되는데, 각 구성 요소의 요구 조건은 매우 다양하기 때문에 이를 단일화된 환경에서 프로그래밍하여 단일 머신에서 수행시킨다는 것은 무척 어렵고 비효율적인 방법이 되므로, 보다 발전된 소프트웨어 개발 환경이나 프로그램 구성 기술 (program construction technique)에 대한 연구가 요구되게 되었다.

복잡한 소프트웨어 시스템을 요구하는 예로는 BF3D라고 불리는 유체 역학 응용 문제를 들 수 있는데 이 문제는 높은 계산 능력과 많은 양의 메모리를 요구하며, 새롭게 생성되는 상당한 양의 처리 결과를 2 차원, 3 차원 그래픽 터미날을 통해 표시해야 한다 [2]. 이때 필요한 알고리즘은 일련의 편미분 방정식을 3 차원 공간에서 풀이하도록 하는 알고리즘이다. 이와 같은 응용 분야에서 필요로 하는 모든 기능을 효율적으로 지원하는 시스템은 존재하지 않으며 또한 존재한다고 하더라도 비용 면에서 실용적이지 않을 것이다. 다만 이 알고리즘은 특성상 병렬 통할 때 병렬 머신들의 몇몇 범주에서 동

*비회원

시에 수행이 가능하다. 따라서 일반적인 멀티프로세서가 지원하기 어려운 높은 대역폭의 외부 I/O 처리 장치와 고성능 그래픽스의 처리를 위한 계산, 결과 처리, 그리고 그래픽 출력 요소를 분리하여 선택적으로 활용하는 것이 가장 자연스러운 해결 방법으로 제안될 수 있다.

복잡한 문제 내의 부 알고리즘들이 각각 서로 다른 수행 요구 사항들을 가지게 되는 또다른 응용 분야의 예로서 지구 환경 시뮬레이션을 들 수 있다 [3]. 이 문제는 다양한 환경 인자들이 오염 물질의 집중과 분산에 어떤 요소로 작용되는가를 분석, 연구하는 것이다. 이 경우에는 유체의 흐름 분석을 위한 벡터 프로세싱, 오염물질의 전이를 모델링하기 위한 분산 멀티프로세싱, 온도의 효과에 대한 시뮬레이션을 위한 고속 스칼라 계산, 그리고 사용자 인터페이스를 위한 실시간 그레픽처리 등의 조건이 시뮬레이션의 최적의 환경으로 요구되며, 단일 시스템 환경에서는 이러한 모든 조건들이 제공되어질 수 없게 된다.

여기서 주목해야 할 것은, 우리는 이미 이러한 대규모 응용 문제를 풀기 위한 다양한 하드웨어들을 보유하고 있으며, 이를 이용하는 다중 동시 계산 모델을 지원할 수 있다는 점이다. 예를 들어 그래픽 워크스테이션, 고성능 스칼라 엔진, 그리고 때때로 멀티프로세서나 벡터 프로세서를 연결하고 있는 고속 지역 연결망은 이미 상용화되어 있으며, 결코 예외적인 요소가 아니다. 따라서, 컴퓨터 망으로 연결된 혼합 기종 컴퓨터들 (heterogeneous computers)을 선택적으로 혼합하여 사용하기 위한 방법들이 제안되었으며, 이러한 방법은 매우 효과적이며 성능 대 비용도 매우 효율적이다. 다만, 이러한 다양한 자원들을 구동시켜 실제로 이용하기 위해서는, 서로 다른 계산 모델 (computational model)과 서로 다른 시스템 구조들을 통합하고 재조정할 노력과 비용이 필요하며, 이들 전체가 모두 프로그래머에 의해 처리되어야 하는 문제가 아직 남아있다.

PVM (Parallel Virtual Machine)의 시작은, 간단하고 효율적인 대규모의 병렬 처리 프로그램을 개발할 수 있도록 하는 단일화된 환경을 제공하려는 시도에서 출발했으며, PVM은 망으로 연결된 혼합 기종 컴퓨터들의 집합을 동시 계산 자원으로 다룰 수 있는 단일화된 시각을 제시한다. PVM에서는 사용자의 정의에 따라서 순차처리 컴퓨터, 병렬 처리 컴퓨터, 벡터 프로세서, 그리고 심지어 MPP (Massively Parallel Processors) 시스템들의 집합을 하나의 대규모 분산 메모리 컴퓨터로 여겨질 수 있도록 환경을 제공하므로써 프로그래머는 부가적인 부담없이 프로그래밍에 전념할 수 있게 된다.

따라서 PVM을 이용한 프로그램 기법과 그 환경에 대하여 살펴보는 것은 매우 유용한 일이라고 할 수 있으므로 본 고에서는 먼저 연결망 상에서의 혼합 기종 컴퓨팅에 대하여 개괄적으로 살펴본 후, PVM 시스템의 전반적인 특징, 프로그래밍, 사용자 인터페이스, 응용분야 그리고 PVM 사용을 도와주는 여러 소프트웨어 패키지에 대해 살펴보도록 한다. 최근 PVM 외에도 메시지 전달 시스템들이 속속 개발됨에 따라 이에 대한 표준을 제정하려는 움직임이 시작되었고, 많은 관련 연구자들이 모여 MPI (Message Passing Interface)라는 규약을 만들었다. 본 고의 마지막 부분에서 MPI 제정에 대한 배경과 MPI 규약을 준수하고 있는 여러 시스템들에 대해서도 간략히 소개하도록 한다.

2. 혼합 기종 컴퓨팅 및 PVM 시스템

PVM은 망으로 연결된 혼합 기종들 사이에서 수행되는 병렬 컴퓨팅을 효율적으로 지원하기 위해 연구 개발되었다. 따라서 이번 장에서는 PVM 시스템을 살펴보기에 앞서 우선 혼합 기종 컴퓨팅 이란 무엇인지를 살펴보려 한다. 그리고 PVM이 개발된 배경에서부터 시스템 전반에 걸친 특징을 살펴보겠다.

2.1 망으로 연결된 혼합 기종 컴퓨팅

망으로 연결된 혼합 기종 컴퓨팅 (heterogeneous computing)은 다양한 구조를 가지는 여러 컴퓨터 시스템들을 한 종류 이상의 망으로 연결시킨 하드웨어 플랫폼에서 출발한다. 이러한 환경에서 응용 프로그램은 각각의 알고리즘에 적합한 프로그래밍 모델, 구현 언어, 요구 사

원 등을 이용하여 수행시킬 수 있으며 이를 통해서 수행 성능의 향상을 이루고자 한다[17]. 특별한 경우로서 동일한 워크스테이션들이 하나의 지역망으로 연결된 클러스터 (cluster) 환경을 포함할 수도 있다. 혼합 기종 컴퓨팅 환경이 그림 1에 보여진다.

위와 같은 환경에서의 컴퓨팅은 구체적으로 다음과 같은 장점을 가질 수 있다. 첫째로 기존에 존재하는 하드웨어를 사용하기 때문에 하드웨어에 소요되는 비용이 감소된다. 둘째로 개별적인 태스크들을 각각의 수행 특성에 따라 가장 적합한 컴퓨터 시스템에 할당시키는 것이 가능하므로 전체 소프트웨어 시스템의 최적화가 가능하다. 세째로 현재 구성된 하드웨어 플랫폼에 새로운 컴퓨터 시스템을 추가시킬 수 있으므로 필요한 성능의 부분 증대가 가능하다. 네째로 프로그램을 개발할 때에 프로그래머는 자신이 사용하던 컴퓨터 시스템의 프로그래밍 환경을 그대로 이용할 수 있다. 다섯째로 혼합 기종 컴퓨팅 환경을 이루게 되는 개별적인 컴퓨터들과 워크스테이션들은 이미 그 안전성이 입증되어 있으며 사용 면에서도 기반이 되는 전문성이 이미 이루어져 있다. 마지막으로 사용자 레벨 또는 프로그램 레벨의 오류 보정 (fault tolerance)이 응용 프로그램 또는 운영체제에서 적은 비용으로 구현이 가능하다 [4]. 이러한 특징을 가지는 혼합 기종 컴퓨팅은 프로그램의 개발 노력, 디버깅 시간, 자원 확보의 어려움 그리고 운영 비용을 전반적으로 감소시킨다. 따라서 응용 프로그램이 보다 효율적으로 수행될 수 있도록 한다.

그러나 이러한 모든 장점이 실제로 유효하기 위해서는 부가적으로 상당한 노력이 필요하게 된다. 즉 혼합 기종 컴퓨팅에서는 MPP 시스템에서와 같이 모든 프로세서가 수행 능력, 자원, 소프트웨어 그리고 통신 속도 면에서 꼭 같은 성능을 보이는 것이 아니므로 다음과 같은 부분에서 발생하는 특성상의 차이를 조정해 주어야 한다. 첫째로 구조상의 차이가 있다. 혼합 기종 컴퓨팅 환경 내에서 가용한 컴퓨터들은 IBM PC 호환 기종, 고성능 워크스테이션, 공유 메모리 멀티프로세서, 벡터 슈퍼 컴퓨터 그

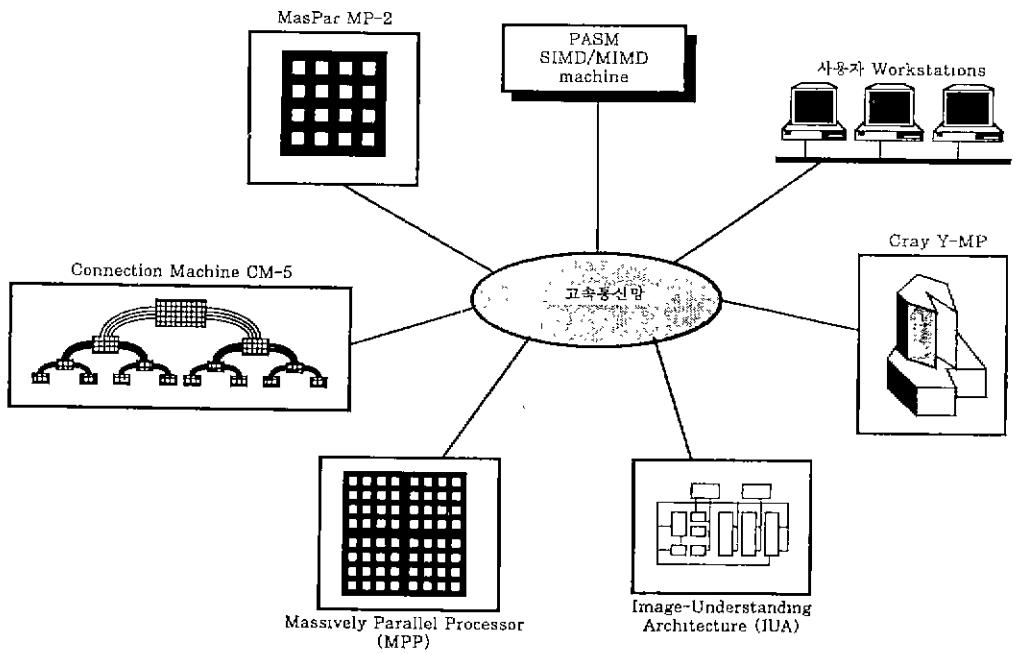


그림 1 혼합 기종 컴퓨팅 환경

리고 대규모 MPP에 이르기까지 그 구성이 다양하다. 각각의 컴퓨터들은 자신만의 이진 코드 형식을 가지며 이들은 종종 다른 컴퓨터들 사이에서 호환이 불가능할 수 있다. 둘째로 데이타 형식의 차이가 있다. 혼합 기종 컴퓨터들의 데이타 형식은 일반적으로 흔히 호환이 되지 않기 때문에 한 컴퓨터에서 다른 컴퓨터로 보내진 데이타가 해독 불가능한 경우가 발생할 수 있다. 따라서 혼합 기종 환경을 위해 개발된 메시지 전달 방식은 모든 컴퓨터가 교환된 데이타를 이해할 수 있도록 지원해야 한다. 이외에도 각각의 프로세서들의 수행 속도 차이로 인하여 로드 불균형의 문제가 생길 수 있으며 여러 컴퓨터들에게 어떤 방법에 의해 작업을 할당하는 가도 문제가 될 수 있다. 또한 여러 망들이 전송 속도를 달리하므로써 생기는 차이도 무시할 수 없는 문제가 된다. 따라서 혼합 기종 컴퓨팅이 제공하는 여러 장점을 사용하기 위해서는 위에서 제시한 문제들과 아울러 그 밖의 문제들이 해결되어야 한다 [4].

PVM은 이러한 혼합 기종 컴퓨팅 사용상의 제약을 해결할 수 있도록 여러 요소를 갖추어 개발되었다. 따라서 다음 절에서는 PVM 시스템에 대하여 좀 더 자세히 살펴보기로 한다.

2.2 PVM 시스템

이번 장에서는 PVM 시스템의 전반적인 특징에 대하여 알아보려 한다. 우선 PVM이란 무

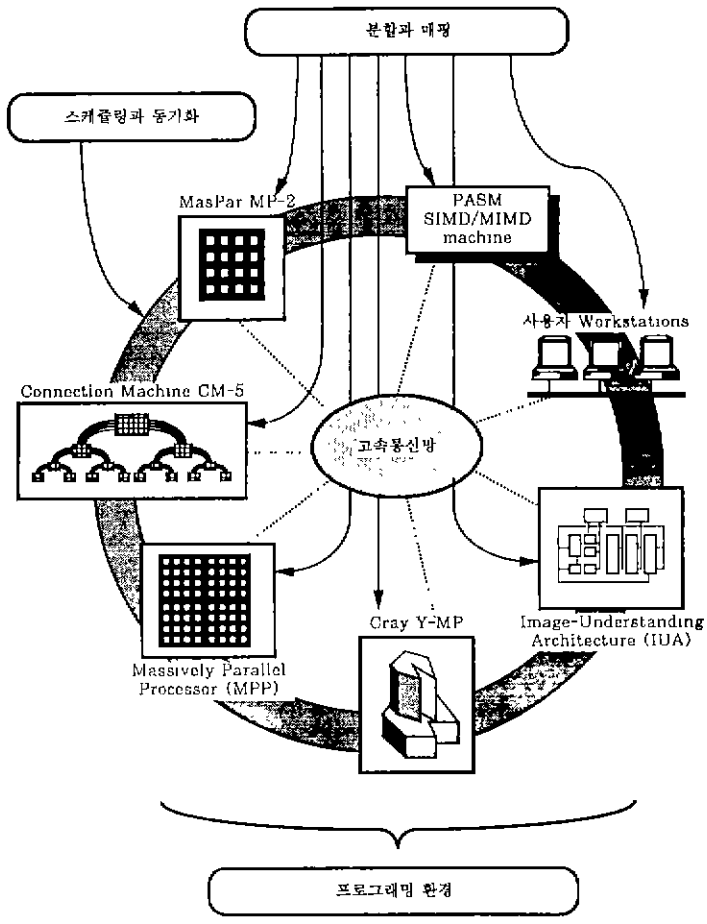


그림 2 PVM 시스템

엇인가를 간단히 살펴보고 그 후에 시스템 전반에 걸친 특징들을 살펴보겠다.

PVM은 망으로 연결된 혼합 기종 (UNIX 컴퓨터)들을 사용자가 하나의 분산 메모리 멀티프로세서로 사용하는 환경을 제공하기 위해 개발되었다. PVM시스템이 제공하는 요소들이 그림 2에 보여진다. 각각의 컴퓨터들은 공유 메모리 또는 지역적인 메모리만을 가지는 멀티프로세서, 벡터 슈퍼 컴퓨터, 그래픽 기능이 강화된 워크스테이션, 또는 일반적인 워크스테이션 등이 모두 가용하며 이들은 여러 종류의 망 즉 ethernet, FDDI 등으로 연결되어 있다. PVM은 사용자에게 자신이 구성한 컴퓨터들의 특정 집합 위에서 프로그램을 수행시킬 수 있도록 한다. 사용자 프로그램은 C나 C++ 또는 Fortran 언어를 사용할 수 있으며 PVM 라이브러리의 함수 (function)를 이용하여 PVM 시스템들과 통신할 수 있다. 여기서 제시한 각각의 특징은 다음 장에서 자세히 설명하기로 한다.

PVM이 개발된 역사를 간단히 살펴보면 이 프로젝트는 1989년 여름 Oak Ridge 국립 연구소에서 시작되었다. 실험 버전인 PVM 1.0은 Vaidy Sunderam과 Al Geist에 의해 만들어졌는데 외부에 발표되지는 않았다. PVM 2.0은 Tennessee 대학에서 만들어져서 1991년 3월에 발표되었으며 그후 많은 과학 계산용 응용 분야에서 사용되었다. 사용자들에 의한 제안 사항들과 이에 대한 개선을 통해 새롭게 구성된 버전 3.0은 1993년 2월 발표되었다. PVM 소프트웨어는 자유롭게 배포될 수 있으며 현재 전 세계에서 계산용 응용 분야에서 사용되고 있다. PVM에 관한 정보를 얻기 위해서는 표 1의 PVM 사이트를 참조할 수 있다.

2.2.1 PVM의 주요 특징

이번 절에서는 위에서 제시한 PVM의 주요 특징들을 자세히 살펴보기로 한다. 따라서 PVM 시스템이 기반을 두고 있는 원칙들을 하나씩 알아보도록 하겠다.

첫째로 PVM은 프로그램을 수행시킬 컴퓨터들의 집합을 사용자가 정의할 수 있도록 설계되었다. 사용자는 가용한 여러 컴퓨터들 중에서 자신의 프로그램 수행에 적합한 컴퓨터들을 선택하여 이들을 호스트로 설정하고 이 집합으로 가상 머신 (Virtual Machine)을 구성한다. 이렇게 정의된 가상 머신은 사용자가 선택한 컴퓨터들의 기능을 모두 포함한 기능을 가진 하나의 대규모 컴퓨터로 여겨지게 되며 각각의 기능을 이용하여 프로그래밍과 수행이 이루어진다. 또한 가상 머신을 이루는 호스트들은 동적으로 추가 삭제될 수 있는데 프로그램 수행 도중에 호스트에 오류가 발생했을 때 해당 호스트를 삭제하고 대치될 호스트를 추가하는 오류 보정 (fault tolerance)에 중요한 도구가 된다. 둘째로 PVM은 사용자가 하드웨어 환경을 반투명한 상태로 접근할 수 있도록 한다. 즉 사용자는 하드웨어 환경을 여러 컴퓨터들의 기능의 합으로 간주하고 이용할 뿐이며 기종간의 차이로 인한 문제들은 PVM상에서 해결되도록 한다. 따라서 사용자는 혼합 기종 컴퓨팅에서 프로그래머에게 부과되었던 부가적인 부담 없이 혼합 기종들을 이용할 수 있다. 세째로 병렬성의 단위는 태스크로 설정하였다. 따라서 각각의 태스크는 통신의 단위 주체가 되며 메시지 송수신 시에는 태스크의 ID를 통해 송수신지가 구별되게 된다. 넷째로 태스크들 사이의 통신은 메시지 전달 방식을 통해서 이루어지도록 한다. 특히 이 특징은 PVM이 기존의 메시지 전달 방식을 완전하게 제공하므로써 메시지 전달 방식에 익숙한 프로그래머들에게 널리 선택되는 이유가 된다. 그러나 PVM에서는 통신을

표 1 인터넷 상의 PVM 사이트

ftp netlib2.cs.utk.edu
URL http : //www.epm.ornl.gov/pvm/pvm_home.html : PVM Web home page
URL http : netlib.org/pvm3/index.html : PVM3의 라이브러리 인덱스
URL http : //www.eece.ksu.edu/pvm3/pvm3.html : The New PVM Web Page

위한 공유 메모리 개념을 지원하지 않기 때문에 병렬 프로그래밍에 익숙하지 않은 사용자들에게는 여전히 사용에 걸림돌이 되기도 한다. 다섯째로 여러 레벨에 따라 다양한 복합성 (heterogeneous)을 지원하도록 설계되었다. PVM 시스템에서 이용할 수 있는 복합성의 레벨은 세 가지인데 응용 프로그램, 머신, 그리고 망 레벨이 있다. 응용 프로그램 레벨에서는 태스크 특성에 따라 특정 호스트를 지정할 수 있다. 머신 레벨에서는 사용자가 특정 구조를 지정할 수 있다. 망 레벨에서는 서로 다른 종류의 망들이 하나의 가상 머신을 이루는 것이 가능하도록 한다.

마지막으로 PVM이 널리 사용될 수 있는 가장 큰 특징 중의 하나인 이식성 (portability)이 있다. PVM의 소스 코드는 간단한 세팅에 의하여 UNIX 기종간에 쉽게 이식될 수 있는데 PVM 버젼 3부터는 비 UNIX 계열 머신과 멀티프로세서 시스템으로도 이식이 가능하게 되었다. 표 2에서 현재 PVM이 수행될 수 있는 환경을 보여주고 있다. 그 외에도 PVM 패키지는 크기가 작으며(소스 코드가 약 1Mb 정도) 인스톨하기가 쉽다. 또한 인스톨하기 위한 특별한 권한이 없기 때문에 계정을 가진 어떤 사용자도 인스톨이 가능하다.

표 2 PVM이 수행 가능한 환경

Single-processor	Multi-processor
80386/80486 with BSDI	Alliant FX/8
DEC Alpha/OSF−1	BBN TC2000
DEC Microvax	Convex
DEC station	Cray YMP and C90
DG Avion	IBM 3090
HP 9000/300	Intel Paragon
HP 9000/700	Intel iPSC/2
IBM RS/6000	Intel iPSC/860
IBM/RT	Kendah Square KSR−1
NeXT	Sequent Symmetry
Silicon Graphics IRIS	Stardent Titan
Sun 3	Thinking Machines CM−2
Sun 4, Sparc	Thinking Machines CM−5

2.2.2 PVM 시스템의 구성

이번 절에서는 PVM 시스템을 이루고 있는 구성 요소를 살펴보기로 한다. PVM 시스템은 크게 두 가지 요소에 의해 구성되는데, 각각의 호스트에 상주하며 인터페이스 역할을 하는 PVM daemon과 이 PVM daemon을 이용하기 위해 필요한 인터페이스 라이브러리 루틴이 있으며 각각에 대하여 살펴보도록 하자.

pvmd3 이라고 불리는 PVM daemon은 가상 머신을 구성하는 모든 컴퓨터 위에서 상주한다. pvmd3는 유효한 계정의 사용자면 누구나 그 머신에서 이 daemon을 수행시킬 수 있도록 설계되었다. 사용자가 PVM을 이용하여 응용 프로그램을 수행시키기 위해서는 우선 가상 머신이 구성되어야 하는데 이 가상 머신은 사용할 모든 호스트에 daemon이 수행되었을 때 구성이 완료된다. 이렇게 모든 호스트에 daemon이 수행되면 이후로 PVM이 제공하는 모든 기능은 daemon을 통해서 각각의 태스크에 전달된다. 또한 태스크가 PVM 기능을 요청할 시에도 daemon을 통해서 이루어지게 된다. PVM 시스템에서 daemon은 가상 머신을 구성하는 호스트에 관한 정보와 자신의 호스트에서 수행중인 태스크에 대한 정보를 관리한다. daemon은 호스트에 관한 정보를 담은 화일이 존재하는 어느 특정 호스트에서 수행되어지면 그 후로 가상 머신 내의 호스트들은 이 daemon에 의해서 수행이 자동으로 이루어지게 된다. 그 후에 PVM 응용 프로그램은 가상 머신 내의 어떤 호스트의 UNIX 프롬프트에서도 수행이 가능하다.

다음으로는 응용 프로그램 상에서 PVM이 제공하는 기능들을 이용하기 위해 필요한 PVM 인터페이스 루틴 라이브러리가 있다. PVM 인터페이스 루틴의 라이브러리는 응용 프로그램의 태스크들 사이에서 협동 작업을 수행하는데 필요한 기본적 기능들을 함수의 형태로 제공한다. 이 라이브러리는 메시지 전달을 위한 루틴, 프로세스를 생성시키기 위한 루틴, 태스크들을 협동하도록 하기 위한 루틴, 그리고 가상 머신을 수정하기 위한 루틴 등을 제공한다. 인터페이스 루틴 라이브러리는 4장에서 다시 자세히 설명하도록 한다.

2.3 PVM을 이용한 응용 분야

PVM은 세계적으로 여러 연구기관에서 분산 과학 계산을 위한 연구에서 채택, 사용되고 있다. 따라서 과학, 산업, 그리고 의학 관련 분야의 많은 응용 프로그램들이, 망으로 연결된 워크스테이션 클러스터링 상에서 PVM을 이용하여 개발되고 있다. PVM과 기타 클러스터링 컴퓨팅 시스템을 사용하게 되는 가장 중요한 동기는 가격대 성능비이다. 일반적으로 클러스터들은 몇몇 종류의 응용분야에서 같은 정도의 수행 능력을 가지는 슈퍼컴퓨터에 비하여 비용면에서 10 배정도 효과적이다. PVM의 사용이 증가하는 또다른 이유는 여러 시스템에 이식시키기가 쉽고 명확하며, 확고한 인터페이스를 제공하기 때문이며 이러한 점이 과학 응용 분야의 개발에 널리 사용되는 요인이 된다. 과거 몇년 동안 많은 응용 프로그램이 PVM을 이용하여 개발되었다. 다음은 응용 프로그램들의 리스트이다.

- Material science
- Global climate modeling
- Atmospheric, oceanic, and space studies
- Meterorological forcasting
- 3-D groundwater modeling
- Weather modeling
- Superconductivity, molecular dynamics
- Monte Carlo CFD application
- 2-D and 3-D seismic imaging
- 3-D underground flow fields
- Particle simulation
- Distributed AVS flow visualization

3. PVM을 이용한 프로그래밍

본 장에서는 PVM을 이용하여 프로그래밍을 하고자 할 때 기본적으로 알아야하는 사항들을 살펴보고자 한다. PVM 시스템은 현재 C, C++, 그리고 Fortran언어를 지원하는데 C와 C++언어에서 PVM 사용자 인터페이스 라이브러리는 여러 개의 함수들로 구성된다. 이 함수들은 C 언어의 함수들에 의해 사용되는 일반적인 규정을 따르고 있으며 이와 함께 매크로 정의가 시스템 상수를 위해서 사용된다. C 나 C++언어를 이용하여 PVM 프로그램을 작성할 때에는 응용 프로그램과 libpvm3. a 라이브러리를 링크시켜서 PVM 라이브러리 함수에 접근한다. Fortran 언어에서는 함수가 아닌 서브루틴을 이용하며 C 언어의 라이브러리 함수와 Fortran의 서브루틴 사이에는 1:1 대응이 가능하다. PVM을 이용한 프로그래밍을 설명하기 전에 기본적으로 사용되는 용어를 살펴보면 다음과 같다.

- 호스트 (host) : 실제 머신으로 예를 들면 UNIX 워크스테이션
- 가상 머신 (virtual machine) : 사용자의 정의에 의해 하나 이상의 머신으로 구성된 메타 머신 (meta machine)
- 프로세스 (process) : 프로그램, 데이타, 스택 등, 예를 들면 UNIX 프로세스, 노드, 프로그램 테스크 (task) : 계산의 기본 단위
- TID : 각각의 태스크에 할당되는 유일한 식별자
- PVMD : PVM daemon
- 메시지 (message) : 태스크들 사이에 송수신되는 데이타의 리스트
- 그룹 (group) : 태스크들의 리스트에 할당되는 이름

PVM을 이용하여 작성된 프로그램은 일반적으로 다음과 같은 형태를 가진다. 사용자는 하나 또는 그 이상의 순차적인 프로그램을 C, C++, 또는 Fortran 77로 작성한다. 이 프로그램에는 PVM 라이브러리에 대한 호출이 추가되어 있어야 하는데, 이때 각각의 프로그램은 응용 프로그램을 구성하는 태스크에 해당된다. 프로그램들은 호스트 집합 내의 특정 머신에서 컴파일되고, 호스트 집합 내의 머신들로부터 접근 가능한 위치에 놓인다. 사용자는 응용 프로그램을 수행시키기 위해서 하나의 태스크를 직접 수행시키고 (보통 "master", "initiating task"), 이 프로세스가 순차적으로 다른 PVM 태스크를 수행시킨다. 각각의 태스크들은 지역적으로 수행되어지고 메시지를 서로 교환하면서 문제를 해결한다.

그림 3은 PVM 프로그램의 예 "hello"이다.

이 기본적인 프로그램은 PVM 프로그래밍의 기본 개념을 보여준다. 이 프로그램은 자신의 태스크 id를 프린트한 뒤에 다른 프로그램 "hello_other"를 pvm_spawn() 함수를 이용하여 시작시킨다. 성공적으로 수행 시작이 끝난 뒤에는 pvm_recv() 함수를 이용하여 다른 프로그램에서 형성한 메시지를 전달받는다. 그림 4에서는 slave 프로그램의 리스트를 보여준다. 첫번째 수행되는 PVM 동작은" master"의 태스크 id를 얻어내기 위한 pvm_parent 호출하고 그후 이 프로그램은 자신의 호스트 이름을 얻어내고 이를 master에게 3 단계의 동작을 통해 전송한다. pvm_initsend()는 전송 버퍼를 초기화한다. 그리고 pvm_send는 숫자 1로 태그한 메시지를 ptid에 의해 명시된 목표지점으로 전송한다.

```
#include "pvm3.h"
main()
{
        int cc, tid, msgtag;
        char buf[100];

        printf("i'm t%x\n", pvm_mytid());
        cc=pvm_spawn("hello_other", (char**)0,0,"",1,&tid);
        if (cc == 1)
        {
                msgtag = 1 ;
                pvm_recv(tid, msgtag);
                pvm_upkstr(buf);
                printf("from t%x: %s\n", tid, buf);
        }
        else
                printf("can't start hello_other\n");
        pvm_exit();
}
```

그림 3 프로그램 hello.c

```
# include "pvm3.h"
main()
{
int ptid, msgtag;
char buf[100];
ptid = pvm_parent();
strcpy(buf, "hello, world from ");
gethostname(buf + strlen(buf), 64 );
        msgtag = 1 ;
pvm_initsend(PvmDataDefalut);
pvm_pkstr(buf);
pvm_send(ptid, msgtag)l
pvm_exit();
}
```

그림 4 프로그램 hello_other.c

4. 사용자 인터페이스

PVM은 사용자가 PVM 시스템을 용이하게 사용할 수 있도록 다양한 인터페이스 루틴을 제공한다. PVM이 제공하는 인터페이스는 사용자가 호출할 수 있는 함수나 서브루틴의 형식을 가진다. C 언어의 함수와 Fortran언어의 서브루틴 사이에는 1 : 1 대응의 관계가 있으므로 본 장의 앞절에서는 C 언어를 사용할 때 필요한 인터페이스만을 살펴보겠다. 본 장의 마지막 절에서는 사용자가 보다 용이하게 PVM을 사용할 수 있도록 도와주는 시각화 도구에 대하여 알아보기로 한다.

4.1 프로세스 제어

```
int tid = pvm_mytid (void)
call pvmfmytid (tid)
```

pvm_mytid()는 프로세스의 TID를 반환한다. 이 함수가 첫번 째로 불려진 PVM 함수라면 이 때 해당 프로세스는 PVM에 등록된다. 이 함수 외에도 모든 PVM 시스템 호출은 태스크를 PVM에 등록시킬 수 있다.

```
int tid = pvm_exit (void)
call pvmfexit (info)
```

pvm_exit() 는 지역적인 pvm daemon에게 이 프로세스가 PVM 시스템에서 빠져나갈 것임을 알린다. 이 함수는 프로세스의 동작을 중지시키는 것이 아니라 다른 UNIX 프로세스와 같이 수행을 계속할 수 있도록 한다. 그러나 일반적으로 사용자는 자신의 프로그램을 중지시키기 바로 이전에 pvm_exit() 함수를 호출한다.

```
int numt = pvm_spawn (char *task, char
**argv,int flag, char *where, int ntask,
int *tids)
call pvmfspawn (task, flag, where, ntask,
tids, numt)
```

pvm_spawn()은 실행 파일인 "task"를 ntask

개수만큼 가상 머신에서 수행시킨다. argv는 task에 보내질 인수들의 배열에 대한 포인터를 지칭한다. flag는 선택사항들을 지칭하기 위해 사용한다. 채택된 선택사항을 나타내는 값들의 합을 가진다.

값	선택	의미
0	PvmTaskDefault	어느 곳에서 프로세스를 수행시킬 지를 PVM이 임의로 결정한다.
1	PvmTaskHost	where 인수가 특정한 호스트를 지칭한다.
2	PvmTaskArch	where 인수는 PVM_ARCH를 나타낸다.
4	PvmTaskDebug	태스크를 디버거를 작동시킨 상황에서 수행시킨다.
8	PvmTaskTrace	트레이스 데이타를 생성시킨다.
16	PvmMppFront	MPP 환경에서 태스크를 수행시킨다.
32	PvmHostCompl	where의 호스트를 제외한 호스트 집합을 사용한다.

반환값으로 numt는 성공적으로 수행이 시작된 프로세스들의 개수를 나타내며 에러 발생시에는 에러 코드를 의미한다.

4.2 수행 정보

```
int tid = pvm_parent (void)
call pvmfparent (tid)
```

pvm_parent는 해당 프로세서를 처음으로 수행시킨 프로세서의 TID를 반환한다. 만약 pvm_spawn()에 의해 수행이 시작된 것이 아니라면 PvmNoParent를 반환한다.

```
int dtid = pvm_tidtohost (int tid)
call pvmftidtohost (tid, dtid)
```

pvm_tidtohost()는 TID 와 같은 호스트에서 수행중인 daemon의 TID인 dtid를 반환한다. 이 함수는 해당 태스크가 어떤 호스트에서 수행중인지를 판별하는데 유용하다. 호스트의 문자이름과 같은 가상 머신에 대한 보다 일반적인 정보는 아래의 함수들을 이용하여 얻어질 수 있다.

```
int info = pvm_config (int
*nhost, int *narch, struct pvmhostfinfo
**hostp)
call pvmfconfig (nhost, narch, dtid, name,
arch, speed, info)
```

pvm_config()는 가상 머신에 대한 정보를 반환한다. nhost는 호스트의 개수, narch는 서로 다른 데이타 형식의 수, hostp는 사용자가 선언한 pvmhostinfo 구조의 배열에 대한 포인터이다.

이 외에도 가상 머신에서 수행중인 태스크에 대한 정보를 반환하는 pvm_tasks() 함수 등이 있다.

4.3 가상 머신의 동적인 구성 및 신호 표시

```
int info = pvm_addhosts (char **hosts,
int nhost, int *infos)
int info = pvm_delhosts (char **hosts, int
nhost, int *infos)
call pvmfaddhost (host, info)
call pvmfdelhost (host, info)
```

위의 C 함수는 복수의 호스트들을 그리고 Fortran 프로시져는 개별적인 호스트를 각각 가상머신에 추가하거나 삭제시킬 수 있다. 구성된 가상머신으로는 문제를 풀기가 어렵다고 결정되었을 때 호스트를 첨가하므로써 응용이 가능한 컴퓨팅 파워를 증가시키도록 한다.

```
int info = pvm_sendsig (int tid, int signum)
call pvmfsendsig (tid, signum, info)
int info = pvm_notify (int what, int
msgtag, int cnt, int tids)
call pvmfnotify (what, msgtag, cnt, tids, info)
```

pvm_sendsig()는 signum신호를 또다른 TID에 의해 지칭되는 또다른 PVM 태스크로

보낸다. pvm_notify는 PVM이 특정한 상황 발생시 호출자에게 신호를 보낼 것을 요청한다. 선택사항은 아래와 같다.

PvmHostExit－태스크가 종료시
PvmHostDelete－호스트가 삭제시
PvmHostAdd－호스트가 추가시

4.4 메시지 전달

메시지 전달시 PVM 에서는 세가지 단계가 필요하다. 제일 먼저 pvm_initsend() 또는 pvm_mkbuf()에 의해 보내질 버퍼를 초기화 시킨다. 다음으로 보내질 메시지는 반드시 버퍼로 "packing"되어야 한다. 이러한 "packing"은 여러 번의 pvm_pk*()함수의 조합으로 이루어진다. 마지막으로 완전한 메시지는 pvm_send()를 호출함으로 다른 프로세서로 전달되거나 pvm_mcast() 호출로 다중 전송될 수 있고 메시지는 블럭 되거나 블럭되지 않은 receive 루틴을 호출하여 받을 수 있다. 그리고 나서 받은 버퍼로부터 팩킹된 데이타의 각각을 unpacking 시킨다. receive 루틴은 어떤 메시지나 받도록 셋팅될 수 있다. 또는 특별한 출처로 부터의 메시지나 또는 특별한 태그를 가지는 메시지를 받도록 또는 특별한 출처로부터 특별한 태그를 가지는 메시지를 받도록 할 수 있고, 또한 메시지가 도착하였는지의 여부만을 알려주는 함수도 있다.

4.4.1 메시지 버퍼

```
int bufid = pvm_initsend (int encoding)
call pvmfinitsend (encoding, bufid)
```

사용자가 하나의 송신 버퍼를 사용하고자 할 때에는 pvm_initsend() 만이 버퍼 루틴을 위해 필요하다. 이 루틴은 새로운 메시지를 버퍼로 보내기 이전에 불려져야 한다. pvm_initsend는 송신 버퍼를 소거시키고 새로운 메시지를 packing하기 위해 새로운 버퍼를 생성한다. packing을 위한 부호화 방법이 encoding에 의해 정하여 진다. 새로운 버퍼의 식별자가 bufid로 반환된다.

부호화를 위한 선택사항은 아래와 같다.

PvmDataDefault－PVM 은 메시지를 보내기 전에 사용자가 이기종 머신을 사용하는 지의 여부를 알 수 없으므로 XDR 부호화 방법이 기본으로 사용된다. 만약 사용자가 데이타 형식에 대한 특별한 부호화가 필요없다는 것을 알았을 때에는 PvmDataRaw를 사용하여 부호화시키는데 필요한 비용을 줄일 수 있다.

PvmDataRaw－어떤 부호화도 취하여지지 않는다.

PvmDataInPlace－데이타는 packing 비용을 줄일 수 있는 곳에 놓여진다. 버퍼는 단지 전송될 메시지의 크기와 포인터만을 가진다. pvm send()가 호출되면 메시지는 사용자의 메모리 밖으로 직접 복사된다.

이 함수외에도 pvm_mkbuf 는 송신을 위한 빈 버퍼를 생성한다. pvm_freebuf는 현재 존재하는 버퍼를 소멸시킨다.

4.4.2 데이타의 packing 및 unpacking

PVM에서 사용하는 packing 함수들은 데이타 타입들의 배열을 packing하여 버퍼로 보낸다. 이 루틴들은 반복되어 호출되서 데이타들을 하나의 메시지로 구성하도록 할 수 있기 때문에 메시지는 서로 다른 데이타 타입의 여러 배열을 보유할 수 있다. PVM에서는 데이타 타입에 따라 다른 packing 함수를 제공하고 있는데 아래에서는 몇 가지만을 소개하고 있다.

```
int info = pvm_pkbyte (char *cp, int nitem,
int stride)
int info = pvm_pkcplx (float *xp, int
nitem, int stride)
int info = pvm_pkdouble (double *dp, int
nitem, int stride)
int info = pvm_pkfloat (float *fp, int
nitem, int stride)
int info = pvm_pkint (int *np, int
nitem, int stride)
int info = pvm_pkstr (char *cp)
```

모든 루틴들은 첫 번째 패러미터로 packing 될 소자 (item)에 대한 포인터를 가진다. nitem

은 이 배열로부터 packing에 사용될 소자들의 총개수를 뜻하며 stride는 배열에서의 소자들의 간격을 의미한다.

PVM에서 unpacking은 버퍼로 부터 받은 데이타를 unpacking하는데 사용되는데 실제로 사용될 때에는 packing될 때의 데이타 타입과 개수 그리고 배열에서의 간격이 일치해야 한다. 아래에 unpacing 함수의 예가 보여진다. 패러미터의 의미는 packing 때와 같다.

```
int info = pvm_upkbyte (char *cp, int
nitem, int stride)
int info = pvm_upkcplx (float *xp, int
nitem, int stride)
int info = pvm_upkdouble (double *dp, int
nitem, int stride)
int info = pvm_upkfloat (float *fp, int
nitem, int stride)
int info = pvm_upkint (int *np, int
nitem, int stride)
int info = pvm_upkstr (char *cp)
```

4.4.3 데이타의 전송 및 수신

PVM에서 데이타를 전송하는 루틴은 두 종류가 존재한다. 특정 태스크에 메시지를 전송할 때에는 pvm_send() 함수를 사용하며 여러 태스크에게 동시에 메시지를 방송할 때에는 pvm_mscat() 함수를 사용한다.

```
int info = pvm_send (int tid, int msgtag)
int info = pvm_mcast (int *tids, int ntask,
int msgtag)
```

msgtag는 메시지에 덧붙여질 메시지 태그를 의미한다. 이 메시지 태그는 메시지를 받을 때에 특정 메시지를 확인하기 위하여 사용한다. tid는 메시지가 전해질 특정 태스크의 TID를 의미하며 tids는 메시지가 방송될 태스크들의 TID들이 저장되어있는 배열을 가리키는 포인터이다.

PVM은 태스크가 메시지를 수신할 수 있도록 하는 여러 방법을 제공한다. 이 함수들은 메시지 전송함수와 짝을 이루는 것이 아니기 때문에 그 기능에 따라 적당한 것을 선택하면된다. 가장 대표적인 수신함수는 아래와 같다.

```
int bufid = pvm_recv (int tid, int msgtag)
```

위의 함수는 TID 태스크로부터 msgtag와 같은 레이블을 가지는 메시지가 도착할 때까지 블럭킹된다. msgtag나 TID가 -1을 가질 때에는 모든 경우에 매치되는 것으로 여긴다.

4.5. PVM 환경 하에서의 시각화 도구 (HeNCE)

PVM 환경이 널리 보급됨에 따라 병렬 응용 프로그램의 중요성이 더욱 증가되었다. 그러나 병렬 프로그램은 그 작성과 디버깅에 있어서 어려운 점이 많다. 이에 PVM에서 사용되는 도구들－프로그래밍, 수행, 디버깅－을 시각화하려는 시도들이 진행되고 있으며 세 가지 시도는 일반적으로 혼합되어 한 시스템으로 구성된다. PVM 환경 하에서 이용 가능한 시각화 도구에는 HeNCE [5], Xab [16] 등이 있으며, 본 고에서는 이들 중 대표적인 시각화 도구인 HeNCE를 살펴보기로 한다.

HeNCE (Heterogeneous Network Computing Environment)는 Oak Ridge 국립 연구소, Tennessee 대학, Emory 대학에서 개발한, PVM을 위한 시각적 프로그래밍 시스템이다. 이 시스템은 병렬성의 의존 관계를 시각적으로 서술하면 PVM 프로그램을 생성해 낸다. 또한, 대화형 인터페이스를 통해 가상 머신의 구성, 응용 프로그램의 수행, 그리고 동작의 시각화에 걸친 일련의 작동 등을 제공한다. 그림 5에 HeNCE에서 제공하는 인터페이스들이 보여진다.

HeNCE의 주요 특징으로는 사용자가 프로세스간의 종속성을 시각적으로 그려서 이를 수행하도록 한다. HeNCE의 시각 언어 요소들은 그림 6과 같다.

HeNCE에서 프로그램을 개발할 때에는 인터페이스를 통해 아래의 5단계를 수행하게 된다.

1. Compose : 시각 언어 및 소스 코드 편집.
2. Config : 가상기계와 Cost Matrix를 명시.

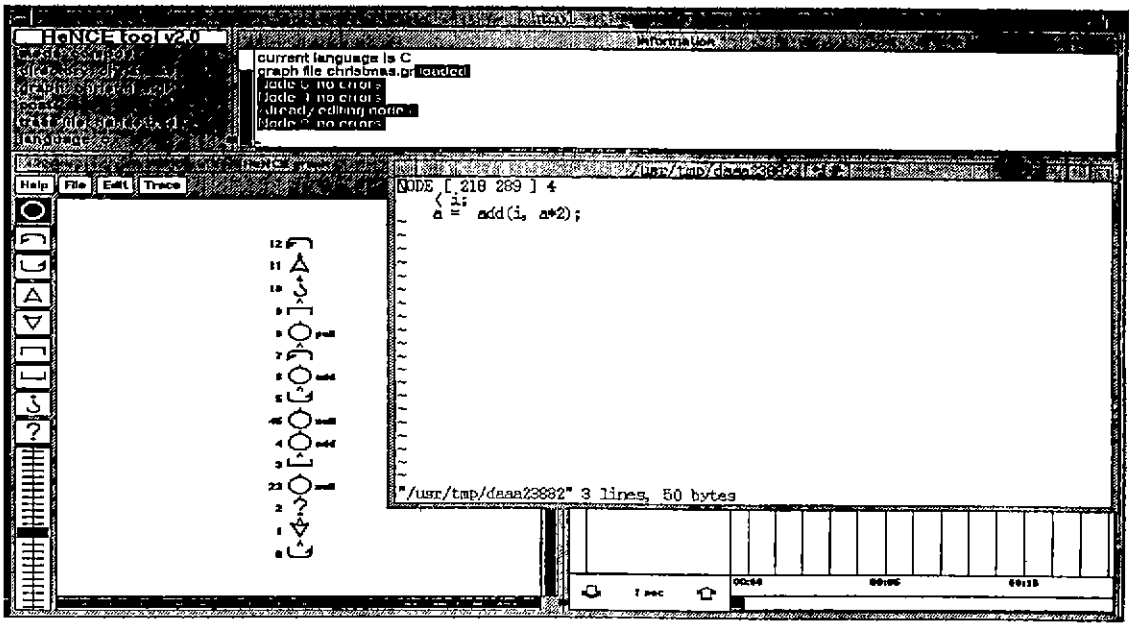


그림 5 HeNCE의 인터페이스

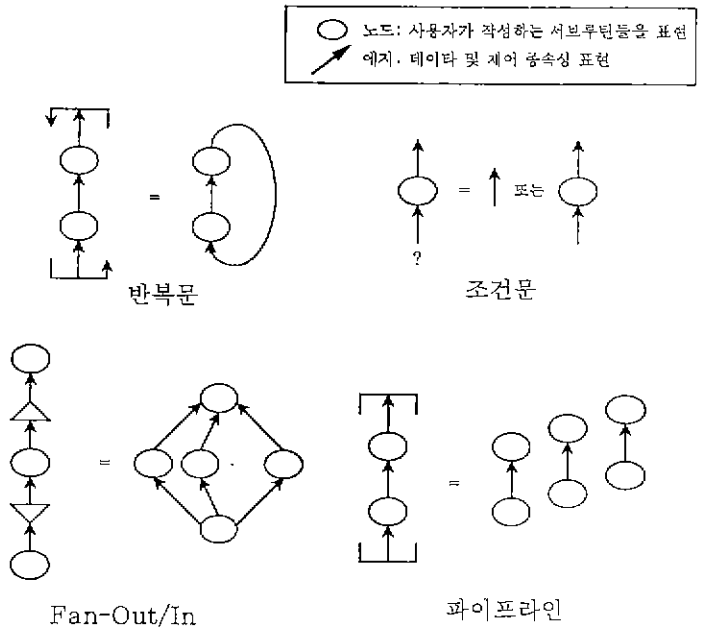


그림 6 HeNCE의 시각 언어 요소

3. Build : 병렬 수행 가능 코드의 컴파일, 생성 및 분배.
4. Exec : 동적 부하 분배(Dynamic Machine Load Balancing)를 통한 프로그램 수행 및 트레이스 데이타 생성.
5. Trace : 시각 언어와 가상 기계 등에서 프

로그램의 수행을 애니메이션으로 표현.

5. MPI

메시지 전달 인터페이스 표준안 (MPI) [15]은 메시지 전달 라이브러리 루틴들의 문법과 의미에 대해 표준안을 제정하려는 노력으로 1994년 4월 완성되었다. MPI를 제정한 목적 중의 하나는 MPP의 제작자들에게 명확하게 정의된 루틴들의 기반 집합을 제공하려는 것이었다.

MPI는 분산 컴퓨팅에 사용될 수 있는 완전하고 독립적인 소프트웨어 제반구조를 의도한 것이 아니기 때문에 프로세스 관리 (태스크의 수행을 시작시킴), 가상 머신의 구성, 그리고 입출력을 위한 지원 등은 포함하지 않는다.

MPI는 기존의 메시지 전달 시스템들 중의 하나를 채택한 것이 아니라, 여러 시스템 각각의 효율적인 측면들을 모아 표준안을 구성한 것이다.

5.1 MPI 표준안의 내용

MPI 표준안은 다음과 같은 가정에서 출발한다.

- 프로세스의 집합은 점대점 메시지 전달을 수행한다.
- 각각의 프로세스에게는 약간의 공유 메모리가 할당되어있다.
- 프로세스는 단일 트레드 (thread)이다.
- 표준안에서는 각각의 실제적인 프로세싱 노드에 어떻게 프로세스가 할당되는가는 다루지 않는다.

MPI 표준안은 다음의 요소들에 대하여 정의하고 있다.

- 기본적인 메시지 전달 루틴
- 프로세스 그룹
- 통신 context
- 버퍼 packing
- 유틸리티

MPI 표준안에 대한 보다 많은 정보는 표 3을 참조하면 얻을 수 있다.

5.2 MPI 구현 시스템

MPI의 중요한 장점 중의 하나가 이 표준안을 구현한 시스템이 많이 존재한다는 것이다. 따라서 사용자는 호환이 가능한 여러 시스템 중에서 하나를 선택하여 사용할 수 있다. 표 4는 MPI를 구현한 시스템들의 종류와 접근할 수 있는 ftp주소를 나타낸다.

6. 결 론

지금까지 살펴본 바와 같이 PVM은 망으로 연결된 혼합 기종 컴퓨팅 환경에서 사용자에게 하나의 가상 분산 메모리 멀티 프로세서 환경

표 3 인터넷 상의 MPI 정보

정보의 종류	인터넷 사이트
MPI 관련 문서	http : //www.mcs.anl.gov/mpi/mpi-report/mpi-report.html
MPI 뉴스 그룹	comp.parallel.mpi
Postscript 문서	netlib@ornl.gov에 "send mpi-report.ps from mpi"라는 메시지를 보낸다.

표 4 MPI를 구현한 시스템들의 ftp 주소

연구기관	ftp 주소	비고
Argonne 국립연구소/ Mississippi 주립 대학	ftp : //info.mcs.anl.gov/pub/mpi	
Edinburgh 병렬 컴퓨터 센터	ftp : //ftp.epcc.ed.ac.uk/pub/chimp/release/chimp.tar.Z	CHIMP 시스템
Mississippi 주립 대학	ftp : ftp.erc.msstate.edu/unify	UNIFY 시스템
Ohio 슈퍼 컴퓨터 센터	ftp : //tbag.osc.edu/pub/lam	LAM 시스템

을 제공해 주는 소프트웨어 라이브러리이다. 날로 복잡해져가는 분산 병렬 처리 소프트웨어 개발을 지원하기 위하여 혼합 기종으로 이루어진 컴퓨팅 자원을 단일화된 관점에서 처리할 수 있도록 필요한 여건들을 제공한다.

PVM은 공개 소프트웨어로서 인터넷 상의 여러 사이트에서 쉽게 구할 수 있으며 또한 여러 기종에 이식이 가능하며 설치가 간단하기 때문에 현재 널리 사용되고 있다. 특히 과학 계산이 요구되는 분야에서 널리 사용되고 있는데 지구 환경 시뮬레이션이나 기상 예측 시스템 등에서 그 성능이 입증되고 있다. 또한 병렬 컴퓨터를 보유하고 있지 않은 학교에서 병렬 프로그래밍을 교육하는데 있어서 좋은 교육용 도구로 쓰일 수도 있다.

이제까지의 PVM 프로그래밍에 있어서의 단점으로는 메시지 전달을 기본으로 하는 분산 메모리 환경만을 제공하기 때문에 병렬 프로그래밍에 익숙하지 않은 사용자가 접근하기가 용이하지 않다는 점이 지적되고 있다. 이에 HeNCE와 같은 시각적 지원도구나 PVM을 이용하는 분산 공유 메모리에 대한 연구가 진행되고 있다. 따라서 앞으로는 사용자에게 보다 편리한 PVM 프로그래밍 환경이 제공될 것으로 기대된다.

참고문헌

[1] K. Hwang, F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, NewYork, 1984.

[2] D. W. Lozier, R. G. Rehm, "Some Performance Comparisons for a Fluid Dynamics Code," *Parallel Computing*, Vol. 11, 1989, pp. 305-320.

[3] H. Narang, G. Flanery, J. Drake, "Design of a Simulation Interface for a Parallel Computing Environment," *Proc. ACM Southern Regional Conference*, April 1990.

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

[5] A. Beguelin, et. al., "HeNCE Users Guide," University of Tennessee *Technical Report*, May 1992.

[6] G. A. Geist and V. S. Sunderam, "The Evolution of the PVM Concurrent Computing System," *Proceedings-26 th IEEE Compcon Symposium*, San Fransisco, February 1993, pp 471-478.

[7] G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM 3 User's Guide and Reference Manual," *Technical Report* TM-12187, Oak Ridge National Laboratory, 1993.

[8] G. A. Geist and V. S. Sunderam, "Network Based Concurrent Computing on the PVM System," *Journal of Concurrency: Practice and Experience*, 4(4), June 1992. pp. 293-311.

[9] V. S. Sunderam, "PVM : A Framework for Parallel Distributed Computing," *Journal of Concurrency: Practice & Experience* Vol. 2 No. 4, Dec 1990.

[10] J. J. Dongarra, G. A. Geist, R. ManChek, V. S. Sunderam, "Integrated PVM Framework Supports Heterogeneous Network Computing," *Computers* in Physics, 7, 2, 1993, pp 166-175.

[11] G. A. Geist, V. S. Sunderam, "Experiences with Network Based Concurrent Computing on the PVM System," *Journal of Concurrency: Practice and Experience*, Vol 4 No. 4, June 1992, pp 293-311.

[12] A. Matrone, P. Schiano, V. Puoto, "LINDA and PVM : A comparison between two environments for parallel programming," *Parallel Computing*, 1993, pp 949-957.

[13] A. Geguelin, J. J. dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam., "A users's guide to PVM parallel virtual machine," *Technical Report* ORLN/TM-11826, Oak Ridge National Laboratory, July 1991.

[14] G. A. Geist, et. al., "PVM : Parallel Virtual Machine A Users's Guide and Tutorial for Networked Parallel Computing," MIT press, 1994.

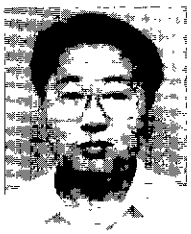
[15] Message Passing Interface Forum, "MPI : A message-passing interface standard,"

Computer Science Dept. *Technical Report* CS-94-230, University of Tennessee, Knoxvile, TN, April 1994.

[16] A. Beguelin, "Xab : A Tool for Monitoring PVM Programs," *Workshop on Heterogeneous Processing*, IEEE Compute Society Press, Los Alamitos, Califonia, April 1993, pp. 92-97.

[17] M. C. Wang, S. D. Kim, M. A. Nichols, R. H. Freund, and H. J. Siegel, "Augmenting the Optimal Selection Theory for Superconcurrency," *Workshop on Heterogeneous Processing : International Parallel Processing Symposium*, March 1992, pp 13-22.

김 신 덕



1982 연세대학교 공과대학 전자공학과(학사)
1987 University of Oklahoma 전기공학(석사)
1991 Purdue University 전기공학(박사)
1993. 2~1994. 2 광운대학교 컴퓨터공학과 조교수
1994. 3~현재 연세대학교 이과대학 컴퓨터과학과 조교수

관심분야 : 병렬처리 시스템, 컴퓨터 구조, heterogeneous computing

양 성 봉

1981 연세대학교 공과대학 요업공학과(학사)
1984 University of Oklahoma 컴퓨터 과학(학부과정)
1986 University of Oklahoma 컴퓨터 과학(석사)
1992 University of Oklahoma 컴퓨터 과학(박사)
1993. 3~1994. 8 전주대학교 전자계산학과 전임강사
1994. 9~현재 연세대학교 이과대학 컴퓨터과학과 조교수

관심분야 : 병렬 알고리즘, genetic 알고리즘, 그래프 알고리즘

변 혜 란

1980 연세대학교 이과대학 수학과(학사)
1983 연세대학교 이과대학 수학과(석사)
1987 University of Illinois 전산학과(석사)
1993 Purdue University 전산학과(박사)
1994. 3~1995. 2 한림대학교 정보공학과 조교수
1995. 3~현재 연세대학교 이과대학 컴퓨터과학과 조교수

관심분야 : 병렬 계산, 신경망, 인공지능

맹 혜 선

1994 연세대학교 이과대학 전산과학과(학사).
1994~현재 연세대학교 대학원 전산과학과 석사과정.

관심분야 : heterogeneous computing, parallel programming