1024 대한기계학회논문집(B) 제21권 제6호, pp. 1024~1033, 1997

〈논 문〉

다중블록 유동해석에서 병렬처리를 위한 시스템의 구조

박 상 근* · 이 건 우**

(1996년 11월 13일 접수)

A Framework for Parallel Processing in Multiblock Flow Computations

Sangkun Park and Kunwoo Lee

Key Words: Parallel Processing(병렬처리), Distributed Networks (분산 네트워크), CFD(전산유체), Master-Slave Model(마스터-슬레이브 모델), PVM(병렬가상기계)

Abstract

The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high performance scientific computing as well as for more general purpose applications. Furthermore with increasing needs to perform the complex flow calculations in an efficient manner, the use of the message passing model on distributed networks has emerged as an important alternative to the expensive supercomputers. This work attempts to provide a generic framework to enable the parallelization of all CFD-related works using the master-slave model. This framework consists of (1) input geometry, (2) domain decomposition, (3) grid generation, (4) flow computations, (5) flow visualization, and (6) output display as the sequential components, but performs computations for (2) to (5) in parallel on the workstation clustering. The flow computations are parallized by having multiple copies of the flow-code to solve a PDE on different spatial regions on different processors, while their flow data are exchanged across the region boundaries, and the solution is time-stepped. The Parallel Virtual Machine(PVM) is used for distributed communication in this work.

1. 병렬계산의 개요

하나의 커다란 문제를 하나의 컴퓨터에서 해결하지 않고, 여러 대의 컴퓨터에서 나누어 동시에 작업함으로써 짧은 시간 내에 문제를 해결하고자 등장한 병렬처리(parallel processing) 분야는 컴퓨터 관련 분야에서 가장 주목받고 있으며, 미래의 과학 분야에 커다란 위치를 차지할 것으로 보고 있다. 과거 수십 년 동안, 고성능, 저가격, 지속적인 생산성 등의 필요성에 의해 성장해 온 이 분야는 과학분야뿐만 아니라 다른 일반적인 목적의 응용분야에서도 그 가능성을 인정받고 널리 사용되고 있다. 이러한 결과를 낳을 수 있었던 것은 다음의 두 가지 주요 발전에 힘입은 탓이다. 즉 massively parallel processors(MPPs)와 distributed computing의 발전이 그것이다.

MPP는 현존하는 가장 강력한 컴퓨터이다. 즉 수백 혹은 수천의 CPU를 가지고 있고 수백 개의 Gigabyte 메모리를 가지고 있다. 이를 바탕으로 MPP는 엄청난 계산 능력을 제공하여, 지구의 기후 모델링(global climate modeling) 등과 같은 거대한 문제(Computational Grand Challenge problems)[1]를 푸는데 대개 사용된다.

*삼성 SDS 정보기술연구소 S/W 응용개발팀
**회원, 서울대학교 기계설계학과

한편, distributed computing은 여러 대의 컴퓨터를 네트워크에 의해 연결해 놓고 하나의 커다란 문제를 연결된 컴퓨터들의 상호협동을 통하여 문제를 해결할 수 있도록 지원해 주는 프로세스로써, 점점 고속도의 네트워크 환경이 구축되면서 MPP에 비해 상당히 저렴한 비용으로 그 막강한 힘을 여러 응용분야에서 발휘하고 있다.

MPP와 distributed computing이 공통적으로 함께 하는 개념은 어떠한 방식으로 메세지를 전달할 것인가이다. 모든 병렬처리 작업 중에는 계산에 참여하는 프로세서들 사이에 데이터 교환이 일어난다. 이러한 데이터 교환을 효율적으로 관리하고 지원하는 몇몇의 제시된 기본방식(paradigm)이 있다. 이에 관한 자세한 내용은 추후 설명하겠다. 제시된 방식 중에 가장 활발히 연구 중에 있고 자주 선택되어지는 것은 메세지 전달 방식(message passing model)이다.

2. 고성능 병렬계산

병렬처리 계산에 의해 고수준의 계산성능을 얻기 위해서는 다음의 세 가지 점에 유의해야 한다.

- 데이터 분배(Data distribution)
- 작업 분배(Work distribution)
- 데이터 교환(Data exchange)

여러 계산에 공통적으로 필요한 데이터(shared data)와 임의의 특정 계산에만 참여하는 데이터(private data)를 구분하여 분류해 놓는다면 데이터에 접근하는 부하를 줄일 수 있고, 또한 각 작업에 필요한 데이터를 미리 분류하여 각 작업에 알맞게 배치해 놓는다면 병렬처리의 성능향상을 꾀할 수 있다. 그리고 커다란 작업을 각 기능별로 분류하여 각각의 부작업들이 독자적으로 그들의 계산능력을 최대한 발휘할 수 있도록 해준다면 병렬처리의 성능향상을 또한 꾀할 수 있다. 한편 병렬처리 계산시에 병렬처리의 실제 수행단위인 각 프로세싱 요소(processing element or task)들 간의 데이터 교환은 자주 등장하고 병렬처리의 성능향상에 커다란 영향을 미친다. 이러한 데이터 교환은 Fig. 1의 세 가지 방식에 의해 구현된다.

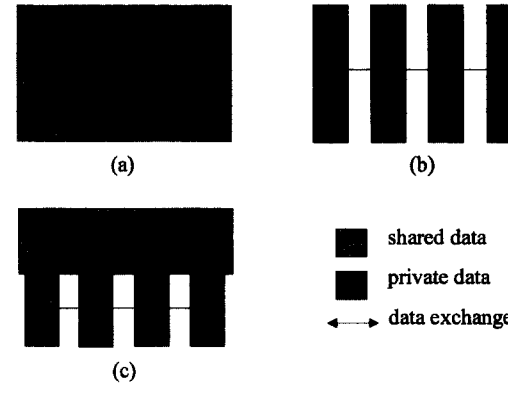


Fig. 1 Schematic diagrams illustrating three data exchange models (a) global addressing (b) message passing (c) explicit shared memory

- global addressing (implicit communication of shared data)

시스템 내부에서 공유데이터 영역을 설정하고 이곳에 모든 정보를 저장하여 각 프로세싱 요소들이 단순히 데이터 주소를 가르킴으로써 원하는 위치에 데이터를 쓰고 읽게 한다.

- message passing (explicit communication of private data)

사용자가 직접 각 프로세싱 요소들 간의 데이터 교환을 설계하여 각 프로세싱 요소들이 원하는 데이터를 주고 받을 수 있게 한다.

- explicit shared memory (low-level explicit communication based on the systems shared memory)

시스템의 공유메모리를 통하여 사용자가 직접 저수준의 데이터 교환을 설계할 수 있게 한다.

이상과 같이 병렬처리의 성능향상을 위해서 사용자는 각 응용목적에 알맞는 데이터 분배와 작업 분배를 설계하고, 위의 세 가지 데이터 교환방식 중에 적당한 하나를 선택하여야 한다. 여기서 알맞은 분배, 적당한 선택이란 작업을 수행하는 최소단위인 프로세싱 요소의 수를 최대화하고, 각 프로세싱 요소 간의 데이터 교환을 최소화하는 분배, 선택을 말한다.

3. 작업부하의 할당

주어진 응용문제를 병렬처리를 통하여 구현하고자 할 때, 사용자 입장에서 가장 고려해야 되는 점은 바로 주어진 전체작업을 빠른 시간 내에 효과적

으로 수행하기 위해서 어떤 기준으로 어떻게 나눌 것인가 그리고 나누어진 작업들을 각각 어느 컴퓨터에서 수행할 것인가 하는 점이다. 즉 작업분할 문제와 작업할당 문제가 실제 병렬처리 프로그램 방식을 선택하기 이전에 결정해야 되는 첫번째 문제이다. 현재 이러한 문제를 지원하기 위해서, 널리 알려진 두 가지 방식이 있는데, 다음과 같다.

- 데이터 분할(data decomposition)
- 기능적 분할(function decomposition)

데이터 분할방식은 주어진 전체 계산작업이 하나 혹은 그 이상의 데이터 구조상에서 수행되는 경우에 유용한 방식으로, 데이터 구조를 분할하고 각 분할된 데이터 구조상에서 계산작업을 수행한다. 그리고 기능적 분할방식이란 전체작업을 각 기능별로 서로 다른 작은 부작업으로 나누어 계산작업을 수행하는 방식으로, 흔히 입력 작업, 프로세싱 작업, 출력 작업 등으로 크게 나누고 각 부작업 안에서 또다시 각 기능별로 작업을 분할하는 방식으로 전체작업을 수행한다. 이 두 방식을 간략히 비교하면, 데이터 분할방식은 같은 테스크가 데이터의 서로 다른 부분에서 계산작업을 수행하는 방식이고, 기능적 분할방식은 기본적으로 서로 다른 테스크가 그들의 고유한 특정작업을 수행하는 방식이다. 이상의 두 분할방식은 개념적으로 서로 독립적이지만 실제 하나의 응용문제에서 두 방식을 모두 구현한다. 즉 순차적인 작업을 기능적 분할방식에 의해 나누고, 각 작업에서의 계산수행은 데이터 분할방식에 의해 이루어진다.

4. 다중블록에 의한 병렬처리 시스템의 구조

4.1 연구 배경

최근들어 병렬처리에 의한 유동장 해석이 전산유체 분야에서 새로운 위치를 자리잡고 있다. 기존의 컴퓨터 환경에서 실행 불가능했던 계산들이 컴퓨터 산업의 발전에 힘입어 서서히 수행되고 있고, 점점 거대한 문제(Grand-challenge problems)를 계산해 보려는 연구가 진행되고 있다. 특히 전산유체 분야의 경우, 응용문제의 형상이 복잡해질수록 그리고 유동상태가 복잡해질수록 더욱더 새로운 컴퓨터 환경을 요구하고 있다. 하나의 프로세서를 가지는 하나의 컴퓨터에서 여러 개의 프로세서를 가지는 하나의 컴퓨터로 그리고 네트워크에 의한 둘 이상의 컴퓨터로, 유동해석을 비롯한 여러 분야에서 그 계산환경은 날로 향상되고 있다. 이러한 현재의 혹은 가까운 미래의 컴퓨터 환경은 기존의 순차적이던 이론 혹은 알고리즘의 변화를 요구하고 있다. 즉 병렬처리의 동시작업을 구현하기 위한 시스템 환경 혹은 프로그래밍 기술 그리고 더 나아가 병렬 알고리즘의 개발 등을 요구하고 있다. 여기서 시스템 환경이란 병렬작업을 지원하는 하드웨어의 사용환경을 말하기도 하지만 주로 네트워크를 통한 이질적 컴퓨터 간의 분산처리 환경을 말한다. 이러한 분산처리 환경을 제공해 주는 몇몇의 소프트웨어가 계속 개발되고 있는데, 가장 널리 알려진 공개된 소프트웨어로서 PVM(Parallel Virtual Machine),[2~4] MPI(Message Passing Interface)[5] 등이 있다. 그리고 병렬처리 프로그래밍 기술이란 PVM 등의 소프트웨어에서 제시하는 일반적인 병렬처리 모델 혹은 방식 등을 말한다. 또한 병렬 알고리즘의 개발이란 특정 응용문제의 특성 및 구조를 파악하여 기존 알고리즘 혹은 새로 개발할려는 알고리즘 등의 병렬처리화를 꾀하는 연구활동을 말한다. 이와같은 새로운 구조의 컴퓨터 환경과 병렬처리를 지원하는 소프트웨어의 개발 그리고 각 분야에서의 계산 알고리즘의 병렬화 혹은 새로운 병렬 알고리즘의 개발 등은 앞으로 미래산업을 선도할 분야로서 여러 분야에서 활발히 연구진행 중에 있다. 특히 전산유체 분야에서의 활동은 오래전부터 진행 중에 있었다.

4.2 시스템의 구조

본 연구에서는 현존의 컴퓨터 환경, 시스템 등을 충분히 활용하여, 복잡한 유동현상을 저렴한 가격으로 병렬처리하여 계산해낼 수 있는 다중블록 유동해석을 위한 병렬처리 시스템 구조를 제시하고자 한다. 제시하는 시스템 구조는 전산유체의 각 분야에서 이미 개발된 기능 혹은 코드 등을 충분히 활용한다면 충분히 구현할 수 있는 구조로서, 새로운 개념을 제시하는 것은 결코 아니다. 단지 전산유체의 모든 과정을 병렬화를 위한 하나의 커다란 작업으로 보고 위에서 설명한 기능적 분할방식과 데이터 분할방식을 함께 적용하여, 전체 작업의 작업효율을 높이고 계산성능을 향상시키려는 하나의 병렬시스템에 관한 예제이다. 그러나 여기서 제시하는

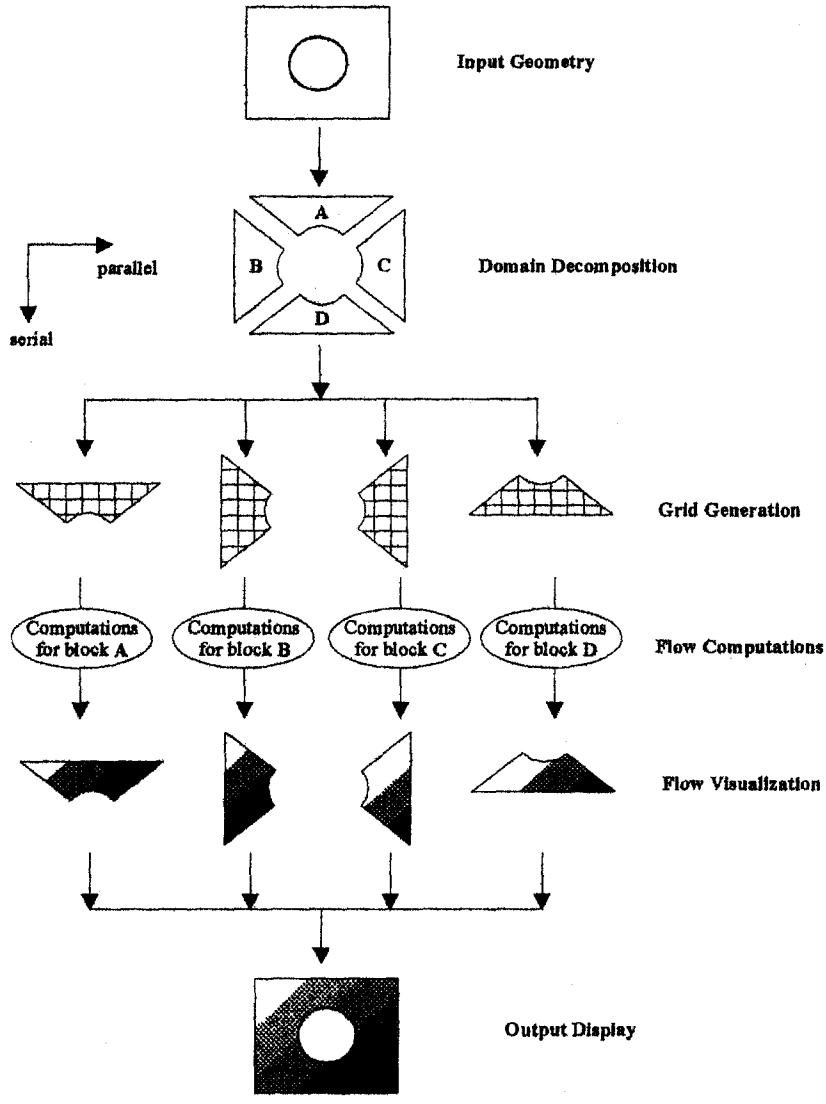


Fig. 2 Global structure of parallel system suggested

시스템의 구조는 새로운 컴퓨터 환경에 대한 투자 없이 사용자가 가지고 있는 컴퓨터 환경을 충분히 활용할 수 있게 설계되었고, 가지고 있는 유동해석 코드의 병렬화를 구현하지 않고도 충분히 유동해석을 병렬처리 할 수 있게 설계되었다. 또한 전산유체에서 일반적으로 수행하는 순차적인 작업순서인 전처리 작업, 실계산 작업, 후처리 작업별로 구조화되어 있어 모듈화의 장점을 가진다. 구체적인 시스템 구조는 Fig. 2와 같다.

먼저 기능적 분할 관점에서 전체작업을 다음과 같이 순차적으로 나눈다.

(1) 형상 입력(Input geometry)
(2) 영역 분할(Domain Decomposition)
(3) 격자 생성(Grid Generation)
(4) 유동 해석(Flow Computations)
(5) 유동 가시화(Flow Visualization)
(6) 결과 출력(Output Display)

위의 순차적 작업들에 대해 병렬화를 구현하기 위해서 가장 필수적인 작업은 영역분할 작업이다. 즉 주어진 전체 물리공간을 몇 개의 부영역으로 나누는 작업이 필요하다. 이 작업을 통하여 궁극적으

로 전체 과정은 병렬처리화 된다. 즉 데이터 분할 관점에서 유동현상이 일어나는 물리영역을 영역분할 작업에 의해 분할함으로써 데이터 분할을 구현한다. 본 연구에서 제시하는 병렬시스템 구조란 결국 영역분할을 통하여 분할된 각 부영역에 대해 각 부영역별로 동시에 격자를 생성하고, 격자생성이 모두 끝나면 또다시 각 부영역별로 동시에 유동해석을 수행하여 그 결과를 가시화하는데 그 목적이 있다. 여기서 가시화 작업은 유동 가시화를 위한 실제 계산작업과 화면상의 그래픽을 위한 그래픽작업으로 나눌 수 있는데, 실제 계산작업은 각 부영역별로 동시에 이루어지며 화면상의 그래픽작업은 계산작업한 결과들을 종합하여 보여준다.

4.3 시스템의 특징

일반적으로 무엇을 병렬화할 것인가는 매우 중요하다. 본 연구의 경우는 바로 유동해석이 수행되는 물리공간을 병렬화를 구현하기 위한 대상으로 삼았다. 전산유체 분야를 포함하여 전산장을 시뮬레이션하는 여러 분야에서 흔히 병렬화를 구현하기 위한 즉 데이터 분할의 기준으로 실계산이 수행되는 대상인 격자 혹은 메쉬를 그 대상으로 삼는다. 계산공간인 실제 물리공간상에서 먼저 격자 혹은 메쉬를 생성하고, 생성된 메쉬에 대해 메쉬의 수가 균등하게 분포되도록 특정 알고리즘에 의해 그룹핑한다. 이렇게 그룹핑 된 몇몇의 메쉬그룹에 대해 계산작업이 나누어 수행된다.[6] 여기서 본 연구방식과 기존방식을 비교해 보면 다음과 같다. 기존의 방식은 격자 혹은 메쉬를 생성한 후에 생성된 메쉬에 대해 영역분할을 수행하여 병렬계산을 위한 작업에 들어간다. 그러나 본 연구에서는 메쉬를 생성하기 전에 먼저 영역분할을 수행한다. 이러한 차이점을 계산 효율 측면과 메모리 한계 측면에서 살펴보면 다음과 같다. 먼저 계산 효율 측면에서, 메쉬 생성 작업이 병렬처리에 의해 이루어지는가를 보면 쉽게 알 수 있다. 기존의 방식은 병렬처리에 들어가기 전에 먼저 메쉬를 생성해야 한다. 그러나 본 연구방식은 메쉬 생성작업 자체를 병렬처리를 통하여 보다 빠르게 구현할 수 있기 때문에 계산속도 측면에서 또한 컴퓨터 활용 측면에서 비교우위에 있게 된다. 한편 메모리 한계 측면에서 보면, 기존 방식은 하나의 컴퓨터 안에서 필요한 모든 메쉬들을 생성하려고 한다. 그러나 그 컴퓨터가 수용할 수 있는 메모리 한계를 초과하여 메쉬생성을 감행한다면 그 컴퓨터는 시스템 에러로 죽게될 것이다. 본 연구방식의 경우에는 생성될 메쉬들이 각각의 컴퓨터 내에서 분산되어 생성되고 저장되기 때문에, 기존연구에 비해 메모리 한계의 초과로 인한 시스템 에러를 막을 수가 있다. 이러한 메모리 문제는 응용문제의 크기가 크면 클수록 매우 중요하게 작용한다. 여기서 응용문제의 크기란 일반적으로, 전산유체 분야의 경우, 유동현상이 일어나는 물리공간의 실제 크기를 말하는 것이 아니고 그 물리공간상에서 생성된 격자 혹은 메쉬의 개수를 의미한다. 이상의 관점에서 격자를 생성한 후, 영역분할을 하는 기존의 방식보다는 영역분할을 수행한 후, 병렬처리에 의한 격자생성을 추구하는 본 연구의 방식이 상대적 우위에 있음을 쉽게 알 수 있다.

결국 본 연구에서 제시한 시스템의 구조는 기능적 분할에 의해 작업의 순서 즉 계산수행의 방향을 결정하고 데이터 분할에 의해 각 작업에서의 병렬처리를 구현한다. 이러한 계산순서의 순차성과 각 계산작업의 병렬성은 오랫동안 인정받은 기존의 안정된 계산구조를 충분히 수용하면서 동시에 병렬처리가 주는 장점을 최대한 실현시키기 위한 적합한 시스템 구조를 제공하게 된다.

4.4 시스템의 환경

이와같은 시스템의 구조를 구현하기 위해서 본 연구에서는 다음과 같은 병렬처리 컴퓨터 환경을 고려하였다. 먼저 하드웨어로서 SGI 워크스테이션을 사용하였고 소프트웨어로서 PVM 시스템에서 제공하는 라이브러리를 사용하였다. 그리고 병렬처리를 위한 하드웨어 구조로서 워크스테이션 여러대를 연결하여 네트워크에 의해 서로의 데이터를 주고 받을 수 있게 하였다. 즉 이질적(heterogeneous) 컴퓨터 환경하에서 네트워크에 기반을 두고 메세지 전달 방식에 의해 데이터 교환을 수행하게

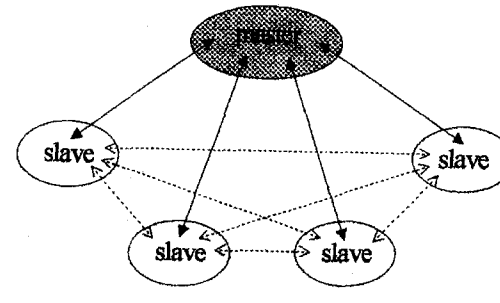


Fig. 3 Master-slave model

하였다. 이러한 구조는 PVM 시스템의 라이브러리를 이용하여 충분히 구현할 수 있다. 이러한 외적인 컴퓨터 환경하에서 병렬처리를 위한 다음의 프로그래밍 기법들을 사용하였다. 앞에서 이미 설명한 것처럼 데이터 분할 및 기능적 분할 관점에서 모든 작업을 순차적으로 병렬화하였고, 각 작업에서의 실제 병렬계산은 Fig. 3의 master-slave model[7]을 선택하였다.

이 모델은 그 구조가 이해하기 쉽고 계산작업의 분할이 계획된 상태라면 비교적 쉽게 구현될 수 있기 때문에, 여러 분야에서 자주 선택되어지는 방식이다. 이와같이 네트워크에 기반을 둔 외적인 컴퓨터 환경과 미리 계획된 작업분할 방식에 의해 선택된 병렬처리 프로그래밍 기법들은 본 연구에서 제시하는 유동해석을 위한 병렬시스템의 구조를 구현하기에 충분한 컴퓨터 환경을 제공하게 된다.

4.5 적용 예제

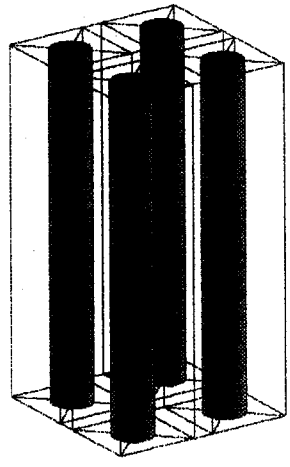
이상의 컴퓨터 환경과 제시한 유동해석 시스템의 구조를 가지고 본 연구에서 실제 구현한 예제를 살펴보면 다음과 같다. 이 예제는 본 연구에서 개발한 다중블록 방식에 의해 영역분할이 수행되고 각 분할된 영역에 대해 격자생성이 이미 끝난 상태 하에서, 유동해석의 병렬처리 과정을 구체적으로 보여주는 예제이다. 본 예제는 튜브뱅크(tube bank) 내부의 유동현상을 해석한다. 튜브뱅크(tube bank) 내부의 유동해석은 열교환기, 보일러, 응축기 등의 설계에 있어서 중요한 요소로서 과거 수십년간 주요 연구대상이 되어왔다. 이 연구는 레이놀즈 수의 변화와 튜브뱅크의 기하학적 형상변화에 의존하여 나타나는 무차원 열전달 계수 및 압력손실 값들을 계산하고 이를 가지고 튜브뱅크의 최적형상 및 최적의 배열위치를 연구한다. 본 연구에서 수행한 튜브뱅크의 해석은 병렬처리에 의한 유동해석의 예를 보여주기 위한 것으로서 일반적인 튜브뱅크의 문제를 단순화시켰다. 즉 실린더 형상을 한 튜브 두개가 나란히 놓여있는 상황에서, 다음의 경계조건을 가지고 층류유동 해석을 수행하였다. 두 튜브의 벽면과 튜브의 길이방향에 수직한 두 면에 점착(no slip adiabatic) 조건을 주었고, 두 튜브의 배열 방향에 수직한 두 면에 각각 입구(subsonic inflow)와 출구(subsonic outflow) 조건을 주었다. 그리고 나머지 경계면에는 대칭경계면(symmetry plane) 조건을 주었다.

여기서 사용한 유동해석 코드는 요즘 연구 중에 있는 병렬코드가 아닌, 기존의 일반적인 직렬코드이다. 병렬코드[8]란 빠른 계산을 목적으로 기존의 유동해석 코드를 수정하거나 혹은 새롭게 유동해석 알고리즘을 연구하여 PVM과 같은 병렬처리 소프트웨어를 연결시켜 병렬처리를 실현하고자 하는 코드를 말한다. 이러한 병렬코드의 연구개발은 전산유체의 새로운 영역으로 등장하여 병렬처리화에 상당한 기여를 하게 될 것이다. 추후 유동해석 병렬코드가 공개되어 본 연구에서 이를 가지고 유동해석을 수행한다면 더욱 빠른 속도로 유동해석이 이루어지게 될 것이다. 이상의 유동해석을 위한 유동조건 및 경계조건을 가지고 각 블록에서 생성된 격자들을 입력으로 해석을 수행하였다. Fig. 4는 각 작업단계에 따른 결과들을 차례대로 보여주고 있다. 즉 영역분할을 수행하고, 분할된 각 블록에서 격자를 생성한 후, 각 블록에서의 유동해석 결과들을 보여주고 있다.

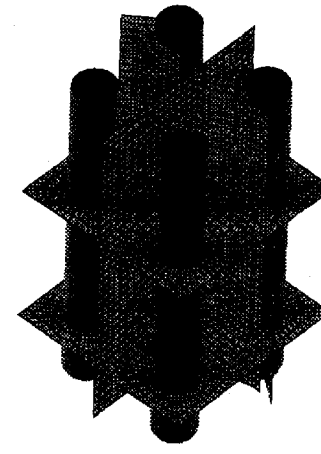
4.6 프로그램의 구조

병렬처리에 의한 해석을 수행하기 위해 선택된 master-slave model의 프로그램 구조를 살펴보면 다음과 같다. 본 연구는 PVM 시스템이 제공하는 라이브러리를 사용하고 있기 때문에 실제 프로그램 코드를 살펴보면 PVM에서 제공하는 여러 함수들을 불러쓴다. 여기서는 그 구조만을 보이겠다. 이 구조는 비단 유동해석뿐만 아니라 여러 관련분야에서 쉽게 응용될 수 있는 간단한 구조이다. 먼저 병렬처리의 주요 구성을 간략히 요약하면 다음과 같다. 영역분할에 의해 분할된 각 부영역에 대해 병렬처리 최소단위인 하나의 테스크가 필요한 모든 계산작업을 수행하도록 구성되어 있다. 즉 하나의 부영역에 하나의 테스크가 생성된다. 이러한 모든 과정을 주관하는 자가 바로 master이고 master의 명령에 의해 각 부영역에서 일하는 자가 slave이다. 본 연구의 경우, 부영역의 개수만큼 slave를 생성시킨다. master program의 주요 구조를 살펴보면 아래의 Table 1과 같다.

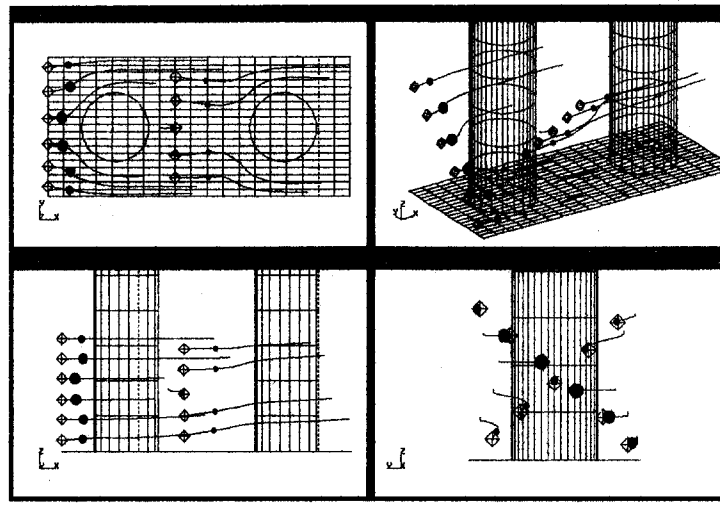
master program은 가장 먼저 자기 자신을 PVM 시스템에 알린 후에 자기 밑에서 일할 slave들을 생성한다. 이 slave들은 각각 병렬처리의 최소단위인 테스크(task)로서 실제 계산작업을 수행하게 된다. 그리고 분할된 각 영역에 필요한 유동데이터를 보낸 후각 slave에서 작업이 완료될 때



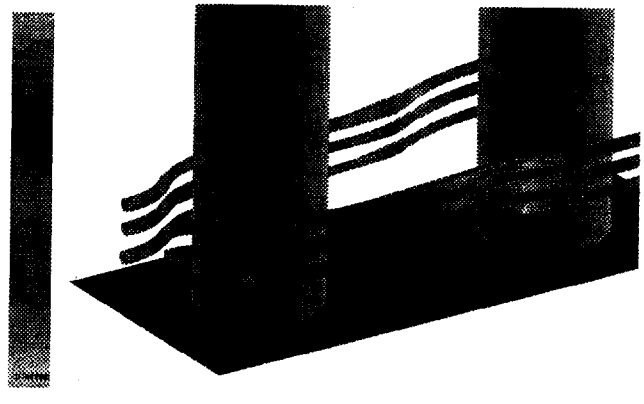
(a) Domain decomposition



(b) Grid generation



(c) Grid generation in a tube bank where diameter of particles means pressure



(d) Six streamtubes in a tube bank where its color and radius indicate flow velocity and pressure, respectively

Fig. 4 A tube-bank example

Table 1 Main steps in master program

	Master program
Step 1	Enroll in pvm-system
Step 2	Spawn the slave tasks
Step 3	Create the initial flow data on each block
Step 4	Send the initial flow data to the slaves
Step 5	Wait for the results and receive them from the slaves
Step 6	Display the flow outputs
Step 7	Kill the slave processes

Table 2 Main steps in slave program

	Slave program
Step 1	Enroll in pvm-system
Step 2	Receive my flow data from the master
	Perform the step 3 to 5 until satisfying the stop conditions
Step 3	Exchange boundary informations with my neighbor blocks
Step 4	Execute the flow solver on my block
Step 5	Update my flow ouputs resulted from the flow solver
Step 6	Send the final outputs back to the master

까지 기다린다. 그 다음 각 slave에서 계산완료된 결과들을 받아보고 이를 종합하여 사용자에게 보고한다. 마지막으로 모든 slave들을 죽이고 작업을 완료한다. 한편 slave program의 주요 구조를 살펴보면 아래의 Table 2와 같다.

slave program은 먼저 자기 자신을 PVM 시스템에 알린 후, master로부터 작업에 필요한 유동데이터를 받는다. 전달받은 유동 데이터를 가지고 다음의 계산작업을 반복수행한다. 먼저 이웃 블록과 필요한 모든 경계정보를 주고 받는다. 이 과정에서
slave들 간의 데이터 교환작업이 일어난다. 그리고 준비된 유동해석 코드에 의해 자기 영역에서 해석작업을 수행한다. 그리고 해석된 중간결과를 가지고 다시 이웃 블록들과 데이터 교환작업을 수행하고 또다시 반복하여 해석작업을 수행한다. 이러한 반복작업은 유동해석이 완료될 때까지 계속된다. 그리고 해석완료된 결과를 master에게 보낸다. 이로써 모든 작업을 완료한다.

이상의 master-slave model에 의해 각 블록에서의 유동해석 작업은 PVM 시스템에서 제공하는 메세지 전달 방식에 의해 이웃블록들과 데이터 교환을 수행하면서 진행한다. 즉 각 블록은 주어진 횟수만큼의 유동해석을 수행하고 그 결과를 이웃 블록에게 보내고 이웃 블록으로부터 해석수행 결과를 받는다. 여기서 두 블록 간에 주고 받는 데이터 정보는 다음과 같다. 인접한 두 블록에 대해 접하고 있는 경계면 사이에서 경계면의 각 격자점 위치에서 계산된 밀도, 속도, 내부 에너지 등의 데이터를 주고 받는다. 또한 경계면에서 시간에 따라 변화해 나가는 압력, 입사각 등의 정보들이 이웃블록의 입구에서 경계조건으로 사용될 수 있도록 데이터 교환정보로서 주고 받는다.

5. 시스템의 성능 및 결론

여기서 데이터 교환이 수행되는 시간은 많은 정보를 주고받아야 되는 응용문제의 경우에 중요한

Table 3 Iteration number and run time in various computing environments

Computing Environments	Iteration Number	Run Time
Parallel with four workstations	500	35min
Parallel with four workstations	1,000	70min
Parallel with two workstations	1,000	110min
Serial with one workstation	1,000	150 min

요소로서 병렬처리의 성능을 좌우하게 된다. 일반적으로 병렬처리에 소요되는 시간을 계산작업에 소요되는 시간만으로 측정해서는 안된다. 데이터 교환이 일어나는 시간을 고려해야 된다. 병렬처리의 성능을 재는 하나의 요소로서 계산작업 시간과 데이터교환 시간 간의 상대적 비율을 측정한다. 같은 병렬처리 시간에 대해 계산작업 시간이 데이터교환 시간보다 많은 비율을 차지하면 할수록 고성능이라고 말한다. 가장 이상적인 경우는 데이터 교환이 전혀 일어나지 않는 경우를 말한다. 여러 관련연구에서 병렬처리에 의한 작업을 계획하고 있을 때 효율적인 병렬처리를 위하여 다음의 문제를 자주 언급한다.[9]

- 각 프로세서에 부과되는 작업량을 균등하게 분배한다.
- 각 프로세서들 간의 데이터 교환양을 최소화한다.

위에서 언급한 것처럼, 사용자 입장에서 가장 이상적인 병렬 프로그램은 각 프로세서에 부과하는 작업량을 균등하게 분배하므로써 계산작업에 소요되는 시간을 최소화하고, 각 프로세서들 간의 데이터 교환양을 최소화하므로써 데이터 교환에 소요되는 시간을 최소화하는 경우이다. 전체작업을 분할하고 분할된 작업 간에 정보교환을 통하여 병렬처리를 수행하는 응용문제의 경우에 이 두 가지 최소화 문제는 항상 서로 충돌한다. 특히 전산유체 분야의 경우에 그러하다. 작업분할을 많이 하면 할수록 계산작업에 드는 시간을 최소화할 수 있다. 그러나 이에 따라 많은 데이터 교환이 수반된다. 데이터 교환을 줄이기 위해 작업분할을 적게하면 할수록 그만큼 계산작업에 드는 시간은 증가하게 된다. 본 연구의 경우도 이러한 상충문제가 있다. 이를 해결하기 위해서는 보다 신중한 영역분할 알고리즘이 개발되어야 하겠다. 즉 주어진 컴퓨터 환경 하에서 문제의 크기에 맞게 작업량을 균등하게 분배하고 데이터 교환양을 최소화하는 영역분할 작업이 연구되어야 하겠다.

한편, 4.5절의 적용 예제에서 보인 튜브뱅크 내부의 유동해석 문제에 관하여, 아래의 Table 3과 같이, 유동해석 작업의 주어진 반복횟수(iteration number) 동안에 소요된 작업시간(run time)을 4가지의 서로 다른 경우를 가지고 비교하면 다음과 같다. 여기서 병렬작업에 참여한 테스크의 수는 각각 9개이다.

위의 표에서 알 수 있드시, 첫번째 경우와 두번째 경우를 비교해 볼 때, 같은 컴퓨터 환경 하에서는 병렬처리 작업의 계산속도, 즉 반복횟수당 작업시간이 일정함을 알 수 있다. 그리고 두번째 경우와 세번째 경우를 비교해 볼 때, 계산에 참여하는 워크스테이션의 수가 증가할수록 작업시간이 비례하여 감소함을 알 수 있다. 여기서 테스크의 수가 같더라도 워크스테이션의 수가 증가하면 그만큼 워크스테이션에 걸리는 작업부하의 크기가 작아지고 이에 따라 테스크의 계산능력이 증대됨을 확인할 수 있다. 마지막으로 세번째 경우인 병렬처리에 의한 작업시간과 그렇지 못한 네번째 경우의 작업시간을 비교해 보면, 같은 계산양(반복횟수)에 대해 거의 2배되는 속도차이를 확인할 수 있다. 이 2배라는 결과는 계산에 참여한 테스크의 수를 고려해 볼 때 비례하는 속도 차이가 아니다. 일반적으로 테스크의 수가 증가할수록 계산속도(효율)의 향상을 확인할 수 있으나 반드시 비례하지는 않는다. 이는 테스크 간의 데이터 교환작업으로 인한 작업부하 때문이다. 그리고 경우에 따라서 특정 수 이상의 테스크 개수에서는 오히려 감소하는 경우가 있다. 이것 또한 데이터 교환작업으로 인한 작업부하가 계산속도의 향상을 방해하기 때문이다. 이상의 병렬처리 결과로부터 본 연구에서 제시한 병렬

시스템의 구조가 계산효율을 향상시킬 수 있는 유효한 구조임을 확인할 수 있다.

후 기

본 연구는 1995년도 교육부 학술연구조성비(기계공학연구)에 의하여 연구되었음

참고문헌

(1) Beguelin, A., Dongarra, J., Geist, G., Manchek, R. and Sunderam, V., 1991, "Solving Computational Grand Challenges Using a Network of Supercomputers," *Proceedings of the Fifth SIAM Conference on Parallel Processing*, D. Sorensen, ed., SIAM, Philadelphia.

(2) Geist, G., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., 1994, *PVM : Parallel Virtual Machine, A Users Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Massachusetts.

(3) Sunderam, V. S., 1990, PVM : A Framework for Parallel Distributed Computing, *Journal of Concurrency : Practice and Experience*, Vol. 2, No. 4, pp. 315~339.

(4) Geist, G. A. and Sunderam, V. S., 1992, Network Based Concurrent Computing on the PVM System, *Journal of Concurrency : Practice and Experience*, Vol. 4, No. 4, pp. 293~311.

(5) Gropp, W., Lusk, E. and Skjellum, A., 1994, *Using MPI* : Portable Parallel Programming with the Message Passing Interface, MIT Press, Cambridge, Massachusetts.

(6) Okusanya, T. and Peraire, J., 1996, "Parallel Unstructured Mesh Generation," *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, pp. 719~729.

(7) Sullivan, M. and Anderson, D., 1989, "Marionette : A System for Parallel Distributed Programming Using a Master/Slave Model," *Proceedings of the 9th ICDCS*, pp. 181188.

(8) Hauser, J. and Williams, R., 1992, "Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines," *International Journal for Numerical Methods in Fluids*, Vol. 14, pp. 51~58.

(9) Moitra, S. and Moitra, A., 1996, "Considerations of Computational Optimality in Parallel Algorithms for Grid Generation," *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, pp. 753~762.